CS 6150: HW 6 – Optimization formulations, review

Submission date: Tuesday, December 14, 2021, 11:59 PM

This assignment has 5 questions, for a total of 50 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

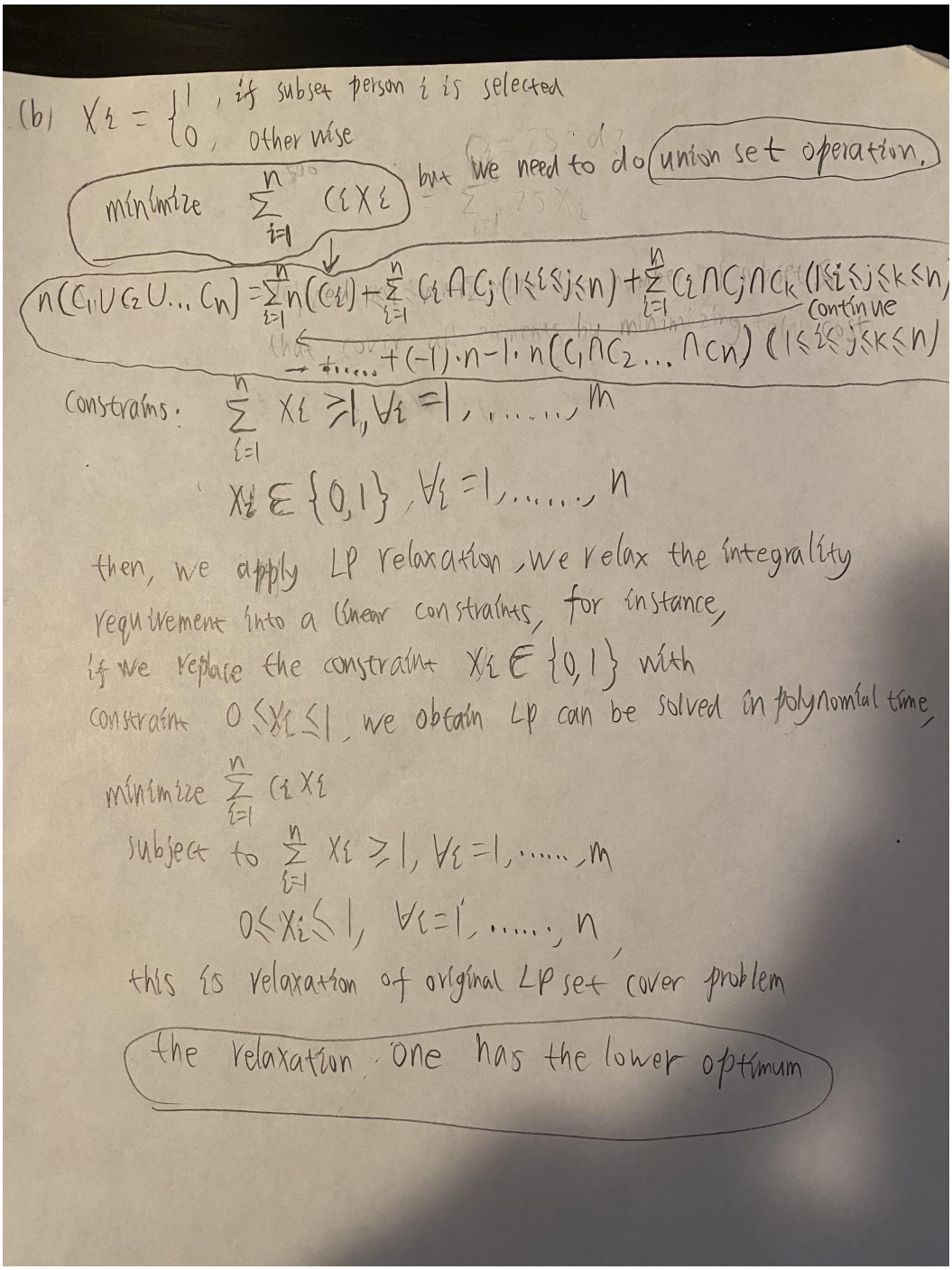
Question	Points	Score
Understanding relax-and-round	16	
The power of two choices	10	
Optimal packaging	10	
Non-negativity in Markov	4	
Distributed independent set	10	
Total:	50	

Question 1: Understanding relax-and-round . . . . . **[16]**

In this question, you will implement the relax and round paradigm using the example of the Set Cover (or hiring) problem. Suppose we have n people (i.e., sets) and m skills that we wish to cover. Let us create an instance with $n = m = 500$. Let $d = 25$ be the size of each skill set.

(a) **[3]** Write code that generates a random instance of set cover, where for each person i, the skill set S_i is a random subset of the m skills, with $|S_i| = d$.
Answer:
https://github.com/prosperousZ/HW6Q1.git
When you run the code, be sure to match the same package name and class name, I am using Java for this question. After you run the code, the 500 arrays will be in Console. I also upload this text file at github, same folder.

(b) **[3]** Write down the integer linear program using variables x_i that indicate if person i is chosen/hired (abstractly, as we did in class). Then write down the linear programming relaxation. Which one has the lower optimum objective value?
Answer:



(c) **[6]** For the instance you created in part (a), solve the linear program from part (b) using an LP solver of your choice, and output the fractional solution.
Answer:
https://github.com/prosperousZ/HW6Q1.git
This github address also include text file with a list of 500 variables and values.

(d) **[4]** Round the fractional solution using randomized rounding, i.e., hire person i with probability $\min(1, tx_i)$.

Try $t = 1, 2, 4, 8$, and in each case, report the (a) total number of people hired, and (b) number of "uncovered" skills (i.e., skills for which none of the people possessing the skill were hired).

Answer:

When t = 1, after I run the result, I have total number of people hired = 51, number of "uncovered" skills = 183. When t = 2, total number of people hired = 93, number of "uncovered" skills = 156. When t = 4, I have total number of people hired = 142, number of "uncovered" skills = 118. When t = 8, I have total number of people hired = 206, number of "uncovered" skills = 69.

Question 2: The power of two choices . . . . . **[10]**

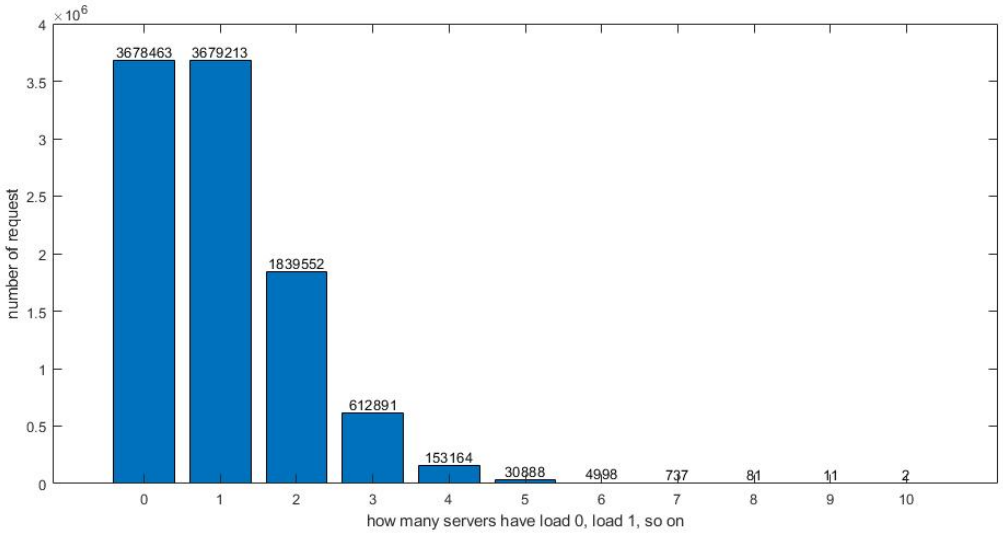
As I mentioned in class, one of the classic applications of random hashing is "on-the-fly" load balancing. In this problem, we will empirically see how "balanced" such an assignment is, and how to make it more balanced.

In what follows, set $N = 10^7$, i.e., 10 million. Suppose we have N servers, and N service requests arrive sequentially.

(a) **[4]** When a request arrives, suppose we generate a random index r between 1 and N and send the request to server r (and we do this independently for each request). Plot a histogram showing the distribution of the "loads" of the servers in the end. I.e., show how many servers have load 0, load 1, and so on. [The load is defined as the number of requests routed to that server.]

Answer:

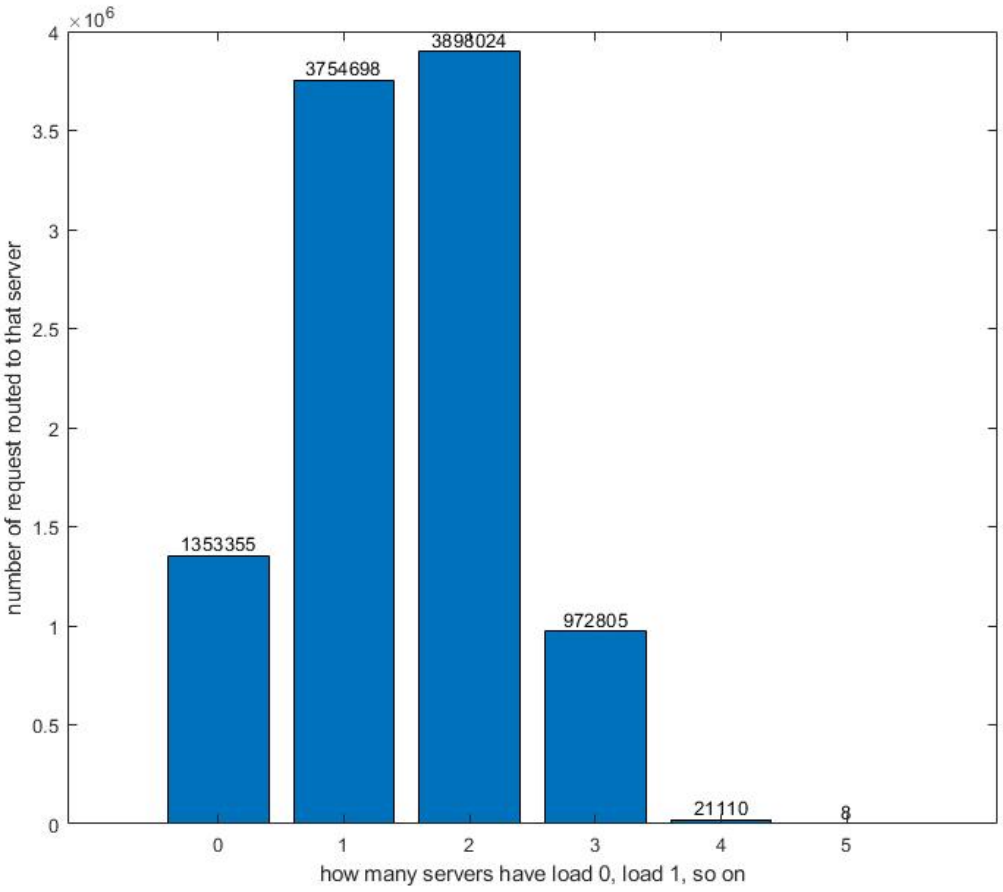
I am using matlab for this question, https://github.com/prosperousZ/HW6Q1.git



(b) **[6]** Now, suppose we do something slightly smarter: when a request arrives, we generate two random indices r_1 and r_2 between 1 and N, query to find the current load on the servers r_1 and r_2, and assign the request to the server with the lesser load (breaking ties arbitrarily). With this allocation, plot the histogram showing the load distribution, as above.

Answer:

I am using matlab for this question, https://github.com/prosperousZ/HW6Q1.git



Question 3: Optimal packaging . . . . . **[10]**

With the holiday season around the corner, company Bezo wants to minimize the number of shipping boxes. Let us consider the one dimensional version of the problem: suppose a customer orders n items of lengths $a_1, a_2, \ldots, a_n$ respectively, and suppose $0 < a_i \leq 1$. The goal is to place them into boxes of length 1 such that the total **number of boxes** is minimized.

It turns out that this is a rather difficult problem. But now, suppose that there are only r distinct values that the lengths could take. In other words, suppose that there is some set $L = \{s_1, \ldots, s_r\}$ such that every $a_i \in L$. Let us think of r is as a small constant. Devise an algorithm that runs in time $O(n^r)$, and computes the optimal number of boxes.

[Hint: first find all the possible "configurations" that can fit in a single box. Then use dynamic programming.]

Code was update to github, run in Java if you want, https://github.com/prosperousZ/HW6Q1.git
Input: r distinct values, means length of L. $L = \{s_1, \ldots, s_r\}$ every $a_i \in L$, r is small, $0 < a_i \leq 1$, n items, big box length of 1, which is sum = 1.
Output: run in time $O(n^r)$, and compute the total number of boxed is minimized.

Variables: Given L[ ] $= \{s_1, \ldots, s_r\}$, let us call it arrA [ ] $= \{s_1, \ldots, s_r\}$, sum = 1, initial currentSum = 0, start = 0, declare a combinationList = null, also declare arrayList named possibleList = empty, which store all possible "configurations" that fit in a single box, using dynamic programming

```
function MAIN
    //call main method first
    arrA [ ] = {s_1, ..., s_r}
    sum = 1
    findSets(arrA, sum)
end function

function FINDSETS(arrA [ ], sum)
    sort(arrA) in ascending order
    List<> combinationList = new ArrayList<> ()
    List<int [ ]> possibleList = new ArrayList<> ()
    combinationUtil(arrA [ ], sum, 0, 0, combinationList, possibleList)
end function

function COMBINATIONUTIL(arrA [ ], sum, currentSum, start, combinationList, possibleList)
    if (currentSum == sum) then
        print out combinationList, this is base case
        possibleList.add(combinationList.toArray())
        return
    end if
    int prevElement = -1
    for (int i = start; i < arrA.length; i ++) do
        if (prevElement != arrA[i]) then
            if (currentSum + arrA[i] > sum) then
                break
                //because array is sorted, no need to check further
            end if
            combinationList.add(arrA[i])
            combinationUtil(arrA, sum, currentSum + arrA[i], i + 1, combinationList, possibleList)
            combinationList.remove(combinationList.size() - 1)
            prevElement = arrA[i]
        end if
    end for
end function
```

For example, let us x10 of each number, we have sum = 10, if arrA = [6, 7, 8, 2, 4, 1, 3, 5, 10], required sum = 10, we have output [1, 2, 3, 4], [1, 2, 7], [1, 3, 6], [1, 4, 5], [2, 3, 5],[2, 8], [3, 7], [4, 6], [10], possibleList has the order of most to less which you can see from the output. Output checks all possible "configurations" that fit in a single box.

Then, check total number of boxes that is minimize, we have n items, with List<>() itemList = = $\{a_1, a_2, \ldots, a_n\}$, this is new variable we need to use, possibleList has all the possible configurations, next step is to modify somthing in the code, we have

```
function MAIN
    call main method first
    arrA [ ] = {s_1, ..., s_r}
    List<>() itemList = {a_1, a_2, ..., a_n}
    sum = 1
    findSets(arrA, itemList, sum)
end function
```

```
function FINDSETS(arrA [ ], itemList, sum)
    sort(arrA) in ascending order
    List<> combinationList = new ArrayList<> ()
    //type of possibleList is a list of array
    List<int [ ]> possibleList = new ArrayList<> ()
    combinationUtil(arrA [ ], sum, 0, 0, combinationList, possibleList)
    //Now, possibleList have all the possible "configuration",
    //we will need to go through every possible "configuration" in itemList
    minNumberOfBoxes = 0
    for (int j = 0; j < possibleList.size(); j ++) do
        declear an array called oneOfPossible [ ] = possibleList.get(j)
        boolean b = true
        while (b == true) do
            for (int q = 0; q < oneOfPossible.length; q ++) do
                if (oneOfPossible[q] in itemList == true) then
                    continue;
                    //need to make sure every element in oneOfPossible [ ] is in itemList
                end if
                if (oneOfPossible[q] in itemList == false) then
                    b = false;
                    break;
                end if
            end for
            if (b == true) then
                //Since every item in oneOfPossible[ ], itemList will have the same items,
                delete these items in itemList
                // mean put these items in one box
                minNumberOfBoxes ++
            end if
        end while
    end for
    // Since all of the possible combination of items that could place into boxes of length 1 was placed.
    // And we delete these items
    // the rest will have to be one item one box, so we have
    minNumberOfBoxes = minNumberOfBoxes + itemList.size()
    return minNumberOfBoxes
end function

function COMBINATIONUTIL(arrA [ ], sum, currentSum, start, combinationList, possibleList)
    if (currentSum == sum) then
        print out combinationList, this is base case
        possibleList.add(combinationList.toArray())
        return
    end if
    int prevElement = -1
    for (int i = start; i < arrA.length; i ++) do
        if (prevElement != arrA[i]) then
            if (currentSum + arrA[i] > sum) then
                break; because array is sorted, no need to check further
            end if
            combinationList.add(arrA[i])
            combinationUtil(arrA, sum, currentSum + arrA[i], i + 1, combinationList, possibleList)
```

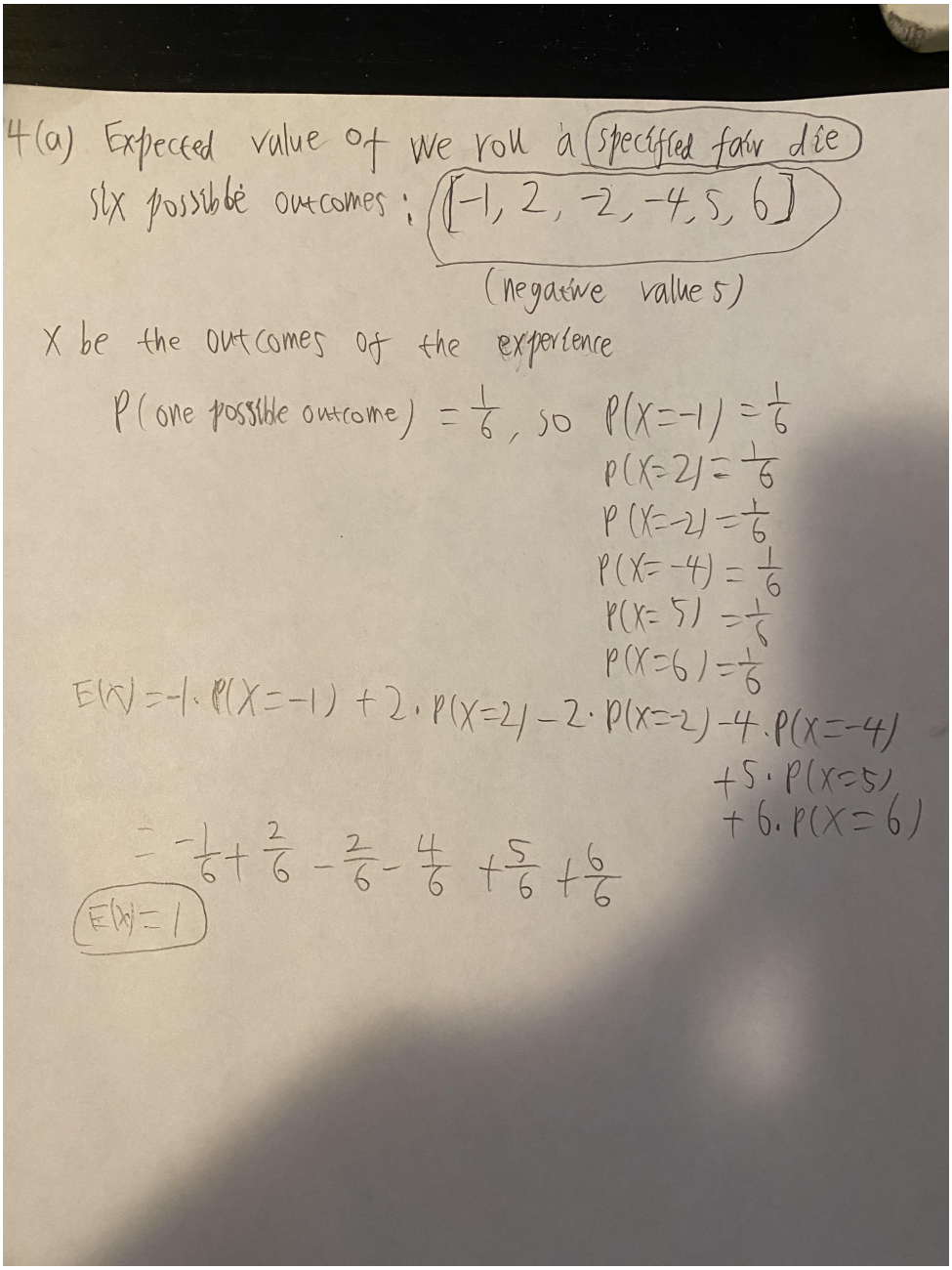
```
            combinationList.remove(combinationList.size() - 1)
            prevElement = arrA[i]
        end if
    end for
end function
```

Running time: From my algorithm, as I said, we need to go through every possible combination, with n items, so that we get running time is $O(n^r)$, and I am pretty sure I use dp in my algorithm.

Question 4: Non-negativity in Markov . . . . . **[4]**

Markov's inequality states that for a non-negative random variable X, we have $\Pr[X > t \cdot \mathbb{E}[X]] \leq 1/t$, for any $t \geq 1$.

The point of this exercise is to show that the non-negativity is important. Give an example of a random variable (that takes negative values), for which (a) $\mathbb{E}[X] = 1$, and (b) $\Pr[X > 5] \geq 0.9$.



(b) $\Pr[X > 5] \geq 0.9$, given an array of number, that [-1,0,1,2,3,4,5,6,...,67,68] which length of array is 70, array has negative number. Pick one number, which is X from this array, Pr[X $>$ 5] = 0.9, extend the array from 68, for example, [-1,0,1,2,3,4,5,6,...,67,68,....,n], n $>=$ 68, $n \in integer$, we will have $\Pr[X > 5] \geq 0.9$.

Question 5: Distributed independent set . . . . . **[10]**

A fundamental problem in distributed algorithms (used in P2P networks, distributed coloring, etc.) is the following: we are given an undirected graph with n vertices where each vertex corresponds to an 'agent'. The goal is to find a large independent set (a set of vertices with no edges between them) in a distributed manner. It turns out that this problem has a nice solution when the degree of each vertex is $\leq d$, for some known parameter d.

Consider the following algorithm. (1) Every vertex becomes active with probability $\frac{1}{2d}$. (2) Every active vertex queries its neighbors, and if any vertex in the neighborhood is also active, it becomes inactive. (Step (2) is done

in parallel; thus if i and j are neighbors and they were both activated in step (1), they both become inactive.) (3) The set of active vertices in the end is output as the independent set.

(a) **[2]** Let X be the random variable that is the number of vertices activated in step (1). Find $\mathbb{E}[X]$.

(b) **[3]** Let Y be the random variable that is the number of edges $\{i, j\}$ both of whose end points are activated in step (1). Find $\mathbb{E}[Y]$ (in terms of m, the total number of edges in the graph).

(c) **[5]** Prove that the size of the independent set output in (3) is at least $X - 2Y$, and thus show that the expectation of this quantity is $\geq n/4d$.

Answer: