Massachusetts Institute of Technology
18.433: Combinatorial Optimization
Michel X. Goemans

Handout 15
April 19th, 2007

Lecture notes on the arborescence problem

Given a digraph $D = (V, A)$ and a special root vertex $r \in V$, an r-arborescence (or just arborescence) is a spanning tree (when viewed as an undirected graph) directed away from r. Thus, in a r-arborescence, every vertex is reachable from the root r. As an r-arborescence has no arc incoming to the root, we assume that D has no such arc.

r-arborescences can be viewed as sets simultaneously independent in two matroids. Let G denote the undirected counterpart of D obtained by disregarding the directions of the arcs. Note that if we have both arcs $a_1 = (u, v)$ and $a_2 = (v, u)$ in D then we get two undirected edges also labelled a_1 and a_2 between u and v in G. Define $M_1 = (A, \mathcal{I}(M_1))$ the graphic matroid corresponding to G, and $M_2 = (A, \mathcal{I}(M_2))$ the partition matroid in which independent sets are those with at most one arc incoming to every vertex $v \neq r$. In other words, we let $\mathcal{I}(M_2) = \{F : |F \cap \delta^-(v)| \leq 1 \text{ for all } v \in V \setminus \{r\}\}$ where $\delta^-(v)$ denotes the set $\{(u, v) \in A\}$ of arcs incoming to v. Thus, any r-arborescence is independent in both matroids F_1 and F_2. Conversely, any set T independent in both M_1 and M_2 and of cardinality $|V| - 1$ (so that it is a base in both matroids) is an r-arborescence.

The minimum cost r-arborescence is the problem of, given a directed graph $D = (V, A)$, a root vertex $r \in V$ and a cost c_a for every arc $a \in A$, finding an r-arboresnce in D of minimum total cost. This can thus be viewed as a weighted matroid intersection problem and we could use the full machinery of matroid intersection algorithms and results. However, here, we are going to develop a simpler algorithm using notions similar to the Hungarian method for the assignment problem. We will assume that the costs are nonnegative.

As an integer program, the problem can be formulated as follows. Letting x_a be 1 for the arcs of an r-arborescence, we have the formulation:

$$OPT = \min \sum_{a \in A} c_a x_a$$

subject to:

$$\begin{aligned} \sum_{a \in \delta^-(S)} x_a &\geq 1 && \forall S \subseteq V \setminus \{r\} \\ \sum_{a \in \delta^-(v)} x_a &= 1 && \forall v \in V \setminus \{r\} \\ x_a &\in \{0, 1\} && a \in A. \end{aligned}$$

In this formulation $\delta^-(S)$ represents the set of arcs $\{(u, v) \in A : u \notin S, v \in S\}$. One can check that any feasible solution to the above corresponds to the incidence vector of an r-arborescence. Notice that this optimization problem has an exponential number of constraints. We are going to show that we can relax both the integrality restrictions to

$x_a \geq 0$ and also remove the equality constraints $\sum_{a \in \delta^-(v)} x_a = 1$ and still there will be an r-arboresence that will be optimum for this relaxed (now linear) program. The relaxed linear program (still with an exponential number of constraints) is:

$$
(P) \qquad \begin{array}{rll}
LP = \min & \sum_{a \in A} c_a x_a & \\
\text{subject to:} & & \\
& \sum_{a \in \delta^-(S)} x_a \geq 1 & \forall S \subseteq V \setminus \{r\} \\
& x_a \geq 0 & a \in A.
\end{array}
$$

The dual of this linear program is:

$$
(D) \qquad \begin{array}{rll}
LP = \max & \sum_{S \subseteq V \setminus \{r\}} y_S & \\
\text{subject to:} & & \\
& \sum_{S : a \in \delta^-(S)} y_S \leq c_a & \\
& y_S \geq 0 & S \subseteq V \setminus \{r\}.
\end{array}
$$

The algorithm will be constructing an arborescence T (and the corresponding incidence vector x with $x_a = 1$ whenever $a \in T$ and 0 otherwise) and a feasible dual solution y which satisfy complementary slackness, and this will show that T corresponds to an optimum solution of (P), and hence is an optimum arborescence. Complementary slackness says:

1. $y_S > 0 \Longrightarrow |T \cap \delta^-(S)| = 1$, and
2. $a \in T \Longrightarrow \sum_{S : a \in \delta^-(S)} y_S = c_a$.

The algorithm will proceed in 2 phases. In the first phase, it will construct a dual feasible solution y and a set F of arcs which has a directed path from the root to every vertex. This may not be an r-arborescence as there might be too many arcs. The arcs in F will satisfy condition 2 above (but not condition 1). In the second phase, the algorithm will remove unnecessary arcs, and will get an r-arborescence satisfying condition 1.

Phase 1 is initialized with $F = \emptyset$ and $y_S = 0$ for all S. While F does not contain a directed path to every vertex in V, the algorithm selects a set S such that (i) inside S, F is strongly connected (i.e. every vertex can reach every vertex) and (ii) $F \cap \delta^-(S) = \emptyset$. This set S exists since we can contract all strongly connected components and in the resulting acyclic digraph, there must be a vertex (which may be coming from the shrinking of a strongly connected component) with no incoming arc (otherwise tracing back from that vertex we would either get to the root or discover a new directed cycle (which we could shrink)). Now we increase

y_S as much as possible until a new inequality, say for arc a_k, $\sum_{S:a_k\in\delta^-(S)} y_S \leq c_{a_k}$ becomes an equality. In so doing, the solution y remains dual feasible and still satisfies condition 2. We can now add a_k to F without violating complementary slackness condition 2, and then we increment k (which at the start we initialized at $k = 1$). And we continue by selecting another set S, and so on, until every vertex is reachable from r in F. We have now such a set $F = \{a_1, a_2, \cdots, a_k\}$ and a dual feasible solution y satisfying condition 2.

In step 2, we eliminate as many arcs as possible, but we consider them in *reverse order* they were added to F. Thus, we let i go from k to 1, and if $F \setminus \{a_i\}$ still contains a directed path from r to every vertex, we remove a_i from F, and continue. We then output the resulting set T of arcs.

The first claim is that T is an arborescence. Indeed, we claim it has exactly $|V| - 1$ arcs with precisely one arc incoming to every vertex $v \in V \setminus \{r\}$. Indeed, if not, there would be two arcs a_i and a_j incoming to some vertex v; say that $i < j$. In the reverse delete step, we should have removed a_j; indeed any vertex reachable from r through a_j could be reached through a_i as well (unless a_i is unnecessary in which case we could get rid of a_i later on).

The second (and final) claim is that the complementary slackness condition 1 is also satisfied. Indeed, assume not, and assume that we have a set S with $y_S > 0$ and $|T \cap \delta^-(S)| > 1$. S was chosen at some point by the algorithm and at that time we added $a_k \in \delta^-(S)$ to F. As there were no other arcs in $\delta^-(S)$ prior to adding a_k to F, it means that all other arcs in $T \cap \delta^-(S)$ must be of the form a_j with $j > k$. In addition, when S was chosen, F was already strongly connected within S; this means that from any vertex inside S, one can go to any other vertex inside S using arcs a_i with $i < k$. We claim that when a_j was considered for removal, it should have been removed. Indeed, assume that a_j is needed to go to vertex v, and that along the path P to v the last vertex in S is $w \in S$. Then we could go to v by using a_k which leads somewhere in S then take arcs a_i with $i < k$ (none of which have been removed yet as $i < k < j$) to $w \in S$ and then continue along path P. So a_j was not really necessary and should have been removed. This shows that complementary slackness condition 1 is also satisfied and hence the arborescence built is optimal.