Uncrossing Partitions

Jochen Könemann and David Pritchard
Department of Combinatorics and Optimization, U Waterloo

Motivation: Steiner Trees

The *Steiner tree problem* is a classical combinatorial optimization problem where the goal is to cheaply connect certain nodes in a graph. The input graph $G = (V, E)$ has edge costs $c_e > 0 \ \ \forall e \in E$ and its node set V is divided into two disjoint sets: *terminals* (denoted R) and *Steiner nodes*. A *Steiner tree* is a tree that connects all terminals and has no Steiner leaves (see Fig. 1(a)). Finding a minimum-cost Steiner tree is NP-hard, even to within 1% [Chlebík & Chlebíková 02], so we seek good polynomial-time approximation algorithms.

Background on Linear Programs (LPs)

Linear programming is a fruitful and intensely-studied technique in combinatorial optimization. One typically assigns a variable to each possible element of a feasible solution (e.g., one variable x_e for each usable edge e in a network design problem). The *characteristic vector of* S assigns 1 to each element in S and 0 to each element not in S. The usual goal is to design an LP so that the *0-1 integral feasible solutions* of the LP are the characteristic vectors of feasible objects for the combinatorial optimization problem at hand. For NP-hard problems the linear programming approach can be used to design approximation algorithms or to implement exact solvers via integer programming.

Reduction to Spanning Hypergraphs

A *full component* of a Steiner tree is an inclusion-maximal subtree with all its leaves terminals and all its internal nodes Steiner. A Steiner tree's edges can be partitioned uniquely into full components, e.g. by splitting at internal terminals (see Fig. 1(b)). All nontrivial approximation algorithms for the Steiner tree problem work in the other direction: compute the cheapest full components of a graph, and use that information to build a cheap Steiner tree.

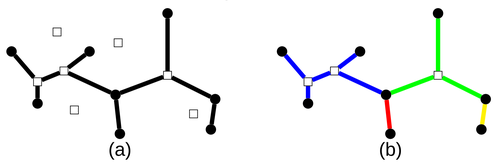

Figure 1: Black nodes are terminals and white nodes are Steiner nodes. (a): a Steiner tree. Note that some Steiner nodes are unused. (b): the edges of the tree are partitioned into full components; there are four.

We rephrase the problem. A *hypergraph* $(V, \mathcal{E})$ has node set V, hyperedge set $\mathcal{E} \subseteq \{E \subseteq V \mid |E| \geq 2\}$ and costs C_E for each $E \in \mathcal{E}$. It is *spanning* if for every partition of V into two nonempty parts, some hyperedge in $\mathcal{E}$ meets both parts. Let $\mathcal{K}$ denote all sets of terminals of size at least 2,

$$\mathcal{K} := \{K \subseteq R \mid |K| \geq 2\},$$

and for a subhypergraph $(R, \mathcal{E})$ of $(R, \mathcal{K})$ (i.e., $\mathcal{E} \subset \mathcal{K}$) its *cost* is defined to be $\sum_{K \in \mathcal{E}} C_K$.

Proposition 1. *Given a Steiner tree instance, define C_K to be the cheapest cost of any full component on leaf set K. The cheapest Steiner tree has the same cost as the cheapest spanning subhypergraph of $(R, \mathcal{K})$.*

Preprocessing

Rather than all $\sim 2^{|R|}$ full components, one usually computes only those on at most k terminals, for a fixed k. This *k-preprocessing* can be done in polynomial time [Dreyfus & Wagner 72] and may increase the optimum cost (e.g., see Fig. 2), but only by a factor of $1 + O(1/\log k)$ [Du, Zhang & Feng 91]. So for $k \to \infty$ preprocessing has negligible cost.

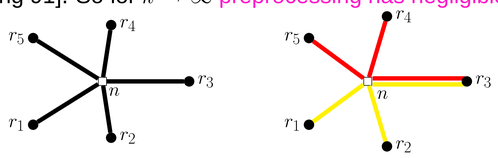

Figure 2: Left: an instance of the Steiner tree problem with five terminals; all edges have cost 1. Any solution using full components on less than 5 terminals must use some edge(s) twice. Right: a solution under 3-preprocessing is shown; the optimal cost has gone up from 5 to 6.

A Spanning Hypergraph LP Using Partitions

A *partition (of R)* is a collection of nonempty sets (*parts*) $\{\pi_1, \pi_2, \ldots, \pi_t\}$ such that each $r \in R$ occurs in exactly one π_i; its *rank*, denoted $r(\pi)$, is the number of parts it has (here t). Given a partition π of R and any $K \in \mathcal{K}$, we define the *rank contribution* $\mathtt{rc}_K^\pi$ to be the number of parts of π spanned by K, minus one. Equivalently, $\mathtt{rc}_K^\pi$ is the decrease in rank incurred by π if we merge together all parts intersecting K. E.g., see Fig. 3.

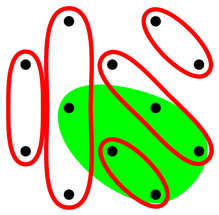

Figure 3: The black dots are the terminal set R. In red is a partition π of R, and in green is a hyperedge K. Here $\mathtt{rc}_K^\pi = 2$.

The following LP, which generalizes a graph spanning tree LP due to [Chopra 89], is central to our study.

$$\text{minimize} \quad \sum_{K \in \mathcal{K}} C_K x_K \qquad (\mathcal{P})$$

$$\forall K \in \mathcal{K}: \quad x_K \geq 0 \qquad (1)$$

$$\forall \text{ partitions } \pi \text{ of } R: \quad \sum_{K \in \mathcal{K}} x_K \mathtt{rc}_K^\pi \geq r(\pi) - 1 \qquad (2)$$

Theorem 2 (Könemann & Tan 06)**.** *Let x be 0-1 integral. Then x is feasible for $(\mathcal{P})$ iff x is the characteristic vector of the hyperedge set of a spanning subhypergraph of $(R, \mathcal{K})$.*

What Does Uncrossing Mean?

Uncrossing has played a critical role in recent work in the area of network design, e.g., [Goemans 06, Jain 98, Lau et al 07, Lau & Singh 07, Melkonian & Tardos 04]. In that setting one uncrosses *sets* in an LP with subtour elimination constraints, rather than partitions as we do here.

In concrete terms, uncrossing is applied to a feasible solution of a linear program (e.g., $(\mathcal{P})$). The goal is to transform an arbitrary solution into an *uncrossed* one without worsening the objective value. If this can be done then in particular there is some optimal uncrossed solution. The idea is that uncrossed solutions have special useful properties.

Under the extra constraint $\sum_K x_K(|K| - 1) = |R| - 1$ our partition uncrossing is in fact equivalent to known subtour uncrossing techniques. In general, however, it seems that a more delicate uncrossing operation is required.

Refinement, Crossing, Chains

Given two partitions π and π' of R, we say that *π' refines π* if each part of π' is contained in some part of π. If neither π nor π' refines the other, we say that π and π' *cross*. It is easy to see that if π'' refines π' and π' refines π, then π'' refines π. A *chain* is a sequence $(\pi^{[1]}, \ldots, \pi^{[t]})$ where $\pi^{[j]}$ refines $\pi^{[i]}$ for all $t \geq j > i \geq 1$; equivalently, a chain is a set of partitions of which no two cross. See Figure 4.

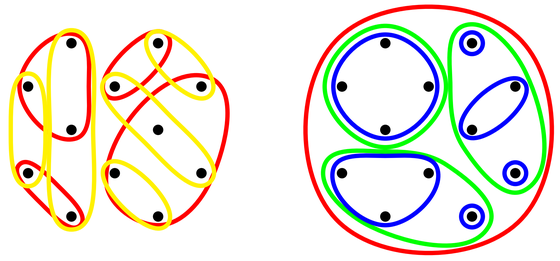

Figure 4: Left: two partitions that cross. Right: a chain of size 3; the blue partition refines the green one which in turn refines the red one.

Main Technical Result

For any feasible point x of the program $(\mathcal{P})$, we say that partition π is *tight* for x if the constraint (2) corresponding to π holds with equality. The following proposition is a straightforward consequence of the general theory of polyhedra.

Proposition 3. *Every basic solution x of $(\mathcal{P})$ is characterized by a family τ of partitions and a set $\mathcal{E} \subset \mathcal{K}$ (its support), with $|\tau| = |\mathcal{E}|$. Namely, x is the unique point such that*

- *$x_E > 0$ if and only if $E \in \mathcal{E}$*
- *all partitions τ are tight for x.*

Our "uncrossing technique" effectively allows us to replace crossed pairs of tight partitions with non-crossed ones. We obtain the following.

Theorem 4. *In Proposition 3, we may also assume*

- *the family τ of tight partitions is a chain.*

Consequences

Notice that in a chain $(\pi^{[1]}, \ldots, \pi^{[t]})$, we have $1 \leq r(\pi^{[1]}) < r(\pi^{[2]}) < \cdots < r(\pi^{[t]}) \leq |R|$. Hence t, the cardinality of the chain, is at most $|R|$ and so Theorem 4 gives the following.

Corollary 5. *Each basic solution of $(\mathcal{P})$ has at most $|R|$ nonzero coordinates (out of $\Theta(|R|^k)$ coordinates in total).*

The original motivation for this work was a comparison of two full component-based LPs for the Steiner tree problem. The uncrossing argument allows us to prove the following.

Theorem 6. *In hypergraphs derived from Steiner tree instances, the objective value of $(\mathcal{P})$ is equal to the objective value of a subtour-based LP due to [Warme 98].*

Like our LP, Warme's LP models Steiner trees via hypergraphs. Theorem 6 does not hold for arbitrary hypergraphs; the key feature of Steiner-derived instances is that when $S \supset T, C_S \geq C_T$ (the costs are *non-decreasing*).

LP duality, together with uncrossing, gives another (surprising) result. Let T be a (non-hyper) tree on node set R. Let T/K denote the graph obtained from T by contracting the node set K and let $mst(T/K)$ denote the minimum cost of any spanning tree of T/K. Define the *gain* of K in T to be the spanning tree cost decrease when K is used,

$$gain_T(K) := cost(T) - mst(T/K) - C_K,$$

and define $t^{\mathcal{K}}$ to be the maximum cost of any tree T such that for *all* K in $\mathcal{K}$, $gain_T(K) \leq 0$ (a *gainless* tree).

Theorem 7. *$t^{\mathcal{K}}$ is equal to the optimum value of $(\mathcal{P})$.*

This quantity $t^{\mathcal{K}}$ was used in [Karpinski & Zelikovsky 97] to analyze a then-best 1.64-approximation algorithm for the Steiner tree problem. In fact, $t^{\mathcal{K}}$ was used as an upper bound in one place and as a lower bound in another; this resembles an LP optimal value already since the optimal value of a minimization LP is a lower (resp. upper) bound on the cost of any feasible primal (resp. dual) solution.

Future Work

The *bidirected cut relaxation* is a well-known Steiner tree LP with many equivalent formulations [Goemans & Myung 93]. It is no stronger than $(\mathcal{P})$, using Theorem 6 and [Polzin & Vahdati Daneshmand 03]. In a *preprocessed graph* all original edges are deleted and then a clone of each full component in $\mathcal{K}$ is added; clones meet only at terminals.

Conjecture 8. *In a preprocessed graph, the bidirected cut relaxation is exactly as strong as $(\mathcal{P})$.*

Using Conjecture 8 one might bound the integrality gap of the bidirected cut relaxation (it is currently known only to be lie between $8/7$ and 2, see [Agarwal & Charikar 04]).

Another possible application of our results would be to use $(\mathcal{P})$ to design a primal-dual Steiner approximation algorithm with better approximation ratio than 1.55, the best that is currently known [Robins & Zelikovsky 00].

The full details of the results in this poster are available in a C&O Research Report (#CORR 2007-11) of the same name. The report is also available online.

University of Waterloo