Homework 6

Jack Wilburn

Dec 10th, 2021

1 Q1: Understanding relax-and-round

1.1 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 $|S_i| = d$.

Done and submitted on canvas (lines 6-15).

1.2 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?

For integer programming, each $x_i \in (0, 1)$. The linear programming relaxation is $0 \leq x_i \leq 1$. The optimal value of the linear programming relaxation is less than or equal to the optimal solution for the integer linear programming scenario. That's because in the relaxation, we have a larger solution space that we can explore for minimums.

1.3 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.

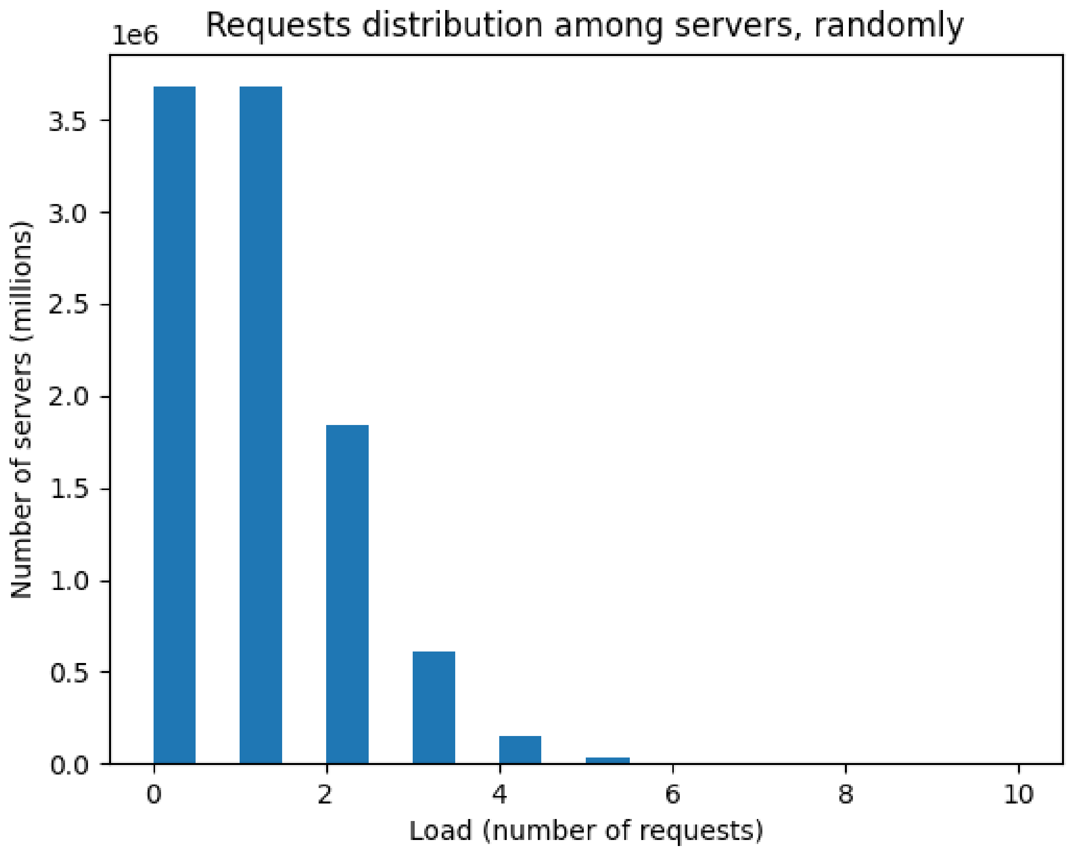
See lines 6-51 in the submitted code on canvas. The optimal value found by the solver was 21.59059202859. The 500 variable coefficients can be found in LPoutput.txt on canvas.

1.4 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).

$t = 1$:
(a) 24 people were hired (b) 146 skills were uncovered
$t = 2$:
(a) 47 people were hired (b) 42 skills were uncovered
$t = 4$:
(a) 88 people were hired (b) 1 skill was uncovered
$t = 8$:
(a) 165 people were hired (b) 0 skills were uncovered

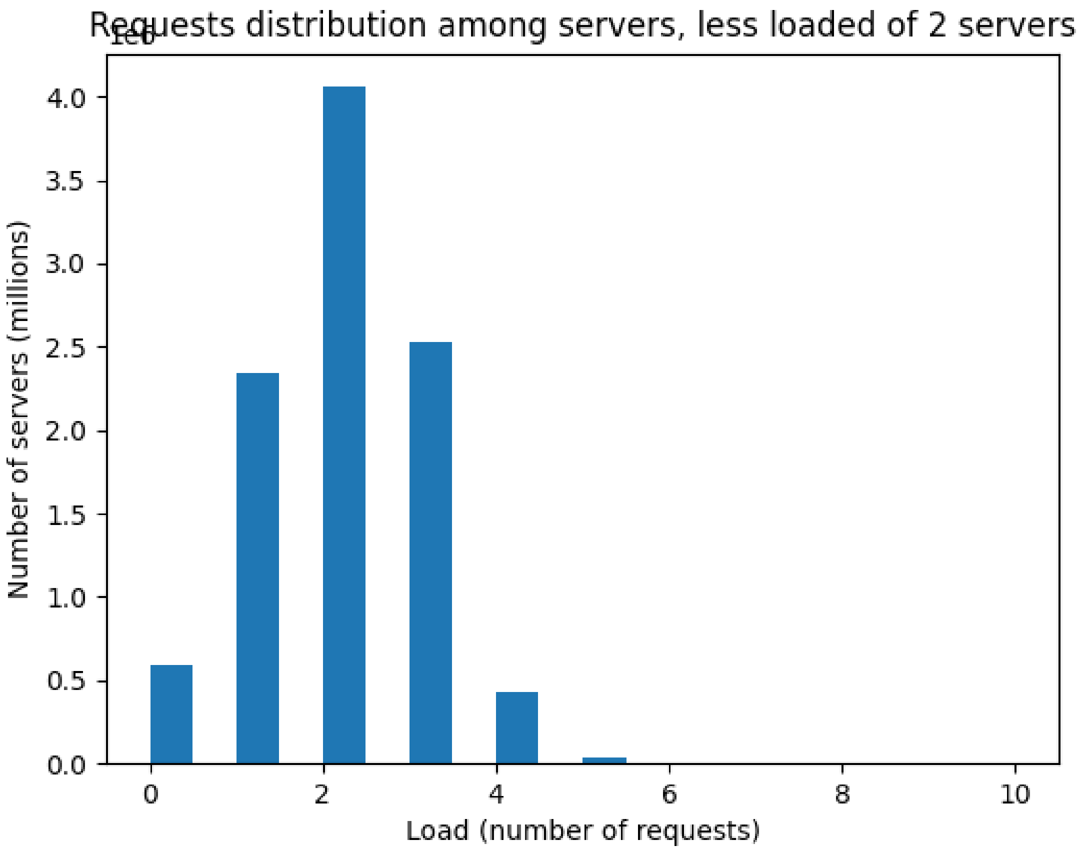
2 Q2: The power of two choices

2.1 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.]



You can see in this scenario that approximately 3.5 million servers are sat idle, while the other 6.5 million are serving one or more requests.

2.2 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.



In this case, only approximately 500,000 servers are sat idle, while the other 9.5 million are serving requests.

3 Q3: Optimal packaging

3.1 Devise an algorithm that runs in time $O(n^r)$, and computes the optimal number of boxes.

Pseudocode

1. Enumerate all of the combinations of sizes that would fit inside a box of length 1, including using duplicates of certain sizes. For all of the ones that do, add them to a new array F.
2. Start with 0 boxes
3. Try each possible box packing $f_i \in F$ and see if we have enough packages of the required sizes to fit in the box. If so, add one to the number of boxes used and remove the number packages that would fit inside the box from consideration. If the we've already seen the remaining combination of packages before use that answer instead of recomputing it.
4. Repeat 3 until there are no more packages. to ship.
5. Find the smallest number of boxes required.

Correctness

This is an exhaustive search using all possible box combinations. Instead of testing each package added to a box individually, we're instead testing combinations of packages in boxes to find the optimal packing.

Running Time

The first step takes time in r (something like r^r), which assuming r is small is insignificant for the overall run time.

The majority of the time spent by this algorithm will be in the DP section of the algorithm.

4 Q4: Non-negativity in Markov

4.1 Give an example of a random variable (that takes negative values), for which (a) $E[X] = 1$ and (b) $\Pr[X > 5] \geq 0.9$.

Let X be defined as 0.9 probability of being 10 and 0.1 probability of being -80. The expected value is $E[X] = (0.9 * 10) + (0.1 * -80) = 1$, thus (a) is satisfied. (b) is satisfied by the definition of the variable, "0.9 probability of being 10", thus $\Pr[X > 5] \geq 0.9$.

5 Q5: Distributed independent set

5.1 Let X be the random variable that is the number of vertices activated in step (1). Find $E[X]$.

The expected value of X can be computed as follows. Let $E[X] = E[x_1] + E[x_2] + ... + E[x_n] = \frac{1}{2d} + \frac{1}{2d} + ... + \frac{1}{2d} = \frac{n}{2d}$. Basically each of the n vertices is activated with probability $\frac{1}{2d}$.

5.2 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 $E[Y]$ (in terms of m, the total number of edges in the graph).

The expected value of Y can be computed as follows. Let $E[Y] = \sum_{a=1}^{m} E[(i_a, j_a)] = \sum_{a=1}^{m} E[i_a] * E[j_a] = m * \dfrac{1}{2d} * \dfrac{1}{2d} = \dfrac{m}{4d^2}$, that is, for each of the m edges, the expected value of both nodes being activated.

5.3 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 $\geq n/4d$.

The number of active vertices when the algorithm gets to step 3 is $X - 2Y$, because we have X nodes that get activated and $2Y$ nodes that get deactivated by being activated neighbors. It's $-2Y$, because each edge in Y has 2 endpoints and Y is the count of edges.

Given that (3) is at least $X - 2Y$, I'll now show $E[X - 2Y] \geq \frac{n}{4d}$.

$E[X - 2Y] = \frac{n}{2d} - 2\frac{m}{4d^2} \geq \frac{n}{4d}$

Now looking to the right of that first equal sign, simplify and multiply both sides by 2d to get $n - \frac{m}{d} \geq \frac{n}{2}$

Subtract $\frac{n}{2}$ and add $\frac{m}{d}$ to yield $\frac{n}{2} \geq m/d$.

Now multiply by $2d$ again to get $nd \geq 2m$

At this point, we can just assert that this is true, because $nd = 2m$. nd is the count of out edges from every node, which counts each edge twice ($2m$).

Thus $E[X - 2Y] \geq \frac{n}{4d}$ and the expected size of set of independent vertices is $\geq E[X - 2Y] \geq \frac{n}{4d}$