PSO #11 Solutions Sketch (Week 13)

Week of 2021-11-15

1 Network Flow

1. Purdue is hosting a large event and needs volunteers to help with a vast array of duties. Faculty, admin staff, and students have all signed up to volunteer, and each volunteer $1 \leq i \leq n$ has provided a list S_i which denotes the timeslots they are able to cover, where $n = f + a + s$. There are a total of m timeslots that need to be covered. Each student is assigned 1 timeslot, staff 2, and faculty 3. We also have that $m = s + 2a + 3f$.

   Develop a max-flow algorithm to assign volunteers to time slots, and report how many timeslots were unable to be filled (if any).

   Solution:

   - Create a source vertex s' and sink vertex t'. Create vertices v_i for each of the n volunteers. Create vertices w_j for each of the m timeslots.
   - For each volunteer i, create an edge from $s' \to v_i$ with capacity 1 if i is a student, 2 for staff, 3 for faculty
   - For each volunteer i, create a capacity 1 edge from $v_i \to w_j$ for each $j \in S_i$
   - For each timeslot j, create an edge from $w_j \to t'$ with capacity 1

   We can then run Ford-Fulkerson, and the max flow value is the number of slots covered, and the edges between the volunteer vertices and slot vertices represent the assignments.

   There are $O(nm)$ different edges with a max flow of at most m, giving a runtime of $O(nm^2)$

2 Turing Machine

1. Design a turing machine (FSA) that accepts the following language: $L = ab[a - b]*$.

   Solution:
   Draw a FSA with 3 nodes, where the first node has a directed edge to the second with a, the second node has a directed edge to the third with b, and the third node is the end state with two self-loops, labeled a and b respectively.

3 Linear Programming

1. Design a Linear Program to find the radius of the largest ball that can be fit inside a polyhedron given a vector pointing to the center x_c. Hint: We define "inside a side" of the polyhedron for a point p to mean if for a side i, $a_i^T p \leq b_i$, where a_i is a unit vector and $b_i \geq 0$.

   Solution: The linear program is as follows:

$$\begin{aligned} \text{maximize} \quad & r \\ \text{subject to} \quad & a_i^T x_c + r \leq b_i \end{aligned}$$

   We reason this to be that the point in the center is x_c, so we determine how far we can deviate from $a_i^T x_c$ which determines which side of the hyperplane defined by $a_i^T p = b_i$ for each side is.

4 Games

1. Given the following payoff matrix A, write the corresponding LP to find the optimal value for Player A if Player B is equally likely to play 0 or 1.

(A/B)		*Player*	A
		0	1
Player	0	$3/-3$	$-4/4$
B	1	$-2/2$	$3/-3.$

   Solution:

$$\begin{aligned} \text{maximize} \quad & v \\ \text{subject to} \quad & 0 \leq \sum_{i \in [0,1]} p_i \leq 1 \\ & \sum_{i \in [0,1]} \frac{1}{2} p_i A_{i,j} \geq v, \forall j \in [0,1] \\ & p_i \geq 0, \forall i \in [0,1] \end{aligned}$$

2. Consider the same payoff matrix as the previous problem, except in the top left we have $1/-1$ instead of $3/-3$. Calculate the random strategy that results in the Nash Equilibrium.

   Solution:
   The expected payoff when Player B plays 0 is $p - 4(1-p) = 5p - 4$, and when Player B plays 1, the expected payoff is $-2p + 3(1-p) = -5p + 3$. The Nash equilibrium is when these are the same, so $5p - 4 = -5p + 3 \rightarrow p = \frac{7}{10}$. Thus, Player A should play 0 $\frac{7}{10}$th's of the time, and 1 $\frac{3}{10}$th's of the time.

5 Reductions

1. Note: if needed, explain optimization vs decision problems

   Given a set I of n items, each with weights w_i and profit p_i, and a max weight W, does there exist a subset of items with total weight less than W, total profit greater than k?

   Suppose you're given an oracle $f(I, W, k)$ that solves the decision version of knapsack in polynomial time. Describe and analyze an algorithm that uses the oracle f a polynomial number of times to generate a set of items to pack that achieves maximum profit. That is, describe a reduction Knapsack Optimization $\leq_p$ Knapsack Decision

   Solution:

   - The max profit any subset can have is the sum of all the profits in that subset, let denote $P = \sum_{i=1}^{n} p_i$. Use binary search on k from $1 \to P$ to find the maximum possible profit of the subset using the oracle. Let this max profit be $k*$
   - For each item $j \in I$, query $f(I - j, W, k*)$
     (a) If yes, then j is not in the maximizing set, so remove it. $I = I - j$
     (b) If no, then j is in the maximizing set, and I remains unchanged
   - The only items remaining in I afterwards will form the maximum profit

2. Let ZAP be a problem with an $\Omega(n^2)$ lower bound. Bob shows that $ZAP <_{O\left(\left(\frac{n}{\log n}\right)^2\right)} B$

   Alice wants to find a $O(n \log n)$ algorithm for problem B, but Bob claims such an algorithm doesn't exist. Who is right, and why?

   Solution:

   The reduction shows that ZAP can be solved using a solver for B with an additional $\left(\frac{n}{\log n}\right)^2$ amount of work. We know ZAP cannot be solved in less than n^2 time.

   Thus, if B can be solved in $< n^2$ time, this would mean we could solve ZAP in $\left(\frac{n}{\log n}\right)^2$ time or the time it takes to solve B, whichever is larger

   However, both of these would be $< n^2$, which would contradict the given assumption that ZAP is $\Omega(n^2)$