Course: Randomized Algorithms by Dr. Zarei

Homework: HW4

Name: Mohammad Mohammadi

Student ID: 402208592

Problem 2.4 [RA_Mot]

Determine the value V_R of the following 2 x 2 matrix game and give optimal mixed strategies for the two players.

$$\begin{bmatrix} 5 & 6 \\ 7 & 4 \end{bmatrix}$$

Answer

Let p be the probability that Player 1 plays the first row and $1-p$ be the probability that Player 1 plays the second row. Similarly, let q be the probability that Player 2 plays the first column and $1-q$ be the probability that Player 2 plays the second column.

The expected payoff for Player 1 when Player 2 plays the first column (q) is:

$$E_1(1) = 5p + 7(1-p)$$

The expected payoff for Player 1 when Player 2 plays the second column ($1-q$) is:

$$E_1(2) = 6p + 4(1-p)$$

For Player 1 to be indifferent between the two columns (maximize the minimum payoff), we set the expected payoffs equal:

$$5p + 7(1-p) = 6p + 4(1-p)$$

Solving for p:

$$5p + 7 - 7p = 6p + 4 - 4p$$

$$7 - 2p = 4 + 2p$$

$$3 = 4p$$

$$p = \frac{3}{4}$$

So, Player 1’s optimal mixed strategy is p=3/4 for the first row and p=1/4 for the second row.

Similarly, the expected payoff for Player 2 when Player 1 plays the first row is:

$$E_2(1) = 5q + 6(1-q)$$

The expected payoff for Player 2 when Player 1 plays the second row is:

$$E_2(2) = 7q + 4(1-q)$$

Setting these equal for indifference:

$$5q + 6(1-q) = 7q + 4(1-q)$$

Solving for q:

$$5q + 6 - 6q = 7q + 4 - 4q$$
$$6 - q = 4 + 3q$$
$$2 = 4q$$
$$q = \frac{1}{2}$$

So, Player 2's optimal mixed strategy is q=1/2 for the first column and 1-q=1/2 for the second column.

The value V_R of the game is the expected payoff when both players use their optimal mixed strategies:

$$V_R = 5 \cdot \frac{3}{4} \cdot \frac{1}{2} + 6 \cdot \frac{3}{4} \cdot \frac{1}{2} + 7 \cdot \frac{1}{4} \cdot \frac{1}{2} + 4 \cdot \frac{1}{4} \cdot \frac{1}{2}$$
$$V_R = \frac{15}{8} + \frac{18}{8} + \frac{7}{8} + \frac{4}{8}$$
$$V_R = \frac{44}{8} = 5.5$$

Problem 2.5 [RA_Mot]

(Due to A.M. Karp.) Let (a_{ij}) be a $m * n$ matrix, let the vector $(P_1, P_2, \ldots, P_m)$ consist of reals in $[0,1]$ such that $\sum_{i=1,m} P_i = 1$, and let $(q_1, q_2, \ldots, q_m)$ consist of reals in $[0,1]$ such that $\sum_{i=1,m} q_i = 1$. Prove algebraically that
$max_q min_j \sum_{j=1,n} a_{ij} q_j \leq min_q max_j \sum_{j=1,m} a_{ij} q_j$.

Answer

To prove that

$$\max_{\mathbf{q}} \min_{j} \sum_{i=1}^{m} a_{ij} q_i \leq \min_{\mathbf{p}} \max_{j} \sum_{i=1}^{m} a_{ij} p_i,$$

we need to use some fundamental results from game theory, particularly related to mixed strategies and the concept of the minimax theorem. Here, $\mathbf{p} = (p_1, p_2, \ldots, p_m)$ and

$\mathbf{q} = (q_1, q_2, \ldots, q_m)$ are probability distributions over the rows and columns of the matrix (a_{ij}) respectively.

1. Formulate the Problem in Game Theoretic Terms:

   Consider a zero-sum game where the matrix (a_{ij}) represents the payoff matrix.

   Player 1 selects a row i with probability p_i.

   Player 2 selects a column j with probability q_j.

2. Expected Payoffs:

   The expected payoff for Player 1 if Player 2 plays according to the strategy $\mathbf{q}$ is:

   $$E(\mathbf{q}) = \sum_{i=1}^{m} p_i \left(\sum_{j=1}^{n} a_{ij} q_j \right)$$

   The expected payoff for Player 2 if Player 1 plays according to the strategy $\mathbf{p}$ is:

   $$E(\mathbf{p}) = \sum_{j=1}^{n} q_j \left(\sum_{i=1}^{m} a_{ij} p_i \right)$$

3. Minimax Theorem Application:

   By the minimax theorem, for any finite two-person zero-sum game with matrix (a_{ij}), the following holds:

   $$\max_{\mathbf{q}} \min_{\mathbf{p}} \sum_{i=1}^{m} p_i \left(\sum_{j=1}^{n} a_{ij} q_j \right) = \min_{\mathbf{p}} \max_{\mathbf{q}} \sum_{j=1}^{n} q_j \left(\sum_{i=1}^{m} a_{ij} p_i \right)$$

4. Rewriting the Objective:

   - The left-hand side of our inequality $\max_{\mathbf{q}} \min_j \sum_{i=1}^{m} a_{ij} q_i$ can be interpreted as finding the best mixed strategy for Player 2 that maximizes their worst-case payoff.
   - The right-hand side $\min_{\mathbf{p}} \max_j \sum_{i=1}^{m} a_{ij} p_i$ can be interpreted as finding the best mixed strategy for Player 1 that minimizes the best response of Player 2.

5. Prove the Inequality:

   By the minimax theorem, we know:

   $$\max_{\mathbf{q}} \min_{\mathbf{p}} \sum_{i=1}^{m} p_i \left(\sum_{j=1}^{n} a_{ij} q_j \right) \leq \min_{\mathbf{p}} \max_{\mathbf{q}} \sum_{j=1}^{n} q_j \left(\sum_{i=1}^{m} a_{ij} p_i \right)$$

   Translating back to our specific terms, we get:

   $$\max_{\mathbf{q}} \min_{j} \sum_{i=1}^{m} a_{ij} q_i \leq \min_{\mathbf{p}} \max_{j} \sum_{i=1}^{m} a_{ij} p_i$$

Problem 2.6 [RA_Mot]

Use Yao's Minimax Principle to prove a lower bound on the expected running time of any Las Vegas algorithm for sorting n numbers.

We can use Yao's Minimax Principle to establish a lower bound on the expected running time of any Las Vegas algorithm for sorting n numbers. A Las Vegas algorithm is a randomized algorithm that always produces the correct result, but its running time is a random variable.

Yao's Minimax Principle states that the expected running time of the best randomized algorithm on the worst-case input is at least the expected running time of the best deterministic algorithm on the average-case input distribution.

Let $\mathcal{A}$ be a Las Vegas algorithm for sorting n numbers, and let $T(\mathcal{A}, x)$ denote the running time of $\mathcal{A}$ on input x.

FOr the lower bound we prove as:

1. **Define the Input Distribution:**

   Consider the uniform distribution over all $n!$ permutations of n distinct numbers. Let $\mathcal{D}$ denote this distribution.

2. **Expected Time for Deterministic Algorithms:**

   Let $\mathcal{A}_\mathcal{D}$ be any deterministic algorithm for sorting. The expected running time of $\mathcal{A}_\mathcal{D}$ under the distribution $\mathcal{D}$ is:

   $$\mathbb{E}_\text{x} \sim \mathcal{D}[T(\mathcal{A}_\mathcal{D}, x)] = \frac{1}{n!} \sum_{x \in S_n} T(\mathcal{A}_\mathcal{D}, x)$$

   where S_n is the set of all $n!$ permutations of n numbers.

3. **Comparison-based Sorting Lower Bound:**

   Any comparison-based deterministic sorting algorithm must perform at least $\log_2(n!)$ comparisons in the worst case. By Stirling's approximation, we have:

   $$\log_2(n!) \approx n \log_2 n - n \log_2 e$$

   Therefore, the expected number of comparisons for any deterministic algorithm under the uniform distribution is at least:

   $$\mathbb{E}_{x \sim \mathcal{D}}[T(\mathcal{A}_D, x)] \geq \log_2(n!)$$

4. **Application of Yao's Minimax Principle:**

   By Yao's Minimax Principle, the expected running time of the best Las Vegas algorithm on the worst-case input is at least the expected running time of the best deterministic algorithm on the average-case input distribution. Hence,

   $$\min_{\mathcal{A}} \max_{x} \mathbb{E}[T(\mathcal{A}, x)] \geq \min_{\mathcal{A}_\mathcal{D}} \mathbb{E}_\text{x} \sim \mathcal{D}[T(\mathcal{A}_\mathcal{D}, x)]$$

   Given the lower bound for deterministic algorithms:

   $$\min_{\mathcal{A}} \max_{x} \mathbb{E}[T(\mathcal{A}, x)] \geq \log_2(n!)$$

5. **Conclusion:**

Using Stirling's approximation again, we conclude that:

$$\log_2(n!) \approx n \log_2 n - n \log_2 e$$

Therefore, the expected running time of any Las Vegas algorithm for sorting n numbers is at least $\Omega(n \log n)$.

So using Yao's Minimax Principle, we have shown that the expected running time of any Las Vegas algorithm for sorting n numbers is at least $\Omega(n \log n)$.

Problem 2.7 [RA_Mot]

(Due to A.M. Karp.) You are given an array A containing n numbers in sorted order. In one step, an algorithm may specify an integer $i \epsilon [1, n]$, and is given the value of $A[i]$ in return. Determine lower and upper bounds on the expected number of steps taken by a Las Vegas randomized algorithm to determine whether or not a given key k is present in the array.

Answer

Given an array A containing n numbers in sorted order, determine lower and upper bounds on the expected number of steps taken by a Las Vegas randomized algorithm to determine whether a given key k is present in the array.

For Upper Bound

A Las Vegas randomized algorithm must always produce the correct result, but its running time is a random variable. One approach is to use randomized binary search:

1. Randomly select an index i from the array.
2. Compare $A[i]$ with the key k:
    - If $A[i] = k$, return the result.
    - If $A[i] < k$, discard the left half and repeat on the right half.
    - If $A[i] > k$, discard the right half and repeat on the left half.

The expected number of steps $E(n)$ to find the key can be derived as follows:

- Initially, the search interval is of size n.
- At each step, the interval size reduces by half (on average), and this reduction continues until the interval size becomes 1.

The number of steps required for the interval to reduce to 1 is $log_2 n$ (base 2 logarithm).

Since we randomly select the midpoint at each step, the expected number of steps is given by the harmonic series sum for binary search:

$$E(n) = 1 + \frac{1}{2}E(n/2) + \frac{1}{2}E(n/2)$$

Simplifying the recurrence relation:

$$E(n) = 1 + E(n/2)$$

Solving this, we get:

$$E(n) = O(\log n)$$

Thus, the upper bound on the expected number of steps is $O(\log n)$.

For Lower Bound

Considering the information-theoretic limit, any comparison-based algorithm must make $\log_2 n$ comparisons in the worst case to determine whether a key is present.

Therefore, the lower bound on the expected number of steps for any comparison-based algorithm, including Las Vegas algorithms, is:

$$\Omega(\log n)$$

So, in conclusion the expected number of steps taken by a Las Vegas randomized algorithm to determine whether a given key k is present in a sorted array A of n numbers is $\Theta(\log n)$.

Problem 3.3 [RA_Mot]

A parallel computer consists of n processors and n memory modules. During a step, each processor sends a memory request to one of the memory modules. A memory module that receives either one or two requests can satisfy its request(s): modules that receive more than two requests will satisfy two requests and discard the rest.

(a) Assuming that each processor chooses a memory module independently and uniformly at random, what is the expected number of processors whose requests are satisfied? Use the approximation $(1 - 1/n)^n \simeq 1/e$ if necessary. (b) Repeat the computation for the case where each memory module can satisfy only one request during a step.

Answer

Part (a)

Assume that each processor chooses a memory module independently and uniformly at random. We want to find the expected number of processors whose requests are satisfied.

Let X_i be the number of processors that request memory module i. Then $X_i \sim \text{Binomial}(n, 1/n)$.

Let Y_i be the indicator random variable that is 1 if memory module i satisfies one or two requests, and 0 otherwise. Then,

$$Y_i = \begin{cases} 1 & \text{if } X_i = 1 \text{ or } X_i = 2 \\ 0 & \text{otherwise} \end{cases}$$

The expected number of satisfied processors is:

$$\mathbb{E}\left[\sum_{i=1}^{n} Y_i\right] = \sum_{i=1}^{n} \mathbb{E}[Y_i]$$

To compute $\mathbb{E}[Y_i]$:

$$\mathbb{E}[Y_i] = \Pr(X_i = 1) + \Pr(X_i = 2)$$

Using the binomial probability formula:

$$\Pr(X_i = 1) = \binom{n}{1}\left(\frac{1}{n}\right)^1\left(1-\frac{1}{n}\right)^{n-1} = n \cdot \frac{1}{n} \cdot \left(1-\frac{1}{n}\right)^{n-1} = \left(1-\frac{1}{n}\right)^{n-1}$$

$$\Pr(X_i = 2) = \binom{n}{2}\left(\frac{1}{n}\right)^2\left(1-\frac{1}{n}\right)^{n-2} = \frac{n(n-1)}{2} \cdot \frac{1}{n^2} \cdot \left(1-\frac{1}{n}\right)^{n-2} = \frac{(n-1)}{2n} \cdot$$

Using the approximation $(1-\frac{1}{n})^n \approx \frac{1}{e}$:

$$\Pr(X_i = 1) \approx \frac{1}{e}$$

$$\Pr(X_i = 2) \approx \frac{(n-1)}{2n} \cdot \frac{1}{e} \approx \frac{1}{2e}$$

Thus,

$$\mathbb{E}[Y_i] \approx \frac{1}{e} + \frac{1}{2e} = \frac{3}{2e}$$

The expected total number of processors whose requests are satisfied is:

$$\mathbb{E}\left[\sum_{i=1}^{n} Y_i\right] = n \cdot \mathbb{E}[Y_i] \approx n \cdot \frac{3}{2e} = \frac{3n}{2e}$$

Part (b)

Now, assume each memory module can satisfy only one request during a step.

Let X_i be the number of processors that request memory module i, and let Z_i be the indicator random variable that is 1 if memory module i satisfies exactly one request and 0 otherwise. Then,

$$Z_i = \begin{cases} 1 & \text{if } X_i = 1 \\ 0 & \text{otherwise} \end{cases}$$

The expected number of satisfied processors is:

$$\mathbb{E}\left[\sum_{i=1}^{n} Z_i\right] = \sum_{i=1}^{n} \mathbb{E}[Z_i]$$

To compute $\mathbb{E}[Z_i]$:

$$\mathbb{E}[Z_i] = \Pr(X_i = 1)$$

From above, we have:

$$\Pr(X_i = 1) \approx \frac{1}{e}$$

Therefore,

$$\mathbb{E}[Z_i] = \frac{1}{e}$$

The expected total number of processors whose requests are satisfied is:

$$\mathbb{E}\left[\sum_{i=1}^{n} Z_i\right] = n \cdot \mathbb{E}[Z_i] \approx n \cdot \frac{1}{e} = \frac{n}{e}$$

Problem 3.4 [RA_Mot]

Consider the following experiment, which proceeds in a sequence of rounds. For the first round, we have n balls, which are thrown independently and uniformly at random into n bins. After round i, for $i \geq 1$, we discard every ball that fell into a bin by itself in round i. The remaining balls are retained for round $i + 1$, in which they are thrown independently and uniformly at random into the n bins. Show that there is a constant c such that with probability $1 - O(1)$, the number of rounds is at most $cloglogn$

Answer

We consider an experiment with n balls and n bins. In each round, balls are thrown independently and uniformly at random into the bins. Balls that fall alone in a bin are discarded, and the remaining balls proceed to the next round. Let's see if with high probability, the number of rounds is at most $c \log \log n$ for some constant c!

Expected Number of Balls After Each Round

Let B_i denote the number of balls remaining after round i.

The probability that a specific bin contains exactly one ball is:

$$\Pr(\text{bin } i \text{ has exactly one ball}) = B\left(\frac{1}{n}\right)\left(1 - \frac{1}{n}\right)^{B-1} \approx \frac{B}{n} e^{-\frac{B}{n}}$$

The expected number of balls that are not alone in a bin is:

$$\mathbb{E}[B_{i+1}] \approx B_i \left(1 - \frac{B_i}{n} e^{-\frac{B_i}{n}}\right)$$

For large B_i, we approximate:

$$B_{i+1} \approx B_i \left(1 - \frac{B_i}{n}\right)$$

Using the exponential decay:

$$B_i \approx n\left(\frac{1}{e}\right)^i$$

To have $B_i \leq 1$:

$$n\left(\frac{1}{e}\right)^i \leq 1$$

Taking the logarithm:

$$\log n - i \log e \leq 0,$$

$$i \geq \log n.$$

Thus, $i \approx \log n$. Considering the exponential decay, the number of rounds R is:

$$R = O(\log \log n).$$

Conclusion

There exists a constant c such that with probability $1 - O(1)$, the number of rounds is at most $c \log \log n$.

Problem 3.10 [RA_Mot]

The sharp threshold result in the coupon collector's problem does not imply that the probability of needing more than $cnlogn$ trials goes to zero at a doubly exponential rate if c were not a constant, but were allowed to grow with n. Let the probability of requiring more than $cnlogn$ trials be $p(c)$. For constant c, show that $1/p(c)$ can be bounded from above and below by polynomials in n.

Answer

We analyze the probability $p(c)$ of requiring more than $cn \log n$ trials in the coupon collector's problem and show that $\frac{1}{p(c)}$ can be bounded by polynomials in n.

Let X be the random variable representing the number of trials needed to collect all n coupons. Using the Central Limit Theorem:

$$X \sim \mathcal{N}(n \log n, n(\log n)^2)$$

We standardize to find $p(c)$:

$$p(c) = \Pr\left(\frac{X - n \log n}{\sqrt{n} \log n} > \frac{cn \log n - n \log n}{\sqrt{n} \log n}\right) = \Pr\left(Z > \sqrt{n}(c-1)\right),$$

where Z is a standard normal variable.

Using the normal tail bound for $z > 0$:

$$\Pr(Z > z) \approx \frac{1}{\sqrt{2\pi}z} e^{-z^2/2}$$

Thus, for large n:

$$p(c) \approx \frac{1}{\sqrt{2\pi\sqrt{n}(c-1)}} e^{-n(c-1)^2/2}$$

Taking the reciprocal of $p(c)$:

$$\frac{1}{p(c)} \approx \sqrt{2\pi\sqrt{n}(c-1)} e^{n(c-1)^2/2}$$

To show $\frac{1}{p(c)}$ can be bounded by polynomials in n:

\textbf{Lower Bound}: For large n, $\frac{1}{p(c)} \geq n^{(c-1)^2/2}$. \textbf{Upper Bound}: Similarly, $\frac{1}{p(c)} \leq n^{(c-1)^2/2+\epsilon}$ for a small positive ϵ.

Thus, there exist constants k_1 and k_2 such that:

$$n^{k_1} \leq \frac{1}{p(c)} \leq n^{k_2},$$

where k_1 and k_2 are related to $(c-1)^2/2$.

We have shown that $\frac{1}{p(c)}$ can be bounded from above and below by polynomials in n.

Problem 3.15 [RA_Mot]

(Due to D. Angluin and L.G. Valiant [28].) Let B denote a random bipartite graph with n vertices in each of the vertex sets U and V. Each possible edge, independently, is present with probability p(n). Consider the following algorithm for constructing a perfect matching (see Section 7.3) in such a random graph. Modify the Proposal Algorithm of Section 3.5 as follows. Each $u \epsilon U$ can propose only to adjacent $v \epsilon V$. A vertex $v \epsilon V$ always accepts a proposal, and if a proposal causes a "divorce," then the newly divorced $u \epsilon U$ is the next to propose. The sampling procedure outlined in Problem 3.14 helps implement the Principle of Deferred Decisions. How small can you make the value of $p(n)$ and still have the algorithm succeed with high probability? The following fact concerning the degree $d(v)$ of a vertex v in B proves useful:

$$\Pr[d(v) \leq (1-\beta)np] = O\left(e^{-\beta^2 np/2}\right)$$

Bs: The Proposal Algorithm is the algorithm in Stable Marriage problem and it is summarized as "man proposes, woman disposes".

Answer

Let B denote a random bipartite graph with n vertices in each of the vertex sets U and V. Each possible edge, independently, is present with probability $p(n)$. We analyze the performance of a modified Proposal Algorithm to find a perfect matching in B and determine the smallest $p(n)$ such that the algorithm succeeds with high probability.

Each $u \in U$ proposes only to adjacent $v \in V$. A vertex $v \in V$ always accepts a proposal, and if a proposal causes a "divorce," the newly divorced u is the next to propose.

The following inequality proves useful in our analysis:

$$\Pr[d(v) \leq (1-\beta)np] = O\left(e^{-\beta^2 np/2}\right)$$

The expected degree of a vertex $v \in V$ is np.

We want the probability of $d(v)$ being too small to be $o(1/n)$. Using the given inequality:

$$\Pr[d(v) \leq (1-\beta)np] = O\left(e^{-\beta^2 np/2}\right)$$

we need:

$$O\left(e^{-\beta^2 np/2}\right) \leq \frac{1}{n}$$

Taking the natural logarithm of both sides:

$$-\frac{\beta^2 np}{2} \leq -\log n$$

$$\frac{\beta^2 np}{2} \geq \log n$$

$$np \geq \frac{2\log n}{\beta^2}$$

Choosing $\beta = 1$, we get:

$$p(n) \geq \frac{2\log n}{n}$$