IP Formulation of MST

Let $A(S)$ be the set of arcs contained in the subgraph of $G = (N, A)$ induced by the node set $S$. Let $x_{ij}$ be a 0-1 variable that indicates whether we select arc $(i, j)$ to be in the spanning tree.

Minimize

$$\sum_{(i,j) \in A} c_{ij} x_{ij}$$

subject to

$$\sum_{(i,j) \in A} x_{ij} = n - 1$$

$$\sum_{(i,j) \in A(S)} x_{ij} \leq |S| - 1 \quad \forall \text{sets of nodes } S$$
LP Relaxation

- For any LP, we can use reduced costs and complementary slackness optimality conditions to assess whether a given feasible solution is optimal.

- Notice that when $S = N$, constraint (3) is redundant. So, we associate a potential $\mu_S$ with every set $S$ of nodes.

- From the dual, we find that $\mu_N$ is free but $\mu_S \geq 0$.

- Then, the reduced cost of arc $(i, j)$ is $c_{ij}^\mu = c_{ij} + \sum_{A(S) : (i, j) \in A(S)} \mu_S$. 
Results

Lemma 1. A solution $x$ of the MST problem is an optimal solution to the LP relaxation of the IP formulation if and only if we can find potentials $\mu_S$ defined on node sets $S$ so that

$$c_{ij}^\mu = \begin{cases} 0 & \text{if } x_{ij} > 0 \\ \geq 0 & \text{if } x_{ij} = 0 \end{cases}$$

Theorem 1. [13.9] If $x$ is the solution generated by Kruskal’s Algorithm, then $x$ solves both the integer program and its LP relaxation.
Defining Potentials

- Set $\mu_N$ to the negative cost of the last arc added to the tree.
- Let $S(i, j)$ be the node component created by adding arc $(i, j)$ to the tree.
- As the algorithm progresses, when it adds arc $(p, q)$ to the tree, it combines node component $S(i, j)$ with one or more other nodes to define a larger component.
- Set $\mu_{S(i, j)} = c_{pq} - c_{ij}$. 
Proving Optimality

- Check reduced cost of every arc. What do we find?

**Theorem 2. [13.10]** The polyhedron defined by the LP relaxation of the packing formulation of the MST problem has integer extreme points.
Matroids

• Notice the the algorithms for finding minimum weight spanning trees depend on two properties:
  – Any acyclic subgraph with fewer than \( n - 1 \) edges can always be extended to a spanning tree.
  – If we have two spanning trees, one of which includes one more edge than the other, the smaller can be extended with an edge from the larger.

• An independence system is \((E, \mathcal{F})\) such that
  – \(\mathcal{F}\) is a collection of subsets of \(E\)
  – If \(F_1 \in \mathcal{F}\) and \(F_2 \subset F_1\), then \(F_2 \in \mathcal{F}\)

• A matroid satisfies the additional property that if \(F_1, F_2 \in \mathcal{F}\) and \(|F_1| = |F_2| + 1\), then \(\exists e \in F_1 \setminus F_2\) such that \(F_2 \cup \{e\} \in \mathcal{F}\).
Examples of Matroids

- Graphic matroid
- Partition matroid
- Metric matroid
Generalizing from Spanning Trees

- Everything we learned from spanning trees can be generalized.
- All maximal independent sets have the same cardinality and are called *bases*.
- A spanning tree is a basis of the graphic matroid.
- A fundamental property of matroids is that it is always possible to find a basis of minimum weight using a *greedy algorithm*.
- In fact, an independence system is a matroid if and only if the greedy algorithm always finds a basis of minimum weight.
Red-Blue Algorithm for the Minimum Spanning Tree Problem

- Start with all edges uncolored.
- The Blue Rule
  - Find a cut with no BLUE edges.
  - Pick an edge of minimum weight and color it BLUE.
- The Red Rule
  - Find a cycle containing no RED edges.
  - Pick an uncolored edge of maximum weight and color it RED.
- Arbitrary application of the RED and BLUE rules result in a minimum weight spanning tree.
Generalizing to Matroids

- A *cycle* is a setwise minimal dependent set.
- A *cut* is a setwise maximal subset that intersects all maximal independent sets.
- The Red-Blue Algorithm can be applied to any matroid to find a basis of minimum weight.
- Matroids arise naturally in many contexts.
- We will see them later in the assignment problem context.
Matching Problems

- MST and Matching Problems are two combinatorial optimization problems that are defined over graphs with a weight associated with each arc.

- A matching in a graph is a set of edges with the property that no two share a node.

- Two well-known matching problems
  - Find a matching that has as many edges as possible.
  - Given weights for each edge, find a matching with the largest total weight.

- Matching algorithms use the concept of augmentations, but detecting and performing augmentations efficiently is more complicated here.
Definitions

• Given a graph $G = (N, A)$, the objective of the matching problem is to find a maximum matching $M$ of $G$.

• We say that the matching is complete or perfect when the cardinality of $M$ is $\left\lfloor \frac{|N|}{2} \right\rfloor$.

• Given a matching $M$ in $G$, edges in $M$ are called matched edges; others are free edges.

• Nodes that are not incident upon any matched edge are called exposed; remaining are matched.
Definitions (con’t)

• A path \( p = [u_1, u_2, \cdots, u_k] \) is called alternating if edges \((u_1, u_2), (u_3, u_4) \cdots \) are free and \((u_2, u_3), (u_4, u_5) \cdots \) are matched.

• An alternating path \( p \) is called augmenting if both \( u_1 \) and \( u_k \) are exposed.
Augmenting a Matching

Lemma 2. Let $P$ be the set of edges on an augmenting path $p = [u_1, u_2, \cdots, u_{2k}]$ in a graph $G$ with respect to the matching $M$. Then $M' = M \oplus P$ is a matching of cardinality $|M| + 1$.

Proof:
Maximum Matching

**Theorem 3.** A matching $M$ in a graph $G$ is maximum if and only if there is no augmenting path in $G$ with respect to $M$.

- Theorem characterizes maximum matchings in terms of augmenting paths.
- Like maximum flow, it suggests an algorithm: Start with any matching. Repeatedly discover augmenting paths.
- All known algorithms for matchings are based on this idea, but the details are quite involved...except for the case of bipartite graphs.
Bipartite Matching and Network Flow

• A graph $G = (N, A)$ is a bipartite graph if we can partition its node set into two subsets $N_1$ and $N_2$ so that for each arc $(i, j) \in A$ either (i) $i \in N_1$ and $j \in N_2$ or (ii) $i \in N_2$ and $j \in N_1$.

• We can reduce bipartite matching problem to maximum flow problem for simple networks and solve efficiently by making use of any algorithm for maximum flow.

• How can we convert the bipartite matching problem into an equivalent maximum flow problem?
Lemma 3. The cardinality of the maximum matching in a bipartite graph equals the value of the maximum flow in the corresponding maximum flow network.

Proof: 1. Given any matching $M$, we can construct a feasible flow in $N(G)$ with value $|M|$.

2. Given a maximum flow in $N(G)$, we can construct a matching with cardinality of the maximum flow value.
Notes on Maximum Cardinality Matching

• We can solve the bipartite matching problem in $O(\sqrt{nm})$ time.

• Asymptotically fastest algorithm for bipartite matching.

• Non-Bipartite Matching
  – Reduction to maximum flow does not seem to carry over.
  – Augmenting path theorem holds for general graphs, so idea of repeatedly augmenting can be extended. Finding augmenting paths is more difficult with non-bipartite structure.
Weighted Matching

• Given the graph $G = (N, A)$ with a corresponding weight $w_{ij}$ for each arc $(i, j)$, the objective of the weighted matching problem is to find a matching with the largest possible sum of weights.

• Assumptions
  – Underlying graph is complete.
  – Underlying graph has even number of nodes.
  – (For bipartite), underlying graph has node sets that are equal in size.

• Another name is Assignment Problem.
Assignment Problem

• Write the IP formulation for the Assignment Problem.
• Assignment Problem is a special case of which network flow problem?
• How can we solve the Assignment Problem?
Matching and the Postman Problem

- Given an undirected graph \((G, A)\), the postman problem is to find the shortest tour that traverses each edge at least once.

- A graph for which it is possible to do this while traversing each edge exactly once is called Eulerian.

- An undirected graph is Eulerian if and only if every node has even degree.

- How can we use this fact to solve the postman problem?

- How can we extend this to directed graphs?