Advanced Operations Research Techniques IE316

Lecture 20

Dr. Ted Ralphs

Reading for This Lecture

- Bertsimas 7.8-7.10

The Assignment Problem

- The *assignment problem* is that of assigning n people to n projects so as to minimize cost.
- An LP formulation is as follows:

$$\begin{aligned} min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}f_{ij} & \\ s.t. \quad \sum_{i=1}^{n} f_{ij} = 1, & \quad j = 1,\ldots,n \\ \sum_{j=1}^{n} f_{ij} = 1, & \quad i = 1,\ldots,n \\ f_{ij} \geq 0, \forall i,j & \end{aligned}$$

- Note that this can be interpreted as a network flow problem, so there always exists an optimal solution for which $f_{ij} \in \{0,1\}$.
- This allows us to interpret the solution as an assignment.

The Dual of the Assignment Problem

- The dual problem has the following form:

$$
\begin{aligned}
&max \sum_{i=1}^{n} + \sum_{j=1}^{n} \\
&s.t. \qquad r_i + p_j \leq c_{ij}, \forall i, j.
\end{aligned}
$$

- In order to maximize $\sum_{i=1}^{n} r_i$, we must have

$$r_i = \min_{j=1,\ldots,n} \{c_{ij} - p_j\}$$

- Hence, we can rewrite the dual as

$$\max \left(\sum_{j=1}^{n} p_j + \sum_{i=1}^{n} \min_j \{c_{ij} - p_j\} \right)$$

- This is an unconstrained optimization problem with a piecewise concave objective function.

The Complementary Slackness Conditions

- The complementary slackness conditions tell us that

$$f_{ij} > 0 \Rightarrow r_i + p_j = c_{ij}$$

- Substituting the previous form for r_i, we get

$$f_{ij} > 0 \Rightarrow p_j - c_{ij} = \max_k \{p_k - c_{ik}\}$$

- If we view p_k as the profit associated with project k, then this says that each person should be assigned to the most profitable project.
- This leads to an algorithm known as the *auction algorithm*, in which we envision each person bidding for projects in multiple rounds.

The Auction Algorithm

- Given a set of prices $p_1, \ldots, p_n$ for the different projects and a partial assignment of people to projects,
  - Each unassigned person finds their best project and "bids" for it by accepting a lower price.
  - The bid is the lowest price that is acceptable for the project:

    (profit of the best project) - (profit of the second best project)

  - Following the bidding phase, every project is assigned to the lowest bidder.
  - The process continues until all projects are assigned.
- Notice that we maintain dual feasibility and complementary slackness.
- We try to achieve primal feasibility.
- Will this algorithm always terminate?

A Modified Auction Algorithm

1. Start with a set of prices $p_1, \ldots, p_n$ and a partial assignment of people to projects.
2. For each unassigned person i, find a best project k_i by maximizing $p_k - c_{ik}$ over all k. Let k_i' be a second best project, and let

$$\Delta_{k_i} = (p_{k_i} - c_{ik_i}) - (p_{k_i'} - c_{ik_i'}).$$

   Person i "bids" $p_{k_i} - \Delta_{k_i} - \epsilon$ for project i.

3. Each project is assigned to the lowest bidder. The new prices are set to the value of the lowest bid.

- Note that ϵ must be positive and $< 1/n$.
- This version is guaranteed to terminate with the optimal solution.

Why the Algorithm Works

- In essence, the perturbation in the dual prices prevents cycling.
- Every project must eventually be assigned since no project can receive an infinite number of bids (the price goes down by at least ϵ each round).
- Let j_i be the project assigned to person i in the final solution.
- The complementary slackness conditions cannot be violated by more than ϵ, which means

$$p_{j_i} - c_{ij_i} \geq \max_j \{p_j - c_{ij}\} - \epsilon, \ \forall i.$$

- Summing, we obtain

$$OPT \leq \sum_{i=1}^{n} c_{ij_i} \leq \sum_{i=1}^{n} \Big(p_{j_i} + \min_i \{c_{ij} - p_j\}\Big) + n\epsilon \leq OPT + n\epsilon < OPT + 1.$$

- Since OPT and all the costs are integer, this shows that $OPT = \sum_{i=1}^{n} c_{ij_i}$.

The Shortest Path Problem

- We are give a directed graph $G = (N, A)$ and a cost or *length* associated with each arc.
- We define the length of a path to be the sum of the lengths of the arcs in the path.
- The basic *shortest path problem* is that of finding the path of minimum length between a given origin and a given destination.
- This is equivalent to a certain minimum cost flow problem (why?).

Shortest Paths Trees

- A tree that consists of a directed path from nodes $1, \ldots, n-1$ to node n is called an *intree rooted at node n*.
- An intree that consists of the shortest paths from nodes $1, \ldots, n-1$ to node n is called a *shortest paths tree*.
- As long as there are no negative length cycles, calculating a shortest paths tree is equivalent to an uncapacitated minimum cost network flow problem with
  - a supply of 1 at nodes $1, \ldots, n-1$, and
  - a demand of $n-1$ at node n.
- Furthermore, assuming $p_n^* = 0$, the unique solution to the dual problem consists of assigning

$$p_i^* = \text{the path length from node } i \text{ to node } n.$$

Bellman's Equation

- Since $b_1 = \cdots = b_{n-1} = 1$, the dual problem has the form

$$min \sum_{i=1}^{n-1} p_i$$
$$s.t. \;\; p_i \leq c_{ij} + p_j \, \forall (i,j) \in A$$

- Hence, p, the vector of shortest path lengths satisfies

$$p_i^* = \min_{k \in O(i)} \{c_{ik} + p_k^*\}$$

- The Bellman-Ford algorithm is to iteratively solve this system of equations.
- As long as there are no negative length cycles, the solution is unique after assigning $p_n = 0$.

Label Correcting Methods and Dijkstra's Algorithm

- *Label correcting methods* are a more efficient class of algorithms.
- We will discuss *Dijkstra's Algorithm*, a simple algorithm that can be applied when all arc costs are nonnegative.
- Algorithm

1. Find a node $l \neq n$ such that $c_{ln} \leq c_{in}$ for all $i \neq n$.
2. For every node $i \neq l, n$, set

$$c_{in} := \min\{c_{in}, c_{il} + c_{ln}\}$$

3. Remove node l from the graph and apply the same steps to the new graph.

The Minimum Spanning Tree Problem

- Consider an undirected graph $G = (N, E)$ with cost vector $c \in \mathbf{Z}^E$.
- The *Minimum Spanning Tree Problem* is to find a spanning tree of minimum total cost.
- This problem has application in network design and many other settings.
- Such a spanning tree can be found using a *greedy algorithm*.
- Greedy Algorithm
  1. Start with a tree (N_1, E_1) consisting of a single node.
  2. At step k, consider the tree (N_k, E_k). Let edge $e^* = \{i^*, j^*\}$ be a cheapest edge among all edges $\{i, j\}$ such that $i \in N_k$ and $j \notin N_k$.
  3. Let

$$\begin{aligned} N_{k+1} &= N_k \cup \{j^*\} \\ E_{k+1} &= E_k \cup \{e^*\} \end{aligned}$$

  4. Go back to Step 2.