CPS296.04: Sequential Decision Theory **November 28, 2009**

Lecture 1 : Sensor Placement and Submodularity

Lecturer: Kamesh Munagala *Scribe: Nikhil Gopalkrishnan*

We first consider a stochastic optimization problem where the solution needs to be constructed upfront just based on probabilistic information about the inputs. In the *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.

Problem Formulation

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. Similarly 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.

$$H(S) = \sum_{\mathbf{x}_S} p(\mathbf{x}_S) \log \frac{1}{p(\mathbf{x}_S)}$$

The entropy of S is within 1 unit of the expected number of bits needed to optimally encode the values sensed by locations in S (which is achieved by Huffman encoding). Clearly, the more the number of bits, higher the information content of the locations. Note that if the locations are *i.i.d.* according to common distribution X, then $H(S) = |S|H(X)$: In this case, all nodes are informative, and knowing the value of one location does not help infer the value at another node. On the other hand, if the locations are perfectly correlated, then $H(S) = H(X)$, and this is the minimum possible entropy. In this case, it is sufficient to place a sensor node at only one of these locations.

The sensor placement problem can now be formalized as: *Find the subset S^* of locations such that $|S^*| \leq k$, and $H(S^*)$ is maximum.*

Submodularity

The interesting aspect of this problem is that we can design a generic algorithm that has reasonable performance guarantees for any probability distribution, and only uses some specific properties of the entropy function to achieve its performance. This property is termed *submodularity*.

Definition 1. *A set function $f : 2^{[n]} \rightarrow \Re^+$ 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:*

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

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

Lemma 1. *When the values at each location follow discrete distributions, $H(S)$ is a monotone sub-modular set function.*

Proof. Note that $H(S\cup\{i\})-H(S) = \sum_{\mathbf{x}_S,x_i} p(\mathbf{x}_S,x_i)\log\frac{1}{p(x_i|\mathbf{x}_S)} = H(X_i|S)$. Therefore, submodularity for H can be restated as, for $T\subset S : H(X_i|T)\geq H(X_i|S)$. To show this, we can assume w.l.o.g. that $T=\Phi$. We now need to show that $H(X_i)\leq \mathbf{E}_{\mathbf{x}_S}[H(X_i|\mathbf{x}_S)]$. To see this, observe that $p(x_i) = \mathbf{E}_{\mathbf{x}_S}[p(x_i|\mathbf{x}_S)]$. Now since H is a concave function of $p(\cdot)$, by Jensen's inequality, $H(X_i)\leq \mathbf{E}_{\mathbf{x}_S}[H(X_i|\mathbf{x}_S)]$. □

Submodularity for H is sometimes called the "information never hurts" principle. Having more information, in this case a larger set S, will not increase uncertainty.

Greedy algorithm

We describe a simple greedy algorithm that repeatedly adds the location which maximizes entropy gain. A lower bound on the solution of this greedy strategy is shown using the monotone submodularity property of entropy.

Algorithm 1 Generic Greedy Algorithm

```
S ← φ
for j = 1 to k do
  i ← arg max_{t∉S} H(X_t|S)
  S ← S ∪ i
end for
```

Computing the location i is in general a hard computational problem. We will see how to estimate i in the next lecture.

Lemma 2. *Let S^* be the optimal subset of locations for sensor placement while S is the subset chosen by the greedy algorithm. Then, for any monotone submodular set function f:*

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

Proof. We lower bound the increase in the value of f at any step of the greedy algorithm relative to the difference between the final optimal value and the current value of f.

For $i,j=1,2,\ldots,k$ let y_j be the j^{th} location chosen by some optimal algorithm, and let S_i be the set of locations chosen by the greedy algorithm after the i^{th} step. Note that $S_k=S$ and $\{y_1,y_2,\ldots,y_k\}=S^*$. Also, let $z_{ij}=f(S_{i-1}\cup\{y_1,y_2,\ldots,y_j\})-f(S_{i-1}\cup\{y_1,y_2,\ldots,y_{j-1}\})$. Then for each $i:\sum_{j=1}^{k} z_{ij} = f(S^*\cup S_{i-1})-f(S_{i-1})\geq f(S^*)-f(S_{i-1})$ (monotonicity).

Let $z_{j^*}=\max_{1,\ldots,k} z_j$. We get: $k(f(S_{i-1}\cup\{y_1,y_2,\ldots,y_{j^*}\})-f(S_{i-1}\cup\{y_1,y_2,\ldots,y_{j^*-1}\}))\geq f(S^*)-f(S_{i-1})$. But $f(S_i)-f(S_{i-1})\geq f(S_{i-1}\cup\{y_{j^*}\})-f(S_{i-1})\geq f(S_{i-1}\cup\{y_1,y_2,\ldots,y_{j^*}\})-f(S_{i-1}\cup\{y_1,y_2,\ldots,y_{j^*-1}\})$ where the first inequality follows from the greedy strategy and the second from submodularity. This allows us to set up the recurrence $f(S_i)\geq(1-\frac{1}{k})f(S_{i-1})+\frac{1}{k}f(S^*)$ which gives $f(S)\geq(1-(1-\frac{1}{k})^k)f(S^*)\geq(1-\frac{1}{e})f(S^*)$ □

CPS296.04: Sequential Decision Theory **November 28, 2009**

Lecture 2 : Sensor Placement: estimating conditional entropies

Lecturer: Kamesh Munagala *Scribe: Nikhil Gopalkrishnan*

Previous lecture

We saw a generic greedy algorithm for the sensor placement problem that gave us a $(1 - e^{-1})$ approximation. At each step the algorithm picked the location that would maximize conditional entropy gain from the already chosen set of locations. In this lecture we will see how to find such a location.

Approximating conditional entropies

Finding the greedy location to place a sensor requires computing conditional entropies. Computing these quantities exactly is a hard problem [1] and so we will look to approximate them. Note that:

$$H(X_t|S) = \sum_{\mathbf{x}_S} p(\mathbf{x}_S) \left(\sum_{x_t} p(x_t|\mathbf{x}_S) \log \frac{1}{p(x_t|\mathbf{x}_S)} \right) = E_S \left(\sum_{x_t} p(x_t|\mathbf{x}_S) \log \frac{1}{p(x_t|\mathbf{x}_S)} \right) = E_S[H(X_t|\mathbf{x}_S)]$$

We will use a sampling approach to estimate $H(X_t|S)$.

Algorithm 1 Sampling approach to estimate conditional entropy $H(X_t|S)$

$\hat{H} = 0$
for $i = 1$ to N **do**
  Generate sample $\mathbf{x}_S$
  Infer $p(x_t|\mathbf{x}_S)$ using a graphical model
  $H(X_t|\mathbf{x}_S) = \sum_{x_t} p(x_t|\mathbf{x}_S) \log \frac{1}{p(x_t|\mathbf{x}_S)}$
  $\hat{H} = \hat{H} + \frac{1}{N} H(X_t|\mathbf{x}_S)$
end for
Return $\hat{H}$

Only a polynomial number of samples N suffice to estimate $H(X_t|S)$ accurately using the following sampling algorithm.

Lemma 1. *For all $\epsilon, \delta > 0$:*

$$N = \left\lceil \frac{1}{2} \left(\frac{\log |domain(X_t)|}{\epsilon} \right)^2 \log \frac{2}{\delta} \right\rceil \qquad \Rightarrow \qquad \Pr\left[|H - \hat{H}| > \epsilon \right] \leq \delta$$

Proof. This follows directly from Hoeffding's inequality, which states that for *i.i.d.* samples $X_1, X_2, \ldots, X_N$ from some distribution in the range $[a, b]$ the sample average $\frac{\sum_i X_i}{N} = \frac{S}{N}$ and the true average μ are related as: $\Pr\left[|\frac{S}{N} - \mu| > \epsilon \right] \leq 2e^{\frac{-2N\epsilon^2}{(b-a)^2}}$. Note that for estimating H, we can set $a = 0$ and $b = \log |domain(X_t)|$, since entropy represents the number of bits in the best possible compression, and a naive binary representation of X_t has $\log |domain(X_t)|$ bits. □

Belief propagation in graphical models

In our sampling algorithm, we need to infer conditional probabilities (also called marginal probabilities). The marginal probability of a discrete random variable is the summation of the probability density function over all possible states of all other variables of the system, which grows exponentially even for binary random variables. We need an efficient way of computing these and this is done using *belief propagation* over graphical models. The most commonly used graphical model is the Bayesian network.

A Bayesian network is a directed acyclic graph. Each node is associated with a discrete random variable. A directed edge depicts conditional dependence between the random variables. For eg., a directed edge (b, a) with weight $p(X_a|X_b)$ denotes the conditional probability of the random variable X_a given X_b. A node may have multiple parents in which case its conditional probaility is defined over all its parents as $p(X_a|X_b, X_c, \dots)$. These conditional probabilities define a joint probability distribution over all the random variables.

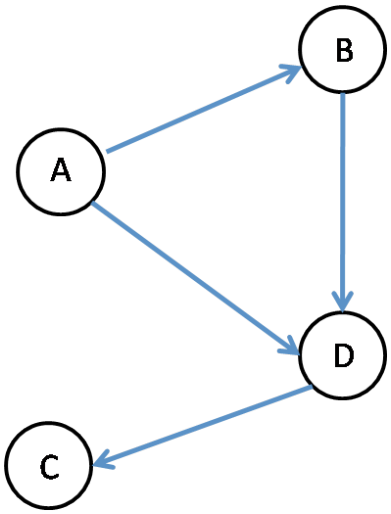

Figure 1: Bayesian network.

Belief propagation is a message passing algorithm for estimating marginal probabilities. The estimated marginal probabilities are called 'beliefs'. The beliefs can be approximated in time linear in number of nodes in the Bayesian network. Polytrees are Bayesian networks with exactly one path between any two nodes. Belief propagation computes exact marginal probabilites in polynomial time in polytrees. For more on belief propagation, see the survey paper [2].

References

[1] A. Krause and C. Guestrin. Optimal nonmyopic value of information in graphical models - efficient algorithms and theoretical limits. In *Proc. of IJCAI*, 2005.

[2] Jonathan Yedidia, William Freeman, and Yair Weiss. Understanding Belief propagation and its generalizations. In *Proc. of IJCAI*, 2001.

CPS296.04: Sequential Decision Theory **November 6, 2009**

Lecture 3 : Influence in social networks

Lecturer: Kamesh Munagala *Scribe: Sayan Bhattacharya*

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 a *social network*.

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 that is guaranteed to converge. 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 standard techniques.

Independent Cascade Model

This model can be viewed as an event-driven process. For every 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 coming out of u. If node v was already active, it remains active irrespective of the outcome at edge (u, v).

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 1. *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}. 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. □

Theorem 1. *The function $f(.)$ is submodular in the independent cascade model.*

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

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

The first equality follows from Lemma 1, 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. □

Linear Threshold Model

Here we impose the additional constraints that sum of the incoming edge weights at any node never exceeds one. 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_i to be the set of active nodes at time step i. At step 0, $A_0 = S$, the initial set of active nodes. At step i, an inactive node v becomes active if sum of the incoming edge weights from already active nodes exceeds its threshold value, that is, if $\sum_{u:u \in A_{i-1}} p_{uv} \geq \theta_v$. Note that once all the threshold values are chosen, the process is completely deterministic.

Consider an alternative random 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.

Lemma 2. *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-hop away from S in the random subgraph τ. The proof is by induction on the number of time steps.

base step Initially $A_0 = B_0 = S$.

induction 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.

Claim 1. *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 and 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. □

Claim 2. *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. $v \notin B_k$ 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 $Pr[v \notin B_k] = 1 - \sum_{u \in B_{k-1}} p_{uv}$.

$v \in B_{k+1} \setminus B_k$ 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 $\sum_{u \in B_{k+1} \setminus B_k} p_{uv}$.

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. □

The induction step holds due Claims 1,2. □

We arrive at the next theorem using Lemma 2. The proof is essentially similar to that of Theorem 1.

Theorem 2. *The function $f(.)$ is submodular in linear threshold model.*

CPS296.04: Sequential Decision Theory **November 28, 2009**

Lecture 4 : Evaluating Conjunctive Queries

Lecturer: Kamesh Munagala *Scribe: Sayan Bhattacharya*

We are given m Boolean predicates (or *filters*) $S_1, \ldots, S_m$. The goal is to evaluate whether $\mathcal{F} = S_1 \wedge S_2 \wedge \cdots \wedge S_m$ returns TRUE or FALSE, and costs c_i to evaluate. Filter S_i evaluates to TRUE with probability p_i. The goal is to determine the optimal order of evaluating the filters so that the expected cost of evaluating the filters is minimized. Note that if a filter is evaluated and returns FALSE, it is immediately known that $\mathcal{F}$ is FALSE, and the evaluation can stop there.

First consider the case where for any two filters S_i, S_j, the probability that S_i evaluates to TRUE is independent of the probability that S_j evaluates to TRUE. In this case, it is easy to check by an exchange argument that the optimal solution orders the filters in increasing order of $\frac{c_i}{1-p_i}$.

We now consider the case where the probabilities that the filters return TRUE are correlated. In this case, we need a tractable method for specifying correlations. One possible method is to specify *scenarios*, where each scenario is an outcome of evaluation of all the filters, and arises with a certain probability or weight. This leads to the following problem formulation.

There is a ground set U of *tuples* (or scenarios). We are given m filters $S_1, \ldots, S_m$ where filter S_i has evaluation cost c_i. If in tuple (or scenario) j, filter S_i returns FALSE, we say that S_i *absorbs* j, or equivalently, $j \in S_i$. In other words, each filter is a subset of U corresponding to the tuples it absorbs (or the scenarios where it returns FALSE). The goal is to compute an ordering of the filters $S_{\sigma(1)} S_{\sigma(2)} \ldots S_{\sigma(m)}$. When a tuple arrives, we go on matching it against the filters one by one till it is absorbed. Thus, it is natural to define the cost of processing a tuple j as the total cost of evaluating filters (w.r.t. the chosen ordering σ) until j is absorbed. We want to find an ordering that minimizes the total (weighted) cost for processing all the tuples. This is also known as the *Min Sum Set Cover Problem*.

In the discussion below, we assume all scenarios have the same probability, so that the weight of all tuples is one. Further, we assume that all filter evaluation costs are one. Therefore, the cost of processing tuple j in ordering σ is simply the rank (w.r.t. σ) of the first filter that absorbs j. A greedy algorithm for this problem is analogous to the greedy set cover algorithm, and is described below. We note that the algorithm chooses the next filter S_i that minimizes $\frac{c_i}{\Pr[S_i = False]}$, where the denominator measures the conditional probability S_i is FALSE conditioned on the previous filters in the ordering returning TRUE (in other words, the probability the tuple is absorbed conditioned on surviving so far).

GREEDY-MSSC

INPUT: A ground set U and subsets $S_1, \ldots, S_m \in U$
OUTPUT: An ordering σ of the subsets given by $S_{\sigma(1)} S_{\sigma(2)} \ldots S_{\sigma(m)}$
  FOR $i = 1$ to m
    $I_i = \{1, \ldots, m\} \setminus \{\sigma(1), \ldots, \sigma(i-1)\}$
    $U_i = U \setminus \cup_{k=1}^{i-1} S_{\sigma(k)}$
    Choose some $t \in \arg\max \{|U_i \cap S_l| : l \in I_i\}$
    Set $\sigma(i) = t$
  END FOR

Figure 1: Greedy algorithm for minimum sum set cover problem

Theorem 1. *The cost of* Greedy-mssc *is at most* 4 *times the optimal cost.*

Proof. Let the ordering returned by Greedy-mssc (resp. optimal ordering) be denoted by $S_{\sigma(1)} \ldots S_{\sigma(m)}$ (resp. $S_{\pi(1)} \ldots S_{\pi(m)}$). Define $G_i = S_{\sigma(i)} \setminus \cup_{k=1}^{i-1} S_{\sigma(k)}$, and $O_i = S_{\pi(i)} \setminus \cup_{k=1}^{i-1} S_{\pi(k)}$. In other words, a tuple j belongs to G_i (resp. O_i) if the subset $S_{\sigma(i)}$ (resp. $S_{\pi(i)}$) is the first one to *absorb* j under ordering σ (resp. π). For all $i, j \in [1, \ldots, m]$, let $b_{ij} = |O_i \cap G_j|$. The cost of the optimal ordering (denoted by *opt*) and that of the greedy ordering (denoted by *greedy*) can now be expressed as follows.

$$opt = \sum_{i=1}^{m} i|O_i| = \sum_{i=1}^{m} i \left(\sum_{j=1}^{m} b_{ij} \right), \text{ and } greedy = \sum_{j=1}^{m} j|G_j| = \sum_{j=1}^{m} j \left(\sum_{i=1}^{m} b_{ij} \right) \tag{1}$$

Greedy-mssc picks the filters in such a way that given $S_{\sigma(1)}, \ldots, S_{\sigma(j-1)}$, the number of new tuples absorbed by $S_{\sigma(j)}$ is at least the number of new tuples absorbed by any $S_{\pi(i)}$. Thus, $\sum_{s=1}^{m} b_{sj} \geq \sum_{r=j}^{m} b_{ir}$ for all i, j, and the optimal objective value of the following (nonlinear) program gives an upper bound on the worst possible ratio of *greedy*/*opt*.

$$\text{Maximize} \quad \frac{\sum_{j=1}^{m} j \left(\sum_{i=1}^{m} b_{ij} \right)}{\sum_{i=1}^{m} i \left(\sum_{j=1}^{m} b_{ij} \right)} \tag{P1}$$

$$\begin{array}{rcll} \sum_{s=1}^{m} b_{sj} & \geq & \sum_{r=j}^{m} b_{ir} & \forall i, j \\ b_{ij} & \geq & 0 & \forall i, j \end{array}$$

The above program is *scale invariant*. Given any feasible solution, scaling down the values of all variables by *same* factor, we get another feasible solution with the same objective value. Thus, we can enforce the denominator of P1 to be equal to one without loosing anything in the optimal objective value. This observation leads us to the *linear* program LP2.

$$\text{Maximize} \quad \sum_{j=1}^{m} j \left(\sum_{i=1}^{m} b_{ij} \right) \tag{LP2}$$

$$\begin{array}{rcll} \sum_{s=1}^{m} b_{sj} & \geq & \sum_{r=j}^{m} b_{ir} & \forall i, j \\ \sum_{i=1}^{m} i \left(\sum_{j=1}^{m} b_{ij} \right) & \leq & 1 & \\ b_{ij} & \geq & 0 & \forall i, j \end{array}$$

Note that in any optimal solution, the constraint $\sum_{i=1}^{m} i \left(\sum_{j=1}^{m} b_{ij} \right) \leq 1$ will always be tight. This ensures that the optimal objective value of LP2 is still an upper bound on *greedy*/*opt*. We next write down the dual of LP2.

$$\text{Minimize} \quad \gamma \tag{LP3}$$

$$\begin{array}{rcll} i\gamma + \sum_{r=1}^{j} \alpha_{ir} & \geq & j + \sum_{s=1}^{n} \alpha_{sj} & \forall i, j \\ \alpha_{ij} & \geq & 0 & \forall i, j \\ \gamma & \geq & 0 & \end{array}$$

We show LP3 has a feasible solution with objective value 4; implying *greedy*/*opt* ≤ 4, that is, Greedy-mssc is a 4-approximation. Set $\alpha_{ij} = 2$ whenever $i < j/2$, and $\alpha_{ij} = 0$ otherwise. Set $\gamma = 4$. This will turn out to be a feasible solution for LP3. Consider any i, j with $i \geq j/2$. Thus, LHS $= i\gamma + \sum_{r=1}^{j} \alpha_{ir} = 4i$, and RHS $= j + \sum_{s=1}^{n} \alpha_{sj} \geq j + 2 \times \frac{j}{2} = 2j$. Since $i \geq j/2$, we

have LHS $\geq$ RHS. Now consider any i, j with $i < j/2$. Thus, LHS $= 4i + 2(j - 2i) = 2j$, and RHS $\leq j + 2 \times \frac{j}{2} = 2j$. We again get LHS $\geq$ RHS. In other words, the optimal objective value of LP3 is lower bounded by 4. This concludes the proof. $\square$

It is shown in [**?**] that obtaining an approximation ratio better than 4 for this problem is not possible under standard complexity theoretic assumptions. This essentially shows that the greedy algorithm is optimal in the worst case.

CPS296.04: Sequential Decision Theory **November 28, 2009**

Lecture 5 : Shared Conjunctive Queries: Fixed Orderings

Lecturer: Kamesh Munagala *Scribe: Sayan Bhattacharya*

In this lecture, we continue the discussion of 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). We assume (unlike in the previous lecture) that these probabilities are independent.

In this case, if there is only one conjunctive query to evaluate, 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, and the goal is to minimize the expected cost of evaluating *all* the queries.

Formally, 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).

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. How good is this space of strategies? Let μ denote the maximum number of queries that a filter can be part of.

Theorem 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, $E[\text{cost}(\text{OPT})] = 1$.

Without any 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,

$$E[\text{cost}(\text{FIXED ORDERING})] \geq \frac{1}{e} \times \frac{n}{2} = \Omega(n) = \Omega(\mu) = \Omega(\mu) \times E[\text{cost}(\text{OPT})]$$

□

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 1. *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

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

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 $E[\text{cost}(\sigma')] < E[\text{cost}(\sigma)]$. Thus, any ordering other than the one returned by FIXED GREEDY cannot be optimal. This completes the proof. □

Theorem 2. *For multiple conjunctive queries,* FIXED GREEDY *is a $O(\mu)$ approximation.*

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

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

The first inequality holds since multiple queries may contain the same filter, the second inequality follows from Lemma 1, and the final inequality holds since a filter may appear in at most μ different queries. □

Stochastic Vertex Cover

Note that the above lower bound assumes each query has at least 3 filters. If each query contains exactly two filters, then the problem indeed admits to a fixed ordering solution that is a constant factor approximation to the optimal adaptive solution. In this case, the problem can be formulated as follows: Construct a undirected graph whose vertices correspond to the filters, and there is an edge (F_i, F_j) if and only if there exists some query containing F_i and F_j. Every vertex F_i has a cost c_i. Whenever we *evaluate* a vertex F_i, it *passes* (or returns TRUE) with some probability p_i, independently of the outcomes at the previously evaluated vertices. We want to (adaptively) evaluate a subset of vertices S of minimum total cost so that for every edge, either one of the vertices has returned FALSE, or both vertices have been evaluated. (This corresponds to resolving the corresponding query to either FALSE or TRUE respectively.)

Note that if $p_i \approx 0$, then any solution will be a vertex cover; this implies that the stochastic vertex cover problem is at least as hard as the vertex cover problem, and the best we can hope for

(given the state of the art in approximation algorithms) is a 2 approximation. We will first present a simple 3 approximation using a fixed ordering, and then present a 2 approximation, again using a fixed ordering.

Let OPT denote the optimal solution. Since every edge is incident to some vertex evaluated by OPT, $E[\text{cost}(\text{OPT})]$ upper bounds the cost of optimal vertex cover. We can utilize this to get a simple algorithm that approximates $E[\text{cost}(\text{OPT})]$ by a factor of 3.

Algorithm: In STEP 1, compute a 2-approximate vertex cover V' where the cost of choosing a vertex F_i is c_i, and evaluate all the corresponding filters. This will resolve some of the edges (or queries). In STEP 2, for each unresolved edge (F_i, F_j) such that $F_i \notin V'$, $F_j \in V'$, evaluate F_i. Note that

$$E[\text{Cost incurred in STEP 1}] \leq 2 \times \text{Cost of optimal vertex cover} \leq 2 \times E[\text{cost}(\text{OPT})]$$

During STEP 2, whenever we are evaluating a vertex F_i, we have an edge (F_i, F_j) such that F_j results in a pass. In this situation, any valid solution must have evaluated F_i. Thus, the expected cost incurred in STEP 2 is upper bounded by $E[\text{cost}(\text{OPT})]$. Combining the costs incurred in STEP 1 and STEP 2, we get that the above algorithm is indeed a 3 approximation to $E[\text{cost}(\text{opt})]$. Note that this algorithm yields a fixed ordering, where the filters in V' are placed first in the ordering in arbitrary order, and the remaining filters are placed next, again in arbitrary order.

Vertex Cover: We will now present a 2-approximation algorithm for the VERTEX COVER problem. In the VERTEX COVER problem, we are given an undirected graph $G = (V, E)$, and every vertex $v \in V$ has a nonnegative cost c_v. We want to find a subset of vertices $V' \subseteq V$ with minimum total cost $\sum_{v \in V'} c_v$ such that every edge $e \in E$ is incident to at least one vertex in V'. Assign a variable $x(v)$ to every vertex $v \in V$, $x(v) = 1$ if v is included in the vertex cover and 0 otherwise. We describe the LP relaxation for VERTEX COVER.

$$\text{Minimize} \quad \sum_{v \in V} c_v x(v) \qquad \text{(VC-PRIMAL)}$$

$$\begin{aligned} x(u) + x(v) &\geq 1 \quad \forall (u, v) \in E \\ x(v) &\geq 0 \quad \forall v \in V \end{aligned}$$

Note that an optimal solution will never set a $x(v)$ to a value strictly greater than 1. Now consider the dual of the above linear program; an edge $e \in \delta(v)$ if e is incident to v.

$$\text{Maximize} \quad \sum_{e \in E} y(e) \qquad \text{(VC-DUAL)}$$

$$\begin{aligned} \textstyle\sum_{e:e \in \delta(v)} y(e) &\leq c_v \quad \forall v \in V \\ y(e) &\geq 0 \quad \forall e \in E \end{aligned}$$

Consider the following algorithm: Initially set $y(e) = 0$ for all $e \in E$. Now increase the $y(e)$ values greedily as far as possible, without violating feasibility of the dual solution. Return the set of vertices S that corresponds to all tight dual constraints.

For every edge $e = (u, v) \in E$, at least one of the constraints corresponding to u and v is tight, otherwise we could have further increased $y(e)$. Thus, S is indeed a valid vertex cover of G. Let OPT denote the cost of optimal vertex cover. Cost of the vertex cover returned by the above

algorithm is given by

$$\begin{aligned}\sum_{v\in S} c_v &= \sum_{v\in S}\sum_{e:e\in\delta(v)} y(e)\\ &\leq \sum_{v\in V}\sum_{e:\in\delta(v)} y(e)\\ &= 2\sum_{e\in E} y(e)\\ &\leq 2\times OPT\end{aligned}$$

The final inequality follows from Weak Duality Theorem. Thus, the algorithm gives a 2-approximation to optimal vertex cover.

Improved Algorithm. The above result yields a 3-approximation for stochastic vertex cover. We now improve it to a 2 approximation by directly encoding the optimal solution as a linear program. For this purpose, define $x(v) = \Pr[\text{OPT evaluates } F_v]$. Let $g_v = 1 - \prod_{u:(u,v)\in E}(1-p_u)$. Consider the following relaxation of OPT.

$$\text{Minimize} \quad \sum_{v\in V} c_v x(v) \qquad \text{(IMPROVED-LP)}$$

$$\begin{aligned} x(u)+x(v) &\geq 1 \quad \forall (u,v)\in E\\ x(v) &\geq g_v \quad \forall v\in V\end{aligned}$$

Lemma 2. *The above LP relaxes OPT.*

Proof. For every edge (or query) $e=(u,v)$, any adaptive solution has to evaluate one of F_u or F_v, so that by union bounds, $x(u)+x(v)\geq 1$. In the event that any neighbor of v has value TRUE, any solution is forced to evaluate F_v. But the probability that some neighbor of v has value TRUE is precisely g_v, so that $x(v)\geq g_v$. □

The first step of the algorithm is to solve the LP. Evaluate the filters corresponding to all vertices v with $x(v)\geq 1/2$. These define a vertex cover, since for any edge $e=(u,v)$, the first constraint in the LP implies either $x(u)\geq 1/2$ or $x(v)\geq 1/2$. After this, for any edge which is still unresolved, evaluate the filter corresponding to the other end-point.

Theorem 3. *The expected cost of the above algorithm is at most $2OPT$.*

Proof. First consider vertices with $x(v)\geq 1/2$. The contribution of such a vertex to the LP objective is at least $c(v)/2$, while the contribution to cost of the algorithm is precisely $c(v)$. Next consider vertices with $x(v)<1/2$. The probability that the algorithm evaluates this filter is precisely g_v, since it is evaluated only if some neighbor u (which must have $x(u)\geq 1/2$) is evaluated and return TRUE. Therefore, the contribution of this vertex to the cost of the algorithm is $c(v)g_v$. Since $x(v)\geq g_v$, the contribution to the LP objective is at least $c(v)g_v$. Therefore, the expected cost of the algorithm is at most $2OPT$. □

CPS296.04: Sequential Decision Theory **November 29, 2009**

Lecture 6 : Shared Conjunctive Queries: Adaptive Orderings

Lecturer: Kamesh Munagala *Scribe: Kamesh Munagala*

In this lecture, we continue the discussion of the shared conjunctive query evaluation problem, and design an adaptive policy based on the set cover algorithm. 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. 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).

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 extend the greedy set cover algorithm to design a $O(\log m)$ approximation for the shared query evaluation problem. Unlike the discussion in the previous lecture, this algorithm does not produce a fixed ordering, but is instead adaptive in nature.

Covering Graph

The key to designing a greedy 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*. If query Q_j contains filter F_i, then there is an edge between (i, j). For an edge $e = (i, j)$, this edge is covered at some step in the algorithm (where a filter is selected and evaluated) if the following two conditions hold:

1. Q_j is unresolved at the beginning of the step.

2. Either F_i is evaluated (and returns TRUE or FALSE), or some F_k (where $k \neq i$ and Q_j contains F_k) is evaluated and returns FALSE.

Note that any adaptive policy needs to evaluate filters in such a fashion that all edges in the covering graph are covered at some point in the execution. To see this, note that if an edge $e = (i, j)$ is not covered, then the corresponding query Q_j is unresolved.

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 $s_i = c_i/\mathbf{E}[\text{Number of edges covered by } F_i]$. 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_i, with probability $(1 - p_i)$, filter F_i evaluated to FALSE and covers all edges (i', j) in G_i such that edge (i, j) exists in G_i. With probability p_i, filter F_i evaluates to TRUE and covers all edges (i, j) in G_i.

Analysis via Dual Prices

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, which we upper-bound in terms of OPT, the cost of the optimal adaptive solution.

Suppose that filter F_i covers edge e_k in the execution of the greedy policy. Then define

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

Note that the price of edge e_k is a random variable that depends the execution of the policy, as well as the outcome of evaluation of F_i. By linearity of expectation, it is easy to check that the cost of the greedy algorithm is given by:

$$GREEDY = \sum_k Price[e_k]$$

Amortizing the Cost: 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 $OC(F_i) = c_i$.

Claim 1. *Whenever the optimal solution evaluates a filter F, charge a cost of $OC(F)$ instead of the actual cost. 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 F_i is independent of this realization, both OPT and $GREEDY$ deterministically decide whether to evaluate F_i. If $GREEDY$ and OPT decide to evaluate F_i, then:

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

On the other hand, if OPT decides to evaluate F_i but not $GREEDY$, then $\mathbf{E}[OC(F_i)] = c_i$ by definition. Therefore, in either case, the expected charge induced by F_i is precisely c_i. □

Bounding the Prices: 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, and also condition on a certain subset Γ of these filters also belonging to OPT.

Claim 2.

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

Proof. Let $\Psi = \{F_1, F_2, \ldots, F_{i-1}, F_i\}$. We have:

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

Since conditioned on the outcome of $F_1, F_2, \ldots, F_{i-1}$, any solution is required to cover the remaining $m-k+1$ edges, it must be the case that:

$$m-k+1 \leq |F_i| \mathbf{1}_{F_i \in OPT} + \sum_{F_l \notin \Psi} n_l \Pr[F_l \in OPT]$$

where n_l is the expected number of edges covered by F_l if it is evaluated immediately after F_{i-1} by the greedy algorithm. (This is the same argument as for set cover.) From the above two inequalities, it follows that:

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

The term $\frac{OC(F_i)}{|F_i|} \mathbf{1}_{F_i \in OPT}$ is precisely $Price[e_k]$ by definition of $OC(F_i)$. Since the greedy choice is optimal, $\min_{F_l \notin \Psi} \frac{c_l}{n_l} \geq Price[e_k]$. This completes the proof of the claim. □

Theorem 1. *The adaptive greedy policy is a $O(\log m)$ approximation.*

Proof. By removing the conditioning in Claim 2, it is easy to see that $\mathbf{E}[Price(e_k)] \leq \frac{OPT}{m-k+1}$. Since $GREEDY = \sum_{k=1}^{m} \mathbf{E}[Price(e_k)]$, we have $GREEDY \leq OPT \times O(\log m)$. □

CPS296.04: Sequential Decision Theory **November 29, 2009**

Lecture 7 : Multi-armed Bandit Problems : Introduction

Lecturer: Kamesh Munagala *Scribe: Harish Chandran*

Introduction

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 than one lever. When each lever is pulled, it provides a numerical reward drawn from a distribution associated with that specific lever. The objective of the gambler is to maximize the sum of collected rewards through iterative pulls. The classical assumption in this problem is that the gambler has no initial knowledge about the levers. The tradeoff that the gambler faces at each trial is between *exploitation* of the lever that has the highest expected payoff and *exploration* to get more information about the expected payoffs of the other levers. This is known as the exploitation vs. exploration tradeoff in reinforcement learning.

Example : 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 $v_1, v_2 \ldots v_n$ based on click throughs. But a user might not click each of the ads $a_1, a_2, \ldots a_n$ 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,

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

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.

Priors and Posteriors

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, you can pull one of the arms resulting in your either getting \$1 or \$0. At some point in time you have played arm A 200 times and observed a reward of \$1 in 100 attempts and you have played arm B 3 times and have observed a reward of \$1 in one of the attempts. 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, we are more confident about our estimate for arm A than arm B. Each time we play an arm, we need to update our estimate on the expected reward we will see for that arm. This notion of updating our belief about the expected reward for each arm after we play it and observe its reward is formalized using priors and posteriors.

For each arm, we can maintain the number of successes and failures encountered in playing that arm thus far as tuple of values along with the expected reward of playing that arm in the next play. For arm A, this tuple is $(100, 100)$ and expected reward is $\frac{1}{2}$ while for arm B, this tuple is $(1, 3)$ and expected reward is $\frac{1}{3}$. These set of values constitute a *prior* belief about that arm. Suppose we play arm B and observe a reward of \$1. Then we need to update the values for that arm to $(2, 2)$ and $\frac{1}{2}$. Similarly, if we observe a reward of \$0 in playing arm B, we need to update the values to $(1, 3)$ and $\frac{1}{4}$. These constitute two different *posterior* beliefs for that arm. These posteriors will become the prior for that arm in the next round of play.

The above is an example of *Beta* priors, which will be discussed in detail in the next section. Beta priors are the most important and widely used class of priors. Another class of priors are the fixed priors where the expected reward of an arm does not change based on the outcome of playing that arm. One can think of this in the following way: if you are tossing a penny, initially you would assume it would turn up heads half the time. Even if you toss it, say 5 times and observe tails, you would attribute to freak chance and would not change you belief that in the next toss it is going to turn up head with probability $\frac{1}{2}$.

What is a correct prior? How do we update it? These are important questions that border on philosophical reasoning. For the purpose of this course, we simply treat the prior as specified in the input. Later on in the course, we will formulate bandit problems without using priors.

Beta Priors

In this section we will examine a class of commonly used priors known as *beta* priors. If the prior distribution of p_i was uniformly at random from $[0, 1]$, and we observe $\alpha - 1$ successes and $\beta - 1$ failures in $\alpha + \beta - 2$ independent and identical Bernoulli trials, then the posterior probability distribution of p_i at this point is given by the beta distribution with parameters α and β. More rigorously, if $p_i \sim \mathbf{UAR}[0, 1]$, and we observe $\alpha - 1$ successes in $\alpha + \beta - 2$ trials, then by Bayes' rule the posterior probability is:

$$Pr(p_i = p | \alpha{-}1 \text{ successes in } \alpha{+}\beta{-}2 \text{ trials}) = \frac{\binom{\alpha-1}{\alpha+\beta-2} p^{\alpha-1}(1-p)^{\beta-1}}{\int_0^1 \binom{\alpha-1}{\alpha+\beta-2} q^{\alpha-1}(1-q)^{\beta-1} dq} = \frac{p^{\alpha-1}(1-p)^{\beta-1}}{\int_0^1 q^{\alpha-1}(1-q)^{\beta-1} dq}$$

which is exactly the probability density function $f(p; \alpha, \beta)$ of $Beta(\alpha, \beta)$ distribution. The expected value of the $Beta(\alpha, \beta)$ is $\frac{\alpha}{\alpha+\beta}$. This is because,

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

If X is a beta random variable with parameters α and β,

$$E[X] = \int_0^1 p \times f(p;\alpha,\beta)dp = \int_0^1 p \times \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}dp = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \times \frac{\Gamma(\alpha+1)\Gamma(\beta)}{\Gamma(\alpha+\beta+1)} = \frac{\alpha}{\alpha+\beta}$$

But if we saw $\alpha - 1$ success in $\alpha + \beta - 2$ trails, one would instinctively expect a reward of $\frac{\alpha}{\alpha+\beta-2}$. This is because in the base case, before we conduct our first trial, we have seen 0 successes in 0 trails, we instinctively assume an expected reward of $\frac{1}{2}$ which is the expectation of $\mathbf{UAR}[0,1]$. Note that $Beta(1,1)$ is the uniform distribution in $[0,1]$. Thus the beta distribution serves as a good prior under the Bayesian assumption. (For more on what is a *good* prior, see below.)

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. How can we generalize this notion? A *single parameter exponential family* is specified by the density function:

$$f_\theta(x) = h(x)e^{\theta x - A(\theta)} \qquad A(\theta) = \log \int_x h(x)e^{\theta x}dx$$

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 $h(x) = 1$ iff $x = 0$ or $x = 1$, we obtain the Bernoulli distribution where $\theta = \log \frac{p}{1-p}$.

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 sample x and prior $r_{a,b}(\theta)$, let us compute the posterior. This is given by Bayes' rule:

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

Therefore, the parameter a behaves as the total 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 sample average. 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 a sample is:

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

If a sample is drawn, the prior is updated, and another sample is drawn, what is the expected value of the second sample? It is easy to check using Bayes' rule that this expected value is the same as that of the first sample. This intuitive fact is termed the *martingale property* of the prior distributions.

Application of the multi armed bandit problem

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 prove beneficial for all the subjects. Allocating $\frac{T}{n}$ subjects to each treatment is not optimal since some of the alternate treatments might not be effective at all while some might be very effective. This sort of partitoning might also anger certain participants: in the later stages of the trials, everyone might want to go for the treatment that looks effective thus far rather than one that does not. This problem can be modeled as a multi armed bandit: each treatment is a lever and we are given T chances to play the bandit. The outcome of a lever depends on the effectiveness of the treatment on some test subject.

Pricing of identical copies of a good

Suppose we are selling some software, creating copies of which are assumed to be of negligible cost. But we do not know how to price the software and are confused among n choices for the price. T customers might arrive in an online manner and request the software. If we offer a price greater than what they are willing to pay, they might not buy it and we get no reward for that play. If they do agree to buy the software, we get a reward equal to the price we asked for. This is a multi armed bandit problem where each of the n different levers has a different reward.

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 with varying negative rewards from some underlying distribution associated with every arm.

CPS296.04: Sequential Decision Theory **November 30, 2009**

Lecture 8 : MDPs and Dynamic Programming

Lecturer: Kamesh Munagala *Scribe: Harish Chandran*

Introduction

In this lecture we shall look at one methodology to formulate policies for the multi armed bandit problem. Let us consider a simple example. We have a two arm bandit with arm A having a prior $Beta(1,1) = \mathbf{UAR}[0,1]$ and arm B having a prior $Beta(101,100)$. Thus the expected reward of arm A is 0.500 and for arm B is $\frac{101}{201} \approx 0.502$. If we are given only one chance to play the bandit, which arm should we choose? Surely, it is arm B as it has a higher expected reward. But if we are given two chances to play the bandit, the choices are no longer apparent. So let us calculate the expected reward in playing each of the arms first.

Suppose we played arm B. We would get an expected reward of $\frac{101}{201}$. Our posterior (or the prior for the next step) would be $Beta(102,100)$ (and expected reward $\frac{102}{202} \geq 0.502$) with probability $\frac{101}{201}$. The posterior would be $Beta(101,101)$ (and expected reward $\frac{100}{202} \leq 0.5$) with probability $\frac{100}{201}$. The former corresponds to a successful play while the later corresponds to failure. At this stage arm A still has an expected reward of 0.5. We would choose arm A if arm B's expected reward is less than 0.5 to get an expected reward of 0.5, else we would play arm B to get an expected reward of $\frac{102}{202}$. Thus the expected reward in playing arm B first is:

$$\mathbf{E}[\text{Reward of playing arm B first}] = \underbrace{\frac{101}{201}}_{\substack{\text{Expected reward}\\ \text{of arm B}}} + \underbrace{\frac{100}{201} \times \frac{1}{2}}_{\substack{\text{Arm B fails}\\ \text{in first play}}} + \underbrace{\frac{101}{201} \times \frac{102}{202}}_{\substack{\text{Arm B succeds}\\ \text{in first play}}} \approx 1.005$$

Similarly, if we play arm A first, we can compute the expected reward as:

$$\mathbf{E}[\text{Reward of playing arm A first}] = \underbrace{\frac{1}{2}}_{\substack{\text{Expected reward}\\ \text{of arm A}}} + \underbrace{\frac{1}{2} \times \frac{101}{201}}_{\substack{\text{Arm A fails}\\ \text{in first play}}} + \underbrace{\frac{1}{2} \times \frac{2}{3}}_{\substack{\text{Arm A succeds}\\ \text{in first play}}} \approx 1.085$$

By this simple calculation, one can see that we should play arm A first if we are given two chances. In some sense, this is the intuitive thing to do, if we don't know anything about an arm (and hence expect a reward of half), we should try it because it could be a very good arm. Thus even from this simple example we can learn a strategy:

Strategy:*Explore the arm with high variance (or uncertainty) initially, especially if the all the arms have approximately the same expected reward.*
It should be noted that in our calculations, we assumed that if we do not play an arm at some point of time, we do not update our beliefs (prior) of that arm in the next step. This seems like a natural assumption and we shall make it often. It should be noted that the multi armed bandit problem has been studied without this assumption in the so called *restless* bandit case.

Bandit Assumption: *If an arm is not played, the state of that arm is unchanged.* It is possible to do such an analysis for simple cases of the multi armed bandit problem, but as the number of arms and the number of chances we get to play grows, these calculations get too laborious. Hence we need to formulate a general method to solve the bandit problem. So let us model the bandit problem as a Markov decisions process (MDP) and try to solve it.

The multi armed bandit as a Markov decision process

Before we model the multi armed bandit problem as a Markov decision process, we first need to define it exactly.

The multi armed bandit problem: *We are given n distinct arms, each with some unknown underlying distribution $f_{\theta_i}(x)$, where θ_i is an unknown parameter (or set of parameters). We also have a prior density $r_i(\theta_i)$ over θ_i. At each time step t, the policy plays some arm i_t and observes a reward, $R(t)$, drawn from $f_{\theta_i}(x)$. This observation also updates the posterior density of the arm by Bayes' rule. The problem is to find a policy that maximizes the expected sum of rewards observed over T steps, i.e.,* $\max \mathbf{E}[\sum_{t=1}^{T} R(t)]$.

We note that the expectation in the above definition is over both the prior distributions over the parameters of the unknown distributions, as well as the rewards drawn *i.i.d.* from the underlying distributions. The priors ensure that each policy has a unique expected value, and hence there is a well-defined notion of an *optimal* policy. This is further formalized below by showing that this problem is a special case of Markov Decision Processes (MDPs).

Now we can define the Markov decision process that models the above problem. What are the states of the MDP? It is the state of the bandit at some time t, which is the collection of states of each arm of the bandit. Each arm i of the bandit at time t is in a state $s_{i,t}$ given by its current posterior. Thus $s_{i,t} = r_i^t(\theta_i)$ where r_i^t is the posterior over θ_i after t plays (and subsequent observation of rewards) of the bandit. Note that these plays are not necessarily just for arm i. The state $\vec{S}_t$ of the bandit at time t is then $\vec{S}_t = \langle s_{1,t}, s_{2,t}, s_{3,t}, \ldots s_{n,t} \rangle$.

Next, we need to define the actions available for each state of the MDP. For each state, we can play one of the n arms and these constitute n distinct actions: $\alpha_1, \alpha_2, \alpha_3, \ldots \alpha_n$. If each of the actions has k different outcomes (which means the underlying distribution has k possible values), then the state is connected to k different states for each action, giving a total of kn new different states. Also, the probability of transition to each of these k states can be computed based on the current prior for the arm. Usually, $k = 2$, i.e., each arm gives out a reward of either 0 or 1 and so in these type of problems, each state of the bandit is connected to $2n$ new different states. Since we have T plays, each of which produces $2n$ new states for each current state, we get a state space of $\Theta((2n)^T)$ different states.

How are the rewards defined? For each state-action pair, we can compute the expected reward we will get for playing that arm based on its current prior. Thus if $\alpha_i = 1$ at time t, then

$$\mathbf{E}[R(t)] = \int_{\theta} r_i^t(\theta) \int_x x f_\theta(x) dx d\theta$$

Let $R_i(s_i)$ denote the expected reward of playing arm i in state s_i.

This completes our description of a MDP that models the bandit problem. One can define each state, compute the expected reward we get for each action in a state along with the transition probabilities to the other states. What is the solution to an MDP? The solution to an MDP is a mapping from states to actions that tells the policy what action to take at each state which maximizes the expected sum of rewards, i.e., come up with a function $\sigma^* : \vec{S} \to \{1, 2, \ldots, n\}$ that maps states of the bandit to actions such that

$$\sigma^* = \arg\max_{\sigma} \mathbf{E}\left[\sum_{t=1}^{T} g(\vec{S}_t, \sigma(\vec{S}_t)\right]$$

where $g(\vec{S}_t, \sigma(\vec{S}_t)$ is the expected reward of playing arm $\sigma(\vec{S}_t)$ when the state at time t is $\vec{S}_t$.

To compute the solution to this MDP, we use a dynamic programming algorithm. First, define $V(\vec{S}, t)$ as the maximum expected reward if the current state of the bandit is $\vec{S}$ and we have $T - t$ plays left. We are interested in maximizing $V(\vec{S}^0, T)$, where $\vec{S}^0$ is the initial state of all the arms. The following dynamic program computes $V(\vec{S}_0, T)$, where $p(s_i, s_i')$ is the probability that if arm i is played in state s_i, it transitions to state s_i':

$$V(\vec{S}, t) = V(\langle s_1, \ldots s_n \rangle, t) = \max_i \left\{ R_i(s_i) + \sum_{\tilde{s}_i} p(s_i, \tilde{s}_i) V(\langle s_1, \ldots, \tilde{s}_i, \ldots s_n \rangle), t+1 \right\}$$

The base case of the dynamic program is trivial: At step $T - 1$, the arm chosen simply maximizes expected reward of the play given the current state (or posterior). Further, note that the states s_j for all $j \neq i$ remain unchanged for playing arm i since we assume the bandit property. As we solve the dynamic program, we can keep track of what is the optimal action for every possible $\vec{S}_t$ and this will be the solution of the MDP. But as noted earlier, the set of possible states is of size $\Theta((kn)^T)$ if there are k outcomes for each arm. Thus it becomes infeasible to compute the optimal policy for larger instances of the problem.

Discounted Reward Problem

To overcome the computational intractability of the finite horizon version, we can make our horizon (the number of plays) infinite and introduce a discount factor. For a given $\gamma \in (0, 1)$ and start state $\vec{S}^0$, the goal is to find a policy that maximizes $\mathbf{E}[\sum_t \gamma^t R(t)]$. 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.

For a finite horizon T, a good approximation would be to set $\gamma = 1 - 1/T$, so that $\gamma^T \approx 1/e$. This would approximate the finite horizon reward and give an approximately optimal solution to the bandit problem.

Analogous to the finite horizon problem, let $V(\vec{S})$ denote the optimal infinite horizon discounted reward when the start state of the bandit is $\vec{S}$. We have the following recurrence to compute the optimal policy:

$$V(\vec{S}) = V(\langle s_1, \ldots s_n \rangle) = \max_i \left\{ R_i(s_i) + \gamma \sum_{\tilde{s}_i} p(s_i, \tilde{s}_i) V(\langle s_1, \ldots, \tilde{s}_i, \ldots s_n \rangle) \right\}$$

This recurrence relation is termed Bellman's equations. These do not lend themselves to backward induction. However, the solution can be computed by *value iteration*. Define the Bellman operator T that operates on a vector $\vec{W}$ of values for each state as follows:

$$TW(\vec{S}) = TW(\langle s_1, \ldots s_n \rangle) = \max_i \left\{ R_i(s_i) + \gamma \sum_{\tilde{s}_i} p(s_i, \tilde{s}_i) W(\langle s_1, \ldots, \tilde{s}_i, \ldots s_n \rangle) \right\}$$

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

Value Iteration: Start with $\vec{W} = \vec{0}$. Set $\vec{W} \leftarrow T\vec{W}$ till it converges to the fixed point $\vec{V}$.

Theorem 1. *Value iteration is well-defined. In other words, the Bellman operator T has a unique fixed point.*

Proof. Define $d(W, W') = \max_{\vec{S}} ||W(\vec{S}) - W'(\vec{S})||_\infty$. This defines a metric on the space of vectors $\vec{W}$. We will first show that $d(T\vec{W}, T\vec{W'}) \leq \gamma d(\vec{W}, \vec{W'})$. Fix some state $\vec{S}$, and let the optimal action for $TW(\vec{S})$ be to play i^*. Then we have:

$$TW(\vec{S}) = R_{i^*}(s_{i^*}) + \gamma \sum_{\tilde{s}_{i^*}} p(s_{i^*}, \tilde{s}_{i^*}) W(\langle s_1, \ldots, \tilde{s}_{i^*}, \ldots s_n \rangle)$$

$$TW'(\vec{S}) \geq R_{i^*}(s_{i^*}) + \gamma \sum_{\tilde{s}_{i^*}} p(s_{i^*}, \tilde{s}_{i^*}) W'(\langle s_1, \ldots, \tilde{s}_{i^*}, \ldots s_n \rangle)$$

where the former equality follows from the optimality of action i^*, and the latter inequality follows from the possible sub-optimality of i^* for TW'. Combining the two, we have:

$$TW(\vec{S}) - TW'(\vec{S}) \leq \gamma \max_{\tilde{s}_i} |W(\langle s_1, \ldots, \tilde{s}_{i^*}, \ldots s_n \rangle) - W'(\langle s_1, \ldots, \tilde{s}_{i^*}, \ldots s_n \rangle)| \leq \gamma \max_{\vec{S'}} |W(\vec{S'}) - W'(\vec{S'})|$$

Similarly, it can be shown that $TW'(\vec{S}) - TW(\vec{S}) \leq \gamma \max_{\vec{S'}} |W'(\vec{S'}) - W(\vec{S'})|$, 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 $T\vec{V} = \vec{V}$. □

For the multi armed bandit problem, value iteration still takes exponential time since the state space $\{\vec{S}\}$ is exponential in the number n of arms. In the next lecture, we will see how to get rid of the exponential dependence on n.

CPS296.04: Sequential Decision Theory **November 29, 2009**

Lecture 9 : The Gittins Index Policy

Lecturer: Kamesh Munagala *Scribe: Harish Chandran*

Introduction

In this lecture we will present the optimal policy for the multi armed bandit problem, termed the Gittins index. We shall describe the proof given by John Tsitsiklis (A short proof of Gittins index theorem). We shall be working with the discounted infinite horizon model of the problem. Let us define the bandit model.

Bandit model

Assume that the bandit has n arms where each arm i has a finite state space ψ_i. To simplify the analysis, we can assume that these states are disjoint and let $\Psi = \psi_1 \cup \psi_2 \cdots \cup \psi_n$. At some point of time, if arm i was in state $x \in \psi_i$ and is played, a random reward $R(x)$ is received. Generalizing the problem a little, we assume that if the arm is played, then for a random period of time $T(x)$, the arm (and hence the entire bandit) is locked up, after which it goes to a random state $Y(x)$. After this time interval, we can play any arm of the bandit again. This little modification of the original problem will prove useful for designing the algorithm.

We also assume that the joint probability distribution of the random vector $\langle T(x), R(x), Y(x)\rangle$ is known and equal to the joint probability distribution of the random vector $\langle T(y), R(y), Y(y)\rangle$ iff $x = y$. Also we assume the outcome of different plays of an arm in the same state are independent.

Accounting for the Reward

A policy for the bandit is a function $\pi : \psi_1 \times \psi_2 \times \cdots \times \psi_n \to \{1, 2, \ldots, n\}$ that maps the state of the bandit to an arm to play. 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. Let $e^{-\beta}$ be the discount factor. The problem is then finding the optimal policy that maximizes the expected discounted reward for every possible initial state:

$$E\left[\sum_{i=0}^{\infty} e^{-\beta t_i} R_i\right]$$

For any policy, the following is true. Let arm played in the i^{th} step be in state x_i. Let Φ_i be the set of random variables realized in the first $i-1$ plays. Observe that x_i and t_i are based on the outcomes of the first $i-1$ plays and hence depend on the set Φ_i. Conditioning on this set, the expected discounted reward of the i^{th} step is given by

$$e^{-\beta t_i} E[R(x_i)|x_i]$$

But instead of giving this reward upfront, one can think of spreading it over the duration of the locked up period, i.e., when an arm at some state x is played, rewards are received at a constant

rate $r(x)$ throughout the duration of that play (the locked up period). Thus,

$$r(x) = \frac{E[R(x)]}{E\left[\int_0^{T(x)} e^{-\beta t} dt\right]}$$

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

$$E\left[\int_{t_i}^{t_i+T(x_i)} e^{-\beta t} r(x_i) dt | \Phi_i\right] = e^{-\beta t_i} r(x_i) E\left[\int_0^{T(x_i)} e^{-\beta t} dt | x_i\right] = e^{-\beta t_i} E[R(x_i)|x_i]$$

Thus both these reward accounting schemes work out to the same final reward. We shall be using this later in the proof.

The Gittins Index Theorem

Recall that $\Psi = \psi_1 \cup \psi_2 \cdots \cup \psi_n$, the union of all the states of each of the n arms. A policy is termed a *priority rule* if there exists an ordering on the states in Ψ such that at any decision point, the arm whose current state is highest in the ordering is chosen for play. The Gittins index theorem states that there is indeed a priority rule that is optimal! The proof proceeds by first observing that $r(x)$ itself serves as a very good priority rule.

Lemma 1. *Let $s^* = \arg\max_{x\in\Psi} r(x)$ and let i^* be the arm such that $s^* \in \psi_{i^*}$. Then the following policy is optimal: whenever arm i^* is in state s^*, arm i^* is played.*

Proof. Consider an optimal policy π. Suppose at time 0, arm i^* is in state s^*. If π plays arm i^*, then the optimal policy is our policy and we are done. If not, π plays some other arm. Let τ be a random variable equal to the first time π plays arm i^*. If the arm is never played, $\tau = \infty$.
Define a new policy $\tilde{\pi}$ that plays arm i^* once and then mimics π. But when π plays arm i^* for the first time, $\tilde{\pi}$ skips that play of arm i^*. Let $\bar{r}(t)$ be the rate of reward as a function of time under the policy π, i.e., if $x(t)$ is the state of the arm played at time t, then $\bar{r}(t) = r(x(t))$. Since $s^* = \arg\max_{x\in\Psi} r(x)$, $\bar{r}(t) \leq r(s^*)$ for all t. If the expected discounted reward under policy π is $J(\pi)$, then

$$J(\pi) = E\left[\int_0^{\tau} \bar{r}(t) e^{-\beta t} dt + e^{-\beta\tau} \int_0^{T(s^*)} r(s^*) e^{-\beta t} dt + \int_{\tau+T(s^*)}^{\infty} \bar{r}(t) e^{-\beta t} dt\right]$$

Similarly,

$$J(\tilde{\pi}) = E\left[\int_0^{T(s^*)} r(s^*) e^{-\beta t} dt + e^{-\beta T(s^*)} \int_0^{\tau} \bar{r}(t) e^{-\beta t} dt + \int_{\tau+T(s^*)}^{\infty} \bar{r}(t) e^{-\beta t} dt\right]$$

If we prove,

$$E\left[(1 - e^{-\beta\tau}) \int_0^{T(s^*)} r(s^*) e^{-\beta t} dt\right] \geq E\left[(1 - e^{-\beta T(s^*)}) \int_0^{\tau} \bar{r}(t) e^{-\beta t} dt\right]$$

then we have proved that $J(\tilde{\pi}) \geq J(\pi)$. To do this, observe that

$$E\left[(1 - e^{-\beta\tau}) \int_0^{T(s^*)} r(s^*) e^{-\beta t} dt\right] = E\left[(1 - e^{-\beta T(s^*)}) \int_0^{\tau} \bar{r}(t) e^{-\beta t} dt\right] \text{ if } \bar{r}(t) = r(s^*) \text{ for all } t$$

But as proved earlier, $\bar{r}(t) \leq r(s^*)$ for all t and hence

$$E\left[(1-e^{-\beta\tau})\int_0^{T(s^*)} r(s^*)e^{-\beta t}dt\right] \geq E\left[(1-e^{-\beta T(s^*)})\int_0^{\tau} \bar{r}(t)e^{-\beta t}dt\right]$$

as the function inside the integral sign has increased for the left hand side. Thus $J(\tilde{\pi}) \geq J(\pi)$ and this implies that $\tilde{\pi}$ is optimal since π is optimal. If it was optimal to give top priority to state s^* at time 0, then it is clearly also optimal to give top priority to state s^* at every decision point. □

Using the above lemma we can prove the Gittins index theorem stated below.

Theorem 1. *If the state space of all the arms are finite, then there exists a priority rule that is optimal.*

Proof. Let $N = |\Psi|$ be the cardinality of the set Ψ. We prove the claim by induction on N. Base case of $N = 1$ is trivial, we have only one arm and the optimal policy will be a priority rule. We assume that the claim is true for all multi armed bandits for which $N = K$, where K is some postive integer. We will show that there exists a priority rule that is optimal for some bandit problem of $N = K + 1$ and this completes the inductive proof of theorem.

The previous lemma tells us that there exists an optimal policy within the set of policies that gives top priority to s^*. Let $\Pi(s^*)$ be this set of policies. Let us find the optimal policy within $\Pi(s^*)$. If s^* is the only possible state of arm i^*, then the policy that always plays arm i^* is optimal and is a priority rule. If not, suppose that arm i^* is in some other state $x \neq s^*$ and is played in this state. If this play made arm i^* to go to state s^*, then arm i^* is played over and over till the arm transitions to some other state different from s^*. We can view this succession of plays as a single composite play which is not interrupted due to our restriction to $\Pi(s^*)$. This composite play will have a random duration $\hat{T}(x)$ which equals the time that has elapsed from changing of the state of arm i^* from x to s^* to the changing of state from s^* to some other state. As we noted earlier, we can use the constant rate reward accounting method and still the total reward is going to be the same. Thus if the constant reward is $\hat{r}(x)$, then

$$\hat{r}(x) = \frac{E\left[\int_0^{\hat{T}(x)} e^{-\beta t}\bar{r}(t)\right]}{E\left[\int_0^{\hat{T}(x)} e^{-\beta t}\right]}$$

This constant reward will be obtained for the time period $\hat{T}(x)$. Thus we may replace arm i^* by a new arm in which state s^* is not present, $T(x)$ and $r(x)$ are replaced with $\hat{T}(x)$ and $\hat{r}(x)$ and suitably modified transition probabilities. This procedure thus removes state s^* entirely.

Thus the problem of finding the optimal policy within $\Pi(s^*)$ is a new multi armed bandit problem with $N = K$ (since we have reduced the state space of arm i^* by one). But by inductive hypothesis, there exists a priority rule $\hat{\pi}$ which is optimal for this problem. Thus there is a priority rule for the original problem ($N = K + 1$) and the rule is: give top priority to s^* and follow $\hat{\pi}$ for the remaining K states. □

Computing the Gittins index

We have proved that there is a priority rule that is optimal. But how do we actually use the theorem to solve the bandit problem? To that end, for each $x \in \Psi$, let us compute the *index* of the state $\gamma(x)$ by the following iterative procedure:

1. Let $s^* = \arg\max_{x \in \Psi} r(x)$. Set $\gamma(s^*) = r(s^*)$. Let i^* be the arm such that $s^* \in \psi_{i^*}$.
2. If ψ_{i^*} is a singleton, remove arm i^* from the bandit. Else, reduce arm i^* by removing s^*.
3. Go to step 1.

But notice that $\hat{T}(x)$, $\hat{r}(x)$ and the transition probabilities of a reduced arm i^* is completely determined by $T(x)$, $r(x)$ and the transistional probabilities of the original arm i^*. Thus the above algorithm can be carried out seperately for each arm! We can now compute these indicies and use it to solve the bandit problem as proved in the following theorem.

Theorem 2. *If the index of a state is computed using the aforementioned procedure, then any priority policy in which states with higher index have higher priority is optimal.*

Proof. By the previous theorem, any priority policy that orders the states in the same order as they are picked by the aforementioned index algorithm is optimal. Thus we need to show that the index algorithm picks a state x before state y iff $\gamma(x) \geq \gamma(y)$. Since the algorithm is recursive, it suffices to show that if the first and second states picked by the algorithm is s^* and q^*, then $\gamma(s^*) \geq \gamma(q^*)$. Let i^* be such that $s^* \in \psi_{i^*}$. If $q^* \in \psi_{i^*}$, then

$$\gamma(q^*) = \hat{r}(q^*) \leq \max_{x \in \Psi} r(x) = r(s^*) = \gamma(s^*)$$

If $q^* \notin \psi_{i^*}$, then

$$\gamma(q^*) = r(q^*) \leq r(s^*) = \gamma(s^*)$$

□

Equivalent Definition: An equivalent procedure for computing the Gittins index $\gamma(s)$ for $s \in \Psi_i$ is the following: Define a new discounted MDP just on space Ψ_i. In this new problem, there is a penalty M per step whenever the arm is played. Let $J(s, M)$ denote the optimal discounted reward minus penalty when the start state is s. The largest value of M for which $J(s, M) > 0$ is the Gittins index $\gamma(s)$. It is easy to check that this definition of the Gittins index is equivalent to the inductive definition above.

Intuition: 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 $F_{a,b}(y) = \Pr[\mu(\theta) > y]$. If the distribution of $\mu(\theta)$ is log-concave, then the Gittins index for discount factor $e^{-\beta}$ is roughly given by $\gamma(a, b) = F_{a,b}^{-1}(1 - e^{-\beta})$. In particular, if the finite time horizon is T, this is approximated by using a discount factor of $1 - 1/T$, so that the Gittins index is roughly $F_{a,b}^{-1}(1/T)$. To see why this makes sense, the probability of the mean exceeding $\gamma(a, b)$ is $1/T$, in which case (assuming the density of the mean is log-concave), we obtain reward of approximately $\gamma(a, b)$ for T steps, so that the expected total reward is $\gamma(a, b)$. Therefore, if the penalty M is set to $\gamma(a, b)$, it cancels out the reward. This rough calculation shows that the Gittins index also captures the variance in the underlying mean, rather than just its expectation.

CPS296.04: Sequential Decision Theory **November 30, 2009**

Lecture 11 : Regret Measure and the UCB1 Policy

Lecturer: Kamesh Munagala *Scribe: Nikhil Gopalkrishnan*

In the previous lectures, we made the assumption that though the reward distribution of each bandit arm is unknown, this distribution is parametrized, and a prior over possible parameter values is known. The goal is to find a policy that maximizes the expected reward over T steps (or the infinite horizon discounted reward for a suitably chosen discount factor), where the expectation is over the prior distributions, and the underlying reward distributions whose parameters are drawn according to the priors. We saw that this problem is a special case of Markov Decision Processes, and presented a polynomial time solution via the Gittins index policy.

In this and subsequent lectures, we remove the assumption that prior distributions are specified as input, and replace it with increasingly worst-case assumptions. In particular, we show that if the number of arms n is much smaller than the time horizon T, then it is relatively straightforward to converge to the arm with the highest expected reward by sampling each arm sufficiently many times. This method obviates the need for priors.

Regret measure

In the presence of priors, it was straightforward to define the optimal policy and its expected reward. In the absence of priors, we need an upper bound on the optimal policy – this is achieved by assuming that this policy is *omniscient* and knows the underlying reward distributions. However, note that 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 that of the optimal omniscient policy. This is termed *regret*, and measures the worst-case loss due to lack of information about the underlying reward distributions.

More formally, each of n arms in a MAB problem gives a reward governed by a random variable $X_i \in [0, 1]$ with mean μ_i. Each play of an arm obtains reward independent of the previous one. Given a finite horizon T, we seek a policy that minimizes *regret*. The omniscient optimum would simply play the arm with highest expected reward $i^* = \arg\max_i \mu_i$ for 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 Bayesian decision theory, where both the optimal policy and our policy work with the same information (priors), and the loss is due to lack of *computational* power in computing the optimal policy.

We will show a policy that obtains a regret of $O(\sqrt{nT \log T})$. 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 $\tilde{O}(nT^{\frac{2}{3}})$.

Before analyzing the algorithm, we state Hoeffding's inequality, which we will repeatedly use:

Algorithm 1 Naive algorithm to minimize regret

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

Lemma 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 average of the draws. Then:*

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

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

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

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

Thus assuming $n \ll T$, we have:

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

Hence with probability $1 - \frac{2}{T}$, the chosen arm i satisfies $\mu_i \geq \mu_{i^*} - \frac{2\sqrt{\log T}}{T^{\frac{1}{3}}}$. Hence,

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

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. □

The UCB1 Policy

The UCB1 algorithm mimics the Gittins index policy. Instead of defining an index using the prior, the index for each arm is simply the one-sided confidence interval for the sample average around which the true average falls with probability $1 - 1/T$. At each step Algorithm 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 $\text{Index}_i = \hat{\mu}_j + \sqrt{\frac{\ln T}{t_j}}$.

Algorithm 2 The UCB1 Policy

Play the arm $k = \arg\max_i \text{Index}_i$.
Update Index_k.

Let i^* denote the arm with highest true mean, μ^*. Let $\Delta_i = \mu^* - \mu_i$ be the regret on playing arm i once and Q_i be the expected number of times Algorithm 2 plays arm i in T steps.

Lemma 3. *For each arm $i \neq i^*$, $E(Q_i) \leq \frac{4 \ln T}{{\Delta_i}^2} + 2$.*

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

1. $\Pr[\text{Index}_{i^*} < \mu^*] \leq 1/T$. To see this, pretend i^* is played continuously. At each step t, $\text{Index}_{i^*}(t) = \hat{\mu}_{i^*} + \sqrt{\frac{\ln T}{t}}$. By Hoeffding's inequality, $\Pr[\text{Index}_{i^*}(t) < \mu^*] \leq 1/T^2$. By union bounds over T steps, $\Pr[\text{Index}_{i^*} < \mu^*] \leq 1/T$.

2. If arm i has been played $s_i = \frac{4 \ln T}{{\Delta_i}^2}$ steps, then $\text{Index}_i = \hat{\mu}_i + \frac{\Delta_i}{2}$. Then, $\Pr[\text{Index}_i > \mu^*] = \Pr[\hat{\mu}_i - \mu_i > \Delta_i/2]$. Again applying Hoeffding's inequality, this probability is at most $1/T$.

If neither of the above events happens, then arm i is played at most $\frac{4 \ln T}{{\Delta_i}^2}$ times. In the other case, w.p. $2/T$, the arm is played at most T times. □

Lemma 4. *Algorithm 2 incurs $O(\sqrt{nT \ln T})$ regret.*

Proof. Note that $\text{Regret} = \sum_i \Delta_i E(Q_i) \approx \sum_i (\frac{4 \ln T}{\Delta_i})$. 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 $\Delta_i \leq \sqrt{\frac{4n \ln T}{T}}$. For the arms in S_1, $\text{Regret} \leq n \frac{4 \ln T}{\sqrt{\frac{4n \ln T}{T}}} = 2\sqrt{nT \ln T}$. For arms in S_2, $\text{Regret} \leq T \max \Delta_i \leq T\sqrt{\frac{4n \ln T}{T}} = 2\sqrt{nT \ln T}$. □

Lower Bound on Regret

We now show that UCB1 is optimal, *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. In particular, 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* and has reward distribution Bernoulli$(1, \frac{1}{2} + \epsilon)$, where $\epsilon = c_1 \sqrt{\frac{n}{T}}$ for small constant c_1.

Lemma 5. *Given an arm with reward distribution Bernoulli$(1, p)$ where p is unknown, determining whether $p = 1/2$ or $p = 1/2 + \epsilon$ (for small ϵ) correctly with probability $1/2$ requires at least $\frac{c_2}{\epsilon^2}$ samples (or plays) in expectation, where c_2 is a small constant and the expectation assumes the underlying distribution is Bernoulli$(1, \frac{1}{2})$.*

Proof. Any sampling scheme is a mapping f from a subset of 0/1 reward vectors $\vec{r}$ to the indicator variable $0, 1$, where 0 denotes $p = 1/2$ and 1 denotes $p = 1/2 + \epsilon$. Let $P_0(\vec{r})$ denote the probability of reward vector $\vec{r}$ when $p = 1/2$, and let $P_1(\vec{r})$ denote the corresponding probability when $p = 1/2 + \epsilon$. Since the sampling scheme distinguishes the two cases w.p. at least $1/2$, it must be the case that:

$$\frac{1}{2} \leq \sum_{\vec{r}} f(\vec{r}) \left(P_1(\vec{r}) - P_0(\vec{r})\right) \leq ||P_0 - P_1||_1$$

We now use a standard result from information theory bounding the l_1 norm of the difference between two distributions in terms of their KL-divergence:

$$||P - Q||_1 \leq \sqrt{2 \ln 2 \times KL(P||Q)} \qquad \text{where} \qquad KL(P||Q) = \sum_{\vec{r}} P(\vec{r}) \log_2 \frac{P(\vec{r})}{Q(\vec{r})}$$

The quantity $KL(P_0||P_1)$ is easy to estimate by the chain rule: Conditioned on the sampling continuing into step t and conditioned on the rewards till step $t-1$, the KL-divergence of the rewards at the next step is simply $KL\left(\frac{1}{2}||\frac{1}{2}+\epsilon\right) \approx 4\epsilon^2$, since the rewards at the next step are independent of the samples so far. Therefore, if m denotes the expected number of samples needed to distinguish the distributions, we have:

$$\frac{1}{2} \leq ||P_0 - P_1||_1 \leq \sqrt{2\ln 2 KL(P_0||P_1)} \leq \sqrt{2\ln 2 \times m \times 4\epsilon^2}$$

Rearranging the terms completes the proof. □

Theorem 1. *Any bandit policy incurs regret* $\Omega(\sqrt{nT})$.

Proof. We use a simple "needle in the haystack" argument. For appropriately small choice of c_1 and $\epsilon = c_1\sqrt{\frac{n}{T}}$, by the above lemma, it takes $2\frac{T}{n}$ samples in expectation to determine whether an arm is the *good* one with probability at least $1/2$. Therefore, any policy can resolve at most $n/2$ such arms in expectation in a horizon of T. Since the *good* arm is a random arm, with probability $1/2$, the policy cannot find the good arm. In this case, it incurs regret $\epsilon T = \Omega(\sqrt{nT})$. □

The above proof is not particularly rigorous (and deliberately so to give intuition): A more rigorous proof is in the readings, and follows essentially the same logic.

CPS296.04: Sequential Decision Theory **December 1, 2009**

Lecture 12 : Adversarial Bandit Problem

Lecturer: Kamesh Munagala *Scribe: Kamesh Munagala*

In the previous lecture, we had assumed the reward of an arm are *i.i.d.* samples from an unknown distribution, and our performance measure was the *expected* regret against the omniscient strategy, where the expectation is over the samples drawn according to the underlying distributions. This was in contrast with the Bayesian assumption where the expectation was over both the prior distributions over the parameters of the unknown distributions, as well as the rewards drawn *i.i.d.* from the underlying distributions.

In this lecture, we ask: Can the notion of regret be made more robust? In particular, can we achieve low regret not just in expectation over the samples, but also in the worst case regardless of what samples materialize. In particular, for *any* reward sequence, can we show $O(\sqrt{T})$ regret against the omniscient policy that chooses the best arm in hindsight for that sample? For this purpose, we redefine regret of an algorithm A as follows:

$$\text{Regret} = \max_{\vec{r}} \left(\max_i \sum_{t=1}^{T} r_t^i - \sum_{t=1}^{T} r_t^A \right)$$

The goal of the *adversarial* bandit problem is to minimize this regret. We note that an alternative way of viewing the problem is by assuming that the rewards at each step are given by an adversary. The algorithm we design can switch arms, while the omniscient algorithm is forced to choose *one* arm, but with the benefit of hindsight. This mirrors the assumptions made in the stochastic bandit problem, where the omniscient strategy has no incentive to switch arms.

Online Prediction

A generic approach to solving the adversarial bandit problem is to define an auxiliary *prediction* problem, develop a low regret algorithm for the prediction problem, and use the prediction problem as a subroutine to solve the bandit problem. The prediction problem differs from the bandit problem in that at each step, after the algorithm plays an arm, it gets to know the rewards of *all* the arms, and not just the arm it played.

Since the prediction problem is closely tied to convex optimization and zero-sum games, before proceeding further, we will standardize our notation. Suppose the rewards lie in $[0, 1]$. Define the loss $l = 1 - r$. Let l_i^t denote the loss of arm i at step t. Let P denote the set of all unit vectors in n dimensions, where vector $\vec{e}_i$ denotes that arm i is chosen. For any algorithm A, let $\vec{x}_t \in P$ denote the decision made at step t. Minimizing regret can now be restated as:

$$\text{Regret} = \max_{\vec{r}} \left(\sum_{t=1}^{T} \vec{l_t} \cdot \vec{x}_t - \min_{\vec{x} \in P} \sum_{t=1}^{T} \vec{l_t} \cdot \vec{x} \right)$$

We will term the above prediction problem with loss vectors $\vec{l_t} \in [0, 1]^n$ and the space P being the unit vectors in n dimensions, as the *vanilla* prediction problem. Note that the way we have stated this problem, it can be generalized to the following scenarios:

Linear Optimization: In this case, P is an arbitrary set, and the only assumption made is that $\min_{\vec{x} \in P} \vec{l} \cdot \vec{x}$ can be efficiently computed for any $\vec{l}$. An example is the *shortest path* problem,

where at each step, the algorithm must choose an $s-t$ path, and subsequently the delays on all edges are revealed to it. The goal is to minimize regret in terms of total delay incurred over T steps against the omniscient policy that chooses one $s-t$ path, but with the benefit of hindsight.

Convex Optimization: In this case, P is a convex set, and the loss is a convex function $l_t(\vec{x_t})$ that is revealed after $\vec{x_t}$ is chosen. As before, the goal is to minimize regret against the omniscient policy that knows all the functions l_t, but is required to choose only one point $\vec{x}^*$.

Deterministic versus Randomized Algorithms

It is relatively easy to see that for the vanilla prediction problem (where the loss vectors are 0/1 vectors), no deterministic policy can incur low regret. Note that any deterministic policy can be viewed as a mapping from loss vectors in previous steps to an action in the current step. Given an algorithm A, the adversary simply sets the loss of the current action to 1 and the losses of the remaining actions to 0. This forces the algorithm A to incur loss T. By a simple averaging argument, there is an arm with loss at most T/n, and this is the loss of the omniscient policy.

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. For the vanilla prediction problem, we show a prediction algorithm with regret $O(\sqrt{T \log n})$. We will then use this as a black-box to design a generic adversarial bandit algorithm with regret $O(T^{2/3} n^{1/3} \log n)$.

One simple way of randomizing the vanilla prediction problem is to consider the set $Q = \{\vec{x} \in [0,1]^n | ||x||_1 = 1\}$ which represents all probability distributions over the n arms. The new regret minimization problem for algorithm A over Q is:

$$\text{Regret} = \max_{\vec{r}} \left(\sum_{t=1}^{T} \vec{l_t} \cdot \vec{x}_t - \min_{\vec{x} \in Q} \sum_{t=1}^{T} \vec{l_t} \cdot \vec{x} \right)$$

If the algorithm A predicts $\vec{x_t} \in Q$ at step t, the actual policy will choose arm i w.p. x_t^i, so that the expected loss is precisely $\vec{l_t} \cdot \vec{x}_t$.

Follow the Regularized Leader

A natural algorithm for the prediction problem is to *follow the leader* (FTL). This is simply:

$$\vec{x}_t = \text{argmin}_{\vec{x} \in Q} \sum_{r=1}^{t-1} \vec{l_r} \cdot \vec{x}$$

This policy will choose that arm which has minimum total loss so far. Since this leads to deterministic decisions, the above lower bound shows this cannot have low regret. In particular, the adversary can make the leader arm have loss 1 each step, causing the leader arm to toggle continuously.

The key idea behind all low-regret prediction algorithms is to add a *regularization* term to the FTL minimization procedure. This regularization term is large enough to prevent the adversary from having a fine-grained control over the leader, but is small enough to ensure the leader is

faithful to the original problem. Therefore, the generic follow the regularized leader (FTRL) is given by:

$$\vec{x}_t = \text{argmin}_{\vec{x} \in Q} \left(\sum_{r=1}^{t-1} \vec{l_r} \cdot \vec{x} + \sqrt{T} \cdot R(\vec{x}) \right)$$

Intuitively, the regularizer $R(\vec{x})$ *spreads out* the mass of $\vec{x}$ so that it is as far away from a deterministic 0/1 vector as possible. Here are some choices:

Entropy: $R(\vec{x}) = \sum_{i=1}^{n} x_i \log \frac{1}{x_i}$. This leads to the weighted majority algorithm for the vanilla prediction problem: Define $w_t^i = (1-\eta)^{\sum_{r=1}^{t-1} l_r^i}$, and choose arm i at step t w.p. proportional to w_t^i. The term $\eta \approx \sqrt{\frac{\ln n}{T}}$. We analyze this algorithm below.

Random Vector: $R(\vec{x}) = \vec{w} \cdot \vec{x}$, where $\vec{w}$ is a vector whose components are chosen uniformly at random in $[0, 1]$. This regularizer is used for the linear optimization problem. Note that in this case, we use $Q = P$, since the randomness is present within the regularizer itself. For the shortest path problem, think about the regularizer as initially adding a random delay of magnitude $\sqrt{T}$ to each edge, and then computing the shortest path in hindsight each step.

Sum of Squares: $R(\vec{x}) = ||\vec{x}||_2^2$. This regularizer is used for the convex optimization problem and is roughly equivalent to the following *infinitesimal gradient descent* rule (where *Proj* is the projection onto set P):

$$\vec{x}_{t+1} = \text{Proj}_P \left(\vec{x}_t - \frac{1}{\sqrt{t}} \nabla l_t(\vec{x}_t) \right)$$

The Weighted Majority Algorithm

The weighted majority algorithm for the vanilla prediction problem uses the entropy function as the regularizer. Let $\eta = \min \left\{ \sqrt{\frac{\ln n}{T}}, \frac{1}{2} \right\}$. The algorithm is given below.

Algorithm 1 The Randomized Weighted Majority (RWM) Algorithm

Initialize: Set $w_1^i = 1, p_1^i = \frac{1}{n}$ for all $i = 1, 2, \ldots, n$.
Set $w_t^i = w_{t-1}^i (1-\eta)^{l_{i-1}^i}$ for all i.
Set $p_t^i = w_t^i / (\sum_{j=1}^{n} w_t^j)$ for all i.
Play arm i with probability p_t^i at step t, and observe $\vec{l_t}$.

Theorem 1. *The RWM algorithm with parameter* $\eta = \min \left\{ \sqrt{\frac{\ln n}{T}}, \frac{1}{2} \right\}$ *has regret* $O(\sqrt{T \ln n})$.

Proof. Let L^* denote the optimal loss in hindsight for T steps, and let L_T^{RWM} denote the loss of the RWM algorithm. We will show:

$$L_T^{RWM} \leq (1+\eta) L_T^* + \frac{\ln n}{\eta}$$

For $\eta = \min \left\{ \sqrt{\frac{\ln n}{T}}, \frac{1}{2} \right\}$, this will imply the regret bound.

We will prove the result assuming $l_t^i \in \{0,1\}$ for all i,t. The proof for $l_t^i \in [0,1]$ is similar. 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. Let

$$F^t = \frac{\sum_{i:l_t^i=1} w_t^i}{W^t} = \mathbf{E}[\text{ Loss of RWM at step } t]$$

Since each i for which $l_t^i = 1$ has $w_{t+1}^i = (1-\eta)w_t^i$, we must have:

$$W^{t+1} = W^t - \eta F^t W^t = W^t(1-\eta F^t)$$

Therefore,

$$W^{T+1} = W^1 \prod_{t=1}^{T}(1-\eta F^t) = n\prod_{t=1}^{T}(1-\eta F^t)$$

However, we also have $W^{T+1} \geq (1-\eta)^{L^*}$. Combining these two identities and taking logs, we have:

$$\begin{aligned} L^* \ln(1-\eta) &\leq \ln n + \sum_{t=1}^{T} \ln(1-\eta F^t) \\ &\leq \ln n - \sum_{t=1}^{T} \eta F^t \\ &= \ln n - \eta L_T^{RWM} \end{aligned}$$

Therefore, assuming $\eta \leq 1/2$, we have:

$$L_T^{RWM} \leq L^* \frac{\ln(1/(1-\eta))}{\eta} + \frac{\ln n}{\eta} \leq (1+\eta)L^* + \frac{\ln n}{\eta}$$

□

The RWM algorithm can be viewed as a primal-dual algorithm for solving two person zero-sum games, where the strategy $\vec{x}_t$ converges to the primal solution and the weights $\vec{w}_t$ converge to the dual solution. We will present a proof of the celebrated Min-Max theorem using the RWM algorithm in a later lecture.

From Prediction to Bandit Algorithms

We now present a generic technique for converting a vanilla prediction algorithm to a bandit algorithm yielding regret $O(T^{2/3}n^{1/3}\ln n)$ for the latter problem. This bound is not optimal, and a more refined algorithm actually yields the optimal $O(n\sqrt{T})$ bound. We present the sub-optimal algorithm as it illustrates a simple and general technique.

Recall that in the bandit version, the algorithm only learns the loss of the arm that it plays in contrast with the prediction problem where the algorithm learns the loss of all the arms. (Similarly, in the linear and convex optimization versions, the bandit algorithm only learns the objective value $l_t(\vec{x}_t)$, rather than the entire function l_t.) The first step is to divide the time horizon T into K phases of length T/K each. Denote these phases $B_1, B_2, \ldots, B_K$, and a generic phase by B_τ.

We first present a FAKE algorithm on which we build later. Let:

$$\vec{c}_\tau = \sum_{t \in B_\tau} \frac{\vec{l}_t}{|B_\tau|} \qquad \vec{C} = \sum_{\tau=1}^{K} \vec{c}_\tau \qquad C^* = \min_i C_i$$

Note that $c_\tau^i \in [0,1]$ assuming $l_t^i \in [0,1]$. At the end of phase τ, feed the vector $\vec{c}_\tau$ to the RWM algorithm and obtain the prediction for phase $\tau+1$. This prediction is a probability distribution $\vec{p}_{\tau+1}$; at each step in phase $\tau+1$, play arm i with probability $p_{\tau+1}^i$. The regret analysis of this algorithm is simple: Since the RWM algorithm is run for K steps:

$$\mathbf{E}[\text{Loss of RWM}] = \sum_{\tau=1}^{K} \vec{p}_\tau \cdot \vec{c}_\tau \leq C^* + O(\sqrt{K \ln n})$$

Therefore, if L^{FAKE} is the loss of the FAKE bandit algorithm and L^* denotes the optimal loss:

$$\begin{aligned} L_T^{FAKE} &= \sum_{\tau=1}^{K} \sum_{t \in B_\tau} \vec{p}_\tau \cdot \vec{l}_t = \frac{T}{K} \sum_{\tau=1}^{K} \vec{p}_\tau \cdot \vec{c}_\tau \\ &\leq \frac{T}{K} \left(C^* + O(\sqrt{K \ln n}) \right) \\ &= L^* + O\left(T \sqrt{\frac{\ln n}{K}} \right) \end{aligned}$$

The main catch in the above analysis is that it is not possible to compute $\vec{c}_\tau$ since it requires knowing the losses for all the arms. The key idea is to develop an *unbiased estimator* for this quantity by sampling, and feed this estimator to the RWM algorithm. To compute the estimator $\hat{c}_\tau$, sample each arm i at a random time step in B_τ. This guarantees that $\mathbf{E}[\hat{c}_\tau] = \vec{c}_\tau$. This sampling is independent of the execution of the RWM algorithm itself.

Let $\hat{C}_K = \sum_{\tau=1}^{K} \hat{c}_\tau$, and let $\hat{C}^* = \min_{i=1}^{n} \hat{C}_K^i$. Let $\vec{p}_\tau$ denote the strategy generated by the RWM algorithm for phase τ. Note that $\hat{c}_\tau$ are random variables that depend on the outcome of sampling, and therefore, so are $\vec{p}_\tau$. However, note that $\vec{p}_\tau$ only depends on the sampling in phases $1, 2, \ldots, \tau-1$, and is hence independent of $\hat{c}_\tau$.

In the discussion below, all expectations are with respect to the randomness in sampling the $\hat{c}_\tau$. By the regret analysis of RWM, we directly have:

$$\sum_{\tau=1}^{K} \vec{p}_\tau \cdot \hat{c}_\tau \leq \hat{C}^* + O(\sqrt{K \ln n})$$

However, by the convexity of the minimum operator, we have:

$$\mathbf{E}[\hat{C}^*] = \mathbf{E}[\min_i \hat{C}_K^i] \leq \min_i \mathbf{E}[\hat{C}_K^i] = \min_i C_K^i = C^*$$

Therefore,

$$\sum_{\tau=1}^{K} \mathbf{E}[\vec{p}_\tau \cdot \hat{c}_\tau] \leq C^* + O(\sqrt{K \ln n})$$

By the independence of $\vec{p}_\tau$ and $\hat{c}_\tau$, we have:

$$\sum_{\tau=1}^{K} \mathbf{E}[\vec{p}_\tau \cdot \hat{c}_\tau] = \sum_{\tau=1}^{K} \mathbf{E}[\vec{p}_\tau] \cdot \mathbf{E}[\hat{c}_\tau] = \sum_{\tau=1}^{K} \mathbf{E}[\vec{p}_\tau] \cdot c_\tau = \frac{K}{T} \sum_{\tau=1}^{K} \sum_{t \in B_\tau} \mathbf{E}[\vec{p}_\tau \cdot \vec{l}_t]$$

Combining the above two inequalities, we have:

$$\mathbf{E}[\text{Loss of Bandit Algorithm}] = \sum_{\tau=1}^{K} \sum_{t \in B_\tau} \mathbf{E}[\vec{p}_\tau \cdot \vec{l}_t] \leq L^* + O\left(T \sqrt{\frac{\ln n}{K}} \right)$$

However, in the above analysis, we have ignored the regret in the sampling steps. There are n such steps each phase, and these are randomly chosen. Therefore, the sampling steps lead to an additional regret of at most nK. Therefore, the total regret is:

$$\mathbf{E}[\text{Total Regret}] = O\left(T\sqrt{\frac{\ln n}{K}} + nK\right) = O(T^{2/3}n^{1/3}\ln n) \qquad \text{for} \qquad K = (T/n)^{2/3}$$

CPS296.04: Sequential Decision Theory **December 1, 2009**

Lecture 19 : Online Algorithms

Lecturer: Kamesh Munagala *Scribe: Dima Korzhyk*

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.

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 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.

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 2. *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 $a_i \leq 2s_i$. If that is the case, then

$$\sum_i a_i \leq 2 \sum_i s_i = 2\text{OPT} \tag{1}$$

Consider the term $\sum_i a_i$.

$$\sum_i a_i = \sum_i t_i + \sum_i (\phi_i - \phi_{i-1}) \tag{2}$$
$$= \text{MTF} + \phi_n - \phi_0 \tag{3}$$
$$= \text{MTF} + \phi_n \tag{4}$$

Thus $a_i \leq 2s_i$ implies $\text{MTF} \leq 2\text{OPT} - \phi_n$, which means that MTF is 2-competitive. To finish the proof, we need to show that $a_i \leq 2s_i$.

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

$$s_i = \sum_y s_i(y) \tag{5}$$
$$t_i = \sum_y t_i(y) \tag{6}$$
$$\phi_{i-1} = \sum_y \phi_{i-1}(y) + C \tag{7}$$
$$\phi_i = \sum_y \phi_i(y) + C \tag{8}$$

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

$$t_i(y) + \phi_i(y) - \phi_{i-1}(y) \leq 2s_i(y) \tag{9}$$
$$\implies t_i + \phi_i - \phi_{i-1} \leq 2s_i \tag{10}$$
$$\implies a_i \leq 2s_i \tag{11}$$

This finishes the proof that MTF is 2-competitive. □

Next, we show that no deterministic ratio can do better than MTF.

Theorem 3. *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

$$\sum_x \mathbb{E}[\text{position of } x] = \frac{Tn}{2}$$

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. □

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 4. *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 5. *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. □

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 6. *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 $\text{OPT} \geq \sum_i \frac{M_i}{2}$. 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

$$\Pr[x_j \notin \text{cache}] = \frac{\binom{k-(j-1)-1}{k-(m_j+j-1)}}{\binom{k-(j-1)}{k-(m_j+j-1)}} \tag{12}$$

$$= \frac{m_j}{k-j+1} \tag{13}$$

$$\leq \frac{M_i}{k-j+1} \tag{14}$$

The expected number of evictions in phase i is

$$\mathbb{E}[\text{number of evictions in phase i}] \leq M_i + \sum_{j=1}^{k} \frac{M_i}{k-j+1} \tag{15}$$

$$\leq M_i(1 + \frac{1}{k} + \frac{1}{k-1} + \ldots + 1) \tag{16}$$

$$\leq M_i(\ln k + 1) \tag{17}$$

Thus randomized paging is $2 \ln k$ -competitive. □

CPS296.04: Sequential Decision Theory **December 1, 2009**

Lecture 20 : Lower Bounds for Randomized Online Algorithms

Lecturer: Kamesh Munagala *Scribe: Dima Korzhyk*

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.

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 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 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 $M(i,j) = \mathrm{ALG}_i(\sigma_j) - c \cdot \mathrm{OPT}(\sigma_j)$, 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:

$$\max_{\sigma_j} \mathbf{E}_{i \in D}[\mathrm{ALG}_i(\sigma_j) - c \cdot \mathrm{OPT}(\sigma_j)] \geq 0 \qquad \forall D$$

This in turn implies

$$\max_{\sigma_j} \frac{\mathbf{E}_{i \in D}[\mathrm{ALG}_i(\sigma_j)}{\mathrm{OPT}(\sigma_j)]} \geq c \qquad \forall D \qquad \Rightarrow \qquad \min_D \max_{\sigma_j} \frac{\mathbf{E}_{i \in D}[\mathrm{ALG}_i(\sigma_j)]}{\mathrm{OPT}(\sigma_j)} \geq c$$

On the other hand, since $\max_P \min_i \mathbf{E}_P[M(i,j)] = 0$, we have:

$$\min_i \mathbf{E}_{\sigma_j \in P}[\mathrm{ALG}_i(\sigma_j)] - c \cdot \mathbf{E}_{\sigma_j \in P}[\mathrm{OPT}(\sigma_j)] \leq 0 \qquad \forall P$$

This in turn implies:

$$\min_i \frac{\mathbf{E}_{\sigma_j \in P}[\mathrm{ALG}_i(\sigma_j)]}{\mathbf{E}_{\sigma_j \in P}[\mathrm{OPT}(\sigma_j)]} \leq c \qquad \forall P \qquad \Rightarrow \qquad \max_P \min_i \frac{\mathbf{E}_{\sigma_j \in P}[\mathrm{ALG}_i(\sigma_j)]}{\mathbf{E}_{\sigma_j \in P}[\mathrm{OPT}(\sigma_j)]} \leq c$$

□

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 3. *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. □

The Min-Max Theorem via Online Prediction

The min-max theorem is a special case of strong LP duality. To see the connection to duality, we can write $\min_{\mathcal{D}} \max_j M(\mathcal{D}, j)$ as the following LP, where $\vec{x}$ denotes the optimal distribution $\mathcal{D}$, and the first constraint encodes that the column player is a maximizer:

$$\text{Minimize } z$$

$$\begin{array}{rcll} \sum_i M(i,j)x_i & \leq & z & \forall j \\ \sum_i x_i & = & 1 & \\ x_i & \geq & 0 & \forall i \end{array}$$

The quantity $\max_{\mathcal{P}} \min_i M(i, \mathcal{P})$ can be written as the following linear program, where $\vec{y}$ is the optimal distribution $\mathcal{P}$, and the first constraint encodes that the row player is a minimizer:

$$\text{Maximize } w$$

$$\begin{array}{rcll} \sum_j M(i,j)y_j & \geq & w & \forall i \\ \sum_j y_j & = & 1 & \\ y_j & \geq & 0 & \forall j \end{array}$$

It is easy to see that the above two programs are duals of each other, and therefore, the min-max theorem is a corollary of strong LP duality. However, we can obtain a simpler algorithmic proof by a reduction to any low-regret prediction algorithm (particularly, the weighted majority algorithm). In effect, this reduction will show that low-regret prediction algorithms are closely tied to primal-dual algorithms to solve packing and covering linear programs.

Proof of the Min-Max Theorem. Let $\lambda^* = \min_{\mathcal{D}} \max_j M(\mathcal{D}, j)$. By weak duality, we have $\lambda^* \geq \max_{\mathcal{P}} \min_i M(i, \mathcal{P})$. For any $\delta > 0$, we will algorithmically construct a D^*_δ, P^*_δ such that:

- $\lambda^* \leq \max_j M(D^*_\sigma, j) \leq \lambda^* + \delta$
- $\lambda^* - \delta \leq \min_i M(i, P^*_\delta) \leq \lambda^*$

We note that $\lambda^* \leq \max_j M(D^*_\sigma, j)$ follows from the definition of λ^*, and $\min_i M(i, P^*_\delta) \leq \lambda^*$ follows from weak duality. Taking the limit as $\delta \to 0$ and denoting the limiting distributions by $\hat{D}, \hat{P}$, this will show that $\max_j M(\hat{D}, j) = \min_i M(i, \hat{P}) = \lambda^*$.

Let $\vec{M}_j$ denote the j^{th} column of M. Assume all entries of M are bounded in $[0, 1]$ without loss of generality. The algorithm for constructing the distributions is as follows: Think of the row player as the prediction algorithm and the column player as the adversary. Suppose the row player predicts distribution $D^{(t)}$ at step t. The column player has a choice of giving one of the columns $\vec{M}_1, \vec{M}_2, \ldots, \vec{M}_n$ as the loss vector; it chooses that column $j^{(t)}$ which maximizes $\vec{M}_{j^{(t)}} \cdot D^{(t)}$. The row player chooses its strategy according to any low-regret algorithm, in particular, the weighted majority algorithm. Note that this algorithm guarantees:

$$\sum_{t=1}^{T} \vec{M}_{j^{(t)}} \cdot D^{(t)} \leq \min_D D \cdot \sum_{t=1}^{T} \vec{M}_{j^{(t)}} + O(\sqrt{T \ln n})$$

Let $\delta = O\left(\sqrt{\frac{\ln n}{T}}\right) \to 0$ as $T \to \infty$. For ease of notation, let $M(D, j) = D \cdot \vec{M}_j$, and so on.

Since $j^{(t)}$ is a best response at time t, we have:

$$\begin{aligned} \lambda^* &\leq \max_j M(D^{(t)}, j) = M(D^{(t)}, j^{(t)}) \\ \implies \lambda^* &\leq \frac{1}{T}\sum_{t=1}^{T} M(D^{(t)}, j^{(t)}) \leq \delta + \frac{1}{T}\sum_{t=1}^{T} M(D^*, j^{(t)}) \end{aligned}$$

Here, $D^* = \text{argmin}_{\mathcal{D}} \max_j M(\mathcal{D}, j)$, and the final inequality follows from the low-regret condition. Since $M(D^*, j^{(t)}) \leq \lambda^*$ by definition, we have:

$$\frac{1}{T}\sum_{t=1}^{T} M(D^{(t)}, j^{(t)}) \leq \lambda^* + \delta$$

Now set $D^*_\delta = \frac{1}{T}\sum_{t=1}^{T} D^{(t)}$ as the average of the strategies of the row player. For this, we have:

$$\begin{aligned} \max_j M(D^*_\delta, j) &= M(D^*_\delta, j^*) \\ &= \frac{1}{T}\sum_{t=1}^{T} M(D^{(t)}, j^*) \\ &\leq \frac{1}{T}\sum_{t=1}^{T} M(D^{(t)}, j^{(t)}) \\ &\leq \lambda^* + \delta \end{aligned}$$

Here, j^* is the optimal column response to D^*_δ, and the second inequality follows since $j^{(t)}$ is the optimal column response to $D^{(t)}$. This shows $\lambda^* \leq \max_j M(D^*_\sigma, j) \leq \lambda^* + \delta$.

Now, let P^*_δ be the distribution induced by $j^{(1)}, j^{(2)}, \ldots, j^{(T)}$. The low-regret condition implies that for all D (and not just D^*) we have,

$$\begin{aligned} \lambda^* &\leq \delta + \frac{1}{T}\sum_{t=1}^{T} M(D, j^{(t)}) \\ \implies \lambda^* - \delta &\leq \frac{1}{T}\sum_{t=1}^{T} M(D, j^{(t)}) = M(D, P^*_\delta) \\ \implies \lambda^* - \delta &\leq \min_D M(D, P^*_\delta) = \min_i M(i, P^*_\delta) \end{aligned}$$

This shows $\lambda^* - \delta \leq \min_i M(i, P^*_\delta) \leq \lambda^*$, completing the proof. □

Algorithmic Implications: Note that as an algorithmic result, we have shown that to solve zero-sum games (where the entries are bounded in $[0, 1]$) to precision ϵ, we need running time $O\left(\frac{n^2 \ln n}{\epsilon^2}\right)$, where the term n^2 is the time needed to compute the optimal column strategy. Further, note the structure of the algorithm: At each step, given the primal (row) variables, find the most violated constraint, and increase its dual variable linearly. Now, minimize the Lagrangean with these dual variables with a regularizing term (entropy) to find the new primal variables. This leads to geometric updates to the primal solution. The final step scales down the primal and dual solutions to make them feasible. This general framework forms the cornerstone of all fast algorithms for approximately solving packing and covering linear programs.

CPS296.04: Sequential Decision Theory **December 2, 2009**

Lecture 21 : The Primal-Dual Method for Online Algorithms

Lecturer: Kamesh Munagala *Scribe: Kamesh Munagala*

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 ← 0
while K increases by 1 do
  if x < 1 then
    z_K ← 1 − x        Δx = (1/B)(x + 1/c)        y_K ← 1;
  end if
end while
```

Lemma 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. □

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.

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 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.

$$\text{Minimize } \sum_i c_i x_i s.t. \qquad \begin{array}{rcll} \sum_{i \in S(j)} x_i & \geq & 1 & \forall j \\ x_i & \geq & 0 & \forall i \end{array}$$

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 ← 0 for all i.
while new element j arrives do
  while Σ_{i∈S(j)} x_i < 1 do
    For all i ∈ S(j), set Δx_i = (1/c_i) (x_i + 1/|S(j)|).
    y_j ← y_j + 1.
  end while
end while
```

Theorem 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.