Optimization under Uncertainty: An Introduction through Approximation

Kamesh Munagala

Contents

I Stochastic Optimization **5**

1 Weakly Coupled LP Relaxations **6**
1.1 A Gentle Introduction: The Maximum Value Problem 6
1.1.1 Optimal Policy and Adaptivity 6
1.1.2 Non-adaptive policies 8
1.1.3 Linear Programming 8
1.1.4 LP Structure: Weak Coupling 10
1.1.5 Rounding 11
1.2 Stochastic Matchings 13
1.2.1 LP Formulation 14
1.2.2 Rounding 15

2 Simpler Policies via Duality **18**
2.1 Simpler Algorithm of Maximum Value 18
2.1.1 Lagrangian 18
2.1.2 Structure of Lagrangian 19
2.1.3 An Aside: Solving the LP Relaxation 20
2.1.4 Rounding Algorithm 21
2.1.5 Adaptivity Gap 24
2.2 The Prophet Inequality 24
2.2.1 A Threshold Policy and Analysis 25
2.3 Bayesian Pricing 27
2.3.1 LP Relaxation 28
2.3.2 Lagrangian and Posted Prices 29
2.3.3 The LP Solution 30
2.3.4 Final Pricing Scheme 31

3 Submodularity **32**
3.1 Submodularity 32
3.2 The Greedy Algorithm 33
3.3 Examples of Submodular Functions 35
3.3.1 Joint Entropy 35
3.3.2 Influence and Epidemics 37

4 Greedy Adaptive Policies: Stochastic Set Cover 40
4.1 Classical Set Cover 40
4.1.1 Direct Analysis 41
4.1.2 Dual Fitting 42
4.2 Evaluating Conjunctive Queries with Shared Filters 43
4.2.1 Fixed Orderings are Sub-optimal 44
4.3 Adaptive Policies for Conjunctive Query Evaluation 46
4.3.1 Covering Graph and Connection to Set Cover 46
4.3.2 Adaptive Greedy Policy 47
4.3.3 Set Cover Type Analysis 47

5 Markov Decision Processes 50
5.1 Modeling Problems as MDPs 52
5.2 Discounted Reward Version 52
5.2.1 Value Iteration 53

6 Multi-armed Bandits 55
6.1 A Bayesian Example 56
6.1.1 Parametrized Distributions 58
6.1.2 Applications of Bayesian Bandits 59
6.2 A General Problem Statement 60
6.2.1 Connecting with Priors and Posteriors 60
6.3 The Gittins Index Theorem 61
6.3.1 Structure of the Gittins Index 61
6.3.2 Iterative Algorithm 62
6.3.3 Inductive Analysis 62
6.4 Structure of the Index in Bayesian Setting 66

II Unknown Distributions and Online Prediction 67

7 The Finite Horizon Multi-armed Bandit Problem 68
7.1 The Regret Measure 69
7.2 A Naive Policy 69
7.3 The UCB1 Policy 71
7.3.1 Analysis 71
7.4 Lower Bounds on Regret: UCB1 is Nearly Optimal 73
7.4.1 Preliminaries: Dissimilarity Measures between Distributions . 73
7.4.2 Two Lower Bounds on Stochastic Bandits 75
7.4.3 Key Analysis Idea: Single Arm Problem 76
7.4.4 Proof of Theorem 7.4.6 77
7.5 An Optimal Policy: Thompson Sampling 79
7.5.1 Recap of Bayesian Bandits 79
7.5.2 A First Attempt 79
7.5.3 The Thompson Sampling Policy 80
7.5.4 Optimal Regret Bound 81

7.6 Bayesian Analysis of Thompson Sampling 82
7.6.1 Proof of Theorem 7.6.1 83

8 The Experts Problem 88
8.0.1 Warmup: Online Prediction 88
8.1 The Experts Problem and Regret Measure 90
8.2 The Weighted Majority Algorithm 92
8.3 Adversarial Multi-armed Bandits 94
8.3.1 A Sub-optimal Algorithm 94
8.3.2 The **Exp3** Algorithm 96

9 Applications of the Experts Framework 100
9.1 Routing Flows in a Network 100
9.1.1 Problem Statement 101
9.1.2 The Lagrangian 101
9.1.3 Generating Gain Vectors 102
9.1.4 The Final Algorithm 103
9.1.5 Analysis 104
9.1.6 Connection to Routing on the Internet 105
9.2 General Recipe for Solving Linear Programs 106
9.3 Boosting 108
9.3.1 Defining Experts and their Losses 108
9.3.2 AdaBoost Algorithm 109

III Online Algorithms and Competitive Analysis 111

10 Introduction: Ski Rental and Paging 112
10.1 Ski Rental Problem 112
10.2 Paging Problem 113
10.3 Randomized Paging Algorithms 114

11 Yao's Minimax Theorem 116
11.1 Yao's Theorem 116
11.2 Lower Bound for Randomized Paging 117

12 Potential Functions 119
12.1 List Update Problem 119

13 Primal-Dual Method 122
13.1 Randomized Ski Rental 122
13.2 Other Online Problems 124
*

Part I

Stochastic Optimization

Chapter 1

Weakly Coupled LP Relaxations

We begin our journey of stochastic optimization with a very simple problem motivated by the theory of optional stopping. We term this the *Maximum Value Problem*.

1.1 A Gentle Introduction: The Maximum Value Problem

There is a gambler who has access to n closed boxes. Box has reward r_i drawn from distribution X_i. The reward distributions in the different boxes are independent random variables. The gambler knows the distributions, but only gets to know the reward value in a box once he opens it. He can open at most k boxes, where $k \leq n$, and choose the reward from at most one box. His goal is to maximize his expected reward. What should the gambler's strategy be?

1.1.1 Optimal Policy and Adaptivity

In the *Maximum Value* problem, the gambler makes choices of which boxes to open. These choices are made sequentially, and can depend on the rewards observed so far. This defines a decision policy, and the gambler needs to find the decision policy whose expected reward is as large as possible, where the expectation is over the random outcomes of opening boxes.

We now present a simple example and the optimal decision policy.

Example 1.1. *There are* $n = 3$ *boxes. Box* 1 *has reward distribution* X_1 *which is Bernoulli*$(50, 1/2)$*, that is takes value* 50 *with probability* $1/2$ *and 0 with probability* $1/2$*. Box* 2 *has reward distribution Bernoulli*$(60, 1/3)$*. Box* 3 *has reward distribution Bernoulli*$(25 + \epsilon, 1)$*, where* $\epsilon > 0$ *is a small constant. The gambler can open* 2 *boxes, so that* $k = 2$*.*

Which two boxes should the gambler open? We immediately come to a trade-off: A gambler wants to maximize the reward present in some opened box, but at the same time, he wants to resolve information about boxes. These two goals can be aligned,

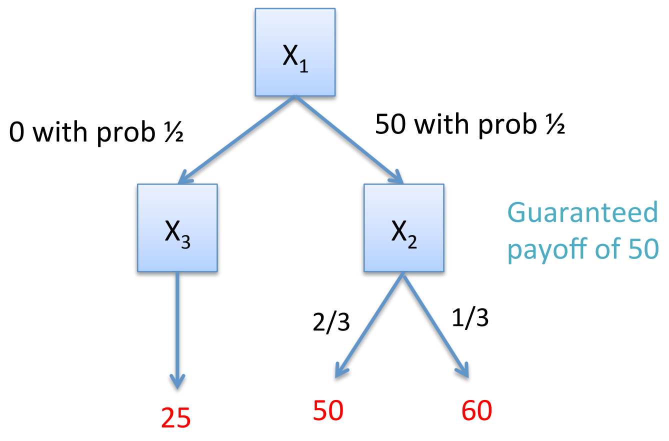

Figure 1.1: Optimal decision tree for $n = 3$ boxes and $k = 2$. $X_1 = \text{Bernoulli}(50, 1/2)$, $X_2 = \text{Bernoulli}(60, 1/3)$, and $X_3 = \text{Bernoulli}(25 + \epsilon, 1)$.

but are not necessarily the same thing. In this example, box 3 has the highest expected reward, but this random variable is deterministic. This means the gambler gains no new information by opening this box, and can save it for later. The optimal policy opens box 1 first. If this value is 0, which happens with probability $1/2$, the gambler should choose the box with highest expected value (box 3) next. If box 1 yields 50, then there is no point in opening box 3, since the gambler already has a guaranteed reward of 50. He will therefore open box 2. If this value is 60, he chooses this reward, else he goes with the reward in box 1.

In summary, with probability $1/2$. box 1 yields 0 and box 3 yields $25 + \epsilon$, so the gambler's payoff is $25 + \epsilon$. With probability $1/6$, box 1 yields 50 and box 2 yields 60, so the gambler's payoff is 60. With probability $1/3$, box 1 yields 50 and box 2 yields 0, so the payoff is 50. Therefore, the expected reward of the gambler is approximately 39.167.

The optimal policy therefore defines a decision tree. Suppose we assume each distribution X_i is discrete with m possible values, then each node in this decision tree has a branching factor of m. The depth of the decision tree is the number of boxes that can be opened, k. Therefore, the worst-case size of the decision tree is $O(m^k)$, which is exponential in k.

The central question therefore becomes:

> Are there succinct decision policies that are efficient to compute and that have objective value comparable to the optimal decision policy?

By succinct, we mean policies that have size that is polynomial in the problem parameters – n, k, and m. Ideally, we would like strongly polynomial time algorithms to compute these policies.

1.1.2 Non-adaptive policies

A simple class of decision policies are those that make all decisions upfront before the exact values of any of the random variables are realized. Such policies are termed *non-adaptive*. For the maximum value problem, such a policy choose k boxes upfront, subsequently opening them and choosing the maximum value. In Example 1.1, the best choice of boxes are box 1 and box 3. With probability $1/2$, the payoff is 50, and otherwise, the payoff is 25. The expected payoff is 37.5. This is worse than the payoff of the best adaptive policy, but the description of the policy is much simpler – there is no decision tree to reason about.

How hard is the optimal non-adaptive policy to compute, and how bad are such policies compared to the optimal adaptive policy? The answer to this question is problem-specific. For the maximum value problem, we can show the following:

- The problem of computing the optimal non-adaptive policy is NP-Complete.
- There is a $(1 - 1/e)$ approximation to the value of the optimal non-adaptive policy.
- The gap between the values of the optimal adaptive and non-adaptive policies is at most 3.

The gap between the values of the optimal adaptive and non-adaptive policies is termed the *adaptivity gap*. In this chapter and the next, we will build towards showing that the adaptivity gap of maximum value is at most 3. As an immediate corollary, we will also present an efficient to compute non-adaptive policy which is a 3-approximation to the optimal adaptive policy. We will present a different non-adaptive policy in later chapters.

1.1.3 Linear Programming

We now present a simple linear programming (LP) relaxation to the optimal decision policy. To come up with this LP formulation, we consider any decision policy and focus on its behavior for a particular box. For box j, let y_j denote the probability over the execution of the policy that the box is opened.

$$y_j = \Pr[\text{Box } j \text{ is opened}]$$

In the decision tree in Figure 1.1, since box 1 is opened upfront, $y_1 = 1$. Box 2 is opened when $X_1 = 0$, so that $y_2 = 1/2$. Similarly, $y_3 = 1/2$. Since k boxes are opened in any decision path, this is true in expectation. This translates to the constraint:

$$\sum_{j=1}^{n} y_j \leq k$$

Since y_j is a probability value, we also have the constraint

$$y_j \in [0, 1] \quad \forall j \in \{1, 2, \ldots, n\}$$

We need another set of variables to capture the event that the reward of a certain box is chosen as the final payoff. Let

$$z_j(v) = \Pr[X_j = v \text{ and the payoff from box } j \text{ is finally chosen}]$$

In the decision tree in Figure 1.1, suppose $X_1 = 50$, an event that happens with probability $1/2$. Then, we open Box 2 and choose the payoff of box 1 when $X_2 = 0$. Since this event happens with probability $2/3$, we have $z_1(50) = 1/2 \times 2/3 = 1/3$. Similarly, $z_2(60) = 1/2 \times 1/2 = 1/6$ and $z_3(25+\epsilon) = 1/2$. The remaining z variables are identically zero.

Let $f_j(v) = \Pr[X_j = v]$. If we consider the event corresponding to $z_j(v)$, two events must happen for this event to materialize:

1. Box j must be opened, and event that happens with probability y_j
2. The value in box j must be v, an event that happens with probability $f_j(v)$.

Note now that the values in the boxes follow independent distributions, the value observed in a box must be independent of when the box is opened. This means the above two events are independent. Therefore, we have

$$\Pr[\text{Box } j \text{ is opened and } X_j = v] = y_j f_j(v)$$

Since this event is a superset of the event that Box j is finally chosen and $X_j = v$, we have the constraint:

$$z_j(v) \leq y_j f_j(v)$$

On any decision path, the payoff from only one box is finally chosen. Since this is true for any decision path, this is also true in expectation. Therefore,

$$1 \geq \mathbf{E}\left[\text{ Boxes whose payoff is chosen }\right] = \sum_{j,v} z_j(v)$$

Finally, the expected reward of the policy can be written as:

$$\text{Expected Reward } = \sum_{j,v} v \cdot z_j(v)$$

Putting all this together, we obtain the following LP relaxation for Maximum value:

The above linear program has $O(nm)$ variables and constraints, where n is the number of boxes and m is the number of possible values that any random variable X_j can take. We are implicitly assuming the random variables are discrete over a small set of possible values, but as we shall see later, we can solve such LPs even for continuous distributions.

Since our LP encodes constraints that are satisfied by any valid policy, we directly have:

$$\boxed{\begin{array}{rcll} \text{Maximize} & & \sum_{j,v} v \cdot z_j(v) & \\ \sum_{j,v} z_j(v) & \leq & 1 & \\ \sum_j y_j & \leq & k & \\ z_j(v) & \leq & y_j \cdot f_j(v) & \forall j, v \\ y_j & \in & [0,1] & \forall j \\ z_j(v) & \geq & 0 & \forall j, v \end{array}}$$

Claim 1.1.1. *The optimal value of the LP relaxation is at least the reward of the optimal decision policy.*

Suppose we solve this LP. Will it yield the optimal policy? Not directly. We have relaxed two constraints that hold for every decision path: First that at most k boxes are opened, and second that the payoff from at most one box is chosen. The LP only enforces these constraints in expectation. For Example 1.1, the LP can set $y_1 = y_2 = 1$; $z_1(50) = 1/2$; and $z_2(60) = 1/3$. This is feasible for all constraints and yields objective value 45. However, this setting of variables does not correspond to any feasible policy. This is because the solution always takes the payoff from box 1 when it yields 50 and from box 2 when it yields 60. But these events occur simultaneously with probability $1/6$, making such a policy infeasible.

In the next chapter, we show that the constraint "at most one box's payoff is chosen in any decision path" can be encoded exactly using a polymatroid. We show that for related problems, this approach yields exact polynomial time algorithms.

1.1.4 LP Structure: Weak Coupling

In the LP presented above, there are only two constraints that run across boxes: $\sum_{j,v} z_j(v) \leq 1$ and $\sum_j y_j \leq k$. The remaining constraints are specific to a single box. Focus on some box j and consider the variables relevant to it. They define a policy restricted to this box; let us call this single-box policy P_j.

Step 1. Open the box with probability y_j, and with the remaining probability do nothing. If box opened, then go to Step 2.

Step 2. If the value in the box is v, choose this as final payoff with probability $\frac{z_j(v)}{y_j \cdot f_j(v)}$.

This describes a space of *feasible* randomized policies for a single box. These single-box policies can be viewed as the actions of the optimal policy when restricted to that box.

Definition 1. *A feasible single-box policy opens the box with some probability, and conditioned on opening it and observing a value, chooses this value as final payoff with certain probability.*

As an example, if we consider the optimal policy in Figure 1.1 and project it onto individual boxes, P_1 opens box 1 with probability 1 and if $X_1 = 50$, chooses this payoff with probability $2/3$. Similarly P_2 opens box 2 with probability $1/2$ and conditioned on observing $X_2 = 60$, chooses the box with probability 1. The policy P_3 opens box 3 with probability $1/2$ and conditioned on observing $X_3 = 25 + \epsilon$, chooses this box with probability 1.

Simpler Notation. In the policy P_j, the probability of the event that $X_j = v$ *and* the payoff of the box is chosen will be exactly $z_j(v)$. To see this, observe that box j is opened and $X_j = v$ with probability $f_j(v)y_j$, and conditioned on this event, the payoff is chosen with probability $\frac{z_j(v)}{y_j \cdot f_j(v)}$.

Define the following probabilities for policy P_j:

$$\begin{aligned} O(P_j) &= \Pr[P_j \text{ opens box } j] = y_j \\ C(P_j) &= \Pr[P_j \text{ chooses the reward from box } j] = \textstyle\sum_v z_j(v) \end{aligned}$$

The expected reward of policy P_j is simply

$$R(P_j) = \mathbf{E}[\text{Reward from } P_j] = \sum_v v \cdot z_j(v)$$

We can now write the LP in a compact form as follows, where the variables are feasible single-box policies P_j (see Def. 1).

$$\boxed{\begin{array}{ccc} \text{Maximize} & & \sum_j R(P_j) \\ \sum_j C(P_j) & \leq & 1 \\ \sum_j O(P_j) & \leq & k \\ \text{Each} & P_j & \text{feasible} \end{array}}$$

In essence, the LP finds one feasible policy for each box so that the sum of the expected number of times these policies choose the payoff is at most 1, and the sum of the expected number of times these policies open the respective boxes is at most k. This in essence is the idea of "weak coupling", where the relaxation is obtained by constructing decision policies over simpler spaces (in our case, single boxes) and coupling them using very few "expected value" constraints. This simplification will be useful later when we reason about these policies using duality.

1.1.5 Rounding

We present the policy *Simple Greedy* in Figure 1.3.

1. Solve this LP and let the output be single box policies $\{P_j\}$.
2. Order the boxes arbitrarily as $1, 2, \ldots, n$ and consider them in this order.
3. When box i is encountered, skip this box with probability $3/4$ and move to box $i+1$.
4. With probability $1/4$, execute P_i. If this policy opens the box and chooses the payoff, then choose this payoff and STOP.
5. After P_i executes, if k boxes have been opened, the STOP.
6. In all other cases, continue to box $i + 1$.

Figure 1.2: Policy *Simple Greedy* for Maximum Value.

Analysis Our analysis will crucially use Markov's inequality. Consider the execution of the policy. For box j, let T_j be a random variable that denotes the "cost" spent by the policy on box j. This cost is measured as follows: If the box is opened, the cost is $1/k$, and if the payoff of the box is chosen, the cost is 1.

Consider the overall policy. When will it encounter box i? It will encounter it if the following two conditions hold:

1. The number of boxes in $1, 2, \ldots, i$ that it has opened is at most $k - 1$; and
2. It did not choose the payoff from any box in $1, 2, \ldots, i - 1$.

If either of the above events happened, the policy would have stopped execution and not encountered box i. If neither of the two events above happen, the total cost spent on boxes $1, 2, \ldots, i - 1$ is at most 1. This is because the cost spent on choosing payoff is identically 0, and less than k boxes each with opening cost $1/k$ are opened. This means the policy encounters box i if $\sum_{j=1}^{i-1} T_j < 1$. We will now bound the probability of this event using Markov's inequality.

In order to do that, we simplify the LP relaxation even further by adding its two constraints as follows:

$$\begin{array}{rcl} \text{Maximize} & & \sum_j R(P_j) \\ \sum_j \left(C(P_j) + \frac{O(P_j)}{k} \right) & \leq & 2 \\ \text{Each} & P_j & \text{feasible} \end{array}$$

Let $T(P_j) = C(P_j) + \frac{O(P_j)}{k}$. The LP constraint implies $\sum_j T(P_j) \leq 2$. We continue with the analysis

Claim 1.1.2.

$$\mathbf{E}[T_j] \leq \frac{T(P_j)}{4}$$

Proof. If the policy stops before encountering box j, then $T_j = 0$. Else, the policy does not skip box j with probability $1/4$. Conditioned on this, $\mathbf{E}[T_j] = T(P_j)$ by the definition of $T(P_j)$. Note that we are crucially using the fact that what happens in box j is independent of the execution of the policy thus far. Removing the conditioning, we have the claim. □

Claim 1.1.3. *Consider some box i. Then*

$$\Pr\left[\sum_{j<i} T_j < 1\right] \geq 1/2$$

Proof. The LP constraint implies $\sum_j T(P_j) \leq 2$. This implies $\sum_{j<i} T(P_j) < 2$. Using the previous claim, this implies

$$\mathbf{E}\left[\sum_{j<i} T_j\right] \leq 1/2$$

By Markov's inequality, we have

$$\Pr\left[\sum_{j<i} T_j < 1\right] \geq 1/2$$

□

The rest of the analysis is straightforward. Box i is encountered with probability at least $1/2$, and conditioned on this, policy P_i is executed with probability $1/4$. Conditioned on both these events, the expected reward from P_i is $R(P_i)$, since what happens in box i is independent of the execution path so far. Removing the conditioning and applying linearity of expectation, we have:

$$\text{Final Reward} = \sum_i \mathbf{E}\left[\text{Reward from Box i}\right] = \sum_i \frac{1}{2} \times \frac{1}{4} \times R(P_i) = \frac{1}{8} \times \text{LP optimum}$$

Since the LP is a relaxation of the optimal decision policy, we finally have:

Theorem 1.1.4. *Policy* Simple Greedy *is a 8-approximation to the optimal decision policy for Maximum value.*

1.2 Stochastic Matchings

We now consider a somewhat more general problem. In the *stochastic matching* problem, there are n vertices on the left (call these men) and m vertices on the right (call these women). Let L denote the set of men and let R denote the set of women. Think about running a dating website. The goal of the website is to sequentially set up blind dates between the men and women (or probe the corresponding edge) so that we maximize the number of successful matches. More formally, suppose we set up $i \in L$ and

$j \in R$ on a blind date (or probe edge (i, j)), the probability of a successful match is p_{ij}. If the match happens, this edge is removed and both the end-point vertices are removed from L and R respectively. This corresponds to the users exiting the dating platform. The successful match yields reward r_{ij}. In case the match fails, the edge (i, j) is removed from further consideration, but the vertices continue to exist.

Each vertex i has a patience t_i in terms of the number of times edges incident to it can be probed. In the dating context, this is the number of times a user is willing to go on blind dates before he or she gives up on the platform. In keeping with the idea of weak coupling, we crucially assume the p_{ij} are independent for different edges.

A decision policy knows the $\{p_{ij}, r_{ij}, t_i\}$ values upfront. It adaptively decides the order of probing edges so that it maximizes the expected reward from successful matches. We have presented this problem in the context of bipartite graphs to motivate it; the exact same problem can be formulated for general graphs and everything we present below works as is.

Let us first show that simple heuristics do not necessarily work. A natural ordering is Greedy, which probes the edges e in decreasing order of expected reward, $r_e p_e$.

Example 1.2. *Let $L = \{b_1, b_2, \dots, b_n\}$, and $R = \{a_1, a_2, \dots, a_n, c_1, c_2, \dots, c_n\}$. There is an edge (a_i, b_i) for every i with $r = 1$ and $p = 1$. There is an edge (c_i, b_j) for every i, j that has $r = \frac{n}{3}$ and $p = \frac{1}{n}$. All patience values are infinity.*

On this example, Greedy probes all edges of the form (a_i, b_i) first. Since this forms a matching, the process stops yielding total reward n. On the other hand, probing the edges between $\{b_1, b_2, \dots, b_n\}$ and $\{c_1, c_2, \dots, c_n\}$ in random order. A simple calculation shows there will be $\Omega(n)$ successful matches with constant probability. This yields reward $\Omega(n^2)$, which is a factor $\Omega(n)$ larger than Greedy.

1.2.1 LP Formulation

The LP relaxation follows the same idea as Maximum Value. Consider some decision policy. For edge (i, j), let

$$x_{ij} = \Pr\left[\text{Edge } (i, j) \text{ is probed}\right]$$

In the event that edge (i, j) is probed, the probability of a successful match is exactly p_{ij} regardless of when the edge is probed. Therefore

$$p_{ij} x_{ij} = \Pr\left[\text{Edge } (i, j) \text{ is matched}\right]$$

We now write some constraints. First, on any decision path, vertex i is probed at most t_i times. Taking expectation over all decision paths and using linearity of expectation, we have:

$$\sum_j x_{ij} \le t_i \ \ \forall \text{ vertices } i$$

Similarly, each vertex is successfully matched at most once on any decision path. Taking expectation over all decision paths, we have:

$$\sum_j p_{ij} x_{ij} \le 1 \ \ \forall \text{ vertices } i$$

Finally, x_{ij} is a probability value, so that

$$x_{ij} \in [0,1] \ \ \forall (i,j)$$

This yields the following LP relaxation of the optimal decision policy:

$$\boxed{\begin{array}{rcll} \text{Maximize} & & \sum_{i,j} r_{ij} p_{ij} x_{ij} & \\ \sum_j p_{ij} x_{ij} & \leq & 1 & \forall i \\ \sum_j x_{ij} & \leq & t_i & \forall i \\ x_{ij} & \in & [0,1] & \forall (i,j) \end{array}}$$

1.2.2 Rounding

Our rounding will mimic that of *Maximum value*. As the policy executes, we say that probing an edge (i,j) is feasible if the following to conditions hold:

- Neither i nor j has participated in successful matches; and
- The number of times an edge incident to i (resp. j) has been probed so far is less than t_i (resp. t_j).

---

1. Solve this LP and let $\vec{x}$ denote the output.
2. Order the edges arbitrarily, and consider them in this order.
3. When edge e is encountered, skip it with probability $7/8$.
4. With the remaining probability, if probing the edge is feasible, then probe it with probability x_e.

Figure 1.3: LP Policy for Stochastic Matching.

Theorem 1.2.1. *The above policy is a* 16 *approximation to the value of the optimal policy.*

Proof. Just as with *Maximum value*, we will use Markov's inequality to bound the probability that any edge (i,j) will be encountered by the policy. For (i,j) to be encountered, all of these events must happen

- Vertex i (resp. j) must not have been matched successfully so far; and
- The number of times an edge incident on vertex i (resp. j) has been probed must be less than t_i (resp. t_j).

Conditioned on encountering some edge e, the probability that edge e is probed by the policy is $\frac{x_e}{8}$. Therefore, the expected number of times an edge incident to vertex i is probed is at most

$$\sum_k \frac{x_{ik}}{8} \le \frac{t_i}{8}$$

by the LP constraint. Therefore, by Markov's inequality,

$$\Pr\left[\text{Number of edges incident to } i \text{ that are probed } \ge t_i\right] \le \frac{t_i/8}{t_i} = \frac{1}{8}$$

A similar statement holds for vertex j. By union bounds, this means that with probability $1 - 2 \times \frac{1}{8} = 3/4$, the number of edges incident on i that have been probed is less than t_i *and* the number of edges incident on j that are probed is less than t_j.

Similarly, conditioned on the policy encountering edge e, the probability it is successfully matched is $\frac{p_e x_e}{8}$, since the event of match conditioned on probing is independent of when the probe happens. Therefore, the expected number of edges incident on i that are matched is at most

$$\sum_k \frac{p_{ik} x_{ik}}{8} \le \frac{1}{8}$$

by the LP constraint. Therefore, by Markov's inequality,

$$\Pr\left[\text{Number of edges incident to } i \text{ that are matched } \ge 1\right] \le \frac{1}{8}$$

A similar statement holds for vertex j. By union bounds, this means that with probability $1 - 2 \times \frac{1}{8} = 3/4$, *both* i and j are unmatched.

Taking union bounds over both the above events, with probability $1/2$, all these events happen and edge (i, j) is encountered by the policy. In this event, this edge is probed with probability $1/8$, in which case it yields expected reward $p_{ij} r_{ij}$. Removing all the conditioning, the expected reward from edge (i, j) is therefore $\frac{p_{ij} r_{ij}}{16}$. By linearity of expectation, this shows the policy yields a 16 approximation to the LP value. $\square$

Chapter 2

Simpler Policies via Duality

In this chapter, we continue our discussion of weakly coupled relaxations. We will show that the dual of these relaxations often encode structure that leads to simpler algorithms with better performance guarantees.

2.1 Simpler Algorithm of Maximum Value

We revisit the Maximum value problem considered in the previous chapter. We will present a 3-approximation using duality. The interesting aspect is that the final policy is *non-adaptive* – it chooses k boxes upfront and opens them simultaneously, subsequently choosing the maximum payoff observed in these boxes.

We start with the compact LP formulation from the previous chapter. Recall that for any box j, we can define feasible single-box policies P_j. Such a policy opens the box with some probability y_j and conditioned on opening it and observing value v, chooses this as payoff with some probability $\frac{z_j(v)}{f_j(v)y_j}$, where $f_j(v) = \Pr[X_j = v]$. Recall that $R(P_j)$ is the expected reward of this policy, $C(P_j)$ is the probability it chooses the payoff in this box, and $O(P_j)$ is the probability it opens the box. The LP relaxation finds a collection of feasible policies $\{P_i, i = 1, 2, \ldots, n\}$ to optimize the following.

$$\begin{array}{rcl} & \text{Maximize} & \sum_j R(P_j) \\ \sum_j \left(C(P_j) + \frac{O(P_j)}{k}\right) & \leq & 2 \\ & \text{Each} & P_j \quad \text{feasible} \end{array}$$

It was shown in the previous chapter that this LP is a relaxation of the optimal decision policy. Let OPT denote the optimal LP value.

2.1.1 Lagrangian

For any real value $\lambda \geq 0$, we can define the following Lagrangian relaxation:

Maximize	$\sum_j \left(R(P_j) - \lambda\left(C(P_j) + \frac{O(P_j)}{k}\right)\right) + 2\lambda$
Each P_j feasible	

Figure 2.1: The Lagrangian for parameter $\lambda \geq 0$.

Let $L(\lambda)$ denote the optimal Lagrangian value. The following is a restatement of weak duality.

Claim 2.1.1. *For any* $\lambda \geq 0$, $L(\lambda) \geq OPT$.

Proof. Let $\{P_i^*\}$ denote the optimal solution to the LP. This is clearly feasible for the Lagrangian since the Lagrangian only has fewer constraints. Since the LP constraint is satisfied, this implies

$$2 - \sum_j \left(C(P_j^*) + \frac{O(P_j^*)}{k}\right) \geq 0$$

This implies that for any $\lambda \geq 0$, $L(\lambda)$ is at least the LP objective of $\sum_j R(P_j^*)$, which is exactly OPT. □

2.1.2 Structure of Lagrangian

Fix some $\lambda \geq 0$ and consider the Lagrangian for this value of λ. There are *no* constraints coupling different arms together. This implies the Lagrangian can be solved as follows: For each box j, find that single-box policy that maximizes

$$R(P_j) - \lambda\left(C(P_j) + \frac{O(P_j)}{k}\right)$$

Let us understand this quantity in terms of costs as follows: When a policy decides to open the box, it pays an *opening cost* of $\frac{\lambda}{k}$. Suppose it decides to choose the payoff in the box, it pays a cost of λ. Suppose the box is opened, value v is observed, and is chosen, then the net payoff, which is actual payoff minus cost, will be $v - \frac{\lambda}{k} - \lambda$. By invoking linearity of expectation, it is easy to check that the expected net payoff of any policy P_j would therefore be given by the expression $R(P_j) - \lambda\left(C(P_j) + \frac{O(P_j)}{k}\right)$.

The Lagrangian therefore finds the optimal single-box policy that maximizes net payoff separately for each box. This is a simple Markov Decision Process with a small state space and set of actions – the state of the system is either "box not opened", or "box opened and value v observed". The actions are either "open/not open" or "choose/not choose payoff". We can use dynamic programming to compute this policy, but it turns out to even have a closed form for payoff!

For box i, what it this policy? This is simple to compute: The policy either opens the box or does not. If it does not, the net payoff is 0. If it does, it pays an upfront cost of $\frac{\lambda}{k}$. Subsequently, it should choose the payoff whenever it is more than the cost of making the choice, meaning that whenever $v \geq \lambda$. This means:

$$\text{Optimal Net Payoff from box } j = \max\left(0, \mathbf{E}\left[(X_j - \lambda)^+\right] - \frac{\lambda}{k}\right)$$

where for any random variable Y, we have used the notation Y^+ to denote the random variable $\max(Y, 0)$.

Note that this single-box optimal policy *deterministically* makes the choice of whether to open the box or not. Conditioned on opening and observing a certain value, it *deterministically* decides whether to take that payoff or leave it. This is a general feature of the optimal policies of Markov decision processes – there is always a deterministic optimal policy.

We simplify notation to ease the discussion below. Fix parameter $\lambda \geq 0$. Let $P_j(\lambda)$ the optimal single-box policy for the Lagrangian, *i.e.* that which maximizes $R(P_j) - \lambda\left(C(P_j) + \frac{O(P_j)}{k}\right)$. Using this policy, define $R_j(\lambda) = R(P_j(\lambda))$ to be its expected reward; $C_j(\lambda) = C(P_j(\lambda))$ to be its probability of choosing the box; and $O_j(\lambda) = O(P_j(\lambda))$ to be the probability of opening the box. This means

$$L(\lambda) = 2\lambda + \sum_j \left(R_j(\lambda) - \lambda\left(C_j(\lambda) + \frac{O_j(\lambda)}{k}\right)\right)$$

Example 2.1. *In Example 1.1, suppose $\lambda = 20$. Since $k = 2$, opening a box has cost 10. The policy $P_3(\lambda)$ would not open box 3 since the net payoff from opening and choosing is $25 - 20 - 10 = -5$. On the other hand, $P_1(\lambda)$ would open box 1 and choose its reward when it is 50, since this has value $-10 + 1/2 \times (50 - 20) = 5$.*

2.1.3 An Aside: Solving the LP Relaxation

We can use the Lagrangian to the solve the LP relaxation. This part can be skipped without losing continuity. First note that as we increase λ, the optimal single-box policy becomes more and more conservative in opening the box or choosing its payoff. If $\lambda = 0$, the policy will surely open the box and always take the payoff; if $\lambda = \infty$, the policy will never open the box. Therefore, the expected value $C_j(\lambda^-) + \frac{O_j(\lambda^-)}{k}$ is monotonically decreasing in λ. This means there will be two values $\lambda^- < \lambda^+$ separated by an arbitrarily small amount, such that

$$\alpha^- = 2 - \sum_j \left(C_j(\lambda^-) + \frac{O_j(\lambda^-)}{k}\right) \geq 0$$

and

$$\alpha^+ = 2 - \sum_j \left(C_j(\lambda^+) + \frac{O_j(\lambda^+)}{k}\right) < 0$$

We can find such λ^-, λ^+ with $|\lambda^+ - \lambda^-| \leq \epsilon$ by binary search over λ. For each λ, the solution to the Lagrangian is simply the closed form discussed above. Consider the corresponding Lagrangian solutions at λ^-, λ^+ and take a convex combination where we multiply the solution at λ^- by $q^- = \frac{-\alpha^+}{\alpha^- - \alpha^+}$ and that at λ^+ by $q^+ = \frac{\alpha^-}{\alpha^- - \alpha^+}$. A convex combination corresponds to executing $P_j(\lambda^-)$ with probability q^- and $P_j(\lambda^+)$ with probability q^+.

Let C_j^*, O_j^*, R_j^* denote the probability of choice, probability of opening, and expected reward for the convex combination for box j. We have:

$$C_j^* = q^- C_j(\lambda^-) + q^+ C_j(\lambda^+)$$

and so on. Using the definition of α^- and α^+, we obtain:

$$0 = \alpha^- q^- + \alpha^+ q^+ = 2(q^- + q^+) - \sum_j \left(C_j^* + \frac{O_j^*}{k} \right)$$

Since $q^- + q^+ = 1$, we have

$$\sum_j \left(C_j^* + \frac{O_j^*}{k} \right) = 2$$

Next, by weak duality, we have

$$\sum_j R_j(\lambda^-) + \alpha^- \times \lambda^- \geq OPT$$

and

$$\sum_j R_j(\lambda^+) + \alpha^+ \times \lambda^+ \geq OPT$$

Taking a convex combination of these, and noting that $|\lambda^+ - \lambda^-| \leq \epsilon$, we have

$$\sum_j R_j^* + \epsilon \geq OPT$$

This yields an additive approximation to the LP optimum, and yields a "combinatorial" method for solving the LP

2.1.4 Rounding Algorithm

We will now present a different approach to policy design that directly uses the Lagrangian. We do not find the λ that solves the original LP.

Before presenting our choice of λ, we introduce a term corresponding to the net payoff (reward of choice minus cost paid) of the optimal single-box policy.

$$\Phi_j(\lambda) = R_j(\lambda) - \lambda \left(C_j(\lambda) + \frac{O_j(\lambda)}{k} \right) = \max \left(0, \mathbf{E}\left[(X_j - \lambda)^+ \right] - \frac{\lambda}{k} \right)$$

Then we have

$$L(\lambda) = 2\lambda + \sum_j \Phi_j(\lambda)$$

Note that when $\lambda = 0$, $\Phi_j(\lambda) = \mathbf{E}[X_j]$ since the box is always opened and the payoff chosen, and when $\lambda = \infty$, $\Phi_j(\lambda) = 0$ since the box is never opened. Further, note that $\Phi_j(\lambda)$ is a continuous function in λ.

Since the single-box policy deterministically decides whether to open the box, it follows that $O_j(\lambda) \in \{0, 1\}$. We also have

$$O_j(\lambda) = 0 \qquad \Rightarrow \qquad \Phi_j(\lambda) = 0$$

Let λ^* be the value where $\lambda^* = \sum_j \Phi_j(\lambda^*)$. This choice of λ is feasible due to the continuity of $\sum_j \Phi_j(\lambda)$, and hence can be computed to an arbitrarily good approximation by binary search. The following lemma simply uses weak duality, Claim 2.1.1.

Lemma 2.1.2. *$\lambda^* \geq OPT/3$ and $\sum_j \Phi_j(\lambda^*) \geq OPT/3$.*

Amortization. Before presenting the algorithm, we present a different way of accounting for the reward of a policy. Note that

$$R_j(\lambda^*) = \Phi_j(\lambda^*) + \lambda^* \left(C_j(\lambda^*) + \frac{O_j(\lambda^*)}{k} \right)$$

The LHS of the above equality is the expected reward of the policy $P_j(\lambda^*)$. What does the RHS mean? Note that $\Phi_j(\lambda^*)$ is a fixed number that can be computed by solving the Lagrangian at λ^*. Suppose we consider an alternate reward accounting model:

- When box j is encountered, give reward $\Phi_j(\lambda^*)$ upfront.
- If the policy decides to open the box, give it reward $\frac{\lambda^*}{k}$.
- If the policy decides to choose a payoff, instead of getting the payoff, pay the policy λ^*.

The expected reward in the new accounting model is exactly $\Phi_j(\lambda^*)+\lambda^* \left(C_j(\lambda^*) + \frac{O_j(\lambda^*)}{k} \right)$, since the probability the policy opens the box is $O_j(\lambda^*)$, and the probability the policy chooses a payoff is exactly $C_j(\lambda^*)$.

Therefore, as long as policy $P_j(\lambda^*)$ is *executed completely*, the expected reward is the same whether it is accounted for using the LHS (actual reward) or RHS (the new accounting scheme).

Final Policy. Recall that each $P_j(\lambda^*)$ is a policy that deterministically decides to open box j or not. We can discard those boxes j whose optimal policies choose not to open the respective boxes. Let S denote the set of remaining boxes. Note that for each $j \in S$, $O_j(\lambda^*) = 1$. For $j \notin S$, note that $\Phi_j(\lambda^*) = 0$. Note however that $|S|$ could either be more or less than k.

The policy *Dual-Balance* is shown in Figure 2.2.

1. Order the boxes in S arbitrarily.
2. When box $i \in S$ is encountered, execute $P_i(\lambda^*)$ completely.
3. If $P_i(\lambda^*)$ chooses the payoff from i, then choose this payoff and STOP.
4. After $P_i(\lambda^*)$ executes, if k boxes have been opened, the STOP.
5. In all other cases, continue to the next box.

Figure 2.2: Policy *Dual-Balance* for Maximum Value.

Analysis. Note that when the policy *Dual-Balance* encounters box i, it executes policy $P_i(\lambda^*)$ completely and this execution is independent of what happened to previous boxes. Therefore, we can account for the expected reward of this execution using the amortized accounting scheme presented above. By linearity of expectation, we can account for the *entire* expected reward of *Dual-Balance* using the amortized accounting scheme:

- When the policy encounters and opens box i, it earns reward $\Phi_i(\lambda^*) + \lambda^*/k$.
- When it chooses the payoff of box i, it earns payoff λ^*.

To compute the expected reward of *Dual-balance* using the amortized scheme, we compute the reward on any decision path and take expectations over the decision paths. Now, there are three cases that can arise on any decision path of *Dual-Balance*.

1. The policy chooses the payoff of some box. This reward is exactly $\lambda^* \geq OPT/3$.
2. The policy stops because it encountered k boxes. For each box, opening it yields reward λ^*/k. This means the total reward collected is $\lambda^* \geq OPT/3$.
3. The policy stops because it runs out of boxes. Each encountered box i yields reward $\Phi_i(\lambda^*)$, so that the total reward is $\sum_{j \in S} \Phi_j(\lambda^*) = \sum_j \Phi_j(\lambda^*) \geq OPT/3$.

The bounds in each case follow by applying Lemma 2.1.2. Since the amortized reward on any decision path in *Dual-balance* is at least $OPT/3$, the same holds in expectation over all decision paths, lower bounding the amortized reward in expectation by $OPT/3$. Finally, noting that the expected amortized reward is the same as the expected actual reward, we have the following theorem:

Theorem 2.1.3. *The* Dual-balance *policy is a* 3*-approximation to the reward of the optimal decision policy for* Maximum Value.

2.1.5 Adaptivity Gap

Consider the *Dual-Balance* policy and let S' denote the first k boxes in the arbitrary ordering of S; if $|S| < k$, then $S' = S$. It is clear that the policy is only confined to these boxes since it stops when k boxes are opened. The policy opens these boxes sequentially, but we can only improve the reward by opening all boxes in S' and choosing the box with highest payoff. Therefore, we have the following non-adaptive policy that is a 3 approximation: Choose λ^* such that

$$\sum_j \Phi_j(\lambda^*) \equiv \sum_j \max\left(0, \mathbf{E}\left[(X_j - \lambda^*)^+\right] - \frac{\lambda^*}{k}\right) = \lambda^*$$

and let S denote the set of boxes that are opened by their corresponding single-arm policies. Consider up to k boxes in S; open them; and choose the box with highest payoff. This yields the following adaptivity gap result:

Corollary 2.1.4. *There is a non-adaptive policy that is a* 3 *approximation to the optimal adaptive decision policy for* Maximum value.

Note that computing this policy only requires a binary search over λ, and for each λ, the single-arm policies have an easy to compute closed form!

2.2 The Prophet Inequality

The prophet inequality is a classical stopping time problem. In the most basic version of this problem, there are n boxes; box i having reward distribution X_i independent of the reward in other boxes. There is a gambler knows all these reward distributions upfront. He is presented with these boxes one at a time in some adversarial order. When presented with some box i, the gambler learns the value in this box. He has two choices: (1) Choose the value as payoff and exit; or (2) Discard this box and its payoff irrevocably, and ask for the next box.

The goal is to design a stopping policy for the gambler that maximizes his expected payoff. At time t, suppose the current box is i, the payoff in it is v, and the set of boxes not yet presented is S, should the gambler choose this payoff and stop, or continue to the next box?

The first question is: What is a benchmark to compare against? Since there is an adversarial aspect to this problem – the order of presenting boxes is adversarial – the benchmark is also necessarily somewhat adversarial. We compare against a "prophet" who knows the exact values in each box and hence chooses the box with the maximum value. Such a prophet has expected reward

$$\text{Payoff of Prophet } = \mathbf{E}\left[\max_j X_j\right]$$

The classical prophet inequality presents a stopping rule for the gambler that has a simple form: Compute a value w based on the distributions of $X_1, X_2, \ldots, X_n$. Stop and choose the first box whose payoff is at least w. Such a rule is termed a *threshold policy*. There are several threshold policies for which it can be shown that

$$\text{Payoff of Gambler } \geq \frac{1}{2}\mathbf{E}\left[\max_j X_j\right]$$

Such a bound is optimal. Consider the following example:

Example 2.2. *X_1 is the deterministic value 1. X_2 is $\frac{1}{\epsilon}$ with probability ϵ and 0 with the remaining probability. On this example, a prophet obtains reward $\mathbf{E}[\max(X_1, X_2)] = \frac{1}{\epsilon} \times \epsilon + 1 \times (1-\epsilon) = 2 - \epsilon$. If presented with box 1 first, the gambler either chooses the payoff 1 or discards it. But he does it without knowing X_2, so that in either case his expected payoff is $\max(\mathbf{E}[X_1], \mathbf{E}[X_2]) = 1$. Therefore, against a prophet benchmark, no strategy of the gambler has approximation ratio better than 2.*

2.2.1 A Threshold Policy and Analysis

Consider the following threshold policy. Choose w so that

$$w = \sum_j \mathbf{E}\left[(X_j - w)^+\right]$$

In order to analyze this threshold policy, we write an LP relaxation for the prophet, which is essentially an LP relaxation for $\mathbf{E}[\max_j X_j]$. The overall analysis exactly mimics that of *Maximum value* and is only simpler.

Let $z_j(v)$ denote the probability that $X_j = v$ and the prophet chooses this box's payoff. The LP is a simplification of the LP for *Maximum value* since there is no longer the constraint that only k boxes are opened.

$$\boxed{\begin{array}{rcll} \text{Maximize} & & \sum_{j,v} v \cdot z_j(v) & \\ \sum_{j,v} z_j(v) & \leq & 1 & \\ z_j(v) & \in & [0, f_j(v)] & \forall j, v \end{array}}$$

Let OPT denote the value of the LP. Note that $OPT \geq \mathbf{E}[\max_j X_j]$. The only constraints that LP enforces are the following:

- The probability that $X_j = v$ and the payoff of j is chosen is at most $\Pr[X_j = v] = f_j(v)$; and
- In expectation, the prophet chooses the payoff of at most one box.

As before, consider a single box policy P_j which chooses the payoff of j conditioned on $X_j = v$ with probability $z_j(v)/f_j(v)$. A policy is feasible if it is of this form. For policy P_j, let $C(P_j)$ denote the probability the policy chooses the payoff of box j, and let $R(P_j)$ denote the expected reward of this policy Then, the LP relaxation becomes choosing one policy P_j per box j so that

$$\begin{array}{rcl} \text{Maximize} & & \sum_j R(P_j) \\ \sum_j C(P_j) & \leq & 1 \\ \text{Each} & P_j & \text{feasible} \end{array}$$

Now we take the Lagrangian for parameter w to obtain:

$$\boxed{\begin{array}{rcl} \text{Maximize} & & \sum_j \left(R(P_j) - wC(P_j)\right) + w \\ \text{Each} & P_j & \text{feasible} \end{array}}$$

As before, we simplify notation. Let $L(w)$ denote the value of the Lagrangian for parameter w. This Lagrangian optimizes separately for each box. Let $P_j(w)$ denote the optimal Lagrangian policy for box j. The Lagrangian problem says that there is a cost w to choosing the payoff of this box. This means the single-box policy has a very simple form: If the value in the box is more than w, choose this payoff, else not. Let $C_j(w)$ denote the probability the policy chooses the payoff of box j and let $R_j(w)$ denote the expected payoff. We therefore have:

$$C_j(w) = \Pr[X_j \geq w]$$

Denote by $\Phi_j(w)$ the net payoff (reward minus cost) for the optimal policy of box j, we have

$$\Phi_j(w) = \mathbf{E}\left[(X_j - w)^+\right]$$

so that

$$L(w) = w + \sum_j \Phi_j(w)$$

We now choose w^* so that $w^* = \sum_j \Phi_j(w^*) \equiv \sum_j \mathbf{E}\left[(X_j - w^*)^+\right]$. By weak duality, since $L(w) \geq OPT$, we have

$$w^* \geq OPT/2 \qquad \text{and} \qquad \sum_j \Phi_j(w^*) \geq OPT/2$$

Since $R_j(w^*) = \Phi_j(w^*) + wC_j(w^*)$, just as before, we can account for the reward of the single-box policy $P_j(w^*)$ as follows: When box j is encountered, give reward $\Phi_j(w^*)$. If the policy chooses this box, give reward w^*. This accounting preserves the expected reward of $P_j(w^*)$ assuming it is completely executed.

The final *Threshold* policy considers the boxes in arbitrary order (the adversary chooses this order). When box j is encountered, the *Threshold* policy executes $P_j(w^*)$. This policy chooses the payoff if $X_j \geq w^*$. In this case we stop, else we continue to the next box.

To analyze the *Threshold* policy, we use the amortized accounting scheme for the expected reward. Since the execution of $P_j(w^*)$ is independent of how this box is reached, we can apply the accounting scheme for the entire *Threshold* policy. For any decision path of this policy, there are two cases:

- The payoff of some box is chosen. This yields reward $w^* \geq OPT/2$.
- All boxes are opened. Since box j yields payoff $\Phi_j(w^*)$ when encountered, the total reward is $\sum_j \Phi_j(w^*) \geq OPT/2$.

Therefore, the amortized reward is at least $OPT/2$ for any decision path, and hence holds in expectation for the *Threshold* policy. Since the expected reward coincides with the amortized reward, this shows that the *Threshold* policy has value at least $\frac{1}{2}\mathbf{E}[\max_j X_j]$, completing the analysis.

Theorem 2.2.1. *For independent, non-negative, continuous random variables $X_1, X_2, \ldots, X_n$, there exists a value w such that*

$$w = \sum_{i=1}^{n} \mathbf{E}\left[(X_i - w)^+\right] \geq \frac{1}{2}\mathbf{E}\left[\max_{i=1}^{n} X_i\right]$$

If the gambler chooses the first box whose reward is at least w, then his expected payoff is at least $\frac{1}{2}\mathbf{E}[\max_{i=1}^{n} X_i]$.

2.3 Bayesian Pricing

Consider the following algorithmic pricing problem: There are n products each available in unlimited supply. For each product, the valuation of users in the population is drawn from an *independent* distribution. The valuation distribution for product i is X_i. The products are substitutes, so that each user is interested in buying at most one product.

The seller wishes to post prices for the products. Suppose the price of product i is p_i. Consider a user who has valuation profile $\vec{v} \sim \prod_j X_j$. The *utility* this user gets from buying product i is $v_i - p_i$. If this is negative, we assume the user's rational strategy is to not buy. Therefore, the utility of the user for buying product i is *quasi-linear*, that is

$$\text{Utility for buying } i \equiv u(v_i, p_i) = \max(0, v_i - p_i)$$

Given the price vector $\vec{p}$, the user buys that product that maximizes his utility, *i.e.*,

$$\text{Product bought by user with valuation } \vec{v} = \text{ argmax}_i u(v_i, p_i)$$

If the maximum utility is negative, we assume the user does not buy any product.

Given price vector $\vec{p}$, denote by $r(\vec{v}, \vec{p})$ denote the price paid by a user with valuation $\vec{v}$. (Note that the price paid can be 0 if the user does not buy any product.) Then the *revenue* of the seller per user is simply:

$$\text{Seller Revenue } = \mathbf{E}_{\vec{v} \sim \prod_i X_i}\left[r(\vec{v}, \vec{p})\right]$$

This is the expected price paid by a user drawn at random from the population. The goal of the seller is to choose a price vector $\vec{p}$ so that the revenue is maximized. The difficulty with this problem is that the number of possible price vectors is exponential in n, the number of products. This means a brute force search over price vectors is not computationally feasible. We next present a pricing scheme that yields a 4-approximation to the optimal revenue.

2.3.1 LP Relaxation

Our LP relaxation will be remarkably simple, but will do the trick. Let x_{ip} be a binary variable that is 1 if the price of product i is set to p. We relax this to define

$$x_i(p) = \Pr\left[\text{Price of i } = p\right]$$

This simply means that the price of i can be chosen from a distribution and posted; an alternative view is that this is a linear relaxation of the binary variable. Let

$$y_i(p,v) = \Pr\left[X_i = v \text{ and Price of i } = p \text{ and } i \text{ is bought }\right]$$

Since the user only buys the item when $v \geq p$, we assume $y_i(p,v) = 0$ if $v < p$.

The LP relaxation is as follows, where $f_i(v) = \Pr[X_i = v]$. We assume all variables are non-negative.

$$\boxed{\begin{array}{rcll} \text{Maximize} & & \sum_{i,p,v \geq p} p \cdot y_i(p,v) & \\ \sum_{i,p,v\geq p} y_i(p,v) & \leq & 1 & \\ \sum_p x_i(p) & \leq & 1 & \forall i \\ y_i(p,v) & \leq & x_i(p) f_i(v) & \forall i, p, v \geq p \end{array}}$$

Why is this a relaxation? The first constraint simply captures that any user buys at most one product, so this must hold in expectation over all users. The second constraint captures that the $\{x_i(p)\}$ values form a probability distribution for any product i; in other words, i has only one (possibly randomly chosen) price. The final constraint captures that if a user has valuation v for product i and bought this product at price p, then two independent events must have happened:

- The price of the product must have been p; and
- The valuation of the user for item i must have been v

Notice that we did not encode the main constraint that seems to distinguish the problem – the fact that the user chooses the product with largest utility. We are replacing it with two weaker constraints:

- The user buys at most one product; and

- If the user buys a product, the valuation is at least the price.

We will see below that these weaker constraints, written only in expectation, are sufficient to obtain a 4-approximation

2.3.2 Lagrangian and Posted Prices

The above LP is weakly coupled – there is only one constraint that runs across products, and this is

$$\sum_{i,p,v \geq p} y_i(p,v) \leq 1$$

We take the Lagrangian of this constraint with multiplier $\lambda \geq 0$ to get

$$\boxed{\begin{array}{rcll} \text{Maximize} & & \sum_{i,p,v \geq p} (p-\lambda) \cdot y_i(p,v) & \\ \sum_p x_i(p) & \leq & 1 & \forall i \\ y_i(p,v) & \leq & x_i(p) f_i(v) & \forall i, p, v \geq p \end{array}}$$

Let $L(\lambda)$ denote the optimal Lagrangian value. The Lagrangian has no constraints connecting different products together, so we can optimize separately for each one. For any i, the optimal solution has simple structure: If $p \geq \lambda$, then it will set $y_i(p,v)$ to be as large as possible, that is, set $y_i(p,v) = x_i(p) f_i(v)$. Else it will set $y_i(p,v) = 0$. We can use this to eliminate the $y_i(p,v)$ variables.

$$\begin{array}{rcll} \text{Maximize} & & \lambda + \sum_i \sum_{p \geq \lambda, v \geq p} (p-\lambda) \cdot x_i(p) f_i(v) & \\ \sum_p x_i(p) & \leq & 1 & \forall i \end{array}$$

Define $F_i(x) = \sum_{v \geq x} f_i(v)$. Then,

$$\sum_{p \geq \lambda, v \geq p} (p-\lambda) \cdot x_i(p) f_i(v) = \sum_{p \geq \lambda} (p-\lambda) x_i(p) F_i(p)$$

For any i, the optimal solution now sets $x_i(p) = 1$ for that p which maximizes $(p-\lambda)F_i(p)$. This means

$$L(\lambda) = \lambda + \sum_i \left(\max_{p \geq \lambda} (p-\lambda) F_i(p) \right)$$

For continuous random variables with mild assumptions, we can now compute a closed form for the inner maximization by differentiating w.r.t. p. Setting the derivative to 0, we get

$$(p-\lambda) f_i(p) = F_i(p) \qquad \Rightarrow \qquad \lambda = p - \frac{F_i(p)}{f_i(p)}$$

Let $p_i(\lambda)$ denote the optimal price for item i in the Lagrangian.

Definition 2. *For a continuous non-negative random variable X with density function f and $F(x) = \int_{y=x}^{\infty} f(y)dy$, define the* virtual value $\varphi(x)$ *as*

$$\varphi(x) = x - \frac{F(x)}{f(x)}$$

A distribution is said to be regular *if $\varphi(x)$ is monotonically non-decreasing in x.*

Using this definition, we get the following closed form:

$$p_i(\lambda) = \varphi_i^{-1}(\lambda)$$

where $\varphi_i(x) = x - \frac{F_i(x)}{f_i(x)}$.

In summary, for any λ, the optimal Lagrangian solution computes a price $p_i(\lambda)$ for each product i. It sets $x_i(p) = 1$ for $p = p_i(\lambda)$, and for this p, sets $y_i(p, v) = f_i(v)$ for $v \geq p$. In other words, the Lagrangian is a collection of single-product policies. For product i, this policy posts price $p_i(\lambda)$, and the user buys this product whenever his value $v_i \geq p_i(\lambda)$.

2.3.3 The LP Solution

Given that we have a closed form for the Lagrangian, we can compute the LP optimum. In this part, we assume continuous and *regular* distributions for which $\varphi(x)$ is monotone in x. These assumptions are not really necessary for the algorithm or analysis, but are used to simplify the presentation.

The LP enforces the constraint that the expected number of products bought is at most 1. If we consider the Lagrangian at λ, it buys product i with probability $F_i(p_i(\lambda))$, so that the expected number of items bought is $\sum_i F_i(p_i(\lambda))$.

We make a similar argument to that in Section 2.1. As λ increases, $p_i(\lambda) = \varphi_i^{-1}(\lambda)$ is monotonically non-decreasing, so that $F_i(p_i(\lambda))$ is monotonically non-increasing. Since the distributions are continuous, there is a value λ^* such that

$$\sum_i F_i(p_i(\lambda^*)) = 1$$

that can be computed by binary search. Since the expected number of items bought by this Lagrangian solution is 1, this is also the optimal LP solution by weak duality. Therefore, we have the following lemma:

Lemma 2.3.1. *For regular distributions, the LP optimum chooses a value λ^* and sets $p_i(\lambda^*) = \varphi_i^{-1}(\lambda^*) \geq \lambda^*$. It sets $x_i(p) = 1$ for $p = p_i(\lambda^*)$ and $y_i(p, v) = f_i(v)$ for $p = p_i(\lambda^*)$ and $v \geq p$. The value λ^* satisfies*

$$\sum_i F_i(p_i(\lambda^*)) = 1$$

The value of the LP optimum is given by

$$\textit{LP Optimum } = \sum_i p_i(\lambda^*) F_i(p_i(\lambda^*))$$

2.3.4 Final Pricing Scheme

For each product i, with probability $1/2$ (independently of other products), set its price to ∞, and with probability $1/2$ set its price to $p_i(\lambda^*)$. In other words, for each product, discard it with probability $1/2$ and use the LP's price with probability $1/2$. A user simply chooses the product that provides maximum utility.

Theorem 2.3.2. *For regular distributions, the above pricing scheme is a* 4*-approximation to the expected revenue of the optimal pricing scheme.*

Proof. Consider any product i. A user buys this product if all the following hold:

- $v_i \geq p_i(\lambda^*)$;
- Product i is not discarded;
- For all $j \neq i$, either j is discarded or $v_j < p_j(\lambda^*)$

For any $j \neq i$,

$$\Pr\left[j \text{ is not discarded and } v_j \geq p_j(\lambda^*)\right] = \frac{1}{2} F_j(p_j(\lambda^*))$$

By union bounds,

$$\Pr\left[\exists j \neq i \text{ s.t. the user can potentially buy } j\right] \leq \sum_j \frac{1}{2} F_j(p_j(\lambda^*)) \leq \frac{1}{2}$$

Therefore, with probability at least $1/2$, for all $j \neq i$, either j is discarded or $v_j < p_j(\lambda^*)$. In this event, i is not discarded with probability $1/2$, and with probability $F_i(p_i(\lambda^*))$, $v_i \geq p_i(\lambda^*)$. Combining these probabilities together, the expected revenue from product i is at least:

$$\text{Revenue from product } i \geq \frac{1}{2} \times \frac{1}{2} \times F_i(p_i(\lambda^*)) \times p_i(\lambda^*)$$

The total revenue is the sum of the revenues from different products by linearity of expectation. Since the LP optimum is $\sum_i p_i(\lambda^*) F_i(p_i(\lambda^*))$, this completes the proof of the 4 approximation. □

Chapter 3

Submodularity

In this chapter, we introduce a concept that is widely used for the design of algorithms when faced with stochastic inputs. The main idea is to abstract out the details of the stochastic function and instead use a generic concept for set functions. This generic concept is termed *submodularity* and has wide applicability.

Consider designing a non-adaptive algorithm for the *Maximum Value* problem. In the previous chapter, we showed that the dual of a simple LP formulation yielded such an algorithm and showed the adaptivity gap as well. Suppose we did not care about the adaptivity gap, but instead simply wanted to an approximation to the optimal non-adaptive strategy, then we can formulate the problem somewhat differently.

The optimal non-adaptive strategy, call it OPT, can be written as:

$$OPT = \max_{S \subseteq \{1,2,\ldots,n\} | |S| \le k} \mathbf{E}\left[\max_{i \in S} X_i\right]$$

This captures choosing a subset S of size k upfront, so that if these boxes are opened and the maximum observed value is chosen, the expected payoff is maximized.

This non-adaptive problem is now a subset selection problem and falls squarely within the complexity class NP. For constructing decision trees, we could not make such a statement, since the size of the optimal decision tree could be exponentially large in the size of the input. It can be shown that the optimal non-adaptive policy is actually NP-Complete; nevertheless, let us explore the structure of this optimization problem.

3.1 Submodularity

Definition 1. *A set function $f : 2^{[n]} \to \Re^+$ is said to be sub-modular if for all $S_1 \subset S_2 \subseteq \{1, 2, \ldots, n\}$ and $i \notin S_2$, it holds that:*

$$f(S_1 \cup \{i\}) - f(S_1) \ge f(S_2 \cup \{i\}) - f(S_2)$$

Furthermore, such a function is said to be monotone if $f(S_1) \le f(S_2)$.

Submodularity is a discrete analog of concavity in the sense that it captures diminishing returns – adding a new element to a larger set only causes the function to increase by a lesser amount. But this connection is not very precise; in fact, submodular functions resemble convex functions as well – both can be minimized efficiently while the same is not true for concave functions.

Going back to *Maximum Value*, we can show the following; we will use this proof idea subsequently to show submodularity of several other stochastic set functions.

Lemma 3.1.1. *For non-negative distributions $X_1, X_2, \ldots, X_n$, the set function $f(S) = \mathbf{E}[\max_{i \in S} X_i]$ is monotone and submodular.*

Proof. It is easy to check that $f(S)$ is monotone. Let σ denote some sample from the joint distribution over $X_1, X_2, \ldots, X_n$. Let $X_i(\sigma)$ denote the value of random variable X_i in sample σ. Then

$$\mathbf{E}\left[\max_{i \in S} X_i\right] = \sum_{\sigma} p(\sigma) \max_{i \in S} X_i(\sigma)$$

where $p(\sigma)$ denotes the probability of sample σ. Let $f_\sigma(S) = \max_{i \in S} X_i(\sigma)$. It is easy to check that for $S' \subset S$ and $t \notin S$,

$$f_\sigma(S' \cup \{t\}) - f_\sigma(S') \geq f_\sigma(S \cup \{t\}) - f_\sigma(S) \qquad \forall \sigma$$

The above inequality is preserved by taking linear combinations. Therefore, $f(S)$ is also submodular. □

Note that the above result does not require the distributions to be independent. In fact, the approximation guarantee we present next works even when distributions are correlated.

3.2 The Greedy Algorithm

The main reason submodular functions are interesting in the context of stochastic optimization is that they are amenable to simple greedy optimization strategies. Consider the problem of choosing a set S of size at most k that maximizes $f(S)$. If $f(S) = \mathbf{E}[\max_{i \in S} X_i]$, then this is exactly the non-adaptive optimum for *Maximum Value*. The natural greedy algorithm is shown in Figure 3.1.

Lemma 3.2.1. *Let $S^* \leftarrow argmax_{Q||Q|=k} f(Q)$ and let S denote the subset chosen by the greedy algorithm. Then, for any monotone, non-negative, submodular set function f with $f(\Phi) = 0$, we have:*

$$f(S) \geq (1 - \frac{1}{e}) f(S^*)$$

Proof. The proof uses a fairly general technique: At any step of the greedy algorithm, suppose we added the entire optimal solution to the current solution, then the final

```
1. $S \leftarrow \Phi$.
2. For $j = 1, 2, \ldots, k$ do:
    (a) Let $i \leftarrow \text{argmax}_{t \notin S} f(S \cup \{t\}) - f(S)$.
    (b) $S \leftarrow S \cup \{i\}$
3. EndFor
4. Output $S$.
```

Figure 3.1: The Greedy Algorithm for Submodular Maximization.

solution will have value at least $f(S^*)$. But submodularity will imply there is one element that yields at least $1/k$ of this increase.

At the i^{th} step of the greedy algorithm, let S_i denote the set found. Note that $S_k = S$. Order the elements of the optimal solution arbitrarily so that $S^* = \{y_1, y_2, \ldots, y_k\}$. Just after the $(i-1)^{st}$ step, instead of making the greedy choice, suppose we add the elements of S^* one by one. Let

$$z_{ij} = f(S_{i-1} \cup \{y_1, y_2, \ldots, y_j\}) - f(S_{i-1} \cup \{y_1, y_2, \ldots, y_{j-1}\})$$

denote the increase by adding y_j. Adding up all these increases,

$$f(S^* \cup S_{i-1}) - f(S_{i-1}) = \sum_{j=1}^{k} z_{ij}$$

By monotonicity, we have

$$f(S^*) - f(S_{i-1}) \leq f(S^* \cup S_{i-1}) - f(S_{i-1}) = \sum_{j=1}^{k} z_{ij}$$

By averaging, there exists j^* such that

$$z_{ij^*} = f\left(S_{i-1} \cup \{y_1, y_2, \ldots, y_{j^*}\}\right) - f\left(S_{i-1} \cup \{y_1, y_2, \ldots, y_{j^*-1}\}\right) \geq \frac{f(S^*) - f(S_{i-1})}{k}$$

Now we use submodularity. The increase if we added y_{j^*} to S_{i-1} is only larger than the increase in adding it to $S_{i-1} \cup \{y_1, y_2, \ldots, y_{j^*-1}\}$. Therefore,

$$f\left(S_{i-1} \cup \{y_{j^*}\}\right) - f\left(S_{i-1}\}\right) \geq \frac{f(S^*) - f(S_{i-1})}{k}$$

The greedy algorithm's choice of i is at least as good. Therefore,

$$f(S_i) - f(S_{i-1}) \geq \frac{f(S^*) - f(S_{i-1})}{k}$$

Rearranging, we have

$$f(S_i) \geq (1 - \frac{1}{k}) f(S_{i-1}) + \frac{1}{k} f(S^*)$$

Assuming $f(\Phi) = 0$, we can now check by induction that:

$$f(S_i) \geq \left(1 - \left(1 - \frac{1}{k}\right)^i\right) f(S^*)$$

so that $f(S) \geq (1 - (1 - \frac{1}{k})^k) f(S^*) \geq (1 - \frac{1}{e}) f(S^*)$. □

As a corollary, we have the following result:

Corollary 3.2.2. *The greedy algorithm is a $e/(e-1)$ approximation to the optimal non-adaptive strategy for* Maximum Value

3.3 Examples of Submodular Functions

There are several subset selection problems where the underlying stochastic set function is submodular. We present some examples that have been widely studied.

3.3.1 Joint Entropy

Consider a *sensor placement* problem, there are n candidate locations where sensors can be placed, and $k \ll n$ sensors. The locations sense correlated data, and the goal is to place the sensors in a fashion that maximizes the amount of information they capture about the physical phenomena being sensed.

Formally, assume location i senses data that follows discrete distribution X_i. Denote by $\mathbf{x}$ a vector of possible sensed values of all n locations, and let $p(\mathbf{x})$ denote the joint probability distribution over $\mathbf{x}$. Denote subsets of locations by S, and $\mathbf{x}_S$ the corresponding sensed values. Now, S has high information content if p_S is very "spread out", and this notion is captured by the *entropy* of the joint distribution.

$$H(S) = \sum_{\mathbf{x}_S} p(\mathbf{x}_S) \log_2 \frac{1}{p(\mathbf{x}_S)} = \mathbf{E}_{\mathbf{x}_S} \left[\log_2 \frac{1}{p(\mathbf{x}_S)}\right]$$

Why is entropy a measure of information content or uncertainty in a random variable? Intuitively, consider two distributions X_1 and X_2 that are perfectly correlated. This means that conditioned on $X_1 = v$, we have $X_2 = v$ with probability 1 and vice versa. Then it is easy to check that $H(X_1, X_2) = H(X_1)$. This means that the information content (or uncertainty) in $X_1, X_2)$ is the same as that for X_1. In other words, conditioned on knowing X_1, there is no uncertainty in X_2.

Similarly, if X_1 and X_2 are *i.i.d.*, then $H(X_1, X_2) = 2H(X_1)$. This means that (X_1, X_2) has twice the information content (or uncertainty) as X_1 or X_2. Or, conditioned on knowing X_1, the uncertainty in X_2 remains the same as before since it is an independent variable. Therefore, the entropy is larger if distributions are independent compared to when they are perfectly correlated.

More formally, it can be shown that entropy of a discrete random variable X is the minimum number of bits per symbol needed to encode symbols drawn $i.i.d.$ according to the distribution of X. The larger the entropy, the less is the distribution "compressible" and hence the larger the information content. We present more details in Chapter **??** when we discuss data compression.

The goal of sensor placement is to choose a subset S of locations where $|S| \leq k$ such that the entropy $H(S)$ is maximized. This roughly corresponds to trying to find locations whose observations are as uncorrelated as possible. As before, this function is monotone and submodular.

Lemma 3.3.1. *For discrete distributions $X_1, X_2, \ldots, X_n$, let $H(S)$ denote the joint entropy of the random variables in set $S \subseteq \{X_1, X_2, \ldots, X_n\}$. Then $H(S)$ is a monotone submodular function.*

Proof. Note that

$$H(S \cup \{X_i\}) - H(S) = \sum_{\mathbf{x}_S, x_i} p(\mathbf{x}_S, x_i) \log_2 \frac{1}{p(x_i|\mathbf{x}_S)} \equiv H(X_i|S)$$

Therefore, submodularity for H can be restated as:

$$T \subset S \text{ and} X_i \notin S \qquad \Rightarrow \qquad H(X_i|T) \geq H(X_i|S)$$

To show this, we can assume w.l.o.g. that $T = \Phi$. Therefore, we need to show that:

$$H(X_i|S) \leq H(X_i)$$

Note now that

$$H(X_i|S) = \mathbf{E}_{\mathbf{x}_S}\left[\sum_{x_i} p(x_i|\mathbf{x}_S) \log_2 \frac{1}{p(x_i|\mathbf{x}_S)}\right] = \sum_{x_i} \mathbf{E}_{\mathbf{x}_S}\left[p(x_i|\mathbf{x}_S) \log_2 \frac{1}{p(x_i|\mathbf{x}_S)}\right]$$

Consider the function $g(r) = r\log_2(1/r)$. This is a concave function for $r \in [0,1]$. For a concave function, we have Jensen's inequality: $\mathbf{E}[g(r)] \leq g(\mathbf{E}[r])$. Applying this to the above equality, we have:

$$\sum_{x_i} \mathbf{E}_{\mathbf{x}_S}\left[p(x_i|\mathbf{x}_S) \log_2 \frac{1}{p(x_i|\mathbf{x}_S)}\right] \leq \sum_{x_i}\left(\mathbf{E}_{\mathbf{x}_S}[p(x_i|\mathbf{x}_S)] \log_2 \frac{1}{\mathbf{E}_{\mathbf{x}_S}[p(x_i|\mathbf{x}_S)]}\right)$$

Note now that $\mathbf{E}_{\mathbf{x}_S}[p(x_i|\mathbf{x}_S)] = p(x_i)$. Therefore, we have:

$$\sum_{x_i}\left(\mathbf{E}_{\mathbf{x}_S}[p(x_i|\mathbf{x}_S)] \log_2 \frac{1}{\mathbf{E}_{\mathbf{x}_S}[p(x_i|\mathbf{x}_S)]}\right) = \sum_{x_i} p(x_i) \log_2 \frac{1}{p(x_i)} = H(X_i)$$

Combining all the above relations, we have $H(X_i|S) \leq H(X_i)$ completing the proof.

Note that

$$H(S \cup \{X_i\}) - H(S) = H(X_i|S) = \mathbf{E}_{\mathbf{x}_S}\left[H(X_i|\mathbf{x}_S)\right]$$

The RHS is the entropy of the discrete random variable $X_i|\mathbf{x}_S$. By definition, this is non-negative. Therefore, $H(S \cup \{X_i\}) - H(S) \geq 0$, showing monotonicity. $\square$

Submodularity for H is sometimes called the "information never hurts" principle. Having more information, in this case having observations from a larger set of locations S, will not increase the uncertainty in the value that will be observed at X.

3.3.2 Influence and Epidemics

A natural strategy for advertising a product to a given population works as follows. The advertisers target a subset of people and convince them to use the product. The initial targets in turn influence other people to become new customers. The process continues, and more individuals adopt the product due to a cascading effect. We model this situation as the spread of an epidemic in a *social network*.

More formally, consider a directed graph where each node denotes an individual. There is a directed edge (u, v) if u *influences* v; the edge weight $p_{uv} \in [0, 1]$ measuring the degree of influence. Whenever a person becomes a customer, the corresponding node is termed *active*. A subset of nodes S are chosen to be active at the beginning. The set of active nodes grow according to some random process. Given an initial set of active nodes S, let S_f denote the random final set of active nodes. Define $f(S)$ to be the expected size of S_f, that is, $f(S) = E[|S_f|]$. For an integer k, the objective is to find some S of size k at which $f(S)$ is maximized. We will consider two different random processes, namely *Independent Cascade Model* and *Linear Threshold Model*, and show that the function $f(S)$ is submodular in both situations. This leads to a $(1 - 1/e)$ approximation using the greedy algorithm.

Independent Cascade Model. In this model, we are given edge probability p_{uv} for each edge (u, v). Whenever node u becomes active, it gets a one-time opportunity to activate v with probability p_{uv}, independently of past history and outcomes at the other edges leaving u. If node v was already active, it remains active forever. Think of "active" as buying the product.

This model is directly borrowed from epidemiology where it is termed the *Susceptible-Infected* (SI) model. Here, p_{uv} is the probability that conditioned on u being infected, it infects v in their one time interaction. Once v is infected, it stays infected and gets one opportunity to infect each of its neighbors. (Of course, maximizing the spread of an infection makes little sense, so we stick to the advertising motivation.)

In this model, let S denote the starting set of nodes that are active/infected. We let the random process evolve till each infected node has received one opportunity to infect each of its neighbors. Let $f(S)$ denote the expected number of active nodes at the end. We will show $f(S)$ is a submodular function. As with the *Maximum value* problem, the proof follows by considering all possible scenarios and taking expectations.

Towards this end, consider an alternative random process. Initially all the edges in the graph are in *blocked* state. Each edge (u, v) is switched into *live* state independently with probability p_{uv}. The set of live edges define a random subgraph σ. Let S_σ denote the random set of nodes reachable from S in σ.

Lemma 3.3.2. *Given an initial set of active nodes S, the set of nodes reachable from S in the random subgraph σ follows the same distribution as that of the random final set of active nodes in independent cascade model.*

Proof. In the independent cascade model, whenever a node u becomes active, for each node v adjacent to u, we perform a coin toss that succeeds with probability p_{uv} and makes v alive. The key observation is that we can perform all such coin tosses upfront and it will not change the final outcome. A node u will become active if and only if there is a path from some $s \in S$ to u such that every node in the path becomes active and the coin tosses across every edge in the path result in success. The lemma follows. $\square$

Theorem 3.3.3. *The function $f(.)$ is monotone and submodular in the independent cascade model.*

Proof. For any given set S, and some node $t \notin S$,

$$\begin{aligned} f(S \cup \{t\}) - f(S) &= E_\sigma[\text{Number of nodes reachable from } S \cup \{t\} \text{ in } \sigma] \\ &\quad -E_\sigma[\text{Number of nodes reachable from } S \text{ in } \sigma] \\ &= E_\sigma[\text{Number of nodes reachable from } t \text{ and not from } S \text{ in } \sigma] \end{aligned}$$

The first equality follows from Lemma 3.3.2, while the second equality follows from linearity of expectation. Consider two different sets of nodes A, B with $A \subseteq B$ and some node $t \notin B$. For every realization of the random graph σ, number of nodes reachable from t and not from A will be at least the number of nodes reachable from t and not from B. Thus, $f(A \cup \{t\}) - f(A) \geq f(B \cup \{t\}) - f(B)$, and $f(.)$ is submodular. The proof of monotonicity is straightforward. $\square$

Linear Threshold Model. This model is based on nodes having different "susceptibilities" for getting infected. In the beginning, each node v chooses a threshold value θ_v uniformly and independently at random from the interval $[0, 1]$. The set of active nodes grow at discrete time steps. Define A_t to be the set of active nodes at time step t. At step 0, $A_0 = S$, the initial set of active nodes. At step t, an inactive node v becomes active if sum of the incoming edge weights from already active nodes exceeds its threshold value, that is, if $\sum_{u:u \in A_{t-1}} p_{uv} \geq \theta_v$. After this time, the node stays active. Note that once all the threshold values are chosen, the process is deterministic.

As before, $f(S)$ will denote the expected size of A_∞ given that $A_0 = S$. We will show $f(S)$ is monotone and submodular. Towards this end, we consider an alternative random process that is stochastically identical and more closely resembles the independent cascade model.

In this new process, initially all the edges are marked *blocked*. Now every node v marks one of its incoming edges (u, v) as *live* with probability p_{uv}. Let τ be the random subgraph defined by the collection of live edges. Given the initial set S, $f(S)$ will be the number of vertices in the subgraph that are reachable from S.

Lemma 3.3.4. *In linear threshold model, given the initial set of active nodes S, the random final set of active nodes follow the same distribution as that of the set of nodes reachable from S in the random subgraph τ.*

Proof. Let B_k denote the set of active nodes that are at most k hops away from S in the random subgraph τ. The proof is by induction on k, the number of time steps. Initially $A_0 = B_0 = S$. For the inductive step, suppose A_i and B_i follow the same distribution for all $i \leq k$. We will show that A_{k+1} and B_{k+1} follow the same distribution. The proof will follow due to Claims 3.3.5 and 3.3.6 proved below. $\square$

Claim 3.3.5. *For every node v, we have*

$$\Pr[v \in A_{k+1} | v \notin A_k] = \frac{\sum_{u \in A_k \setminus A_{k-1}} p_{uv}}{1 - \sum_{u \in A_{k-1}} p_{uv}}$$

Proof. $v \notin A_k$ if and only if edges coming into v from A_{k-1} fail to activate v, that is, when $\sum_{u \in A_{k-1}} p_{uv} \leq \theta_v$. Since θ_v is chosen independently at random from $[0, 1]$, we have $\Pr[v \notin A_k] = 1 - \sum_{u \in A_{k-1}} p_{uv}$.

$v \in A_{k+1} \setminus A_k$ if and only if the edges coming into v from A_k succeed in activating v but edges from A_{k-1} fail to do so. This happens only when $\sum_{u \in A_k} p_{uv} \geq \theta_v$, and $\sum_{u \in A_{k-1}} p_{uv} \leq \theta_v$; or equivalently, when θ_v lies in an interval of span $\sum_{u \in A_k \setminus A_{k-1}} p_{uv}$. Since θ_v is chosen uniformly and independently at random from $[0, 1]$, we have $\Pr[v \in A_k \notin A_{k-1}] = \sum_{u \in A_k \setminus A_{k-1}} p_{uv}$.

Note that $\Pr[v \in A_{k+1} | v \notin A_k] = \Pr[v \in A_{k+1} \setminus A_k] / Pr[v \notin A_k]$, and the claim follows. $\square$

Claim 3.3.6. *For every node v, we have*

$$\Pr[v \in B_{k+1} | v \notin B_k] = \frac{\sum_{u \in B_k \setminus B_{k-1}} p_{uv}}{1 - \sum_{u \in B_{k-1}} p_{uv}}$$

Proof. Note that $v \notin B_k$ if and only if v does not mark any incoming edge from B_{k-1}. Since v marks the incoming edges (u, v) in a mutually exclusive manner with corresponding probabilities p_{uv}, we have $\Pr[v \notin B_k] = 1 - \sum_{u \in B_{k-1}} p_{uv}$.

Next note that $v \in B_{k+1} \setminus B_k$ if and only if v does not mark any incoming edge from B_{k-1}, but marks some incoming edge from $B_{k+1} \setminus B_k$. This happens with probability $\sum_{u \in B_{k+1} \setminus B_k} p_{uv}$.

Finally note that $\Pr[v \in B_{k+1} | v \notin B_k] = \Pr[v \in B_{k+1} \setminus B_k] / Pr[v \notin B_k]$, and the claim follows. $\square$

Now that we have proved Lemma 3.3.4, the next theorem follows by applying the same idea as Theorem 3.3.3.

Theorem 3.3.7. *The function $f(.)$ is monotone and submodular in linear threshold model.*

Chapter 4

Greedy Adaptive Policies: Stochastic Set Cover

So far, we have used weakly coupled relaxations and submodularity to design policies that compute a fixed ordering of the options. We now present problems for which such orderings that are computed upfront can be hugely suboptimal. This motivates the need to compute *adaptive* policies. We will present greedy design techniques in the next few chapters, and subsequently present techniques based on linear programming. In this chapter, we will consider a stochastic version of the classical set cover problem, and show that the greedy algorithm for set cover has a simple but non-trivial modification that achieves a provably good adaptive policy. Our analysis will use ideas from submodularity in the previous chapter – append the optimal solution and argue that some part of it could have been added by the greedy solution to its advantage. However, in the stochastic setting, this is easier said than done.

4.1 Classical Set Cover

We first present the deterministic set cover problem and its analysis. The *Set Cover* problem is closely related to submodular maximization. In this problem, there is a universe U of n elements $\{e_1, e_2, \ldots, e_n\}$. There are m sets $S_1, S_2, \ldots, S_m \subseteq U$. Set i has cost c_i. The goal is to choose the collection of sets such that their union is U and the sum of the costs of these sets is as small as possible. The reason this problem is called *Set Cover* is that we can think of a setS_i as covering an element e_j if $e_j \in S_i$. The goal then becomes to choose the cheapest collection of sets to cover all elements.

A closely related problem is *Maximum Coverage*. Here all costs are unit, and the goal is to choose k sets so that size of their union is as large as possible. Let T denote a collection of sets and let $f(T)$ denote the size of their union. It is easy to check that $f(T)$ is a monotone submodular function. Therefore, the greedy algorithm yields a $e/(e-1)$ approximation to *Maximum Coverage*. It is interesting to note that under mild complexity assumptions, this is the best possible approximation ratio that can be achieved in polynomial time.

Set Cover is a minimization problem, nevertheless we can apply the same greedy algorithm. At some step, suppose T is the collection of sets chosen so far. An element e_j is uncovered if $e_j \notin \cup_{S \in T} S$. For set $S_i \notin T$, define the per-unit cost as

$$\text{Per-unit cost of } S_i = \frac{c_i}{|\{e \in S_i \text{ and } e \text{ is uncovered }\}|}$$

Let S^* denote the set with minimum per-unit cost; add S^* to T. Repeat this procedure till there are no uncovered elements.

4.1.1 Direct Analysis

Let $GREEDY$ denote the total cost of the sets chosen by the greedy solution. Let OPT denote the corresponding quantity for the optimal solution. For an element e, let price$[e]$ denote the per-unit cost of the set that ends up covering e. By the definition of per-unit cost, we have:

$$GREEDY = \sum_{e \in U} \text{price}[e]$$

Lemma 4.1.1. *Order the elements of U in the order in which they are covered by the greedy algorithm as $e_1, e_2, \ldots, e_n$. Then*

$$\textit{price}[e_j] \leq \frac{OPT}{n - j + 1}$$

Proof. The idea of the proof is the same as that for submodular maximization from the previous chapter. Consider the point in time when greedy is covering e_j. Let $S_1, S_2, \ldots, S_i$ denote the sets chosen by the greedy algorithm, where S_i covers e_j. Let $\Psi = \{S_1, S_2, \ldots, S_{i-1}\}$. Consider adding the entire optimal solution O to Ψ. Clearly this will cover the remaining $n - j + 1$ elements $\{e_j, e_{j+1}, \ldots, e_n\}$.

For $S \notin \Psi$, let n_S denote the coverage of S if it is added immediately after Ψ. Since the sets in O together cover the $n - j + 1$ elements, we must have:

$$n - j + 1 \leq \sum_{S \in O \setminus \Psi} n_S$$

Note that the above relation is an inequality since n_S is the coverage if S is added immediately after Ψ, where it can possibly cover more elements than its marginal coverage in the optimal solution.

By definition (and denoting the cost of a generic set S by c_S), we have

$$OPT \geq \sum_{S \in O \setminus \Psi} c_S$$

Dividing these relations, we have

$$\frac{OPT}{n - j + 1} \geq \min_{S \in O \setminus \Psi} \frac{c_S}{n_S}$$

Now the greedy algorithm chooses that $S \notin \Psi$ that minimizes c_S/n_S, and since this minimum value is price$[e_j]$, we have

$$\frac{OPT}{n-j+1} \geq \min_{S \in O \setminus \Psi} \frac{c_S}{n_S} \geq \min_{S \notin \Psi} \frac{c_S}{n_S} \equiv \text{price}[e_j]$$

□

Theorem 4.1.2. *The greedy algorithm is a* $1 + \log n$ *approximation for the* Set Cover *problem.*

Proof. We just sum the bound in the previous lemma over all elements:

$$GREEDY = \sum_{j=1}^{n} \text{price}[e_j] \leq \sum_{j=1}^{n} \frac{OPT}{n-j+1} \leq OPT \times (1 + \log n)$$

□

4.1.2 Dual Fitting

The greedy algorithm actually has a stronger guarantee: It is a $1+\log n$ approximation against not just the optimum solution, but also a linear programming relaxation of the optimal solution. Since the latter value can often be smaller, this will show that the greedy algorithm can perform better than what the previous analysis implies.

The analysis idea we present will be used later in the book as well, and is a powerful technique for showing performance bounds for greedy schemes. The basic idea is to take the dual of the LP relaxation for set cover. Any feasible dual solution will be at most the value of the LP optimum by weak duality, and hence at most OPT. Now we define a feasible dual solution D whose value is at least a factor $1/\alpha$ of $GREEDY$, where $\alpha \geq 1$. Then we have the sequence of inequalities:

$$OPT \geq \text{ LP Optimum } \geq D \geq \frac{GREEDY}{\alpha}$$

where the first inequality is because the LP is a relaxation of a minimization problem and the second inequality is because of weak duality. This now implies that $GREEDY \leq \alpha \times OPT$. We will exhibit such a feasible solution for $\alpha \leq 1 + \log n$.

The first step is to write the LP relaxation for set cover. There is a variable x_S for every set S which is set to 1 if S belongs to the final collection, and 0 otherwise. Since we need each element to be covered, the sum of the x_S values for all sets that contain an element must be at least 1. This yields the following relaxation:

$$\begin{array}{rcll} \text{Minimize} & & \sum_S c_S x_S & \\ \sum_{S|e \in S} x_S & \geq & 1 & \forall e \in U \\ x_S & \geq & 0 & \forall S \in \{S_1, S_2, \ldots, S_m\} \end{array}$$

It is clear that the optimum solution to this LP is at most OPT. We next take the dual of this program, where there is a multiplier y_e for each element $e \in U$.

$$\boxed{\begin{array}{rcll} \text{Maximize} & & \sum_{e\in U} y_e & \\ \sum_{e\in S} y_e & \le & c_S & \forall S \in \{S_1, S_2, \ldots, S_m\} \\ y_e & \ge & 0 & \forall e \in U \end{array}}$$

Theorem 4.1.3. *There is a feasible solution to the dual program whose value is at least* $\frac{GREEDY}{1+\log n}$. *This implies that the greedy algorithm is a* $1 + \log n$ *approximation to* Set Cover.

Proof. We use the definition of price$[e]$ from the previous analysis. Define $y_e = \frac{\text{price}[e]}{1+\log m}$. Since $GREEDY = \sum_e \text{price}[e]$, we have $\sum_e y_e = \frac{GREEDY}{1+\log m}$. All we need to show is that this setting of $\{y_e\}$ is feasible for the constraints of the dual solution.

Consider some set $S \in \{S_1, S_2, \ldots, S_m\}$. (Note that this is any set, not just those in the optimal or greedy solutions.) Number the elements of *this set* in the order in which greedy covers them as $e_1, e_2, \ldots, e_k$, where $k = |S|$. Consider the point at which e_j is being covered. At this point, the per-unit cost of S is at most $c_S/(k-j+1)$ since there are $k - j + 1$ elements of S that are still uncovered. Since greedy chooses a set with minimum per unit cost and this set covers e_j, we have

$$\text{price}[e_j] \le \frac{c_S}{k - j + 1}$$

Summing this over all elements in S, we have

$$\sum_{j=1}^{k} \text{price}[e_j] \le \sum_{j=1}^{k} \frac{c_S}{k-j+1} \le c_S \times (1 + \log k) \le c_S \times (1 + \log n)$$

Since $y_e = \frac{\text{price}[e]}{1+\log m}$, this shows that $\sum_{e\in S} y_e \le c_S$, completing the proof. ☐

4.2 Evaluating Conjunctive Queries with Shared Filters

We now consider a stochastic version of set cover. We present it in terms of a practical problem arising in the context of databases. In the *conjunctive query evaluation problem*, we have n Boolean predicates or *filters* $F_1, \ldots, F_n$, where filter F_i has cost of evaluation c_i and evaluates to *True* with probability p_i (also called the selectivity of the filter). We assume that these probabilities are independent.

There are m *queries* $Q_1, \ldots, Q_m$. Each query Q_j is the Boolean conjunction of a subset R_j of the filters. The goal is to determine a possibly adaptive order of evaluating the filters so that the expected cost spent in evaluating *all* the queries is minimized. The expectation is with respect to the random outcome of evaluating a filter (which is independent of the outcome of evaluation of all other filters).

Example 4.1. *Suppose we have only one query, $Q = F_1 \wedge F_2$. Then, Q is* False *if either F_1 or F_2 evaluates to* False. *Suppose we want to evaluate Q. We could evaluate F_1 and F_2 simultaneously, but if these predicates are expensive to evaluate, a cheaper solution would be to evaluate them sequentially. Suppose we evaluate F_1 first. Then this costs c_1 and with probability $1 - p_1$, $F_1 =$* False *so that Q evaluates to* False. *If F_1 evaluates to* True, *then we will not know the value of Q unless we evaluates F_2 as well, and this costs c_2. Therefore the expected cost of this evaluation plan is $c_1 + p_1c_2$. On the other hand, we could have decided to evaluate F_2 first and only evaluated F_1 if F_2 returned* True. *This plan has expected cost $c_2 + p_2c_1$. An optimal policy would choose the cheaper of the two plans.*

In the above example, there is only one conjunctive query to evaluate. In this setting, the ordering of filters that minimizes expected cost of evaluation is easy: Order the filters in increasing order of $\frac{c_i}{1-p_i}$. (See below for a proof.)

We now consider the case where there are multiple conjunctive queries each over a possibly different subset of the filters, and the goal is to minimize the expected cost of evaluating *all* the queries. The challenge is that the queries *share* filters, so that evaluating a filter can potentially answer several queries at once.

4.2.1 Fixed Orderings are Sub-optimal

Intuitively, we should evaluate filters with low cost and low selectivity upfront. One question in this regard is the benefit of being adaptive in the evaluation of filters. In particular, consider the following space of solutions: Order the filters somehow, and evaluate in this order, skipping filters whose evaluation is not necessary given the outcomes of filters evaluated so far. This generalizes the optimal ordering policy for a single query. How good is this space of strategies? Let μ denote the maximum number of queries that a filter can be part of.

Theorem 4.2.1. *Any fixed ordering solution is a $\Omega(\mu)$ approximation to the optimal adaptive ordering.*

Proof. Suppose there are n filters $F_1, \ldots, F_n$, each having *zero* cost and selectivity $1/n$. There are n additional filters $H_1, \ldots, H_n$, each having *unit* cost and selectivity *zero*. There are $m = n^2$ queries, partitioned into n disjoint groups $G_1, \ldots, G_n$. Each group G_i contains n queries $Q_{i1}, \ldots, Q_{in}$. A query Q_{ij} contains the filters F_i, H_i and H_j. Note that $\mu = \Theta(n)$. Furthermore, if F_i *does not* pass an item, it resolves all the queries in group G_i. Let *OPT* denote the optimal adaptive ordering. It works as follows.

The optimal solution evaluates the filters $F_1, \ldots, F_n$. For all groups G_i that are not resolved in the first phase, evaluate filter H_i. Since each F_i has selectivity $1/n$, expected number of groups that are not resolved in the first phase equals $n \times (1/n) = 1$. The total cost incurred in the first phase is zero. The total cost incurred in the second phase equals the number of groups surviving the first phase. Thus, $E[\text{cost}(\textit{OPT})] = 1$.

Without loss of generality, we can assume that a fixed ordering solution evaluates the filters in the order $F_1, \ldots, F_n, H_1, \ldots, H_n$. The probability that all but one group is resolved on evaluating $F_1, \ldots, F_n$ is given by $n \times (1/n) \times (1-1/n)^{n-1} \approx (1/e)$. Let the unresolved group be G_i. In this scenario, the fixed ordering solution must evaluate H_i, but in expectation, H_i occurs at position $n/2$ in the ordering $H_1, \ldots, H_n$. Thus,

$$\mathbf{E}[\text{cost(}\textit{Fixed Ordering}\text{)}] \geq \frac{1}{e} \times \frac{n}{2} = \Omega(n) = \Omega(\mu) = \Omega(\mu) \times \mathbf{E}[\text{cost(}\textit{OPT}\text{)}]$$

□

To show the above bound is tight, consider the following greedy fixed ordering, called *Fixed-Greedy*. Sort the filters in increasing order of $c_i/(1-p_i)$ (which is the same as the greedy rule for one conjunctive query). Evaluate the filters in this order, avoiding any redundancy. That is, if all the queries involving some filter F_i have already been resolved to *True* or *False* by previously evaluated filters, do not evaluate F_i.

Lemma 4.2.2. *If there is only one query ($m = 1$), then* Fixed Greedy *returns the optimal solution.*

Proof. If $m = 1$, there is no need to consider an adaptive strategy. Without any loss of generality, assume the query contains all the filters. Consider some non-adaptive ordering $\sigma : F_{i_1}, \ldots, F_{i_n}$ that is different from the one returned by *Fixed Greedy*. In other words, there exist two filters F_{i_k}, F_{i_l} with $k < l$ such that $c_{i_k}/(1-p_{i_k}) > c_{i_l}/(1-p_{i_l})$. The expected cost of this ordering is given by

$$\mathbf{E}[\text{cost}(\sigma)] = \sum_{t=1}^{n} c_{i_t} \times \Pr[\text{Filter } F_{i_t} \text{ is evaluated}] = \sum_{t=1}^{n} c_{i_t} \left(\prod_{r=1}^{t-1} p_{i_r} \right)$$

From σ, construct a new ordering σ' by swapping the positions of F_{i_k} and F_{i_l}. Since $c_{i_l} + p_{i_l} c_{i_k} < c_{i_k} + p_{i_k} c_{i_l}$, we get $\mathbf{E}[\text{cost}(\sigma')] < \mathbf{E}[\text{cost}(\sigma)]$. Thus, any ordering other than the one returned by *Fixed Greedy* cannot be optimal. This completes the proof. □

Theorem 4.2.3. *For multiple conjunctive queries,* Fixed Greedy *is a $O(\mu)$ approximation.*

Proof. Let *OPT* denote the optimal adaptive ordering. We now have

$$\begin{aligned}
\mathbf{E}[\text{cost(}\textit{Fixed Greedy}\text{)}] &\leq \sum_{j=1}^{m} \mathbf{E}[\text{cost to resolve query } Q_j \text{ in } \textit{Fixed Greedy}] \\
&\leq \sum_{j=1}^{m} \mathbf{E}[\text{cost to resolve query } Q_j \text{ in } \textit{OPT}] \\
&\leq \mu \times \sum_{i=1}^{n} \mathbf{E}[\text{cost to evaluate filter } F_i \text{ in } \textit{OPT}] \\
&= \mu \times \mathbf{E}[\text{cost(}\textit{OPT}\text{)}]
\end{aligned}$$

The first inequality holds since multiple queries may contain the same filter, the second inequality follows from Lemma 4.2.2, and the final inequality holds since a filter may appear in at most μ different queries. □

4.3 Adaptive Policies for Conjunctive Query Evaluation

Recall that in the conjunctive query evaluation problem, we have n *filters* $F_1, \ldots, F_n$, where filter F_i has cost of evaluation c_i and evaluates to *True* with probability p_i (also called the selectivity of the filter). There are m *queries* $Q_1, \ldots, Q_m$. Each query Q_j is the Boolean conjunction of a subset R_j of the filters. The goal is to determine a possibly adaptive order of evaluating the filters so that the expected cost spent in evaluating *all* the queries is minimized.

Note that if all $p_i \approx 0$, this problem reduces to that of choosing the cheapest collection of filters to *cover* all the queries, where a filter covers a query if it is present in the query. Therefore, the set cover problem is a special case of the shared query evaluation problem, and we cannot hope to design a $o(\log m)$ approximation algorithm in polynomial time under standard complexity theoretic assumptions.

We will now present a $O(\log m + \log n)$ approximation to this problem. Recall that fixed orderings would be a poor approximation to this more general problem. Therefore our algorithm does not produce a fixed ordering, but is instead adaptive in nature. We will in fact develop an adaptive version of the greedy set cover algorithm!

4.3.1 Covering Graph and Connection to Set Cover

The key to designing the algorithm is to have a notion of *improvement* at every step. Consider the naive greedy algorithm that at each point in time, chooses an unevaluated filter F_i that minimizes c_i/n_i, where:

$$n_i = (1 - p_i) \times \text{ Number of unresolved queries containing } F_i$$

This is the natural extension of the greedy set cover algorithm (where $p_i \approx 0$). However, this algorithm does not lend itself to the set cover type analysis, since there is no natural notion of improvement at every step. In particular, suppose a filter is evaluated and it returns *True*, then it does not help resolve any query that it contains, and it does not appear that any progress has been made.

Instead define the following bipartite *covering graph* G. The vertex sets are denoted $\mathcal{Q}$ and $\mathcal{F}$ respectively. If query Q_j contains filter F_i, then there is an edge between $i \in \mathcal{Q}$ and $j \in \mathcal{F}$. Let E denote the edge set of G. For an edge $e = (i, j)$, this edge is *covered* at some step in the policy if one of the following two conditions hold:

1. F_i is already evaluated (and returned *True* or *False*); or
2. Some F_k, where $k \neq i$ and $(k, j) \in E$, was already evaluated and returned *False*, so that $Q_j =$ *False*.

Note that if some edge (i, j) is uncovered then F_i has not been evaluated, and all evaluations of filters relevant to Q_j returned *True*. This means Q_j is unresolved, so that any decision policy must continue execution. Therefore, the goal of any decision policy is to cover all edges of the covering graph by spending as little expected cost as possible.

4.3.2 Adaptive Greedy Policy

In view of the above definition of covering, the *GREEDY* algorithm is as follows:

Choose that unevaluated filter F_i that minimizes the per-unit cost:

$$s_i = \frac{c_i}{\mathbf{E}[\text{Number of edges covered by } F_i]}$$

The denominator is computed as follows: Consider the covering graph with only the edges that have not been covered in some previous step. In this graph G', with probability $(1 - p_i)$, filter F_i evaluated to *False* and covers all edges (i', j) in G' such that edge (i, j) also exists in G'. With probability p_i, filter F_i evaluates to *True* and covers all edges (i, j) in G'. The expectation is over these two possibilities.

4.3.3 Set Cover Type Analysis

As in the analysis of the greedy set cover algorithm, we amortize the cost of each filter (or set) among the edges in the covering graph. This yields a price for each edge based on which filter covered it, which we upper-bound in terms of OPT, the cost of the optimal adaptive solution. However, unlike the analysis of the classical greedy algorithm, the analysis is trickier because we have an adaptive decision policy. More precisely, when an edge is covered, we already know the outcome of the filter that covers it. However, the greedy algorithm is using a price that is computed *before* the filter is evaluated. In classical set cover, these two quantities coincide, but in the stochastic case, they do not. We therefore need a slightly different charging scheme for the cost of the optimal and greedy solutions.

In the *GREEDY* algorithm, the per-unit cost s_i is computed *before* filter F_i is evaluated. Further note that s_i changes as other filters are evaluated. To fix notation, we therefore consider that point in time when *GREEDY* is about to evaluate F_i and define s_i to be its per-unit cost at this point in time. We charge this quantity to the edges that are *actually* covered by F_i when .it is evaluated.

Formally, suppose F_i when evaluated covers edge e_k. Then define

$$\text{Price}[e_k] = s_i$$

Therefore, for any edge e_k, the price is a random variable whose value is equal to the per-unit cost of the filter that eventually covers it.

Claim 4.3.1. *The expected cost of the greedy algorithm is precisely* $\sum_k \mathbf{E}[\textit{Price}[e_k]]$.

Proof. At the point when F_i is evaluated, we have by definition of s_i that

$$c_i = \mathbf{E}[\text{Number of edges covered by } F_i] \times s_i$$

The contribution to the cost of *GREEDY* from this step is precisely c_i. Since each edge covered is assigned price s_i, the RHS is the increase in total price. The claim follows by linearity of expectation. □

For the purpose of analysis, it is convenient to amortize the cost of the filter evaluation based on whether the optimal solution evaluates it or not. In particular, consider any realization of the filter values. Conditioned on this realization, the greedy algorithm and the optimal solution each evaluate a certain subset of filters. If $F_i \in GREEDY \cap OPT$, then define:

$$OC(F_i) = s_i \times \text{Actual number of edges covered in GREEDY}$$

else define $OC(F_i) = c_i$. We need this new accounting scheme to address the concern raised above – if we are ordering edges based on when they are covered, we already know the outcome of the corresponding filter covering this set.

Claim 4.3.2. *Suppose the optimal solution has evaluated a subset Γ of filters. Suppose $\mathcal{F}$ denotes their outcomes, and conditioned on this, suppose it decides to evaluate filter F_i. Suppose we charge a cost of $\mathbf{E}[OC(F_i)|\mathcal{F}]$ instead of the actual cost for this evaluation. Then, the expected total charge is precisely OPT.*

Proof. For any filter F_i, fix a realization of the values of all other filters. Since the outcome of evaluating F_i is independent of this realization, both OPT and $GREEDY$ deterministically decide whether to evaluate F_i.

Now conditioned on $\mathcal{F}$, consider all scenarios where GREEDY decides to evaluate F. At the point of evaluation,

$$\mathbf{E}[OC(F_i)] = s_i \times \mathbf{E}[\text{Actual number of edges covered in GREEDY}]$$

where the expectation is over the outcome of evaluating F_i. But by the definition of s_i, the above term is exactly c_i. Therefore,

$$\mathbf{E}[OC(F_i)|\mathcal{F} \text{ and GREEDY evaluates } F_i] = c_i$$

But, trivially we have:

$$\mathbf{E}[OC(F_i)|\mathcal{F} \text{ and GREEDY does not evaluate } F_i] = c_i$$

Therefore, $\mathbf{E}[OC(F_i)|\mathcal{F}] = c_i$, completing the proof. □

Number the edges (adaptively) in the order in which they are covered by the *GREEDY* algorithm. Consider the point in the execution of the greedy algorithm where it is covering the k^{th} edge. Suppose the filters evaluated so far are $F_1, F_2, \ldots, F_i$, and note that F_i covers e_k. Condition on the outcome of these filter evaluations.

Claim 4.3.3. *Let N denote the total number of edges in the covering graph. Then*

$$\frac{\mathbf{E}[OPT|F_1, F_2, \ldots, F_i]}{N - k + 1} \geq Price[e_k]$$

Proof. Let $\Psi = \{F_1, F_2, \ldots, F_{i-1}, F_i\}$. Condition on the outcome of Ψ. Suppose we execute OPT after executing F_{i-1}; if OPT evaluates a filter from Ψ, we use the preexisting outcome and corresponding decision branch. To simplify notation, we denote the conditional policy by Cond-OPT.

We have:

$$\mathbf{E}[OPT|F_1, F_2, \ldots, F_i] \geq OC(F_i) \times \Pr[F_i \in \text{Cond-OPT}] + \sum_{F_l \notin \Psi} c_l \times \Pr[F_l \in \text{Cond-OPT}]$$

Conditioned on the outcome of $F_1, F_2, \ldots, F_{i-1}$, any solution is required to cover the remaining $N - k + 1$ edges. In the execution of Cond-OPT, the coverage of any filter is only increased if it is evaluated right after F_{i-1}. Combining these two facts:

$$N - k + 1 \leq |F_i| \times \Pr[F_i \in \text{Cond-OPT}] + \sum_{F_l \notin \Psi} n_l \Pr[F_l \in \text{Cond-OPT}]$$

where n_l is the expected number of edges covered by F_l if it is evaluated immediately after F_{i-1}. Also, $|F_i|$ is the actual number of edges covered by F_i when evaluated by the *GREEDY* algorithm.

From the above two inequalities, it follows that:

$$\frac{\mathbf{E}[OPT|F_1, F_2, \ldots, F_i]}{N - k + 1} \geq \min \left\{ \frac{OC(F_i)}{|F_i|}, \min_{F_l \notin \Psi} \frac{c_l}{n_l} \right\}$$

The first term in the summation is is precisely Price$[e_k]$ by definition of $OC(F_i)$. Since the greedy choice is optimal, $\min_{F_l \notin \Psi} \frac{c_l}{n_l} \geq$ Price$[e_k]$. Combining these two facts, completes the proof of the claim[1]. □

Theorem 4.3.4. *The adaptive greedy policy is a $O(\log N) = O(\log m + \log n)$ approximation.*

Proof. By removing the conditioning in Claim 4.3.3, it is easy to see that $\mathbf{E}[Price(e_k)] \leq \frac{OPT}{N-k+1}$. Since $GREEDY = \sum_{k=1}^{N} \mathbf{E}[Price(e_k)]$, we have $GREEDY \leq OPT \times O(\log N)$. Finally note that the number N of edges in the covering graph is at most $m \times n$. □

[1]Note that we have assumed $\Pr[F_i \in \text{Cond-OPT}] > 0$ in the above proof. When $\Pr[F_i \in \text{Cond-OPT}] = 0$, the proof is only simpler since the first terms in the summation do not exist.

Chapter 5

Markov Decision Processes

In this chapter, we present a general and widely studied framework for stochastic optimization problems, along with an optimal dynamic programming algorithm. The catch of course is that for the problems considered so far, the size of the formulation and hence the running time of the dynamic program will be exponential in the problem parameters. In the next chapter, we will present a subclass of this framework admits a surprising greedy optimal algorithm. This class of problems is termed *multiarmed bandits* and the greedy optimal algorithm is termed the Gittins index. But before going there, we need to present the general framework, which is termed Markov Decision Processes (or MDP).

Any stochastic optimization problem can be written as an MDP. In an MDP, we have a system that can be one of many possible states. Denote the set of states by $\mathcal{S}$. At each state, one of several actions are available. We denote the set of actions by $\mathcal{A}$. Suppose the system is in state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ is taken. Then the system yields reward $R(s, a)$, and transitions to state $s' \in \mathcal{S}$ with probability $p_a(s, s')$. We assume $\sum_{s' \in \mathcal{S}} p_a(s, s') = 1$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, so that $p_a(\cdot, \cdot)$ defines a *transition matrix* between states.

Assume time is discrete. At time $t = 1$, the system is in some start state s_1. There is a finite time horizon T. A *decision policy* $\mathcal{P}$ specifies an action $a(s, t)$ for every possible state $s \in \mathcal{S}$ and time $t \in \{1, 2, \ldots, T\}$ the system could be in. When such a policy is executed, it executes action $a(s_1, 1)$ at time 1, which makes the system transition to a random state s_2 according to the transition matrix $p_{a(s_1,1)}(s_1, \cdot)$. The system applies action $a(s_2, 2)$ to this state, and so on. Let $s_1, s_2, \ldots, s_T$ denote the random set of states encountered by the system. Then the *value* of the policy given start state s_1 is given by:

$$V(\mathcal{P}, s_1) = \mathbf{E}\left[\sum_{t=1}^{T} R(s_t, a(s_t, t))\right]$$

where the expectation is over $\{s_1, s_2, \ldots, s_T\}$ encountered by the system. The goal of an algorithm designer is to find the policy $\mathcal{P}^*$ that maximizes $V(\mathcal{P}, s_1)$.

The reason the above framework is termed *Markov* is that the state transition matrix

$p_a(s, \cdot)$ only depends on the current state and action taken, and not on how this state was reached. In other words, the future evolution of the system is independent of the past conditioned on knowing the current state, and this implies the optimal action $a(s,t)$ only depends on the current state s and the time left t, and not on how we got to this state.

Myopic Policies. One simple, yet sub-optimal policy for an MDP is what is called the *myopic* policy: Suppose the system is in state s at time t. Then it takes that action $a \in \mathcal{A}$ that gives it the maximum reward at this step. In other words,

$$a(s,t) = \text{argmax}_{a \in \mathcal{A}} R(s,a)$$

The reason such policies are sub-optimal is that they do not take into account the state transition matrix – it could be that an action that yields less immediate reward has a greater chance of transitioning the system into a state where the myopic reward is larger.

As a simple example, consider the *Quiz Problem*, which is a maximization version of the single conjunctive query evaluation problem. There are n questions, and one quiz taker. For question i, the reward for answering it correctly is r_i and the probability of answering it correctly is p_i. The quiz taker can adaptively choose the order of answering questions. The first time she answers incorrectly, the process stops. The goal of the quiz taker is to choose an adaptive ordering that maximizes her expected reward.

Example 5.1. *Suppose there are only $n = 2$ questions, q_1 and q_2. Suppose $r_1 = 1$ and $p_1 = 1$, while $r_2 = 3$ and $p_2 = 1/2$. There are two actions available at the outset – choose q_1 or choose q_2. The myopic policy chooses that action whose expected reward is larger. Answering q_1 has expected reward 1, while answering q_2 has expected reward 1.5. The policy therefore chooses q_2. With probability $1/2$, it stops there, and with probability $1/2$, it answers correctly and then chooses q_1. The expected reward of the policy is $1.5 + 1/2 \times 1 = 2$.*

The optimal policy answers q_1 first, and with probability 1, proceeds to answering q_2. This has expected reward $1 + 1.5 = 2.5$.

The same proof as Lemma 4.2.2 shows that the optimal policy is a fixed ordering, and considers questions in decreasing order of $\frac{r_i}{1-p_i}$. This is not the same as the myopic policy that considers questions in decreasing order of $r_i p_i$.

Optimal Policies and Dynamic Programming. Since the state evolution of the system is Markov, the optimal policy is a simple application of dynamic programming. Let us overload notation and denote by $V(s,t)$ as the expected value of an optimal policy for the problem where the system starts in state s and has a time horizon of t. We are ultimately interested in computing $V(s_1, T)$.

Consider the optimal policy corresponding to $V(s,1)$. This policy starts in state s, and can execute for one time step. Such a policy will make the myopically optimal decision, so that:

$$V(s,1) = \max_{a \in \mathcal{A}} R(s,a)$$

For $t > 1$, the optimal policy corresponding to $V(s,t)$ will choose that action $a \in \mathcal{A}$ at the current time step that maximizes the sum of the current reward and the future reward assuming optimal actions are made in the future. Therefore,

$$V(s,t) = \max_{a \in \mathcal{A}} \left(R(s,a) + \sum_{s' \in \mathcal{S}} p_a(s,s') V(s', t-1) \right)$$

It is easy to check by induction over t that this is indeed the reward of the optimal policy. Therefore, the optimal policy can be computed in time $O(T|\mathcal{A}||\mathcal{S}|^2)$, which is polynomial in the size of the state space and number of actions.

5.1 Modeling Problems as MDPs

Any stochastic optimization problem where inputs are specified as distributions is a special case of an MDP. If the optimal policy can be computed in polynomial time, why have we resorted to approximation? The simple reason is that modeling these problems as MDPs requires state space that is exponential in the problem parameters. Let us present some examples to make this clear.

Consider the shared conjunctive query evaluation problem. Here, $T = \infty$. The problem is one of minimizing cost instead of maximizing reward. The state of the system is captured by $\langle S, O \rangle$, where S is the set of filters evaluated so far, and O is the set of Boolean outcomes observed for these evaluations. At each state $\langle S, O \rangle$, there are $n - |S|$ actions available, where n is the total number of filters; each action corresponds to evaluating a filter $i \notin S$ and has cost c_i. If this action is taken, the state transitions to one of two states: $\langle S', O_t \rangle$ and $\langle S', O_f \rangle$, where $S' = S \cup \{i\}$, and $O_t = O \cup \{i = \text{ True }\}$, and $O_f = O \cup \{i = \text{ False }\}$. The former transition happens with probability p_i and the latter with probability $1 - p_i$. If the current state $\langle S, O \rangle$ resolves all the queries, we assume there are no actions available for this state, and the system stops there.

The complexity arises because the number of possible states is exponential in n, the number of filters. We note that this observation is true even when the number of queries is 1, but in this case, the optimal policy is a fixed ordering and can be computed in polynomial time. Therefore, just because a problem has state space of exponential size does not mean computing its optimal policy will necessarily take exponential time. We will see another example in the next chapter, when we discuss multi-armed bandits.

5.2 Discounted Reward Version

The above dynamic programming formulation is termed the *finite horizon* version of the MDP. This is because there is a finite time horizon T over which the policy executes. The thorny issue is that the current time t becomes part of the state of the system, and the optimal action not only depends on the current state, but also the number of remaining time steps. In fact, any policy becomes more myopic as the number of future time steps (or "time-to-go") reduces.

We get around this issue by making the time horizon infinite and introduce a discount factor. For a given $\gamma \in (0, 1)$ and start state s_0, the goal is to find a policy that maximizes

$$\text{Discounted Reward } = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a(s_t, t))\right]$$

An interpretation of discounting is that at any step, the policy is forced to stop w.p. $1-\gamma$ independent of the plays and outcomes so far. This is a smoother and more memoryless stopping condition than insisting that the policy is forced to stop after exactly T steps.

Since the time horizon is infinite, the optimal action only depends on the current state, and is independent of the time-to-go. Therefore, we can define $V(s)$ as the optimal infinite horizon discounted reward when the start state of the system is s. We have the following recurrence to compute the optimal policy, which is obtained by simply observing that the optimal policy maximizes the sum of the current reward and the discounted reward from the optimal policy of whatever next state it reaches:

$$V(s) = \max_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{\tilde{s} \in \mathcal{S}} p_a(s, \tilde{s}) V(\tilde{s}) \right\}$$

This recurrence relation is termed Bellman's equations. These do not lend themselves to backward induction since there is no time horizon to do induction over.

5.2.1 Value Iteration

The solution to the above recurrence can be computed by a method termed *value iteration*. Define the Bellman operator T that operates on the value of each state as follows:

$$TW(s) = \max_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{\tilde{s} \in \mathcal{S}} p_a(s, \tilde{s}) W(\tilde{s}) \right\}$$

Given a vector $\vec{W}$ of dimension equal to the number of states $|\mathcal{S}|$, the Bellman operator T outputs another vector $T\vec{W}$ that also has dimension equal to $|\mathcal{S}|$.

Note that the optimal solution $\vec{V}$ is the fixed point of the Bellman operator T, that is the vector $\vec{V}$ for which $T\vec{V} = \vec{V}$. We show below that the fixed point indeed exists and is unique.

Towards this end, we define a procedure termed *value iteration* in Figure 5.1.

Theorem 5.2.1. *Value iteration is well-defined. In other words, the Bellman operator T has a unique fixed point and value iteration finds this point.*

Proof. Define $d(W, W') = \max_s ||W(s) - W'(s))||_\infty$. This defines a metric on the space of vectors $\vec{W}$. We will first show that $d(T\vec{W}, T\vec{W'}) \le \gamma d(\vec{W}, \vec{W'})$. Fix some state s, and let the optimal action for $TW(s)$ be action $a^* \in \mathcal{A}$. In other words, let:

$$TW(s) = R(s, a^*) + \gamma \sum_{\tilde{s}} p_{a^*}(s, \tilde{s}) W(\tilde{s})$$

```
1. $\epsilon > 0$ is some small constant.
2. $\vec{W} \leftarrow \vec{0}$.
3. While $|T\vec{W} - \vec{W}| > \epsilon$ do:
    (a) $\vec{W} \leftarrow T\vec{W}$.
4. End While
5. Output $\vec{W}$.
```

Figure 5.1: The Value Iteration Algorithm.

By definition of the operator T, we also have:

$$TW'(s) \geq R(s, a^*) + \gamma \sum_{\tilde{s}} p_{a^*}(s, \tilde{s}) W'(\tilde{s})$$

Combining the two inequalities, we have:

$$TW(s) - TW'(s) \leq \gamma \max_{\tilde{s}} |W(\tilde{s}) - W'(\tilde{s})|$$

Similarly, by considering the optimal action for $W'(s)$, it can be shown that $TW'(s) - TW(s) \leq \gamma \max_{\tilde{s}} |W'(\tilde{s}) - W(\tilde{s})|$, showing that $d(T\vec{W}, T\vec{W'}) \leq \gamma d(\vec{W}, \vec{W'})$.

Suppose there are two solutions $\vec{V_1}$ and $\vec{V_2}$ to Bellman's equations. Then, $d(T\vec{V_1}, T\vec{V_2}) = d(\vec{V_1}, \vec{V_2})$ since $T\vec{V_1} = \vec{V_1}$ and $T\vec{V_2} = \vec{V_2}$. On the other hand, $d(T\vec{V_1}, T\vec{V_2}) \leq \gamma d(\vec{V_1}, \vec{V_2})$, showing that $d(\vec{V_1}, \vec{V_2}) = 0$, which is true only if $\vec{V_1} = \vec{V_2}$.

To show the existence of a fixed point, we use value iteration. Let $\vec{V}_{k+1} = T\vec{V_k}$. Then, $d(\vec{V}_{k+1}, \vec{V_k}) \leq \gamma^k d(\vec{V_1}, \vec{V_0})$. This shows that $\{\vec{V_k}\}$ is a Cauchy sequence and converges to $\vec{V}$ such that $T\vec{V} = \vec{V}$. □

Chapter 6

Multi-armed Bandits

We now present a celebrated decision problem with exponentially large state space for which there is a non-trivial exact policy that can be computed by a greedy algorithm. This policy, termed the *Gittins Index* is arguably the most surprising and celebrated result in all of decision theory.

The multi-armed bandit problem, originally described by Robbins in 1952, is a machine learning problem based on an analogy with a slot machine with more n levers. When a lever i is pulled, it provides a numerical reward drawn $i.i.d.$ from a reward distribution D_i associated with that lever. The objective of the gambler is to maximize the sum of discounted rewards collected through iterative pulls.

The classical assumption in this problem is that the gambler has no initial knowledge about the payoffs from levers. The tradeoff that the gambler faces at each trial is between *exploitation* of the lever that has the highest expected payoff given current knowledge, and *exploration*, to obtain more information about the payoff distributions of the other levers. This is known as the exploitation versus exploration tradeoff in reinforcement learning.

Keyword Allocations. Consider the allocation of keywords to advertisements (and hence to advertisers) by search engines. Let us assume that for each keyword search, the search engine is allowed to show one advertisement, say a_i to the user. If the user clicks the ad, then the advertiser i pays the search engine an amount v_i which is initially agreed upon. For a given keyword, there might be n advertisers willing to pay the search engine $v_1, v_2 \dots v_n$ based on click throughs. But a user might not click each of the ads $a_1, a_2, \dots a_n$ with equal probability. So let the probability of the user clicking on ad a_i be p_i. Thus the expected revenue for showing ad a_i is $p_i v_i$. Typically, multiple users will search for the same keyword and so the search engine gets multiple opportunities to display the ads for that keyword. Suppose there are T such opportunities (or time steps), then the search engine would like to maximize its expected revenue,

$$\max \mathbf{E}\left[\sum_{t=1}^{T} R_t\right]$$

where R_t is the revenue earned at time step t. We want to design a policy that will achieve this maximum revenue in expectation. If all the p_i are known, then there is a simple policy that will obtain the max revenue: always show a_{i^*} where

$$i^* = \arg\max_i p_i v_i$$

But if the p_i are not known, then we want to explore for a while to learn the p_i and exploit what we think i^* will be. To do this we need an estimate of "goodness" of an arm based on the outcomes of playing that arm so far. This leads us to the notion of priors and posteriors.

6.1 A Bayesian Example

In simplest terms, a prior is a distribution over possible values of the parameter p_i. (Though this definition seems specific to Bernoulli distributions, we generalize it later.) We will assume that this distribution is specified as input. When the arm is played, the prior distribution is updated according to Bayes' rule into a posterior distribution. This is illustrated by the following example.

Suppose there is a bandit with two arms, A and B. At any point in time, the gambler can pull one of the arms resulting in reward either \$1 or \$0. At some point in time the gambler has played arm A 200 times and observed a reward of \$1 in 100 trials. He has played arm B 3 times and has observed a reward of \$1 in one of the trials. What is an estimate of the expected rewards of each arm in the next play? A reasonable estimate is $\$\frac{1}{2}$ for arm A and $\$\frac{1}{3}$ for arm B. Of course, the gambler is more confident about his estimate for arm A than for arm B. Each time he plays an arm, he needs to update our estimate on the expected reward he will see for that arm. This notion of updating the belief about the expected reward for each arm after playing it is formalized using priors and posteriors.

Beta Priors. In the above example, suppose the gambler had not played arm A or B at all. Then how would we encode his belief? We can suppose that the underlying reward distribution of arm A is Bernoulli$(1, p_A)$ and that of arm B is Bernoulli$(1, p_B)$. Since neither arm has been played, the gambler has no information about p_A or p_B. He can therefore assume that p_A is uniformly distributed over $[0, 1]$ and the same holds for p_B. We call this the *prior* distribution of the arm.

Suppose now that the gambler plays arm A, and observes $\alpha - 1$ rewards of value 1, and $\beta - 1$ rewards of value 0 in $\alpha + \beta - 2$ plays, where each play is an $i.i.d.$ draw from the underlying Bernoulli$(1, p_A)$ distribution. Given this, what should the gambler's belief about p_A be? This can be obtained by Bayes' rule as follows: Let $\eta(\alpha, \beta)$ denote the event of observing $\alpha - 1$ rewards of value 1, and $\beta - 1$ rewards of value 0 in $\alpha + \beta - 2$ plays.

$$\begin{aligned}
\Pr\left[p_A = p|\eta(\alpha,\beta) \text{ and Prior}\right] &= \frac{\Pr\left[\eta(\alpha,\beta)|p_A = p\right] \times \Pr[p_A = p| \text{ Prior}]}{\Pr\left[\eta(\alpha,\beta)| \text{ Prior}\right]} \\
&= \frac{\binom{\alpha-1}{\alpha+\beta-2} p^{\alpha-1}(1-p)^{\beta-1}}{\int_0^1 \binom{\alpha-1}{\alpha+\beta-2} q^{\alpha-1}(1-q)^{\beta-1} dq} \\
&= \frac{p^{\alpha-1}(1-p)^{\beta-1}}{\int_0^1 q^{\alpha-1}(1-q)^{\beta-1} dq}
\end{aligned}$$

The distribution with density function given by

$$f(p;\alpha,\beta) = \frac{p^{\alpha-1}(1-p)^{\beta-1}}{\int_0^1 q^{\alpha-1}(1-q)^{\beta-1} dq}$$

is termed Beta(α,β). It is the posterior density of p if the prior is Uniform$(0,1)$ and we observe $\alpha - 1$ rewards of value 1, and $\beta - 1$ rewards of value 0 in $\alpha + \beta - 2$ plays of the arm.

Suppose the posterior density of the arm is Beta(α,β) and suppose this arm is played. What is the expected reward observed? This will be $\mathbf{E}[\text{Beta}(\alpha,\beta)]$, which is $\frac{\alpha}{\alpha+\beta}$. To see this, we first write a closed form for $\int_0^1 q^{\alpha-1}(1-q)^{\beta-1} dq$ as

$$\int_0^1 q^{\alpha-1}(1-q)^{\beta-1} dq = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

where $\Gamma(n) = (n-1)!$. Now, we can write

$$f(p;\alpha,\beta) = \frac{p^{\alpha-1}(1-p)^{\beta-1}}{\int_0^1 q^{\alpha-1}(1-q)^{\beta-1} dq} = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}$$

Therefore:

$$\begin{aligned}
\mathbf{E}[\text{Beta}(\alpha,\beta)] &= \int_0^1 p \times f(p;\alpha,\beta) dp \\
&= \int_0^1 p \times \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1} dp \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \times \frac{\Gamma(\alpha+1)\Gamma(\beta)}{\Gamma(\alpha+\beta+1)} \\
&= \frac{\alpha}{\alpha+\beta}
\end{aligned}$$

Therefore if we saw $\alpha - 1$ success in $\alpha + \beta - 2$ trails, the expected reward of the next play is $\frac{\alpha}{\alpha+\beta}$. This is because in the base case, we have seen 0 1's in 0 trails, and we expect a reward of $\frac{1}{2}$ in the next play. This is the expectation of Uniform$[0,1]$, which is also the same as Beta$(1,1)$. Thus the beta distribution serves as a good prior under the Bayesian assumption.

What is the state space of a single arm? We can parametrize a state as the tuple (α, β), which captures the observed rewards for the arm. The start state of the arm is $(1, 1)$. In each state, there are two actions available: Play or not play. If the arm is not played, the state does not change and there is no reward. Conditioned on being in state (α, β) and playing the arm, the expected reward is $q = \frac{\alpha}{\alpha+\beta}$. The arm transitions with probability q to the state $(\alpha + 1, \beta)$ and with probability $1 - q$ to state $(\alpha, \beta + 1)$.

Though the above state space appears succinct, note that this is the state space of a *single* arm. At any point in time, each arm is in a certain state, so that the state space of an n armed bandit is the Cartesian product of the single-arm state spaces, which has size exponential in n.

6.1.1 Parametrized Distributions

In the case of Beta priors, the underlying distribution is Bernoulli$(1, p)$, and the unknown parameter of the distribution is $p \in [0, 1]$. The prior Beta(α, β) specifies a distribution of the parameter p. We can generalize this to the class of single parameter distributions (and beyond): Let $f_\theta(x)$ denote the density function of the underlying distribution, where θ is the unknown parameter. A prior would be a distribution over the parameter θ that gets refined as samples are drawn from the distribution.

The key property that makes a prior *good* is that the posterior according to Bayes' rule has the same general form as the prior. This was true for the Beta priors, where the posterior is also a Beta density. We now generalize this notion.

A *single parameter exponential family* is specified by the density function:

$$f_\theta(x) = \frac{h(x)e^{\theta x}}{e^{A(\theta)}} \qquad \text{where} \qquad e^{A(\theta)} = \int_x h(x)e^{\theta x}dx$$

Different density functions differ in the choice of $h(x)$. For suitable choices, we obtain, Bernoulli, Gaussian (with unknown mean), Poisson, Exponential, and Gamma distributions. For instance, if $h(x) = 1$ if $x = 0$ or $x = 1$ and $h(x) = 0$ otherwise, we obtain the Bernoulli$(1, p)$ distribution, where $\theta = \log \frac{p}{1-p}$.

Conjugate Prior. For any single parameter exponential family distribution, we can define the *conjugate prior* on the parameter θ as $r_{a,b}(\theta) \propto e^{a\theta - bA(\theta)}$, where $b \geq 0$. Here, a, b are the parameters of the prior, and get refined with samples from the underlying density. For instance, it is easy to check that for Bernoulli distributions, $r_{a,b}(\theta)$ is the density Beta$(a + 1, b - a + 1)$.

Now, given a prior $r_{a,b}(\theta)$ over the parameter θ, suppose the arm is played and the observed reward is x, then let us compute the posterior of θ. This is given by Bayes' rule:

$$\begin{aligned}
\Pr[\theta|x] &= \frac{\Pr[x|\theta]\Pr[\theta]}{\Pr[x]} = \frac{f_\theta(x)r_{a,b}(\theta)}{\int_\theta r_{a,b}(\theta)f_\theta(x)d\theta} \\
&\propto h(x)e^{\theta x - A(\theta)}e^{a\theta - bA(\theta)} \\
&\propto e^{(a+x)\theta - (b+1)A(\theta)} = r_{a+x,b+1}(\theta)
\end{aligned}$$

Therefore, the parameter a behaves as the sum of the rewards, x, of the samples seen so far, and the parameter b behaves as the number of observations made so far, so that a/b is the empirical average reward per play. The posterior has the same form as the prior, hence the name *conjugate prior*.

Martingale Property: Given a prior $r_{a,b}(\theta)$, the expected value of the reward when the arm is played is:

$$\mu(a,b) = \int_\theta r_{a,b}(\theta) \int_x x f_\theta(x) dx d\theta$$

Suppose the arm is played and the prior is updated; and subsequently the arm is played again. What is the expected value of the reward of the second play, where the expectation is over both plays? It is easy to check using Bayes' rule that this expected value is the same as that of the first play. This intuitive fact is termed the *Martingale property* of the prior updates. We note that the martingale property is only a special case of the more general general multi-armed bandit problem we present later.

6.1.2 Applications of Bayesian Bandits

The multi-armed bandit problem as formulated in the previous section has several applications, and has been widely studied for several decades. We list some applications below.

Design of clinical trials. Suppose we have n alternate treatments for some disease and want to test these treatments on T subjects who show up for the trials in an online manner. We need to design a strategy that will be as beneficial to the subjects as possible. Allocating $\frac{T}{n}$ subjects to each treatment upfront is not optimal since some of the alternate treatments might not be effective at all while some might be very effective. It is better to adaptively decide which patient to allocate to which treatment This problem can be modeled as a multi armed bandit: each treatment is a bandit lever and we are given T chances to play the bandit. The outcome of a lever depends on the effectiveness of the corresponding treatment, where we have implicitly assumed that for treatment i, the effectiveness on a patient is a random variable drawn from the same underlying effectiveness distribution D_i.

Pricing identical copies of a good. Suppose we are selling some software, creating copies of which are assumed to be of negligible cost. But as a seller, we do not know how to price the software. We can choose between n possible price points. There are T customers who arrive in an online manner and request the software. If we offer a price greater than what they are willing to pay, they will not buy it and we get no reward for that offer. If they do agree to buy the software, we get a reward equal to the price offered. Suppose the valuations of the buyers are drawn $i.i.d.$ from distribution X, then the payoff if the price is set to q is Bernoulli$(q, \Pr[X \geq q])$. This is a multi armed bandit problem where the n levers .are the n prices.

DNS lookup. Suppose we have n different DNS servers to choose from every time we need to lookup an internet address. Some of these servers might be fast, while others might be slow depending on the volume of requests, capacity, distance to the server, etc. We want to minimize the lookup time we suffer over a period of T requests. This is a multi armed bandit where the arms are different servers, and the cost is the latency of access.

6.2 A General Problem Statement

The celebrated multi-armed bandit problem is a fairly general model for weakly coupled decision systems. There are n arms. Arm i has state space $\mathcal{S}_i$. If arm i in state $s \in \mathcal{S}_i$ is *played*, it yields reward r_{is} and its state transitions to $t \in \mathcal{S}_i$ with probability $p_i(s,t)$. We assume that for all $s \in \mathcal{S}_i$, $\sum_{t \in \mathcal{S}_i} p_i(s,t) = 1$, and these transition probabilities are specified as input. The states spaces of different arms are independent, so that the transition probabilities $p_i(s,\cdot)$ only depend on the state s of arm i, and not on the states of other arms.

If an arm is not played, its state does not change. This feature of the problem is termed the *bandit property*. Any decision policy is restricted to playing at most one arm at any time step. There is a discount factor γ, and the goal is to solve the infinite horizon discounted reward problem.

We wish to design policies whose running time depends on $\sum_i |\mathcal{S}_i|$. Notice that applying dynamic programming directly is computationally infeasible. This is because the state space of the system is the Cartesian product of $\mathcal{S}_i$. So why is this problem simpler than a general MDP with exponentially large state space? The main reasons have already been said – the state evolution of an arm does not depend on the other arms (weak coupling), and the option of only playing one arm per step (bandit property).

6.2.1 Connecting with Priors and Posteriors

Consider the case described before where each arm has an underlying Bernoulli reward distribution. Suppose arm i has distribution Bernoulli$(1, p_i)$. If the current prior on the parameter p_i is Beta(α_i, β_i), then the *state* of the arm is parametrized by (α_i, β_i). If the arm is played in this state, the expected reward is $r_i = \frac{\alpha_i}{\alpha_i+\beta_i}$. With probability r_i, the reward observed is 1, and the new state corresponds to the new posterior, $(\alpha_i + 1, \beta_i)$. With the remaining probability, the new state is corresponds to observing reward 0, and is hence $(\alpha_i, \beta_i + 1)$. Given initial priors on each arm, the goal is to find a policy for playing the arm that maximizes the expected discounted reward.

Note that the state space in the general bandit problem *need not* arise from priors and posteriors; each arm can have an arbitrary state space $\mathcal{S}_i$ with an arbitrary transition matrix. All we assume is that this space can be specified explicitly as input, and the running time of computing and executing the policy should be polynomial in the input size.

6.3 The Gittins Index Theorem

An *index policy* works as follows: It associates a number $\pi(s)$ for every state $s \in \mathcal{S}_i$ of every arm i. For state $s \in \mathcal{S}_i$, the index computation *only* depends on $\mathcal{S}_i$ and not on the other arms. In other words, the computation of indices relevant to any arm can be done in isolation disregarding the other arms entirely. The policy execution is greedy: At time step t, let s_{it} denote the state of arm i. Then the policy plays the following arm:

$$\text{Arm played at time } t = \text{argmax}_i \pi(s_{it})$$

Though such a policy appears myopic, it is actually not. The index π is constructed taking future state evolution into account; however, it only takes the states of the corresponding arm into account. This makes the policy non-myopic, yet computable in polynomial time.

We now describe the computation of the Gittins index for states in $\mathcal{S}_i$. For this purpose, we define a *penalty* version of the single-arm problem. In this new problem, the state space is $\mathcal{S}_i$. There are only 2 actions available in each state: *play* or *stop*. There is a penalty value λ. When the arm is played in state $s \in \mathcal{S}_i$, the net payoff is $r_{is} - \lambda$, and as before, its state transitions to $t \in \mathcal{S}_i$ with probability $p_i(s,t)$. When the action is *stop*, the payoff is 0, and the policy stops execution. The goal is to find a policy that maximizes the infinite horizon net payoff.

Let $V_i(s, \lambda)$ denote the optimal payoff of a policy whose start state is $s \in \mathcal{S}_i$, when the penalty is λ, and let $\mathcal{P}_i(s, \lambda)$ denote the corresponding policy. Then we have the recurrence relation:

$$V_i(s, \lambda) = \max \left\{ 0, r_{is} - \lambda + \gamma \sum_{t \in \mathcal{S}_i} p_i(s,t) V_i(t, \lambda) \right\}$$

As λ increases. the above quantity is clearly non-increasing. Therefore, there exists a value λ_i^* defined as:

$$\lambda_i^*(s) = \max\{\lambda | V_i(s, \lambda) \geq 0\}$$

In other words, this is the maximum penalty for which the policy $\mathcal{P}_i(s, \lambda)$ is indifferent between playing and stopping, since either action fetches zero discounted reward.

Definition 3. *The Gittins index of arm i in state $s \in \mathcal{S}_i$ is the quantity $\lambda_i^*(s)$.*

This completes the definition of the Gittins index policy. To execute the policy, we pre-compute this index for each $s \in \mathcal{S}_i$, for all arms i. At any time step t, the policy plays that arm whose current state has the largest Gittins index.

6.3.1 Structure of the Gittins Index

We first present a basic structural property of the Gittins index that we will use subsequently in our analysis. The property is simply that the policy $\mathcal{P}_i(s, \lambda_i^*(s))$ corresponding to the Gittins index at state s will play the arm in all states with higher Gittins index and stop when it encounters a state with lower Gittins index than that of state s.

Claim 6.3.1. *For $s \in \mathcal{S}_i$, let $\eta_i(s) = \{s' \in \mathcal{S}_i | \lambda_i^*(s') \geq \lambda_i^*(s)\}$. The policy $\mathcal{P}_i(s, \lambda_i^*(s))$ plays arm i whenever it is in a state $s' \in \eta_i(s)$, and stops playing when it encounters a state $s' \notin \eta_i(s)$.*

Proof. Consider the policy $\mathcal{P}_i(s, \lambda)$ for $\lambda = \lambda_i^*(s)$. On execution, if it encounters a state s', it executes the policy $\mathcal{P}_i(s', \lambda)$. If $s' \in \eta_i(s)$, the Gittins index of state s' is at least λ. But this means $V_i(s', \lambda) \geq 0$, so that the policy's action is to play in this state. Similarly, if $s' \notin \eta_i(s)$, we have $V_i(s', \lambda) = 0$, so that the optimal action is to not play. □

Corollary 6.3.2. *Suppose s_i^* is the state in $\mathcal{S}_i$ with the highest Gittins index. Then, $\lambda_i^*(s_i^*) = r_{is_i^*}$. Let $\tilde{s}$ be the state in $\mathcal{S}_i$ with the largest reward $r_{i\tilde{s}}$. Then $\lambda_i^*(\tilde{s}) = r_{i\tilde{s}}$.*

To see why this is true, observe that the corresponding policy only plays the arm in this state, and stops if it exits this state. But the penalty at which such a policy sees no payoff must be the same as the reward of this state.

6.3.2 Iterative Algorithm

Using the above characterization, we can rewrite the Gittins index. This needs some additional notation. Consider any single arm policy with start state s, and let $\mathcal{C}$ be a set of states. Let $T(s, \mathcal{C})$ denote the first time after exiting state s that the policy encounters a state not in $\mathcal{C}$. Note that $T(s, \mathcal{C})$ is a random variable. Let $R(s, \mathcal{C})$ denote the discounted reward collected in this period, and let $\Gamma(s, \mathcal{C}) = \sum_{t=0}^{T(s,\mathcal{C})-1} \gamma^t$.

Lemma 6.3.3. *For $s \in \mathcal{S}_i$, recall that $\eta_i(s) = \{s' \in \mathcal{S}_i | \lambda_i^*(s') \geq \lambda_i^*(s)\}$. We have:*

$$\lambda_i^*(s) = \frac{\mathbf{E}[R(s, \eta_i(s))]}{\mathbf{E}[\Gamma(s, \eta_i(s))]}$$

Proof. By Claim 6.3.1, the policy $\mathcal{P}_i(s, \lambda_i^*(s))$ plays the arm in all states in $\eta_i(s)$, and stops when it encounters any other state. By definition of $\lambda_i^*(s)$, the discounted reward of this policy is the same as the discounted penalty. Therefore,

$$\mathbf{E}[R(s, \eta_i(s)] = \lambda_i^*(s) \times \sum_{t=0}^{T(s,\mathcal{C})-1} \gamma^t$$

The proof follows. □

Using the above characterization, we derive an iterative procedure for computing the Gittins index, which will be more amenable to an inductive analysis. We present the procedure in Figure 6.1. Here, $\mathcal{C}$ is the set of states for which the Gittins index has already been computed. The correctness of this procedure now follows from Lemma 6.3.3, and is left as an exercise.

6.3.3 Inductive Analysis

We now show that the Gittins index policy is indeed optimal. The proof makes clever use of an exchange argument. We will simply derive the algorithm in Figure 6.1 from first principles, hence showing it is optimal.

```
• Let $s^* \leftarrow \{\text{argmax}_{s \in \mathcal{S}_i} r_{is}\}$.
• Set $\lambda_i^*(s^*) \leftarrow r_{is^*}$ and $\mathcal{C} \leftarrow \{s^*\}$.
• While $|\mathcal{C}| < |\mathcal{S}_i|$ do:
    1. For all $s \in \mathcal{S} \setminus \mathcal{C}$, define $q(s, \mathcal{C}) = \frac{\mathbf{E}[R(s,\mathcal{C})]}{\mathbf{E}[\Gamma(s,\mathcal{C})]}$.
    2. $s^* \leftarrow \text{argmax}_{s \in \mathcal{S}_i \setminus \mathcal{C}} q(s, \mathcal{C})$.
    3. $\mathcal{C} \leftarrow \mathcal{C} \cup \{s^*\}$
    4. $\lambda_i^*(s^*) \leftarrow q(\mathcal{C}, s^*)$
• End While
```

Figure 6.1: Gittins Index Computation for state space $\mathcal{S}_i$.

Step 1. We first generalize the bandit problem a bit. Assume that if arm i is played in state $s \in \mathcal{S}_i$, there is a random reward $R(s)$ obtained; the arm *locks up* for random time $T(s)$, and subsequently transitions to a random state $Y(s)$. In the lock up period, no other arm can be played. These random variables can be correlated, and the joint distribution of $\langle R(s), T(s), Y(s)\rangle$ is known. Note that this new problem is only a minor modification, and the Gittins index and its structure follow easily from the previous discussion. The reason for introducing the lock up period will become clear shortly.

Step 2. A policy for the bandit is a function mapping the joint state space $\mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_n$ to the arm in $\{1, 2, \ldots, n\}$ that is played. For any given policy, define the following random variables: t_i is the time at which the i^{th} play starts and R_i is the reward of that play. Recall that γ is the discount factor. The problem is then finding the optimal policy that maximizes the expected discounted reward:

$$\mathbf{E}\left[\sum_{i=0}^{\infty} \gamma^{t_i} R_i\right]$$

We now show that the random variables corresponding to reward are superfluous and can be eliminated, so that the only random variables are the duration of play and the next state. Let the arm played in the i^{th} play be in state s_i. Let Φ_i be the state at time $t_i - 1$. Observe that s_i and t_i are based on the outcomes of the first $i-1$ plays and hence depend on Φ_i. Conditioned on the current state, the expected discounted reward of the i^{th} play is given by

$$\gamma^{t_i}\mathbf{E}[R(s_i)|s_i]$$

But instead of giving this reward upfront, one can think of spreading it over the duration of the locked up period, *i.e.*, reward is received at a constant rate $r(s_i)$ for $T(s_i)$ steps. Thus,

$$r(s_i) = \frac{\mathbf{E}[R(s_i)]}{\mathbf{E}\left[\sum_{t=0}^{T(s_i)-1} \gamma^t\right]}$$

Note that $r(s_i)$ is a deterministic value that depends on the state s_i, and not a random variable. So we are pretending that for the random period $T(s_i)$, we receive *deterministic* reward $r(s_i)$ per time step.

Under this scheme, the expected discounted reward of the i^{th} play conditioned on Φ_i is given by

$$\mathbf{E}\left[\sum_{t=t_i}^{t_i+T(s_i)-1} \gamma^t r(s_i)|\Phi_i\right] = \gamma^{t_i} r(s_i)\mathbf{E}\left[\sum_{t=0}^{T(s_i)-1}\gamma^t|s_i\right] = \gamma^{t_i}\mathbf{E}[R(s_i)|s_i]$$

Thus both these reward accounting schemes work out to the same final reward, and we can use linearity of expectation to show that this holds for the entire policy, not just for one play. We shall be using this later in the proof.

Step 3. We now show that if some arm is in the state with maximum per-step reward, playing it is optimal. This lemma is the crux of the proof; the rest is induction on this statement. Let $\Psi = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdot \times \mathcal{S}_n$ denote the joint state space of the n arms.

Lemma 6.3.4. *Let $s^* = \arg\max_{s\in\Psi} r(s)$ and let k be the arm such that $s^* \in \mathcal{S}_k$. Then there exists an optimal policy that, whenever arm k is in state s^*, plays k regardless of the states of the other arms.*

Proof. Consider an optimal policy OPT. Since the time horizon is infinite, we can assume without loss of generality that the current time step is time 0, and arm k is in state s^*. For the sake of contradiction, assume OPT does not play arm k, but plays some other arm. Let τ be a random variable equal to the first time OPT plays arm k. If the arm is never played, $\tau = \infty$. Let Ψ_{-k} denote the state space of arms not equal to k.

Define a new policy $\tilde{OPT}$ that plays arm k once and then mimics OPT. This means that suppose OPT plays arm j conditioned on some state in Ψ_{-k}, $\tilde{OPT}$ plays the same arm. When OPT plays arm k for the first time, $\tilde{OPT}$ skips that play of arm k, since it has already played that arm. Subsequently, both $\tilde{OPT}$ and OPT are *coupled* in that they have the same state in Ψ, and perform the same actions. Note that to perform the exchange argument and coupling, we are crucially using the *bandit property*: Plays of other arms do not change the state of arm k, and vice versa. If moving the play of arm k ahead changed the states of other arms, this transformation would not work.

Let $\bar{r}(t)$ be the rate of reward as a function of time under the policy OPT, *i.e.*, if $s(t)$ is the state of the arm played at time t, then $\bar{r}(t) = r(s(t))$. Since $s^* = \arg\max_{s\in\Psi} r(s)$, $\bar{r}(t) \leq r(s^*)$ for all t. If the expected discounted reward under policy OPT is $J(OPT)$, then

$$J(OPT) = \mathbf{E}\left[\sum_{t=0}^{\tau-1}\bar{r}(t)\gamma^t + \gamma^\tau \sum_{t=0}^{T(s^*)-1} r(s^*)\gamma^t + \sum_{t=\tau+T(s^*)}^{\infty}\bar{r}(t)\gamma^t\right]$$

Similarly,

$$J(\tilde{OPT}) = \mathbf{E}\left[\sum_{t=0}^{T(s^*)-1} r(s^*)\gamma^t + \gamma^{T(s^*)}\sum_{t=0}^{\tau-1}\bar{r}(t)\gamma^t + \sum_{t=\tau+T(s^*)}^{\infty}\bar{r}(t)\gamma^t\right]$$

If we prove the following, then we would have shown that $J(\tilde{OPT}) \geq J(OPT)$, proving the lemma.

$$\mathbf{E}\left[(1-\gamma^{\tau}) \sum_{t=0}^{T(s^*)-1} r(s^*)\gamma^t\right] \geq \mathbf{E}\left[(1-\gamma^{T(s^*)}) \sum_{t=0}^{\tau-1} \bar{r}(t)\gamma^t\right]$$

To do this, observe that $T(s^*)$ is independent of τ. This is because $T(s^*)$ depends on the outcome of the play of arm k in state s^*, while τ depends on the outcomes of the plays of other arms. Therefore, we need to show:

$$\mathbf{E}\left[(1-\gamma^{\tau})\right] \mathbf{E}\left[\sum_{t=0}^{T(s^*)-1} r(s^*)\gamma^t\right] \geq \mathbf{E}\left[(1-\gamma^{T(s^*)})\right] \mathbf{E}\left[\sum_{t=0}^{\tau-1} \bar{r}(t)\gamma^t\right]$$

Since $\bar{r}(t) \leq r(s^*)$ for all t, it is sufficient to show the following:

$$\mathbf{E}\left[(1-\gamma^{\tau})\right] \mathbf{E}\left[\sum_{t=0}^{T(s^*)-1} r(s^*)\gamma^t\right] \geq \mathbf{E}\left[(1-\gamma^{T(s^*)})\right] \mathbf{E}\left[\sum_{t=0}^{\tau-1} r(s^*)\gamma^t\right]$$

Canceling out the deterministic value $r(s^*)$ from both sides, and contracting the geometric series, we need to show:

$$\mathbf{E}\left[(1-\gamma^{\tau})\right] \mathbf{E}\left[(1-\gamma^{T(s^*)})\right] \geq \mathbf{E}\left[(1-\gamma^{T(s^*)})\right] \mathbf{E}\left[(1-\gamma^{\tau})\right]$$

which is trivially true. This completes the proof. $\square$

We finally have the Gittins index theorem.

Theorem 6.3.5. *If the state space of all the arms are finite, then the Gittins index policy described in Figure 6.1 is optimal.*

Proof. Let $N = |\Psi|$ be the cardinality of the set Ψ. We prove the claim by induction on N. The previous lemma shows that assigning the highest priority to state $s^* = \arg\max_{s \in \Psi} r(s)$ is optimal. Given this, we can contract this state as follows. Suppose this state belongs to arm k. For each $s \in \mathcal{S}_k$ where $s \neq s^*$, consider the following process: Play the arm in state s; if the state exits from s to $s' \neq s^*$, then stop playing. Else, continue playing until the state exits s^*, and stop. Let $R(s)$ denote the discounted reward of this composite play, $T(s)$ denote the time elapsed, and $Y(s)$ denote the random state at which the process stops. Redefine

$$r(s) = \frac{\mathbf{E}[R(s)]}{\mathbf{E}\left[\sum_{t=0}^{T(s)-1} \gamma^t\right]}$$

This new state space $\mathcal{S}_k$ eliminates the state s^*. This yields a new multi-armed bandit instance. The claim is that the optimal policy for this new instance is the same as the optimal policy for the original instance. The reason is simple: If the original bandit had entered state s^* because of a play to arm k in state s, the optimal policy would have continued to play arm k in s^*. The new bandit mimics this behavior within the new composite state s, fetching the same random reward and the same random next state.

For the new bandit, the optimal policy again plays the state with highest per-step reward if it observes the corresponding arm in that state. This yields an iterative procedure for computing priorities of the states which is exactly the same as the procedure in Figure 6.1. The details are simple to fill in, and left as an exercise. $\square$

6.4 Structure of the Index in Bayesian Setting

The Gittins index appears somewhat mysterious. However, in the case of the Bayesian bandit, it roughly corresponds to doing the most natural thing.

Suppose the arm has an underlying reward distribution $f_\theta(x)$, and the states are priors $r_{a,b}(\theta)$ over the parameter θ of this distribution. Let $\mu(\theta) = \int_x x f_\theta(x) dx$ denote the mean reward of the arm given parameter θ. For prior $r_{a,b}(\theta)$, let $F_{a,b}(y) = \Pr[\mu(\theta) > y]$. Now, the Gittins index for discount factor γ is roughly given by $\lambda^*(a,b) = F_{a,b}^{-1}(1-\gamma)$. In particular, if there is a finite time horizon of T on the policy execution, the policy can approximated by using a discount factor of $\gamma = 1 - 1/T$. Then, the Gittins index is roughly $F_{a,b}^{-1}(1/T)$. To see why this makes sense, the probability of the mean exceeding $\lambda^*(a,b)$ is $1/T$, in which case we obtain reward of approximately $\lambda^*(a,b)$ for T steps, yielding an expected total reward is $\lambda^*(a,b)$. Therefore, if the penalty per step is set to $\lambda^*(a,b)$, it cancels out the reward, so that the Gittins index is roughly this value of penalty.

Though this calculation is heuristic, it shows an important principle of *optimism in the face of uncertainty*: The policy uses the optimistic tail bound on the reward distribution as index – the reward value that can be achieved with probability $1/T$. Given two arms with comparable mean, the index will therefore be higher for an arm with larger variance. This captures the *exploratory* nature of the policy – use initial plays to resolve arms with high variance in reward. We will see the same principle applied later when we discuss policies that do not need/use prior information.

Part II

Unknown Distributions and Online Prediction

Chapter 7

The Finite Horizon Multi-armed Bandit Problem

So far, we have considered the case where there is a probabilistic model of uncertainty is specified upfront. The policy we design *assumes* this model is the ground truth, and nature chooses a realization from the probability distribution specified by the model. In this case, the problem of designing a policy becomes computational – the state space that a policy must reason about is usually exponential in size and the goal of the algorithm designer is to make this space (approximately) tractable.

One criticism of this approach is that it is often not possible to specify a probabilistic model of uncertainty; further, the algorithms tend to be brittle in the sense that if the model is mis-specified, the performance of the algorithm can significantly deteriorate. For instance, the Gittins index policy for Bayesian bandits requires knowledge of the prior distribution of each arm. What if this prior is wrong? More importantly, how do we even find out if the prior is right or wrong? What made us use one prior versus another? These concerns can be mitigated if there is sufficient data to learn a model from, but this is often not the case.

Such criticism can be ameliorated by having a more robust model of uncertainty, something that does not require specifying a complete probabilistic model of how nature will behave. One extreme is to assume that nature can behave anyhow it pleases, and we analyze our policy over the worst possible outcome of the uncertain inputs. This leads to the area of *online algorithms* which will be the focus of the third part of the book.

In this part, we consider an intermediate model – the input is not worst case, but follows an underlying probabilistic model. However, the algorithm designer does not know the model, and has to *learn it* while optimizing the desired objective function at the same time. For instance, in the case of Bayesian bandits, suppose all we know is that each arm has an underlying reward distribution from which rewards are drawn $i.i.d.$ every time the arm is pulled. But unlike the Bayesian case where a prior is specified over possible parameters of this distribution, suppose the algorithm designer does not have any information about what the underlying distribution can be.

7.1 The Regret Measure

In the presence of prior distributions over the parameters of the underlying reward distributions, it was straightforward to define the optimal policy and its expected reward. In the absence of such prior distributional knowledge, we need an upper bound on the optimal policy. This is simply achieved by assuming the optimal policy is *omniscient* and knows the parameters of the underlying reward distributions.

More formally, each of n arms in the bandit problem gives a reward governed by a random variable $X_i \in [0,1]$ with mean μ_i. The policy is not aware of the reward distributions of the arms. Each play of an arm yields a reward drawn $i.i.d.$ according to X_i. Given a finite horizon T, we seek a policy that maximizes the expected total reward over T plays.

As we show below, when $n \ll T$, it is relatively straightforward to design policies whose expected reward is very close to that of the omniscient policy. A better and more fine-grained measure of performance is therefore the difference between the reward of our policy and the omniscient policy. This is termed *regret*, and measures the worst-case loss of our policy due to lack of information about the underlying reward distributions.

In order to define regret, note that the omniscient optimum would simply play the arm with highest expected reward $i^* = \arg\max\limits_i \mu_i$ for all T steps. Suppose our policy obtains reward R_t at each step. Then:

$$\text{Regret} = \max_{\{X_i : i=1,\ldots,n\}} \left(T\mu_{i^*} - \mathbf{E}\left[\sum_{t=1}^{T} R_t\right]\right)$$

Note that regret is an *informational* notion, and measure loss due to lack of information. This is in contrast with the notion of approximation in stochastic optimization problems, where both the optimal policy and our policy work with the same distributional information, and the approximation loss is due to lack of *computational* power in computing the optimal policy.

7.2 A Naive Policy

We will end up showing a policy that obtains a regret that grows as $\log T$. Thus, per step regret vanishes for large T. The idea is to balance the regret incurred when exploring new arms against the reward obtained by exploiting the best known arm uncovered so far. First, we describe a naive policy that incurs a total regret of $\tilde{O}(nT^{\frac{2}{3}})$ assuming $n \leq T^{1/3}$.

Before analyzing the algorithm, we state Hoeffding's inequality, which we will repeatedly use. The proof is omitted and can be found in any textbook on randomized algorithms.

Lemma 7.2.1 (Hoeffding's inequality)**.** *Let $X_1, X_2, \ldots, X_k$ be k independent draws from a distribution on $[0,1]$ with mean μ. Let $s = \frac{1}{k}\sum_{i=1}^{k} X_i$ denote the sample*

- Play each arm for $T^{\frac{2}{3}}$ steps;
- Choose the arm with maximum sample average reward and play it for the remaining $T - nT^{\frac{2}{3}}$ steps.

Figure 7.1: Naive Algorithm for the stochastic MAB problem.

average of the draws. Then:

$$\Pr\left[|s - \mu| > \epsilon\right] \leq 2e^{-2k\epsilon^2}$$

We will use Hoeffding's inequality to upper bound the number of plays needed to estimate the mean reward μ of an arm: For any $\epsilon, \delta \geq 0$, the sample average $\hat{\mu}$ of $m = O\left(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\right)$ plays satisfies:

$$\Pr\left[|\hat{\mu} - \mu| \leq \epsilon\right] \geq 1 - \delta$$

In other words, learning the mean reward of an arm to within error ϵ requires $O(1/\epsilon^2)$ plays of the arm. Most analyses of the stochastic bandit problem proceed using this bound. In Section 7.4, we show a matching lower bound of $\Omega(1/\epsilon^2)$ for this problem.

Lemma 7.2.2. *Algorithm 7.1 incurs $O(nT^{\frac{2}{3}}\sqrt{\log(nT)})$ regret, assuming $n \leq T^{1/3}$.*

Proof. Let $\hat{\mu}_i$ be the sample average of arm i. By Hoeffding's inequality (using $k = T^{2/3}$):

$$\Pr\left[|\mu_i - \hat{\mu}_i| > \frac{\sqrt{\log(nT)}}{T^{\frac{1}{3}}}\right] \leq \frac{2}{n^2T^2}$$

By union bounds, we have:

$$\Pr\left[\text{There exists } i \text{ such that } |\mu_i - \hat{\mu}_i| > \frac{\sqrt{\log(nT)}}{T^{\frac{1}{3}}}\right] \leq \frac{2}{nT}$$

Hence with probability $1 - \frac{2}{nT}$, all sample means are close to the true means. This means the chosen arm i (with largest sample mean) satisfies the following relation for its true mean:

$$\mu_i \geq \mu_{i^*} - \frac{2\sqrt{\log(nT)}}{T^{\frac{1}{3}}}$$

Now, the regret has three components, which we add below:

$$\text{Regret} \leq nT^{2/3} + T \times \frac{2\sqrt{\log(nT)}}{T^{\frac{1}{3}}} + \frac{2}{nT} \times T = O(nT^{\frac{2}{3}}\sqrt{\log(nT)})$$

The first term above is the regret due to exploring the arms for $nT^{2/3}$ steps, the second term is the regret due to choosing an arm that is sub-optimal by $\frac{2\sqrt{\log T}}{T^{\frac{1}{3}}}$ each step and playing it for at most T steps, and the final term is due to the low-probability event of choosing an arm that is sub-optimal by 1 each step. $\square$

7.3 The UCB1 Policy

The reason the naive algorithm is suboptimal is that it continues sampling an arm even when there is no hope this arm has the highest mean reward. In contrast, a policy like the Gittins index only plays an arm when the per-step expected reward from continuously playing it is best possible. The UCB1 algorithm mimics this behavior, and can essentially be thought of as the Gittins index without the prior.

Instead of defining an index using the prior, the index for each arm is simply the high confidence upper bound on the true mean. At each step Algorithm 7.2 plays the arm with the highest index and updates the value of the index. The index value of each arm depends only on the sample average of the arm and the number of times it has been played and hence can be updated quickly.

Formally, for arm i, let $\hat{\mu_i}$ be the sample average so far and t_i be the number of times it has been played. We define

$$\text{Index}_i = \hat{\mu}_j + \sqrt{\frac{\ln T}{t_j}}$$

Note that using Hoeffding's inequality,

$$\Pr[\mu_i > \text{Index}_i] \leq \frac{1}{T}$$

Therefore, the index is an optimistic upper bound on the mean given the sample average. The Gittins index for the Bayesian bandits was also essentially using the same definition, but basing its estimate on the prior instead of using Hoeffding's inequality.

- Play the arm $k = \arg\max_i \text{Index}_i$.
- Update Index_k based on the observed reward.

Figure 7.2: The UCB1 Policy.

7.3.1 Analysis

Let i^* denote the arm with highest true mean, μ^*. Let $\Delta_i = \mu^* - \mu_i$ be the regret on playing arm i once. We will bound the overall regret in terms of these parameters.

Let Q_i be the expected number of times Algorithm 7.2 plays arm i in T steps.

Lemma 7.3.1. *For each arm $i \neq i^*$,*

$$\mathbf{E}[Q_i] \leq \frac{4 \ln T}{{\Delta_i}^2} + 2$$

Proof. We use Hoeffding's inequality in two parts.

First, we will show that $\Pr[\text{Index}_{i^*} < \mu^*] \leq 1/T$. To see this, pretend arm i^* is played continuously. At each step t,

$$\text{Index}_{i^*}(t) = \hat{\mu}_{i^*} + \sqrt{\frac{\ln T}{t}}$$

By Hoeffding's inequality applied to step t,

$$\Pr[\text{Index}_{i^*}(t) < \mu^*] \leq 1/T^2$$

By union bounds over T steps, this implies $\Pr[\text{Index}_{i^*} < \mu^*] \leq 1/T$. Since the outcomes of the plays of arm i^* are independent of the plays of other arms, the above is true even when the plays of arm i^* are arbitrarily interspersed with the plays of other arms.

Secondly, if arm i has already been played $s_i = \frac{4 \ln T}{{\Delta_i}^2}$ steps, then

$$\text{Index}_i = \hat{\mu}_i + \frac{\Delta_i}{2}$$

This implies

$$\Pr[\text{Index}_i > \mu^*] = \Pr[\hat{\mu}_i - \mu_i > \Delta_i/2]$$

Again applying Hoeffding's inequality, this probability is at most $1/T$.

Therefore, if arm i has been played s_i steps, then with probability $1 - 2/T$, the index of this arm is below μ^* *and* the index of arm i^* always stays above μ^*. This means that arm i is played at most $s_i = \frac{4 \ln T}{{\Delta_i}^2}$ times with probability $1 - 2/T$. With the remaining probability, the arm is played at most T steps. Taking expectations,

$$\mathbf{E}[Q_i] \leq \frac{4 \ln T}{{\Delta_i}^2} \times \left(1 - \frac{2}{T}\right) + \frac{2}{T} \times T \leq \frac{4 \ln T}{{\Delta_i}^2} + 2$$

□

Lemma 7.3.2. *Algorithm 7.2 incurs expected regret*

$$\sum_{i \neq i^*} \left(\frac{4 \ln T}{\Delta_i} + 2\Delta_i \right)$$

In the worst case, this value is $O(\sqrt{nT \ln T})$.

Proof. The expected regret can be written as

$$\text{Regret} = \sum_{i \neq i^*} \Delta_i \mathbf{E}[Q_i]$$

since whenever arm i is played, the expected regret of that play is Δ_i independent of all other plays. Using the bound for $\mathbf{E}[Q_i]$ from the previous lemma, we have

$$\text{Regret} = \sum_{i \neq i^*} \left(\frac{4 \ln T}{\Delta_i} + 2\Delta_i \right)$$

We now compute the worst case value of the above bound over choices of $\{\Delta_i\}$. Define S_1 as the set of arms with $\Delta_i > \sqrt{\frac{4n\ln T}{T}}$ and S_2 be the rest of the arms, with $\Delta_i \le \sqrt{\frac{4n\ln T}{T}}$. For the arms in S_1,

$$\text{Regret} \le n\frac{4\ln T}{\sqrt{\frac{4n\ln T}{T}}} = 2\sqrt{nT\ln T}$$

For arms in S_2,

$$\text{Regret} \le T\max\Delta_i \le T\sqrt{\frac{4n\ln T}{T}} = 2\sqrt{nT\ln T}$$

Adding these two bounds up, the worst case regret is $O(\sqrt{nT\ln T})$. □

7.4 Lower Bounds on Regret: UCB1 is Nearly Optimal

We now show that UCB1 is optimal up to low order terms, *i.e.*, any policy for the finite horizon bandit problem (with n arms and horizon T) must suffer from $\Omega(\sqrt{nT})$ regret simply because it does not know the underlying reward distributions, while the policy against which it is compared knows the distributions. We will state two different lower bounds, but will prove only one of them.

7.4.1 Preliminaries: Dissimilarity Measures between Distributions

For stating as well as proving the lower bounds, it is useful to first define some distance measures between distributions. For simplicity, we assume throughout that when comparing two distributions, the support (or domain) is the same.

Definition 4. *Given two probability distributions P and Q over the same domain Z, the* KL-divergence *of P with respect to Q is defined as:*

$$D(P,Q) = \sum_{\sigma\in Z} P(\sigma)\log_2\frac{P(\sigma)}{Q(\sigma)} = \mathbf{E}_{\sigma\sim P}\left[\log_2\frac{P(\sigma)}{Q(\sigma)}\right]$$

We can think of $D(P,Q)$ as being an asymmetric distance measure between P and Q. We note the following properties.

Property 1: $D(P,P)=0$. Further, as we will see below, the converse is also true: If $D(P,Q)=0$, then P is the same distribution as Q.

Property 2: $D(P,Q)\ge 0$. To see this, note that $\log$ is a concave function. Therefore, we can use Jensen's inequality:

Lemma 7.4.1 (Jensen's Inequality)**.** *Let f be a concave function, and X denote a random variable. Then*

$$\mathbf{E}[f(X)] \le f(\mathbf{E}[X])$$

We will use $X = \frac{Q(\sigma)}{P(\sigma)}$, and $f = \log_2$. The expectation is over the distribution induced by P. Then,

$$\begin{aligned} D(P,Q) &= -\mathbf{E}\left[f(X)\right] \geq -f(\mathbf{E}[X]) \\ &= -\log_2\left(\mathbf{E}_{\sigma\sim P}\left[\frac{Q(\sigma)}{P(\sigma)}\right]\right) \\ &= -\log_2\left(\sum_\sigma P(\sigma)\cdot\frac{Q(\sigma)}{P(\sigma)}\right) \\ &= -\log_2 1 = 0 \end{aligned}$$

Property 3: Chain Rule. Suppose P and Q are both joint distributions over two random variables X and Y. We denote these distributions as $P(X,Y)$ and $Q(X,Y)$. Let $p(x)$ denote the probability that $X = x$ according to P; similarly define $q(x)$. Similarly, denote by $P(X)$ the marginal distribution of X induced by P, and $Q(X)$ as the marginal distribution of X induced by Q.

We can now define the KL-Divergence conditioned on X as:

$$\begin{aligned} D(P,Q\,|X) &= \sum_{x\in X} p(x) D(P(Y|X=x), Q(Y|X=x)) \\ &= \mathbf{E}_{x\sim P(X)}\left[D(P(Y|X=x), Q(Y|X=x))\right] \end{aligned}$$

Lemma 7.4.2 (Chain Rule)**.**

$$D(P,Q) = D(P(X),Q(X)) + D(P,Q\,|X)$$

The proof follows a similar outline to the proof of Lemma 3.3.1, and we leave it as an exercise. As an easy corollary, note that when $P(X,Y) = P(X)P(Y)$, and $Q(X,Y) = Q(X)Q(Y)$, that is, the variables X and Y are independent, then the KL-Divergence is additive, a property we will use repeatedly:

$$D(P,Q) = D(P(X),Q(X)) + D(P(Y),Q(Y))$$

Total Variation Distance and Pinsker's Inequality

We now present a different notion of distance – the distance between the distributions according to the l_1-norm. Formally, the Total Variation Distance (TVD) between two distributions P and Q is given by:

$$TVD(P,Q) = ||P-Q||_1 = \sum_\sigma |P(\sigma) - Q(\sigma)|$$

Note that $TVD \in [0,2]$, and defines a metric over the distributions. In deriving our bounds, we will need to obtain an upper bound on the TVD between two distributions over a complex event space. This can be hard to do directly. However, we can use a general result that upper bounds the TVD by the KL-Divergence. Once we do this, we can use the chain rule to decompose the KL-Divergence in the complex space to a sum of KL-Divergences of elementary events.

Theorem 7.4.3 (Pinsker's Inequality.)**.**

$$TVD(P,Q) \leq \sqrt{2 \ln 2 \cdot D(P,Q)}$$

Proof. Suppose $P = \text{Bernoulli}(1,p)$ and $Q = \text{Bernoulli}(1,q)$. Then elementary calculus shows that the inequality is true, and this is left as an exercise.

The general case follows by reducing to the Bernoulli case as follows: Let $A = \{\sigma | P(\sigma) \geq Q(\sigma)\}$. Define two Bernoulli distributions P_A and Q_A, where P_A is 1 with probability $\sum_{\sigma \in A} P(\sigma)$, and Q_A is 1 with probability $\sum_{\sigma \in A} Q(\sigma)$. Define a random variable Z to be 1 if $\sigma \in A$ and 0 otherwise.

It is easy to check that $TVD(P,Q) = TVD(P_A, Q_A)$. Since the latter distributions are Bernoulli, we have:

$$TVD(P,Q) = TVD(P_A,Q_A) \leq \sqrt{2 \ln 2 \cdot D(P_A,Q_A)}$$

By the Chain Rule and non-negativity of the KL-Divergence,

$$D(P,Q) = D(P_A,Q_A) + D(P,Q\,|Z) \geq D(P_A,Q_A)$$

Combining the above two inequalities, we complete the proof. $\square$

We will use Pinsker's inequality in the form of this corollary, whose proof we leave as an easy exercise.

Corollary 7.4.4. *Let f be any function mapping each σ to a value in $[0,M]$. Then*

$$|\mathbf{E}_{\sigma \sim P}[f(\sigma)] - \mathbf{E}_{\sigma \sim Q}[f(\sigma)]| \leq \frac{M}{2} \cdot TVD(P,Q) \leq \frac{M}{2} \cdot \sqrt{2 \ln 2 \cdot D(P,Q)}$$

7.4.2 Two Lower Bounds on Stochastic Bandits

We now present the two lower bounds on the regret for the stochastic MAB problem. Let arm i^* have the highest mean. Recall that for arm $i \neq i^*$, $Q_i(T)$ is the random variable denoting the number of times this arm is played by the policy assuming a time horizon of T.

Theorem 7.4.5 (Lai and Robbins '84)**.** *Suppose the reward distribution of arm i is $P_i = \text{Bernoulli}(1,p_i)$. Consider any bandit policy that satisfies for some $a \in (0,1)$ and all $i \neq i^*$ the condition:*

$$\mathbf{E}[Q_i(T)] = o(T^a)$$

Then,

$$\lim_{T \to \infty} \frac{\text{Regret}(T)}{\ln T} \geq \sum_{i \neq i^*} \frac{\Delta_i}{D(P_i, P_{i^*})}$$

where $\Delta_i = p_{i^*} - p_i$ and

$$D(P_i,P_{i^*}) = p_i \log_2 \frac{p_i}{p_{i^*}} + (1-p_i) \log_2 \frac{1-p_i}{1-p_{i^*}}$$

This shows that for fixed $\{\Delta_i\}$, the dependence of the regret on $O(\ln T)$ is unavoidable in the UCB1 policy.

We will not prove this theorem. Instead, we will prove the following theorem that holds uniformly over time (instead of in the limit) and in the worst case over all underlying distributions:

Theorem 7.4.6 (Auer, Cesa-Bianchi, Freund, Schapire. '94)**.** *Any deterministic multi-armed bandit policy with n arms and time horizon T suffers regret at least $\Omega(\sqrt{nT})$.*

7.4.3 Key Analysis Idea: Single Arm Problem

Suppose we are given an arm whose reward distribution is Bernoulli$(1, p)$, where we are only told that p is either $p = 1/2$ or $p = 1/2 + \epsilon$. How many plays of the arm are needed before we can correctly identify p with constant probability, say $4/5$?

We can use Pinsker's inequality and the Chain rule to lower bound the number of plays m needed by any successful policy. Any policy is a mapping f from the observed m-dimensional $0/1$ reward vectors $\vec{r}$ to the indicator variable $0, 1$, where 0 denotes we identify $p = 1/2$ and 1 denotes we identify $p = 1/2+\epsilon$. Let $P(\vec{r})$ denote the probability of reward vector $\vec{r}$ when $p = 1/2$, and let $Q(\vec{r})$ denote the corresponding probability when $p = 1/2 + \epsilon$.

Since the policy identifies p correctly with probability $4/5$, this translates to the following conditions:

$$\mathbf{E}_{\vec{r}\sim P^m}\left[f(\vec{r})\right] \le \frac{1}{5} \qquad \text{and} \qquad \mathbf{E}_{\vec{r}\sim Q^m}\left[f(\vec{r})\right] \ge \frac{4}{5}$$

By Corollary 7.4.4 applied to this function f, we have:

$$\frac{3}{5} \le \frac{1}{2} \cdot \sqrt{2 \ln 2 \cdot D(P^m, Q^m)}$$

Note that P^m and Q^m are distributions of the reward vectors for m $i.i.d.$ draws from P, respectively Q. Since the KL-Divergence is additive for independent events, we have:

$$D(P^m, Q^m) = m \cdot D(P, Q)$$

The quantity $D(P, Q)$ is easy to estimate, since it is the KL-Divergence between Bernoulli$(1, 1/2)$ and Bernoulli$(1, 1/2 + \epsilon)$. For $\epsilon \le 1/4$, it is easy to show that $D(P, Q) \le c_1\epsilon^2$ for constant value c_1. Plugging this into Pinsker's bound, we get:

$$m \ge \frac{c_2}{\epsilon^2}$$

for constant value c_2. The point is that the number of plays needed is $\Omega(1/\epsilon^2)$; the exact constant factors in the bound are uninteresting and omitted.

7.4.4 Proof of Theorem 7.4.6

In the case of a single arm, the only decision the policy has is to stop versus play. Till it stops, the rewards it observes are independent draws from the same distribution. In the case of multiple arms, this is no longer true – a policy's choice of arm affects the rewards it observes. Nevertheless, we can apply Pinsker's inequality in a fashion that is very close to the single arm case.

The instance achieving the regret bound is simple: All arms except one have reward distribution Bernoulli$(1, \frac{1}{2})$, while a single randomly chosen arm is *good* (or is *biased*) and has reward distribution Bernoulli$(1, \frac{1}{2} + \epsilon)$, where $\epsilon = c\sqrt{\frac{n}{T}}$ for small constant c. The omniscient policy knows the identity of this arm.

The basic idea in the analysis is the following: Fix an algorithm A. Suppose none of the arms had a bias, so they were all identical. Then there exist many arms whose probability of being played by A is small at any time step, just by an averaging argument. Further, there are many arms that will not be played too many times, again by an averaging argument. This means a randomly placed biased arm is quite likely to have been played few enough times that it cannot be distinguished from an unbiased arm. Formalizing this argument requires some work, and showcases the power of KL-Divergence as an analytic tool.

Any bandit policy in this setting is a mapping of observed rewards until time t to an action, α_t, where α_t is the arm chosen to play at time t. We assume without loss of generality that α_t is also the policy's current guess of the biased arm.

Consider $n + 1$ different "worlds": In world $j \in \{1, 2, \ldots, n\}$, arm j is the good arm. In world 0, no arm is biased and all n arms are Bernoulli$(1, 1/2)$. Fix a time step t, and let $\vec{r}$ denote the reward sequence observed by the policy in time steps $1, 2, \ldots, t-1$. (We implicitly index each reward by the arm it was observed for, so that this is indeed a complex space of events!) Let P_j denote the distribution of this reward sequence in world j. The policy deterministically maps each reward sequence to an action α_t, so that each P_j also specifies a distribution over α_t.

The key quantity we will be interested in is the value $\Pr_{P_j}[\alpha_t = j]$. This is the probability that arm j is played by the policy at step t given that we are in a world where j is the good arm. The claim is that this value cannot be large for all j. The proof will crucially use the Chain rule, and shows why we needed the whole machinery of KL-Divergence.

Lemma 7.4.7. *Assume $t \le c\frac{n}{\epsilon^2}$ for small enough constant c. Further assume n is larger than a constant, and ϵ is a small enough constant. For any bandit algorithm A, there exists a set $S_t \subseteq \{1, 2, \ldots, n\}$ of arms such that:*

- $|S_t| \ge n/3$*; and*
- *For all $j \in S_t$,* $\Pr_{P_j}[\alpha_t = j] \le 1/2$.

Proof. First note that the set S_t should only be defined based on the description of algorithm A and not on its execution path. Towards this end, consider the execution of A in world 0 where no arm is biased. Recall that Q_j is the random variable denoting the number of times arm j is played by the policy.

Define two sets of arms:

$$S_1 = \left\{ j \mid \mathbf{E}_{P_0}[Q_j] \le \frac{3t}{n} \right\}$$

$$S_2 = \left\{ j \mid \Pr_{P_0}[\alpha_t = j] \le \frac{3}{n} \right\}$$

By an averaging argument, since the total number of arms played is at most t, and at most 1 arm is played at step t, both S_1 and S_2 have size at least $2n/3$. This means $S_1 \cap S_2$ has size at least $n/3$. Call this set S_t. In words, S_t is the set of arms that, assuming no arm is biased, are played by A at most $3t/n$ times in expectation, and whose probability of being chosen at time t is at most $3/n$. That is arms that have been "neglected" by A in the unbiased world.

The claim is that these arms will still be neglected at time t in the world where that arm is biased! Focus on some arm $j \in S_t$. Let f denote the function that maps each reward sequence to the binary event $\alpha_t = j$, so that $f(\vec{r}) = 1$ if $\alpha_t = j$ and 0 otherwise. By Pinsker's inequality:

$$\left| \Pr_{P_j}[\alpha_t = j] - \Pr_{P_0}[\alpha_t = j] \right| = \left| \mathbf{E}_{P_j}[f] - \mathbf{E}_{P_0}[f] \right| \le \frac{1}{2}\sqrt{2\ln 2 \cdot D(P_j, P_0)}$$

This means

$$\Pr_{P_j}[\alpha_t = j] \le \frac{3}{n} + \frac{1}{2}\sqrt{2\ln 2 \cdot D(P_j, P_0)}$$

This is where using KL-Divergence instead of TVD comes in really handy. Both P_j and P_0 are distributions of rewards observed in steps $1, 2, \ldots, t-1$, which is a complex event space. For vector $\vec{r}$ of rewards, let $\vec{r_s}$ denote the rewards observed in the first s steps, where $s \le t-1$. Let r_s denote the reward at step s. We will use the Chain rule as follows.

$$D(P_j, P_0) = \sum_{s=1}^{t-1} \left(\sum_{\vec{r_s}} \Pr_{P_0}[\vec{r_s}] \times D(P_j(r_{s+1}), P_0(r_{s+1}) \mid \vec{r_s}) \right)$$

Observe now that conditioned on observing $\vec{r_s}$ as rewards, policy A deterministically decides on the next arm to play regardless of which underlying world it is in. Since the arms yield $i.i.d.$ rewards, this means that $D(P_j(r_{s+1}), P_0(r_{s+1}) \mid \vec{r_s})$ is simply the KL-Divergence of the reward distribution of the arm played at step $s+1$ in the worlds 0 and j. If the arm played is not j, the reward distribution is Bernoulli$(1, 1/2)$ in both worlds, so that $D(P_j(r_{s+1}), P_0(r_{s+1}) \mid \vec{r_s}) = 0$. On the other hand, if the arm played is j, then this value is the KL-Divergence between Bernoulli$(1, 1/2)$ and Bernoulli$(1, 1/2 + \epsilon)$, which is at most $2\epsilon^2$ for ϵ a small enough constant.

This yields the first part of the inequality below; the second part follows from the choice of $j \in S_t$:

$$D(P_j, P_0) \le \sum_{s=1}^{t-1} \Pr_{P_0}[\alpha_{s+1} = j] \times 2\epsilon^2 = \mathbf{E}_{P_0}[Q_j] \times 2\epsilon^2 \le \frac{3t}{n} \times 2\epsilon^2$$

Plugging this bound into Pinsker's inequality, we finally obtain:

$$\Pr_{P_j}\left[\alpha_t = j\right] \leq \frac{3}{n} + \frac{1}{2}\sqrt{c_3 \frac{t}{n}\epsilon^2} \leq \frac{1}{2}$$

assuming the constants are appropriately chosen. □

The rest of the proof is now simple. If the biased arm is placed uniformly at random, then at any step t, this arm lies within S_t with probability $1/3$. In this event, the algorithm A chooses this arm with probability at most $1/2$, and with the remaining $1/2$ probability, it incurs regret ϵ. Therefore, the expected regret of the algorithm over T steps is $\Omega(\epsilon T)$. Plugging in $\epsilon = c\sqrt{\frac{n}{T}}$, the regret is $\Omega(\sqrt{nT})$ completing the proof.

7.5 An Optimal Policy: Thompson Sampling

Thompson sampling is the oldest heuristic for the MAB problem, and in fact empirically the best known. It takes a Bayesian view of the MAB problem, and applies a technique termed *probability matching*.

7.5.1 Recap of Bayesian Bandits

Let us assume for simplicity that each arm i has an underlying distribution Bernoulli$(1, p_i)$ where p_i are unknown initially. As discussed in the previous chapter, in a Bayesian world, we impose a prior distribution over the parameter p_i. The most natural prior for Bernoulli distributions is the Beta density. This density is captured by two parameters α and β. The distribution Beta(α, β) corresponds to the following: Suppose we initially know nothing about p. Then this corresponds to p being a value uniformly at random in $[0, 1]$. The uniform distribution is the same as Beta$(1, 1)$. Subsequently suppose we play the arm m times and observe m_1 1's and $m - m_1$ 0's. Then our belief about p changes. We can compute the probability that any given value of p would have generated the sequence we observed. We then compute the *posterior* distribution over possible values of p by using Bayes rule. The distribution we get is exactly Beta$(m_1 + 1, m - m_1 + 1)$. The expected value of this distribution is $\frac{m_1+1}{m+2}$, which is the expected value of p conditioned on our observations and the prior we started with. If we play the arm again, we would expect to observe 1 with exactly this probability. Conditioned on observing 1, the posterior would become Beta$(m_1 + 2, m - m_1 + 2)$, since we would have observed $m_1 + 1$ successes in $m + 1$ plays of the arm.

The posterior density therefore captures our current belief about the underlying parameter of the arm. Recall that in the discounted reward version, the Gittins index policy uses this information optimally to maximize expected reward. However, such optimality requires knowing the discount factor, or the time horizon over which the policy will execute.

7.5.2 A First Attempt

The Thompson sampling policy uses the posterior, but in a way that is oblivious to the overall time horizon. Given these posterior densities, it computes for each arm i the probability that this arm's underlying parameter is the maximum. Call this quantity q_i. We will derive a policy that uses the $\{q_i\}$ values at any point in time to decide which arm to play at that time instant. Intuitively, q_i captures the probability arm i is the eventual best arm, so using these values should give us a meaningful policy. What should this policy be?

A first attempt is to we the arm with the highest q_i at every time step. We will show this has regret linear in T. To see this, consider two arms: Arm 1 has Bernoulli$(1, p_1)$, and arm 2 has Bernoulli$(1, p_2)$. For arm 1, suppose the policy knows that $p_1 = 2/3$. This corresponds to Beta$(2n/3, n)$ for some large value of n. For arm 2, suppose the prior on p_2 is Beta$(1, 3)$, so that $\mathbf{E}[p_2] = 1/3$. For the distribution Beta$(1, 3)$, we have

$$\Pr\left[\text{Beta}(1,3) \geq p\right] = 1 - p$$

Therefore, $q_2 = \Pr[\text{Beta}(1,3) > 2/3] = 1 - 2/3 = 1/3$. This means the policy plays arm 1 at the current time step. However, since p_1 is known to be $1/2$, the posterior density of both the arms at the next time step remains the same as the current posterior. This means q_1 and q_2 remain unchanged, and the policy repeatedly plays arm 1 for all T steps, incurring reward $2T/3$ in expectation. Consider next a policy that plays arm 2 initially till it learns its mean reward with reasonable certainty. If the mean reward is more than $4/5$, the policy plays this arm for all T steps, else it switches to arm 1. Given the Beta$(1, 3)$ prior on p_2, the probability that p_2 is at least $4/5$ is $1/5$. This means the policy has expected reward per time step at least $\frac{1}{5} \times \frac{4}{5} + \frac{4}{5} \times \frac{2}{3} = 0.6933$, where the first term is the reward from playing arm 2, and the second term is the reward from switching to arm 1. This means that over T steps, this policy has reward at least $0.02T$ more than the policy that plays the maximum q every time step.

7.5.3 The Thompson Sampling Policy

The drawback of greedily playing the arm with the largest q_i is that it only exploits and does not explore. In fact, such a policy is not really using the meaning of q correctly. In the above example, q_2 was equal to $1/3$. This meant that there was a non-trivial chance that arm 2 was the best arm. Any reasonable policy therefore needs to explore arm 2. However, the policy also needs to respect the fact that a a priori, arm 1 is more likely to be the best arm in hindsight.

At this point, the Thompson Sampling policy will seem natural – play arm i with probability equal to q_i. In the above example, it plays arm 1 with probability $2/3$ and arm 2 with probability $1/3$. It updates the posterior of the played arm based on the observed reward, and repeats. In the long run, this policy is exploring arm 2 in addition to playing arm 1. Such a strategy is termed *probability matching*, since it matches the probability of taking a decision to the probability that this decision would be the best.

We now present the Thompson Sampling policy in Figure 7.3, assuming the arms are Bernoulli. We will generalize it to arbitrary distributions immediately thereafter. In the description of the algorithm, note that the probability the sampling step chooses arm k is exactly equal to q_{kt}, the probability that $\mathcal{D}_{kt}$ is the maximum. The sampling step makes the policy easy to implement in practice.

```
• Input: $\mathcal{D}_{i0} = \text{Beta}(\alpha_{i0}, \beta_{i0})$ denotes the prior distribution for arm $i$.
• For $t = 1, 2, \ldots, T$:
    – Sample $s_{it}$ from the posterior distribution $\mathcal{D}_{it}$.
    – Let $k = \text{argmax}_i s_{it}$.
    – Play arm $k$ and observe reward $r \in \{0, 1\}$.
    – If $r = 1$, set $\mathcal{D}_{kt+1} = \text{Beta}(\alpha_{kt} + 1, \beta_{kt})$; else set $\mathcal{D}_{kt+1} = \text{Beta}(\alpha_{kt}, \beta_{kt} + 1)$.
    – For $i \neq k$, set $\mathcal{D}_{it+1} = \mathcal{D}_{it}$.
```

Figure 7.3: The Thompson Sampling Policy for Beta Priors.

Frequentist View. So far, we have presented Thompson sampling as a Bayesian decision policy, where the prior distributions capture our initial knowledge of the parameters of the arms. This is a different view from UCB1, where we did not assume any knowledge of the reward distributions of the arms and the policy was competing with an optimal policy that knew the parameters of the arms. Let's call this the *frequentist* view. Turns out that Thompson sampling is well-defined here as well. We can run the policy by simply setting the initial priors to Beta$(1, 1)$, that is the Uniform distribution on $[0, 1]$. Now, the role of the prior is not to capture our initial knowledge of the parameters; instead it provides a more refined upper confidence bound on the parameter than the UCB1 policy. Note that the UCB1 bound was obtained by applying Hoeffding's inequality, and captures the high probability interval in which the parameter will lie in. In contrast, the Beta posterior density captures the entire distribution of this parameter and not just its high confidence upper bound. In that sense, it provides a more refined bound than UCB1.

In this frequentist view, it is easy to generalize Thompson sampling to arbitrary reward distributions. We keep the algorithm in Figure 7.3 the same. We start with priors being Beta$(1, 1)$ for each arm. Suppose we observe reward $r \in [0, 1]$ for arm i, then we toss a coin with bias r, and pretend the resulting $0/1$ value is the reward that is used for updating the posterior. In other words, we can convert any reward into binary rewards with the above trick, and use it to update the Beta densities in Figure 7.3. In fact, Thompson Sampling can be defined pretty much for any learning setting and is empirically observed to work well, though bounding its performance becomes progressively trickier.

7.5.4 Optimal Regret Bound

As in the analysis of the UCB1 policy, assume arm i^* has the highest mean reward μ^*. Let $\Delta_i = \mu^* - \mu_i$. Assuming the rewards are $0/1$, we have the following theorem:

Theorem 7.5.1 (Agrawal and Goyal; Kauffman *et al.*)**.** *For any $\epsilon > 0$, Algorithm 7.3* *run with priors Beta*$(1,1)$ *incurs expected regret*

$$\textit{Regret } \leq (1+\epsilon)\sum_{i\neq i^*}\frac{\ln T}{D(\mu^*,\mu_i)}\Delta_i + O\left(\frac{n}{\epsilon^2}\right)$$

where n *is the number of arms, and* $D(\mu^*,\mu_i)$ *is the KL-Divergence between the distributions Bernoulli*$(1,\mu^*)$ *and Bernoulli*$(1,\mu_i)$.

Note that this upper bound *exactly* matches the lower bound of Lai and Robbins in Theorem 7.4.5. In that sense, Thompson sampling is *the optimal policy for stochastic MAB*. For the proof, we would need to bound the probability that a sub-optimal arm is played at any time step. The key idea is to bound it in terms of the probability that the optimal arm is played at that time step. The details are technical and beyond the scope of this chapter.

7.6 Bayesian Analysis of Thompson Sampling

We will present a different type of analysis that gives intuition about why Thompson sampling works well. Any bandit policy has to play a sub-optimal arm (whose parameter value is not the largest) as few times as possible, zooming into the best arm quickly. Consider any policy that starts with the priors $\{\mathcal{D}_{i0}\}$ for the arms i, and a time horizon of T. Suppose the goal is to play the sub-optimal arm as few times as possible, then this defines a valid stochastic optimization problem. Any decision policy Π is simply a mapping from the current posterior densities $\{\mathcal{D}_{it}\}$ to an arm k to play at that time step. The probability this was a play of a sub-optimal arm is simply $1-q_{kt}$, where

$$q_{kt} = \Pr\left[\mathcal{D}_{kt} > \max_{i\neq k}\mathcal{D}_{it}\right]$$

Let X_t denote the random variable that is 1 if the play at step t was for a sub-optimal arm. Then the above argument can be rephrased as:

$$\mathbf{E}\left[X_t \mid \text{Current state } = \{\mathcal{D}_{it}\} \text{ and arm } k \text{ is played }\right] = 1 - q_{kt}$$

The goal is to find a decision policy Π that minimizes the expected number of mistakes, or plays of a sub-optimal arm. In other words, find a mapping Π from the posterior states to an arm to play, so that over a horizon of T steps, we minimize:

$$\mathbf{E}[\text{Mistakes}] = \mathbf{E}\left[\sum_{t=1}^{T} X_t\right]$$

It is instructive to consider the case when $T = 1$. Given the priors $\{\mathcal{D}_i\}$ let q_i denote the probability that arm i has the maximum parameter. Note that $\sum_i q_i = 1$ by definition. Any policy has to choose one arm to play. If arm i is played, then $\mathbf{E}[X_1] = 1 - q_i$. Since the optimal policy minimizes the number of mistakes, it will choose $i^* = \text{argmax}_i q_i$, and the expected number of mistakes is $1 - q_{i^*}$. What does

Thompson sampling do? It plays arm i with probability q_i. The expected number of mistakes made is therefore

$$\mathbf{E}[\text{Mistakes in Thompson Sampling}] = \sum_i q_i(1-q_i)$$

This is because if arm i is played, the expected value of mistakes is $(1-q_i)$, but arm i is played only with probability q_i. The expression then follows by linearity of expectation.

Let us compare the optimum mistake value $\min_i(1-q_i)$ with the Thompson Sampling mistake value $\sum_i q_i(1-q_i)$. We can write the latter as:

$$\begin{aligned}\sum_i q_i(1-q_i) &= \sum_{i \neq i^*} q_i(1-q_i) + q_{i^*}(1-q_{i^*}) \\ &\leq \sum_{i \neq i^*} q_i + q_{i^*}(1-q_{i^*}) \\ &= 1 - q_{i^*} + q_{i^*}(1-q_{i^*}) \\ &\leq 2(1-q_{i^*})\end{aligned}$$

This means that Thompson sampling, while not optimizing the number of mistakes, is a 2-approximation on this metric to the optimal policy Π that starts with the same prior information, and runs on the same time horizon.

We will show that the above result is not specific to $T = 1$. For two arms and *any* time horizon, Thompson sampling is a 2 approximation to the number of mistakes made by the optimal policy that uses the same prior information and runs over the same time horizon.

Theorem 7.6.1. *For $n = 2$ arms with arbitrary initial prior distributions $\{\mathcal{D}_i\}$ and an arbitrary time horizon $T \geq 1$, Thompson Sampling is a 2 approximation to the expected number of mistakes made by the optimal policy $OPT(T, \{\mathcal{D}_i\})$ that runs on the same time horizon T and uses the same initial priors.*

The above theorem is a fairly strong statement. As Theorem 7.4.5 shows, the number of mistakes (or regret) is a small quantity that grows sub-linearly in T, and Thompson sampling is competing successfully with this small value. In fact, we conjecture the above theorem holds for arbitrary number of arms, and arbitrary prior distributions that are independent across arms.

7.6.1 Proof of Theorem 7.6.1

Our proof will not require any specific properties of the prior distributions, and will only use a dynamic programming characterization of OPT in an elementary way. We will use the dynamic program to convert the statement of Theorem 7.6.1 into purely a statement about Thompson Sampling. Subsequently, we will prove this statement by elementary induction over time. The entire proof can be viewed as an exercise in reasoning about decision trees and dynamic programs.

We fix some notation first. In the proof below, we will assume that the prior distribution is over the mean reward. Note that for Bernoulli$(1, p)$ distributions, the Beta prior is over p, which is also the mean reward. Since the goal of any policy is to find the arm with largest mean, a mistake would correspond to playing the arm whose mean reward is not the largest. Therefore, without loss of generality, all we need to worry about is the mean reward of the arm and the prior distribution over it.

Let $\mathcal{D}$ denote the joint prior distribution over the mean rewards of the arms. Denote a generic action and reward at time t by $\sigma_t = (a_t, r_t)$, where $a_t \in \{1, 2, \ldots, n\}$ and r_t is the observed reward from this play. Let $\vec{\sigma_t} = \sigma_1 \sigma_2 \cdots \sigma_{t-1}$ denote a sequence of actions and corresponding rewards till time t. At time t, any decision policy's *state* is encoded by some $\vec{\sigma_t}$. Define

$$q_i(\vec{\sigma}) = \Pr\left[\text{Arm with maximum mean } = i \mid \mathcal{D}, \vec{\sigma}\right]$$

as the probability arm i has the maximum mean reward given the state $\vec{\sigma}$, and the prior $\mathcal{D}$. This probability can be computed by updating the prior $\mathcal{D}$ to the posterior $\mathcal{D}(\vec{\sigma})$ using Bayes' rule, and computing the probability that i has the maximum mean given the posterior $\mathcal{D}(\vec{\sigma})$. Similarly, let $\mathcal{D}_i(\vec{\sigma})$ denote the posterior distribution of the mean reward of arm i given the state $\vec{\sigma}$.

Step 1. Dynamic Program for OPT. We first present the dynamic programming characterization of the optimal decision policy that minimizes the expected number of mistakes. Though this involves some notation, the characterization is an easy exercise in dynamic programming.

Let $OPT[\vec{\sigma}, t]$ denote the number of mistakes of the optimal decision policy conditioned on being in state $\vec{\sigma}$ of size $T - t$, with a horizon of t time steps to go. This policy has the choice of playing one of n arms; if it plays arm i, the mistakes incurred by this play is $1 - q_i(\vec{\sigma})$ and the policy observes a reward r with mean θ_i, where the parameter $\theta_i \sim \mathcal{D}_i(\vec{\sigma})$. For notational convenience, we denote this draw as $r \sim \mathcal{D}_i(\vec{\sigma})$. The optimal policy is now the solution to the following dynamic program.

$$OPT[\vec{\sigma}, t] = \min_{i=1}^{n} \left(1 - q_i(\vec{\sigma}) + \mathbf{E}_{r \sim \mathcal{D}_i(\vec{\sigma})}\left[OPT[(\vec{\sigma} \cdot (i, r)), t - 1]\right]\right) \tag{7.1}$$

The base case when $t = 1$ is simply: $OPT[\vec{\sigma}, 1] = \min_{i=1}^{n}(1 - q_i(\vec{\sigma}))$.

Step 2. Martingale Property of q. The rest of the proof hinges on the fact that q_i for any arm defines a martingale sequence over time. In fact, it is the *only* property of $\{q_i\}$ that we will use in the entire proof. The proof of the property follows almost by definition, and is left as an easy exercise.

Lemma 7.6.2 (Martingale Property). *For all $i, j \in \{1, 2, \ldots, n\}$ and all $\vec{\sigma}$ we have:*

$$q_j(\vec{\sigma}) = \mathbf{E}_{r \sim \mathcal{D}_i(\vec{\sigma})}\left[q_j(\vec{\sigma} \cdot (i, r))\right]$$

This means that if arm i is played, the expected value of any q_j at the next state is equal to its current value before the play. In other words, any q_j evolves as a martingale as plays are made, regardless of which arms the plays correspond to.

Step 3. Reducing to Statement about TS. We need to reason about how the number of mistakes made by Thompson Sampling (TS) relates to OPT. We will now reduce this to reasoning about just TS alone, eliminating the need to further characterize OPT.

Consider the Thompson Sampling (TS) policy. Faced with state $\vec{\sigma}$ and a remaining horizon of $t \geq 1$ steps, the policy plays arm i with probability $q_i(\vec{\sigma})$, and hence we have:

$$TS[\vec{\sigma}, t] = \sum_{i=1}^{n} q_i(\vec{\sigma}) \times \left(1 - q_i(\vec{\sigma}) + \mathbf{E}_{r \sim \mathcal{D}_i(\vec{\sigma})} \left[TS[(\vec{\sigma} \cdot (i, r)), t-1]\right]\right)$$

with the base case being $TS[\vec{\sigma}, 1] = \sum_{i=1}^{n} q_i(\vec{\sigma}) \times (1 - q_i(\vec{\sigma}))$.

Let $TS[i, \vec{\sigma}, t]$ denote the number of mistakes of the policy that plays arm i at the first time step, and subsequently executes Thompson Sampling for the remaining $t-1$ time steps. We have:

$$TS[i, \vec{\sigma}, t] = 1 - q_i(\vec{\sigma}) + \mathbf{E}_{r \sim \mathcal{D}_i(\vec{\sigma})} \left[TS[(\vec{\sigma} \cdot (i, r)), t-1]\right]$$

so that we have: $TS[\vec{\sigma}, t] = \sum_{i=1}^{n} q_i(\vec{\sigma}) TS[i, \vec{\sigma}, t]$.

The next lemma reduces the approximation guarantee to a property of the function TS. The statement holds for any policy, and not just the Thompson sampling policy.

Lemma 7.6.3. *Given a prior $\mathcal{D}$, horizon T, and a policy $\mathcal{P}$ with number of mistakes V, suppose that for all $T \geq t \geq 1$, all $\vec{\sigma}$ (of size $T-t$), and all $1 \leq i \leq n$ we have:*

$$V[\vec{\sigma}, t] \leq V[i, \vec{\sigma}, t] + c(1 - q_i(\vec{\sigma})) \tag{7.2}$$

Suppose further that $V[\vec{\sigma}, 1] \leq (c+1) OPT[\vec{\sigma}, 1]$. Then for all $t \leq T$ and $\vec{\sigma}$ of size $T-t$, we have $V[\vec{\sigma}, t] \leq (c+1) OPT[\vec{\sigma}, t]$.

Proof. We prove this by induction over the remaining horizon t; the base case follows by assumption. Suppose the claim is true for horizon $t-1$ and all $\vec{\sigma}$. Then for horizon t and $\vec{\sigma}$, we have:

$$\begin{aligned}
V[\vec{\sigma}, t] &\leq V[i, \vec{\sigma}, t] + c(1 - q_i(\vec{\sigma})) && \forall i \\
&= (c+1)(1 - q_i(\vec{\sigma})) + \mathbf{E}_{r \sim \mathcal{D}_i(\vec{\sigma})} \left[V[(\vec{\sigma} \cdot (i, r)), t-1]\right] && \forall i \\
&\leq (c+1) \left((1 - q_i(\vec{\sigma})) + \mathbf{E}_{r \sim \mathcal{D}_i(\vec{\sigma})} \left[OPT[(\vec{\sigma} \cdot (i, r)), t-1]\right]\right) && \forall i \\
&\leq (c+1) \min_{i=1}^{n} \left((1 - q_i(\vec{\sigma})) + \mathbf{E}_{r \sim \mathcal{D}_i(\vec{\sigma})} \left[OPT[(\vec{\sigma} \cdot (i, r)), t-1]\right]\right) \\
&= (c+1) OPT[\vec{\sigma}, t]
\end{aligned}$$

Here, the first inequality follows from assumption and the second inequality from the inductive hypothesis. This completes the proof. □

Step 4. Proving Precondition (7.2) for $n = 2$ Arms. For the Thompson Sampling policy, we have shown above that $TS[\vec{\sigma}, 1] \leq 2 OPT[\vec{\sigma}, 1]$. Therefore, to show TS is a 2 approximation, it suffices to establish Precondition (7.2) when $c = 1$. We will show this for $n = 2$ arms via induction over the remaining horizon. This will show that Thompson sampling is a 2 approximation to the number of mistakes for arbitrary priors $\mathcal{D}$ and time horizons T, when there are $n = 2$ arms, completing the proof.

Denote the two arms by $\{a, b\}$. The following lemma presents an equivalent characterization of precondition (7.2). In effect it says that we need to bound the difference in the expected number of mistakes made by Thompson sampling if it plays one arm versus the other at the current step.

Lemma 7.6.4. *For the case of $n = 2$ arms denoted $\{a, b\}$, we have:*

$$\begin{aligned} \forall \vec{\sigma}, t: \qquad & TS[\vec{\sigma}, t] \le TS[x, \vec{\sigma}, t] + 1 - q_x(\vec{\sigma}) \;\; \forall x = \{a, b\} \\ & \iff \;\; |TS[a, \vec{\sigma}, t] - TS[b, \vec{\sigma}, t]| \le 1 \end{aligned}$$

Proof. Fix some $\vec{\sigma}, t$, and omit these from the notation. Suppose $TS \le TS[a] + 1 - q_a$. Expanding $TS = q_a TS[a] + q_b TS[b]$, and observing that $q_a + q_b = 1$, we obtain $TS[b] - TS[a] \le 1$. Conversely, if $TS[a] \le 1 + TS[b]$, then $TS = q_a TS[a] + q_b TS[b] \le TS[b] + 1 - q_b$. Reversing the roles of a and b completes the proof. $\square$

Assume w.l.o.g. that the current state corresponds to $\vec{\sigma} = \phi$; we omit this state in the notation when obvious. Let $\sigma_a(r)$ denote the state if arm a is played and r is observed; let $\sigma_{ab}(r, s)$ denote the state if arm a is played and r observed, followed by b played and s observed. The rest of the proof is by induction over time. Assume precondition (7.2) (and its consequence via Lemma 7.6.4) is true for horizons less than t. Consider playing arm a first. Note that $q_a + q_b = 1$ for all states. We have:

$$\begin{aligned} TS[a, \phi, t] &= 1 - q_a + \mathbf{E}_{r \sim \mathcal{D}_a} \left[TS[\sigma_a(r), t-1]\right. \\ &\le 1 - q_a + \mathbf{E}_{r \sim D_a} \left[1 - q_b(\sigma_a(r)) + TS[b, \sigma_a(r), t-1]\right] \\ &= 1 - q_a + \mathbf{E}_{r \sim D_a} \left[2q_a(\sigma_a(r)) + \mathbf{E}_{s \sim \mathcal{D}_b(\sigma_a(r))} \left[TS\left[\sigma_{ab}(r, s), t-2\right]\right]\right] \\ &= 1 + q_a + \mathbf{E}_{r \sim D_a} \left[\mathbf{E}_{s \sim \mathcal{D}_b(\sigma_a(r))} \left[TS\left[\sigma_{ab}(r, s), t-2\right]\right]\right] \end{aligned}$$

Here, the first inequality follows from the inductive hypothesis applied with $i = b$; the following equality applies because $q_a + q_b = 1$ for all states; and the final equality holds by the Martingale property of q_a (Lemma 7.6.2). Similarly, if arm b is played first (using the obvious change in notation):

$$\begin{aligned} TS[b, \phi, t] &= 1 - q_b + \mathbf{E}_{s \sim \mathcal{D}_b} \left[TS[\sigma_b(s), t-1]\right] \\ &= q_a + \mathbf{E}_{s \sim \mathcal{D}_b} \left[q_a(\sigma_b(s)) TS[a, \sigma_b(s), t-1] + q_b(\sigma_b(s)) TS[b, \sigma_b(s), t-1]\right] \\ &\ge q_a + \mathbf{E}_{s \sim \mathcal{D}_b} \left[TS[a, \sigma_b(s), t-1] - q_b(\sigma_b(s))\right] \\ &= q_a - q_b + \mathbf{E}_{s \sim \mathcal{D}_b} \left[TS[a, \sigma_b(s), t-1]\right] \\ &= q_a - q_b + \mathbf{E}_{s \sim \mathcal{D}_b} \left[1 - q_a(\sigma_a(s)) + \mathbf{E}_{r \sim \mathcal{D}_a(\sigma_b(s))} \left[TS[\sigma_{ba}(s, r), t-2]\right]\right] \\ &= q_a - q_b + q_b + \mathbf{E}_{s \sim \mathcal{D}_b} \left[\mathbf{E}_{r \sim \mathcal{D}_a(\sigma_b(s))} \left[TS[\sigma_{ba}(s, r), t-2]\right]\right] \\ &= q_a + \mathbf{E}_{r \sim D_a} \left[\mathbf{E}_{s \sim \mathcal{D}_b(\sigma_a(r))} \left[TS\left[\sigma_{ab}(r, s), t-2\right]\right]\right] \end{aligned}$$

Here, the first inequality follows by the inductive hypothesis combined with Lemma 7.6.4. The next equality and the penultimate one follow from the martingale property (Lemma 7.6.2), and the final equality follows since the plays of the arms are independent, so that the final states are statistically identical whether arm a is played first and then arm b, or the other way around. This shows $TS[a, \phi, t] - TS[b, \phi, t] \leq 1$. Switching the roles of a and b in the above argument shows that $|TS[a, \phi, t] - TS[b, \phi, t]| \leq 1$. Combining this with Lemma 7.6.4, precondition (7.2) follows. This completes the proof of Theorem 7.6.1.

We note that the above proof of 2 approximation does not need the prior $\mathcal{D}$ to be a product distribution over $\mathcal{D}_a$ and $\mathcal{D}_b$; it holds for arbitrary priors over the joint distribution of the rewards of the arms. All it needs is that the plays produce $i.i.d.$ rewards from the underlying joint distribution. We conjecture Precondition (7.2) holds for any $n \geq 2$ arms as long as the priors are independent across arms, but this seems tricky to prove.

Chapter 8

The Experts Problem

In this chapter, we will present what is arguably one of the most influential algorithmic ideas in optimization, loosely termed the "online prediction" problem or the "experts problem". Its use pervades both theory and practice, and cuts across various disciplines such as machine learning, economics, and optimization. Given this, it is no surprise that variants of essentially the same algorithm has been rediscovered multiple times; in fact, the observation that these algorithms are all essentially the same has also been made multiple times!

We will present the basic experts problem, a simple algorithm for solving it, and then give several very different applications from stochastic optimization and machine learning. These applications require clever reductions to the experts problem that are interesting in their own right.

The experts problem is very similar to the MAB problem. A policy has a to make a decision every time step, and there are n *experts* that a policy has access to. At each time step t, the policy chooses to go with some expert i. It does this without knowing how wrong this expert will turn out to be. At the end of the time step, the policy incurs a loss depending on how wrong this expert was; let us call this value l_i^t. We assume the policy learns the losses of all experts at the end of the time step. The goal of the policy is to minimize the total loss it accrues over a horizon of T steps.

8.0.1 Warmup: Online Prediction

As a concrete example, suppose each expert forecasts whether an event will happen or not that day. (For instance, the experts could be weather forecasters and the event is whether or not it will rain that day.) On day t, a set experts are correct and the remaining are wrong. At the beginning of the day, each expert makes a prediction, and we pick an expert whose advice to follow that day and make a prediction based on that expert's recommendation. At the end of the day, we get to know whether the event happened or not, that is, we get to know the set of correct and incorrect experts. This repeats for T days. Our goal is to minimize the number of wrong predictions (or mistakes) made.

Suppose there is an expert that is correct all T days; call this expert i^*. Then it is easy to devise a policy that makes at most $\log n$ mistakes, where n is the number of experts. On each day, go with the forecast that the majority of the experts make. Suppose we do this and it was a mistake. This means that more than $1/2$ the experts were wrong. This means the size of the set i^* lies in has shrunk by a factor of at least 2. After $\log n$ mistakes, we can uniquely identify i^*, and we can subsequently follow its predictions every day.

However, life is often not that simple. There are no perfect forecasters, and a majority of them could be consistently wrong. In this case, a reasonable measure of performance is to compare against the expert that makes the fewest mistakes over the T steps. Consider a modification of the "go with the majority" strategy: We maintain a weight w_i^t for each expert i. Initially, the weight of each expert is 1. At each time step t, let S_t denote the set of experts that predict the event will happen and let S_t' denote the remaining experts that predict otherwise. Consider the total weight of experts in S_t and S_t', and choose to predict according to the set with larger weight. At the end of the day, reduce the weights of all experts that predict incorrectly by a factor of $(1-\eta)$ for some constant $\eta \in [0, 1]$. We call this algorithm the *weighted majority* algorithm.

Theorem 8.0.1. *Let L^T denote the total number of wrong predictions made by the weighted majority algorithm, and let L_i^T denote the total number of mistakes made by expert i. Then for all $i \in \{1, 2, \ldots, n\}$, we have:*

$$L^T \leq 2(1+\eta)L_i^T + \frac{2\ln n}{\eta}$$

Proof. Define a potential function $\Phi^t = \sum_{i=1}^n w_i^t$. Then, $W^1 = n$. Let l_i^t be 1 if expert i made a mistake at time t and 0 otherwise. Since we reduce the weight of every incorrect expert by a factor of $(1-\eta)$, it is clear that $w_i^t = w_i^{t-1}(1-\eta)^{l_i^t}$. If we make a mistake, the weighted majority also makes a mistake, which means at least half the total weight of all experts decreases by a factor of $(1-\eta)$. This means that whenever we made a mistake,

$$\Phi^{t+1} \leq \Phi^t \left(\frac{1}{2} + \frac{1-\eta}{2}\right) = \Phi^t \left(1 - \frac{\eta}{2}\right)$$

Multiplying this over time steps that the algorithm makes a mistake, we get:

$$\Phi^{T+1} \leq n\left(1 - \frac{\eta}{2}\right)^{L^T}$$

where L^T is the number of mistakes However, the final weight of each expert i is equal to $(1-\eta)^{L_i^T}$, where L_i^T is the number of mistakes this expert makes. This means

$$\Phi^{T+1} \geq \sum_{i=1}^n (1-\eta)^{L_i^T}$$

Comparing the two expressions above, we get that for any expert i:

$$(1-\eta)^{L_i^T} \leq n\left(1 - \frac{\eta}{2}\right)^{L^T}$$

Taking logs and observing that $-\ln(1-\eta) \le \eta + \eta^2$, the theorem now follows. $\square$

In essence, this is a 2 approximation algorithm if the benchmark is the number of mistakes made by the best expert. The factor 2 in the above proof is best possible for the space of *deterministic strategies*. Surprisingly, we can get arbitrarily close to the best expert if we consider randomized strategies, and that is the focus of the rest of the chapter. Even more surprisingly, the algorithms for the humble prediction problem can be applied widely in contexts that on the face of it, have nothing to do with prediction at all!

8.1 The Experts Problem and Regret Measure

The experts problem significantly generalizes the online prediction problem as follows: First, the experts don't make any predictions, but simply incur an a priori unknown loss if chosen. Secondly, the losses can be more general numbers and not just 0 or 1.

More formally, let $l_i^t \in [0,1]$ denote the loss of expert i at time t. At time t, the policy knows $i_j^{t'}$ for all $t' < t$, and $j \in \{1, 2, \ldots, n\}$. Based on this information, the policy chooses an expert $\sigma(t)$ at step t, and incurs loss $l_{\sigma(t)}^t$ at time t. At the end of the time step, the losses l_i^t of all experts for that time step are revealed to the policy. The total loss of of the policy is

$$\text{Total loss} = \sum_{t=1}^{T} l_{\sigma(t)}^t$$

On the face of it, this seems like a hopeless problem if the loss sequence is completely unpredictable and generated by an adversary. At each step, suppose there is one expert who is "good" and has loss 0, while all other experts are "bad" and have loss 1. Suppose this good expert is a randomly chosen expert every time step. Then any algorithm incurs loss at least $(1 - 1/n)T$ regardless of its strategy. But had we known the loss sequence in advance, there is a way of choosing experts every step that has loss 0.

However, such a comparison is unfair – we are comparing against an optimal solution that knows the future and can do the best thing every time step. This is akin to comparing a stock trading strategy against a hypothetical trader who could use tomorrow's information, which makes no sense. A more fair performance benchmark is to do what we did in the previous chapter: Assume the losses $\vec{l^t}$ are drawn $i.i.d$ from a distribution D every step. A policy that knows D will choose that expert every time step whose expected loss according to D, that is $\mathbf{E}[l_i]$, is minimum. Call this expert i^*; then the optimal omniscient policy incurs total expected loss $T\mathbf{E}[l_{i^*}]$. Our policy does not know the distribution, and the goal as before is to optimize the regret, *i.e.*, the difference in loss between our policy and the optimal policy.

It turns out that the problem is equally well-posed (and somewhat more general) if we now go ahead and remove the stochastic assumption. Suppose instead we assume the loss sequence is adversarially generated. But just as before, we will compare ourselves against an optimal policy that chooses *one* expert for all time steps, albeit with full knowledge of this sequence. If the expert chosen by our policy at step t is $\sigma(t)$, we can write the regret of our policy as:

$$\text{Regret} = \sum_{t=1}^{T} l_{\sigma(t)}^t - \min_{i=1}^{n} \sum_{t=1}^{T} l_i^t$$

The latter term is the loss of the best expert assuming the entire loss sequence is known. The loss of our policy is compared with this benchmark, and the goal is to devise a policy that minimizes regret. In particular, just as in the previous chapter, we seek policies whose regret grows sub-linearly in T, so that the average regret per time step vanishes as T becomes large. (Note that in the online prediction problem, we also compared against the best expert in hindsight.)

Deterministic versus Randomized Policies. It is relatively easy to see that even when the losses take values in $\{0, 1\}$, no deterministic policy can incur low regret. This is unlike online prediction where we presented a deterministic strategy that is roughly at most a factor 2 away from the optimum.

Note that any deterministic policy can be viewed as a mapping from loss vectors in previous steps to a choice of expert for the current step. Given any such algorithm A, the adversary simply sets the loss of the currently chosen expert to 1 and the losses of the remaining experts to 0 at this time step. This forces the algorithm A to incur total loss of T. Since there is only one non-zero loss value every time step, the sum of the losses of all experts over all time steps is exactly T. This means there exists an expert for which $\sum_{t=1}^{T} l_i^t \le T/n$. The regret is therefore $T(1 - 1/n)$, which is as bad as it can get.

In particular, this shows the following policy does not yield low regret: At each step t, let $L_i^t = \sum_{t'=1}^{t-1} l_i^{t'}$ denote the total loss incurred by expert i in the previous time steps. Choose that expert i at time t that minimizes L_i^t. This strategy is called *follow the leader* (FTL). The problem with such a strategy is what is termed *overfitting*: Its prediction of which expert will be good uses past data in the optimal way, which can severely limit its ability to *generalize* to changing input patterns.

In view of this negative result, we will focus on randomized algorithms, where the adversary cannot adapt to the random choices made by the algorithm. In other words, the adversary fixes the loss function for all time steps in advance, and the algorithm's strategy uses information from past time steps as well as randomization. The regret is now in expectation over the randomness introduced by the algorithm.

More formally, a randomized policy is a mapping from the loss vectors at the previous time steps to a distribution $\vec{p^t}$ over the experts, so that expert i is chosen with probability p_i^t. The expected regret can be written as:

$$\text{Regret} = \sum_{t=1}^{T} \sum_{i=1}^{n} p_i^t l_i^t - \min_{i=1}^{n} \sum_{t=1}^{T} l_i^t$$

A different way of interpreting randomization is to say that the space of decisions at any time step is continuous (all probability distributions over experts) as opposed to discrete (all experts). This view will be useful when we generalize the experts problem later in the chapter.

8.2 The Weighted Majority Algorithm

The basic idea behind the algorithm is simple: Do FTL, but instead of using the expert with minimum past loss, use a *soft max* strategy that chooses an expert with probability that depends exponentially on the total loss. A different way of interpreting the algorithm (assuming losses are either 0 or 1) is that it is a randomized version of the algorithm presented above for online prediction. Instead of going with the majority, randomly choose an expert based on its weight. The formal algorithm, assuming the losses lie in $[0, 1]$, is given in Figure 8.1.

- Let $\eta = \min\left\{\sqrt{\frac{\ln n}{T}}, \frac{1}{2}\right\}$, $w_i^1 = 1, p_i^1 = \frac{1}{n}$ for all $i = 1, 2, \ldots, n$.
- Set $w_i^t = w_i^{t-1}(1 - \eta l_i^{t-1})$ for all i.
- Set $p_i^t = w_i^t / (\sum_{j=1}^n w_j^t)$ for all i.
- Choose expert i with probability p_i^t at step t, and observe the loss vector, $\vec{l^t}$.

Figure 8.1: The Randomized Weighted Majority (RWM) Algorithm.

Theorem 8.2.1. *Let L^* denote the optimal loss in hindsight for T steps, and let L_T^{RWM} denote the loss of the RWM algorithm. Then, for $\eta \in [0, 1/2]$ and assuming losses lie in $[0, 1]$, we have:*

$$L_T^{RWM} \le (1 + \eta) L_T^* + \frac{\ln n}{\eta}$$

Setting $\eta = \min\left\{\sqrt{\frac{\ln n}{T}}, \frac{1}{2}\right\}$ this implies regret $O(\sqrt{T \ln n})$.

Proof. Recall that we are assuming $l_t^i \in [0, 1]$ for all i, t. Also recall that $L_i^t = \sum_{t'=1}^{t-1} l_i^{t'}$. Let $W^t = \sum_{j=1}^n w_t^j$. This is the total weight at time t. The proof essentially shows that if RWM incurs a large loss, the weight also decreases by a correspondingly large amount. In other words, the weight acts as a potential function.

Let L^* denote the loss of the best expert in hindsight. Since the weight of an expert is simply the exponentiation of its total loss, the final weight satisfies the following inequalities:

$$\begin{aligned} W_{T+1} &= \sum_{i=1}^n \prod_{t=1}^T (1 - \eta l_i^t) \ge \sum_{i=1}^n (1 - \eta)^{l_i^t} \\ &= \sum_{i=1}^n (1 - \eta)^{L_i^{T+1}} \ge (1 - \eta)^{L^*} \end{aligned}$$

where we have used the inequality $(1-\eta)^x \leq 1 - x\eta$ for $\eta \in [0,1]$ and $x \in [0,1]$.

We now upper bound W_{T+1} in terms of the loss of RWM, and this will complete the proof. Define

$$F^t = \frac{\sum_{i=1}^n l_i^t w_t^i}{W^t} = \mathbf{E}[\text{ Loss of RWM at step } t]$$

By the definition of the algorithm, we have:

$$\begin{aligned} W^{t+1} &= \sum_{i=1}^n w_i^{t+1} = \sum_{i=1}^n w_i^t \left(1 - \eta l_i^t\right) \\ &= \left(\sum_{i=1}^n w_i^t\right)\left(1 - \eta \frac{\sum_{i=1}^n l_i^t w_t^i}{\sum_{i=1}^n w_i^t}\right) \\ &= W^t(1-\eta F^t) \end{aligned}$$

Multiplying the above inequalities, and using $W^1 = n$, we have:

$$W^{T+1} \leq W^1 \prod_{t=1}^T (1-\eta F^t) = n \prod_{t=1}^T (1-\eta F^t)$$

However, as observed above, $W^{T+1} \geq (1-\eta)^{L^*}$. Combining these two identities and taking logs, we have:

$$\begin{aligned} L^* \ln(1-\eta) &\leq \ln n + \sum_{t=1}^T \ln(1-\eta F^t) \\ &\leq \ln n - \sum_{t=1}^T \eta F^t \\ &= \ln n - \eta L_T^{RWM} \end{aligned}$$

where assume $\eta \in [0, 1/2]$ so that we can use the inequality $\ln(1-x) \leq -x$ for $x \in [0, 1/2]$. Again, assuming $\eta \leq 1/2$ so that $\ln(1/(1-\eta)) \leq \eta(1_\eta)$, we finally have:

$$L_T^{RWM} \leq L^* \frac{\ln(1/(1-\eta))}{\eta} + \frac{\ln n}{\eta} \leq (1+\eta)L^* + \frac{\ln n}{\eta}$$

$\square$

Some Generalizations. If the losses lie in $[0,\rho]$, then we can scale them down by ρ so that they lie in $[0,1]$ and then run RWM. In the scaled down world, we have the inequality $L^* \leq \ln n - \eta L_T^{RWM}$. When we scale the losses back up, we have $L^* \leq \rho \ln n - \eta L_T^{RWM}$, so that we finally get the corollary:

Corollary 8.2.2. *When the loss values lie in* $[0,\rho]$, *the RWM algorithm run on losses scaled down to lie in* $[0,1]$ *and with* $\eta \in [0,1/2]$ *achieves the following guarantee:*

$$L_T^{RWM} \leq (1+\eta)L^* + \frac{\rho \ln n}{\eta}$$

Further, if all the losses are negative, so that the *gain* vector at time t is $\vec{g^t}$ and the goal of the policy is to maximize gain, we can run the algorithm in Figure 8.1 using $\vec{l^t} = -\vec{g^t}$, that is, with negative loss vectors. The analysis is similar, and we state the following result without proof.

Theorem 8.2.3 (Gain Version of RWM)**.** *Let G^* denote the optimal gain in hindsight for T steps, and let G_T^{RWM} denote the gain of the RWM algorithm. Then, for $\eta \in [0, 1/2]$ and assuming gains lie in $[0, \rho]$, we have:*

$$G_T^{RWM} \geq (1 - \eta) G_T^* - \frac{\rho \ln n}{\eta}$$

Note that the regret guarantee is fairly strong – we can achieve ηL^* regret for any small enough η, as long as we also pay a fixed loss of $\frac{\ln n}{\eta}$. It wasn't a priori obvious that such a guarantee was achievable! Note also that because all past losses are available to the policy, the regret has the form $O(\sqrt{T \log n})$, which has an exponentially better dependence on n than the stochastic bandit setting, where the regret was $\Omega(\sqrt{nT})$. Note that unlike the stochastic bandit problem, in the experts problem, we do not assume the distribution of losses is independent across the arms. In fact, the regret guarantee can be reasonable even with exponentially many experts!

8.3 Adversarial Multi-armed Bandits

We now consider the experts problem with *bandit feedback*. This means that when expert i is chosen at step t, *only* its loss is revealed to the algorithm. The losses of other experts are hidden. This is a generalization of the stochastic MAB problem where we assumed the losses are generated $i.i.d.$ from underlying distributions. Now we assume the losses are generated by an adversary.

The regret measure is the same as before – we compare against an omniscient algorithm that knows the losses of *all* experts at all time steps, but is restricted to choose one expert for all time. There are n experts and a time horizon of T.

Surprisingly, the regret bound that is achievable in this setting almost matches the worst case regret bound in the stochastic setting. The bound we obtain is $O(\sqrt{nT \log T})$ and is achieved by a somewhat magical algorithm called **Exp 3**. Note that the lower bound of stochastic MAB applies to the adversarial case as well, so that any policy incurs regret at least $\Omega(\sqrt{nT})$. Therefore, **Exp3** is almost optimal.

8.3.1 A Sub-optimal Algorithm

We first present a simpler algorithm that gives a worse regret bound. Bandit problems are somewhat harder with gains than with losses, so we will consider the gain version of the problem.

The basic idea is to reduce bandits to prediction by constructing an *unbiased estimator*. The reduction from bandits to prediction can be done by defining a *fake* gain vector that is 0 for the unobserved arms, and $\frac{g_i^t}{p_i^t}$ for the observed arm i. With this setting, the fake gain is an unbiased estimator, meaning that its expectation is equal to the real gain for every expert. We feed this fake gain to the RWM algorithm. There is a catch though – the fake gain has a much larger numerical range than the real gain if the probability of playing the arm becomes very small. We can control this by making sure each arm is played with at least a certain minimum probability every step. Intuitively, we explore all the arms with a small probability, and this prevents the RWM algorithm from zooming into a sub-optimal arm too quickly.

The final algorithm is presented in Figure 8.2. It uses a parameter $\gamma \in (0,1)$ to control the range of fake gain values. It uses the RWM algorithm from Figure 8.1 as a subroutine. The weights computed by RWM, when normalized, yield the probabilities of playing each expert. These probabilities are modified by γ/n before the adversarial bandit algorithm uses them.

1. Let $p_i^1 = 1$ for all $i = 1, 2, \dots, n$. Let $\rho = n/\gamma$.
2. For $t = 1, 2, \dots, T$ do:
   - Let $q_i^t = (1-\gamma)p_i^t + \frac{\gamma}{n}$.
   - Sample expert i with probability q_i^t and observe its gain g_i^t.
   - Define the unbiased estimator: $\tilde{g}_j^t = 0$ if $j \neq i$, and $\tilde{g}_i^t = \frac{g_i^t}{q_i^t}$.
   - Feed $\left\{ \frac{-\tilde{g}^t}{\rho} \right\}$ to RWM to obtain the new probability vector $\vec{p^{t+1}}$.

Figure 8.2: A Simple Adversarial Bandit Algorithm.

The analysis is quite simple. The crucial observation is that the fake gain $\tilde{g}$ is an unbiased estimator of the real gain regardless of the execution trace of the algorithm thus far.

Claim 8.3.1 (Unbiased estimator). $\mathbf{E}[\tilde{g_j^t}] = g_j^t$ *regardless of the probability vector* $\vec{q^t}$.

Proof. The coin toss that computes the fake gain value is independent of the coin toss that decided which expert to play. This means that regardless of the probability vector $\vec{q^t}$, the expected fake gain of *any* expert is equal to its real gain. ☐

Next note that the fake gains have value at most n/γ, assuming real gains lie in $[0,1]$. This is because $q_i^t \geq \frac{\gamma}{n}$ by definition. Note that this means the algorithm in Figure 8.2 scales down the gains by this factor before passing it to RWM.

We can now use the guarantee of the RWM algorithm from Theorem 8.2.3:

$$\sum_{t=1}^{T} \sum_{i=1}^{n} p_i^t \tilde{g}_i^t \geq (1-\eta) \max_{i=1}^{n} \sum_{t=1}^{T} \tilde{g}_i^t - \frac{n}{\gamma} \frac{\ln n}{\eta}$$

Let i_t denote the expert played by the algorithm at time step t. Note that $q_i^t \geq (1-\gamma)p_i^t$. This means:

$$\sum_{i=1}^{n} p_i^t \tilde{g}_i^t \leq \frac{1}{1-\gamma} \sum_{i=1}^{n} q_i^t \tilde{g}_i^t = \frac{1}{1-\gamma} q_{i_t}^t \frac{g_{i_t}^t}{q_{i_t}^t} = \frac{g_{i_t}^t}{1-\gamma} \tag{8.1}$$

Combining the above two relations, we have:

$$\frac{1}{1-\gamma}\sum_{t=1}^{T} g_{i_t}^t \geq (1-\eta)\max_{i=1}^{n}\sum_{t=1}^{T}\tilde{g}_i^t - \frac{n}{\gamma}\frac{\ln n}{\eta}$$

Taking expectation over the execution of the algorithm, we have:

$$\frac{1}{1-\gamma}\mathbf{E}\left[\sum_{t=1}^{T} g_{i_t}^t\right] \geq (1-\eta)\mathbf{E}\left[\max_{i=1}^{n}\sum_{t=1}^{T}\tilde{g}_i^t\right] - \frac{n}{\gamma}\frac{\ln n}{\eta}$$

The LHS is simply the gain of the algorithm. Note now that expected value of the max of a set of random variables is at least the maximum of their expectations. Using this on the RHS of the above inequality, we have

$$\frac{1}{1-\gamma}\mathbf{E}\left[\sum_{t=1}^{T} g_{i_t}^t\right] \geq (1-\eta)\max_{i=1}^{n}\sum_{t=1}^{T}\mathbf{E}[\tilde{g}_i^t] - \frac{n}{\gamma}\frac{\ln n}{\eta}$$

Since $\tilde{g}$ is an unbiased estimator of g regardless of the execution trace of the algorithm, we have:

$$\frac{1}{1-\gamma}\mathbf{E}\left[\sum_{t=1}^{T} g_{i_t}^t\right] \geq (1-\eta)\max_{i=1}^{n}\sum_{t=1}^{T} g_i^t - \frac{n}{\gamma}\frac{\ln n}{\eta}$$

Using the fact that the gains are bounded in $[0,1]$, we have:

$$\mathbf{E}\left[\sum_{t=1}^{T} g_{i_t}^t\right] \geq \max_{i=1}^{n}\sum_{t=1}^{T} g_i^t - (\eta+\gamma)T - \frac{n(1-\gamma)}{\gamma}\frac{\ln n}{\eta}$$

Suppose we choose $\eta = \gamma$, and further set $\gamma = \left(\frac{n\ln n}{T}\right)^{1/3}$, we obtain a regret bound of $O\left((n\ln n)^{1/3}T^{2/3}\right)$. This yields the following theorem:

Theorem 8.3.2. *When the gains lie in* $[0,1]$*, the regret of the algorithm in Figure 8.2 for the setting* $\eta = \gamma = \left(\frac{n\ln n}{T}\right)^{1/3}$ *is* $O\left((n\ln n)^{1/3}T^{2/3}\right)$.

8.3.2 The Exp3 Algorithm

The regret bound we obtained above is sub-linear in T. However, it has a worse dependence on T than what we promised. The issue is that the update rule is too conservative – we are forced to sample each expert sufficiently frequently to control the range of fake gains; however, the algorithm has to scale down its updates by this factor, making it adapt too slowly to the best expert. Mathematically, the parameters η and γ multiply in the regret bound and this is unavoidable if we use RWM in the fashion described above.

The key trick to speed this up is to update exponentially in the gain, without scaling anything. The exponential function has huge variance, however, since the range of fake gains is bounded, we can approximate it with a quadratic function. The error

```
1. Let $p_i^1 = 1$ and $\tilde{L}_i^0 = 0$ for all $i = 1, 2, \dots, n$..
2. For $t = 1, 2, \dots, T$ do:
    • Let $q_i^t = (1-\gamma)p_i^t + \frac{\gamma}{n}$.
    • Sample expert $i$ with probability $q_i^t$ and observe its gain $g_i^t$.
    • Define the unbiased estimator: $\tilde{g}_j^t = 0$ if $j \neq i$, and $\tilde{g}_i^t = \frac{g_i^t}{q_i^t}$.
    • Set $\tilde{G}_j^t = \tilde{G}_j^{t-1} + \tilde{g}_j^t$ for all $j = 1, 2, \dots, n$.
    • Set $p_j^{t+1} = \frac{e^{\eta \tilde{G}_j^t}}{\sum_{k=1}^n e^{\eta \tilde{G}_k^t}}$ for all $j = 1, 2, \dots, n$.
```

Figure 8.3: The **Exp3** Bandit Algorithm.

because the range of gains is large now goes into the second order approximation of the exponential function instead of into the first order approximation. This means η and γ affect the regret bound separately and not as a product. The final algorithm, that uses exponential functions, is presented in Figure 8.3.

The critical observation is that the exponential function can be upper bounded by its second order approximation:

Claim 8.3.3. *For any $\eta > 0$ and $x \in [0, 1/\eta]$,*

$$e^{\eta x} \leq 1 + \eta x + (e-2)\eta^2 x^2$$

The proof of the above statement is simple calculus and is left as an exercise. The key point is the following: Suppose the fake gains lie in $[0, 1/\eta]$ while the real gains lie in $[0, 1]$. Then the new update rule is aggressive, since it updates by roughly $1 + \eta x$, which is the first order approximation of the exponential function. On the other hand, RWM would have updated by $1 + \eta^2 x$, since it would scale down the update amount by the range of the fake gains. This makes the update rule comparable to that of the full-information experts problem on the real gains which lie in $[0, 1]$. However, all this is at the expense of the quadratic term, that is the second order upper bound on the exponential function. This increases the variance of the update rule; controlling this requires bounding the variance of our unbiased estimator. This step the crux of the argument for any adversarial bandit analysis. The rest is details.

Theorem 8.3.4. *The **Exp3** algorithm with parameter $\eta = \gamma/n$ has expected regret $O(\sqrt{nT \log n})$ when gains are bounded in $[0, 1]$.*

Proof. The potential function will be the log of the total probability mass. In other words, let

$$\Phi^t = \frac{1}{\eta} \log\left(\sum_{i=1}^n e^{\eta \tilde{G}_i^{t-1}}\right)$$

As before, we will both lower and upper bound the change in potential; the upper bound comes from the optimal policy, and the lower bound comes from **Exp3**. The proof follows roughly the same outline as before, and we need some additional details to deal with the exponential function. This is tedious but not difficult.

Let i^* denote the expert with maximum total gain, that is

$$i^* = \text{argmax}_{i=1}^{n} \sum_{t=1}^{T} g_i^t$$

Denote the total actual gain of expert i till step t as G_i^t. We have:

$$\Phi^{T+1} \geq \frac{1}{\eta} \log \left(e^{\eta \tilde{G}_{i^*}^T} \right) = \tilde{G}_{i^*}^T$$

Taking expectations over the randomness of the algorithm and noting that the fake loss at any step is an unbiased estimator of the real loss, we have:

$$\mathbf{E}\left[\Phi^{T+1}\right] \geq \mathbf{E}\left[\tilde{G}_{i^*}^T\right] = G_{i^*}^T$$

Note that $\Phi_1 = \frac{1}{\eta} \log n$. Therefore

$$\mathbf{E}\left[\Phi^{T+1}\right] - \Phi_1 \geq G_{i^*}^T - \frac{1}{\eta} \log n \tag{8.2}$$

We will now upper bound the increase in potential. Note that

$$\Phi^{t+1} - \Phi^t = \frac{1}{\eta} \log \left(\frac{\sum_{i=1}^n e^{\eta \tilde{G}_i^t}}{\sum_{i=1}^n e^{\eta \tilde{G}_i^{t-1}}} \right) = \frac{1}{\eta} \log \left(\frac{\sum_{i=1}^n e^{\eta \tilde{G}_i^{t-1} + \eta \tilde{g}_i^t}}{\sum_{i=1}^n e^{\eta \tilde{G}_i^{t-1}}} \right)$$

Using the definition of p_i^t, the above can be written as:

$$\Phi^{t+1} - \Phi^t = \frac{1}{\eta} \log \left(\sum_{i=1}^n p_i^t e^{\eta \tilde{g}_i^t} \right)$$

As observed in the proof of the simpler algorithm, the fake gains are bounded by $M = \frac{n}{\gamma}$. Since we set $\eta = \frac{n}{\gamma}$, this means the fake gains lie in $[0, 1/\eta]$. We can now replace the exponential function by its second order approximation using Claim 8.3.3.

$$\Phi^{t+1} - \Phi^t \leq \frac{1}{\eta} \log \left(\sum_{i=1}^n p_i^t \left(1 + \eta \tilde{g}_i^t + (e-2)\eta^2 \left(\tilde{g}_i^t\right)^2 \right) \right)$$

Since $\sum_{i=1}^n p_i^t = 1$, the above simplifies to:

$$\Phi^{t+1} - \Phi^t \leq \frac{1}{\eta} \log \left(1 + \sum_{i=1}^n p_i^t \left(\eta \tilde{g}_i^t + (e-2)\eta^2 \left(\tilde{g}_i^t\right)^2 \right) \right)$$

Using $\log(1+x) \le x$ for $x \ge 0$, we have:

$$\Phi^{t+1} - \Phi^t \le \frac{1}{\eta} \sum_{i=1}^{n} p_i^t \left(\eta \tilde{g}_i^t + (e-2)\eta^2 \left(\tilde{g}_i^t\right)^2 \right)$$

Let i_t denote the expert played by the algorithm at step t. Using Equation (8.1), we have:

$$\Phi^{t+1} - \Phi^t \le \frac{g_{i_t}^t}{1-\gamma} + (e-2)\eta \sum_{i=1}^{n} p_i^t \left(\tilde{g}_i^t\right)^2$$

Let $G_{ALG} = \mathbf{E}[\sum_{t=1}^{T} g_{i_t}^t]$ is the expected gain of the algorithm. We now take the expectation over time steps and obtain:

$$\mathbf{E}\left[\Phi^{T+1}\right] - \Phi^1 \le \frac{G_{ALG}}{1-\gamma} + (e-2)\eta \sum_{t=1}^{T} \sum_{i=1}^{n} \mathbf{E}\left[p_i^t \left(\tilde{g}_i^t\right)^2 \right]$$

This is where we bound the variance of the fake gains. Note that regardless of the execution trace of the algorithm, the fake gain is $\tilde{g}_i^t$ is Bernoulli$\left(\frac{g_i^t}{q_i^t}, q_i^t\right)$, with mean equal to the real gain g_i^t. All we need to do now is compute the second moment which is $\frac{(g_i^t)^2}{q_i^t}$. Therefore, the second term on the RHS simplifies, and we get:

$$\mathbf{E}\left[\Phi^{T+1}\right] - \Phi^1 \le \frac{G_{ALG}}{1-\gamma} + (e-2)\eta \sum_{t=1}^{T} \sum_{i=1}^{n} \frac{p_i^t}{q_i^t} \left(g_i^t\right)^2$$

Note that $q_i^t \ge p_i^t(1-\gamma)$, and $g_i^t \le 1$ for all i. Therefore,

$$\mathbf{E}\left[\Phi^{T+1}\right] - \Phi^1 \le \frac{G_{ALG}}{1-\gamma} + \frac{(e-2)\eta}{(1-\gamma)} nT$$

Comparing this with Eq (8.2), we get:

$$G_{ALG} \ge (1-\gamma) G_{i^*}^T - (e-2)\eta n T - \frac{1-\gamma}{\eta} \log n$$

Since the total gain of any expert is at most T, this implies the regret is at most:

$$\text{Regret} \le \left(\gamma + (e-2)\eta n\right) T + \frac{1-\gamma}{\eta} \log n$$

Note that $\eta = \gamma/n$. If we set $\gamma = \sqrt{\frac{n \log n}{(e-1)T}}$, we obtain the desired regret bounds. We omit the simple details. $\square$

Chapter 9

Applications of the Experts Framework

One way to view the experts problem is as a two player game: The strategies for the row player are the experts. The column player is nature that chooses a "world" to materialize each step. The loss of the row player depends on the expert chosen and "world" realized. The row player chooses a mixed strategy, and the column player subsequently reveals the "world". The goal of the row player is to converge to a Nash equilibrium of the game without a priori knowing what the possible worlds are. In this view, the column player (or adversary) myopically chooses the worst possible loss vector given the row player's current strategy. Given an application, we can both design what the experts should be and what the adversary should be – such a mapping to suitable experts and adversaries is often not straightforward, and we focus on such applications below.

A more benign view of the experts problem is the one mentioned before: The loss vectors are drawn $i.i.d.$ from an underlying distribution (that can potentially be correlated across experts), and the policy's goal is to learn this distribution while simultaneously optimizing total loss. In this view, mapping an application to experts is somewhat straightforward, and we defer such applications to when we discuss generalizations of the experts framework.

9.1 Routing Flows in a Network

The experts algorithm is as powerful as linear programming – in fact, it yields a surprisingly simple algorithm for the latter that runs in pseudo-polynomial time to find solutions of desired accuracy. For several special classes of linear programs, the running time is actually polynomial; in fact, some of the fastest algorithms for network flow type problems use this approach.

9.1.1 Problem Statement

We first consider a simple problem motivated by network routing. The algorithm we design ends up being (at a high level) similar to the TCP congestion control algorithm. There is a directed graph $G(V, E)$, where edge e has capacity $c(e)$. There are k pairs of terminals; pair i has source vertex s_i and sink vertex t_i. The corresponding user i wants to send one unit of flow from s_i to t_i. The input to the problem is the graph, the capacities on the edges, and the terminal pairs.

Large Capacity Assumption. We assume each edge capacity $c(e)$ is at least 1, so that in the least, a single terminal pair could have used any edge without incurring congestion. It is possible to circumvent this assumption with a clever modification of the algorithm we present below, but this is beyond the scope of this book.

A feasible solution is a *flow* of value 1 from each s_i to the corresponding t_i. Let p denote a path, and let f_p be a variable denoting the flow value assigned to this path. We use the notation $e \in p$ to denote whether edge e belongs to path p. Let $\mathcal{P}_i$ denote the set of paths between s_i and t_i. Then, a feasible flow satisfies:

$$(C1) \qquad \forall i : \qquad \sum_{p \in \mathcal{P}_i} f_p \geq 1$$

Given a feasible flow f, the congestion $C(f)$ of this flow is given by:

$$C(f) = \max_{e \in E} \frac{\sum_{p | e \in p} f_p}{c(e)}$$

This is the maximum overflow on any edge relative to the edge capacity.

The goal is to decide if there is a feasible flow f with $C(f) \leq 1$. Note that this is a practically well-motivated question: An (s_i, t_i) pair can be thought of as a user on the internet who has to send packets from s_i to t_i. The congestion on a link (or router) is a measure of transmission delay, and we want to keep this at most 1.

Note that the above problem can be solved using linear programming algorithms. We will show how to model this problem using the experts framework. The advantage of this approach is that it yields an algorithm with nice structure. The reduction to experts involves several steps, and it is worthwhile to write these out in gory detail, since exactly the same approach solves pretty much any linear program. At the end, we discuss why the running time can become pseudo-polynomial in general.

9.1.2 The Lagrangian

We can write the congestion as a set of constraints:

$$(C2) \qquad \forall e \in E : \qquad \sum_{p | e \in p} \frac{f_p}{c(e)} \leq 1$$

The goal is to decide the feasibility of the constraints $(C1)$ and $(C2)$. The key idea is to treat each constraint in $(C2)$ as an expert. Suppose we assign some weights $w_e \geq 0$ to each expert $e \in E$. We need to specify the losses that result.

Suppose we multiply the LHS of each constraint in $(C2)$ by its corresponding weight and sum these up. As seen before, this yields the *Lagrangian*, which defines the following auxiliary minimization problem:

$$\begin{aligned}\mathcal{L}(\vec{w}) &= \min_{f \text{ feasible for } (C1)} \left(\sum_e w_e \sum_{p|e\in p} \frac{f_p}{c(e)} \right) \\ &= \min_{f \text{ feasible for } (C1)} \left(\sum_{i=1}^{k} \sum_{p\in\mathcal{P}_i} f_p \left(\sum_{e\in p} \frac{w_e}{c(e)} \right) \right)\end{aligned}$$

The following observation is called *weak duality*.

Lemma 9.1.1 (Weak Duality)**.** *Suppose there exists flow f^* that satisfies constraints $(C1)$ and $(C2)$, then for any $\vec{w} \geq 0$,*

$$\mathcal{L}(\vec{w}) \leq \sum_e w_e$$

Proof. Note that for any flow that satisfies $(C1)$, in particular for f^*, we must have:

$$\mathcal{L}(\vec{w}) \leq \sum_e w_e \sum_{p|e\in p} \frac{f_p^*}{c(e)}$$

Since f^* is also feasible for $(C2)$, we also have:

$$\sum_{p|e\in p} \frac{f_p^*}{c(e)} \leq 1 \qquad \forall e \in E$$

Multiplying the latter inequalities by w_e and adding them up, the lemma follows. $\square$

9.1.3 Generating Gain Vectors

The adversary's role is the following: Given the policy's weights, it minimizes $\mathcal{L}(\vec{w})$ subject to the constraints $(C1)$. This is now an easier problem, since we get a separate minimization problem for each terminal pair i. The problem for terminal pair i can be written as:

$$\text{Minimize} \qquad \sum_{p\in\mathcal{P}_i} f_p \left(\sum_{e\in p} \frac{w_e}{c(e)} \right)$$

subject to

$$\sum_{p\in\mathcal{P}_i} f_p \geq 1; \qquad \text{and} \qquad f_p \geq 0 \;\; \forall p \in \mathcal{P}_i$$

For any p, the quantity $\sum_{e \in p} \frac{w_e}{c(e)}$ is the length of p if each edge e has length $w_e/c(e)$. If we have to allocate one total unit of f_p, the optimal way to do it is to allocate it to the *shortest path* from s_i to t_i according to this length metric. This means the optimal solution to the $\mathcal{L}(\vec{w})$ will set all flow values to 0 except for these paths:

$$\forall i : \qquad f_{p_i} = 1 \text{ for } p_i = \text{argmin}_{p \in \mathcal{P}_i} \left(\sum_{e \in p} \frac{w_e}{c(e)} \right)$$

Call this flow $f(\vec{w})$. Adversary sets the gain of edge e to be the congestion of edge e according to this flow. In other words, the gain vector is given by

$$\text{Gain of edge } e = g(e, \vec{w}) = \frac{\sum_{p | e \in p} f_p(\vec{w})}{c(e)} \tag{9.1}$$

Note that the gains generated by the adversary are a fixed function of the weight vector chosen by the policy. In a sense, the adversary is not really "adversarial", but is trying to help out the algorithm by giving high gain to edges which are congested by the natural flow induced by the algorithm's weight vector. This causes the algorithm to increase the weight assigned to these edges (hence increasing path lengths passing through these edges), which in turn causes the algorithm to push less flow along these edges in future steps, reducing its congestion.

We have now completely specified the experts and how their gain are chosen. We first show that the gains are in a bounded range, since this directly affects our running time.

Claim 9.1.2. $g(e, \vec{w}) \in [0, k]$ *for all edges* e *and all possible weight vectors* $\vec{w}$*, where* k *is the number of terminal pairs.*

Proof. Since there are k terminal pairs, the flow on any edge can be at most k in the *Lagrangian*. We assumed $c(e) \geq 1$, which means the gain is at at most k. □

The above claim is not a mere technicality – the range of the gains (or losses) goes linearly into the running time of the algorithm, so that if the optimal solution to the Lagrangian violates an individual constraint by a large factor, this is the range of the gains. This factor is termed the *width* of the formulation. For the specific problem at hand, the factor of k is not too bad; in general, it could have depended on the input numbers, and clever reformulations are often needed to reduce the width to a manageable number.

9.1.4 The Final Algorithm

The final algorithm is exactly the same as that in Figure 8.1, except that it has to handle the case where constraints $(C1)$ and $(C2)$ have no feasible solution. As before, the algorithm takes a parameter $\eta \in [0, 1/2]$ as input. We will compute T in the analysis below.

Note that since we have gains, we need to convert them to losses by negation; further, since these lie in $[0, k]$, we need to scale them down by factor k. After this transformation, the update rule for the weights is exactly the same as in Figure 8.1.

```
1. Let $w_e^1 = 1$ for all $i = 1, 2, \ldots, n$.
2. For $t = 1, 2, \ldots, T$ do:
    • Let $W^t = \sum_e w_e^t$. Set $w_e^t \leftarrow w_e^t / W^t$, so that $||\vec{w^t}||_1 = 1$.
    • Solve $\mathcal{L}(\vec{w^t})$. Let the optimal flow be $f^t$.
    • If $\mathcal{L}(\vec{w^t}) > \sum_e w_e^t = 1$, then declare *Infeasible* and terminate.
    • Define the gain $g_e^t = \frac{\sum_{p|e \in p} f_p^t}{c(e)}$ as in Eq (9.1).
    • Set $w_e^{t+1} = w_e^t \left(1 + \frac{\eta g_e^t}{k}\right)$ for all $i$.
3. Output $\hat{f} = \frac{1}{T} \sum_{t=1}^T f^t$.
```

Figure 9.1: The RWM algorithm specialized to Congestion Minimization.

9.1.5 Analysis

Let us analyze this procedure. The following claim is just a restatement of Lemma 9.1.1.

Claim 9.1.3. *If the algorithm declares infeasibility at any step, then there is no flow that can satisfy* $(C1)$ *and* $(C2)$.

We therefore focus on the case where the algorithm terminates declaring feasibility. Note that $\hat{f}$ is the average of a set of flows that satisfies $(C1)$, so that it always satisfies $(C1)$. The only thing to show is that it (approximately) satisfies $(C2)$.

There are two key inequalities that will complete the proof. First, the guarantee of the RWM algorithm (Theorem 8.2.3) is the following:

$$\sum_{t=1}^T \sum_e w_e^t \cdot g_e^t \geq (1 - \eta) \max_e \left(\sum_{t=1}^T g_e^t \right) - \frac{k \ln n}{\eta} \tag{9.2}$$

Next note that the gain vector is related to the Lagrangian solution as:

$$\mathcal{L}(\vec{w^t}) = \sum_e w_e^t \cdot g_e^t$$

Since the algorithm never declares infeasibility, we have that for all steps t:

$$\mathcal{L}(\vec{w^t}) = \sum_e w_e^t \cdot g_e^t \leq 1$$

Summing this over T steps, we obtain the second key inequality:

$$T \geq \sum_{t=1}^T \sum_e w_e^t \cdot g_e^t \tag{9.3}$$

Combining Eq (9.2) and (9.3) and dividing by T, we get:

$$\forall e \in E : 1 \geq \frac{1}{T} \sum_{t=1}^{T} \sum_{e} w_e^t \cdot g_e^t \geq (1-\eta) \frac{\sum_{t=1}^{T} g_e^t}{T} - \frac{k \ln n}{\eta T}$$

Rewriting,

$$\forall e \in E : \frac{\sum_{t=1}^{T} g_e^t}{T} \leq \frac{1}{1-\eta} + \frac{k \ln n}{\eta(1-\eta)T}$$

The LHS is simply the congestion on edge e induced by the average flow $\hat{f}$. This follows since $g_e^t = \frac{\sum_{p|e\in p} f_p^t}{c(e)}$, and since $\hat{f} = \frac{1}{T}\sum_{t=1}^{T} f^t$. We therefore have:

$$\forall e \in E : \qquad \frac{\sum_{p|e\in p} \hat{f}_p}{c(e)} \leq \frac{1}{1-\eta} + \frac{k \ln n}{\eta(1-\eta)T}$$

Suppose we want $\hat{f}$ to satisfy each constraint in $(C2)$ to an additive error of δ, then we need

$$1 + \delta \geq \frac{1}{1-\eta} + \frac{k \ln n}{\eta(1-\eta)T}$$

For $\delta \in [0, 1/6]$, we can choose $\eta = \frac{\delta}{3}$, and $T = \frac{6k \ln n}{\delta^2}$. This shows the following:

Theorem 9.1.4. *For any $\delta \in [0,1]$, the algorithm in Figure 9.1 run with $\eta = \delta/3$ and $T = \frac{6k \ln n}{\delta^2}$ either terminates declaring infeasibility or finds a $\hat{f}$ that is feasible for $(C1)$, and satisfies each constraint in $(C2)$ to within an additive factor of δ. Each iteration requires computing k shortest paths assuming non-negative edge lengths.*

9.1.6 Connection to Routing on the Internet

The nice aspect of the above algorithm is that it is *fully distributed*, which means that the behavior of the edges and the terminal pairs can be separately implemented – the edges respond to flow through them by adjusting weights, and terminal pairs respond to weights by adjusting flow along shortest paths. This style of algorithms are suitable for implementation either on parallel computing systems or in a distributed environment.

In fact, routing on the Internet uses roughly this framework. The network links (or routers) set weights based on how congested they are given current traffic through them. Given these weights, protocols such as Open Shortest Path First (OSPF) compute shortest paths through the network. Each end-node that is routing traffic to a destination uses the current shortest path. The RWM algorithm suggests that end-hosts increase traffic *linearly*, meaning that they add η flow every time step, while the weights are set to depend *exponentially* on the congestion the router or link perceives. Additive increase is pretty close to what actually happens in an end-node protocol like TCP.

Of course the details are quite different – a real congestion control protocol will also have to reduce flow if there is too much congestion, deal with online arrival of other end-nodes, and deal with link failures, but at a very high level, the RWM algorithm shows why distributed schemes that separate out the behavior of a link (setting weights) from end-nodes (routing along shortest paths) could possibly yield near-optimal routes in terms of congestion.

9.2 General Recipe for Solving Linear Programs

Everything described above generalizes to solving general linear programs. In fact, we will see another example when we discuss solving LPs over time with $i.i.d$ inputs. As a general recipe, this requires the following ingredients:

- The LP should be separable into a set of hard constraints $\mathcal{C}$ and a set of easy constraints $\mathcal{P}$.
- The weighted linear combination of $\mathcal{C}$ (that is, the Lagrangian), should be efficiently optimizable over $\mathcal{P}$.
- For any weight vector, the optimal Lagrangian solution should violate any constraint in $\mathcal{C}$ by a multiplicative (or additive) factor at most the *width* ρ. This parameter goes either linearly or quadratically into the number of iterations required by RWM.
- Finally, the procedure either correctly declares infeasibility or finds a solution that violates each constraint in $\mathcal{C}$ by an amount δ. The factor $1/\delta^2$ goes into the number of iterations required by RWM.

Packing Problems. More formally, consider the following *packing constraint* setting. There is a set $\mathcal{C}$ of *packing constraints* of the form $A\vec{x} \le 1$ and all entries in A are non-negative. There is another convex set $\mathcal{P}$ of feasibility constraints over $\vec{x}$. We will assume that for all $\vec{x} \in \mathcal{P}$, $A\vec{x} \ge 0$. The goal is to find $\vec{x} \in \mathcal{P}$ that approximately satisfies all constraints in $\mathcal{C}$.

For $\vec{w} \ge 0$, define the Lagrangian as:

$$\mathcal{L}(\vec{w}) = \min_{\vec{x} \in \mathcal{P}} \sum_i w_i \sum_j (A_{ij} x_j)$$

Let $\vec{x}(\vec{w})$ denote the optimal solution to $\mathcal{L}(\vec{w})$. Futher, ;et $\vec{A_i}$ denote the i^{th} row of A.

The *width* ρ is defined as the maximum possible violation in any constraint induced by a Lagrangian optimum:

$$\max_{\vec{w} \ge 0} \max_i \vec{A_i} \cdot \vec{x}(\vec{w})$$

The algorithm can now be described as in Figure 9.2.

In other words, we solve the Lagrangian, and increase the weight of constraints that are most violated, so that the Lagrangian minimization at the next step makes these constraints "more feasible". The guarantees follow using exactly the same proof as above, and we obtain the following theorem:

Theorem 9.2.1. *For any $\delta \in [0,1]$, the algorithm in Figure 9.2 run with $\eta = \delta/3$ and $T = \frac{6\rho \ln n}{\delta^2}$ either terminates declaring infeasibility or finds a $\hat{x} \in \mathcal{P}$, and satisfies $A_i \hat{x} \le 1 + \delta$ for all i. Each iteration requires computing the optimum solution to the Lagrangian formulation $\mathcal{L}(\vec{w})$ for some $\vec{w} \ge 0$.*

```
1. Let $w_e^1 = 1$ for all $i = 1, 2, \ldots, n$.
2. For $t = 1, 2, \ldots, T$ do:
    • Let $W^t = \sum_i w_i^t$. Set $w_i^t \leftarrow w_i^t / W^t$, so that $||\vec{w^t}||_1 = 1$.
    • Solve $\mathcal{L}(\vec{w^t})$. Let the optimal value be $x^t$.
    • If $\mathcal{L}(\vec{x^t}) > \sum_i w_i^t = 1$, then declare Infeasible and terminate.
    • Define the gain $g_i^t = \vec{A_i} \cdot \vec{x^t}$.
    • Set $w_i^{t+1} = w_i^t \left(1 + \frac{\eta g_i^t}{\rho}\right)$ for all $i$.
3. Output $\hat{x} = \frac{1}{T} \sum_{t=1}^T \vec{x^t}$.
```

Figure 9.2: The RWM algorithm for Packing Linear Programs.

General LPs. In a general LP, we assume $\mathcal{C}$ is a set of constraints of the form $A\vec{x} \le \vec{b}$, where we impose no constraints on A and $\vec{b}$. As before, there is a feasible region $\mathcal{P}$ and the goal is to find an $\vec{x} \in \mathcal{P}$ that approximately satisfies all constraints in $\mathcal{C}$.

As before, the Lagrangian, for any $\vec{w} \ge 0$ is defined as:

$$\mathcal{L}(\vec{w}) = \min_{\vec{x} \in \mathcal{P}} \sum_i w_i \sum_j (A_{ij} x_j)$$

Since quantities can become negative, we use the *additive* violation in the constraints to define the gain. For a given $\vec{w}$, let $\vec{x}(\vec{w})$ denote the Lagrangian optimum. Then we define the gain as:

$$g_i(\vec{w}) = (A_i \cdot \vec{x}(\vec{w}) - b_i)$$

This captures how much constraint i is additively violated. Note that the gain can now be either positive or negative.

As before, the width ρ is the maximum possible value any gain can take:

$$\rho = \max_{\vec{w} \ge 0} \max_i |(A_i \cdot \vec{x}(\vec{w}) - b_i)|$$

The RWM algorithm is now exactly the same as in Figure 9.2. The only difference is that at every step t, the algorithm checks if

$$\mathcal{L}(\vec{x^t}) > \vec{w^t} \cdot \vec{b}$$

If so, the algorithm terminates declaring infeasibility, and this is correct by weak duality. The rest of the pseudocode remains the same. Since the gains could become negative, the regret bounds become somewhat worse, and we only state the final theorem without proof:

Theorem 9.2.2. *For any $\delta \in [0, 1]$, the algorithm in Figure 9.2 run with $\eta = \delta/3$ and $T = O\left(\frac{\rho^2 \ln n}{\delta^2}\right)$ either terminates declaring infeasibility or finds a $\hat{x} \in \mathcal{P}$, and* *satisfies $A_i\hat{x} \leq b_i + \delta$ for all i. Each iteration requires computing the optimum solution to the Lagrangian formulation $\mathcal{L}(\vec{w})$ for some $\vec{w} \geq 0$.*

9.3 Boosting

By far the most influential application of the experts framework has been to machine learning. Boosting is a very general paradigm that attempts to develop a "good" classifier from a collection of "ok" classifiers. The basic setting is the following: We have a set U of n labeled examples of the form $\{x_i, \ell_i\}$ for $i = 1, 2, \ldots, n$. Here, x_i is the i^{th} example and $\ell_i \in \{0, 1\}$ is the true label. For instance, in a spam detection application, x_i is an email message, the label is 1 if it is spam and 0 otherwise.

Constructing a classifier that identifies spam with high probability is typically hard. On the other hand, given a distribution over examples, it is relatively straightforward to construct some classifier that works "reasonably well" for that distribution. For instance, examples drawn from this distribution tend to have a certain word in their spam examples, and simply detecting this word yields a classifier that is correct with reasonable probability. But then, if the distribution over examples were different, maybe this becomes a bad classifier.

We capture this by positing a *weak learner* that the following property: Given any distribution $\mathcal{D}$ over U, the weak learner produces a hypothesis $h_{\mathcal{D}}$ for labeling each example. This hypothesis has the following property:

$$\Pr_{i \sim \mathcal{D}} [h_{\mathcal{D}}(x_i) = \ell_i] \geq \frac{1}{2} + \gamma$$

for some $\gamma > 0$.

Boosting is a meta-algorithm that converts such a weak learner into a *strong learner* that produces a composite hypothesis h^* such that for small $\epsilon > 0$,

$$|\{i \,|h^*(x_i) = \ell_i\} \geq n(1 - \epsilon)$$

9.3.1 Defining Experts and their Losses

Each example i defines an expert. A distribution $\mathcal{D}$ specifies a probability value p_i for each example, so that $\sum_{i=1}^n p_i = 1$. If we interpret this as weight, then distributions over the examples simply correspond to weight vectors. As in the network routing application, the goal is to iteratively increase weight of examples that are misclassified by the current hypothesis. This makes the weak learner focus on correctly classifying these examples in future iterations.

Formally, given a weight vector $\vec{p}$, let $h = h_{\vec{p}}$ denote the hypothesis produced by the weak learner. The loss of expert (example) i is given by

$$l_i(\vec{p}) = 1 \quad \text{if} \quad h_{\vec{p}}(x_i) = \ell_i \qquad \text{and} \qquad l_i(\vec{p}) = 0 \quad \text{if} \quad h_{\vec{p}}(x_i) \neq \ell_i$$

Note that having high loss on a correctly classified example reduces its weight in subsequent iterations compared to incorrectly classified examples.

9.3.2 AdaBoost Algorithm

The RWM algorithm applied to this problem is termed *AdaBoost* and is described in Figure 9.3. Recall that γ is the gain over random classification in the weak learner, and ϵ is the desired error probability of the strong learner when run on a uniform distribution over the examples.

1. Let $w_e^1 = 1$ for all $i = 1, 2, \ldots, n$. Let $T = \frac{2}{\gamma^2} \log \frac{1}{\epsilon}$.
2. For $t = 1, 2, \ldots, T$ do:
   - Let $W^t = \sum_i w_i^t$. Set $p_i^t \leftarrow w_i^t / W^t$.
   - The weak learner using $\vec{p^t}$ as distribution produces hypothesis h^t.
   - Define the loss $l_i^t = 1$ if $h^t(x_i) = \ell_i$ and 0 otherwise.
   - Set $w_i^{t+1} = w_i^t\,(1 - \gamma l_i^t)$ for all i.
3. The final hypothesis $\hat{h}(x)$ outputs the majority of $\{h^1(x), h^2(x), \ldots, h^T(x)\}$.

Figure 9.3: The AdaBoost learning algorithm.

The analysis requires modifying the proof of Theorem 8.2.1 for this specific setting. We need to account for the fact that many experts have large loss even in the optimum solution.

Theorem 9.3.1. *The AdaBoost algorithm produces a strong learner.*

Proof. Let B denote the set of examples that are misclassified by the final hypothesis $\hat{h}$. By definition, more than half the hypotheses h^t produced by AdaBoost must have misclassified these examples. Since the loss on misclassification is 0, this means:

$$x \in B \;\Rightarrow\; \sum_t l_i^t \leq \frac{T}{2}$$

Since W^{T+1} is the final weight, and since each correct classification reduces weight of $x \in B$ by $(1-\gamma)$ while a misclassification preserves weight, we have:

$$W^{T+1} \geq \sum_{i \in B} (1-\gamma)^{\sum_{t=1}^T l_i^t} \geq |B| \times (1-\gamma)^{T/2}$$

Let L denote the total loss of AdaBoost. By the property of the weak learner, at each step t, the probability of misclassification is at most $\frac{1}{2} - \gamma$. This means the loss is at least $\frac{1}{2} + \gamma$. Therefore,

$$L \geq T\left(\frac{1}{2} + \gamma\right)$$

The analysis in the proof of Theorem 8.2.1 showed that

$$W^{T+1} \leq W^1\,(1 - \gamma L) \leq n e^{-\gamma L}$$

Combining with the bound on L, we get

$$W^{T+1} \leq ne^{-\gamma T\left(\frac{1}{2}+\gamma\right)}$$

Combining the lower and upper bounds on W^{T+1}, we observe:

$$|B| \cdot (1-\gamma)^{T/2} \leq W^{T+1} \leq ne^{-\gamma T\left(\frac{1}{2}+\gamma\right)}$$

Taking logarithms and rearranging,

$$\ln\left(\frac{|B|}{n}\right) \leq -\frac{\gamma^2 T}{2} = \log\frac{1}{\epsilon}$$

Therefore, $|B| \leq n\epsilon$, which means the final hypothesis $\hat{h}$ misclassifies at most an ϵ fraction of examples. □

Part III

Online Algorithms and Competitive Analysis

Chapter 10

Introduction: Ski Rental and Paging

In an online algorithm, the input arrives one piece at a time, and a decision has to be made for that piece of input before the rest of the input arrives. Once a decision has been made, it can not be reversed. Consider the following problem.

10.1 Ski Rental Problem

A skier arrives at a ski resort. She can rent skis for 1 dollar a day, or buy skis for B dollars. If she buys skis, they can be used as long as needed. The skier is going to stay at the resort for X days, until the season is over. The length of the season X is not known in advance. What is the optimal decision making rule for buying skis?

If X is known, the optimal strategy (OPT) is obvious:

- If $X \leq B$, rent skis for X days.
- If $X > B$, buy skis on the first day.

But an online algorithm has to make the decision to either rent or buy skis every day without knowing X. To measure performance of an online algorithm, we use *competitive ratio*, defined as follows.

$$\text{Competitive Ratio} = \frac{\text{cost of online algorithm}}{\text{cost of the optimal algorithm}},$$

where the optimal algorithm knows the future. (In case of the ski rental problem, the optimal algorithm knows X in advance.)

Consider the following online algorithm ALG: rent skis for B days, then buy skis on day $(B + 1)$.

Theorem 10.1.1. *ALG is 2-competitive.*

Proof. Consider the following 2 cases.

- If $X \leq B$, both ALG and OPT spend X. The competitive ratio is 1.
- If $X > B$, ALG spends $2B$, and OPT spends B. The competitive ratio is 2.

□

We can beat the competitive ratio with a randomized algorithm. The intuition behind the randomized algorithm is that we should be more likely to buy skis as X approaches B. The competitive ratio will be less than 2 because the adversary is oblivious of the specific action that ALG is going to take. We will consider such a randomized algorithm later in the course.

10.2 Paging Problem

Suppose we need to process requests for memory pages. Each requested page must be copied into the cache in order to be processed by the user. The size of the cache is k. The problem is, which page to evict from the cache when the cache is full, and the requested page is not in the cache?

The cost of evicting a page from the cache is 1. The cost of copying a page into the cache is also 1.

One intuitive algorithm is LRU, which evicts the least recently used page. At any step, if eviction is needed, LRU evicts the page that was requested the furthest in the past.

For example, suppose the cache contains pages $\{2, 3, 5\}$, and the requested sequence is

$$2\ \ 3\ \ 2\ \ 3\ \ 5\ \ 3\ \ 5\ \ 3\ \ 6$$

When page 6 is requested, an eviction is necessary. The times of last requests for pages 2, 3, 5 are 3, 8, 7, correspondingly. Thus LRU evicts page 2 as the least recently used page.

A more general idea is used by *marking algorithms*. A marking algorithm divides the request sequence into phases, such that each phase is the longest sequence of requests in which no more than k distinct pages are requested. For example, if $k = 3$ and the request sequence is $(2, 4, 2, 6, 5)$, then the first phase has length 4.

A marking algorithm is described as follows.

- At the beginning of each phase, all pages in the cache are "unmarked".
- If a page in the cache is requested, it is "marked".
- If an eviction is necessary, an unmarked page is evicted.
- If an eviction is necessary and all pages are marked, unmark all pages and start a new phase.

Theorem 10.2.1. *Any marking algorithm for the paging problem is k-competitive.*

Proof. Note that the number of evictions made by a marking algorithm at each phase does not exceed k.

Consider the pages requested in some phase plus the first request of the following phase. In total, there are $k+1$ distinct pages requested, so at least one page must be evicted during these $k+1$ requests. Thus any optimal algorithm has to evict at least one page per phase. Thus the competitive ratio of a marking algorithm is less than or equal to k. □

Theorem 10.2.2. *There is no deterministic algorithm with competitive ratio less than k.*

Proof. Suppose there are $k+1$ different pages in total. If an adversary keeps requesting the page that is not in cache, the cost induced by the deterministic algorithm is the number of requests T. On the other hand, an optimal algorithm that knows the future can evict the page requested furthest in the future. The fault rate of such OPT is 1 in k. Thus no deterministic algorithm can beat the competitive ratio of k. □

10.3 Randomized Paging Algorithms

Consider again marking algorithms for the paging problem. Each phase is the longest sequence of requests in which no more than k distinct pages are requested. In each phase, a marking algorithm operates as follows.

- At the beginning of the phase, all pages are unmarked.
- If a page from the cache is requested, that page is marked
- If a page which is not in the cache is requested, then (a) some unmarked page is evicted from the cache, and (b) a new page is brought into the cache and marked.

If a page fault occurs when all pages in the cache are marked, the phase ends. Note that the marked pages are the pages that were requested more recently than the unmarked pages.

To beat the competitive ratio of k for deterministic algorithms, we can make a randomized algorithm which *evicts a random unmarked page*. The adversary is *oblivious* to the randomness introduced by the algorithm; in particular, we assume the adversary chooses the request sequence in advance. In this case, the adversary can not build a sequence that faults every page.

Theorem 10.3.1. *Random marking is $2H_k$-competitive, where H_k is harmonic sequence.*

$$H_k = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} \approx \log k$$

Proof. We will use the same definition of phases as before. Denote the set of distinct pages requested at phase i but not at phase $i-1$ by New_i. Also, let $M_i = |\text{New}_i|$.

We will claim that OPT $\geq \sum_i \frac{M_i}{2}$. To show that, let s_i be the number of faults for the optimal algorithm OPT in phase i. There are $k + M_i$ distinct requests in phases $i-1$ and i combined. Thus $s_i + s_{i-1} \geq k + M_i$, which implies

$$\text{OPT} = \sum_i s_i \geq \frac{\sum_i M_i}{2}$$

Next, we will show that the randomized marking algorithm makes no more than $M_i \log k$ faults at phase i.

At phase i, consider the first request of a page that was not requested in the previous phase. Denote the j-th old page requested by x_j. Let m_j be the number of distinct pages requested before x_j. In the end, we have $m_j = M_i$.

We claim that

$$\Pr[x_j \text{ not in cache}] \leq \frac{M_i}{k-j+1}$$

The probability that the first requested page is evicted is $\frac{M_i}{k}$. The number of marked pages is equal to $m_j + j - 1$ because there were $(j-1)$ old requests and m_j new requests, all of which required markings.

Thus the number of unmarked pages is $k - (m_j + j - 1)$. The number of old pages unrequested before x_j is requested is $k - (j-1)$.

Note that unmarked pages must be a random subset of marked pages. Since there are $k - (m_j + j - 1)$ unmarked pages and $k - (j-1)$ unrequested old pages, we have

$$\Pr[x_j \notin \text{cache}] = \frac{\binom{k-(j-1)-1}{k-(m_j+j-1)}}{\binom{k-(j-1)}{k-(m_j+j-1)}} \tag{10.1}$$

$$= \frac{m_j}{k-j+1} \tag{10.2}$$

$$\leq \frac{M_i}{k-j+1} \tag{10.3}$$

The expected number of evictions in phase i is

$$\mathbf{E}[\text{number of evictions in phase i}] \leq M_i + \sum_{j=1}^{k} \frac{M_i}{k-j+1} \tag{10.4}$$

$$\leq M_i\left(1 + \frac{1}{k} + \frac{1}{k-1} + \ldots + 1\right) \tag{10.5}$$

$$\leq M_i(\ln k + 1) \tag{10.6}$$

Thus randomized paging is $2 \ln k$ -competitive. $\square$

Chapter 11

Yao's Minimax Theorem

In this lecture, we present Yao's theorem, which gives a technique for lower bounding the competitive ratio of randomized online algorithms (where the adversary is *oblivious* to the randomness). The lower bound follows directly from the *min-max* theorem for two-person zero sum games.

11.1 Yao's Theorem

The two-person zero-sum game model is the following: Player one (row player) can take one of the actions from the set $\{1, \ldots, m\}$, player two (column player) can take an action from $\{1, \ldots, n\}$, and the cost of player one is defined by $M(i, j)$, where i and j are the actions taken by the first and the second players, correspondingly. Since the game is zero-sum, the cost of the second player is $-M(i, j)$. Each player wants to minimize her cost.

The optimal mixed strategy for the row player is the distribution $\mathcal{D}$ over the actions $\{1, \ldots, m\}$ such that

$$\mathcal{D} = \arg\min_{\mathcal{D}} \max_{j} M(\mathcal{D}, j),$$

where

$$M(\mathcal{D}, j) = \sum_{i} P_{\mathcal{D}}(i) M(i, j)$$

We can think of the first player as of a randomized algorithm, and the second player is the adversary.

On the other hand, the optimal mixed strategy $\mathcal{P}$ for the second player is defined by

$$\mathcal{P} = \arg\max_{\mathcal{P}} \min_{i} M(i, \mathcal{P}).$$

Theorem 11.1.1 (Min-max Theorem)**.**

$$\min_{\mathcal{D}} \max_{j} M(\mathcal{D}, j) = \max_{\mathcal{P}} \min_{i} M(i, \mathcal{P})$$

We will prove the above theorem later by using a reduction from the low-regret prediction problem. We will now use it to show Yao's theorem that lower bounds the performance of randomized online algorithms. We can view any randomized algorithm as a probability distribution over deterministic algorithms. We denote deterministic algorithms by ALG$_i$, distributions over algorithms by D, the inputs by σ_j, and distributions over inputs by P.

Theorem 11.1.2 (Yao's Theorem)**.**

$$\min_D \max_{\sigma_j} \frac{\mathbf{E}_{i\in D}[ALG_i(\sigma_j)]}{OPT(\sigma_j)} \geq \max_P \min_i \frac{\mathbf{E}_{\sigma_j \in P}[ALG_i(\sigma_j)]}{\mathbf{E}_{\sigma_j\in P}[OPT(\sigma_j)]}$$

Proof. Consider the min-max theorem.

$$\min_D \max_j \mathbf{E}_D[M(i,j)] = \max_P \min_i \mathbf{E}_P[M(i,j)]$$

Choose $M(i,j) = \text{ALG}_i(\sigma_j) - c\cdot \text{OPT}(\sigma_j)$, where c is chosen so that

$$\min_D \max_{\sigma_j} \mathbf{E}_D[M(i,j)] = \max_P \min_i \mathbf{E}_P[M(i,j)] = 0$$

Since $\min_D \max_{\sigma_j} \mathbf{E}_D[M(i,j)] = 0$, this implies:

$$\max_{\sigma_j} \mathbf{E}_{i\in D}[\text{ALG}_i(\sigma_j) - c\cdot \text{OPT}(\sigma_j)] \geq 0 \qquad \forall D$$

This in turn implies

$$\max_{\sigma_j} \frac{\mathbf{E}_{i\in D}[\text{ALG}_i(\sigma_j)}{\text{OPT}(\sigma_j)]} \geq c \qquad \forall D \qquad \Rightarrow \qquad \min_D \max_{\sigma_j} \frac{\mathbf{E}_{i\in D}[\text{ALG}_i(\sigma_j)]}{\text{OPT}(\sigma_j)} \geq c$$

On the other hand, since $\max_P \min_i \mathbf{E}_P[M(i,j)] = 0$, we have:

$$\min_i \mathbf{E}_{\sigma_j\in P}[\text{ALG}_i(\sigma_j)] - c\cdot \mathbf{E}_{\sigma_j\in P}[\text{OPT}(\sigma_j)] \leq 0 \qquad \forall P$$

This in turn implies:

$$\min_i \frac{\mathbf{E}_{\sigma_j\in P}[\text{ALG}_i(\sigma_j)]}{\mathbf{E}_{\sigma_j\in P}[\text{OPT}(\sigma_j)]} \leq c \qquad \forall P \qquad \Rightarrow \qquad \max_P \min_i \frac{\mathbf{E}_{\sigma_j\in P}[\text{ALG}_i(\sigma_j)]}{\mathbf{E}_{\sigma_j\in P}[\text{OPT}(\sigma_j)]} \leq c$$

□

11.2 Lower Bound for Randomized Paging

Yao's theorem lower bounds the competitive ratio of any randomized online algorithm by the maximum over all distributions over the inputs of the expected competitive ratio of deterministic algorithms for inputs drawn from that distribution. Therefore, in order to show a lower bound for a randomized algorithm, it is sufficient to construct a distribution over inputs for which any deterministic algorithm has bad expected competitive ratio.

As an example, using Yao's principle, we can prove that no randomized paging algorithm beats the competitive ratio of $\log k$. We will show this by constructing a stochastic request sequence on which any deterministic algorithm has expected competitive ratio $\Omega(\log k)$. By *expected competitive ratio*, we mean the ratio of the expected performance of the online algorithm and the expected performance of the optimal algorithm that knows the entire request sequence, where the expectation is over the distribution of possible request sequences.

Theorem 11.2.1. *Any randomized paging algorithm is* $\Omega(\log k)$*-competitive.*

Proof. Construct a sequence with $k + 1$ different pages. At each step, a random page is requested. Consider any deterministic algorithm (that knows the distribution from which the input is presented). Any such algorithm faults with probability $\frac{1}{k+1}$ on each request. Therefore, for a long sequence of n requests, the expected number of page faults is $\frac{n}{k+1}$.

We will now bound the expected number of page faults incurred by the optimal algorithm OPT, that can make decisions with knowledge of the future requests. Note that the expectation is still with respect to the distribution over possible request sequence. The optimal algorithm, on a page fault, evicts the page that will be requested furthest in the future. As before, define a phase as the longest sequence in which k distinct pages are requested. Then OPT faults once per such phase.

By the coupon collector's lemma, since each request is $i.i.d.$, the expected length of a phase is kH_k. Since the lengths of consecutive phases are also $i.i.d.$, by renewal theory, the expected number of phases in a long sequence of length n is $\frac{n}{kH_k}$, and this is precisely the expected number of page faults OPT makes. Therefore, the expected competitive ratio of any deterministic algorithm is $\Omega(\log k)$, and by Yao's theorem, this yields the lower bound on randomized paging. □

Chapter 12

Potential Functions

12.1 List Update Problem

A linked list with n elements is given. At each step, we get a request to find element i in the list. The incurred cost to find element i is equal to the position of i in the list. After element i is found, we are allowed to move i to any earlier position in the list.

Consider the following two intuitive algorithms.

- MTF (Move to front): On each step, move the requested element i to the front of the list.
- FREQ: Sort the elements in the order of decreasing frequency of requests. That is, at each step, if element i has been requested more times than element j so far, then element i must precede element j in the list.

We can show that FREQ is not c-competitive, where c is a constant that does not depend on the number n of elements in the list. Consider the sequence of requests in which the first element is requested n times, then the second element is requested n times, and so on.

Note that FREQ will never rearrange the list in this case. The cost incurred by FREQ is

$$\text{FREQ} = n + 2n + 3n + \ldots + n^2 = \frac{n^2(n+1)}{2}$$

We can see that MTF achieves the optimal cost on this sequence. Note that when element i is requested for the first time, it must be at the same position in the list as it was on the first step. Thus the total cost must be at least $n(n+1)/2 + n(n-1)$, where the first term is the minimum possible cost incurred by the first request for each element, and the second term is the minimum possible cost for the other $n(n-1)$ requests. It is easy to see that MTF achieves this minimum cost.

Thus FREQ incurs cost $O(n^3)$ on this sequence, while the optimal cost is $O(n^2)$, which proves that FREQ is not c-competitive.

Theorem 12.1.1. *MTF is a 2-competitive strategy for the list access problem.*

Proof. The intuition behind the proof is that if MTF induces a high cost at time t, then then the list "looks good" (in terms of some potential function) at time $t+1$.

Denote by ϕ_i the number of pairs of elements x, y in the list such that the relative order of x and y in the lists built by MTF and OPT at time t is different. For example, suppose the lists built by MTF and OPT at time i look as follows.

$$\begin{array}{ccc} 1 & 3 & 2 \\ 3 & 1 & 2 \end{array}$$

Then $\phi_i = 1$, because the only flipped pair is $\{1, 3\}$.

Let s_i be the cost of OPT at step i, and t_i be the cost of MTF at step i. Denote the *amortized cost* of MTF by

$$a_i = t_i + \phi_i - \phi_{i-1}$$

We will claim that $a_i \le 2s_i$. If that is the case, then

$$\sum_i a_i \le 2\sum_i s_i = 2\text{OPT} \tag{12.1}$$

Consider the term $\sum_i a_i$.

$$\sum_i a_i = \sum_i t_i + \sum_i (\phi_i - \phi_{i-1}) \tag{12.2}$$

$$= \text{MTF} + \phi_n - \phi_0 \tag{12.3}$$

$$= \text{MTF} + \phi_n \tag{12.4}$$

Thus $a_i \le 2s_i$ implies MTF $\le$ 2OPT $- \phi_n$, which means that MTF is 2-competitive. To finish the proof, we need to show that $a_i \le 2s_i$.

Suppose key x is requested on step i. Define the following functions.

$s_i(y) = 1$ if $y < x$ in OPT before step i, and 0 otherwise.

$t_i(y) = 1$ if $y < x$ in MTF before step i, and 0 otherwise.

$\phi_i(y) = 1$ if $\{x, y\}$ appear in different order in MTF and OPT at the end of time i.

Also, $s_i(x) = 1$ and $t_i(x) = 1$.

We have

$$s_i = \sum_y s_i(y) \tag{12.5}$$

$$t_i = \sum_y t_i(y) \tag{12.6}$$

$$\phi_{i-1} = \sum_y \phi_{i-1}(y) + C \tag{12.7}$$

$$\phi_i = \sum_y \phi_i(y) + C \tag{12.8}$$

Here C is a term that represents the contribution of elements other than x, and that contribution is the same in both expressions for ϕ_{i-1} and ϕ_i.

We will show that $t_i(y) + \phi_i(y) - \phi_{i-1}(y) \leq 2s_i(y)$ by considering each of the possible cases in the following table.

OPT	MTF	$s_i(y)$	$t_i(y)$	$\phi_{i-1}(y)$	$\phi_i(y)$
$x < y$	$x < y$	0	0	0	0
$x < y$	$y < x$	0	1	1	0
$y < x$	$x < y$	1	0	1	0/1
$y < x$	$y < x$	1	1	0	0/1

Thus we have

$$t_i(y) + \phi_i(y) - \phi_{i-1}(y) \leq 2s_i(y) \tag{12.9}$$

$$\implies t_i + \phi_i - \phi_{i-1} \leq 2s_i \tag{12.10}$$

$$\implies a_i \leq 2s_i \tag{12.11}$$

This finishes the proof that MTF is 2-competitive. □

Next, we show that no deterministic ratio can do better than MTF.

Theorem 12.1.2. *No deterministic algorithm has a competitive ratio better than 2 for the list update problem.*

Proof. Suppose the adversary always requests the last element in the algorithm's list. If the number of elements in the list is n and the number of requests is T, then the cost of the deterministic algorithm is Tn.

On the other hand, consider an algorithm that orders the elements randomly. The expected cost is

$$\sum_x \mathbf{E}[\text{position of } x] = \frac{Tn}{2}$$

Here, the summation is over all requests x. Thus OPT has cost less than or equal to $\frac{Tn}{2}$, which means the deterministic algorithm must have competitive ratio greater than or equal to 2. □

Chapter 13

Primal-Dual Method

13.1 Randomized Ski Rental

Consider again the ski rental problem. We are staying at a rental resort for K days, where K is unknown. We can either rent skis for 1 dollar a day or buy skis for B dollars. As was shown earlier, a deterministic algorithm that rents for B days and then buys skis achieves a competitive ratio of 2. We will show that a randomized algorithm achieves a competitive ratio of $\frac{e}{e-1} \approx 1.63$ by modeling the problem as a covering integer program, and developing a primal-dual algorithm to solve it.

Define the following variables.

- $z_j = 1$ if we rent skis on day j, 0 otherwise.
- $x = 1$ if we buy skis at some point in time.

We can write the following covering integer program whose solution is the optimal offline solution to the ski rental problem (where K is assumed to be known).

$$\text{Minimize } Bx + \sum_{j=1}^{K} z_j \qquad s.t. \qquad \begin{array}{rcll} x + z_j & \geq & 1 & \forall j \\ x, z_j & \in & \{0,1\} & \forall j \end{array}$$

Of course, if K is known, the solution to the above IP is trivial: If $K < B$, set $z_j = 1$ for $j = 1, 2, \ldots, K$; else set $x = 1$. The hard part is the online constraint. In the online problem, K is not known in advance. This corresponds to the constraints $x + z_j \geq 1$ appearing one at a time for increasing values of j. An *online* solution is one where the variable x can only monotonically increase (or more precisely, not decrease) as new constraints arrive. This assumption is natural: The skis are bought at some point and this decision is irrevocable.

General Idea: In order to develop a randomized algorithm, we will take an LP relaxation of this program where the integrality constraints are replaced with $x, z_i \in [0, 1]$. We will then solve the fractional version in an online fashion, and then develop a randomized rounding procedure that achieves the desired competitive ratio in expectation.

The key difference with the optimal primal-dual algorithm for zero sum games (the weighted majority algorithm) is that we are now constrained to keep increasing the variable x. This wasn't an issue in zero sum games, where we scaled down the primal and dual solutions at the end. However, despite this difference, even in the online setting, we will update the dual solution linearly and increase the primal solution in a geometric fashion; the devil of course is in the details of this process.

The Primal-Dual Algorithm: As a first step, we take the dual of the fractional covering program to yield the following fractional packing program (where the variables y_j now arrive online):

$$\text{Maximize} \sum_{j=1}^{K} y_j \qquad s.t. \qquad \begin{array}{rcll} \sum_{j=1}^{K} y_j & \leq & B & \\ y_j & \in & [0,1] & \forall j \end{array}$$

The primal-dual algorithm is as follows (where $c = e - 1$):

Algorithm 1 Primal-dual algorithm for fractional ski rental

```
$x \leftarrow 0$
**while** $K$ increases by 1 **do**
  **if** $x < 1$ **then**
    $z_K \leftarrow 1 - x$      $\Delta x = \frac{1}{B}\left(x + \frac{1}{c}\right)$      $y_K \leftarrow 1$;
  **end if**
**end while**
```

Lemma 13.1.1. *In the above primal-dual algorithm, we have the following properties at every point in time:*

1. *The variables $\{x, z_j, y_j\}$ are feasible for the primal and dual programs.*
2. *The gap between the primal and dual solutions is $e/(e-1)$, showing that the primal solution is a $e/(e-1)$ approximation.*

Proof. At the point where constraint j appears, either $x \geq 1$, or the choice of $z_j = 1 - x$ ensures that $x + z_j \geq 1$. Beyond this point, z_j remains unchanged, while x only increases, so that this constraint stays feasible. Therefore, the primal solution is always feasible.

To show feasibility of the dual, we will show that y_j is set to 1 for $j = 1, 2, \ldots, m$ for some $m \leq B$. Note that if $y_j = 1$, then at the point when constraint j appears, it must have been the case that $x < 1$. For how many values of j can this happen? To see this, note that x increases geometrically, so that x after B days is $\frac{(1+1/B)^B - 1}{c} = 1$ if we set $c = (1 + 1/B)^B - 1 \approx e - 1$. This shows the dual solution is always feasible.

To bound the gap between the primal and dual solutions, at each step when the primal and dual increase (so that $x < 1$), the increase in the dual objective is 1. The increase in the primal objective is:

$$\Delta\text{Primal} = B\Delta x + z_j = B \times \frac{1}{B}\left(x + \frac{1}{c}\right) + 1 - x = \frac{1}{c} + 1 \approx \frac{e}{e-1}$$

Since the primal solution is within a factor of $e/(e-1)$ of a feasible dual solution, this shows that the primal solution is a $e/(e-1)$ approximation to the optimal primal solution. $\square$

Note that we could have increased the primal variable x at a slower rate and hoped to reduce the gap between the primal and dual solution; however this leads to the dual becoming infeasible and requiring to be scaled down, hence increasing the gap. The choice of parameters balances these two conflicting quantities.

Randomized Rounding: The rounding is simple. Choose $\alpha \in [0,1]$ uniformly at random at the outset. If any primal variable is more than α, set it to 1. In particular, buy the skis when the fractional x crosses the randomly chosen α, and rent skis before that. Note that the probability (over the random choice of α) that x (resp. z_j) is set to 1 is precisely x (resp. z_j). By linearity of expectation, the expected value of the primal objective after the rounding is the same as that before rounding. Furthermore, since the fractional x was monotonically non-decreasing with time, the integral x is also monotone. Also, it is easy to check that the integral solution is feasible for all the primal constraints. This shows that the integral solution is feasible for the online problem.

13.2 Other Online Problems

The above algorithm can be generalized (with roughly similar proofs) to several other online problems. We will present two examples omitting the proofs (which are left as exercises).

Budgeted Allocations. Unlike the ski rental problem, which is a covering problem, this is a packing problem. There are n buyers, where buyer i has budget B_i and valuation b_{ij} for item j. The items arrive online, and each must be allocated to a buyer as soon as it arrives. The money that can be extracted from the allocated buyer is the minimum of the remaining budget of the buyer, and the valuation of the item. The goal is to design an online allocation scheme that extracts as much revenue as possible.

This problem generalizes online matching, and the following greedy algorithm is $1/2$ competitive (the proof is an easy exercise): When an item arrives, allocate it to the buyer from which the most money can be extracted. We will now use the primal-dual framework to design a better $1-1/e$ competitive deterministic algorithm.

As before, the first step is to write the LP relaxation. Let $y_{ij} = 1$ if item j is allocated to buyer i, and $y_{ij} = 0$ otherwise. The primal problem is the following (i denotes buyers; j denotes items):

$$\text{Maximize } \sum_{i,j} b_{ij} y_{ij} \qquad s.t. \qquad \begin{array}{rcll} \sum_i^K y_{ij} & \leq & 1 & \forall j \\ \sum_j b_{ij} y_{ij} & \leq & B_i & \forall i \\ y_{ij} & \geq & 0 & \forall i,j \end{array}$$

The dual program is the following:

$$\text{Minimize} \sum_i B_i x_i + \sum_j z_j \qquad s.t. \qquad \begin{array}{rcll} b_{ij}x_i + z_j & \geq & b_{ij} & \forall i,j \\ x_i, z_j & \geq & 0 & \forall i,j \end{array}$$

Let $R_{\max} = \max_{i,j} \frac{b_{ij}}{B_i}$. Let $c = (1 + R_{\max})^{1/R_{\max}}$. The primal-dual algorithm is presented below:

Algorithm 2 Primal-dual algorithm for budgeted allocation

```
$x_i \leftarrow 0$ for all $i$.
while new item $j$ arrives do
  $k = \text{argmax}_i b_{ij}(1 - x_i)$.
  if $x_k \leq 1$ then
    $y_{kj} \leftarrow 1$ and allocate item $j$ to buyer $k$.
    $z_j \leftarrow b_{kj}(1 - x_k)$.
    $\Delta x_k = \frac{b_{kj}}{B_k}\left(x_k + \frac{1}{c-1}\right)$.
  end if
end while
```

Theorem 13.2.1. *The primal-dual algorithm achieves a competitive ratio* $(1-1/c)(1-R_{\max}) \to (1-1/e)$ *as* $R_{\max} \to 0$. *This follows from the following claims that hold at all points in the execution:*

1. *The dual solution* $\{x_i, z_j\}$ *is feasible for all the constraints.*
2. $x_i \geq \frac{1}{c-1}\left(c^{\frac{\sum_j b_{ij}y_{ij}}{B_i}} - 1\right)$.
3. *The primal solution* $\{y_{ij}\}$ *violates the second constraint by at most the value of one bid. Therefore, if this solution is scaled down by* $(1 - R_{\max})$, *it is feasible (the first constraint being trivially satisfied).*
4. *The gap between the primal and dual solutions (or the ratio between their increase in any step) is at most* $c/(c-1)$.

Online Set Cover. In this problem, we are given a collection of m sets, where set i has cost $c_i \geq 1$. The elements arrive online, and the goal is to maintain a set cover in an online fashion. The competitive ratio is the ratio of the cost of the set cover at any point in time and the cost of the optimal set cover for that instance. We present a polynomial time algorithm that is $O(\log m \log n)$ competitive, where n is the number of elements that have arrived so far.

As before, we write the LP relaxation of the set cover problem. This is a fractional covering problem where x_i denotes the fraction to which set i is chosen – in the online problem, this variable must be monotonically non-decreasing with time. The elements j arrive online (and so do the corresponding constraints); let $S(j)$ denote the collection of sets containing j. We denote sets by i and elements by j.

$$\text{Minimize} \sum_i c_i x_i \qquad s.t. \qquad \begin{array}{rcll} \sum_{i\in S(j)} x_i & \geq & 1 & \forall j \\ x_i & \geq & 0 & \forall i \end{array}$$

The dual of this program is the following:

$$\text{Maximize} \sum_j y_j \qquad s.t. \qquad \begin{array}{rcll} \sum_{j|i\in S(j)} y_j & \leq & c_i & \forall i \\ y_j & \geq & 0 & \forall j \end{array}$$

Algorithm 3 Primal-dual algorithm for fractional set cover

```
$x_i \leftarrow 0$ for all $i$.
while new element $j$ arrives do
  while $\sum_{i\in S(j)} x_i < 1$ do
    For all $i \in S(j)$, set $\Delta x_i = \frac{1}{c_i}\left(x_i + \frac{1}{|S(j)|}\right)$.
    $y_j \leftarrow y_j + 1$.
  end while
end while
```

Theorem 13.2.2. *The following sequence of claims hold at the end of the inner* while *loop, and show that the fractional primal solution is $O(\log m)$ competitive:*

1. *The primal solution $\{x_i\}$ is feasible.*
2. *The gap between the primal and dual solutions (or the ratio of their increases) is at most a factor of 2.*
3. $x_i \geq \frac{1}{m}\left(\left(1+\frac{1}{c_i}\right)^{\sum_{j|i\in S(j)} y_j} - 1\right)$
4. *Each dual constraint is violated by a factor of at most $\log(1+3m)$.*

The fractional solution can be rounded as follows: Let s_n denote the minimum of $\log n$ variables drawn uniformly at random in $[0,1]$. If $x_i \geq s$, set $x_i = 1$. As n increases, the value s_n can be maintained in an online fashion quite easily. It is easy to check that $\Pr[\text{set } i \text{ chosen}] = 1 - (1-x_i)^{\log n} \geq \min(1, \frac{x_i}{e}\log n)$. This shows that with high probability, all elements are covered by the integral solution. Furthermore, the rounding guarantees that the integral x_i is monotone if the fractional x_i is monotone. This shows a $O(\log m \log n)$ competitive algorithm.