CMPUT 675: Topics in Combinatorics and Optimization **Fall 2016**

Lecture 19 (November 12, 2016): Minimal Trees and Arboresences

Lecturer: Zachary Friggstad *Scribe: Jacob Denson*

We will now discuss how minimal spanning trees fits in the general context of linear programming, in particular the duality properties of the LP formulation. We then use these techniques to analyze the similar problem of minimal arboresences.

19.1 Linear Programming and Minimal Spanning Trees

Minimal Spanning Trees are a standard part of the undergraduate algorithms curriculum, a standard example of how a 'greedy' strategy may be applied to problems with have been carefully analyzed to obtain very fast solutions. Here we shall discuss the standard minimal spanning tree algorithm, but we will use linear programming to gain intuition on why the standard greedy approach works. Later on in the course we will gain more insight when we learn the theory of matroids.

Let G be an undirected graph, together with a (possibly-negative) edge cost function c. A **spanning tree** T on G is a connected subset of edges containing no cycles (a tree) such that every vertex is covered by an edge in the tree. Note that this definition is equivalent to T having no cycles, and $n-1$ edges (a fact proven most easily by induction, by removing vertexes and edges from T which form the leaves of T). There is another equivalent definition which is more important to the spanning tree problems we'll discuss. T is a spanning tree if it has $n-1$ edges, and $|E(S)| \leq |S| - 1$ for all vertex sets S (where $E(S) = \{uv : u, v \in S, uv \in T\}$).

The minimum spanning tree problem is self explanatory – given G, find a spanning tree containing the fewest edges. We can formulate the problem as an LP in exponentially many constraints in functions $x : E \to \mathbf{R}$,

$$\begin{aligned} \min \quad & \sum_e c(e)x(e) \\ \text{s.t.} \quad & x(E(S)) \geq |S| - 1 \quad \forall S \subsetneq V \\ & x(E(V)) = n - 1 \\ & 0 \leq x \end{aligned}$$

Note that the constraint $x \leq 1$ is already encoded in the problem by taking S to be a pair of vertices corresponding to an edge. Our discussion above entails that any integral solution to this algorithm corresponds to a spanning tree in G. It can be shown that all extreme points are integral, but we shall only show that we can choose an optimal extreme point which is integral.

In it's current form, the minimal spanning tree linear program is unfeasible, having far too many constraints. We obtain a much better optimization problem if we switch to the dual of this program. This can be motivated by the fact that x behaves more like a linear functional than a vector in this linear program (the constraints of the program are expressed via x's operations on the vector space $\mathbf{R} \cdot E$ generated by the edges E), so that the program is the dual of some other, more easily understood linear program. The dual program finds functions

$y : 2^V - \{\emptyset\} \to \mathbf{R}$ which satisfy

$$\begin{aligned} \max \quad & -\sum (1 - |S|) y(S) \\ \text{s.t.} \quad & \sum_{e \in E(S)} y(S) \geq -c(e) \quad \forall e \in E(S) \\ & y(S) \geq 0 \qquad \forall S \neq V \end{aligned}$$

Kruskal's algorithm, a method for finding the minimal spanning tree, can be viewed as a combinatorial method to solving the dual LP of the spanning tree problem. We first recall the simple, greedy method to form a spanning tree. We can verify this algorithm's correctness using the duality of linear programming. Suppose

Algorithm 1 Kruskal's Algorithm

1: Set $T = \emptyset$, $K = E$.
2: **while** T is not a spanning tree **do**
3: Remove $e \in K$ with minimal weight.
4: Append e to T if it connects two connected components of T.
5: **return** T

that T contains the edges $e_1, \ldots, e_n$, which are placed in the order they were added to T. Let S_k be the component containing e_k in the graph consisting only of the edges $e_1, \ldots, e_k$ (it is the component that was freshly merged together in the k'th iteration of the algorithm). Let $x : E \to \{0, 1\}$ be the indicator function of T, and $y : 2^V - \{\emptyset\} \to \mathbf{R}$ be defined by letting $y(S_i) = c(e_j) - c(e_i)$, where j is the smallest index greater than i such that one of the endpoints of e_j contains points in X_i, and define $y(S_n) = y(V) = -c(e_n)$. Define $y(S) = 0$ otherwise. For any edge e, by the telescoping sum property of our definition we have

$$\sum_{e \subset S} y(S) = -c(e_i)$$

where i is the smallest index such that X_i contains both endpoints of e. The way we selected edges guarantees that $c(e_i) \leq c(e)$, so that our constraints are satisfied, and our tight for $e_1, \ldots, e_n$.

We now verify complementary slackness, so that x is verified optimal. If $x(e) > 0$, then our calculation above shows

$$-\sum_{e \subset S} y(S) = c(e)$$

If $y(S) > 0$, then $S = S_i$ for some i, hence S_i is a connected tree, and contains $n-1$ vertices, so $x(E(S)) = |S|-1$. Complementary slackness guarantees that x and y are optimal solutions to their corresponding algorithms, verifying correctness of Kruskal's algorithm.

19.2 Arboresences

An (out) **arboresence** in a directed graph with root r is a subset T of $n-1$ edges, such that there is a unique directed path from r to any other vertex. Given a cost function $c \geq 0$ and vertex r, we want to try and find the min-cost arboresence rooted at r. Like the minimal cost spanning tree problem, we can express this problem as an LP,

$$\begin{aligned} \min \quad & \sum_e c(e) x(e) \\ \text{s.t.} \quad & x(\delta^{\text{in}}(U)) \geq 1 \quad \forall\, \emptyset \subsetneq U \subset V - \{r\} \\ & x \geq 0 \end{aligned}$$

If T is an arbitrary arboresence at v, then the corresponding characteristic function certainly is a feasible solution to the problem. The problem with this LP is that there are solutions (possibly even optimal ones) which don't look like arboresences. However, we shall not use the LP to solve linear programs, but instead use properties of linear programs to guarantee optimality.

We can form a simple argument to show the existence of optimal integer solutions which are arboresence. First, note that we only every have upper bounds to the algorithm, and there is no need to have $x(e) > 1$ to satisfy any of the constraints, due to the positivity of x, so we may always assume that an optimal integral solution takes value in $\{0, 1\}$, and thus correspond to characteristic functions of edges. Second, we see that the subgraph formed contains paths from v to any other vertex. Since $x(\delta^{\text{in}}(V - r) \geq 1$, there is an edge connecting s to some over vertex v_1. Then $x(\delta^{\text{in}}(V - \{s, v_1\}) \geq 1$, so there is some edge from r to v_1, or v_1 to v_2. In either case, there is an edge to v_2. Continuing this process gives you a path to any vertex in the graph. We may now assume our subgraph is a tree, because we can always remove edges to obtain an algorithm that is at least as optimal. Thus we have argued that there is an optimal solution which is an arboresence, like we are looking for.

We now show, in fact, that there are optimal LP solutions that are integral and correspond to an arborescence.

Let us form the dual LP, we will describe an algorithm that constructs an arborescence and a corresponding feasible dual solution to certify optimality of the arborescence.

$$\begin{aligned} \max \quad & \sum_{\emptyset \subsetneq S \subseteq V - \{r\}} y(S) \\ \text{s.t.} \quad & \sum_{e \in \delta^{\text{in}}(S)} y(S) \leq c(e) \\ & y \geq 0 \end{aligned}$$

We now describe the algorithm which gives us minimal cost arboresences.

Notation: if we contract some nodes of a graph G then for any vertex of v we say the **image** of v in the contracted graph is the vertex it was contracted in to (or the original vertex itself if it was not contracted). Similarly, for an edge $e = uv$ whose endpoints are not contracted to the same vertex (so the edge exists in the contracted graph) we say the image of e is the edge between the images of u and v.

The idea is simple, pick the cheapest edge entering any non-root vertex. If this forms an arborescence (equivalently, there are no cycles) then we are done. Otherwise, we view the cost of each edge uv bought so far as having been paid by v. We contract all cycles and modify the remaining edge costs; a surviving edge uv has its cost decreased by the amount v paid so far. Recursively, we find an arborescence. For any cycle (which was contracted to a single node), a single edge uv was bought that entered this cycle. We remove the cycle edge entering v and keep uv instead.

Algorithm 2 Edmond's Algorithm

1: Set $F = \{e_w : w \in V - \{r\}\}$, where e_w is the cheapest edge entering $w \neq r$ ▷ break ties arbitrarily
2: **if** F has no cycle **then**
3: **return** F
4: **else**
5: $G' \leftarrow$ the graph obtained by contracting each cycle of F.
6: For $v \in V$, let $\alpha(v)$ be the image of v in G'.
7: For $e = uv \in E$ where $\alpha(u) \neq \alpha(v)$ let $\alpha(e)$ be the image of e in G'.
8: $c'(\alpha(uv)) = c(uv) - c(e_v)$ for $uv \in E$ with $\alpha(u) \neq \alpha(v)$. ▷ preserves nonnegativity of costs
9: $F' \leftarrow$ an arborescence recursively found for G', c'.
10: $F \leftarrow F \cup F' - \{e_w \in F : w$ on a cycle and $\alpha(uw) \in F'$ for some $uw \in E\}$.
11: **return** F

Correctness (in the sense that an arborescence is returned) is fairly simple to argue by induction. If F has no cycle then if we start at a vertex v and iteratively follow the single incoming edge we must reach r eventually (or else we found a cycle). Inductively, if we find an arborescence in G' then we can reach every vertex in G using $F \cup F'$. Removing the cycle edge e_w when $\alpha(uw) \in F'$ still leaves all vertices on the cycle reachable because any walk from r will reach w in the cycle, and then all other vertices in the cycle can be reached using the remaining edges of F.

To see optimality, consider the following recursive construction of a dual.

If F is acyclic, let $y(\{v\}) = c(e_v)$ for each $v \neq r$ and $y(S) = 0$ for all other S. Feasibility is easily checked, every $e \in F$ has the corresponding dual constraint being tight, trivially the only $y(S) > 0$ have $S = \{v\}$ for some $v \neq r$ and these have indegree 1. So complementary slackness holds, meaning F is a min-cost arborescence.

Inductively, suppose we find a dual solution y' certifying optimality of F' as a min-cost arborescence of G'. Set $y(\{v\}) = c(e_v)$ for $v \neq r$, $y(S) = y'(\alpha(S))$ if S does not cut a cycle (i.e. all vertices of any cycle either lie all in S or all outside of S), and $y(S) = 0$ otherwise. It is straightforward to verify feasibility of y and that complementary slackness holds with y and the natural integer solution corresponding to F.

Summarizing:

Theorem 1 *Algorithm 2 returns an arborescence F whose cost equals the optimum solution value of the min-cost arborescence LP relaxation.*

References

B. Korte and J. Vygen, Combinatorial Algorithms - Theory and Algorithms, *Springer-Verlag, Berlin Heidelberg New York Tokyo Paris Milano*, 2012, pg. 131-141.