CS 6150: HW 6 – Optimization formulations, review

Submission date: Wednesday, December 6, 2023, 11:59 PM

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

Question	Points	Score
Regression	5	
Sufficiency of constraints	16	
Minimum vertex cover revisited	16	
Optimal packaging	10	
Balancing sums	8	
Total:	55	

Question 1: Regression .... **[5]**

The min-error linear regression problem is defined as follows: we are given n vectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ in $\mathbb{R}^d$, along with real valued "outputs" $y_1, y_2, \ldots, y_n$. The goal in regression is to find an $x \in \mathbb{R}^d$ such that $\langle \mathbf{u}_i, x \rangle \approx y_i$ for all i. This is formalized as:

$$\text{find } x \in \mathbb{R}^d \text{ that minimizes } \max_{1 \le i \le n} |y_i - \langle \mathbf{u}_i, x \rangle| .$$

Phrase the min-error linear regression problem as a linear program.

Ans)

$u_1 \rightarrow y_1$
$u_2 \rightarrow y_2$
$\vdots$
$u_n \rightarrow y_n$

Let us introduce a new variable z
minimizing $max|y_i - < u_i, x > |$

This would be converted to
$|y_i - < u_i, x > | \le z$
for i=1,2...n
This would be our new objective function and our goal would be to minimize z

Variables:
$x \in R^d$
Therefore
$z \in R^d$

Objective function:
Minimize z

Constraints:
$|y_i - < u_i, x > | \le z$
for i=1,2...n

Question 2: Sufficiency of constraints .... **[16]**

Recall the maximum independent set problem: we are given an undirected graph $G = (V, E)$ and the goal is to find a subset S of vertices of the largest possible size such that no two vertices in S have an edge.

(a) [**6**] Consider the following optimization problem:

$$\max \sum_{u \in V} x_u \text{ subject to}$$
$$x_u^2 = x_u \text{ for all } u \in V$$
$$x_u + x_v \le 1 \text{ for all } \{u, v\} \in E$$

Prove that it captures the problem exactly; i.e., any feasible solution to formulation yields a feasible solution to original problem **and vice versa**.

Ans)

Variables:
x_v and x_u representing vertices forming independent set.

Constraints:
$(x_u)^2 = x_u \ u \in V$
This represents that value of x can only be binary

$x_u + x_v \leq 1 \ u, v \in E$
This represents that between two adjacent vertices that belong to an edge, only one edge can be selected.

Objective function:

$\max \sum_{u \in V} x_u$

We can solve this question using contradiction.
To prove: Any feasible solution to the formulation yields solution to original problem

Let us consider a situation

That both x_v and x_u are chosen
Therefore
$x_u + x_v \geq 1$
This contradicts the second constraint $x_u + x_v \leq 1$
Therefore solution doesn't exist.

(b) [**5**] Now consider the relaxation in which the constraint $x_u^2 = x_u$ is replaced with $0 \leq x_u \leq 1$, for all u. This is now a linear programming relaxation. Consider the case in which the graph G is a clique with n vertices. Prove that the relaxation has a feasible solution $\{x_u\}$ such that $\sum_u x_u = n/2$. What is the value of the "true" maximum independent set in this case?
Ans)
New constraint is:
$0 \leq x_u \leq 1$
Graph G is clique and has n vertices.

One feasible solution that we can take is $x_u = \frac{1}{2}$
We can see that it satisfies both the constraints.
Therefore

$x_u = x_v = \frac{1}{2}$ according to the second constraint
Now we calculate $\sum_{u \in V} x_u$
$\sum_{u \in V} \frac{1}{2}$

Therefore $\frac{n}{2}$

But in a clique, maximum vertices in an independent set can be 1.
Therefore this would be the true maximum independent set in this case.

(c) [**5**] Why are these answers different? Could we just use the formulation in part a) to solve Maximum Independent Set quickly?
Ans)
Answers are different because one of the constraint has been relaxed.
First problem is type of ILP problem and it is a type of NP complete problem.
After relaxation it is a LP problem but still type of NP complete problem.
Therefore using the formulation won't result in any significant change in terms of complexity as

both are NP complete.

Question 3: Minimum vertex cover revisited .... **[16]**

Recall the minimum vertex cover problem that we saw in HW 3. We are given an undirected graph $G = (V, E)$ and the goal is to select a subset S of the vertices of the smallest possible size that can "monitor" all the edges, i.e., for every edge $\{i, j\} \in E$, at least one of i, j is in S (so the objective is to minimize $|S|$ subject to the above).

(a) **[5]** What is a natural "decision version" of the problem? (Recall how we did this for independent set in class.)

Ans)
We would solve this by introducing a new variable/parameter k.
We can convert the vertex cover problem by considering a subset of vertices that are at most k.
$|S| \leq k$
Therefore we find the vertex cover of size at-most k covering all the edges.

(b) **[5]** Show that the min vertex cover problem is in the class NP. Specifically, give a *verification algorithm* and describe the appropriate *certificate*.

Ans)

Verification Algorithm:
True if size of S is at most k, else false
Check if $S \leq k$
For each edge, we need to check if at least one edge exists in S.
Check the size of S that it doesn't exceed k
If above are satisfied then it is true.

Certificate: Subset of S vertices
It has 2 conditions:
i) Size of S is at most K
ii) At least one edge should be in S.

If both are true then certificate is valid.

(c) **[6]** We studied the linear programming (LP) relaxation for vertex cover. Recall that it is as follows:

$$\text{minimize} \sum_{u \in V} x_u \text{ subject to}$$
$$0 \leq x_u \leq 1 \text{ for all } u \in V$$
$$x_u + x_v \geq 1 \text{ for all } \{u, v\} \in E.$$

In class, we saw a rounding algorithm that takes a feasible solution $\{x_u\}$ to the LP above and produces a **feasible, binary** solution whose objective value is at most $2 \sum_u x_u$. Now, suppose we were lucky and the LP solution had all the x_u satisfying $x_u \in [0, 0.2) \cup (0.8, 1]$. In this case, prove that rounding produces a feasible, binary solution whose cost is at most $(1.25) \sum_u x_u$.

Ans)
We know the solution exists in $x_u \in [0, 0.2) \cup (0.8, 1]$
We apply the rounding algorithm
For $[0, 0.2)$ we take the value 0.
For $[0.8, 1)$ we take the value 1.
We are give that $x_u + x_v \geq 1$
This means that there exists a x_u whose value is greater than 0.8

Therefore x_u is at least 0.8 before we round it.
Therefore the cost is at most $\frac{1}{0.8}$
Cost is at most 1.25

Question 4: 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 we have only a small number of *distinct lengths*. I.e., suppose that there is some set $L = \{s_1, \ldots, s_r\}$ such that all the $a_i \in L$ (and think of r is as a small constant). Devise an algorithm that runs in time $O(n^{3r})$ (or better), 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.]

Ans)
Sort length arrays
Initialize the array of size $(n + 1) * (distinct_lengths + 1)$
Set DP[0][0]
for i from 1 to n
for k from 1 to distinct$_lengths$
DP[i][k]=min(DP[i][k], j[0,i1] min(DP[j [k1]+numBoxes(lengths[i1],distinct$_lengths[: k], k)))$
Minimum number of boxes given by min DP[n][k]
Calculate the number of boxes needed for a configuration considering lengths in the range [1, k].
Iterate over all possible configurations of lengths that can fit into a single box.
Find the configuration that minimizes the total length while accommodating length in the k-th box.

Time complexity would be $(O(n^{2r}))$

Algorithm 1 Minimize Boxes (2D DP)

function MinimizeBoxes(n, lengths, distinct_lengths)
    Sort distinct_lengths in ascending order
    Initialize DP array with dimensions $n+1 \times (|\text{distinct\_lengths}|+1)$ **for** $i \leftarrow 0$ *to* n **do**
    $k \leftarrow 1$ to $|\text{distinct\_lengths}|$
    $DP[i][k] \leftarrow \infty$
    $DP[0][0] \leftarrow 0$
    for $i \leftarrow 1$ *to* n **do**
    $k \leftarrow 1$ to $|\text{distinct\_lengths}|$
    $DP[i][k] \leftarrow \min\left(DP[i][k], \min_{j\in[0,i-1]}(DP[j][k-1] + \text{NumBoxes}(\text{lengths}[i-1], \text{distinct\_lengths}[:k], k))\right)$
    Find the minimum number of boxes needed:
    $min\_boxes \leftarrow \min(DP[n][k] \mid k \in [1, |\text{distinct\_lengths}|])$
    return min_boxes
end function
function NumBoxes(length, config, k)
    Helper function to calculate the number of boxes needed for a configuration
    considering lengths in the range $[1,k]$
    $total\_length \leftarrow 0$
    $num\_boxes \leftarrow 0$
    for *length_in_config in config* **do**
    length_in_config $\leq k$
    $total\_length \leftarrow total\_length + \text{length\_in\_config}$
    $num\_boxes \leftarrow num\_boxes + 1$
    return num_boxes if $total\_length \geq 1$ else ∞
end function

Question 5: Balancing sums .... **[8]**

Suppose we have n real numbers $a_1, a_2, \ldots, a_n \in (0,1)$. The "balancing" problem asks to find $\pm$ *signs* such that the signed sum of the a_i is as small as possible. Formally, the goal is to find signs $s_i \in \{+1,-1\}$ for every $1 \leq i \leq n$ such that $|\sum_i s_i a_i|$ is minimized. Suppose for a moment that a_i are all rational numbers (fractions) with a common denominator p. Give an algorithm for this problem with running time polynomial in p, n.

Ans)
First we create a DP table named dp of size nX2p
We initialize dp[i][j]
set $dp[0][p] = 0$

Now the dynamic programming step

for i from 1 to n
for j from o to 2p-1
DP will have 3 choices
$dp[i-1][j-1] + |a_i - \frac{j}{p}|$ (using the negative sign)
$dp[i-1][j] + |a_i - \frac{j}{p}|$ (no sign change)
$dp[i-1][j+1] + a_i - \frac{j}{p}$ (using the positive sign)

We do back tracking from the last row of $dp[n][:]$ to find the sign that minimalizes the sum

Run-time analysis:
We have n iterations and each iteration has $O(P)$
So the total run-time complexity would be $O(n * p^2)$

Algorithm 2 Balancing Problem

procedure OptimalSigns(a, p, n)
    Create a table DP of size $n \times 2p$ **for** $i \leftarrow 0$ *to* n **do**
    $j \leftarrow 0$ to $2p - 1$
    $DP[i][j] \leftarrow \infty$
    $DP[0][p] \leftarrow 0$
    for $i \leftarrow 1$ *to* n **do**
    $j \leftarrow 0$ to $2p - 1$
    $DP[i][j] \leftarrow \min(DP[i][j], DP[i-1][j-1] + |a_i - j/p|)$ ▷ Using a negative sign
    $DP[i][j] \leftarrow \min(DP[i][j], DP[i-1][j] + |a_i - j/p|)$ ▷ No sign change
    $DP[i][j] \leftarrow \min(DP[i][j], DP[i-1][j+1] + |a_i - j/p|)$ ▷ Using a positive sign
    Backtrack to find optimal signs and calculate the optimal signed sum
    Return Optimal signs and the optimal signed sum
end procedure