CS 170 DIS 08

Released on 2018-10-22

1 Taking a Dual

Consider the following linear program:

$$\begin{aligned} \max\ & 4x_1 + 7x_2 \\ x_1 + 2x_2 &\leq 10 \\ 3x_1 + x_2 &\leq 14 \\ 2x_1 + 3x_2 &\leq 11 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Construct the dual of the above linear program.

Solution: If we scale the first constraint by $y_1 \geq 0$, the second by $y_2 \geq 0$, the third by $y_3 \geq 0$, and we add them up, we get an upperbound of $(y_1+3y_2+2y_3)x_1+(2y_1+y_2+3y_3)x_2 \leq (10y_1 + 14y_2 + 11y_3)$. Minimizing for a bound for $4x_1 + 7x_2$, we get the tightest possible upperbound by

$$\begin{aligned} \min\ & 10y_1 + 14y_2 + 11y_3 \\ y_1 + 3y_2 + 2y_3 &\geq 4 \\ 2y_1 + y_2 + 3y_3 &\geq 7 \\ y_1, y_2, y_3 &\geq 0 \end{aligned}$$

2 Weighted Rock-Paper-Scissors

You and your friend used to play rock-paper-scissors, and have the loser pay the winner 1 dollar. However, you then learned in CS170 that the best strategy is to pick each move uniformly at random, which took all the fun out of the game.

Your friend, trying to make the game interesting again, suggests playing the following variant: If you win by beating rock with paper, you get 5 dollars from your opponent. If you win by beating scissors with rock, you get 3 dollars. If you win by beating paper with scissors, you get 1 dollar.

(a) Draw the payoff matrix for this game.

(b) Write a linear program to find the optimal strategy.

Solution:

(a)

You: \ Your Friend:	rock	paper	scissors
rock	0	-5	3
paper	5	0	-1
scissors	-3	1	0

(b) Let r, p, s be the probabilities that you play rock, paper, scissors respectively. Let z stand for the expected payoff, if your opponent plays optimally as well.

$$\begin{aligned} \max \quad & z \\ 5p - 3s &\geq z && \text{(Opponent chooses rock)} \\ s - 5r &\geq z && \text{(Opponent chooses paper)} \\ 3r - p &\geq z && \text{(Opponent chooses scissors)} \\ r + p + s &= 1 \\ r, p, s &\geq 0 \end{aligned}$$

3 Domination

In this problem, we explore a concept called *dominated strategies*. Consider a zero-sum game with the following payoff matrix for the row player:

		Column: A	B	C
Row:	D	1	2	-3
	E	3	2	-2
	F	-1	-2	2

(a) If the row player plays optimally, can you find the probability that they pick D without directly solving for the optimal strategy? (Hint: Notice that the payoff for E is always greater than the payoff for D. When this happens, we say that E *dominates* D, i.e. D is a *dominated strategy*).

(b) Given the answer to part a, if the both players play optimally, what is the probability that the column player picks A?

(c) Given the answers to part a and b, what are both players' optimal strategies? (You might be able to figure this out without writing or solving any LP).

Solution:

(a) 0. Regardless of what option the column player chooses, the row player always gets a higher payoff picking E than D, so any strategy that involves a non-zero probability of picking E can be improved by instead picking D.

(b) 0. We know that the row player is never going to pick D, i.e. will always pick either E or F. But in this case, picking B is always better for the column player than picking A (A is only better if the row player picks D). That is, conditioned on the row player playing optimally, B dominates A.

(c) Based on the previous two parts, we only have to consider the probabilities the row player picks E or F and the column player picks B or C. Looking at the 2-by-2 submatrix corresponding to these options, it follows that the optimal strategy for the row player is to pick E and F with probability $1/2$, and similarly the column player should pick B, C with probability $1/2$.

4 MT2 Practice: Greedy

Assume there are n activities each with its own start time a_i and end time b_i such that $a_i < b_i$. All these activities share a common resource (think computers trying to use the same printer). A feasible schedule of the activities is one such that no two activities are using the common resource simultaneously. Mathematically, the time intervals are disjoint: $(a_i, b_i) \cap (a_j, b_j) = \emptyset$. The goal is to find a feasible schedule that maximizes the number of activities k.

Here are two potential greedy algorithms for the problem.

Algorithm A: Select the shortest-duration activity that doesn't conflict with those already selected until no more can be selected.

Algorithm B: Select the earliest-ending activity that doesn't conflict with those already selected until no more can be selected.

(a) Show that Algorithm A can fail to produce an optimal output.

(b) Show that Algorithm B will always produce an optimal output.

(c) **Challenge Problem:** Show that Algorithm A will always produce an output at least half as large as the optimal output.

Solution:

(a) There are many examples that work, but here is a simple one, easiest explained pictorally (each line represents an activity):

The optimal strategy is to pick the two longer activities, but strategy B picks the shorter activity, and then cannot pick another.

(b) Consider any optimal schedule S, and suppose the first i activities in S are the same as the first i activities chosen by Algorithm B (i might be zero). If both schedules have i activities, Algorithm B's solution is the same as S, so the proof is done. Otherwise, both Algorithm B's solution and the optimal solution must have a $(i+1)$st activity: If Algorithm B's solution is a subset of S, it would have added one of the remaining activities in S, and if Algorithm B's solution has more than i activities but S only has i activites, S is not optimal. In either case, we have a contradiction.

Then, consider swapping the $(i+1)$st activity in the optimal solution with the $(i+1)$st activity in our solution. By definition of Algorithm B, its $(i+1)$st activity ends at least

as early as the $(i+1)$st activity in S. This means it can't overlap with later activities in S, so the feasibility and size of S are maintained by this swap. This means that given any optimal schedule where the first i activities are the same as in Algorithm B's solution, we can find an optimal schedule where the first $i+1$ activities are the same as in Algorithm B's solution. Starting from any optimal schedule and applying this repeatedly, we find an optimal schedule which is exactly the same as in Algorithm B's solution.

(c) Let I_{opt} be any optimal set of activities and I be the activities chosen by the shortest-duration greedy strategy.

We show the following:

(a) Every activity in I_{opt} overlaps at least 1 activity in I.
Suppose there exists activities in I_{opt} that does not overlap with any activity in I. Let $x \in I_{opt}$ be such an activity of shortest duration. If x had shorter duration than any other activities in I, it would have been added before the others were; contradiction. Otherwise, x would have been added just after all those in I are added; contradiction.

(b) Any activity in I can overlap at most 2 activities in I_{opt}.
Let I_k be the set of activities added by the greedy strategy just after the kth iteration. We show by induction that any activity in I_k overlaps with at most 2 activities in I_{opt}.

- Base case: $I_0 = \emptyset$; vacuously true.
- Inductive case: Assume true for I_k and let x be the activity added during the $k+1$th iteration. If $x \in I_{opt}$, we are done. Otherwise, suppose for the purpose of contradiction that x overlapped with strictly more than two activities in I_{opt}. Then it must be that one such activity begins and ends while x is still active; contradiction.

The original claim follows immediately as a corollary.

The two claims above imply $|I_{opt}| \leq 2|I|$.

5 MT2 Practice: MSTs

Let $G = (V, E)$ be an undirected, connected graph.

(a) Prove that there is a unique MST if all edge weights are distinct.

(b) True or False? If G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a MST.

(c) True or False? If the lightest edge in G is unique, then it must be a part of every MST.

Solution:

(a) Suppose the graph has two different MSTs T_1 and T_2. Let e be the lightest edge present in just one of the trees. Without loss of generality, say $e \in T_1$. Thus, adding e to T_2 gives a cycle. However, this cycle must contain an edge e' not in T_1 which connects the two components of $T_1 - e$. Since e is the lightest edge in just one of the trees, e is lighter than e'. Thus, we can add e to T_2 and remove e' to get a better MST. This contradicts our claim of two different MSTs. Thus, there is only one unique MST in this graph.

(b) False; consider the case when removing the heaviest edge disconnects the graph. Then any spanning tree must include the heaviest edge in the tree.

(c) True; if the lightest edge e is not part of some MST, there exists a cycle connecting the two endpoints of e, so adding e and removing another edge of the cycle produces a lighter tree, a contradiction.

6 MT2 Practice: Dynamic Programming

Professor Rao loves to play golf, but as a combinatorial mathematician, he prefers to play a version called discrete golf. The goal of discrete golf is to hit a golf ball from checkpoint 1 to checkpoint n on a golf course in as few strokes as possible. Based on the terrain of the course, he knows that from checkpoint i, he can hit the ball up to $d(i)$ checkpoints away in one stroke. That is, if the ball is at checkpoint i, in one stroke he can hit the ball to any of checkpoints $i+1, i+2 \dots i+d(i)$.

(a) Suppose he hits the ball as far as possible every stroke, i.e. if he is at checkpoint i, he hits the ball to checkpoint $i + d(i)$. Give a counterexample where this greedy algorithm does not achieve the minimum number of strokes needed to reach checkpoint n from checkpoint 1.

(b) Suppose that all $d(i)$ are at most D. Give a $O(nD)$ time algorithm for computing the minimum number of strokes Professor Rao needs to reach point n. How much memory does this algorithm need?

Solution:

(a) Consider when $n = 5, d(1) = 2, d(2) = 3, d(3) = 1, d(4) = 1$. The greedy algorithm will visit checkpoints $1, 3, 4, 5$, the optimal route is to visit checkpoints $1, 2, 5$.

(b) Let $s(i)$ be the minimum number of strokes needed to reach checkpoint n from checkpoint i. Then the base case is $s(n) = 0$, and for $i < n$, $s(i) = 1 + \min_{i+1 \le j \le i+d(i)} s(j)$. The DP table has n entries, and each takes $O(D)$ time to update, so the runtime is $O(nD)$. Since each entry of the DP table only relies on the next D entries, it suffices to only store D entries, i.e. only $O(D)$ space is needed.

One can also define $s(i)$ to be the minimum number of strokes needed to reach checkpoint i from checkpoint 1, but then the recurrence relation becomes a bit more messy.