CS 6150: HW5 – Randomized algorithms, optimization

Submission date: Monday, Nov 29, 2021 (11:59 PM)
Collaborators: Andrew Golightly, Thomas Burr, Taylor Allred, Alex Carpenteri, Brian Schnepp, Trey Lavery, Frost Office Hours, Yo Office Hours

points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Question	Points	Score
Trade-offs in sampling	12	
Satisfying ordering constraints	10	
Writing constraints	6	
LP Basics	10	
Identifying Corners	6	
Checking feasibility vs optimization	6	
Total:	50	

Instructions. For all problems in which you are asked to develop an algorithm, write down the pseudocode, along with a rough argument for correctness and an analysis of the running time (unless specified otherwise). Failure to do this may result in a penalty. If you are unsure how much detail to provide, please contact the instructors on Piazza.

Question 1: Trade-offs in sampling ..... **[12]**

For this problem, you need to run some basic experiments and write down the results you obtained. You **do not need to submit your code**, but if you prefer, you may add a publicly accessible link to the code (e.g., on github).

Suppose we have a population of size 10 million, and suppose 52% of them vote for A and 48% of them vote for B.

(a) [**4**] Randomly pick samples of size (a) 20, (b) 100, (c) 400, and evaluate the probability that A is majority even in your sample (by running the experiment say 100 times and taking the count of times A is the majority in your sample). Write down the values you observe for these probabilities in the cases (a-c).
Code link: https://gist.github.com/mozartfish/b5e8a2dba17ec1ba9c8fb19a51b4fe5c
a) sample size 20 - Percent A: 43%
b) sample size 100 - Percent A: 63%
c) sample size 400 - Percent A: 81%

(b) [**4**] What is the size of the sample you need for the probability above to become 0.9? (Your answer can be within 20% of the 'best' value.)
Sample size needed to achieve a probability of 0.9 with 100 trials: 900 elements

(c) [**4**] Suppose the population is more biased —55% of them vote for A and 45% of them vote for B— and re-solve part (b).
Sample size needed to achieve a probability of 0.9 with 100 trials: 175 elements
Note: This code was written with python and uses python's random number generator with randint(). According to the docs https://docs.python.org/3/library/random.html Python uses the Mersenne twister to generate random numbers so if a particular experiment is run enough times a specified target probability can be achieved.

Question 2: Satisfying ordering constraints ..... **[10]**

This problem is meant to illustrate a rather cool paradigm for solving problems, which is picking a *random solution*, and understanding how well it does. Indeed, there are many problems ("3-SAT", a version of boolean satisfiability, being the chief of them) where doing better than a random solution is actually NP-hard!

Suppose we have n elements, labeled $1, 2, \ldots, n$. Our goal is to find an ordering of these elements that satisfies as many "constraints" as possible, of the kind described below. We are given m constraints, where each constraint is given by a triple (a, b, c), where $a, b, c \in \{1, 2, \ldots, n\}$. The constraint is said to be *satisfied* if in the ordering we output, a does **not** lie "between" b and c (but the order between b and c doesn't matter). For example, if $n = 4$ and we consider the ordering 2431, then the constraint $(1, 4, 3)$ is satisfied, but $(3, 1, 2)$ is not.

Given the constraints, the goal is to find an ordering that satisfies as many constraints as possible (for simplicity, assume in what follows that m **is a multiple of** 3). For large m, n, this problem becomes very difficult.

(a) **[6]** As a baseline, let us consider a *uniformly random* ordering. What is the expected number of constraints that are satisfied by this ordering? [*Hint:* define appropriate random variables whose sum is the quantity of interest, and apply the linearity of expectation.]

Let us consider two values for elements n where $n = 3$ and $n = 4$. These represent the set of elements to be chosen for a triplet. For a triplet, we want a unique ordering of the digits so to calculate this we consider the permutation ${}_nP_3$. This yields the total number of constraints m which for $n = 3$ is 6 and for $n = 4$ is 24. Through trial and error for writing out the pattern for $n = 4$ $n = 5$, $n = 6$, $\frac{2}{3}$ of the constraints are valid and $\frac{1}{3}$ are invalid. For the ordering of (a, b, c) there are 6 different ways of arranging (a, b, c). Since we are only selecting 3 values for a triplet, we will always get $\frac{2}{3}$ of the constraints are valid regardless of what value of n is chosen.

(b) **[4]** Let X be the random variable which is the number of constraints satisfied by a random ordering, and let E denote its expectation (which we computed in part (a)). Now, Markov's inequality tells us, for example, that $\Pr[X \geq 2E] \leq 1/2$. But it does not directly imply that $\Pr[X \geq E]$ is "large enough" (which we need if we want to say that generating a few random orderings and picking the best one leads to many constraints being satisfied with high probability). Use the definition of X above to conclude that $\Pr[X \geq E] \geq 1/m$.
[*Hint:* X is a non-negative integer!]

Let m represent the number of total constraints
$\rightarrow E[x] = \int_0^\infty x \cdot f(x) = \int_0^{kE} x \cdot f(x)dx + \int_{kE}^\infty x \cdot f(x)dx$
$\rightarrow E[x] = \sum_{i=0}^{E-1} i \cdot Pr[X = i] + \sum_{i=E}^{m} i \cdot Pr[X = i]$
$\rightarrow 1 \leq \sum_{i=E}^{m} i \cdot Pr[X = i] \leq m \cdot Pr[X \geq E]$
$\rightarrow \frac{1}{m} \leq Pr[x \geq E]$

Question 3: Writing constraints ..... **[6]**

We will see how writing down constraints can be tricky when formulating problems as optimization.

Recall the traveling salesman problem (TSP): we are given a directed graph $G = (V, E)$ with edge lengths $w(e)$. The goal is to find a *directed cycle* that (a) visits every vertex exactly once, and (b) minimizes the total length (sum of lengths of the edges of the cycle).

Consider the following optimization formulation. We have variables $x_{uv} \in \{0, 1\}$, one for each edge (u, v) (remember that the graph is directed). The objective is to minimize $\sum_{(u,v)\in E} w(uv)x_{uv}$. The constraints are as follows: let $N_+(u)$ and $N_-(u)$ denote the out-neighbors and in-neighbors of u; then we consider the constraints:

$$\forall\ u, \quad \sum_{v \in N_+(u)} x_{uv} = 1,$$

$$\forall\ u, \quad \sum_{w \in N_-(u)} x_{wu} = 1.$$

(Intuitively, the constraints say that exactly one incoming edge and one outgoing edge must be chosen — as in a directed cycle.) Does an optimal solution to this optimization problem *always yield* an optimal solution to TSP? Provide a proof or a counterexample with a full explanation.

[*Hint:* consider a feasible solution to the optimization problem and focus on the edges with $x_{uv} = 1$. Do they have to form a *single* cycle?]

The constraints given in the problem allow for multiple cycles that can be disconnected because there has to be an incoming edge and outgoing edge for a node in the graph. Thus this Linear Program does not yield an optimal solution to the Traveling Salesman Problem because the TSP requires a single cycle.

Question 4: LP Basics ..... **[10]**

Consider the following two-variable linear program. The variables are $x, y \in \mathbb{R}$. And the constraints are:

$$\begin{aligned} 2x + y &\leq 10, \\ x &\geq 0, \\ y &\geq -1, \\ x + 3y &\leq 8. \end{aligned}$$

(a) **[4]** Plot the feasible region for the linear program. (Diagram can be drawn approximately to scale + scanned).
See attached photo with assignment

(b) **[2]** Suppose the objective function is to *maximize* $x + y$. Find the maximum value and the point at which this is attained.
The maximum point that satisfies the constraint is (4.4, 1.2)

(c) **[4]** Say the maximum value found in part (b) is C. Then could you have concluded that $x + y \leq C$ by just "looking at" the equations? [*Hint:* does adding equations, possibly with constants, yield a bound?]
We can define 4 equations as the following:
1) $2x + y \leq 10$
2) $x \geq 0$
3) $y \geq -1$
4) $x + 3y \leq 8$

We can ignore equations 2) 3) because they will not be affecting the bound.

Operation 1: Multiply Equation 1 by 2
1) $\rightarrow 4x + 2y \leq 20$ 2)$x + 3y \leq 8$

Operation 2: Add Equations 1 and equation 2
$5x + 5y \leq 28$
$x + y \leq \frac{28}{5}$
In the above operations, we have shown a bound according to the format in the problem $x + y \leq C$. Thus to achieve the maximum from part b, we can use the following equation
$x + y \leq \frac{28}{5}$

Question 5: Identifying Corners ..... **[6]**

Consider the following linear program, with variables $x_1, x_2, \ldots, x_n \in \mathbb{R}$. Suppose that the

constraints are:

$$0 \leq x_i \leq 1 \text{ for all } i,$$
$$x_1 + x_2 + \cdots + x_n \leq k,$$

where k is some integer between 1 and n. Consider any point $(a_1, a_2, \ldots, a_n)$ where $a_i \in \{0, 1\}$, and exactly k of the $\{a_i\}$ are 1. Prove that any such point is (a) a feasible point for the LP, and (b) a corner point of the polytope defining the feasible set.

Part A

In order for a be a feasible point, both constraints 1) and 2) have to be satisfied. According to the definition in the problem, there are only k components of a that are 1 and the rest of the components of a are 0. Let the first k components of a be 1 and the rest of the components a_i be 0. Then $\sum_{i=0}^{n} a_i = \sum_{i=0}^{k} 1 + \sum_{k+1}^{n} 0 = k$. This satisfies constraint 2) from the problem statement. By proving constraint 2) we have satisfied constraint 1) in the process, thus satisfying both constraints 1) and 2), showing that any point a is a feasible point for the LP.

Part B

Consider a point a where each a_i is within the domain of $\in 0, 1$. According to the definition of a, a_i can take on either a value of 0 or 1. Consider a non-zero vector z where each element in the vector z_i can take on three different values: negative, zero or positive.

Now consider the following cases where we attempt to add values of a_i and z_i according to the rules of vector addition and subtraction.

Case 1: $a_i = 0$ $z_i = -1$

If we add two vectors a and z, $a + z$ where some element a_i and z_i are 0 and -1 respectively the resultant vector violates the first constraint thus showing that the resultant vector is a corner point because the result -1 is outside the domain of $\in 0, 1$.

Case 2: $a_i = 0$ $z_i = 1$

If we subtract vectors a and z, $a - z$ where some element a_i and z_i are 0 and 1 respectively the resultant vector violates the first constraint thus showing that the resultant vector is a corner point because the result -1 is outside the domain of $\in 0, 1$.

Case 3: $a_i = 1$ $z_i = -1$

If we subtract vectors a and z, $z - a$ where some element a_i and z_i are 1 and -1 respectively the resultant vector violates the first constraint thus showing that the resultant vector is a corner point because the result -2 is outside the domain of $\in 0, 1$.

Case 4: $a_i = 1$ $z_i = 1$

If we subtract vectors a and z, $a + z$ where some element a_i and z_i are 1 and 1 respectively the resultant vector violates the first constraint thus showing that the resultant vector is a corner point because the result 2 is outside the domain of $\in 0, 1$.

Case 5: $a_i = 0$ $z_i = -1$

If we subtract vectors a and z, $a - z$ where some element a_i and z_i are 0 and -1 respectively,

then the resultant vector violates the second constraint because we have added a new 1 component to the vector result which in turn produces a sum $> k$ where there are exactly k a_i elements that are 1.

Case 6: $a_i = 0$ $z_i = 1$
If we subtract vectors a and z, $a+z$ where some element a_i and z_i are 0 and 1 respectively, then the resultant vector violates the second constraint because we have added a new 1 component to the vector result which in turn produces a sum $> k$ where there are exactly k a_i elements that are 1.

Thus we have shown in the above cases that for any nonzero vector z a is a corner point.

[*Hint:* To prove that a point is a corner point, we use the definition we mentioned in class: a feasible point x is a corner point if and only if for *all* nonzero vectors z, either $x + z$ or $x - z$ is infeasible.]

[*Remark:* It turns out that these are the only corner points of the polytope, and so it is an example of a polytope defined by $2n + 1$ constraints and having $\binom{n}{k}$ corner points.]

Question 6: Checking feasibility vs optimization ..... **[6]**

Some of the algorithms for linear programming (e.g. simplex) start off with one of the corner points of the feasible set. This turns out to be tricky in general. In this problem, we will see that **in general**, finding one feasible point is as difficult as actually performing the optimization!

Consider the following linear program (in n variables $x_1, \ldots, x_n$, represented by the vector x):

$$\begin{gathered}\text{minimize } c^T x \text{ subject to} \\ a_1^T x \geq b_1 \\ a_2^T x \geq b_2 \\ \cdots \\ a_m^T x \geq b_m.\end{gathered}$$

Suppose you know that the optimum value (i.e. the minimum of $c^T x$ over the feasible set) lies in the interval $[-M, M]$ for some real number M (this is typically possible in practice). Suppose also that you have an **oracle** that can take any linear program and say whether it is feasible or not. Prove that using $O(\log(M/\epsilon))$ calls to the oracle, one can determine the optimum value of the LP above up to an error of $\pm\epsilon$, for any given accuracy $\epsilon > 0$. [*Hint:* can you write a new LP that is feasible only if the LP above has optimum value $\leq z$, for some z?]
Since $\pm\epsilon$ is the error tolerance, we can add the total ϵ for the bound to be 2ϵ. Then we can use a variation of binary search to find a linear program that satisfies the requirements of the constraints.

Proof of Correctness
This algorithm is a variation of the binary search algorithm. We define the lower interval bound as $-M$ and the upper interval bound as M. The stopping condition for this algorithm is when the difference between *hi* and *lo* is greater than 2ϵ since ϵ represents the tolerance range between for a z value beween the *hi* and the *lo* bounds. We use the oracle as stated in the

Algorithm 1 Linear Programming Binary Search

Input: Constraints C, optimum value $c^T x, \epsilon$, interval $[-M, M]$
Output: Some minmum optimum value that Z that satisfies the constraints C

lo = -M, hi = M
while $hi - lo > 2\epsilon$ **do**
    z = (hi - lo) / 2
    Use oracle to check if z is feasible. If all constraints satisfied - LP solution, otherwise not an LP solution
    if z is a LP Solution **then** hi = z
    else
lo = z
    end if
end while
return z = (lo + hi) / 2 - z is calculated using the midpoint formula for the upper and lower bounds

problem to determine whether some linear program z satisfies the constraints C defined in the problem. When we call the oracle over the sample space we are looking over a boundary of small slices of ϵ which yields $\frac{1}{\epsilon}$ in the overall runtime when we have to check over the interval range with binary search. **Runtime**
The runtime of the algorithm as given in the problem is $O(\log \frac{M}{\epsilon})$