

Lecture 12

1 Linear Programming as an Optimization Problem

Linear Programming is an example of an optimization problem. A typical linear programming problem is,

$$\begin{aligned} & \text{minimize} && \sum_{i=1}^{i=n} c_i x_i \\ & \text{subject to:} && \mathbf{A} \vec{x} \preceq \vec{b} \end{aligned}$$

where \preceq denotes component-wise inequality and \mathbf{A} is an $m \times n$ matrix, \vec{x} is an $n \times 1$ vector and \vec{b} is an $m \times 1$ vector. We will refer to the above form of a linear programming problem as the standard form of a linear programming problem. We shall denote the optimal solution to the above LP by \vec{x}^* .

This optimization problem is called a linear program because all constraints and the objective function are linear. Very efficient techniques are known that solve linear programming problems very quickly. With such algorithms it is possible to solve large instances of such problems (containing tens of thousands variables and constraints) in minutes. Examples of commercial linear programming problem solvers include CPLEX and MATLAB.

A lot of problems that have such linear constraints can be reduced to the standard form. For example a problem with equality constraints can be reduced to the standard form LP as follows. Consider

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && \mathbf{A}x = b \end{aligned}$$

The constraint $\mathbf{A}x = b$ can be equivalently written as the set of constraints $\mathbf{A}x \preceq b$ and $\mathbf{A}x \succeq b$. i.e. the following LP is equivalent to the one presented above.

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && \begin{bmatrix} -A \\ A \end{bmatrix} x \preceq \begin{bmatrix} -b \\ b \end{bmatrix} \end{aligned}$$

2 Linear programming and Max Flow

We are interested in linear programming because we can write the max flow problem as an LP. Consider the max-flow problem over a graph $G = (V, E)$ with specified capacities $c_e, e \in E$. Let the value of the $s - t$ flow (s - source node, t - terminal node) be represented by R (where R needs

to be maximized). The max-flow problem can be represented as

$$\begin{aligned}
& \text{maximize } R \\
& \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ into } s} f(e) = R \\
& \sum_{e \text{ out of } t} f(e) - \sum_{e \text{ into } t} f(e) = -R \\
& \sum_{e \text{ out of } v} f(e) - \sum_{e \text{ into } v} f(e) = 0 \text{ for } v \in V - \{s, t\} \\
& 0 \leq f(e) \leq c_e \text{ for } e \in E
\end{aligned}$$

In this optimization the decision variables that need to be determined are the flow variables $f(e), e \in E$ and the value of the flow R i.e. we seek (f^*, R^*) . It can be seen that this is a linear program in the decision variables.

Let us rewrite this LP in the matrix form. Define two vectors \vec{x} and \vec{c} as follows:

$$\vec{x} = \begin{bmatrix} R \\ f(e_1) \\ \dots \\ \dots \\ \dots \\ f(e_{|E|}) \end{bmatrix},$$

$$\vec{c} = \begin{bmatrix} -1 \\ 0 \\ \dots \\ \dots \\ \dots \\ 0 \end{bmatrix}.$$

Then the max flow linear programming problem reduces to

$$\begin{aligned}
& \text{minimize } \vec{c}^T x \\
& \mathbf{A}\vec{x} = 0 \\
& 0 \leq x_e \leq c_e \text{ for } e \in E.
\end{aligned}$$

where x is a vector of length $|E| + 1$. The first component of x is the rate variable and the remaining variables are the flows over the edges.

The matrix \mathbf{A} is of dimension $|V| \times (|E| + 1)$. It can be represented as $\mathbf{A} = [\mathbf{A}_R \mid \mathbf{A}_E]$. where \mathbf{A}_R represents the column vector multiplying the variable R and \mathbf{A}_E represents the matrix multiplying the flow vector. The matrix A_E is the node-edge (or node-arc) incidence matrix of the graph G . Each column corresponds to an edge $e \in E$. If $e = (i, j)$ (under some numbering of the nodes) the $A_E(i, e) = -1$ and $A_E(j, e) = +1$. Similarly $A_R(s, 1) = 1$ and $A_R(t, 1) = -1$. The example below serves to clarify this.

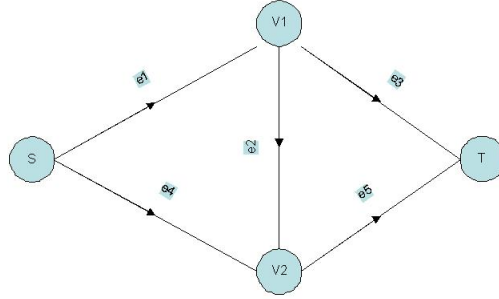


Figure 1: Network with four nodes and five edges. All edges have unit capacities.

In the above graph,

$$\begin{aligned}
 f(e_3) + f(e_2) - f(e_1) &= 0 \\
 f(e_5) - f(e_2) - f(e_4) &= 0 \\
 f(e_1) + f(e_4) &= R \\
 -f(e_3) - f(e_5) &= -R \\
 0 \leq f(e_i) \leq 1 &\text{ for } 1 \leq i \leq 5.
 \end{aligned}$$

Therefore,

$$\vec{x} = \begin{bmatrix} R \\ f(e_1) \\ f(e_2) \\ f(e_3) \\ f(e_4) \\ f(e_5) \end{bmatrix}.$$

and

$$\mathbf{A} = \begin{bmatrix} -1 & 1 & 0 & 0 & 1 & 0 \\ 0 & -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 1 \\ 1 & 0 & 0 & -1 & 0 & -1 \end{bmatrix}.$$

In the matrix A above, row 1 corresponds to the source node and row 4 corresponds to the terminal node.

In general if we enforce the decision variables to be integers then the linear programming problem becomes an ILP (Integer Linear Programming) which is an NP-hard problem. In the special case

of network flow problem (which is a special type of LP) if all capacities are integers it is possible to show that there exists a feasible integral solution to the LP. Recall that max-flow algorithm presented earlier finds such an integral solution.

3 Multicast problem

The multicast problem is one of determining the maximum flow that can be supported simultaneously from a given source to a set of terminal nodes in a directed graph. Finding the value of this maximum flow depends heavily on whether we allow the network to perform network coding. We classify the cases of interest below.

1. Unicast: The single source single terminal problem can be solved efficiently using specialized Max-flow algorithms or Linear Programming. In this case since we can find edge-disjoint paths, routing is sufficient.
2. Multicast: In this case there is one source and a proper subset of the remaining vertices are the terminals. Under routing it can be shown that we need to pack edge-disjoint trees rooted at the source such that all terminals have a path from the source on each tree. These trees are called Steiner trees. It is known that finding a Steiner tree is an NP-hard problem. However when network coding is allowed, the problem becomes a simple linear programming problem.
3. Broadcast: In this scenario there is a single source and all the remaining vertices are terminals. It turns out due to a celebrated result of Edmonds that in this case routing suffices. The result of Edmonds shows that one can always pack k edge-disjoint spanning trees rooted at the source, where $k = \min_{v \in V - \{s\}} \text{max-flow}(s, v)$.

Thus the advantage of using network coding is that this problem reduces to polynomial time. Network coding is intuitively easier because we are not looking for disjoint structures in this case. The multicast problem here is just a collection of single source single terminal path finding problems where the resultant solution is just the union of the individual paths. Basically, we find a set of flows P_i from s to T_i for all i and interpret the flow values to find a subgraph over which network coding can be done.

In the network coding linear programming problem, we have a flow vector f^i and each individual flow ($f^i, i = 1, 2, \dots, |T|$) satisfies flow conservation. However, the flows are allowed to overlap resulting in a conceptual flow that satisfies the capacity constraint. (Conceptual flow need not satisfy the flow conservation)

Our max flow problem can be written in a more concise form as:

$$\begin{aligned} & \text{maximize } R \\ & \text{subject to } \sum_{e \text{ out of } j} f^i(e) - \sum_{e \text{ into } j} f^i(e) = \sigma_j^{(i)} \text{ where } j \in V \\ & \quad 0 \leq f^i(e) \leq c_e \text{ for } e \in E \text{ and } i = 1, 2, \dots, |T| \end{aligned}$$

where $\sigma_j^{(i)} = \begin{cases} 0 & j \in V - \{s, T_i\} \\ R & j = s \\ -R & j = T_i \end{cases}$. Note that in this LP we have a set of flow variables f^i for each terminal T_i and that we are trying to find the maximum R that can be supported simultaneously

to all the T_i 's. Flows corresponding to different terminals are allowed to share edges as long as the maximum value of the flow variable on any edge is at most its capacity.

More generally, we are interested in solving the minimum cost multicast problem. In this variant of the problem, there is a cost associated with the usage of each edge e . We denote this by $\text{cost}(e)$. We specify the value of the flow that we want to support and search for flow allocations that minimize the cost. Our linear programming problem now becomes,

$$\begin{aligned} & \text{minimize } \sum_{e \in E} \text{cost}(e) z_e \\ & \text{subject to } \sum_{e \text{ out of } j} f^i(e) - \sum_{e \text{ into } j} f^i(e) = \sigma_j^{(i)} \text{ where } j \in V \\ & \quad 0 \leq f^i(e) \leq z_e \leq c_e \text{ for } e \in E \text{ and } i = 1, 2, \dots, |T|. \end{aligned}$$

i.e. we now introduce a variable z_e corresponding to the maximum of the flow variables on each edge e and ensure that it remains lower than the capacity of e . The objective function is modified to be the weighted sum of the z_e 's. Note that in this formulation we shall now have

$$\sigma_j^{(i)} = \begin{cases} 0 & j \in V - \{s, T_i\} \\ R_1 & j = s \\ -R_1 & s = T_i \end{cases}, \text{ where } R_1 \text{ is a constant that can be determined in advance and has}$$

to be less than the value of the max flow that can be supported to all the terminals.