CS526-HW3

Zongqi Cui

October 2024

Problem 1

According to the problem, we can build the payoff matrix as follows:

$$\begin{pmatrix} 3 & 1 & 1 & 1 \\ \frac{3}{2} & 2 & 2 & 1 \\ 1 & \frac{4}{3} & \frac{5}{3} & 1 \\ 1 & \frac{4}{3} & \frac{5}{3} & 2 \end{pmatrix}$$

(a) If we are the first to announce the strategy, we must have $z = \max_d \mathbb{E}_t[r(t, d)]$. Therefore, we have the following system of inequalities:

$$\begin{aligned}
&\max z \\
&z \geq 3q_1 + q_2 + q_3 + q_4 \\
&z \geq \frac{3}{2}q_1 + 2q_2 + q_3 + q_4 \\
&z \geq q_1 + \frac{4}{3}q_2 + \frac{5}{3}q_3 + q_4 \\
&z \geq q_1 + \frac{4}{3}q_2 + \frac{5}{3}q_3 + 2q_4 \\
&q_1 + q_2 + q_3 + q_4 = 1
\end{aligned}$$

We divide z for all the equations and substitute q_i's with x_i's, resulting in the system:

$$\begin{aligned}
&1 \geq 3x_1 + x_2 + x_3 + x_4 \\
&1 \geq \frac{3}{2}x_1 + 2x_2 + x_3 + x_4 \\
&1 \geq x_1 + \frac{4}{3}x_2 + \frac{5}{3}x_3 + x_4 \\
&1 \geq x_1 + \frac{4}{3}x_2 + \frac{5}{3}x_3 + 2x_4
\end{aligned}$$

Using the `optimize` function from the `scipy` package to solve this LP problem:

```
from scipy import optimize as opt
import numpy as np

c = np.array([1, 1, 1, 1])
a = np.array([[3, 1, 1, 1], [1.5, 2, 1, 1], [1, 4/3, 5/3, 1], [1, 4/3, 5/3, 2]])
b = np.array([1, 1, 1, 1])

ans = opt.linprog(-c, a, b)
p = 1 / sum(ans['x'])
ans_list = [i * p for i in ans['x']]
print(ans_list)
print(sum(ans_list))
```

The probabilities for choosing each number are:

$$p_1 \approx 21.05\%, \quad p_2 \approx 31.58\%, \quad p_3 \approx 47.37\%, \quad p_4 \approx 0\%$$

we can find that $min_P = 1.42$.
(b) For $\min_d \mathbb{E}_d[r(t,d)]$, after transformation, we get:

$$\begin{aligned}
1 &\leq 3x_1 + \frac{3}{2}x_2 + x_3 + x_4 \\
1 &\leq x_1 + 2x_2 + \frac{4}{3}x_3 + \frac{4}{3}x_4 \\
1 &\leq x_1 + x_2 + \frac{5}{3}x_3 + \frac{5}{3}x_4 \\
1 &\leq x_1 + x_2 + x_3 + 2x_4
\end{aligned}$$

Using similar code as above, the probabilities are:

$$q_1 \approx 0\%, \quad q_2 \approx 0\%, \quad q_3 \approx 1\%, \quad q_4 \approx 0\%$$

$$P = \frac{5}{3}$$

Problem 2

We aim to avoid two cases:

- **Bad1**: Some points are not covered. $\Pr[\text{Bad1}] \geq n \cdot \Pr[\text{a is not covered}] = \left(\frac{1}{e}\right)^c$
- **Bad2**: The cost C is too large.

$$\Pr[\text{Bad2}] = \Pr[\text{cost}(C) > (1+\epsilon)(\ln n) \cdot \text{OPT}_f] \leq \frac{\ln n + c}{(1+\epsilon) \cdot \ln n}$$

We want Pr[Bad1] to be small and Pr[Bad2] close to 1. Set:

$$\Pr[\text{Bad1}] < \frac{\epsilon}{8}, \quad \Pr[\text{Bad2}] < 1 - \frac{\epsilon}{4}$$

Thus, Pr[Success] $> \frac{\epsilon}{8}$. After repeating the process $\frac{8}{\epsilon}$ times:

$$\Pr[\text{no success}] < \left(1 - \frac{\epsilon}{8}\right)^{\frac{8}{\epsilon}} \leq \frac{1}{e} < \frac{1}{2}$$

Therefore, there is at least a 50% chance of finding a successful C.

Problem 3

(a) Let $A \subseteq B \subseteq \{1, 2, \ldots, m\}$ be two sets of indices and let $x \notin B$. We need to show that:

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$$

The function $f(I) = \left|\bigcup_{i \in I} S_i\right|$ gives the size of the union of sets S_i indexed by I. This implies that f measures the number of distinct elements in the union of the sets.

Now, consider the following: - $f(A \cup \{x\}) - f(A)$ represents the number of new elements introduced by adding the set S_x to the union of sets indexed by A. - Similarly, $f(B \cup \{x\}) - f(B)$ represents the number of new elements introduced by adding S_x to the union of sets indexed by B.

Since $A \subseteq B$, the union $\bigcup_{i \in B} S_i$ already contains at least as many elements as $\bigcup_{i \in A} S_i$. Therefore, the contribution of S_x (i.e., the number of new elements introduced by adding S_x) will be greater when added to A than when added to B. In other words:

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$$

Thus, the function f satisfies the diminishing returns property, and we conclude that f is submodular.

(b) The idea behind the code is simple. For each iteration, union each item in set T with the previous result and choose the maximum combination. Repeat K times to ensure there are K items in the final result.

(c) In the worst case, function f needs to be called $O(m \cdot k)$ times.

(d) Set $n_t = \text{OPT}_k - f(I)$ after t iterations. Since the covered numbers are in OPT_k, there must be n_t/k sets. Hence:

$$n_{t+1} \leq n_t - \frac{n_t}{k}$$

Thus:

$$n_t \leq \text{OPT}_k \cdot \left(1 - \frac{1}{k}\right)^t$$

Transforming, we get:

$$f(I) \geq \left[1 - \left(1 - \frac{1}{k}\right)^k\right] \cdot \text{OPT}_k$$

Algorithm 1 Greedy Algorithm for Problem 3.b

```
T = S_1 + S_2 + ... + S_i
solu ← 0
solu_set ← ∅
curr_max ← ∅
for i ← 1, k do
    for t ∈ T_1, T_last do
        if solu ≤ f(solu_set ∪ t) then
            curr_max ← t
        end if
    end for
    solu ← f(solu_set ∪ curr_max)
    solu_set ← solu_set ∪ curr_max
    curr_max ← ∅
    T ← T \ curr_max
end for
```

Problem 4

(a) Let x_i represent choosing an actor and y_j represent choosing an investor. The profit for each investor is represented as $y_j p_j - x_i s_i$. The ILP is formulated as:

$$\max \left(\sum y_j p_j - \sum x_i s_i \right)$$

subject to:

$$y_j \leq x_i \quad \forall j \in [1, n], i \in L_j, \quad x_i, y_j \in \{0, 1\}$$

(b) If there is a non-integer solution, a better integer solution exists. If all $x_i < 1$ and the total profit is negative, set $x_i = 0$. If x_i is between 0 and 1, let $r = \max x_i$ and multiply x_i by $1/r$ until all $x_i = 1$.

Problem 5

(a) Start with all vertices having a "fattest path capacity" of 0, except for the source s, which starts with infinity (to represent no restriction at the beginning). For each vertex v, try to relax the fattest path by updating the capacity of its neighbors along the edges.
Use a max-priority queue (or a binary heap) to always expand the vertex with the largest current path capacity.
The algorithm terminates when the sink t is reached. In Dijkstra's algorithm, we minimize the distance (or cost) to each vertex, while in this algorithm, we maximize the capacity of the smallest edge along a path. The priority is based on capacity, not cost.

(b) To find the maximum flow of a graph G, this can be transformed into finding the minimum cut. The number of edges in the minimum cut is at most $|E|$, and the maximum flow is the sum of edges in the minimum cut. So we can conclude that the maximum flow for G will have at most $|E|$ edges.

(c) The number of edges in the maximum flow cannot exceed $|E|$. After increasing flow F, we could get a equation like this

$$F_{t+1} \leq F_t - \frac{F_t}{|E|}$$

, and the run time is $O(|E| \cdot \log F)$.