Optimal Trees

Thomas L. Magnanti and Laurence A. Wolsey

OR 290-94 April 1994

Optimal Trees *†

Thomas L. Magnanti ‡ Laurence A. Wolsey §

April, 1994

*This paper has been prepared as a chapter for the *Handbooks in Operations Research and Managament Science* series volume entitled *Networks*.

†The preparation of this paper was supported, in part, by NATO Collaborative Research Grant CRG 900281

‡Sloan School of Management and Operations Research Center, MIT

§C.O.R.E., Université Catholique de Louvain

Contents

List of Figures

1 Introduction

Trees are particular types of graphs that on the surface appear to be quite specialized, so much so that they might not seem to merit in-depth investigation. Perhaps, surprisingly, just the opposite is true. As we will see in this chapter, tree optimization problems arise in many applications, pose significant modeling and algorithmic challenges, are building blocks for constructing many complex models, and provide a concrete setting for illustrating many key ideas from the field of combinatorial optimization.

A tree[1] is a connected graph containing no cycles. A tree (or subtree) of a general undirected graph $G = (V, E)$ with a node (or vertex) set V and edge set E is a connected subgraph $T = (V', E')$ containing no cycles. We say that the tree spans the nodes V'. For convenience, we sometimes refer to a tree by its set of edges with the understanding that the tree also contains the nodes incident to these edges. We say that T is a spanning tree (of G) if T spans all the nodes V of G, that is, $V' = V$. Recall that adding an edge $\{i, j\}$ joining two nodes in a tree T creates a unique cycle with the edges already in the tree. Moreover, a graph with n nodes is a spanning tree if and only if it is connected and contains $n - 1$ edges

Trees are important for several reasons:

(i) Trees are the minimal graphs that connect any set of nodes, thereby permitting all the nodes to communicate with each other without any redundancies (that is, no extra arcs are needed to ensure connectivity). As a result, if the arcs of a network have positive costs, the minimum cost subgraph connecting all the nodes is a tree that spans all of the nodes. We refer to any such tree as a spanning tree of the network.

(ii) Many tree optimization problems are quite easy to solve; for example, efficient types of greedy, or single pass, algorithms are able to find the least cost spanning tree of a network (we define and analyze this problem in Section 2). In this setting, we are given a general network and wish to find an optimal tree within this network. In another class of models, we wish to solve an optimization problem defined on a tree, for example, find an optimal set of

[1]Throughout this chapter, we assume familiarity with the basic definitions of graphs including such concepts as paths and cycles, cuts, edges incident to a node, node degrees, and connected graphs. We also assume familiarity with the max-flow min-cut theorem of network flows and with the elements of linear programming. The final few sections require some basic concepts from integer programming.

facility locations on a tree. In this setting, dynamic programming algorithms typically are efficient methods for finding optimal solutions.

(iii) Tree optimization problems arise in a surprisingly large number of applications in such fields as computer networking, energy distribution, facility location, manufacturing, and telecommunications.

(iv) Trees provide optimal solutions to many network optimization problems.

Indeed, any network flow problem with a concave objective function always has an optimal tree solution (in a sense that we will define later). In particular, because (spanning) tree solutions correspond to basic solutions of linear programs, linear programming network problems always have (spanning) tree solutions.

(v) A tree is a core combinatorial object that embodies key structural properties that other, more general, combinatorial models share. For example, spanning trees are the maximal independent sets of one of the simplest types of matroids, and so the study of trees provides considerable insight about both the structure and solution methods for matroids (for example, the greedy algorithm for solving these problems, or linear programming representations of the problems). Because trees are the simplest type of network design model, the study of trees also provides valuable lessons concerning the analysis of more general network design problems.

(vi) Many optimization models, such as the ubiquitous traveling salesman problem, have embedded tree structure; algorithms for solving these models can often exploit the embedded tree structure.

Coverage

This paper has two broad objectives. First, it describes a number of core results concerning tree optimization problems. These results show that even though trees are rather simple combinatorial objects, their analysis raises a number of fascinating issues that require fairly deep insight to resolve. Second, because the analysis of optimal trees poses many of the same issues that arise in more general settings of combinatorial optimization and integer programming, the study of optimal trees provides an accessible and yet fertile arena for introducing many key ideas from the branch of combinatorial optimization known as polyhedral combinatorics (the study of integer polyhedra).

In addressing these issues, we will consider the following questions:

- Can we devise computationally efficient algorithms for solving tree optimization problems?
- What is the relationship between various (integer programming) formulations of tree optimization problems?
- Can we describe the underlying mathematical structure of these models, particularly the structure of the polyhedra that are defined by relaxing the integrality restrictions in their integer programming formulations?
- How can we use the study of optimal tree problems to learn about key ideas from the field of combinatorial optimization such as the design and analysis of combinatorial algorithms, the use of bounding procedures (particularly, Lagrangian relaxation) as an analytic tool, and basic approaches and proof methods from the field of polyhedral combinatorics?

We begin in Section 2 with a taxonomy of tree optimization problems together with illustrations of optimal tree applications in such fields as telecommunications, electric power distribution, vehicle routing, computer chip design, and production planning. In Section 3, we study the renowned minimum spanning tree problem. We introduce and analyze a greedy solution procedure and examine the polyhedral structure of the convex hull of incidence vectors of spanning trees. In the context of this discussion, we examine the relationship between eight different formulations of the minimum spanning tree problem that are variants of certain packing, cut, and network flow models.

In Section 4, we examine another basic tree optimization problem, finding an optimal rooted tree within a tree. After showing how to solve this problem efficiently using dynamic programming, we then use three different arguments (a network flow argument, a dynamic programming argument, and a general "optimal" inequality argument from the field of polyhedral combinatorics) to show that a particular linear programming formulation defines the convex hull of incidence vectors of rooted trees. Because the basic result in this section is fairly easy to establish, this problem provides an attractive setting for introducing these important proof techniques.

In Section 5, we consider two other tree models that can be solved efficiently by combinatorial algorithms—a degree constrained minimum spanning tree problem (with a degree constraint imposed upon a single node) and a directed version of the minimum spanning tree problem. For both problems, we describe an efficient algorithmic procedure and fully describe the underlying integer polyhedron.

In Sections 6-9 we consider more general models that are, from the perspective of computational complexity theory, difficult to solve. For each of these problems, we provide a partial description of the underlying integer polyhedron and describe one or more solution approaches.

We begin in Section 6 by studying a network version of the well-known Steiner tree problem. Actually, we consider a more general problem known as the node weighted Steiner tree problem. Generalizing our discussion of the spanning tree problem in Section 3, we examine the relationship between the polyhedron defined by five different formulations of the problem. For one model, we show that the objective value for a linear programming relaxation of the Steiner tree problem has an optimal objective value no more than twice the cost of an optimal Steiner tree. Using this result, we are able to show that a particular spanning tree heuristic always produces a solution whose cost is no more than twice the cost of an optimal Steiner tree. In this discussion, we also comment briefly on solution methods for solving the Steiner tree problem.

In Section 7, we study the problem of packing rooted trees in a given tree. This model arises in certain applications in production planning (the economic lot-sizing problem) and in facility location on a tree (for example, in locating message handling facilities in a telecommunications network). We show how to solve uncapacitated versions of this problem by dynamic programming and, in this case, we completely describe the structure of the underlying integer polyhedron. For more complex constrained problems, we show how to "paste" together the convex hull of certain subproblems to obtain the convex hull of the overall problem (this is one of the few results of this type in the field of combinatorial optimization). We also describe three different solution approaches for solving the problem—a cutting plane procedure, a column generation procedure, and a Lagrangian relaxation procedure.

In Section 8, we consider the more general problem of packing subtrees in a general graph. This problem arises in such varied problem settings as

multi-item production planning, clustering, computer networking, and vehicle routing. This class of models permits constraints that limit the number of subtrees or that limit the size (number of nodes) of any subtree. Our discussion focuses on extending the algorithms we have considered previously in Section 7 when we considered optimal subtrees of a tree.

In Section 9, we briefly introduce one final set of models, hierarchical tree problems that contain two types of edges—those with high reliability versus those with low reliability (or high capacity versus low capacity). In these instances, we need to connect certain "primary" nodes with the highly reliable (or high capacity) edges. We describe an integer programming formulation of this problem that combines formulations of the minimum spanning tree and Steiner tree problems as well as a heuristic algorithm; we also give a bound on how far both the heuristic solution and the optimal objective value of the linear programming relaxation can be from optimality.

Section 10 is a brief summary of the chapter and Section 11 contains notes and references for each section.

Notation

Frequently in our discussion, we want to consider a subset of the edges in a graph $G = (V, E)$. We use the following notation. If S and T are any two subsets of nodes, not necessarily distinct, we let $E(S,T) = \{e = \{i,j\} \in E : i \in S \text{ and } j \in T\}$ denote the set of edges with one end node in S and the other end node in T. We let $E(S) \equiv E(S,S)$ denote the set of edges whose end nodes are both in S. $\bar{S} = V \setminus S$ denotes the complement of S and let $\delta(S)$ denote the cutset determined by S, that is, $\delta(S) = E(S,\bar{S})$ $=\{e = \{i,j\} \in E : i \in S \text{ and } j \in \bar{S}\}$. For any graph G, we let $V(G)$ denote its set of nodes and for any set of edges $\bar{E}$ of any graph, we let $V(\bar{E})$ denote the set of nodes that are incident to one of the edges in $\bar{E}$.

At times, we consider directed graphs, or digraphs, $D = (V, A)$ containing a set A of directed arcs. In these situations, we let $\delta^+(S) = \{e = (i,j) \in A : i \in S \text{ and } j \in \bar{S}\}$ denote the cutset directed *out of* the node set S and let $\delta^-(S) = \{e = (i,j) \in A : i \in \bar{S} \text{ and } j \in S\}$ denote the cutset directed *into* the node set S. We also let $A(S) = \{e = (i,j) \in E : i \in S \text{ and } j \in S\}$ and define $V(D)$ and $V(\bar{A})$ for any set $\bar{A}$ of arcs, respectively, as the nodes in the digraph D and the nodes in the digraph that are incident to one of the arcs in $\bar{A}$.

As shorthand notation, for any node v, we let $\delta(v) = \delta(\{v\})$, $\delta^+(v) = \delta^+(\{v\})$, and $\delta^-(v) = \delta^-(\{v\})$.

We also let **1** denote a vector of ones, whose dimension will be clear from context, let R^m denote the space of m-dimensional real numbers, and let Z^m denote the space of m-dimensional integer vectors.

The set notation $A \subset B$ denotes $A \subseteq B$ and $A \neq B$.

For any set S, we let $conv(S)$ denote the convex hull of S, that is the set of points $x = \{\sum_{j=1}^{k} \lambda_j s^j : \sum_{j=1}^{k} \lambda_j = 1$ and $\lambda_j \geq 0$ for $j = 1, \ldots, k$, and some points $s^j \in S\}$ obtained as weighted combinations of points in S.

Recall that a polyhedron in R^n is the set of solutions of a finite number of linear inequalities (and equalites). If a polyhedron is bounded, then it also is the convex hull of its extreme points. If each extreme point is an integer vector, we say that the polyhedron is an *integer polyhedron.*

Let A and B be two given matrices and b be a column vector, all with the same number of rows. Frequently, we will consider systems of inequalites $Ax + Dy \leq b$ defined by two sets x and y of variables. We refer to the set of points $\{x : Ax + Dy \leq b$ for some vector $y\}$ as the set of x-feasible solutions to this system. Note that $Q = \{x : Ax + Dy \leq b$ for some vector $y\}$ is the projection of the polyhedron $P = \{(x, y) : Ax + Dy \leq b\}$ onto the space of x-variables. As is well known, Q itself is a polyhedron, that is, can be expressed as the set of solutions of a finite number of inequalities involving only the x-variables.

2 Tree Optimization Problems

Tree optimization problems arise in many forms. In this chapter, we consider two generic problem types:

(a) *Optimal trees.* Given a graph $G = (V, E)$ with node set V and edge set E and with a weight w_e defined on each edge $e \in E$, find a tree T in G that optimizes (maximizes or minimizes) the total weight of the edges in T. The tree might also have a designated root node r and have various constraints imposed on the root node or on the subtrees created if we eliminate the root node and its incident edges.

(b) *Optimal subtrees of a tree (packing subtrees in a tree).* Given a tree T, suppose we wish to find a set of node disjoint subtrees of T, each with a designated root node. Each node v has a weight w_v^r that depends upon the root node r of the subtree that contains it, and we wish to optimize (maximize or minimize) the total weight of the nodes in the subtrees. Note that this model permits edge weights as well as node weights since once we have selected a root node for each subtree, each edge of the tree has a uniquely associated node, namely the first node on the path connecting that edge to the root. Therefore, by associating the edge weight with this node, we can formulate the tree packing problem with edge weights as an equivalent model with weights defined only on the nodes.

Figures 1 and 2 give a schematic representation of both of these problem types. We might also consider another problem type: packing subtrees in a general network. In principle, we might view this problem as a composite of the other two: first, we find a tree in the network and then we pack subtrees in this tree.

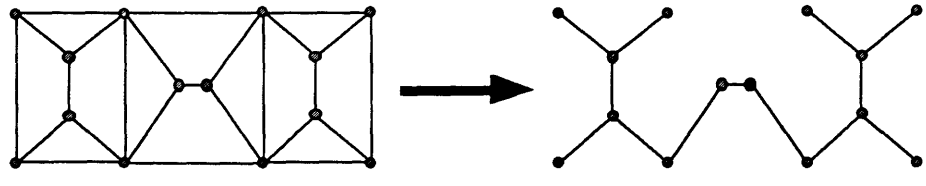


Figure 1: Optimal Tree in a Graph

Both the optimal tree problem and optimal subtrees in a tree problem arise in several different forms, depending upon the restrictions we impose

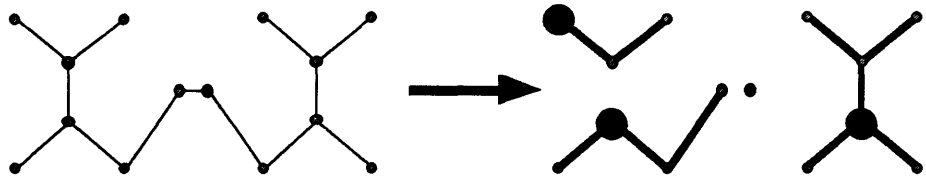


Figure 2: Optimal Rooted Subtrees of a Tree

upon the set of (feasible) trees/subtrees we wish to consider. The following constraints arise in several practical problem settings (the root node in these constraints is either a designated single root in the optimal tree problem, or the root node of any subtree in the packing subtree problem):

- A size constraint imposed upon the number of nodes in any (sub)tree.
- Topological constraints imposed on any (sub)tree [e.g., maximum or minimum node degrees, restrictions that certain specific nodes need to be included in a (sub)tree]. In particular, we might impose a degree constraint on the root node.
- A size constraint imposed upon the subtrees formed by removing the root node of any (sub)tree. More generally, each node might have an associated weight, and we might wish to restrict the total node weight of any subtree.
- Bounds (maximum and/or minimum) imposed upon the number of subtrees in the packing subtrees of a tree problem.
- A flow requirement imposed upon the (sub)trees, together with capacity constraints imposed upon the total throughput of any edge or node; and/or
- The availability of multiple types of edges and root nodes, with restrictions imposed upon the types of facilities (edges and nodes) used. For example, each type of edge or node might have an associated cost or capacity and we might seek a minimum cost solution that loads the network with certain required capacity. Or, we might impose multi-layered requirements, for example, certain primary nodes be connected

by high capacity (or reliability) edges and secondary nodes by any type of facility (high or low capacity).

These constraints arise for a variety of reasons. In several applications, root nodes represent service facilities, for example, plants or warehouses in a production system, hospitals in an urban network, centralized computers in a computer network, or multiplexers in a communication network. A size constraint on each subtree might model capacity limitations on the service facility and a cardinality constraint on the number of subtrees might model limited availability of service facilities or a managerial or engineering decision to limit the number of facilities. In some applications, the edges adjacent to the root node represent a physical entity such as a loading dock in a warehouse or a communication port in a centralized computer. In these settings, a degree constraint on the root node might represent a physical limit on the availability of these facilities. In addition, nodes in the subtrees formed by removing the edges adjacent to the root node might represent the customers served by each entity (port) at the root node. For reliability reasons, or to model capacities, we might wish to limit the size (number of nodes) in each subtree.

We could, of course, add even further wrinkles on these various problem types. For example, rather than viewing just the root nodes as "special" and imposing various restrictions on them or on the subtrees formed by removing them from the solution, we could consider layered problems with root nodes, first-level nodes (those adjacent to the root nodes), second-level nodes, and so forth, and impose restrictions on the nodes at each level. Models with constraints imposed upon the root node appear to capture many of the issues encountered in practice and studied by researchers in the past, so we focus on these versions of the problems.

Certain special versions of these problems are either standard topics in the literature or arise frequently as subproblems in more general combinatorial optimization applications.

- *Minimum spanning tree problem.* In this basic "tree in a network model", we wish to find a tree that spans (contains) all the nodes of a graph G and that minimizes the overall weight of the edges in the tree. Note that in this case, we impose no topological or capacity restrictions on the tree.

- *Rooted subtree problem.* Given a tree and a root node r, we wish to find a subtree rooted at (containing) this node and that minimizes the overall weight of the nodes (and/or arcs) in the subtree. This problem is a core model for the class of subtrees of a tree problems, much like the minimum spanning tree is a core model for the class of optimal tree problems.

- *Steiner tree problem.* Let $G = (V, E)$ be a given graph with a weight (cost) defined on each edge $e \in E$. Given a set T of terminal nodes that need to be connected to each other, we wish to find a tree of G that contains these nodes and whose total edge weight is as small as possible. Note that the optimal tree might contain nodes, called Steiner nodes, other than the terminal nodes T.

- *K-median problem.* Find K or fewer node disjoint subtrees of a network, each with a root, that minimizes the total edge weight of the edges in the subtrees. The K-median problem on a tree is the version of this problem defined on a tree.

- *C-capacitated tree problem.* In this version of the rooted subtrees of a network problem, each subtree is limited to containing at most C nodes. The C-capacitated problem on a tree is the version of this problem defined on a tree.

In order to make this problem taxonomy more concrete, we next briefly consider a few important application contexts.

Designing Telecommunications and Electric Power Networks. Suppose that we wish to design a network that connects customers in a telecommunications or electrical power network. The links are very expensive (in part, because we might need to dig trenches to house them) and the routing costs are negligible: once we have installed the line facilities (edges), the routing cost is very small. Therefore, we want to connect the customers using the least expensive tree. If all the nodes of the network are customers, the problem is the classical minimum spanning tree problem. If we need to connect only some of the nodes, and can use other nodes as intermediate nodes, the problem become the classical Steiner tree problem. In a multi-layered version of this class of problems, we wish to connect certain "key" users using

highly reliable communication lines or high voltage transmission lines. We can use less reliable lines or low voltage lines at a lower cost to connect the other users of the system.

Similar types of applications arise in other settings as well. For example, in constructing a highway infrastructure in a developing country, our first objective might be to solve a minimum spanning tree problem or Steiner tree problem to connect all the major cities as cheaply as possible. Or, we might wish to solve a multi-layered problem, ensuring that we connect all the major cities by highways and all the cities, whether major or not, through the use of any combination of highways or secondary access roads.

Facility Location. In a distribution system of geographically dispersed customers on a network, we wish to establish warehouses (distribution centers) to fulfill customer orders. Suppose that for administrative reasons (simplicity of paperwork or ease in monitoring and controlling the system), we wish to service each customer from a single warehouse. Moreover, suppose we wish to satisfy a contiguity property: if a warehouse at location r services customer i along a path that passes though the location of customer j then warehouse i must also service customer j. Then each feasible solution is set of subtrees, each with a root node which is the location of the warehouse serving the customers in that subtree.

This basic facility location problem arises in many alternate forms and in many guises. For example, we might impose capacities (for example, a limit on the number of customers served) on each service facility or we might restrict the total number of service facilities. These versions of the problem would be C-capacity subtree and K-median problems.

Figure 3 illustrates another application context that arises in telecommunications. Most current telecommunications systems use a tree (typically of copper cable) to connect subscribers to physical devices called local switches that route calls between the subscribers. Each subscriber is connected to a switching center in the "local access" tree by a dedicated circuit (telephone line). Each edge of the tree has a capacity (the number of physical telephone lines installed on that edge) and as the demand for services increases, the network might have insufficient capacity. In this example the nodes 3, 6, and 7 require a total of 300 circuits and the edge $\{1, 3\}$ between these nodes and the switch has a capacity of only 200 circuits. Two other edges in the tree

have insufficient capacity: (a) nodes 5, 8, and 9 require 250 circuits, and the edge {2, 5} has a capacity of only 200 circuits; and (b) nodes 2, 4, 5, 8, and 9 require 400 circuits, and the edge {1, 2} has a capacity of only 140 circuits

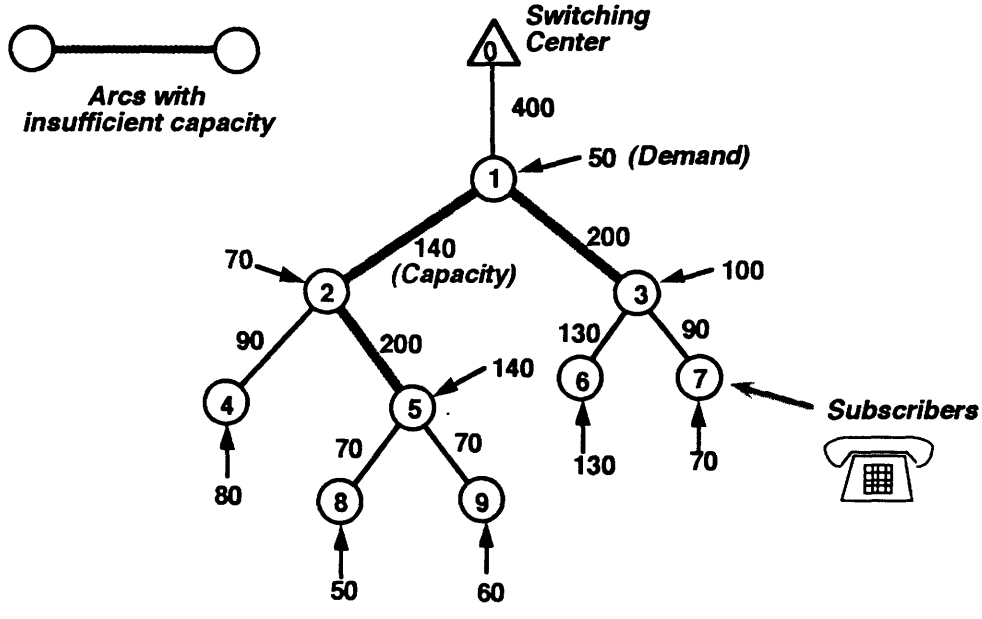
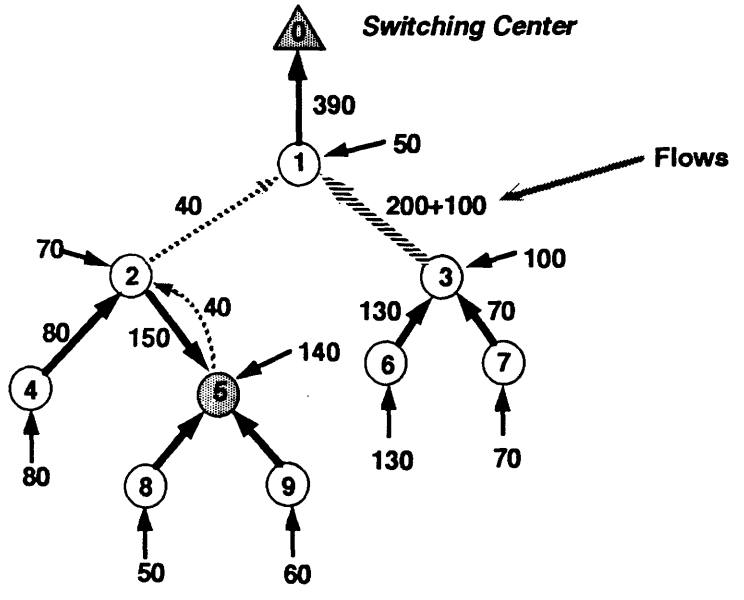


Figure 3: Local Access Telcommunication Network

One way to meet the excess demand is to add additional capacity (copper cable) on edges with insufficient capacity. Another option is to add sophisticated equipment known as concentrators (or alternative equipment known as multiplexers, or remote switches) that compress messages so that they require less circuit capacity (that is, so that calls can share lines). Figure 4b shows one possible solution for fulfilling the demand of all the nodes in Figure 3. In this case, we have added 100 extra lines on edge $\{1, 3\}$ and added a concentrator at node 5 that serves the subscribers of nodes 2, 4, 5, 8, and

9. This concentrator uses a compression ratio of 10 to 1 so the 400 circuits assigned to it use only 40 circuits on the downstream path 5-2-1-0 connecting node 5 to the switching center. Consequently, this path has sufficient capacity for all the subscribers that use it.



Expansion plan

- **Install 10 to 1 compression concentrator at node 5 with capacity 400 circuits. Nodes 2, 4, 5, 8, and 9 home on this concentrator.**
- **Expand cable capacity between nodes 1 and 3 by 100 circuits.**

Figure 4: Local Access Expansion Strategy as Subtree Packing

Note two properties of the solution shown in Figure 4. First, the solution assigns all of the demand at each node either directly to the switching center (the nodes 1, 3, 6, and 7) or to the concentrator at node 5 (the nodes 2, 4, 5, 8, and 9). In addition, the solution satisfies a contiguity property: if the solution assigns node u to the switching center (or to the concentrator) and node v lies on the path connecting node u to the switching center (concentrator) then it also assigns node v to the switching center (concentrator). These two

assumptions imply that the solution decomposes the tree into a set of rooted subtrees; one contains the switching center and each other one contains a single concentrator. Therefore, the problem is an application of packing rooted subtrees in a tree.

Routing Problems. Figure 5 shows a solution to an illustrative vehicle routing problem. In this application context, we are given a fleet of vehicles domiciled at a depot, node 0, and wish to find a set of tours (cycles that are node disjoint except at the depot) that contain all the customer nodes $1, 2, \ldots, n$. We incur a cost for traversing any edge and wish to find the minimum cost routing plan. Note that if we are given any set of tours and eliminate all of the edges incident to the depot, the resulting graphical structure is a set of node disjoint paths that contain all the nodes $1, 2, \ldots, n$. Therefore, the solution is a very special type of tree packing problem, one in which each tree must be a path. Since we wish to include every node in the solution, we might refer to this problem as a path partitioning problem since we are partitioning the non-depot nodes into paths.

In the simplest version of this problem, the customers are identical and each vehicle has sufficient capacity to serve all of the customers, so any path partitioning of the nodes $1, 2, \ldots, n$ will be feasible. If we impose additional restrictions on the solution, then the problem becomes a special version (because the trees must be paths) of one of the alternative tree problems we have introduced previously.

For example, if we have K available vehicles, the problem becomes a K-median version of the path partitioning problem since we wish to use at most K paths to cover the nodes $1, 2, \ldots, n$. In particular, if $K = 1$, the vehicle routing problem becomes the renowned traveling salesman problem; in this case, any feasible solution to the associated path partitioning problem is a Hamiltonian path (that is, a single path containing all the nodes).

If the customers are identical, that is, have the same demands, which by scaling we can assume are all one unit, and each vehicle has a capacity of C units, then each tour, and so each path in the path partitioning problem, can contain at most C customer points. Therefore, the problem becomes a C-capacitated subtree version of the path partitioning problem.

In practice, applications often have other important problem features; for example, (a) the demands typically vary across the customers, or (b) each

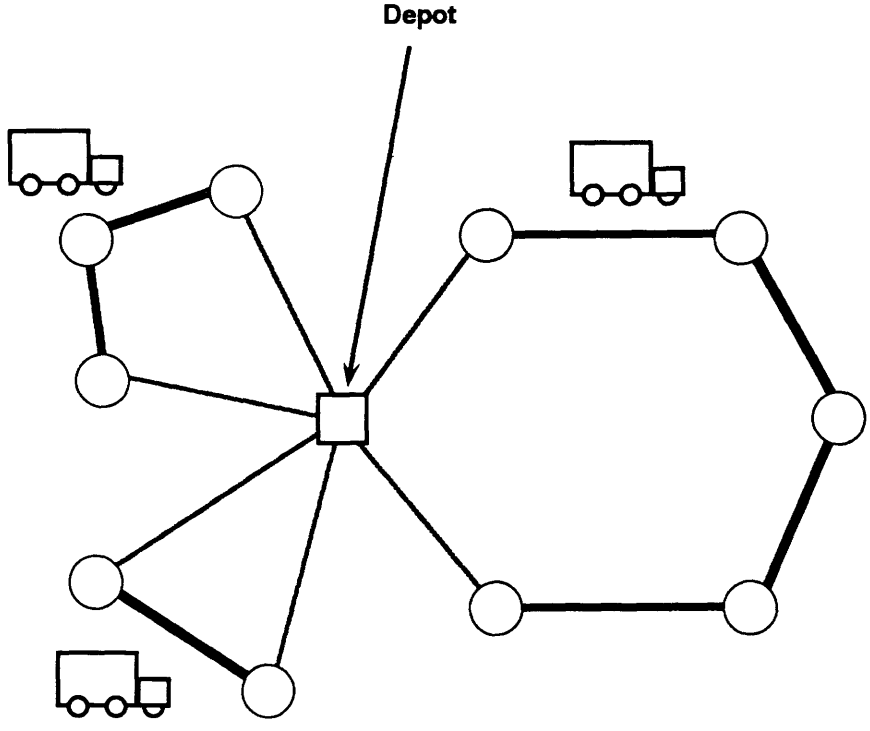


Figure 5: Vehicle Routing as Packing Paths in a Network

edge might have an associated travel time and we might have a limit on the overall travel time of every route (and so each path in the associated path partitioning problem) or we might have specified time windows on the delivery (pick-up) time of each customer. In these instances, the Hamiltonian paths in any feasible solution will have other constraints imposed upon them, for example, restrictions on the total length (travel time) of each path.

We might note that the "vehicle routing" problem, and so its associated tree problems, arise in other application contexts—for example, machine scheduling. In this setting, we associate each vehicle with a machine and the customers are jobs that we wish to perform on the machines. Each "vehicle tour" becomes a sequence of jobs that each machine will process. That is, the machine "visits" the jobs. When we have K identical machines, the problem becomes a K-median version of the problem. When we impose capacities on the machines and processing times of the jobs (which correspond to demands), we obtain other versions of the vehicle routing problem and, therefore, of tree packing problems.

Clustering. In cluster analysis, we wish to partition (cluster) a set of data so that data in the same partition (cluster) are "similar" to each other. Suppose that we can represent the data in an n-dimensional space (the dimensions might, for example, represent different symptoms in a medical diagnosis). Suppose we view two points as close to each other if they are close to each other in Euclidean distance, and measure similarity of a set of points as the length of the (Euclidean) minimum spanning tree that connects these points. Then if we want to find the best k clusters of the points, we need to find the best set of k disjoint trees that contain all the points; we connect the points in any one of these k sets by a minimum spanning tree defined on these points. In practice, we might solve this problem for all values of k and then use some secondary criteria (e.g., human judgment) to choose the best value of k.

VLSI Design. The designers of very large scale integrated (VLSI) chips often wish to connect a set of modules on the surface of a chip using the least total length of wire. The physical layout of the chip usually requires that wires can be routed only along "channels" that are aligned along north/south or east/west directions of the surface. Thus the wiring distance metric be-

tween a pair of modules is the rectilinear or manhattan metric. If we represent each module location as a point (although physically the modules occupy a nonzero area), this problem is a Steiner tree problem with the set of module location points as the terminal nodes T; in theory, the Steiner nodes could be anywhere on the chip's surface (see Figure 6a). Since we are measuring distances between nodes according to the rectilinear norm, researchers often refer to this type of problem as the rectilinear Steiner tree problem.

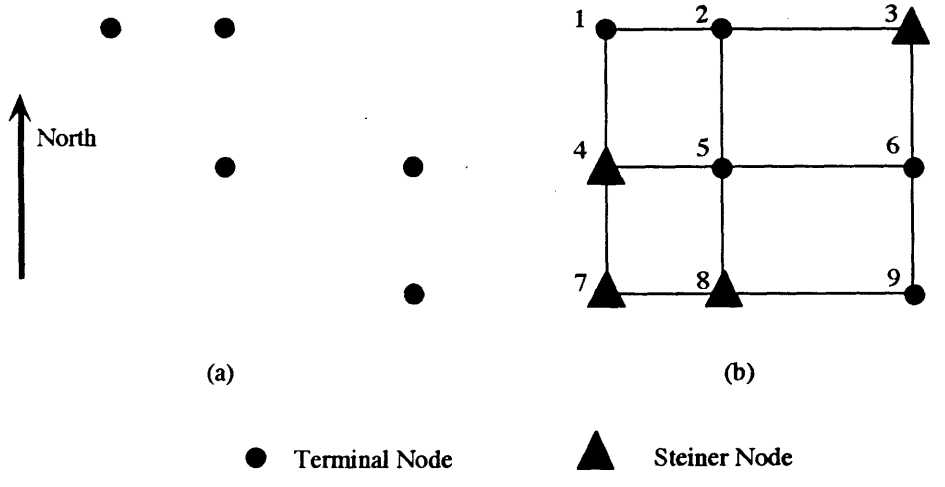


Figure 6: (a) Set of Points Representing a Rectilinear Steiner Tree Problem; (b) Grid Graph Representation of Rectilinear Steiner Tree Example

One popular model for this application models the surface of the chip as a grid graph (see Figure 6b) with each node chosen as either the location of a module or the intersection point of some north/south and east/west line that passes through one of the module locations. Wires can run only along the edges of the grid graph. In Figure 6.b, nodes 1, 2, 5, 6 and 9 are the terminal nodes and nodes 3, 4, 7 and 8 are Steiner nodes arising from lines passing through these nodes. This derived grid graph model is a special case of the classical Steiner tree problem.

In practice, the design of a chip usually involves many different sets of modules, each set needing to be connected together. When multiple sets of modules use any channel in their Steiner tree solution, multiple wires will use the same edge of the underlying grid graph. In this application setting, each channel on the chip surface can accommodate only a limited number of wires;

so this more general problem is a variant of the Steiner tree problem with several rectilinear Steiner tree problems defined on the same grid graph, but with a limit on the number of Steiner trees that use any edge of the graph. That is, the problem becomes a problem of "packing" Steiner trees into the capacitated edges.

In this discussion, we have considered a rectilinear Steiner tree model for connecting modules of a computer chip; the same type of model arises in the design of printed circuit boards.

Production Planning. Suppose we wish to find a production and inventory plan that will meet the demand $d_t > 0$ of product over each of T time periods $t = 1, 2, \ldots, T$. If we produce x_t units of the product in period t, we incur a fixed (set-up) plus variable cost: that is, the cost is $f_t + c_t x_t$. Moreover, if we carry s_t units of inventory (stock) from period t to period $t+1$, we incur an inventory cost of $h_t s_t$. We wish to find the production and inventory plan that minimizes the total production and inventory costs. We refer to this problem as the single item uncapacitated economic lot-sizing problem. We can view this problem as defined on the network shown in Figure 7. This network contains one node for each demand period and one node that is the source for all production.

On the surface, this problem might not appear to be a tree optimization model. As shown by the following result, however, the problem has a directed spanning tree solution, that is, it always has at least one optimal production plan whose set of flow carrying arcs (that is, those corresponding to $x_t > 0$ and $s_t > 0$) is a spanning tree with exactly one arc directed into each demand node.

Theorem 2.1 *The single item uncapacitated economic lot-sizing problem always has a directed spanning tree solution.*

Proof. First note that since the demand d_t in each period is positive, at least one of x_t and s_{t-1} is positive in any feasible solution. Consider any given feasible solution to the problem and suppose that it is not a directed spanning tree solution. We will show we can construct a directed spanning tree solution with a cost as small as the cost of the given solution. Suppose $x_t > 0, s_{t-1} > 0$ and x_τ is the last period prior to period t with $x_\tau > 0$. Let $\epsilon = min\{x_\tau, s_{t-1}\}$. Note that if $x_\tau < s_{t-1}$, then $s_{\tau-1} > 0$.

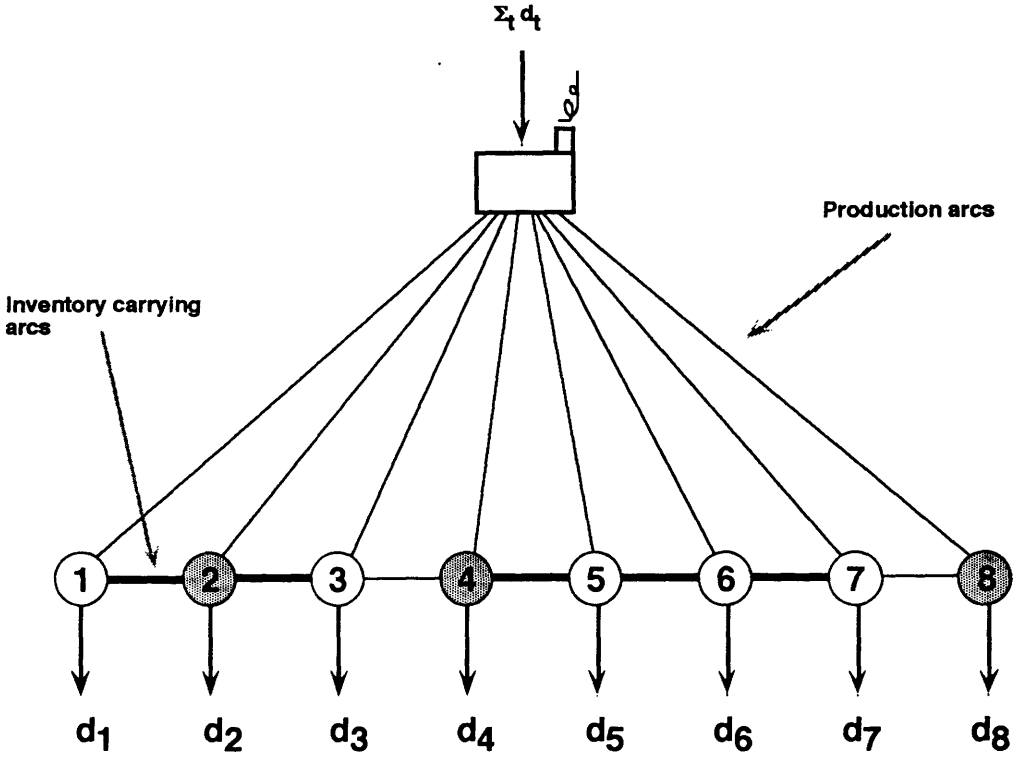


Figure 7: Production Lot-sizing as Packing (Rooted) Subtrees in Trees

Consider the two costs c_t and $c_{\tau t} = c_\tau + h_\tau + h_{\tau+1} + \ldots + h_{t-1}$. If $c_t \leq c_{\tau t}$, we set $x_t \leftarrow x_t + \epsilon, x_\tau \leftarrow x_\tau - \epsilon$ and $s_j \leftarrow s_j - \epsilon$ for all $j = \tau, \ldots, t-1$; if $c_t > c_{\tau t}$, we set $x_t \leftarrow 0, x_\tau \leftarrow x_\tau + x_t$ and $s_j \leftarrow s_j + x_t$ for all $j = \tau, \ldots, t-1$. In both cases, we obtain a solution with at least as small a cost as the given solution and with one less period with $x_q > 0$ and $s_{q-1} > 0$. (If $\epsilon = x_\tau \leq s_{t-1}$ and $c_t \leq c_{\tau t}$, then $q = \tau$; otherwise, $q = t$.) By repeating this process as many times as necessary, we eventually obtain a directed spanning tree solution with a cost as small as that of the given solution. ∎

Note that for a directed spanning tree production plan, we never simultaneously produce in any period and carry inventory into that period. Therefore, whenever we produce, we must produce for an integral number of periods (from the current period until just before the next production period). This property permits us to use a simple dynamic programming algorithm to solve the problem very efficiently. Let v_t be the value of a minimum cost solution of the problem for periods $1, 2, \ldots, t$ assuming $s_t = 0$. Set $v_0 = 0$. Then

$$v_t = \min_{1 \leq \tau \leq t} \{v_{\tau-1} + f_\tau + \sum_{\tau \leq j \leq t} c_{\tau j} d_j\}$$

since in any optimal directed spanning tree solution, we must produce for the last time in some period τ, carry zero inventory into period τ, and incur inventory carrying costs for all periods $\tau, \tau+1, \ldots, t-1$.

Finally, we might note that we can view this problem as a subtree optimization problem on the line graph containing only the demand nodes $1, 2, \ldots, T$ (see Figure 7). Since whenever we produce, we always produce for an integral number of consecutive periods, the problem always has an optimal solution that decomposes the line graph into a collection of interval subgraphs each containing a consecutive set of nodes: if a subgraph has nodes $t, t+1, \ldots, q$, its root is node t and the weight of any node j in the subgraph is given by: $w_t^t = f_t + c_t d_t$, and $w_j^t = c_{tj} d_t$, for $j \neq t$. In this model, we choose the root node of any interval as the leftmost node of the interval since we do not permit backlogging (that is each $s_t \geq 0$). If we permit backlogging, then we could choose any node τ in the interval as its root (that is, as the production point) for that interval and the weight w_j^τ for any node j to the left of the root (that is, $j < \tau$) would be the cost of supplying the demand d_j by backlogged production from time t.

Trees and Network Flow Problems

As illustrated by the production planning example we have just considered, trees can arise as solutions to optimization problems that on the surface are unrelated to trees; that is, the problem is not defined on a tree nor does it explicitly seek a tree solution. This example is a special case of a more general result in the field of linear programming. Consider any optimization problem of the form $\min\{cx : Nx = b, 0 \leq x \leq u\}$ and suppose that each column of the n by m matrix N has at most two nonzero entries, which have the values ± 1; moreover, if a column has two nonzero entries, one is a $+1$ and the other a -1. Define a directed graph G with $n + 1$ nodes, numbered 0 to n, and with m directed edges: G contains one node for each row of N, plus one additional node (node 0), and one edge for each column of N. We define the graph as follows: if a column of N has a $+1$ entry in row i and a minus one entry in row j, the graph G contains arc (i, j). If a column has a single nonzero entry and it is $+1$ in row i, the graph contains the arc $(i, 0)$, and if its single nonzero entry is -1 in row j, the graph has the arc $(0, j)$. We can then interpret the variables x as flows on the edges of this graph; the jth constraint of $Nx = b$ is a mass balance constraint stating that the total flow out of node j minus the total flow into that node equals the supply b_j at that node. We wish to find a flow that meets these mass balance restrictions, the bounding constraints $0 \leq x \leq u$, and that has the smallest possible flow cost cx.

From the theory of linear programming, we know that this linear program always has an optimal solution corresponding to a basis B of N. That is, the solution has the property that we can set each variable x_e not in B to either value 0 or u_e, and then solve for the basic variables from the system $Nx = b$. But now we observe that any basis corresponds to a spanning tree of the graph G if we ignore the orientation of the arcs. Recall that each column of B corresponds to an arc in G. Let $A(B)$ denote the set of arcs corresponding to the columns of B. If the graph defined by the edges $A(B)$ contains an undirected cycle C, then as we traverse this cycle we encounter any arc $e = (i, j)$ either first at node i or first at node j. Let $y_e = +1$ in the former case, $y_e = -1$ in the latter case, and $y_e = 0$ if arc e does not belong to the cycle C. Then $By = 0$ and so B is not a basis matrix of N. Therefore, the subgraph T of G corresponding to any basis matrix cannot contain any

cycles. Therefore, it either is a tree or a collection of disjoint subtrees.

The fact that network flow problems always have tree solutions, in the sense we have just discussed, has far reaching implications. It permits us to solve these problems very effectively by searching efficiently among tree solutions (the simplex method has this interpretation) using the underlying graph to implement the algorithms using graphical methods in place of more complex matrix operations. Although we will not discuss these methods in this chapter, we note that the fact that network flow problems have spanning tree solutions, and are solvable efficently using tree manipulation methods, is one of the primary reasons why tree optimization problems are so important in both theory and practice.

The optimal tree property of network flows also has polyhedral implications. It implies that the extreme points of the system $\{x : Nx = b, 0 \leq x \leq u\}$ are integer whenever the vectors b and u are integer. In this case, it is easy to show that the incidence vector of any basic solution to the linear program is integer since (i) the flows on all arcs e not corresponding to the basis are set to value 0 or u_e, which are integer; (ii) setting the values x_e of any nonbasic arc $e = (i, j)$ to value $\overline{x}_e = 0$ or $\overline{x}_e = u_e$ has the effect of updating the vector b by adding the integer $\overline{x}_e$ to b_j and subtracting $\overline{x}_e$ from b_i, so the updated value $\overline{b}$ of the vector b, once we have made these assignment of variables, remains integer; and (iii) solving for the values of the arcs in a tree for any integer vector $\overline{b}$ gives integer values for the following reason. Note that the tree always has at least one node v (actually at least two nodes) with a single arc e in the tree incident to it (that is, a degree one node in the tree). Therefore, the value of x_e is $\pm\overline{b}_v$. If we set x_e to this value, we update the b vector by adding and subtracting the value $\pm\overline{b}_v$ from each of the two components of b corresponding to the nodes v and q that are incident to arc e. If we now eliminate node v and arc e from the tree, we obtain a new tree with one fewer node. The new tree will again have at least one degree one node so we can find an integer value for one other component of the vector x. By repeating this process, we determine integer values for all the basic (tree) variables.

This discussion shows that every basic solution is integer valued, and the theory of linear programming implies that every extreme point to the system $\{x : Nx = b, 0 \leq x \leq u\}$ is integer valued.

3 Minimum Spanning Trees

In this section we study the minimum spanning tree problem. We begin by describing a fundamental solution method, known as the greedy algorithm, for solving this problem. We establish the validity of this algorithm in two ways: (i) using a direct combinatorial argument, and (ii) using a mathematical programming lower bounding argument based upon relaxing part of the problem constraints. Both of these arguments are representative of methods used frequently in the field of combinatorial optimization. In order to highlight the interaction between algorithms and theory, we then use this algorithm to give a constructive proof of a polyhedral representation of the spanning tree polytope; namely, we show that the extreme points of a basic "packing" or "subtour breaking" linear programming formulation of the problem are the incidence vectors of spanning trees.

We then introduce and study variants of two other "natural" formulations of the minimum spanning tree problem: a cutset model and a flow model. We show how to improve the formulation of both of these models, using the notion of multicuts in the cutset formulation and using multicommodity and directed versions of the flow formulation. These modeling variations produce a hierarchy of models for the minimum spanning tree problem. When formulated as integer or mixed integer programs, all these models are equivalent; some of them give better (more accurate) linear programming relaxations of the problem and in our discussion, we show the relationship between these relaxations and the linear programming relaxation of the basic packing formulation.

3.1 The Greedy Algorithm

The greedy algorithm is a simple one-pass procedure for solving the minimum spanning tree problem: the algorithm orders the edges in a given graph $G = (V, E)$ from smallest to largest weight (breaking ties arbitrarily) and considers the edges in this order one at a time, at each stage either accepting or rejecting an edge as a member of the tree it is constructing. The decision rule for each edge is very simple: if the edge forms a cycle with those already chosen, the method discards it from further consideration; otherwise,

the method adds it to the tree it is forming. To illustrate this algorithm, consider the example shown in Figure 8a, with the edges ordered from smallest to largest weight as $a, b, c, d, e, f, i, j, g, h$. The method accepts the edges a, b, c, d, e, and f since they do not form any cycles. Edge i forms a cycle with edges a, b and e and edge j forms a cycle with edges c, d, and f, so the algorithm rejects those arcs. It then accepts edge g and rejects edge h since it forms a cycle with the edges e, f, and g. Figure 8b shows the tree that the algorithm produces.

Does the greedy algorithm solve the minimum spanning tree problem? If so, how might we prove this result? We will answer these questions by considering two different proof techniques, one combinatorial and one based upon a mathematical programming bounding argument. In the process, we show that the greedy algorithm actually solves a more general "componentwise" optimization tree problem and show the relationship between the greedy algorithm and the proof that a polyhedron is an integer polyhedron.

Combinatorial Argument

In our example, the greedy algorithm chooses the edges of a greedy tree T_{greedy} in the order a, b, c, d, e, f, g. Suppose that T, with edges a, b, g, d, e, f, h, is any spanning tree. We will show that we can find a sequence of trees $T = T_0, T_1, T_2,$
$\ldots, T_k = T_{greedy}$ satisfying the property that for $j = 1, 2, \ldots, k-1$, each tree T_{j+1} has a weight at least as small as its predecessor T_j. Therefore, the weight of T_{greedy} is as small as the weight of T and since T is an arbitrary spanning tree, T_{greedy} is a minimum spanning tree.

We first note that if the tree T does not contain the edge a, adding a to the tree T creates a unique cycle and removing any edge from this cycle creates another tree T_1. Since the greedy algorithm chooses edge a as a minimum weight edge in the graph, the weight of the tree T_1 is at least as small as the weight of the tree T. So for any tree T, we can find a tree T_1 whose weight is at least as small that of T and that contains the edge a. Now suppose that after several steps of adding edges from T_{greedy}, one at a time, to the trees $T_0, T_1, \ldots$ we have obtained a tree T_4, say, whose weight is no more than T and that contains the first five edges a, b, c, d, e of the greedy tree. Adding edge f to this tree creates a cycle. The steps of the greedy algorithm imply that this cycle must contain at least one edge q other than

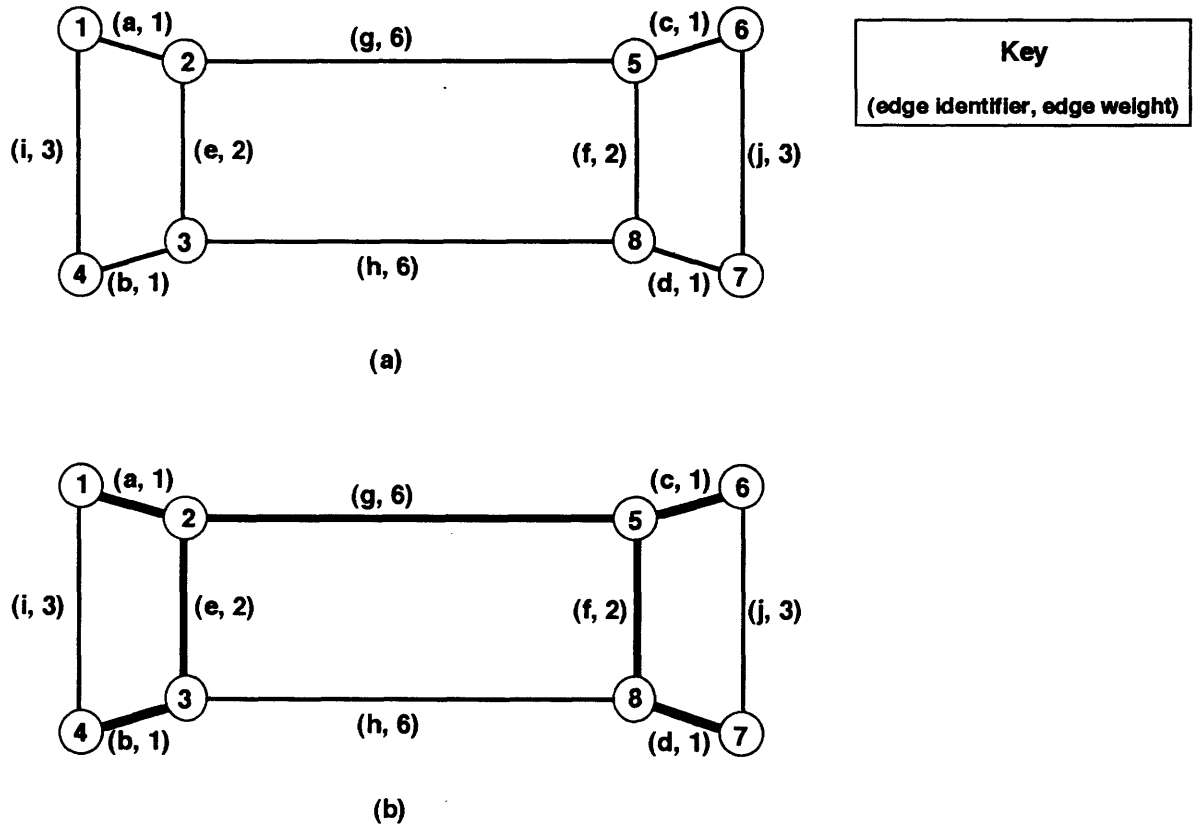


Figure 8: Minimum Spanning Tree Example: (a) Data; (b) An Optimal Tree (Bold Edges)

the edges a, b, c, d, e. Moreover, the greedy algorithm also implies that the weight of the edge f is as small as the weight of the edge q (otherwise the algorithm would have added q before adding the edge f). Therefore, we can replace edge q in T_4 by the edge f, creating a new tree T_5 with the edges a, b, c, d, e, f whose weight is as small as T_4.

Continuing in this way, we eventually add each of the edges of the greedy tree and the resulting tree $T_k = T_{greedy}$ has a weight as small as T. Therefore, we have proved that the weight of the greedy tree is as small as the weight of any other tree T, so T_{greedy} is a minimum spanning tree.

We might note that in this argument we have actually proved a stronger result. Let T_1 and T_2 be any two trees of a graph G and suppose that we order the weights of the edges in each tree from smallest to largest. That is, $w_a \leq w_b \leq \ldots \leq w_g$ are the weights of the edges a, b, c, d, e, f, g in T_1 and $w_\alpha \leq w_\beta \leq \ldots \leq w_\eta$ are the weights of the edges $\alpha, \beta, \gamma, \delta, \epsilon, \nu, \eta$ in T_2. We say that T_1 is componentwise as small as T_2 if $w_a \leq w_\alpha, w_b \leq w_\beta, \ldots., w_g \leq w_\eta$. Note that if T_1 is componentwise as small as T_2, then the total weight of T_1 is at least as small as the total weight of T_2. Also note the transitivity property: if T_1 is componentwise as small as T_2 and T_2 is componentwise as small as T_3, then T_1 is componentwise as small as T_3. We refer to a tree as componentwise minimum if it is componentwise as small as any other tree. We might note that there is no a priori guarantee that a componentwise minimum spanning tree exists. Indeed, most classes of combinatorial optimization problems do not contain componentwise optimal solutions.

Consider two subsequent trees T_q and T_{q+1} in the argument we have just given for showing that the greedy algorithm produces a minimum spanning tree. We obtained T_{q+1} by replacing one edge of T_q by an edge with a weight at least as small. Therefore, T_{q+1} is componentwise as small as T_q. But if each tree T_{q+1} in the sequence is componentwise smaller than its predecessor T_q, then the final tree T_{greedy} is componentwise as small as the tree T. Since T was an arbitrary tree, we have established the following property.

Theorem 3.1 *The greedy algorithm produces a componentwise minimum spanning tree.*

We next give an alternative proof that the greedy algorithm generates a minimum spanning tree, and in doing so illustrate ideas from the field of mathematical programming that have been proven to have wide applicability in combinatorial optimization.

Lower Bounding Argument

Consider the following integer programming formulation of the minimum spanning tree problem:

$$\begin{aligned} \min \quad & \textstyle\sum_{e\in E} w_e x_e \quad . && (3.1)\\ \text{subject to} & \\ & \sum_{e\in E} x_e = n-1 && (3.2)\\ & \sum_{e\in E(S)} x_e \leq |S|-1 \text{ for any nonempty set } S \subset V \text{of nodes} && (3.3)\\ & x_e \geq 0 \text{ and integer for all edges } e. && (3.4) \end{aligned}$$

In this formulation, the 0-1 variable x_e indicates whether we select edge e as part of the chosen spanning tree (note that the second set of constraints with $|S| = 2$ implies that each $x_e \leq 1$). The constraint (3.2) is a cardinality constraint implying that we choose exactly $n-1$ edges, and the "packing" constraints (3.3) imply that the set of chosen edges contain no cycles (if the chosen solution contained a cycle, and S were the set of nodes on this cycle, then the solution would violate this constraint). Note that as a function of the number of nodes in the network, this model contains an exponential number of constraints. Nevertheless, as we will show, we can solve it very efficiently by applying the greedy algorithm.

To develop a bound on the weight of a minimum spanning tree, suppose that we associate a "Lagrange multiplier" μ_V with constraint (3.2) and nonnegative Lagrange multipliers μ_S with constraints (3.3), and add the weighted combination of these constraints to the objective function, creating the following optimization problem:

$$\begin{aligned} \min \sum_{e\in E} w_e x_e \quad + \quad & \mu_V[\sum_{e\in E} x_e - (n-1)] + \\ & \sum_{\phi\subset S\subset V} \mu_S[\sum_{e\in E(S)} x_e - (|S|-1)] && (3.5)\\ \text{subject to} & \\ & \sum_{e\in E} x_e = n-1 && (3.6)\\ & \sum_{e\in E(S)} x_e \leq (|S|-1) \text{ for any nonempty set } S \subset V \text{ of nodes} && (3.7) \end{aligned}$$

$$x_e \geq 0 \text{ and integer for all edges } e. \tag{3.8}$$

Note that for any feasible solution x to the problem and any value of the multiplier μ_V, the term $\mu_V[\sum_{e\in E} x_e - (n-1)]$ is zero. Moreover, for any feasible solution to the problem and any nonnegative choice of the multipliers μ_S for $S \subset V$, the last term in (3.5) is nonpositive. Therefore, the optimal value of this modified problem is a lower bound on the weight of any minimum spanning tree. Moreover, if we remove the constraints from this problem except for the 0-1 bounds on the variables, the problem's optimal objective value cannot become any larger, so the optimal objective value of the problem

$$\min \sum_{e\in E} w_e x_e + \mu_V[\sum_{e\in E} x_e - (n-1)] + \sum_{\phi\subset S\subset V} \mu_S[\sum_{e\in E(S)} x_e - (|S|-1)] \tag{3.9}$$

subject to

$$0 \leq x_e \leq 1 \text{ for all edges } e \tag{3.10}$$

is also a lower bound on the weight of any minimum spanning tree.

Let us record this result formally as the following property.

Lower Bounding Property *If μ_V is any scalar (positive, negative, or zero) and μ_S for each node nonempty set $S \subset V$ is any nonnegative scalar, then the optimal objective value of the problem (3.9)-(3.10) is a lower bound on the weight of any minimum spanning tree.*

In order to use this bounding property, let us collect together the terms in the objective function (3.9) by defining a "reduced weight" for any edge e as follows:

$$w_e^\mu = w_e + \sum_{E(S) \text{ contains edge } e} \mu_S.$$

The last term in this expression contains the multiplier μ_V associated with the constraint $\sum_{e\in E} x_e = n-1$ (corresponding to $S = V$). Using the reduced weight notation, we can write the lower bounding problem (3.9)-(3.10) as follows:

$$\min \sum_{e\in E} w_e^\mu x_e - \mu_V(n-1) - \sum_{\phi\subset S\subset V} \mu_S(|S|-1) \tag{3.11}$$

subject to

$$0 \leq x_e \leq 1 \text{ for all edges } e. \tag{3.12}$$

Observe that this problem is easy to solve. If $w_e^\mu < 0$, set $x_e = 1$; if $w_e^\mu > 0$, set $x_e = 0$; and if $w_e^\mu = 0$, set x_e to any value between 0 and 1. Since the problems (3.9)-(3.10) and (3.11)-(3.12), which are equivalent, provide us with a lower bound on the weight of any minimum spanning tree, if we can find a spanning tree whose weight equals the value of the lower bound, we can be guaranteed that the tree is a minimum spanning tree. To show that the greedy algorithm generates a minimum spanning tree, we use the greedy tree, together with a set of multipliers μ_S, to provide a certificate of optimality; that is, we use the tree and the multiplier data to ensure that the tree is a minimum spanning tree without the need to make any further computations (in particular, we need not explicitly consider the exponential number of other spanning trees).

Certificate of Optimality: *Suppose that the incidence vector y of a spanning tree T and the set of multipliers, μ_V unconstrained and $\mu_S \geq 0$ for all nonempty sets $S \subset V$, satisfy the following "complementary slackness" properties:*

$$(a) \quad \mu_S[\sum_{e \in E(S)} y_e - (|S| - 1)] = 0 \text{ for all nonempty } S \subset V \tag{3.13}$$

$$(b) \quad w_e^\mu = 0 \text{ if } y_e > 0 \tag{3.14}$$

$$(c) \quad w_e^\mu \geq 0 \text{ if } y_e = 0. \tag{3.15}$$

Then T is a minimum spanning tree.

Proof. Since y is a feasible spanning tree, $\sum_{e \in E} y_e = (n-1)$; when combined with condition (a), this result implies that the objective function (3.9), or equivalently (3.11), equals the weight $\sum_{e \in E} w_e y_e$ of the tree y. Therefore, if we can show that y solves the lower bounding problem (3.9)-(3.10), then we know that its weight is as small as the weight of any spanning tree and so it is a minimum spanning tree. But since the only constraints in problem (3.11)-(3.12) are the bounding conditions $0 \leq x_e \leq 1$ for all edges e, conditions (b) and (c) imply that y solves this problem. ∎

Example 3.1. As an illustration of this result, consider the tree generated by the greedy algorithm for our example. Suppose that we define the multipliers μ_S as shown in Table 1.

Set S	μ_S
{1,2}	1
{3,4}	1
{5,6}	1
{7,8}	1
{1,2,3,4}	4
{5,6,7,8}	4
{1,2,3,4,5,6,7,8}	-6

Table 1: Appropriate Lagrange multipliers

We set the multipliers of all other node sets S to value zero. With this choice of multipliers, the edges have the reduced weights shown in Table 2.

Let us make a few observations about the greedy solution y and the multipliers we have chosen. First, each step in the greedy algorithm introduces an edge joining two nodes i and j, and therefore forms a connected component $S(i,j)$ of nodes. For example, when the algorithm added edge f, it formed a connected component containing the nodes $5, 6, 7$, and 8. The number of edges in this component is $|S(i,j)| - 1 = 4 - 1 = 3$, so the set $S = S(i,j)$ of nodes satisfies the constraint $\sum_{e \in E(S)} y_e = |S| - 1$. Consequently, the multipliers and greedy solution y satisfy the first optimality condition (a). The only multipliers that we have set to nonzero values correspond to these sets (and to the overall set V). Consequently, since the greedy algorithm adds exactly $n - 1$ edges to the spanning tree, at most $n - 1$ of the multipliers are nonzero.

Note that in this case, since $y_a = y_b = y_c = y_d = y_e = y_f = y_g = 1$, and $y_h = y_i = y_j = 0$, the reduced weights satisfy the optimality conditions (b) and (c). Since the the greedy solution y and the multipliers also satisfy condition (a), we have in hand a certificate showing that the greedy solution is optimal.

In this case, the reduced weight for any edge not in the greedy tree is the difference in weight between that edge and the largest weight of any edge in the cycle formed by adding that edge to the tree. For example, if we add edge i to the greedy tree, it forms a cycle with the edges a, b, and e; edge e has the largest weight in this cycle and so the reduced weight for the edge i

Edge e	Reduced weight $w_e^\mu = w_e + \sum_{E(S) \text{ contains edge } e} \mu_S$.
a	1 + 1 + 4 - 6 = 0
b	1 + 1 + 4 - 6 = 0
c	1 + 1 + 4 - 6 = 0
d	1 + 1 + 4 - 6 = 0
e	2 + 4 - 6 = 0
f	2 + 4 - 6 = 0
g	6 - 6 = 0
h	6 - 6 = 0
i	3 + 4 - 6 = 1
j	3 + 4 - 6 = 1.

Table 2: Edge reduced weights

is $3 - 2 = 1$. The reason for this is that the edge i is contained in exactly the same sets $E(S)$ whose defining nodes S have positive multipliers as the edge e and so the difference in their reduced weights $w_e^\mu = w_e + \sum_{E(S) \text{ contains edge } e} \mu_S$ is the difference in their original weights.

To conclude this discussion, we might indicate how we chose values for the multipliers μ_S so that every edge in the greedy tree has a zero reduced weight. We set $\mu_V = -w_g$, the negative of the weight of the last edge added to the tree. When the greedy algorithm adds an edge $\alpha = \{i, j\}$ at any step, it combines the previous components of nodes containing nodes i and j, forming (in our earlier notation) a new connected component $S(i, j)$. To determine the multiplier μ_S, we consider what the algorithm does at a later step. At some subsequent step, it adds another another edge $\beta = \{p, q\}$ that enlarges the component $S(i, j)$ by connecting it to some other component. We set $\mu_{S(i,j)} = w_\beta - w_\alpha \geq 0$ (the provisions of the greedy algorithm ensure that edge β weighs at least as much as edge α). After the greedy algorithm has added edge α, at later steps it adds other edges $\beta, \gamma, ..., \phi, \nu$ that form increasingly larger sized components containing edge α. By our choice of the multipliers, these are the only node sets S, with $\alpha \in E(S)$, that receive nonzero multipliers. But our choice of the multipliers implies that $\sum_{E(S) \text{ contains edge } \alpha} \mu_S = (w_\beta - w_\alpha) + (w_\gamma - w_\beta) + ... + (w_\phi -$

$w_\nu) + w_\nu = -w_\alpha$. Therefore, the reduced weight of edge α is zero. Since α is an arbitrary edge in the greedy tree, this choice of multipliers ensures that every edge in the greedy tree receives a zero reduced weight.

This argument applies in general to the greedy tree produced by any application of the the greedy algorithm for any network and, therefore, provides an alternative proof that the greedy tree is a minimum spanning tree.

3.2 Polyhedral Representations

In the previous section, we showed how to use lower bounding information about the integer programming model (3.1)-(3.4) to demonstrate that the greedy algorithm generates a minimum spanning tree. In this section, we study the linear programming relaxation of this problem obtained by removing the integrality restrictions imposed on the variables. We also introduce several other formulations of the minimum spanning tree problem and study the polyhedra defined by their linear programming relaxations.

The study of integer programming models like (3.1)-(3.4) has become a fruitful topic within the field of combinatorial optimization; indeed, we will see its use in many of the following sections of this chapter as we consider more complex tree optimization problems. In general, because integer programming problems are hard to solve, the optimization community frequently solves some type of more easily solved relaxation of the problem. In the last subsection we considered one such type of relaxation, known as Lagrangian relaxation.

Perhaps the most popular type of relaxation is the linear programming relaxation obtained by eliminating the restriction that the decision variables x in the model (3.1)-(3.4) need to be integer. In general, since we have eliminated the integrality restriction from the model, the linear programming relaxation will have a lower optimal objective value than does the integer program. As we show next, for the minimum spanning tree problem, this is not the case. In this setting, the integer program and linear program have the same optimal objective value since any solution to the integer program (in particular, the greedy solution) solves the linear programming relaxation.

Although we have not noted this important result before, we have actually already established it. For suppose that we start with the linear programming

relaxation of the minimum spanning tree formulations (3.1)-(3.4), obtained by replacing the constraints "$x_e \geq 0$ and integer for all edges e" by the relaxed constraints "$x_e \geq 0$ for all edges e", and apply the same lower bounding arguments that we used for proving that the greedy solution solves the integer programming model. Then we also see that the the greedy solution solves the linear programming relaxation. In fact, we might interpret our lower bounding argument as follows: let z^{ST} denote the optimal objective value of the spanning tree integer program and let z^{lp} denote the optimal objective function value of its linear programming relaxation. Suppose that we form the following linear programming dual of the linear programming relaxation.

$$\max -\mu_V(n-1) - \sum_{S \neq V} \mu_S(|S|-1) \tag{3.16}$$

subject to

$$-\sum_{EA(S) \text{ contains edge } (i,j)} \mu_S \leq w_e \text{ for all edges } e \tag{3.17}$$

$$\mu_S \geq 0 \text{ for all } S \neq V. \tag{3.18}$$

As before, the expression to the lefthand side of the inequality (3.17) contains the term μ_V. Since the linear program is a relaxation of the integer programming model $z^{lp} \leq z^{ST}$ and by linear programming duality theory the value of the linear program equals the value of its dual linear program (3.16)-(3.18). The multiplier values that we defined in the lower bounding argument satisfy the constraints (3.17) and (3.18) of the linear programming dual problem. Moreover, note that the way we have defined the multipliers, if w_α is the weight of any edge added to the greedy tree, then w_α contributes to the objective function (3.16) in one or more terms: (i) it contributes $-w_\alpha$ to μ_S, corresponding to the set S of nodes in a single component that is formed when we add edge α to the tree; and (ii) it contributes w_α to the two the multipliers μ_Q and μ_P of the sets Q and P of nodes that define the components that are combined when we add edge α to the tree. But since $(|S|-1) = (|Q|-1) + (|P|-1) + 1$, the overall effect is to add w_α to the objective function. Since this is true for every edge α that we add to the greedy tree, the objective function of the linear program has the same value $\sum_{e \in T_{greedy}} w_e x_e$ as the greedy tree. Therefore, $z^{lp} = z^{ST}$ and so the greedy tree, which is feasible in the linear program, solves this problem.

Note that this development has not only shown that the optimal value of the integer programming problem (3.1)-(3.4) and its linear programming

relaxation are the same, but has also shown that that the linear programming relaxation always has an integer optimal solution (which is an incidence vector of a spanning tree) for any choice of the objective function weights. Moreover, we can always choose the weights so that any particular spanning tree is a solution to the linear program. A standard result in linear programming shows that the incidence vectors of spanning trees must then be the extreme points of the linear programming relaxation. Therefore, we have established the following important result.

Theorem 3.2 *The extreme points of the polyhedron defined by the linear programming relaxation of the spanning tree model (3.1)-(3.4) are the 0-1 incidence vectors of spanning trees.*

Alternative Formulations

Generally, it is possible to devise many valid formulations of integer programming problems, using alternate sets of variables and/or constraints.

One formulation might be preferable to another for some purposes, for example, because it is easier to study theoretically or because it is solvable by a particular solution procedure. The model (3.1)-(3.4) that we have introduced in this section is a "natural packing" formulation. It is a "natural" formulation because it uses the natural 0-1 decision variables, indicating whether or not we include the underlying edges in the spanning tree; we refer to it as a packing formulation because the constraints $\sum_{e \in E(S)} x_e \leq |S| - 1$ restrict the number of edges that we can pack within any set S of nodes. In this discussion we examine two alternative approaches for formulating the minimum spanning tree problem, one involving cuts and another involving flows.

Cutset Formulations

Let $\mathcal{S}$ denote the set of incidence vectors of spanning trees of a given graph $G = (V, E)$. In formulating the spanning tree problem as an integer program, we used the property that a spanning tree on an n node graph is any subgraph containing $n-1$ edges and no cycles. Therefore, at most $|S|-1$ edges in any tree can connect any set S of nodes, and so $\mathcal{S} = \{x \in Z^{|E|} : 0 \leq x \leq \mathbf{1}, \sum_{e \in E} x_e = n-1$, and $\sum_{e \in E(S)} x_e \leq |S|-1$ for any nonempty $S \subset V\}$.

As an alternative, we could use another equivalent definition of a spanning tree: it is a connected subgraph containing $n-1$ edges. This definition leads to a cutset formulation: $\mathcal{S} = \{x \in Z^{|E|} : 0 \leq x \leq \mathbf{1}, \sum_{e \in E} x_e = n-1$, and $\sum_{e \in \delta(S)} x_e \geq 1$ for all nonempty node sets $S \subset V\}$. As we saw in Theorem 3.2, if we relaxed the integrality restrictions in the packing (subtour) formulation, then the resulting polyhedron P_{sub} equals $conv(\mathcal{S})$. Let P_{cut} denote the polyhedron formed if we remove the integrality restrictions from the cutset formulation, that is, $P_{cut} = \{x \in R^{|E|} : 0 \leq x \leq \mathbf{1}, \sum_{e \in E} x_e = n-1$, and $\sum_{e \in \delta(S)} x_e \geq 1$ for all nonempty nodes sets $S \subset V\}$. As we show next, the polyhedron P_{cut} can be larger than the convex hull of S.

Proposition 3.3 *$P_{cut} \supseteq P_{sub}$. In general, P_{cut} has fractional extreme points and so is larger than P_{sub}.*

Proof. Note that for any set S of nodes, $E = E(S) \cup \delta(S) \cup E(\overline{S})$. If $x \in P_{sub}$, then $\sum_{e \in E(S)} x_e \leq |S|-1$ and $\sum_{e \in E(\overline{S})} x_e \leq |\overline{S}|-1$. Therefore, since $\sum_{e \in E} x_e = n-1, \sum_{e \in \delta(S)} x_e \geq 1$. Consequently, $x \in P_{cut}$ and so $P_{cut} \supseteq P_{sub}$.

To establish the second part of the proposition, consider Figure 9. Recall that a polyhedron has integer extreme points if and only if a linear program defined over it has an integer optimal objective value for all choices of integer objective coefficients (whenever the optimal value is finite). In Figure 9, if we define a linear program $min\ \{\gamma x : x \in P_{cut}\}$ by setting the objective coefficients of edges $\{1,2\}, \{1,3\}$ and $\{2,4\}$ to value 1 and the coefficients of edges $\{3,4\}, \{4,5\}$, and $\{3,5\}$ to value 0, then the optimal solution is the fractional solution x shown in the figure with an objective value of 3/2. Note that x belongs to P_{cut} but not to P_{sub} since the edges $E(S)$ in the 3-node set $S = \{3,4,5\}$ have a weight 5/2. ∎

This result shows that the linear programming relaxation of a "simple" cutset formulation does not define the convex hull of incidence vectors of spanning trees. We next show that if we replace simple cutsets by "multicuts," then the linear programming relaxation will define $conv(S)$.

Given a $k+1$ node partition of V, that is disjoint nonempty sets $C_0, C_1, \ldots, C_k$ of nodes whose union is V, we call the set of edges, denoted as $\delta(C_0, C_1, \ldots, C_k)$, with one end point in one of these node sets and the other end point in another, a *multicut*. Note that a (usual) cut is just a multicut with $k = 1$. Since any spanning tree contains at least k edges joining the sets $C_0, C_1, \ldots, C_k$, $|\delta(C_0, C_1, \ldots, C_k) \cap T| \geq k$. Therefore, any spanning tree lies

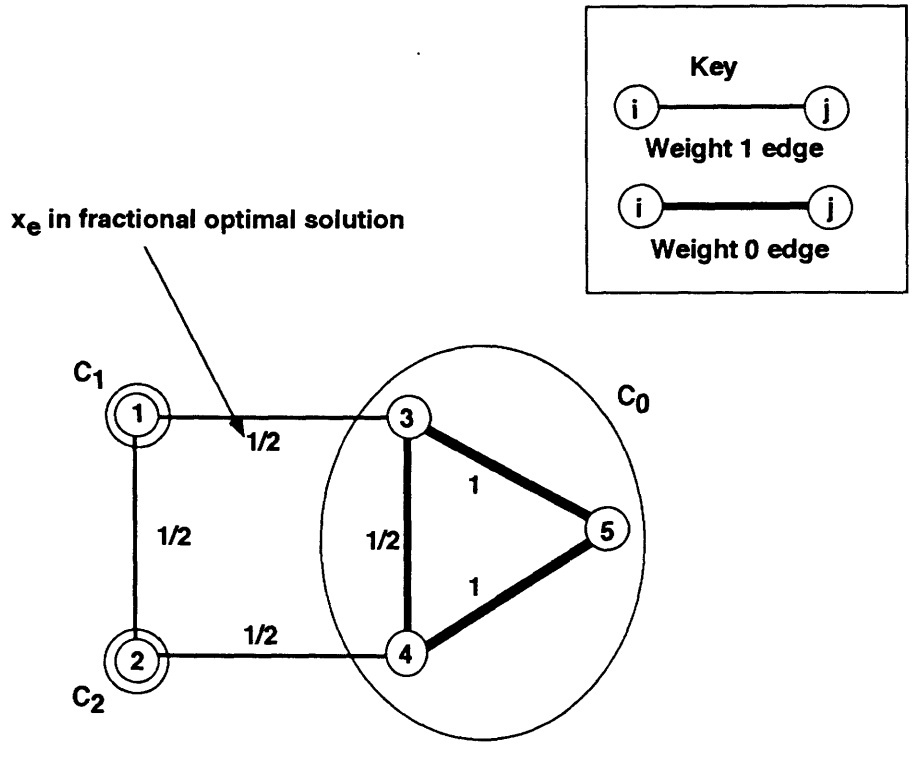


Figure 9: Fractional Optimal Solution to Cut Relaxation

in the multicut polyhedron $P_{mcut} = \{x \in R^{|E|} : 0 \leq x \leq 1, \sum_{e \in E} x_e = n-1,$ and $\sum_{e \in \delta(C_0, C_1, \ldots, C_k)} x_e \geq k$ for all node partitions $C_0, C_1, \ldots, C_k$ of $V\}$. The following result gives a strengthening of Proposition 3.3.

Theorem 3.4 *$P_{mcut} = conv(\mathcal{S})$.*

Proof. Consider any set S of $n-k$ nodes and node partition with $C_0 = S$, and C_j for $j = 1, \ldots, k$ as singletons each containing one of the nodes of $\overline{S}$.

Since $E = E(S) + \delta(C_0, C_1, \ldots, C_k)$, if $0 \leq x \leq 1$ and $\sum_{e \in E} x_e = n-1$, then $\sum_{e \in E(S)} x_e \leq |S| - 1$ if and only if $\sum_{e \in \delta(C_0, C_1, \ldots, C_k)} x_e \geq n - 1 - (|S| - 1) = k$. Consequently, $x \in P_{sub}$ if and only if $x \in P_{mcut}$. Therefore, $P_{mcut} = P_{sub}$ and, by Theorem 3.1, $P_{mcut} = conv(\mathcal{S})$. ∎

This proof uses our prior results concerning the relationship between the greedy algorithm and the polyhedron P_{sub}. The following alternative proof, based on a lower bounding or duality argument, is also instructive; it shows a direct relationship between the greedy algorithm and the polyhedron P_{mcut}.

Alternative Proof. Consider the following dual to the the linear program $\min\{wx : x \in P_{mcut}\}$:

$$\begin{aligned} & \max \quad \sum \quad k\mu_{C_0,\ldots,C_k} \\ & \text{subject to} \\ \sum_{e \in \delta(C_0,\ldots,C_k)} & \mu_{C_0,\ldots,C_k} \leq w_e \text{ for all } e \in E \\ & \mu_{C_0,\ldots,C_k} \geq 0 \quad \text{whenever } k \leq n-2. \end{aligned}$$

In this formulation, $\mu_{C_0,\ldots,C_k}$ is the dual variable corresponding to the constraint $\sum_{e \in \delta(C_0,\ldots,C_k)} x_e \geq k$, $\mu_{\{1\},\ldots,\{n\}}$ is the dual variable corresponding to the constraint $\sum_{e \in E} x_e = n-1$, and the sum in the objective function of this problem is taken over all multicuts $\{C_0, \ldots, C_k\}$.

Suppose the greedy algorithm chooses edges S_1 of weight w_{S_1}, S_2 of weight w_{S_2}, and so forth and that $w_{S_1} < w_{S_2} < \ldots < w_{S_r}$. Suppose further that $(C_0^i, \ldots, C_{k_i}^i)$ are the connected components of the graph spanned by the edges $\cup_{j=1}^{i} S_j$, and that we set $\mu_{\{1\},\ldots,\{n\}} = w_{S_1}$, $\mu_{C_0^i,\ldots,C_{k_i}^i} = w_{S_{i+1}} - w_{S_i}$, for $i = 1, \ldots, r-1$, and $\mu_{C_0,\ldots,C_k} = 0$ for any other multicut $\{C_0, \ldots, C_k\}$. It is easy to see that the values for $\mu_{C_0,\ldots,C_k}$ provide an optimal dual solution

of the same value as the greedy solution, and therefore, as we have argued before, all the extreme points of the polyhedron P_{mcut} are integral [2]. ∎

Example 3.2.

Consider again the example in Figure 8. In this case:

- $k_0 = 7$ and $\mu_0 \equiv \mu_{\{1\},\ldots,\{n\}} = 1$.
- $k_1 = 3$ and $S_1 = \{\{1,2\},\{3,4\},\{5,6\},\{7,8\}\}$. Therefore, $\mu_1 \equiv \mu_{\{1,2\},\{3,4\},\{5,6\},\{7,8\}} = 2 - 1 = 1$.
- $k_2 = 1$ and $S_2 = \{\{1,2,3,4\},\{5,6,7,8\}\}$. Consequently, $\mu_2 \equiv \mu_{\{1,2,3,4\},\{5,6,7,8\}} = 6 - 2 = 4$..

Observe that $\sum k_i\mu_i = 14$ equals the objective value of the greedy solution. ∎

Note that in the first proof of Theorem 3.4, we have actually proved a slightly stronger version of the theorem. In selecting the multicuts in the definition of P_{mcut}, we can choose all but one of them to be a singleton. Therefore, the multicuts have one large node set S; the remaining node sets are singletons. To contrast the result in Proposition 3.3 and Theorem 3.4, once again consider the example in Figure 9. Let $C_0 = \{3,4,5\}, C_1 = \{1\}$, and $C_2 = \{2\}$. Note that $\sum_{e\in\delta(C_0,C_1,C_2)} x_e = 3/2$ so this solution does not satisfy the multicut constraint $\sum_{e\in\delta(C_0,C_1,C_2)} x_e \geq 2$.

Flow Formulations

Another way to conceive of the minimum spanning tree problem is as a special version of a network design problem: in this setting, we wish to send flow between the nodes of the network and view the edge variable x_e as indicating whether not we install the edge e to be available to carry any

[2]This proof also shows that if we eliminate the cardinality constraint $\sum_{e\in E} x_e = n - 1$ from the polyhedron (the polyhedron still has the constraint $\sum_{e\in E} x_e \geq n - 1$), the resulting polyhedron, which we denote as P^+_{mcut} also has integer extreme points (corresponding to spanning trees). In this case, if any weight $w_e < 0$, then we let x_e approach $+\infty$, showing that the linear program $\min\sum_{e\in E}\{w_e x_e : e \in P^+_{mcut}\}$ has no optimal solution. If each $w_e \geq 0$, then the argument in the proof shows that this linear program has an optimal solution with x as the incidence vector of a spanning tree.

flow. We consider three such flow models: a single commodity model, a multicommodity model, and a refined multicommodity model. In each of these models, although the edges are undirected, the flow variables will be directed. That is, for each edge $e = \{i, j\}$, we will have flow in both the directions i to j and j to i.

In the *single commodity model*, one of the nodes, say node 1, serves as a source node; it must send one unit of flow to every other node. Let f_{ij} denote the flow on edge $e = \{i, j\}$ in the direction i to j. The model is:

$$\begin{aligned}
& \min \quad wx && (3.19)\\
& \text{subject to} \\
& \sum_{e \in \delta^+(1)} f_e - \sum_{e \in \delta^-(1)} f_e = n - 1 && (3.20)\\
& \sum_{e \in \delta^-(v)} f_e - \sum_{e \in \delta^+(v)} f_e = 1 \text{ for all } v \neq 1, v \in V && (3.21)\\
& f_{ij} \leq (n-1)x_e \text{ for every edge } e = \{i, j\} && (3.22)\\
& f_{ji} \leq (n-1)x_e \text{ for every edge } e = \{i, j\} && (3.23)\\
& \sum_{e \in E} x_e = n - 1 && (3.24)\\
& f \geq 0, \text{ and } 0 \leq x_e \leq 1 \text{ for all edges } e && (3.25)\\
& x_e \text{ integer for all edges } e \in E. && (3.26)
\end{aligned}$$

In this model, equations (3.20) and (3.21) model flow balances at the nodes; the forcing constraints (3.22) and (3.23) are redundant if $x_e = 1$ and state that the flow on edge e, in both directions, is zero if $x_e = 0$. Note that this model imposes no costs on the flows. The mass balance equations imply that the network defined by any solution x (that is, those edges with $x_e = 1$) must be connected. Since the constraint (3.24) states that the network defined by any solution contains $n - 1$ edges, every feasible solution must be a spanning tree. Therefore, when projected into the space of the x variables, this formulation correctly models the minimum spanning tree problem.

If we ignore the integrality restrictions on the x variables, the constraints of this model determine a polyhedron. Let P_{flo} denote the set of feasible solutions to this linear program, that is to the system (3.20)-(3.25), in the x-space. We will use Figure 10 to illustrate the fact that this formulation

of the minimum spanning tree problem is quite weak in the sense that the linear program $min\{\gamma x : x \in P_{flo}\}$ can be a poor representation of the weight of a minimum spanning tree as determined by the integer program min $\{\gamma x : x$ is an incidence vector of a spanning tree$\}$. This is the same example used in Figure 9; the edge weights γ_e all have values zero or one as shown. The optimal weight of a minimum spanning tree is 2 and, as we saw before, the optimal weight of the linear programmimng relaxation of the cut formulation is 3/2. In this case, suppose we choose node 1 as the root node. It has a supply of 4 units. The optimal solution sends the three units of demand destined for nodes 2, 4, and 5 on edge $\{1,3\}$ and the one unit of demand destined for node 2 on edge $\{1,2\}$. To do so, it requires a capacity $(n-1)x_e = 3$ on edge $\{1,3\}$ and a capacity of $(n-1)x_e = 1$ on edge $\{1,2\}$, giving a weight $x_e = 3/4$ on edge $\{1,3\}$ and a weight of $1/4$ on edge $\{1,2\}$. So in this case, the linear programming relaxation of the flow formulation has value 1, which is even less than value of the linear programming relaxation of the cut formulation.

As we will see, even though the differences between the minimum spanning tree problem, the cut formulation, and the flow formulation apply in general, we can considerably strengthen the cut and flow formulations. Indeed, we will derive tight cut and flow models, that is, formulations whose underlying polyhedrons equal the convex hull of the incidence vectors of the spanning trees. We begin by introducing a directed version of P_{sub}.

Note that if we select any node r as a root node for any spanning tree, then we can direct the edges of the tree so that the path from the root node to any other node is directed from the root to that node. To develop a model for this directed version of the problem, we consider a digraph $D = (V, A)$ formed by replacing each edge $\{i,j\}$ in E by arcs (i,j) and (j,i) in A. Let $y_{ij} = 1$ if the tree contains arc (i,j) when we root it at node r.

The *directed model* is:

$$\sum_{e \in A(S)} y_e \leq |S| - 1 \text{ for any nonempty set } S \subset V \text{of nodes} \quad (3.27)$$

$$\sum_{e \in \delta^-(v)} y_e = 1 \text{ for all } v \in V \setminus \{r\} \quad (3.28)$$

$$\sum_{e \in A} y_e = n - 1 \quad (3.29)$$

$$y_e \geq 0 \text{ for all edges } e \in A \quad (3.30)$$

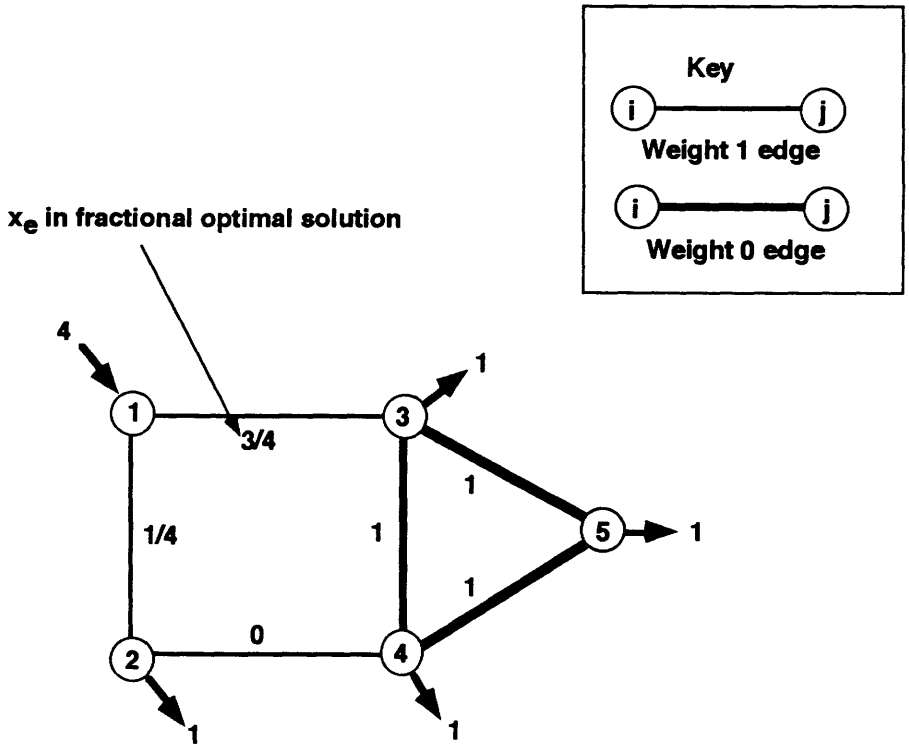


Figure 10: Fractional Optimal Solution to the Flow Relaxation

$$x_e = y_{ij} + y_{ji} \text{ for all edges } e \in E. \tag{3.31}$$

Note that the constraints (3.28) in this model imply that $\sum_{k \neq r} \sum_{e \in \delta(k)} y_e = n - 1$ which, combined with equation (3.29), implies that $\sum_{e \in \delta(r)} y_e = 0$. Therefore, $y_e = 0$ for every arc e directed into the root node.

Let P_{dsub} denote the feasible set for this model in x-space. Since every tree can be directed, $\mathcal{S} \subseteq P_{dsub}$ and so $\text{conv}(\mathcal{S}) = P_{sub} \subseteq P_{dsub}$. Note that if $x \in P_{dsub}$, then since y_{ij} and y_{ji} occur together in the constraints (3.27) and (3.29), $x \geq 0$ satisfies constraints (3.2) and (3.3) and so $x \in P_{sub}$ implying that $P_{dsub} \subseteq P_{sub}$. We have therefore established the following result[3].

Proposition 3.5 $P_{sub} = P_{dsub}$.

Now consider a *directed cut model* with some arbitrary node r chosen as a "root node":

$$\sum_{e \in \delta^+(C)} y_e \geq 1 \text{ for all } C \text{ with } r \in C \tag{3.32}$$

$$\sum_{e \in A} y_e = n - 1 \tag{3.33}$$

$$y_e \geq 0 \text{ for all edges } e \in A \tag{3.34}$$

$$x_e = y_{ij} + y_{ji} \text{ for all edges } e \in E. \tag{3.35}$$

Let P_{dcut} denote the set feasible solutions of this model in x-space. The constraint (3.32) in this model states that every directed cut $\delta(C)$ that separates the root node r from any set of nodes $\overline{C}$ must contain at least one arc.

Before studying the directed cut model, let us make one observation about its formulation. If we set $C = V \setminus \{k\}$ in constraint (3.32), then for each node $k \neq r$, this model contains the constraint $\sum_{e \in \delta^-(k)} y_e \geq 1$. In contrast, the directed formulation states these constraints as equalities (3.28). Note that if we add the inequalities $\sum_{e \in \delta^-(k)} y_e \geq 1$ over all nodes $k \neq r$, we obtain the inequality $\sum_{e \in A(V \setminus \{r\})} y_e + \sum_{e \in \delta^+(r)} y_e \geq n - 1$. If any vector y were to satisfy any inequality $\sum_{e \in \delta^-(k)} y_e \geq 1$ in (3.32) as a strict inequality, it would also satisfy the last inequality a strict inequality as well, contradicting equation (3.33). Therefore, every feasible solution y in the directed model satisfies

[3]In this proof, we have used the integrality property $conv(\mathcal{S}) = P_{sub}$. In establishing Theorem 6.4, we give a proof of a more general result without using the integrality property.

equation (3.28), that is, $\sum_{e \in \delta^-(k)} y_e = 1$ for all $k \neq r$. (As before, these equalities imply that the weight $y_e = 0$ for every arc e directed into the root node.) These conclusions tells us something about the relationship between the directed model and the directed cut model. As the next result shows, we can say even more.

Proposition 3.6 $P_{dsub} = P_{dcut}$.

Proof. Let S be any set of nodes containing node r. Since $A = A(\overline{S}) \cup \delta^+(S) \cup_{k \in S} \delta^-(k)$, we can rewrite equation (3.29) as

$$\sum_{e \in A(\overline{S})} y_e + \sum_{e \in \delta^+(S)} y_e + \sum_{k \in S} \sum_{e \in \delta^-(k)} y_e = n - 1. \tag{3.36}$$

The constraints (3.28), which as we have just seen are valid in both the directed model and the directed cut model, and the condition that $y_e = 0$ for every edge directed into the root node r, imply that the last term on the left-hand side of this equation equals $|S| - 1$ and so equation (3.36) becomes

$$\sum_{e \in A(\overline{S})} y_e + \sum_{e \in \delta^+(S)} y_e = n - 1 - (|S| - 1) = |\overline{S}|. \tag{3.37}$$

But this inequality implies that (3.27) is valid for $\overline{S}$ if and only if (3.32) is valid for S. Therefore, the constraints (3.32) are equivalent to the constraints (3.27) applied to all sets that do not contain the root node r.

If r is not contained in S, then the last term in equation (3.36) equals $|S|$ and so the right-hand side of equation (3.37) becomes $|\overline{S}| - 1$. But then, since the second term in this equation is nonnegative, the eqaution implies that $\sum_{e \in A(\overline{S})} \leq |S| - 1$ and so the inequality (3.27) is valid whenever r does not belong to S as well.

But these two conclusion imply that a vector y is feasible in the directed model if and only if it is feasible in the directed cut model and, therefore, $P_{dsub} = P_{dcut}$. ∎

The max-flow min-cut theorem immediately provides us with a reformulation, called the *directed multicommodity flow model*, of the directed cut formulation. In this model every node $k \neq r$ defines a commodity: one unit

of commodity k originates at the root node r and must be delivered to node k. Letting f_{ij}^k be the flow of commodity k in arc (i,j), we formulate this model as follows:

$$\sum_{e\in\delta^-(r)} f_e^k - \sum_{e\in\delta^+(r)} f_e^k = -1 \text{ for all } k \neq r \tag{3.38}$$

$$\sum_{e\in\delta^-(v)} f_e^k - \sum_{e\in\delta^+(v)} f_e^k = 0 \text{ for all } v \neq r, v \neq k, \text{and all } k \tag{3.39}$$

$$\sum_{e\in\delta^-(k)} f_e^k - \sum_{e\in\delta^+(k)} f_e^k = 1 \text{ for all } k \neq r \in V \tag{3.40}$$

$$f_{ij}^k \leq y_{ij} \text{ for every arc } (i,j) \text{ and all } k \neq r \tag{3.41}$$

$$\sum_{e\in A} y_e = n-1 \tag{3.42}$$

$$y_{ij} + y_{ji} = x_e \text{ for every edge } e \in E \tag{3.43}$$

$$f \geq 0, \text{and } y_e \geq 0 \text{ for all arcs } e \in A \tag{3.44}$$

$$y_e \text{ integer for all arcs } e \in A. \tag{3.45}$$

In this model, the variable y_{ij} defines a capacity for the flow of each commodity k in arc (i,j). The *forcing constraint* (3.41) implies that we can send flow of each commodity on arc (i,j) only if that arc is a member of the directed spanning tree defined by the variables y. We let P_{dflo} denote the set of feasible solutions of the linear programming relaxation of this model in the x-space, that is, of the constraints (3.38)-(3.44).

Proposition 3.7 $P_{dflo} = P_{dcut}$.

Proof. From the max-flow min-cut theorem, $\sum_{e\in\delta^+(C)} y_e \geq 1$ for all C with $r \in C$ and $k \notin C$ if and only if the digraph has a feasible flow of 1 unit from the root to node k with arc capacities y_{ij}, that is, if and only if the system (3.38)-(3.41) has a feasible solution with $f^k \geq 0$. This observation implies the proposition. ∎

We obtain a closely related formulation by eliminating the y_{ij} variables. The resulting formulation is (3.38)-(3.42), plus $f \geq 0$, plus

$$\sum_{e\in E} x_e = n-1 \tag{3.46}$$

$$f_{ij}^k + f_{ji}^{k'} \leq x_e \text{ for all } k, k' \text{ and all } e \in E. \tag{3.47}$$

We refer to this model as the *extended multicommodity flow formulation* and let $P_{mc'flo}$ denote its set of feasible solutions in x-space. Observe that since we have eliminated the variables y_e in constructing the extended multicommodity flow formulation, this model is defined on the undirected graph $G = (V, E)$, even though for each commodity k, we permit flows f_{ij}^k and f_{ji}^k in both directions on edge $e = \{i, j\}$. The *bidirectional flow inequalities* (3.46) in this formulation link the flow of different commodities flowing in different directions on the edge $\{i, j\}$. These constraints model the following fact: in any feasible spanning tree, if we eliminate edge $\{i, j\}$, we divide the nodes into two components; any commodity whose associated node lies in the same component as the root node does not flow on edge $\{i, j\}$; any two commodities whose associated nodes both lie in the component without the root both flow on edge $\{i, j\}$ in the same direction. So, whenever two commodities k and k' both flow on edge $\{i, j\}$, they both flow in the same direction and so one of f_{ij}^k and f_{ji}^k equals zero.

Note that equalities (3.42) and (3.43) imply (3.46) and the inequalities (3.41) and equalities (3.43) imply (3.47). Therefore, $P_{dflo} \subseteq P_{mc'flo}$. Conversely, suppose the vectors x and f are feasible in the extended multicommodity flow problem; for each edge $e = \{i, j\}$, choose an arc direction (i, j) arbitrarily, and define $y_{ij} = \max_{k \neq r} f_{ij}^k$ and $y_{ji} = x_e - y_{ij}$. Then y and f are feasible in the directed flow formulation. Therefore, we have established the following result.

Proposition 3.8 $P_{dflo} = P_{mc'flo}$.

We obtain one final formulation, which we refer to as the *undirected multicommodity flow model*, by replacing the inequalities (3.47) by the weaker constraints

$$f_{ij}^k \leq x_e \text{ for all } k \neq r \text{ and } e \in E.$$

Let P_{mcflow} denote the set of feasible solutions of this model in the x-space. As in the proof of Proposition 3.6, the max-flow min-cut theorem implies that this model is equivalent to a cut formulation in the following sense.

Proposition 3.9 $P_{mcflow} = P_{cut}$.

Let us pause to reflect upon the results we have just established. In this section we have examined the polyhedra defined by the linear programming

$$\left\{ \begin{array}{c} P_{sub} \\ P_{mcut} \\ P_{mc'flo} \\ P_{dsub} \\ P_{dcut} \\ P_{dflo} \end{array} \right\} \subseteq \left\{ \begin{array}{c} P_{cut} \\ P_{mcflo} \end{array} \right\} \subseteq P_{flo}$$

Figure 11: Relationship Between Underlying Polyhedra

relaxation of nine different formulations of the minimum spanning tree problem. Figure 11 shows the relationship between these polyhedra. Six of the polyhedra—the subtour, multicut, extended multicommodity flow, directed spanning tree, directed cut, and directed flow polyhedra are the same (the latter three when transformed into the space of x variables); each has integer extreme points. The cut and multicommodity flow are the same; like the weaker flow formulation, they can have fractional extreme points and so they do not define the spanning tree polyhedron.

There are several lessons to be learned from this development. First, for undirected formulations, multicuts improve upon models formulated with cuts and the bidirectional flow inequalities (3.47) improve upon the multicommodity flow formulation (at the expense of adding considerably more constraints to the formulation). Second, (extended) multicommodity flow formulations, which have the disadvantage of introducing many additional flow variables, improve upon single commodity flow formulations. Third, even though the polyhedra P_{sub} and $P_{mc'flo}$ are the same, we obtain $P_{mc'flo}$ by projecting out the flow variables from the multicommodity flow formulation; this formulation, and indeed each of the flow formulations we have examined, are "compact" in the sense that the number of variables and constraints in these models is polynomial in the size of the underlying graph. The subtour formulation and cut formulations contain fewer variables, but are "exponential" in the sense that the number of constraints in these models grows exponentially in the graph's size. We obtain the compact formulations by introducing "auxiliary" flow variables beyond the natural 0-1 edge variables. Finally, we have shown that a directed flow model gives a better representation of the spanning tree polyhedron than does the undirected flow

model (unless we add the bidirectional contraints (3.47) to the undirected model).

These observations provide powerful modeling lessons that extend well beyond the realm of spanning trees. For example, they apply to several network design and vehicle routing problems. Later in this chapter, we will illustrate the use of these ideas again as we study other versions of optimal tree problems (especially variations of the Steiner tree problem).

Linear Programs, Cutting Planes, and Separation

To conclude this section, we consider an important algorithmic implication of the formulations we have considered. As we have noted, both the basic packing formulation (3.1)-(3.4) and the directed multicommodity flow formulation (3.38)-(3.44) are large optimization models; the number of contraints in the packing formulation grows exponentially in the number $|V|$ of nodes in the underlying graph; the directed cutset formulation has $|V|^3$ flow variables and forcing constraints (3.41)—1 million for a problem with 1000 nodes. Fortunately, we need not explicitly solve either model since we can use the greedy algorithm to solve the minimum spanning tree problem very efficiently.

What happens, however, when the spanning tree problem is a subproblem in a more general model that cannot be solved using a combinatorial algorithm like the greedy algorithm? Examples are spanning tree problems with additional constraints imposed upon the network topology (see Section 8) or the traveling salesman, with its embedded spanning tree structure. In solving these problems, many algorithms atempt to solve the linear programming relaxation of the problem (and then often use an enumeration procedure such as branch and bound). If we model spanning tree solutions using the packing formulation, any feasible point in the linear programming relaxation of these more general problems must both lie in the set P_{sub} and satisfy any additional contraints imposed upon the problem variables. Since we cannot possibly even list all the contraints of the packing formulation for any reasonably sized problem, linear programming algorithms often adopt a "cutting plane" or "constraint generation" procedure that works as follows.

We first formulate and solve a linear programming model ignoring all but a few of the packing contraints (3.3). If the solution $\overline{x}$ of this model happens to satisfy all the packing constraints that we have ignored, then $\overline{x} \in P_{sub}$.

To discover if this is the case, we would like to determine if the solution $\overline{x}$ violates any packing constraint: that is, if possible, we would like to find a set of nodes S for which $\sum_{e \in E(S)} \overline{x}_e > |S| - 1$. If we found such a set, we would add the constraint $\sum_{e \in E(S)} x_e \leq |S| - 1$ to the linear programming model. In polyhedral combinatorics, the problem of finding such a violated contraint, or cut as it is known, is called the *separation problem* since we are finding a violated inequality that separates the vector $\overline{x}$ from the polyhedron P_{sub}.

Solving the Separation Problem

How can we solve the separation problem? The directed cutset formulation and the development in this section have implicitly provided us with an answer to this question. As we have seen, $P_{sub} = P_{dflo}$. Stated in another way, $\overline{x} \in P_{sub}$ if and only if $\overline{y}$, with $\overline{x}_e = \overline{y}_{ij} + \overline{y}_{ji}$ for all edges $e = \{i, j\}$, together with some flow vector $\overline{f}$ is feasible in the directed multicommodity flow formulation (3.38)-(3.44). Suppose we set the capacity of each arc (i, j) in this model to x_e; then $\overline{x} \in P_{sub}$ if and only if the capacitated network has a flow of one unit from the root node r to every other node k (since we can always find the maximum flow between two nodes by sending flow only in one of the arcs (i, j) and (j, i)). Any maximum flow algorithm that finds a minimum cut as well as a maximum flow will give us this answer: for suppose for each node k, we find a maximum flow from the root node to node k. If the maximum flow has value $\theta < 1$, then by the max-flow min-cut theorem, some directed cut $\delta^+(S)$ has a capacity less than 1, that is, $\sum_{e \in \delta(S)} \overline{x}_e = \sum_{e \in \delta^+(S)} \overline{x}_e < 1$. But since $\sum_{e \in E} \overline{x}_e = |V| - 1$, either $\sum_{e \in S} \overline{x}_e > |S| - 1$ or $\sum_{e \in \overline{S}} \overline{x}_e > |\overline{S}| - 1$ and so the minimum cut provides us with a violated packing inequality. We could then add this inequality to the linear program, solve it again, and repeat the separation procedure.

Most Violated Constraint[4]

The method we have described is guaranteed to find a violated packing inequality, or show that none exists, by solving $|V| - 1$ maximum flow problems. It isn't, however, guaranteed to find a most violated packing inequality, that is, one that maximizes $\sum_{e \in E(S)} \overline{x}_e - (|S| - 1)$ over all node sets S. We next show that by cleverly defining an ancilliary maximum flow network,

[4]Readers can skip the next two paragraphs without any loss of continuity.

we can find a most violated constraint using the same number of maximum flow computations. To find a most violated inequality we wish to maximize $\sum_{e \in E(S)} \overline{x}_e - (|S| - 1)$, or, equivalently, minimize $|S| - \sum_{e \in E(S)} \overline{x}_e$ over all nonempty node sets $S \subset V$. Since $\sum_{e \in E} \overline{x}_e = |V| - 1$, a constant, this minimization is equivalent to minimizing $|S| + \sum_{e \notin E(S)} \overline{x}_e$. We will show how to minimize this quantity. If its minimum has a value of at least $|V|$, then the solution $\overline{x}$ satisfies all the packing constraints.

As before, we solve $|V| - 1$ maximum flow (minimum cut) problems, each defined on a directed version $G^* = (V, A)$ of the network $G = (V, E)$. In the k^{th} such problem, we find a cut that separates the root node r from node k. For every edge $e = \{i, j\}$ in E, G^* contains the arcs (i, j) and (j, i); we set the capacity of any directed arc (p, q) obtained from edge e to $(1/2)\overline{x}_e + (1/2) \sum_{e \in \delta(q)} \overline{x}_e$. For each node $v \in V$, we also include an arc (v, k), which might be a parallel arc, with a capacity of one unit. Note that if S is any node set with $r \in S$ and $k \in \overline{S}$, then the unit capacity arcs we just added contribute a capacity of $|S|$ to the cut $\delta^+(S)$. Moreover, the other arcs in $\delta(S)$ have a capacity $\sum_{e \notin E(S)} \overline{x}_e$ since our definition of capacities double counts $\overline{x}_e$ for each edge $e = \{p, q\}$ with both of its endpoints in $\overline{S}$—this accounts for the factor of $1/2$ in the definition of the arc capacities. Therefore, the minimum capacity cut in the kth maximum flow problem minimizes $|S| + \sum_{e \notin E(S)} \overline{x}_e$ over all cuts $\delta(S)$ separating the root node r from node k. Consequently, by solving $|V| - 1$ maximum flow problems, one for each node $k \neq r$, we solve the linear programming separation problem.

3.3 A Linear Programming Bound

In Section 3.2, we saw that $P_{mcut} = P_{sub}$ (as well as several other equivalent polyhedra) are the convex hull of incidence vectors of spanning trees. We also saw, through an example, that P_{cut} might be a larger polyhedron. Therefore, for any vector $w = (w_e)$ of objective coefficients, the optimal value of the linear program $z^{LP}_{cut} = \min_{x \in P_{cut}} wx$ will generally be less than the optimal value of of the linear program $z^{LP}_{mcut} = \min_{x \in P_{mcut}} wx$. That is, $z^{LP}_{mcut}/z^{LP}_{cut} = r \geq 1$. How large can the ratio r become? In our example in Figure 9, $r = 2/1.5 = 4/3$.

Recall that $P_{cut} = \{x \in R^{|E|} : \sum_{e \in E} x_e = n - 1$, and $\sum_{e \in \delta(S)} x_e \geq 1$

for all nonempty node sets $S \subset V\}$ and that $P_{mcut} = \{x \in R^{|E|} : 0 \leq x \leq 1, \sum_{e \in E} x_e = n-1$, and $\sum_{e \in \delta(C_0, C_1, \ldots, C_k)} x_e \geq k$ for all node partitions $C_0, C_1, \ldots, C_k$ of $V\}$. Throughout this analysis, we assume $w \geq 0$.

To obtain a bound on $z^{LP}_{mcut}/z^{LP}_{cut}$, let us first consider the polyhedra P_{mcut} and P_{cut} without the cardinality constraint $\sum_{e \in E} x_e = n - 1$ and the upper bound constraints $x_e \leq 1$ imposed upon the edges e. Let P^+_{mcut} and P^+_{cut} denote these polyhedra. Note that any integer point in either of these polyhedra corresponds to a "supertree", that is, a graph that contains a spanning tree (i.e., a spanning tree plus additional edges which can, if any $x_e > 1$, duplicate some edges). Let $z^{LP}_{mcut+} = \min_{x \in P^+_{mcut}} wx$ and $z^{LP}_{cut+} = \min_{x \in P^+_{cut}} wx$ be the optimal objective values of the linear programs with objective function coefficients w over these polyhedra. We wish to establish a bound on the value of these linear programs.

Let $\bar{x}$ be any point in P^+_{cut} and let $\delta(C_0, C_1, \ldots, C_k)$ be any multicut. Suppose we add the constraints $\sum_{e \in \delta(C_q, V \setminus C_q)} x_e \geq 1$ for all $q = 0, 1, \ldots, k$. In doing so, we include each edge in $\delta(C_0, C_1, \ldots, C_k)$ twice, and so the resulting inequality is

$$2 \sum_{e \in \delta(C_0, C_1, \ldots, C_k)} x_e \geq k + 1.$$

or, equivalently, $\frac{2k}{k+1} \sum_{e \in \delta(C_0, C_1, \ldots, C_k)} x_e \geq k$.

Note that since $k + 1 \leq |V|$, $1 - 1/|V| \geq 1 - 1/(k + 1) = k/(k + 1)$. Consequently, the point $\bar{x}$ also satisfies the following inequality

$$2(1 - 1/|V|) \sum_{e \in \delta(C_0, C_1, \ldots, C_k)} x_e \geq k.$$

Therefore, the point $\bar{z} = 2(1 - 1/|V|)\bar{x}$ belongs to P^+_{mcut}. Note that this point has an objective value of $w[2(1 - 1/|V|)]\bar{x} = w\bar{z}$. Thus, for *any* point $x \in P^+_{cut}$, the point $z = 2(1 - 1/|V|)x$ with objective value wz belongs to P^+_{mcut}, that is, $[2(1 - 1/|V|)]wx = wz \geq z^{LP}_{mcut+}$. Since this inequality is valid for all points $x \in P^+_{cut}$, including any optimal point, it implies the following bound:

Proposition 3.10 $z^{LP}_{mcut+}/z^{LP}_{cut+} \leq 2(1 - 1/|V|)$.

This result shows that, in general, the objective value for any linear program defined over P^+_{mcut} is no more than twice the objective value of a linear

program with the same objective function defined over the polyhedron P^+_{cut}. If the underlying graph is a cycle with $|V|$ edges, and if all the edge costs $w_e = +1$, then the optimal solution to $\min\{wx : e \in P^+_{mcut+}\}$ sets $x_e = 1$ for all but one edge and the optimal solution to $\min\{wx : e \in P^+_{cut+}\}$ sets $x_e = 1/2$ for all edges. Therefore, $z^{LP}_{mcut+}/z^{LP}_{cut+} = (|V|-1)/(|V|/2) = 2(1-1/|V|)$ achieves the bound in this proposition.

Now consider the problems $z^{LP}_{mcut} = \min_{x \in P_{mcut}} wx$ and $z^{LP}_{cut} = \min_{x \in P_{cut}} wx$. We first make the following observations:

(1) In the multicut polyhedron P_{mcut}, for any edge $\bar{e}$, the upper bound constraints $x_{\bar{e}} \leq 1$ are redundant since the constraint $\sum_{e \in E} x_e = n - 1$ and the multicut constraint $\sum_{e \neq \bar{e}} x_e \geq n - 2$ (here C_0 contains just the end nodes of edge $\bar{e}$ and all the other C_k are singletons) imply that $x_{\bar{e}} \leq 1$.

(2) If $w_e \geq 0$, then the cardinality constraint $\sum_{e \in E} x_e = n - 1$ is redundant in the linear program $z^{LP}_{mcut} = \min_{x \in P_{mcut}} wx$ in the sense that the problem without the cardinality constraint always has a solution satisfying this constraint. This result follows from the second proof of Theorem 3.4 which shows that since $w_e \geq 0$ for all $e \in E$, the dual linear program has an optimal solution in which the dual variable on the constraint $\sum_{e \in E} x_e = \sum_{\delta(\{1\},\ldots,\{n\})} x_e = n - 1$ is nonnegative.

We can now establish the following result.

Proposition 3.11 $z^{LP}_{mcut}/z^{LP}_{cut} \leq 2(1 - 1/|V|)$.

Proof. Since the polyhedron P_{cut} contains one more constraint than the polyhedron P^+_{cut}, $z^{LP}_{cut} \geq z^{LP}_{cut+}$ and as we have just seen, since $w \geq 0$, $z^{LP}_{mcut} = z^{LP}_{mcut+}$. Therefore, Proposition 3.10 shows that

$$[2(1 - 1/|V|)]z^{LP}_{cut} \geq [2(1 - 1/|V|)]z^{LP}_{cut+} \geq z^{LP}_{mcut+} = z^{LP}_{mcut}.$$

■

Observe that since the polyhedron P_{mcut} is integral, $z^{LP}_{mcut} = Z$, the optimal value of the spanning tree problem. Therefore, Proposition 3.11 bounds the ratio of the optimal integer programming value to the optimal objective value of the linear programming relaxation of the cut formulation. In Section 6.3, we consider a generalization of this bound. Using a deeper result than we have used in this analysis (known as the parsimonious property), we are able to show that the bound of $2(1 - 1/|V|)$ applies to Steiner trees as well as spanning trees.

4 Rooted Subtrees of a Tree

In Section 2, we considered the core tree problem defined on a general network. We now consider the core problem encountered in packing trees within a tree: the rooted subtree problem. Given a tree T with a root node r and a weight w_v on each node v of T, we wish to find a subtree T^* of T containing the root that has the largest possible total weight $\sum_{j \in T^*} w_v$. We permit T^* to be the empty tree (with zero weight). In Section 4.2 we consider an extension of this model by introducing capacity restrictions of the tree T^*.

4.1 Basic Model

We begin by setting some notation. Let $p(v)$, the predecessor of v, be the first node $u \neq v$ on the unique path in T connecting node v and the root r, and let $S(v)$ be the immediate successors of node v; that is, all nodes u with $p(u) = v$. For any node v of T, let $T(v)$ denote the subtree of T rooted at node v; that is $T(v)$ is the tree formed if we cut T by removing the edge {p(v),v} just above node v.

Dynamic Programming Solution

The solution to this problem illustrates the type of dynamic programming procedure that solves many problems defined on a tree. For any node v of T, let $H(v)$ denote the optimal solution of the rooted subtree problem defined on the tree $T(v)$ with node v as the root. If v is a leaf node of $T, H(v) = \max\{0, w_v\}$ since the only two rooted subtrees of $T(v)$ are the single node $\{v\}$ and the empty tree. The dynamic programming algorithm moves "up the tree" from the leaf nodes to the root. Suppose that we have computed $H(u)$ for all successors of node v; then we can determine $H(v)$ using the following recursion:

$$H(v) = \max\{0, w_v + \sum_{u \in S(v)} H(u)\}. \tag{4.1}$$

This recursion accounts for two cases: the optimal subtree of $T(v)$ rooted at node v is either (a) empty, or (b) contains node v. In the latter case, the tree also contains (the possibly empty) optimal rooted subtree of each node u in $S(v)$. Note that since each node u, except the root, is contained in exactly

one subset $S(v)$, this recursion is very efficient: it requires one addition and one comparison for each node of T. After moving up the tree to its root and finding $H(r)$, the optimal value of subtree problem defined over the entire tree T, we can determine an optimal rooted subtree T^* by deleting from T all subtrees $T(u)$ with $H(u) = 0$.

Example 4.1. For the example problem shown in Figure 12 with root $r = 1$, we start by computing $H(4) = 4, H(6) = 0, H(7) = 2, H(8) = 4, H(10) = 0$, and $H(11) = 3$ for the leaf nodes of the tree. We then find that $H(9) = \max\{0, -5+0+3\} = 0, H(5) = \max\{0, -1+4+0\} = 3, H(2) = \max\{0, -5+4+3\} = 2, H(3) = \max\{0, -1+0+2\} = 1$, and finally $H(1) = \max\{0, 2+2+1\} = 5$. Since $H(9) = H(6) = 0$, as shown in Figure 12(b), the optimal rooted tree does not contain the subtrees rooted at these nodes.

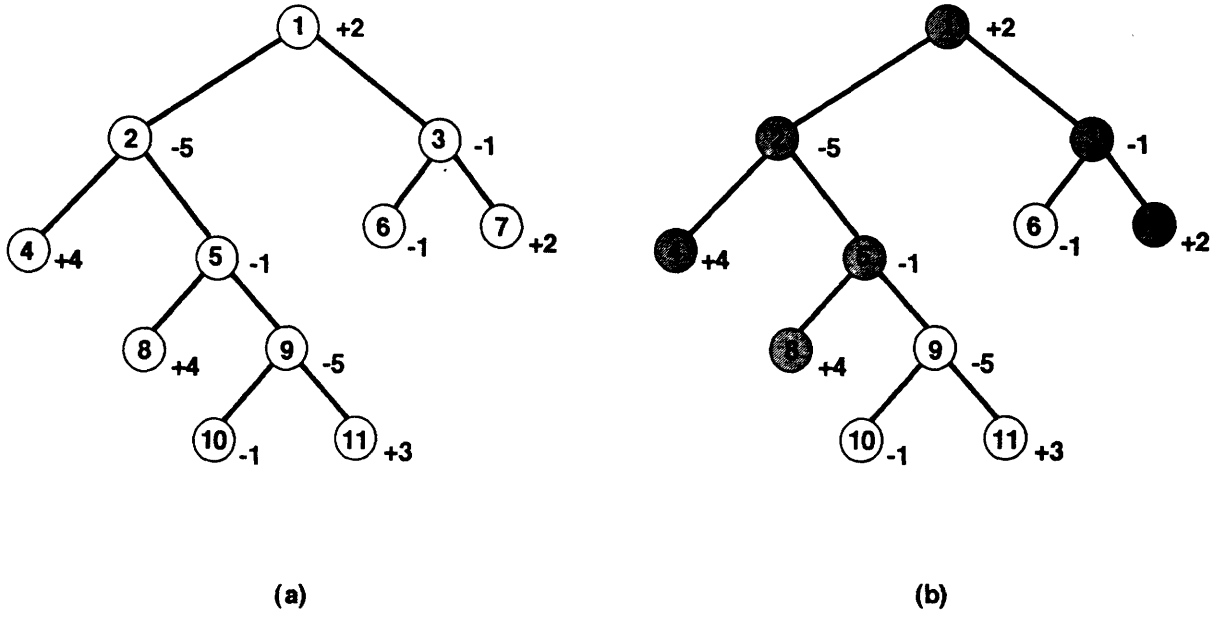


Figure 12: (a) Given Tree with Node Weights w_e; (b) Optimal Rooted Subtree (Shaded Nodes)

Variants and enhancements of this recursion apply to many other problems defined on trees. We will examine one such example in Section 4.2

where we consider the addition of capacity constraints to the subtree of a tree problem. In later sections, we consider several other similar dynamic programming algorithms.

Polyhedral Description

Let x_v be a zero-one variable indicating whether ($x_v = 1$) or not ($x_v = 0$) we include node v in a rooted subtree of T, and let X denote the set of the incidence vectors $x = (x_v)$ of subtrees rooted at node r (more precisely, X is the incidence vectors of nodes in a subtree rooted at node r—for convenience we will refer to this set as the set of subtrees rooted at node r). Note that points in X satisfy the following inequalities

$$0 \leq x_r \leq 1, 0 \leq x_v \leq x_{p(v)} \text{ for all nodes } v \neq r \text{ of } T. \tag{4.2}$$

Since every point in X satisfies these inequalities, so does the convex hull, $conv(X)$, of X, that is, $\{x : 0 \leq x_r \leq 1, 0 \leq x_v \leq x_{p(v)}$ for all nodes $v \neq r$ of $T\} \supseteq conv(X)$. We will show that the inequalities in (4.2) actually completely describe r-rooted subtrees of T in the sense that the convex hull of X equals the set of solutions to (4.2). That is, we will establish the following result:

Theorem 4.1 *The set of solutions to the linear inequality system (4.2) is the convex hull of the 0-1 incidence vectors of subtrees of T rooted at node r.*

This theorem shows that the extreme points of the polyhedron $P = \{x \in R^{|V|} : 0 \leq x_r \leq 1, 0 \leq x_v \leq x_{p(v)}$ for all nodes $v \neq r$ of $T\}$ are the 0-1 incidence vectors of subtrees of T rooted at node r. Notice that $X = P \cap Z^{|V|}$, that is, every integer point in P is the incidence vector of a subtree rooted at node r and so we will establish the theorem if we show that every extreme point of the polyhedron P is integer valued.

We will prove Theorem 4.1 using three different arguments that underlie many results in the fields of combinatorial optimization and polyhedral combinatorics: a network flow argument, a dynamic programming argument, and an argument based upon the nature of optimal solutions to the optimization problem min $\{wx : x \in X\}$ as we vary the weights w. All three of these arguments rely on basic results from linear programming.

Approach 1 (Network Flow Argument)

Consider the (primal) linear programming problem $\max\{wx : 0 \leq x_r \leq 1, 0 \leq x_v \leq x_{p(v)}$ for all nodes $v \neq r$ of $T\}$ for any choice w of integer node weights. Since this problem has one inequality (in addition to the nonnegativity restriction) for each node v of T, its linear programming dual problem $\min\{y_r : y_v - \sum_{u \in S(v)} y_u \geq w_v$ and $y_v \geq 0$ for all nodes v of $T\}$ has one dual variable y_v for each node of v. Note that the dual problem is a network flow problem since each variable y_q for $q \neq r$ appears in exactly two constraints: in constraint $v = q$ with a coefficient of $+1$ and in the constraint $v = p(q)$ with a coefficient of -1. Since node r has no predecessor, it appears in only the constraint $v = r$ with a coefficient of $+1$. The theory of network flows shows that whenever the cost data of any network flow problem are integral, its dual linear program always has an integer optimal solution (see the discussion at the end of Section 1). In this case, since the network flow problem $\min\{y_r : y_v - \sum_{u \in S(v)} y_u \geq w_v$ and $y_v \geq 0\}$ has integer objective function coefficients (zeros and the single $+1$ coefficient for y_r), for *any* choice of integer weights w, the dual problem $\max\{wx : 0 \leq x_r \leq 1, 0 \leq x_v \leq x_{p(v)}$ for all nodes $v \neq r$ of $T\} = \max\{wx : x \in P\}$ has an integer optimal solution. But then the theory of linear programming theory shows that every extreme point of P is integer valued, and so by our previous observation, the extreme points are the incidence vectors of subtrees rooted at node r.

Approach 2 (Dynamic Programming Argument)

This argument is similar to the linear programming duality argument that we gave in the last section when we studied the minimum spanning tree problem, though it uses information provided by the dynamic programming solution procedure instead of the greedy procedure to set values for the linear programming dual variables. Consider the (primal) linear programming problem $\max\{wx : 0 \leq x_r \leq 1, 0 \leq x_v \leq x_{p(v)}$ for all nodes $v \neq r$ of $T\}$. As we noted in the last subsection, its linear programming dual problem is $\min\{y_r : y_v - \sum_{u \in S(v)} y_u \geq w_v$ and $y_v \geq 0$ for all nodes v of $T\}$.

Let T^* be the optimal rooted subtree of T determined by the dynamic programming recursion (4.1) and let $x_v = 1$ if node v belongs to T^* and let $x_v = 0$ otherwise. This solution is feasible in the primal linear program

and its objective value is $H(r)$, the optimal objective value determined by the dynamic programming recursion. Since the values of $H(v)$, for v in T, satisfy the recursion (4.1), $H(v) \geq w_v + \sum_{u \in S(v)} H(u)$. Therefore, the choice $y_v = H(v)$, for all $v \in V$, of the variables in the dual linear program is feasible. Moreover, since the dual objective value is $y_r = H(r)$, we have shown that for every choice of objective function coefficients w for the primal problem, this problem has an integer optimal solution (since its objective value equals the objective value of some dual feasible solution). But then linear programming theory implies that the extreme points of the polyhedron $P = \{x : 0 \leq x_r \leq 1, 0 \leq x_v \leq x_{p(v)}$ for all nodes $v \neq r$ of $T\}$ are the 0-1 incidence vectors of subtrees of T rooted at node r.

Approach 3 (Optimal Inequality Argument)

This proof uses an argument from the field of polyhedral combinatorics. Let w be any n-vector of weight coefficients and consider the optimization problem max $\{wy : y \in Y\}$ for any finite set Y. Let $Q = \{a_j x \leq b_j$ for $j = 1, 2, \ldots, m\}$ be any bounded polyhedron that contains Y. Suppose that we can show that for any choice of $w \neq 0$, the set of optimal solutions to the problem max $\{wy : y \in Y\}$ all lie on the same inequality $a_j y \leq b_j$ of the polyhedron Q. We refer to any such inequality (for a given choice of w) as an *optimal inequality*. Note that for a particular choice of w, the polyhedron P might contain more than one optimal inequality and as we vary w, we will find different optimal inequalities. We might distinguish between two types of inequalities: an inequality $a_j y \leq b_j$ is *binding* if it is satisfied as an equality by all points in Y and is *nonbinding* otherwise.

Nonbinding optimal inequalities are useful for the following reason: suppose w defines a *facet* of the convex hull $conv(Y)$ of Y in the sense that for some constant w_0, $Y \subseteq \{y : wy \leq w_0\}$ and the set of solutions of the system $\{y : y \in Y$ and $wy = w_0\}$ has dimension one less than the dimension of $conv(Y)$. Then the set of optimal solutions to the optimization problem max $\{wy : y \in Y\}$ are just the points of Y on the facet. In this case, any optimal inequality $a_j y \leq b_j$ contains all the points on the facet. This result implies that if the polyhedron Q contains a nonbinding optimal inequality for every choice of the weight vector $w \neq 0$, then the polyhedron is the convex hull of Y.

To utilize this proof technique, we set $Y = X$ and $Q = P$, with X and P as defined previously. Note that in this case every inequality $x_v \leq x_{p(v)}$ in the definition of P is nonbinding since we can choose a rooted feasible tree with $x_v = 0$ and $x_{p(v)} = 1$. The inequality $x_r \leq 1$ is nonbinding because the zero vector is a feasible rooted tree and the inequalities $0 \leq x_v$ are nonbinding since the tree with each $x_v = 1$ is feasible. Since each defining inequality of P is nonbinding, to use the optimal inequality argument, we wish to show that for every choice of $w \neq 0$, the polyhedron P contains an optimal inequality.

Let $z = \max\{wx : x \in X\}$ with $w \neq 0$. Since the zero vector is feasible, $z \geq 0$. If $z > 0$, then all optimal solutions satisfy the condition $x_r = 1$ and so $x_r \leq 1$ is an optimal inequality. So suppose that $z = 0$. Note that since we are assuming that the weight vector $w \neq 0$, some component of w must be negative. Otherwise, setting $x_v = 1$ for all v gives a solution with $z > 0$. So we suppose $w_v < 0$ for some node v of T.

If $x_v = 0$ in every optimal solution to the problem, then $x_v \geq 0$ is an optimal inequality. So suppose that T^* is an optimal subtree solution containing node v. If the weight of the subtree $T(v) \cap T^*$ rooted at node v is negative, then by eliminating $T(v) \cap T^*$ from T^*, we would obtain a rooted tree with a weight that exceeds 0, contrary to our assumption that $z = 0$ (this situation corresponds to the fact that $H(v) \geq 0$ in the dynamic programming solution to the problem). Therefore, since $w_v < 0$, at least one successor u of node v must satisfy the property that the total weight of the subtree $T(u) \cap T^*$ rooted at node u is positive (that is $H(u) > 0$ in the dynamic program). We claim that $x_u \leq x_v$ is an optimal inequality. If $x_v = 0$ in any optimal solution, then $x_u = x_v = 0$. So suppose we have a optimal tree with $x_v = 1$. If $x_u = 0$, we could add $T(u) \cap T^*$ to this optimal tree and obtain a feasible solution with $z > 0$, again contrary to our assumption that $z = 0$. Therefore, $x_u = 1$ and so $x_u \leq x_v$ is an optimal inequality.

These arguments show that for any choice of $w \neq 0$, one of the inequalities in the system (4.2) is a nonbinding optimal inequality to the problem $\max\{wx : x \in X\}$, which implies the conclusion of the theorem. ∎

4.2 Constrained Subtrees

In some application contexts we are not free to choose every rooted subtree as a possible solution to the rooted subtree of a tree problem. For example, the root node might correspond to a concentrator in a telecommunication system

that is serving demands d_v at the nodes v of the tree. If the concentrator has a limited throughput capacity C, then the node incidence vector $x = (x_v)$ of any feasible rooted tree must satisfy the following capacity constraint:

$$\sum_{v \in V(T)} d_v x_v \leq C. \tag{4.3}$$

Recall that $V(T)$ denotes the set of nodes in the tree T.

We assume that each demand d_v is a positive integer that does not exceed the capacity C. We refer to this problem as the capacitated (rooted) subtree of a tree problem.

As an important special case, each node v has a demand $d_v = 1$ and so the capacity constraint becomes a cardinality constraint

$$\sum_{v \in V(T)} x_v \leq K \tag{4.4}$$

stating that the chosen tree can contain at most $K = C$ nodes. We refer to this version of the problem as the cardinality constrained rooted subtree of a tree problem.

A Solution Procedure

To find an optimal capacitated subtree of a tree, we can once again use a dynamic programming algorithm. Let $H(u, q)$ denote the optimal objective value of an optimal subtree for the constrained subtree of a tree problem defined by the tree $T(u)$ with node u as the root and with the integer $q \leq C$ as the capacity. $H(r, C)$ is the optimal objective value for the original problem.

We can solve the original problem by again working from the leaf nodes toward the root using a dynamic programming recursion:

$$H(v, q) = \max\{0, w_v + \max_{\{q_u : \sum_{u \in S(v)} q_u \leq q - d_v\}} \sum_{u \in S(v)} H(u, q_u)\}. \tag{4.5}$$

We initiate the recursion by setting $H(u, q) = \max\{w_u, 0\}$ for any leaf node with $d_u \leq q$ and $H(u, q) = 0$ for any leaf node with $d_u > q$.

This recursion says that the optimal solution on the tree $T(v)$ either does not use node v and so has value zero or uses node v, consuming d_v units of the capacity and leaving $q - d_v$ units of capacity to be shared by the subtrees rooted on the successor nodes $S(v)$ of node v.

Note that for the cardinality version of the problem, this recursion is particularly simple since its says that if we include node v in the optimal subtree of $T(u)$, then at most $q - 1$ nodes are available for distribution to the subtrees on node u's successor nodes.

If we order the successors of each node, it is easy to implement the basic dynamic programming recursion using $O(nC)$ computations (that is, the number of computations is a polynomial with a constant times nC as the leading term). For the special cardinality version of the problem, this implementation requires $O(n^2)$ computations. Let $u_1, ...u_r$ be the successors of node v. We will find the optimal value of $H(v, q)$ by building up the solution from the optimal solution over the subtree $T(u_1)$, then $T(u_1)$ and $T(u_2)$, then $T(u_1), T(u_2)$, and $T(u_3)$, and so forth until we consider all the successors. To do so, we let $G_j(v, t)$ denote the objective value of an optimal forest containing rooted subtrees from the trees $T(u_1), T(u_2), \ldots, T(u_j)$, given that the nodes in the solution contain a total demand of at most t. For each index $j = 1, 2, \ldots, r$, let $G_j(v, 0) = 0$. Then for all $t = 1, 2, \ldots, C$, $G_1(v, t) = H(u_1, t)$ and for each $j = 2, 3, \ldots, r$, $G_j(v, t) = \max_{0 \le s \le t}\{G_{j-1}(v, s) + H(u_j, t - s)\}$. Finally, we find $H(v, q) = \max\{0, w_v + G_r(v, q - d_u)\}$.

Polyhedral Considerations

Let X^C and X^K denote the set of feasible solutions to the capacitated rooted subtree of a tree problem, that is, the rooted subtree of a tree problem with the additional constraints (4.3) or (4.4). Let $P^C \supseteq X^C$ denote the polyhedra defined by the constraints (4.2) and (4.3) and let $P^K \supseteq X^K$ denote the polyhedra defined by the constraints (4.2) and (4.4).

As shown by the following example, unlike the (uncapacitated) subtree of a tree problem, in general the polyhedra P^C and P^K are not integral.

Example 4.2. Consider the cardinality constrained problem with $K = 4$ for the example shown in Figure 12. In this case the optimal tree T^* contains nodes 1, 3, and 7 and has a weight 3. The fractional solution $\hat{x}_1 = 1$, $\hat{x}_2 = \hat{x}_3 = \hat{x}_4 = \hat{x}_5 = \hat{x}_7 = \hat{x}_8 = 1/2$, and $\hat{x}_6 = \hat{x}_9 = \hat{x}_{10} = \hat{x}_{11} = 0$ satisfies all the constraints of the system (4.3) as well as the inequality (4.4) and

so lies in P^K. The point $\hat{x}$ is the unique solution to the linear program $\max\{\sum_{v \in V(T)} w_v x_v : x = (x_v) \in P^K\}$ and so by linear programming theory it is a fractional extreme point of the polyhedra P^K. ∎

In order to obtain a polyhedral description of $conv(X^C)$ or of $conv(X^K)$, we need to add additional valid inequalities to P^C and P^K. Finding enough valid inequalities to completely describe these convex hulls appears to be quite difficult. To illustrate how complicated the convex hulls are, we will illustrate two sets of valid inequalities.

Let us call a subtree T' of T rooted at node r a *cover* if $\sum_{v \in V(T')} d_v$ exceeds C. Note that for the cardinality constrained problem, a tree cover is just a subtree rooted at node r with at least $K + 1$ nodes. The cover condition implies that we cannot include all the nodes of T' in any feasible rooted subtree of T.

To eliminate this possibility, we can use the following *tree cover inequality*:

$$\sum_{v \in T'} (x_{p(v)} - x_v) \geq x_r.$$

Proposition 4.2 *Feasible points in X^C satisfy every tree cover inequality.*

Proof. Consider any tree cover inequality. The null tree $\overline{x} = 0$ clearly satisfies it. For any other feasible solution $\overline{x}$, $\overline{x}_r = 1$. Note that the constraints $x_v \leq x_{p(v)}$ in the system (4,2) imply that every term on the left-hand side of the cover inequality is nonnegative. Therefore, if the solution $\overline{x}$ to the problem violates this inequality, then $\overline{x}_{p(v)} - \overline{x}_v = 0$ for all $v \in V(T')$. But then since $\overline{x}_r = 1$, $\overline{x}_v = 1$ for all the nodes $v \in V(T')$, contradicting the fact that T' is a cover. ∎

To illustrate the tree cover inequalities, consider the cardinality constrained tree problem of Example 4.2 once again (with $K = 4$). In this case, let T' contain the five nodes $1, 2, 4, 5,$ and 8. The tree cover inequality for T' is:

$$(x_5 - x_8) + (x_2 - x_4) + (x_2 - x_5) + (x_1 - x_2) \geq x_1$$

or

$$x_2 \geq x_4 + x_8.$$

Since $\hat{x}_2 = \hat{x}_4 = \hat{x}_8 = 1/2$, the fractional solution $\hat{x}$ from Example 4.2 does not satisfy this inequality. Therefore, by adding the tree cover inequalities

to the polyhedron P^K, we obtain a better representation of the cardinality constrained polyhedron $conv(P^K)$.

We obtain the second set of inequalities by considering another special class of subtrees rooted at node r. Given a subtree T' containing the root node, let $L(T')$ denote its leaf nodes, and for any node set $S \subset V(T)$, let $Cl(S)$, the *closure* of S, be the tree determined by the union of the paths from each node $v \in S$ to the root.

Proposition 4.3 *Let T' be a subtree of T rooted at node r. If for some positive integer q, the closure $Cl(S)$ is a tree cover of the tree T for all node sets $S \subset L(T')$ with $|S| = q$, then any feasible solution of X^C satisfies the following leaf cover inequality:*

$$\sum_{u \in L(T')} x_u \leq (q-1)x_r.$$

Proof. This inequality is clearly valid for $\overline{x} = 0$. If $\sum_{u \in L(T')} \overline{x}_u \geq q\overline{x}_r$ and $\overline{x}_r = 1$, then the subtree of T' defined by the nodes u with $\overline{x}_u = 1$ would contain a tree cover and so be infeasible. So every feasible solution satisfies every leaf inequality. ∎

Example 4.3. Consider the cardinality constrained subtree problem with $K = 5$ on the tree T shown in Figure 13. In this case, we can obtain four tree covers by deleting any single leaf node from T. The tree cover inequalities for these subtrees (once we have collected terms) are:

$$\begin{aligned} x_1 + x_2 &\geq x_4 + x_5 + x_6 \\ x_1 + x_2 &\geq x_4 + x_5 + x_7 \\ x_1 + x_3 &\geq x_3 + x_6 + x_7 \\ x_1 + x_3 &\geq x_4 + x_6 + x_7 \end{aligned}$$

Let $T' = T$ and $q = 3$. Then the tree T' satisfies the conditions of the leaf inequality and so the following leaf inequality is valid:

$$x_4 + x_5 + x_6 + x_7 \leq 2x_1.$$

The fractional solution shown in Figure 13 satisfies all four tree cover inequalities as well as the inequalities in the system (4.2). However, it violates this leaf inequality. ∎

Let P^K_{TC} and $P^K_{TC,L}$ denote the polyhedra defined by adding the tree cover inequalities and both the tree cover and leaf inequalities to P^K. This example and Example 4.2 show that in general,

$$conv(X^K) \subset P^K_{TC,L} \subset P^K_{TC} \subset P^K.$$

That is, in general, as in this case, by adding the tree cover inequalities and then the leaf inequalities to P^K, we obtain better polyhedral approximations to $conv(X^K)$.

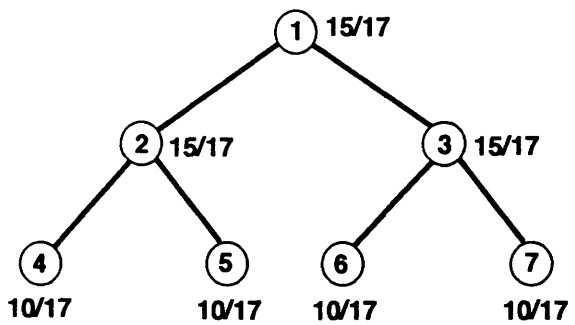


Figure 13: Fractional Values x_v Violating a Leaf Inequality

As this discussion has shown, the addition of a single inequality, even a very simple cardinality constraint, can add considerably to the complexity of an integer polyhedron.

5 Polynomially Solvable Extensions/Variations

In the last two sections, we considered two core tree optimization problems—the minimum spanning tree problem and the rooted subtree of a tree problem. For each of these problems, we were able to develop an algorithm (the greedy algorithm and a dynamic programming algorithm) that solves the problem using a number of computations that is polynomial in the problem's size (as measured by the number of nodes and edges in the underlying graph). In this section, we consider two other polynomially-solvable versions of tree problems; both are variations of the minimum spanning tree problem. In one problem, the degree-constrained spanning tree problem, we impose a degree constraint on one of the nodes and in the other problem, defined over a directed graph, we seek a directed version of a minimum spanning tree, a so-called optimal branching or optimal arborescence. We not only describe polynomial algorithms for these problems, but also show that we can describe the convex hull of incidence vectors of feasible solutions of these problems by using minor variants of the formulations we have already studied for the minimum spanning tree problem.

5.1 Degree-Constrained Spanning Trees

Suppose we wish to find a minimum spanning tree in a graph G, but require that the tree satisfies the property that the degree of a particular node r, which we will call the root node, is fixed at value k. That is, the root node must have k incident edges. We call any such tree a *degree-constrained minimum spanning tree.* In this section, we show how to solve this problem efficiently for all values of k and, for any fixed k, we give a polyhedral description of the incidence vectors of degree-constrained spanning trees.

To solve the degree-constrained minimum spanning tree, we consider a parametric minimum spanning tree problem with edge weights chosen as follows: the weight of every edge $e = \{r, j\}$ incident to the root node is $w_e + \theta$ for some scalar parameter θ, and the weight of any edge e not incident to the root node is the constant w_e. Note that if θ is sufficiently large, the solution to the unconstrained minimum spanning tree problem will be a spanning tree T_0 in G containing the fewest possible number k_{min} of edges

incident to node r. (Why?) As we decrease θ, the edges incident to the root node become more attractive. For each edge $e = \{r, j\}$ not in T^0, let P_j denote the path in T^0 that connects node j to the root node. As we decrease θ, for the first time at some point $\theta = \theta_j^-$ (that is, θ_j minus any positive amount), the edge $e = \{r, j\}$ has a weight smaller than one of the edges on the path P_j. Among all choices of node j, we choose one with a maximum value of θ_j and add edge $e = \{r, j\}$ to T_0, dropping the edge from P_j whose weight is larger than $w_e + \theta_j^-$. Note that since we add the same amount θ to the cost of every edge incident to the root node, the edge we drop will not be incident to the root. Therefore, this operation gives us a new tree T_1 with $k_{min} + 1$ edges incident to the root node. We then repeat this step using T_1 in place of T_0 (and so with new paths P_j). By continuing in this way, for each $i = 1, 2, \ldots$, we obtain a spanning tree T_i with $k_{min} + i$ edges until the tree T_i contains every edge $\{r, j\}$ in G emanating from the root node. In Theorem 5, we show that each intermediate tree T_i is a minimum spanning tree for the value of θ at which we converted the tree T_{i-1} into the tree T_i.

Suppose we maintain and update two labels for each node: (i) a precedence label that indicates for each node j the next node q on the unique path in the tree T_i connecting node j to the root node, and (ii) a second label l that denotes the maximum weight of the edges on P_j, not including the edge incident to the root. With these labels, determining the next edge $e = \{r, j\}$ to add to the tree at each step and finding the edge on the path P_j to drop from the tree requires at most n operations. Since updating the node labels when we add edge e to T_i requires at most n operations as well (the node labels change only for the nodes on P_j), this algorithm requires $O(n^2)$ operations[5] plus the amount of work needed to find the initial spanning tree T_0.

We note that as a byproduct, the algorithm we have just described provides a polyhedral characterization of degree-constrained minimal spanning trees. Let $Q = \{x \geq 0 : \sum_{e \in E} x_e = n - 1$, and $\sum_{e \in E(S)} x_e \leq |S| - 1$ for all $S \neq V\}$ be the polyhedron whose extreme points are incidence vectors of spanning trees (see Section 3).

Theorem 5.1 *Suppose that node r in a given graph has k or more incident arcs. Then the incidence vectors of spanning trees with degree k at the root*

[5]That is, a number of computations that is no more than some polynomial with a lead term of cn^2 for some scalar c.

node r are the extreme points of the polyhedron $P = Q \cap \{x : \sum_{e \in \delta(r)} x_e = k\}$.

Proof. From the theory of linear programming this result is true if the optimization problem $\min\{wx : x \in P\}$ has an integer solution for every choice of the weight vector w. As we have argued in Section 3, for any value of θ, the optimal objective value of the problem $\min\{wx + \theta(\sum_{e \in \delta(r)} x_e - k) : x \in Q\}$ is a lower bound on the optimal objective value of this problem . At some point in the course of the algorithm we have just described at a value $\theta = \theta^*$, the solution x^* to the problem $\min\{wx + \theta^*(\sum_{e \in \delta(r)} x_e - k) : x \in Q\}$ is a spanning tree with k nodes incident to node r. But then since the lower bound $wx^* + \theta^*(\sum_{e \in \delta(r)} x_e^* - k)$ equals the cost cx^* of the vector x^* and x^* is feasible to the problem $\min\{wx : x \in P\}$, x^* solves this problem. Therefore, this problem has an integer solution for every choice of the cost vector c and so P has integer extreme points. ∎

Note that for every value of k the polyhedron $P = Q \cap \{x : \sum_{e \in \delta(r)} x_e = k\}$ is a slice through the polyhedron Q along the hyperplane $\{x : \sum_{e \in \delta(r)} x_e = k\}$. The last theorem shows that every such slice has integer extreme points.

5.2 Optimal Branchings

Let $D = (V, A)$ be a directed graph with a designated root node r. A *branching* is a directed subgraph (V, B) that satisfies two properties: (i) it is a tree if we ignore arc orientations, and (ii) it contains a directed path from the root to each node $v \neq r$. It is easy to see that we obtain an equivalent definition if we replace (ii) by the following condition: (ii′) the network has exactly one arc directed into each node $v \neq r$. Given a weight w_e on each arc e of D, we would like to solve the optimal branching problem of finding a branching with the smallest possible total arc weight.

For convenience, we usually refer to a branching by its arc set B.

Branching Models

In Section 3.2, we introduced the following integer programming packing model of the optimal branching problem. For $e \in A$, we let $y_e = 1$ if e is in the branching, and $y_e = 0$ otherwise.

$$
\begin{array}{rll}
\min & \sum_{e \in A} & w_e y_e \\
\text{subject to} & & \\
\sum_{e \in A(S)} y_e & \leq & |S| - 1 \text{ for all nonempty sets } S \subset V \\
\sum_{e \in \delta^-(v)} y_e & = & 1 \text{ for all } v \in V \setminus (\{r\}) \\
\sum_{e \in \delta^-(r)} y_e & = & 0 \\
y & \geq 0 & \text{and integer.}
\end{array}
$$

Notice that the equality constraints in this formulation remain the same if we replace the constraint $\sum_{e \in \delta^-(r)} y_e = 0$ by $\sum_{e \in A} y_e = n - 1$. For simplicity in the following discussion, we assume that we have eliminated all the arcs directed into the root node r and so we can delete the constraint $\sum_{e \in \delta^-(r)} y_e = 0$.

Let P be the polyhedron defined by this system if we ignore the integrality restrictions on the variables y. The results in Section 3 imply that if the digraph is symmetric in the sense that $(j,i) \in A$ whenever $(i,j) \in A$ and $w_{ij} = w_{ji}$, then the linear programming relaxation of this problem always has an integer optimal solution. We also showed that the greedy algorithm for the (undirected) minimum spanning tree problem solves this special case. In this section, we establish a more general polyhedral result: the extreme points of P are the incidence vectors of branchings and so the linear programming relaxation of the problem always has an integer optimal solution, even for the asymmetric case. We also develop an algorithm for finding an optimal branching.

Rather than work with the polyhedron P directly, we will consider an equivalent cutset formulation. Let $Q = \{y \in R^{|A|} : y \geq 0, \sum_{e \in \delta^-(v)} y_e = 1$ for all $v \in V \setminus (\{r\})$, and $\sum_{e \in \delta^+(S)} y_e \geq 1$ for all nonempty sets S with $r \in S \subset V\}$. Our discussion of directed models of the minimum spanning tree problem in Section 3 shows that, as formalized by the following proposition, we can formulate the branching model in an equivalent cutset form.

Proposition 5.2 $P = Q$.

In the following discussion, we consider a related class of subgraphs (V, B') of D, called *superbranchings*, that contain one or more paths directed from the root node to every other node (and so contains one or more arcs directed into each node $v \neq r$). Since any superbranching B' contains a directed path to any node $v \neq r$, it contains a directed arc across every cutset, that is, the superbranching and every set $S \subset V$ of nodes containing the root node r satisfy the cutset condition $|\delta^+(S) \cap B'| \geq 1$.

Note that if B' is a superbranching and $\overline{B} \subseteq B'$ is a branching on a subset $\overline{V}$ of the nodes V, then if $\overline{V} \subset V$ we can extend $\overline{B}$ to a branching on a larger subset of nodes by adding any arc from $\delta^+(\overline{V}) \cap B'$ to $\overline{B}$. This observation implies that every superbranching contains a branching.

Finding an Optimal Branching

As we have already noted, any branching rooted at node r satisfies three properties: (i) node r has no incoming arc, (ii) every node $v \neq r$ has exactly one incoming arc, and (iii) every directed cutset $\delta^-(V(C))$, with $r \notin C$, contains at least one arc (or, equivalently, since $P = Q$, if we ignore the orientation of arcs, the branching contains no cycles)[6]. As the first step of an algorithm for solving the branching problem, we will ignore the last condition. The resulting problem is very easy to solve: for each node $v \neq r$, we simply choose a minimum weight arc that is directed into that node. We refer to this solution as a *node greedy solution.* If the node greedy solution NG contains no cycles, then it solves the optimal branching problem; otherwise, this solution contains one or more cycles C with $r \notin C$. Note that, since every node has exactly one incoming arc, the cycle will be directed.

Our next result tells us even though the node greedy solution (typically) does not solve the branching problem, it contains considerable information about an optimal solution—enough to permit us to devise an efficient algorithm for finding an optimal branching.

To simplify our notation, we first transform the weights so that the minimum weight arc e directed into each node $v \neq r$ has weight zero. Suppose we subtract any constant q_v from the weight of all the arcs directed into any node v. Since any branching has exactly one arc directed into each node, the transformed and original problems have exactly the same optimal solutions, since *any* feasible solution to the transformed problem costs q_v less than that

[6]Recall that $V(C)$ denotes the set of nodes in the cycle C.

of the same solution in the original problem. If we choose q_v as the minimum weight of the arcs in $\delta^-(v)$, then the minimum weight arc directed into node v in the transformed digraph has weight zero.

Proposition 5.3 *Suppose that a node greedy solution NG contains a directed cycle C not containing the root node r. Then the optimal branching problem always has an optimal solution with exactly $|C| - 1$ arcs from the cycle C (and so exactly one arc in the directed cutset $\delta^-(V(C))$).*

Proof. Note that by our transformation of arc weights, every arc in C has weight zero.

Let B be any optimal branching. Since B is a branching, the arc set $B^-(C) \equiv \delta^-(V(C)) \cap B$ satisfies the cutset condition $|B^-(C)| \geq 1$. If $|B^-(C)| = 1$, B satisfies the conclusion of the proposition; so, assume $|B^-(C)| > 1$. We will use an induction argument on the size of $|B^-(C)|$. Suppose $|B^-(C)| = k$ and the induction hypothesis (that is, the conclusion of the proposition) is true for all cycles C satisfying the condition $|B^-(C)| \leq k-1$.

Since $|B^-(C)| > 1$, B contains at least two arcs (i,j) and (p,q) directed into the cycle C, that is, with $i, p \notin V(C)$ and $j, q \in V(C)$. By definition, the graph B contains a (unique) directed path from the root node to every other node; therefore, it must contain a path from the root node to node j or node q, say node j, that does not pass through the other node. Let P_j denote this path. Let $B' = C \cup (B \setminus (p,q))$. Note that B' contains a path from the root node r to node q: the path P_j plus the arcs on the path from node j to node q in the cycle C. Therefore, B' contains a path to every node $v \neq r$ and so it is a superbranching. Consequently, it contains a branching $\overline{B}$. Since the arcs in C that we added to the branching B in this construction have zero weight, the weight $w(\overline{B})$ of $\overline{B}$ is no more than the weight $w(B)$ of B and so $\overline{B}$ is also an optimal branching. But since in constructing B', we eliminated the arc (p,q) from B and added the arcs in C, $|\overline{B}^-(C)| \equiv |\delta^-(V(C)) \cap \overline{B}| \leq k-1$. Therefore, the induction hypothesis implies that the problem has an optimal branching B^* containing exactly $|C| - 1$ arcs from $|C|$. ∎

Figure 14 gives an example of a branching problem. In this case, the node greedy solution contains a cycle C_1 on the nodes $4, 5, 6,$ and 7 and so Proposition 5.3 implies that we can find an optimal branching B_1 containing three arcs in this cycle C_1 and exactly one arc directed into this cycle. Moreover, since we have transformed the costs so that each arc in this cycle has zero weight, we are indifferent as to the set of arcs that we choose from the cycle.

The node greedy solution in Figure 14 also contains a cycle C_2 on the nodes 8 and 9. Since C_2 contains two nodes, Proposition 5.3 implies that the problem has an optimal branching B_2 containing exactly one arc in the cycle C_2 and exactly one arc directed into this cycle as well.

Note that if we use the construction in the proof of Proposition 5.3 as applied to the cycle C_2 and the optimal branching B_1, which contains $|C_1|-1$ arcs from C_1, we produce an optimal branching B_2 by adding $|C_2|-1$ arcs from C_2 to B_1 and deleting some arcs from the set $\delta^-(V(C_2))$. But since the cycles C_1 and C_2 are node disjoint, $\delta^-(V(C_2)) \cap C_1 = \phi$ and so the branching B_2 continues to contain $|C_1|-1$ arcs from the cycle C_1. Therefore, we obtain an optimal branching containing $|C_1|-1$ arcs from the cycle C_1 and $|C_2|-1$ arcs from the cycle C_2. Therefore, we can simultaneously obtain an optimal branching that satisfies the conclusion of Proposition 5.3 for both the cycles C_1 and C_2. This argument applies to any set of disjoint cycles and so permits us to establish the following strengthening of Proposition 5.3.

Proposition 5.4 *Suppose that a node greedy solution NG contains node disjoint directed cycles $C_1, C_2, \ldots, C_J$, none containing the root node r. Then the optimal branching problem always has an optimal branching containing exactly $|C_j|-1$ arcs from each cycle C_j, for $j = 1, 2, \ldots J$ (and so exactly one arc in each directed cutset $\delta^-(V(C_j))$).*

These observations imply that if we contract all the nodes of each cycle from a node greedy solution into a *pseudo node*, then in the resulting reduced digraph we once again need to find an optimal branching. Any optimal branching in the reduced digraph has exactly one directed arc (i, j) into any cycle C in the node greedy solution. Let $(k, j) \in C$ be the arc directed into node j in the node greedy solution. The proof of Propositions 5.3 and 5.4 imply that we obtain an optimal solution of the original branching problem by "expanding" every pseudo node; that is, adding the arcs $C \setminus \{(k, j)\}$ into the optimal branching of the reduced problem.

The discussion shows that we can solve the optimal branching problem using the following algorithm:

(i) *Transform costs.* By subtracting a node-dependent constant γ_v from the weight of the arcs directed into every node $v \neq r$, transform the weights so that the minimum weight arc directed into each node has weight zero.

(ii) *Solve a Relaxed Problem.* Find a node greedy solution NG on the transformed digraph. If the node greedy solution contains no cycles, stop.

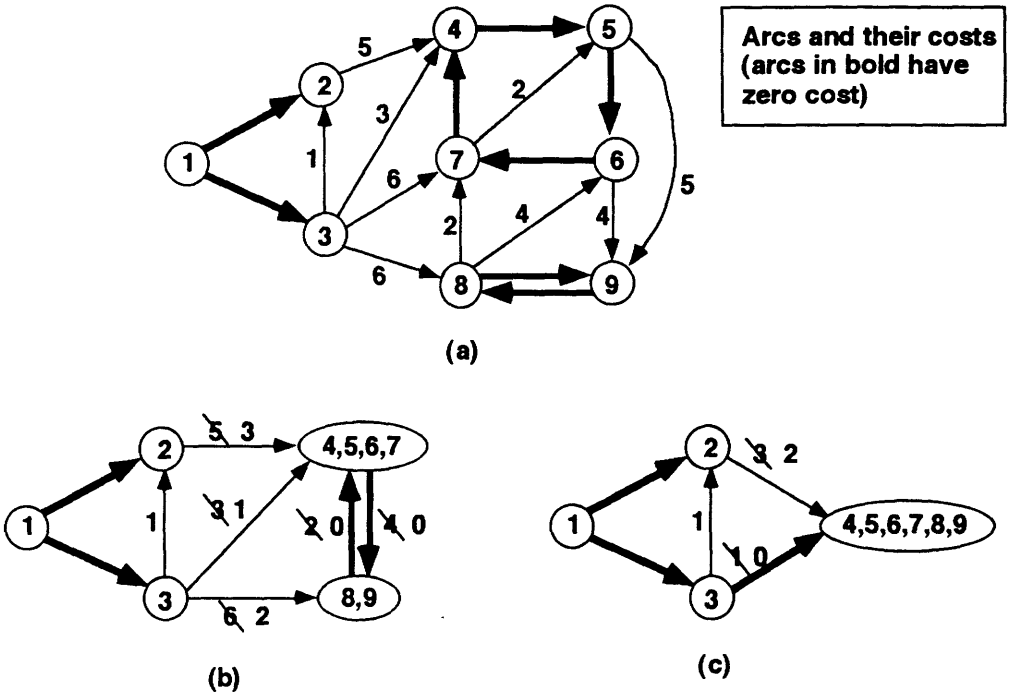


Figure 14: Determining an Optimal Branching

The node greedy solution is optimal and the optimal branching has a weight $\sum_{v\ in V} \gamma_v$. If the node greedy solution contains any cycle, then contract the digraph D into a reduced digraph D_1 by replacing every such cycle C by a single (pseudo) node. Any arc incident into or out of a node in any cycle C becomes an arc, with the same weight, incident into or out of the pseudo node corresponding to that arc. (Note that the reduced digraph might contain parallel arcs; we can eliminate all but the least weight arc of any parallel arcs).

(iii) *Solve the Reducœd Problem.* Find an optimal branching B^R on the reduced digraph and use it to create an optimal branching in the original digraph by expanding every pseudo node. ∎

This algorithm reduces the optimal branching problem to solving an optimal branching problem on the smaller digraph. To solve the reduced problem, we would once again apply the algorithm starting with step (i). Eventually, the node greedy solution will contain no cycles (in the limit, it would have only a single edge). At that point, the algorithm will terminate

Proposition 5.4 and a simple induction argument show that the algorithm finds an optimal branching.

Example 5.1. To illustrate the algorithm, consider the example in Figure 14. As we have noted before, the bold-faced arcs in Figure 14(a) are the arcs in the node greedy solution. This solution contains two cycles; contracting them gives the digraph D_1 shown in Figure 14(b). Note that we have eliminated the parallel arcs created by the arcs $(3,7)$, $(5,9)$, and $(8,6)$.

We next reapply the algorithm on the reduced digraph. First, we subtract $\gamma_{4,5,6,7} = 2$ from the weights of arcs directed into the pseudo node $\{4,5,6,7\}$ and $\gamma_{8,9} = 4$ from the arcs directed into the pseudo node $\{8,9\}$. The bold arcs in Figure 14(b) define the node greedy solution on the reduced digraph. Since this solution contains a cycle (on the two pseudo nodes), we contract it, giving the reduced digraph in Figure 14(c). We decrease the weights of the arcs directed into the pseudo node $\{4,5,6,7,8,9\}$ by $\gamma_{4,5,6,7,8,9} = 1$. Since the node greedy solution (the bold arcs) on the digraph D_2 shown in Figure 14(c) contains no cycle, we stop. ∎

To recover the solution to the original problem, we need to retrace our steps by expanding pseudo nodes, beginning with the optimal branching on the final reduced digraph D_2 (see Figure 15). Since the arc $(3, \{4,5,6,7,8,9\})$

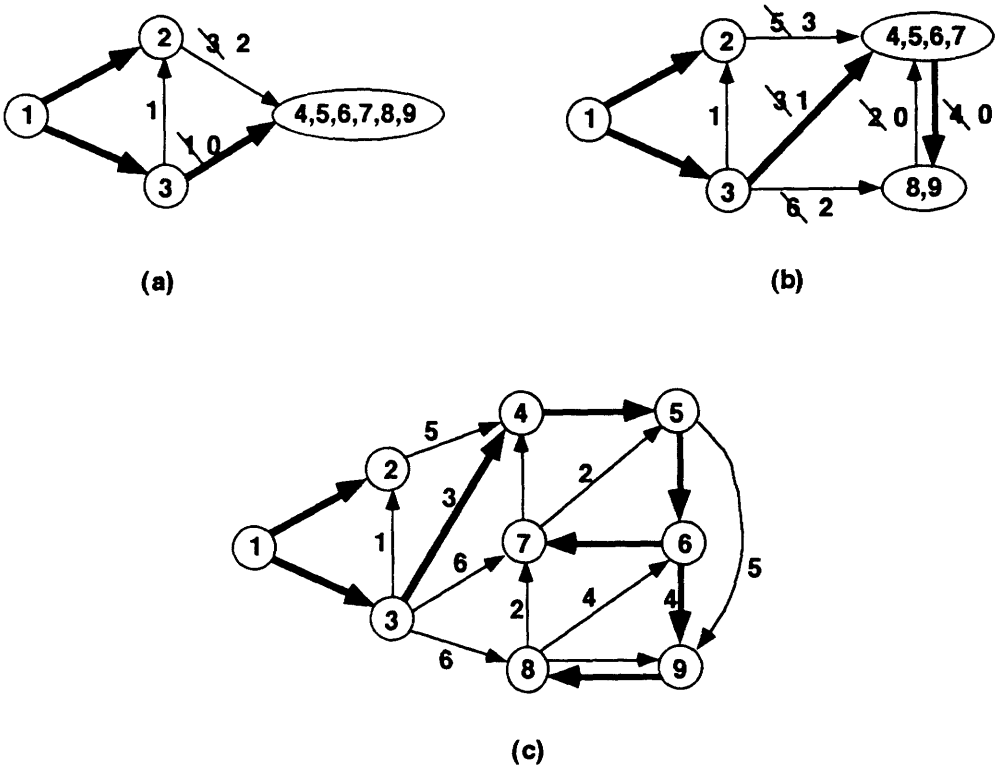


Figure 15: Expanding Pseudo Nodes

directed into the pseudo node $\{4,5,6,7,8,9\}$ corresponds to arc $(3,\{4,5,6,7\})$ in the digraph D_1 (see Figure 14(b)), we delete arc $(\{8,9\}),\{4,5,6,7\})$ from the cycle in the node greedy solution shown in Figure 14(b). Figure 15(b) gives the resulting solution.

We next expand the two pseudo nodes $\{4,5,6,7\})$ and $\{8,9\}$. As we expand these pseudo nodes, we eliminate the arcs $(7,4)$ and $(8,9)$ from the cycles that defined them in Figure 14(a). Figure 15(c) shows the resulting solution. Note that its weight equals $\gamma_{4,5,6,7} + \gamma_{8,9} + \gamma_{4,5,6,7,8,9} = 2 + 4 + 1$.

Polyhedral Implications

Using a combinatorial argument (the development in the proof of Propositions 5.3 and 5.4), we have shown that the branching algorithm produces an optimal branching. We now give an alternative linear programming-based proof.

It is easy to see by inspection that the branching shown in Figure 15(c) is optimal. Any branching must contain at least one arc directed into the node set $\{4,5,6,7\}$ and at least one arc directed into the node set $\{8,9\}$. Since the minimum weight arc in $\delta^-(\{4,5,6,7\})$ is arc $(8,7)$ with weight 2 and in $\delta^-(\{8,9\})$ is arc $(6,9)$ with weight 4, and the sets $\delta^-(\{4,5,6,7\})$ and $\delta^-(\{8,9\})$ are disjoint, *any* feasible branching must have a weight of at least 6. Note that we achieve a weight of 6 only if we choose both the arc $(8,7)$ from $\delta^-(\{4,5,6,7\})$ and $(6,9)$ from $\delta^-(\{8,9\})$. Otherwise, the weight of the chosen arcs is at least 7. But if we choose the arcs $(8,7)$ and $(6,9)$, then the cutset $\delta^-(\{4,5,6,7,8,9\})$ must contain at least one other arc and so the total weight of the branching will exceed 7. Therefore, in every case, the weight of any branching must be at least 7. But since the solution in Figure 15(c) achieves this lower bound, it must be optimal.

This argument is reminiscent of the lower bounding argument we have used in Section 3 for proving that the greedy algorithm solves the (undirected) minimum spanning tree problem. We will use a version of this argument to not only provide another proof that the branching algorithm produces an optimal branching, but also to give a constructive proof of the following integrality theorem.

Theorem 5.5 *The polyhedron $P = Q$ is the convex hull of incidence vectors of branchings.*

Proof. Let us first set some notation. Let $\mathcal{S}$ denote the set of nonempty subsets of $V \setminus \{r\}$ and let $\mathcal{S}_p \subseteq \mathcal{S}$ denote the set of all node sets that define some pseudo node. That is, if $S \in \mathcal{S}_p$ then S is exactly the set of nodes from the original graph in one of the pseudo nodes.

We will establish the result using the cutset representation $Q = \{y \in R^{|A|} : y \geq 0, \sum_{e \in \delta^-(v)} y_e = 1 \text{ for all } v \in V \setminus (\{r\}), \text{ and } \sum_{e \in \delta^-(S)} y_e \geq 1 \text{ for all } S \in \mathcal{S}\}$.

Consider the linear program:

$$\min \{\sum_e w_e y_e : y \in Q\}, \tag{5.1}$$

and its linear programming dual:

$$\max \sum_{S \in \mathcal{S}} \alpha_S \tag{5.2}$$

subject to

$$\sum_{S \in \mathcal{S}} \sum_{\delta^-(S) \ni e} \alpha_S \geq w_e \text{ for all arcs } e \in A \tag{5.3}$$

$$\alpha_S \geq 0 \text{ for all } S \in \mathcal{S} \text{ with } |S| \geq 2. \tag{5.4}$$

When $S = \{v\}$ and $v \neq r$, the dual variable α_S corresponds to the constraint $\sum_{e \in \delta^-(v)} y_e = 1$ in Q. For any set of nodes $S \in \mathcal{S}$ with $|S| \geq 2$, α_S is the dual variable corresponding to the constraint $\sum_{e \in \delta^-(S)} y_e \geq 1$.

Let γ_v for $v \in V \setminus \{r\}$ and γ_S for $S \in \mathcal{S}_p$ be the node-dependent constants used in the weight transformation steps in the branching algorithm. These constants correspond to the nodes v of the original digraph D at the first step of the algorithm and to pseudo nodes (whose node sets are S) at later steps. For any node $v \neq r$, suppose that we set $\alpha_{\{v\}} = \gamma_v$ and for any set $S \in \mathcal{S}_p$ corresponding to a pseudo node, suppose that we set $\alpha_S = \gamma_S \geq 0$. Define $\alpha_S = 0$ otherwise.

The steps of the branching algorithm imply that final weights $\overline{w}_e$ for each arc e are nonnegative. The branching algorithm also implies that if edge e is directed into node $v^+(e)$, then $\overline{w}_e = w_e - \sum_{\{S \in \mathcal{S}_p \text{ with } \delta^-(S) \ni e\}} \gamma_S - \gamma_v^+(e)$. The last term includes the node weight γ_v of the node that the edge e is directed into. Therefore, the variables γ_v and γ_S are feasible in the linear programming

dual. As we have already seen, the branching found by the algorithm has a cost $\sum_{v \neq r} \gamma_v + \sum_{S \in \mathcal{S}_p} \gamma_S$. Therefore, the algorithm constructs an integer feasible solution to the linear program (5.1) and its weight equals the weight of a feasible dual solution. As a result, linear programming theory implies that (i) the branching solution solves the linear program (5.1); therefore, the linear program has an integer solution (the incidence vector of a branching) for every choice of objective function, and, consequently, (ii) the incidence vector of branchings are the extreme points of the underlying polyhedron $Q = P$. ∎

6 The Steiner Tree Problem.

As we noted in Section 2, the (undirected) Steiner tree (ST) problem is defined by a graph $G = (V, E)$, with edge weights w_e on the edges $e \in E$, and with a subset of the nodes T called *terminal nodes*. The objective is to find a minimum weight subtree of G that spans all the nodes in T. The subtree might or might not include some of the the other (optional) nodes $S = V \setminus T$, which we refer to as *Steiner nodes*.

Two special versions of the Steiner tree problem are easy to solve. When $|T| = 2$ and $w_e \geq 0$ for $e \in E$, the problem reduces to the shortest path problem, and when $T = V$, it is the spanning tree problem that we examined in Section 3. In general, the problem is difficult to solve (in the parlance of computational complexity theory, it is NP-complete), and so we will not be able to solve it using combinatorial or dynamic programming algorithms of the type we have considered in the earlier sections. For more complicated models like this, an important step is often to find a "good" linear programming representation of the problem. Starting with an initial integer or mixed integer programming formulation, the goal is to add new valid constraints or new variables to the model so that the resulting linear program provides a tight bound on the value of the optimal solution— in some cases even an integral and, hence, optimal solution.

This section is divided into three parts. We begin by examining different integer programming formulations of a slight generalization of (ST), called the node weighted Steiner tree problem ($NWST$), and showing that the values of the linear programming relaxations of all these formulations are the same. This discussion generalizes our development in Section 3 of alternate formulations of the minimum spanning tree problem. We then briefly discuss computational studies based on these formulations. Finally, we consider the strength of a linear programing relaxation for the Steiner problem and present results on the worst case behavior of simple heuristics for (ST) and (NWST).

6.1 The node weighted Steiner tree problem ($NWST$)

Given a graph $G = (V, E)$ with edge weights w_e for $e \in E$, we designate one node r as a root node. Node r will be the only terminal node (that is,

$T = \{r\}$). For all the other nodes $j \in V \setminus \{r\}$, we incur a profit d_j if the Steiner tree contains node j.

To formulate the node weighted Steiner tree problem ($NWST$), we let $z_j = 1$ if node j is in the Steiner tree, and $z_j = 0$ otherwise; $x_e = 1$ if edge e is in the tree and $x_e = 0$ otherwise. The first formulation we present, called *the subtour formulation*, is a natural generalization of the subtour formulation of the spanning tree problem, namely:

$$\min \sum_{e \in E} w_e x_e - \sum_{i \in V} d_i z_i \tag{6.1}$$

subject to

$$\sum_{e \in E(U)} x_e \leq \sum_{i \in U \setminus \{k\}} z_i \text{ for all } U \subset V \text{ and for all } k \in U \tag{6.2}$$

$$\sum_{e \in E} x_e = \sum_{i \in V \setminus \{r\}} z_i \tag{6.3}$$

$$z_r = 1 \tag{6.4}$$

$$0 \leq x_e \leq 1, 0 \leq z_i \leq 1 \tag{6.5}$$

$$x, z \text{ integer.} \tag{6.6}$$

Note that the first set of constraints (6.2), called *generalized subtour elimination constraints*, imply that the solution contains no cycles on the subgraph of G determined by the selected nodes (those with $z_i = 1$). The second set of constraints (6.3) ensures that the solution spans the set of selected nodes, and so defines a Steiner tree. We let P_{sub} denote the set of feasible solutions to (6.2)-(6.5) in the (x, z)-space.

Before proceeding, let us make one observation about this formulation. Note that if $r \in U$ and $k \neq r \in U$, then

$$\sum_{i \in U \setminus \{r\}} z_i = \sum_{i \in U \setminus \{k\}} z_i + z_k - z_r \leq \sum_{i \in U \setminus \{k\}} z_i$$

since $z_r = 1$ and $z_k \leq 1$. Therefore, when $r \in U$, all the constraints (6.2) with $k \neq r \in U$ are redundant.

It is instructive to view the node weighted Steiner tree problem in another way: as a spanning tree problem with additional constraints. To develop this interpretation, suppose that we let $\bar{z}_r = z_r$ and complement the node

variables z_i for $i \neq r$ in the previous model; that is, let $\bar{z}_i = 1 - z_i$ and replace z_i with $1 - \bar{z}_i$, creating the following alternative formulation:

$$\min \sum_{e \in E} w_e x_e + \sum_{i \in V} d_i \bar{z}_i - \sum_{i \in V} d_i \tag{6.7}$$

subject to

$$\sum_{e \in E(U)} x_e + \sum_{i \in U \setminus \{k\}} \bar{z}_i \leq |U| - 1 \text{ for all } U \subset V \text{ and for all } k \in U \tag{6.8}$$

$$\sum_{e \in E} x_e + \sum_{i \in V \setminus \{r\}} \bar{z}_i = |V| - 1 \tag{6.9}$$

$$\bar{z}_r = 1 \tag{6.10}$$

$$0 \leq x_e \leq 1, 0 \leq \bar{z}_i \leq 1 \tag{6.11}$$

$$x, \bar{z} \text{ integer.} \tag{6.12}$$

To interpret this problem as a variant of the minimum spanning tree problem, suppose that we add a supernode 0 to the underlying graph G with an edge $\{0, i\}$ between this node and every node i in G. Let $\bar{G}$ denote the resulting graph. Then the zero-one variable $\bar{z}_i$ for $r \neq r$ indicates whether or not we include edge $\{0, i\}$ in the minimum spanning tree on $\bar{G}$ (observe that we *include* edge $\{0, i\}$ in the tree, that is, $\bar{z}_i = 1$, when we *exclude* node i in the previous formulation, that is, $z_i = 0$). We always include edge $\{0, r\}$.

The constraint (6.8) with $U = \{i, j\}$ for any two nodes $i \neq r$ and $j \neq r$ implies that $x_e + \bar{z}_i \leq 1$ for all $e \in \delta(i)$. That is, if the chosen tree contains edge $\{0, i\}$, then it contains no edge from G incident to that node. In the formulation (6.1)-(6.8), this statement says that if we exclude node i from the tree, then the tree cannot contain any edge incident to that node. The inequalities $x_e + \bar{z}_i \leq 1$ and the fact that any solution to the model (6.7)-(6.12) contains edge $\{0, r\}$ implies that any spanning tree solution to the formulation (6.7)-(6.12) has the following form: if we remove node 0 and its incident edges, the resulting forest is a subtree containing the node r as well as a set of isolated nodes j (those with $\bar{z}_j \leq 1$).

Next consider the subtour constraint (6.8) in this model for any set U that does not contain the root node r. In the spanning tree formulation, we would have written this constraint without the $\bar{z}_i$ variables as $\sum_{e \in E(U)} x_e \leq |U| - 1$. The inequality (6.8) is a strengthening of that inequality: if $\bar{z}_i = 0$ for all nodes $i \in U$, then the set $E(U)$ can contain as many as $|U| - 1$

edges; every time we add an edge $\{0, i\}$ to the tree for any node $i \in U$, node i become isolated, so the effective size of $|U|$ decreases by one and the tree can contain one fewer edge from $E(U)$. If the set U contains node r, then as we saw previously, the constraints (6.8) with $k \neq r$ are redundant, so the only effective inequality in the complimented model (6.7)-(6.12) is $\sum_{e \in E(U)} x_e + \sum_{i \in U \setminus \{r\}} \bar{z}_i \leq |U| - 1$. Note that these constraint are exactly the usual subtour breaking constraints on the node set $U \cup \{0\}$, given that the solution contains the edge $\{0, r\}$.

Our next model, the so-called *multicut formulation*, is a generalization of the multicut formulation of the minimum spanning tree problem. Let P_{mcut} be the set of solutions in the (x, z) variables to the constraints (6.3)-(6.5) as well as the additional inequalities

$$\sum_{e \in \delta(C_0, \ldots, C_s)} x_e \geq \sum_{j=1}^{s} z_{i_j} \tag{6.13}$$

over all node partitions $(C_0, C_1, \ldots, C_s)$ of V with $r \in C_0$. In this expression, as in our earlier discussion, $\delta(C_0, C_1, \ldots, C_s)$ is the multicut defined by the node partition (i.e., the set of edges having endpoints in different subsets C_i, C_j), and $i_j \in C_j$ for $j = 1, \ldots, s$. These constraints say that if k of the sets $(C_1, \ldots, C_s)$ contain nodes of the Steiner tree, and C_0 contains the root node, then the multicut must contain at least k edges of the Steiner tree.

Proposition 6.1 $P_{sub} = P_{mcut}$.

Proof. Summing the inequalities (6.2) for the sets $U = C_0, C_1, \ldots, C_s$ with $k = i_j$ gives

$$\sum_{t=0}^{s} \sum_{e \in E(C_t)} x_e \leq \sum_{t=0}^{s} \sum_{i \in (C_t \setminus \{i_t\})} z_i.$$

Taking $i_0 = r$ and subtracting from (6.3) gives (6.13). Thus, $P_{sub} \subseteq P_{mcut}$.

Conversely, suppose first that $r \in U$. Consider the inequality (6.13) with $C_0 = U$, and with $C_1, \ldots, C_k$ as singletons whose union is $V \setminus U$. Subtracting from (6.3) gives $\sum_{e \in E(U)} x_e \leq \sum_{i \in U \setminus \{r\}} z_i$. As we have noted previously, this inequality implies (6.3) for all nodes $k \in U$.

If $r \notin U$, take $C_0 = \{r\}, C_1 = U$ and $C_2, \ldots, C_k$ as singletons whose union is $V \setminus (U \cup \{r\})$. Subtracting (6.13) from (6.3) gives $\sum_{e \in E(U)} x_e \leq \sum_{i \in U \setminus \{i_1\}} z_i$ for $i_1 \in U$. Thus, $P_{mcut} \subseteq P_{sub}$. ∎

The next formulation again uses the observation that we can orient any Steiner tree to obtain a directed Steiner tree with node r as its root. In this formulation, the vector y represents an edge incidence vector of this directed Steiner tree. The *directed subtour formulation* is given by the constraints (6.2)-(6.5) and the *the dicut inequality*

$$\sum_{(i,j)\in\delta^-(j)} y_{ij} = z_j \text{ for } j \in V \setminus \{r\} \tag{6.14}$$

$$y_{ij} + y_{ji} = x_e \text{ for } e = \{i,j\} \in E \tag{6.15}$$

$$y_{ij}, y_{ji} \geq 0 \text{ for } e = \{i,j\} \in E. \tag{6.16}$$

Constraints (6.14) say that if node j is in the Steiner tree, one arc of the directed Steiner tree enters this node. We let P_{dsub} denote the set of feasible solutions in the (x,z) space.

Similarly *the directed cut formulation* is given by the constraints (6.3)-(6.5),(6.15),(6.16) and

$$\sum_{(i,j)\in\delta^+(C)} y_{ij} \geq z_k \text{ for all } k \in V, \text{ and all } C \text{ with } r \in C \subseteq V \setminus \{k\}. \tag{6.17}$$

These constraints say that if k is in the Steiner tree, any directed cut separating nodes r and k must contain at least one arc of the directed Steiner tree. We let P_{dcut} denote the set of feasible solutions in the (x,z) space.

Proposition 6.2 $P_{dsub} = P_{dcut}$.

Proof. The proof mimics the proof of Proposition 3.6. In this case, when $r \in \overline{S}$ the right-hand side of equation (3.36) becomes $\sum_{k\in V\setminus\{r\}} z_k$ instead of $n-1$ and by equation (6.14) the last term on the left-hand side of equation (3.36) equals $\sum_{k\in\overline{S}} z_k$. Therefore, the right-hand side of equation (3.37) becomes $\sum_{k\in V\setminus\{r\}} z_k - \sum_{k\in\overline{S}} z_k = \sum_{i\in S} z_i$. So this equation becomes

$$\sum_{e\in A(S)} y_e + \sum_{e\in\delta^+(\overline{S})} y_e = \sum_{i\in S} z_i.$$

This equality implies that y satisfies inequality (6.2) for S if and only if it satisfies the inequality (6.17) for $\overline{S}$.

If $r \in S$, then the arguments used in the proof of Proposition 3.6 show that the right-hand side of the last displayed equation is $\sum_{i \in S \setminus \{r\}} z_i$, and since the last term on the left-hand side of this equation is nonnegative, the equation implies (6.2) for $k = r$, which as we have seen implies (6.2) for all $k \in S$.

These arguments show that $y \in P_{dsub}$ if and only if $y \in P_{dcut}$. ∎

The four formulations we have described so far all have an exponential number of constraints. The next formulation we consider has only a polynomial number of constraints, but an additional $0(n^3)$ variables. We obtain this *directed flow formulation* by remodelling the cut inequalities (6.17). More specifically, viewing the values of the variables y_{ij} as capacities, (6.16) and (6.17) say that every cut separating nodes r and k has capacity at least z_k. By the max-flow min-cut theorem, these inequalities are valid if and only if the network has a feasible flow of z_k units from node r to node k. Letting f_{ij}^k denote the flow destined for node k in arc (i,j), we obtain P_{dflo} which is the set of feasible solutions in the (x,z)-space of the constraints (6.3)-(6.5), (6.15), (6.16) and the equations describing such a flow:

$$\sum_{i \in \delta^+(j)} f_{ji}^k - \sum_{i \in \delta^-(j)} f_{ij}^k = 0 \text{ for all } j \neq r, j \neq k, k \neq r \qquad (6.18)$$

$$\sum_{i \in \delta^+(k)} f_{ki}^k - \sum_{i \in \delta^-(k)} f_{ik}^k = -z_k \text{ for all } k \neq r \qquad (6.19)$$

$$0 \leq f_{ij}^k \leq y_{ij} \text{ for all } (i,j) \in A, k \neq r \qquad (6.20)$$

As was the case for the minimum spanning tree problem, the directed flow and directed cut formulations are equivalent in the following sense.

Proposition 6.3 $P_{dcut} = P_{dflo}$.

The directed formulation is as strong as the undirected formulation (e.g., $P_{dsub} \subseteq P_{sub}$) since it contains additional constraints. Are the directed formulations P_{dsub} (respectively P_{dcut}, P_{dflo}) stronger than the undirected formulations P_{sub} (respectively P_{mcut})? As for the case of trees, it turns out that these polyhedra are identical even though, in general, they are no longer integral.

Theorem 6.4 $P_{sub} = P_{dsub}$.

Proof. As we have just noted, $P_{dsub} \subseteq P_{sub}$. To show the converse, we need to show that for every $(x', z') \in P_{sub}$, there exists a y with $(x', z', y) \in P_{dsub}$. In other words, we need to verify that the system

$$\begin{aligned} \sum_{i \in \delta^-(r)} y_{ir} &= 0 \\ \sum_{i \in \delta^-(j)} y_{ij} &= z'_j \text{ for } j \in V \setminus \{r\} \\ y_{ij} + y_{ji} &= x'_e \text{ for } e = \{i,j\} \in E \\ y_{ij}, y_{ji} &\geq 0 \text{ for } e = \{i,j\} \in E \end{aligned}$$

has a feasible solution. This is a feasibility transportation problem with a supply 0 for node $j = r$ and a supply z'_j for each node $j \in V \setminus \{r\}$ and demand x'_e for each edge $e \in E$. Equation (6.3) implies that the sum of supplies equals the sum of the demands. To determine if this system has a feasible solution, we can formulate a maximum flow problem with a node for each node $v \in V$, a node for each edge $e \in E$, and a directed arc (v, e) of infinite capacity whenever v is incident to edge e. We also introduce a source node s and terminal node t and directed arcs (s, v) with capacity z'_v (with $z'_r = 0$) for each $v \in V$ and directed arcs (e, t) with capacity x'_e for each $e \in E$. The transportation problem has a feasible flow if and only if the maximum flow problem has a flow of value $\sum_{e \in E} x'_e$ from node s to node t, which by the max-flow min-cut theorem is true if and only if the capacity of every cutset is at least $\sum_{e \in E} x'_e$.

Let $S \subseteq V$ and $F \subseteq E$ be the supplies and demands on the t side of a cut. The cut has infinite capacity if it has an edge between $V \setminus S$ and F. Thus, the cut capacity is finite only if $F \subseteq E(S)$. For given sets S and F, the capacity is $\sum_{i \in S \setminus \{r\}} z'_i + \sum_{e \in E \setminus F} x'_e$ which is minimized when $F = E(S)$ giving $\sum_{i \in S \setminus \{r\}} z'_i + \sum_{e \in E \setminus E(S)} x'_e$. This sum is at least $\sum_{e \in E} x'_e$ if and only if $\sum_{e \in E(S)} x'_e \leq \sum_{i \in S \setminus \{r\}} z'_i$. But (6.2) implies that this inequality is valid for all (x', z'), implying the claim. ∎

Example 6.1. Figure 16 shows a fractional solution in the (x, z) variables satisfying constraints (6.3)-(6.5), and a compatible directed (x, z, y) solution satisfying constraints (6.14)-(6.16) as well. Observe that the fractional solution violates the subtour inequality with $C = \{2, 3, 4\}$ and $k = 3$, the multicut inequality with $C_0 = \{1\}, C_1 = \{2, 3, 4\}, C_2 = \{5\}, C_3 = \{6\}$ and

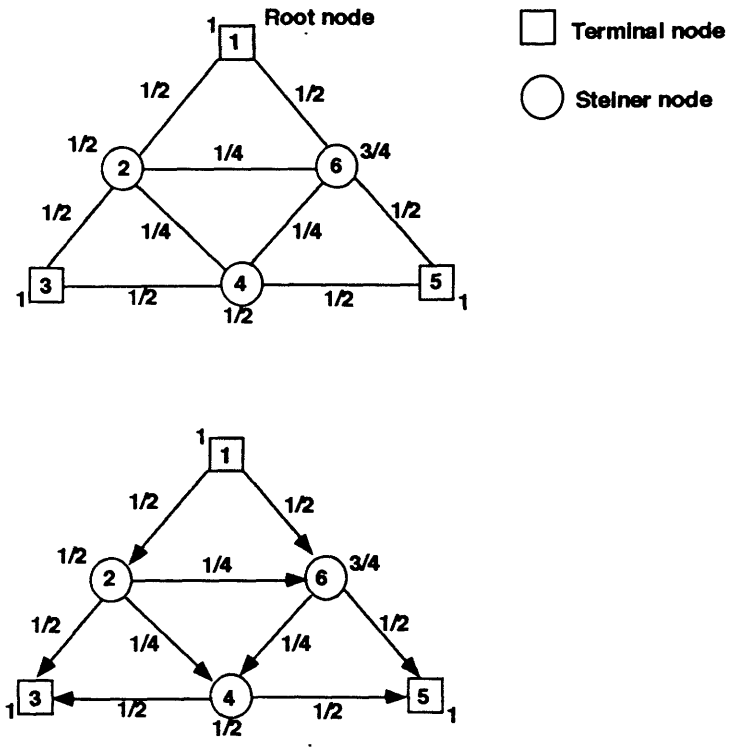


Figure 16: Fractional Solution that Violates the Subtour, Multicut, and Dicut Formulations: (a) Undirected version; (b) Directed version

$i_1 = 3$, and the dicut inequality with $C = \{1, 5, 6\}$ and $k = 3$, each by 1/4 unit.

The final *directed branchings* formulation is also of polynomial size, but based on the separation problem for the generalized subtour inequalities (6.2). It is described by the constraints (6.3)-(6.5) and the constraints:

$$\sum_{i \in \delta^-(j)} y^k_{ij} \leq z_j \text{ for } j \in V \setminus \{k\}, \text{ and all } k \tag{6.21}$$

$$\sum_{i \in \delta^-(k)} y^k_{ik} \leq 0 \text{ for all } k \tag{6.22}$$

$$y^k_{ij} + y^k_{ji} = x_e \text{ for } e = (i,j) \in E, \text{ and all } k \tag{6.23}$$

$$y^k_{ij}, y^k_{ji} \geq 0 \text{ for } e = (i,j) \in E, \text{ and all } k \tag{6.24}$$

Let P_{dbran} denote the set of (x, z) solutions to this model.

Note that once the Steiner tree is fixed, then for each node k in the tree, we can orient the edges to construct a directed Steiner tree rooted at node k. Suppose we define $y^k_{ij} = 1$ if the arc (i,j) belongs to this branching and $y^k_{ij} = 0$ otherwise. If k is not in the tree, we can choose r as the root. Consequently, this formulation is valid.

Proposition 6.5 $P_{dbran} = P_{sub}$.

Proof. Summing the inequalities (6.21) over $C \setminus \{k\}$ and adding (6.22) gives

$$\sum_{e \in A(C)} y^k_e + \sum_{(i,j) \in \delta^-(C)} y^k_{ij} \leq \sum_{i \in C \setminus \{k\}} z_i.$$

Thus $P_{dbran} \subseteq P_{sub}$.

Conversely, note that (x, z) satisfies the generalized subtour inequality (6.2) for all node sets C containing node k if and only if

$$\eta^k = \max_{\nu \in \{0,1\}} \{ \sum_{e=(i,j) \in E} x_e \nu_i \nu_j - \sum_{j \neq k} z_j \nu_j : \nu_k = 1 \} = 0.$$

In this expression, ν_i=1 if $i \in C$ and $\nu_i = 0$ otherwise. This problem can be solved as a linear program, namely:

$$\eta^k = \max \sum_{e \in E} x_e \alpha_e \quad - \quad \sum_{j \neq k} z_j \nu_j$$

subject to

$$\begin{aligned} \alpha_e - \nu_i &\leq 0 \text{ for all } i \in V \text{ and all } e \in \delta^+(i) \\ \alpha_e - \nu_j &\leq 0 \text{ for all } j \in V \text{ and all } e \in \delta^-(j) \\ \nu_i &\geq 0 \text{ for all } j \in V. \end{aligned}$$

Since the constraints in this system are homogeneous, ν^k equals either 0 or $+\infty$. Because the constraint matrix for this model is totally unimodular, its extreme rays have $0 - 1$ coefficients. Thus the solution (x, z) satisfies the subtour inequality if and only if $\eta^k = 0$, which by linear programming theory is true if and only if the following dual problem is feasible

$$\begin{aligned} y^k_{ij} + y^k_{ji} &= x_e \\ -\sum_{i \in \delta^-(j)} y^k_{ij} &\geq -z_j \text{ for all } j \in V \setminus \{k\} \\ -\sum_{i \in \delta^-(k)} y^k_{ik} &\geq 0 \\ y^k_{ij} &\geq 0. \end{aligned}$$

The conclusion establishes the claim. ∎

We close this discussion with three observations.

(1) As we noted before, the formulation P_{dflo} contains $0(n^3)$ variables ($n = |V|$), which leads to impractically large linear programs. However, this formulation indicates how to carry out separation for the formulations P_{dcut} and P_{sub}. Just as the directed flow formulation provided us with a separation procedure for the subtour inequalities of the spanning tree polyhedron, P_{dflo} provides a separation procedure, via a series of maximum flow computations for the inequalities (6.17) or (6.2).

(2) To obtain a formulation of the minimum spanning tree problem from each of the formulations we have considered in this section, we would add the constraints $z_v = 1$ for all nodes $v \neq r$ to each of these models. Since, as we have shown, the node weighted Steiner tree polyhedra for each of these formulations are the same, their intersection with the constraints $z_v = 1$ will be the same as well. Therefore, the results in this section generalize those in Section 3. Moreover, the formulation P_{dbran} with the additional stipulation that $z_v = 1$ for all $v \neq r$ provides us with yet another integer formulation of

the spanning tree polyhedron that is equivalent to the six integer formulations we examined in Section 3.

(3) In addition to the models we have considered in this discussion, we could formulate straightforward extensions of the single commodity flow and cut formulations of the spanning tree problem for the node-weighted Steiner tree problem. These formulations will, once again, be weak, in the sense that their linear programming relaxations will provide poor approximations for the underlying integer polyhedra. We could also state a multicommodity flow formulation with bidirectional forcing constraints as in (3.47). The arguments given in Section 3 show that this formulation is equivalent to the directed formulation.

6.2 The Steiner Problem

What happens when we specialize the node weighted formulations for the Steiner problem by taking $z_i = 1$ for all $i \in T$ and setting $d_j = 0$ for all $j \in V \setminus T$? The first obvious alternative is to work with the six extended formulations we have just examined.

A second approach is to find a formulation without the node variables z. Note that formulation P_{dcut} easily provides one such formulation. We simply eliminate the cardinality constraint (6.3) and the dicut constraints (6.17) whenever $k \notin T$. The resulting formulation is (6.15), (6.16) and

$$\sum_{(i,j)\in\delta^+(C)} y_{ij} \geq 1 \text{ for all } C \text{ with } r \in C \subseteq V \text{ and } (V \setminus C) \cap T \neq \phi \quad (6.25)$$

The integer solutions of this formulation are Steiner trees and their supersets. Eliminating constraints in a similar fashion for the formulation P_{mcut}, gives

$$\sum_{e\in\delta(C_0,\ldots,C_s)} x_e \geq s \quad (6.26)$$

over all node partitions $(C_0, C_1, \ldots, C_s)$ of V with $r \in C_0$ and $C_i \cap T \neq \phi$ for $i = 1, \ldots, s$. Note, however, that the resulting directed cut and multicut formulations no longer are equivalent. The fractional solution shown in Figure 16 satisfies all the multicut inequalities (6.26), but is infeasible for the dicut polyhedron (6.15), (6.16) and (6.25).

For the other four formulations, there is no obvious way to eliminate constraints to obtain a formulation of this type.

A third approach would be to find an explicit description of $Q_{sub} = proj_x(P_{sub})$ and, ideally, of $Q_{ST} = \text{conv}(Q_{sub} \cap \{x : x \text{ integer}\})$.

Researchers have obtained various classes of facet-defining inequalities for Q_{ST}. (Facet-defining inequalities are inequalities that are necessary for describing a region defined by linear inequalities. All the others are implied by the facet-defining inequalities and are thus theoretically less important. However, in designing practical algorithms, facets might be hard to find or identify, and so we often need to use inequalities that are not facet-defining).

Proposition 6.6 *(Steiner Partition Inequalities) Let $C_1, \ldots, C_s$ be a partition of V with $T \cap C_i \neq \phi$ for $i = 1, \ldots, s$, then the multicut inequality:*

$$\sum_{e \in \delta(C_1, \ldots, C_s)} x_e \geq s - 1$$

defines a facet of Q_{ST} if
i) the graph defined by shrinking each node set C_i into a single node is two-connected, and
ii) for $i = 1, \ldots, s$, the subgraph induced by each C_i is connected.

Another class of facet-defining inequalities are the "odd holes". Consider a graph $G^t = (V, E)$ on $2t$ nodes, with t odd, V composed of terminal nodes $T = \{u_1, \ldots, u_t\}$ and Steiner nodes $V \setminus T = \{v_1, \ldots, v_t\}$ and $E \supseteq E_t = \{(u_i, v_i)_{i=1}^t, (v_i, v_{i+1})_{i=1}^t, (v_i, u_{i+1})_{i=1}^t\}$. In this expression, $v_{t+1} = v_1$ and $u_{t+1} = u_1$.

Proposition 6.7 *The inequality*

$$\sum_{e \in E_t} x_e + 2 \sum_{e \in E \setminus E_t} x_e \geq 2(t-1)$$

is a facet defining inequality for G^t.

In the odd hole (V, E_t), each terminal node u_i is at least two edges away from any other terminal node u_j, so using only the edges in E_t to connect the terminal nodes requires at least $2(t-1)$ edges. Every edge in $E \setminus E_t$ that we add to the Steiner tree can replace two such edges. This fact accounts for the factor of 2 on the edges in $E. \setminus E_t$.

Example 6.1. The graph shown in Figure 16 is itself an odd hole with $t = 3$. Note that the fractional edges values in this figure belong to Q_{sub} and satisfy all the multicut inequalities. This solution does, however, violate the following odd hole inequality:

$$x_{12} + x_{16} + x_{26} + x_{24} + x_{23} + x_{34} + x_{46} + x_{45} + x_{56} \geq 2(3 - 1) = 4.$$

■

Another known extensive class are the so called combinatorial design facets. All three classes are valid for Q_{sub} and thus would be generated implicitly by any vector that satisfies all violated generalized subtour inequalities for P_{sub}. However, surprisingly, the separation problem for the Steiner Partition inequalities is NP-complete.

Now consider the algorithmic question of how to solve the Steiner problem. The latest and perhaps most successful work has been based on the formulations we have examined.. Three recent computational studies with branch and cut algorithms have used the formulations P_{sub} and P_{dcut} with the edge variables x_e eliminated by substitution. Two other recent approaches have been dual. One involves using dual ascent heuristics to approximately solve the P_{dcut} formulation. Another has been to use Lagrangian relaxation by dualizing the constraints $x_e + \bar{z}_i$ for all edges $e \in \delta(i)$ in the model (6.7)-(6.12). If we further relax (drop) the variables $\bar{z}_i$ from the constraints (6.8), the resulting subproblem is a minimum spanning tree problem.

6.3 Linear Programming and Heuristic Bounds for the Steiner Problem.

Considerable practical experience over the last two decades has shown that a formulation of an integer program is effective computationally only when the optimal objective value of the linear programming relaxation is close to the optimal value of the integer program (within a few percent). Moreover solution methods often rely on good bounds on the optimal solution value. These considerations partly explain our efforts to understand different formulations.

Just how good a lower bound does the linear programming relaxation P_{sub} provide for the optimal value of the Steiner problem? Unfortunately, nothing appears to be known about this question. However a bound is available for

the weaker cut formulation introduced at the end of Section 3, and which extends naturally to (SP), namely

$$\begin{aligned} Z &= \min \sum_{e \in E} w_e x_e \\ \text{subject to} & \\ \sum_{e \in \delta(S)} x_e &\geq 1 \text{ for } S \subset V \text{with } S \cap T \neq \phi, T \setminus S \neq \phi \\ x_e &\in \{0,1\} \text{ for } e \in E. \end{aligned}$$

Note that this formulation is a special case of the Survivable Network Problem formulation:

$$\begin{aligned} Z &=. \min \sum_{e/inE} w_e x_e \\ \text{subject to} & \\ \sum_{e \in \delta(U)} x_e &\geq r_U \text{ for } U \subset V \\ x_e &\geq 0, x_e \text{ integral for } e \in E \end{aligned}$$

treated in greater detail in Chapter qq. Here we just note that we obtain the Steiner problem by taking $r_U = 1$ whenever $\phi \subset U \subset V, U \cap T \neq \phi, T \setminus U \neq \phi$, and $r_U = 0$ otherwise.

Consider a restriction of the problem obtained by replacing the inequalities by equalities for the singleton sets $U = \{i\}$ for $i \in D \subseteq V$. The resulting problem is:

$$\begin{aligned} Z(D) &= \min \sum_{e} w_e x_e \\ \text{subject to} & \\ \sum_{e \in \delta(U)} x_e &\geq r_U \text{ for } U \subset V \\ & . \\ \sum_{e \in \delta(\{i\})} x_e &= r_{\{i\}} \text{ for } i \in D \\ x_e &\geq 0, x_e \text{ integral for all } e \in E. \end{aligned}$$

Thus $Z = Z(\phi)$. We let $Z^{LP}(D)$ denote the value of the corresponding linear programming relaxation and let $Z^{LP} = Z^{LP}(\phi)$ be the value of the basic linear programming relaxation. The following surprising result concerning this linear program is known as the "Parsimonious Property".

Theorem 6.8 *If the distances w_e satisfy the triangle inequality, then $Z^{LP} = Z^{LP}(D)$ for all $D \subseteq V$.*

We obtain a particularly interesting case by choosing D to be the set of Steiner nodes, i.e., $D = V \setminus T$. Since $\sum_{e \in \delta(\{i\})} x_e = 0$ for $i \notin T$, the problem reduces to:

$$\begin{aligned} Z(V \setminus T) &= \min \sum_e w_e x_e \\ \text{subject to} & \\ \sum_{e \in \delta(U)} x_e &\geq 1 \text{ for } \phi \subset U \subset T \\ x_e &\geq 0, x_e \text{ integral for } e \in E(T), \end{aligned}$$

namely, the spanning tree problem on the graph induced by the terminal nodes T. Applying Theorem 6.8 to the graph $G' = (T, E(T))$ shows that

$$\begin{aligned} Z^{LP}(V \setminus T) &= \min \sum_e w_e x_e \\ \text{subject to} & \\ \sum_{e \in \delta(U)} x_e &\geq 1 \text{ for } U \subset T \\ \sum_{e \in \delta(\{i\})} x_e &= 1 \text{ for } i \in T \\ x_e &\geq 0 \text{ for } e \in E(T). \end{aligned}$$

If we multiply the right hand side of the constraints by two, the resulting model is a well-known formulation for the traveling salesman problem (two edges, one "entering" and one "leaving", are incident to each node and every cutset contains at least two edges) on the graph $G' = (T, E(T))$. The corresponding linear programming relaxation is known as the Held and Karp relaxation; we let $Z^{HK}(T)$ denote its value. We have established the following result.

Proposition 6.9 *If the distances w_e satisfy the triangle inequality, then* $Z^{LP} = (1/2)Z^{HK}(T)$.

This result permits us to obtain worst case bounds both for the value of the linear programming relaxation Z^{LP} and of a simple heuristic for the Steiner tree problem.

To apply this result, we either need to assume that the distances satisfy the triangle inequality or we can first triangularize the problem using the following procedure: replace the weight w_e on each edge $e = (i, j)$ by the shortest path distance d_e between nodes i and j. (To use this result, we need to assume that the shortest path distances exist: that is, the graph contains no negative cost cycles. We will, in fact, assume that $w_e \geq 0$ for all edges e).

Proposition 6.10 *The Steiner tree problem with the distances $w_e \geq 0$ and the Steiner tree problem with the shortest path distances d_e have the same solutions and same optimal objective values.*

Proof. Since $d_e \leq w_e$ for every edge e, the optimal value of the triangularized problem cannot exceed the optimal value of the problem with the distances d_e. Let ST be an optimal Steiner tree for the problem with distances w_e. If $d_e = w_e$ for every edge $e \in ST$, then ST also solves the triangularized Steiner problem and we will have completed the proof. Suppose $d_{\bar{e}} < w_{\bar{e}}$ for some edge $\bar{e} \in ST$. Then we delete edge $\bar{e}$ from ST and add the edges not already in ST that lie on the shortest path joining the end nodes of edge $\bar{e}$. The resulting graph G* might not be a tree, but we can eliminate edges (which have costs $w_e \geq 0$) until the graph G* becomes a tree. The new Steiner tree has a cost less than the cost of ST; this contradiction shows that $d_e = w_e$ for every edge $e \in ST$ and thus completes the proof.[7] ∎

The Tree Heuristic for the Steiner tree problem. If the distances w_e satisfy the triangle inequality, construct a minimum cost spanning tree on

[7]This same argument applies, without removing the edges at the end, to any network survivability problem, even when some of the costs are negative (as long as the graph does not contain any negative cost cycles, so that shortest paths exist). If we permit the solution to the Steiner tree problem to be any graph that contains a Steiner tree, that is, a Steiner tree plus additional edges, then the result applies to this problem as well when some of the costs are negative.

the graph induced by the terminal nodes T. If the distances w_e do not satisfy the triangle inequality,

Step 1. Compute the shortest path lengths $\{d_e\}$.

Step 2. Compute the minimum spanning tree with lengths $\{d_e\}$ on the complete graph induced by T.

Step 3. Construct a subnetwork of G by replacing each edge in the tree by a corresponding shortest path.

Step 4. Determine a minimum spanning tree in this subnetwork.

Step 5. Delete all Steiner nodes of degree 1 from this tree.

Let z^H be the cost of the heuristic solution and let $Z_\Delta^{HK}(T)$ and Z_Δ^{LP} denote the Held and Karp value for the graph $G' = (T, E(T))$ and the optimal linear programming value when we use the distances d_e in place of w_e.

Theorem 6.11 *If $w_e \geq 0$ for all edges e, then $Z \leq z^H \leq (2 - 2/|T|)Z^{LP}$.*

Proof. Since by Proposition 6.10, the optimal value of the Steiner tree problem is the same with the costs w_e and d_e, and z^H is the value of a feasible solution for the triangularized problem, $Z \leq z^H$.

Let x^* be an optimal solution of the Held and Karp relaxation with the shortest path distances d_e. It is easy to show that x^* also satisfies the conditions $\sum_{e \in E(S)} x_e^* \leq |S| - 1$ for all $S \subset T$, and $\sum_{e \in E(T)} x_e^* = |T|$. Now $\bar{x} = (1 - 1/|T|)x^*$ satisfies $\sum_{e \in E(S)} x_e \leq |S| - 1$ for all $S \subset T$, $\sum_{e \in E(T)} x_e = |T| - 1$, and $x \geq 0$, so $\bar{x}$ lies in the spanning tree polytope (see Section 3) and thus $w\bar{x} \geq z^H$. However Proposition 6.9 shows that $Z_\Delta^{HK}(T) = wx^* = 2Z_\Delta^{LP}$. Thus $z^H \leq w\bar{x} = w(1 - 1/|T|)x^* = 2(1 - 1/|T|)Z_\Delta^{LP}$. But since $w \geq d$, $Z_\Delta^{LP} \leq Z^{LP}$, implying the conclusion of the theorem ∎

In Theorem 6.11, we have shown that $Z \leq z^H \leq (2 - 2/|T|)Z^{HK} = (2 - 2/|T|)Z^{LP}$. We could obtain the conclusion of the theorem, without the intermediate use of Z^{HK}, by noting that the linear programming value $Z^{LP}(V \setminus T)$ is the same as the value of the linear programming relaxation of the cutset formulation, without the cardinality constraint $\sum_{e \in E} x_e = n - 1$, of the minimum spanning tree problem on the graph $G' = (T, E(T))$. As we have shown in Section 3.3, the optimal objective value of this problem,

which equals, Z^H is no more than $(2 - 2/|T|)Z^{LP}(V \setminus T)$, the value of the linear program on the graph $G' = (T, E(T))$. Therefore, $Z \leq z^H \leq (2 - 2/|T|)Z^{LP}(V \setminus T)$. But by the parsimonious property, $Z^{LP} = Z^{LP}(V \setminus T)$ and so $Z \leq z^H \leq (2 - 2/|T|)Z^{LP}$.

A Linear Programming/Tree Heuristic for the Node Weighted Steiner Tree Problem.

Various heuristics based upon minimum spanning tree computations also apply to the node weighted Steiner tree problem. We consider the model (6.1)-(6.6) with the objective function

$$z^{NWST} = \min \sum_{e \in E} w_e x_e + \sum_{i \in V} \pi_i (1 - z_i).$$

We assume that $w_e \geq 0$ for all $e \in E$. In this model, $\pi_i \geq 0$ is a penalty incurred if node i is not in the Steiner tree T. Note that we can impose $i \in T$ (or $z_i = 1$) by setting π_i to be very large. As before, we assume that r is a node that must necessarily be in any feasible solution—that is, $z_r = 1$.

Linear Programming/Tree Heuristic for the Node Weighted Steiner Tree Problem.

Step 1. Solve the linear programming relaxation of the cut formulation (the analog of P_{cut} in Section 3).

$$\begin{aligned} \min \sum_{e \in E} w_e x_e \quad &+ \quad \sum_{i \in V} \pi_i (1 - z_i) \\ \text{subject to} & \\ \sum_{e \in \delta(S)} x_e \quad &\geq \quad z_i \text{ for all } i \text{ and } S \text{ with } r \notin S, i \in S \\ 0 \leq z_i \quad &\leq \quad 1 \text{ for all } i \in V \\ x_e \quad &\geq \quad 0 \text{ for all } e \in E. \end{aligned}$$

Let (x^*, z^*) be an optimal solution to this problem.

Step 2. Let $U = \{i : z_i^* \geq 2/3\}$.

Step 3. Apply the tree heuristic for the Steiner tree problem with the terminal nodes U, producing a heuristic solution $(\tilde{x}, \tilde{z})$ with value z^H.

Theorem 6.12 *Let Z^{NWST} denote the optimal objective value for the node weighted Steiner tree problem and let z^H denote the objective value of the linear programming/tree heuristic solution when applied to the problem. Then $z^H/z^{NWST} \leq 3$.*

Proof. First observe that if $Q = \{i : \tilde{z}_i = 1\}$ is the set of nodes in the heuristic solution, then $Q \supseteq U$ and so

$$\begin{aligned}
\sum_{i \in V} \pi_i(1 - \tilde{z}_i) &= \sum_{i \in V \setminus Q} \pi_i \\
&\leq \sum_{i \in V \setminus U} \pi_i \\
&\leq 3 \sum_{i \in V \setminus U} \pi_i(1 - z_i^*) \\
&\leq 3 \sum_{i \in V} \pi_i(1 - z_i^*).
\end{aligned}$$

The second inequality uses the definition of U, and the third inequality the nonnegativity of π.

Now let $\bar{x} = \frac{3}{2}x^*$. Observe that if $i \in U \setminus \{r\}$ and $S \ni i$, then

$$\sum_{e \in \delta(S)} \bar{x}_e = \frac{3}{2} \sum_{e \in \delta(S)} x_e^* \geq \frac{3}{2} z_i^* \geq 1,$$

so $\bar{x}$ is a feasible solution of the cut formulation

$$\begin{aligned}
z^{LP}(U) &= \min \sum_{e \in E} w_e x_e \\
&\text{subject to} \\
\sum_{e \in \delta(S)} x_e &\geq 1 \text{ for all } i \text{ and } S \text{ with } r \notin S, i \in U \cap S \\
x_e &\geq 0
\end{aligned}$$

and thus $z^{LP}(U) \leq w\bar{x}$. Also, by Theorem 6.11 $w\tilde{x} \leq 2z^{LP}(U)$ and so $w\tilde{x} \leq 2w\bar{x} = 3wx^*$. Thus, $z^H = w\tilde{x} + \sum_{i \in V} \pi_i(1 - \tilde{z}_i) = w\tilde{x} + \sum_{i \in V \setminus Q} \pi_i \leq 3wx^* + 3\sum_{i \in V} \pi(1 - z_i^*) \leq 3z^{NWST}$. ∎

7 Packing Subtrees of a Tree

In Section 2 we saw how to view several problems—lot-sizing, facility location, and vehicle routing— as packing subtrees of a graph. The special case when the underlying graph is itself a tree is of particular interest; we treat this problem in this section. Our discussion builds upon our investigation in Section 4 of the subproblem of finding a best (single) optimal subtree of a tree.

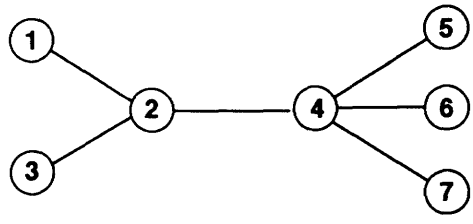
Our principal results will be threefold: (i) to show how to use the type of dynamic programming algorithm we developed in Section 4 to solve the more general problem of packing subtrees of a tree, (ii) to discuss several algorithms for solving problems of packing trees in a tree, and (iii) to understand if and when we can obtain integral polyhedra, or tight linear programming formulations. In the next section we develop extensions of the algorithms introduced in this section to tackle harder problems on general graphs.

7.1 Simple Optimal Subtree Packing Problem

We start by examining the simplest problem. Given a tree $G = (V, E)$, suppose we are given a finite set $F^1, \ldots, F^q$ of subtrees, $F^i \subseteq V$, with associated values $c_1, \ldots, c_q$. The *simple optimal subtree packing problem* (SOSP) is to find a set of node disjoint subtrees of maximum value. Suppose that A is the node-subtree incidence matrix, i.e., $a_{ij} = 1$ if node i is in subtree F^j, and $a_{ij} = 0$ otherwise. Figure 17 shows an example of such a matrix. Letting $\lambda_j = 1$ if we choose subtree F^j, and $\lambda_j = 0$ otherwise, and letting λ be the vector (λ_j), we can formulate problem (SOSP) as the following optimization model:

$$\max\{\sum_j c_j\lambda_j : A\lambda \leq 1, \lambda \in \{0,1\}^q\}.$$

To describe a dynamic programming algorithm, we first introduce some notation. Given the tree G, we arbitrarily choose a root node r and thereby obtain a partial ordering $(V, \preceq)$ on the nodes, by defining $u \preceq v$ if and only if node u lies on the path (r, v). We define the predecessor $p(u)$ of node u as the last node before u on the path (r, u), $S(u) = \{w : p(w) = u\}$ as the set of successors of node u, and $S(F^j)$ as the set of successors of subtree F^j, i.e.,



A tree graph G=(V,E)

Nodes ↓	1	2	3	4	5	6	7	8	← Subtrees
1	1	0	0	0	0	0	1	1	
2	1	1	1	0	0	1	1	1	
3	1	0	0	0	0	1	1	0	
4	1	1	1	1	0	1	0	0	
5	0	1	0	1	0	0	0	0	
6	0	0	1	0	0	0	0	0	
7	0	1	0	1	1	0	0	0	
c values →	4	5	3	3	1	2	3	2	

A node-subtree incidence matrix

Figure 17: Optimal Subtree Packing Problem

$S(F^j) = \{w : p(w) \in F^j, w \notin F^j\}$. We also define the root $r(F^j)$ of F^j to be the unique node in F^j satisfying the condition that $r(F^j) \preceq u$ for all $u \in F^j$.

For the example shown in Figure 17, with node 1 as the root, $p(1) = \phi, p(2) = 1, p(3) = 2,$, and so forth; the set of successors of node 2 is $S(2) = \{3, 4\}$. The set of successors of the subtree F^2 on the nodes $2, 4, 5, 7$, is $S(F^2) = \{3, 6\}$ and the root of this tree is $r(F^2) = 2$.

Algorithm for the SOSP Problem

The algorithm is based on the following recursion:

$$H(u) = \max\{\sum_{w \in S(u)} H(w), \max_{\{j: r(F^j)=u\}} [c_j + \sum_{w \in S(F^j)} H(w)]\}.$$

In this expression, $H(u)$ is the value of an optimal packing of the subtree induced by the node set $V^u = \{v : u \preceq v\}$ and the set of subtrees $\{F^j\}$ with $F^j \subseteq V^u$. The recursion follows from the observation that in an optimal solution of value $H(u)$,
i) if node u is not in any subtree, the solution is composed of optimal solutions for each subgraph induced by V^w with $w \in S(u)$. Thus $H(u) = \sum_{w \in S(u)} H(w)$; or
ii) if node u is in one of the subtrees $F^j \subseteq V^u$, then necessarily $r(F^j) = u$, and the optimal solution must be composed of F^j and optimal solutions for each subtree induced by V^w with $w \in S(F^j)$. Thus $H(u) = \max_{\{j: r(F^j)=u\}}[c_j + \sum_{w \in S(F^j)} H(w)]\}$

Starting from the leaves and working in towards the root, the dynamic programming algorithm recursively calculates the values $H(v)$ for all $v \in V$. $H(r)$ is the value of the optimal solution. To obtain the subtrees in an optimal solution, we iterate back from the root r to see how the algorithm obtained the value $H(r)$. ∎

Example 7.1.

We consider the (SOSP) problem instance shown in Figure 17 with node 1 chosen as the root. Working in towards the root, the algorithm gives:

$$\begin{aligned} H(7) &= \max\{0, c_5\} = 1 \\ H(6) &= 0 \end{aligned}$$

$$
\begin{aligned}
H(5) &= 0 \\
H(3) &= 0 \\
H(4) &= \max\{H(5) + H(6) + H(7), c_4 + H(6)\} = 3 \\
H(2) &= \max\{H(3) + H(4), c_2 + H(3) + H(6), c_3 + H(3) + H(5) + H(7), \\
&\quad c_6 + H(5) + H(6) + H(7)\} = 5 \\
H(1) &= \max\{H(2), c_1 + H(5) + H(6) + H(7), c_7 + H(4), \\
&\quad c_8 + H(3) + H(4)\} = 6.
\end{aligned}
$$

Thus the optimal solution value is 6. To obtain the corresponding solution, we observe that $H(1) = c_7 + H(4), H(4) = c_4 + H(6), H(6) = 0$, so subtrees 7 and 4 give an optimal packing of value 6.

The linear program $\max\{\sum_j c_j\lambda_j : A\lambda \leq 1, \lambda \geq 0\}$ has a primal solution with $\lambda_4 = \lambda_7 = 1$, and $\lambda_j = 0$ otherwise. Observe that if we calculate the values $\pi_u = H(u) - \sum_{w \in S(u)} H(w)$ for $u \in V$, i.e., $\pi_1 = H(1) - H(2) = 6 - 5 = 1$, etc., $\pi = (1, 2, 0, 2, 0, 0, 1)$ is a dual feasible solution to this linear program and its objective value $\sum_{j \in V} \pi_j$ equals $H(r) = 6$. ∎

It is easy to see that this observation concerning the dual variables $\pi_u = H(u) - \sum_{w \in S(u)} H(w)$ holds in general. The recursion for $H(u)$ implies that $\pi_u \geq 0$. For a tree F^j with $r(F^j) = u, \sum_{v \in F^j} \pi_v = \sum_{v \in F^j}(H(v) - \sum_{w \in S(v)} H(w) = H(u) - \sum_{w \in S(F^j)} H(w)$, and the recursion implies that the last term is greater than or equal to c_j. This observation permits us to verify that the algorithm is correct, and the primal-dual argument used in Sections 3 and 4 immediately leads to a proof of an integrality result.

Theorem 7.1 *Given a family of subtrees of a tree, if A is the corresponding node-subtree incidence matrix, then the polyhedron $\{x : Ax \leq \mathbf{1}, x \geq 0\}$ is integral.*

7.2 More General Models

In Section 2, we described three problems of packing subtrees of a tree, namely the lot-sizing problem, the facility location problem with the contiguity property, and the telecommunications application shown in Figure 2.3.

In each of these application contexts, we are not given the subtrees and their values explicitly; instead, typically the problem is defined by an exponential family of subtrees, each whose value or cost we can calculate. We now consider how to model and solve such problems.

The Optimal Subtree Packing Problem.

The Optimal Subtree Packing Problem (OSP) is defined by a tree $G = (V, E)$, families $\mathcal{F}^k$ of subtrees associated with each node $k \in V$, and a value function $c^k(F)$ for $F \in \mathcal{F}^k$. Each nonempty tree of $\mathcal{F}^k$ contains node k. We wish to choose a set of node disjoint subtrees of maximum value, selecting at most one tree from each family.

(OSP) differs from (SOSP) in that neither the subtrees in each family $\mathcal{F}^k$ nor their costs $c^k(F)$ need be given explicitly. We now consider how to model the three problems mentioned previously as OSP problems. For the first two problems, the objective is a linear function, that is a function of the form $c^k(F) = \sum_{j \in F} c^k_j$.

Uncapacitated Lot-Sizing (ULS).

Given a finite number of periods, demands $\{d_t\}_{t=1}^T$, production costs $\{p_t\}_{t=1}^T$, storage costs $\{h_t\}_{t=1}^n$ (which can be transformed by substitution to be zero without any loss of generality), and set-up (or fixed) costs $\{f_t\}_{t=1}^T$ (if production is positive in the period), find a minimum cost production plan that satisfies the demands. From Theorem 2.1 (see Figure 7), we know that this problem always has a directed spanning tree solution. Taking the tree graph to be a path from node 1 to node $T = |V|$, the family of subpaths $\mathcal{F}^k$ associated with node k are of the form $(k, k+1, \ldots, t)$ for $t = k, \ldots, T$ corresponding to a decision to produce the demand for periods k up to t in period k. The costs are $c^k_k = f_k + p_k d_k$, $c^k_j = p_k d_j$ for $j > k$ and $c^k_j = \infty$ for $j < k$.

Facility Location on a Tree.

Given a tree graph, edge weights α_e for $e \in E$, a distance function $d_{ij} = \sum_{e \in Path(i,j)} \alpha_e$ and weights f_j for $j \in V$, locate a set of depots $U \subseteq V$ and assign each node to the nearest depot to minimize the sum of travel distances and node weights: $\min_{U \subseteq V} \{\sum_{j \in U} f_j + \sum_{i \in V} (\min_{j \in U} d_{ij})\}$. Here we

take $c_k^k = f_k$ and $c_j^k = d_{kj}$ for $j \neq k$.

Each of the previous two applications is a special case of an (OSP) problem with linear costs that we can formulate as the following integer program:

$$\max\{\sum_{k\in V}\sum_{j\in V} c_j^k x_j^k : \sum_{k\in V} x_j^k \leq 1 \text{ for } j \in V, x^k \in X^k \text{ for } k \in V\}.$$

In this model, X^k is the set of node incidence vectors of all subtrees rooted at node k plus the vector 0 corresponding to the empty tree. We can interprete the coefficient c_j^k in this model as the cost of assigning node j to a subtree rooted at node k. In practice, frequently node k will contain a "service" facility: c_k^k will be the fixed cost of establishing a service facility at node k and c_j^k will be the cost of servicing node j from node k.

In Section 4 we showed that if $p(j,k)$ denotes the predecessor of node j on the path from node j to node k, then $conv(X^k) = \{x^k \in R_+^V : x_k^k \leq 1, x_j^k \leq x_{p(j,k)}^k \text{ for } j \neq k\}$. Thus, we obtain the formulation

$$\max \sum_{k\in V} c^k x^k \tag{7.1}$$

subject to

$$\sum_{k\in V} x_j^k \leq 1 \text{ for } j \in V \tag{7.2}$$

$$x_j^k \leq x_{p(j,k)}^k \text{ for } j \neq k, k \in V \tag{7.3}$$

$$x_j^k \geq 0 \text{ for } j,k \in V \tag{7.4}$$

$$x_j^k \text{ integer for } j,k \in V. \tag{7.5}$$

Later in this section, we consider the effectiveness of this formulation, particularly the strength of its linear programming relaxation.

A More Complex Model

The telecommunications example requires a more intricate modeling approach since the objective function $c^k(F)$ is nonlinear with respect to the nodes in the subtree F. In addition as we have seen in Section 4.2, in many models it is natural to consider situations in which some of the subtrees rooted at node k are infeasible. We now describe a formulation that allows us to handle both these generalizations.

This formulation contains a set X^k of incidence vectors of the subtrees in $\mathcal{F}^k$. We let x^k be a an incidence vector of a particular subtree rooted at node k. The formulation also contains a set of auxiliary variables w^k that model other aspects of the problem. For example, for the telecommunications problem, the variables w^k correspond to the flow and capacity expansion variables associated with a subtree $F \in \mathcal{F}^k$ defined by x^k. In Section 8 we consider other applications: for example, in a multi-item production planning problem, the variables x^k indicate when we produce product k and the variables w^k model the amount of product k that we produce and hold in inventory to meet customer demand. In this more general problem setting, we are given a set W^k that models the feasible combinations of the x^k and w^k variables. We obtain the underlying trees from W^k by projecting out the w^k variables, that is, $proj_{x^k}(W^k) = X^k$

For any particular choice x^k of the x variables, let $c^k(x^k) = \max\{e^k x^k + f^k w^k : (x^k, w^k) \in W^k\}$ denote the optimal value of the tree defined by x^k obtained by optimizing over the auxiliary variables w. Once again, we assume that $0 \in X^k$ and $c^k(0) = 0$. We are interested in solving the following *optimal constrained subtree packing problem (OCSP)*:

$$\max \sum_k e^k x^k + \sum_k f^k w^k \tag{7.6}$$

subject to

$$\sum_{k \in V} x_j^k \leq 1 \text{ for } j \in V \tag{7.7}$$

$$(x^k, w^k) \in W^k \text{ for } k \in V. \tag{7.8}$$

This modeling framework permits us to consider applications without auxiliary variables as well. In this case, $W^k = X^k$ can model constraints imposed upon the x variables and so only a subset of the subtrees rooted at k will be feasible. In Section 4.2 we considered one such application, a capacitated model with knapsack constraints of the form: $\sum_{j \in V} d_j x_j^k \leq C$.

7.3 Algorithmic Approaches

We briefly outline four algorithmic strategies for solving OCSP.

Algorithm A. Primal Column Generation Algorithm using Subtree Optimization.

We know from Theorem 7.1 that if we could explicitly write out all feasible subtrees and calculate their values for each $k \in V$, the resulting linear program

$$z^A = \max\{\sum_k c^k \lambda^k : \sum_k A^k \lambda^k \leq 1, \lambda^k \geq 0 \text{ for } k \in V\}, \tag{7.9}$$

called the *Master Problem*, or its corresponding integer program with λ^k integer, would solve OCSP. In this model, A^k is the node-tree incidence vector for all feasible subtrees rooted at node k and $\lambda^k = (\lambda_j^k)$ is vector that, when integer, tells which tree we choose (i.e., if $\lambda_j^k = 1$ we choose the tree with incidence vector A_j^k). $c^k = (c_j^k)$ is a vector of tree costs; that is, c_j^k is the cost of the tree with the incidence vector A_j^k. When the problem has auxiliary variables w, $c_j^k = c^k(A^k) = \max\{e^k A^k + f^k w^k : (A^k, w^k) \in W^k\}$ is the optimal cost of the tree rooted at node k with the incidence vector A^k.

Typically, this linear program is impractical to formulate explicitly because of the enormous number of subtrees and/or the difficulty of evaluating their values. Therefore, we might attempt to solve it using the idea of a column generation algorithm; that is, work with just a subset of the columns (subtrees), and generate missing ones if and when we need them.

At iteration t, we have a *Restricted Master Problem:*

$$\begin{aligned} \max & \sum_{k \in V} c^{k,t} \lambda^{k,t} \\ & \text{subject to} \\ & \sum_{k \in V} A^{k,t} \lambda^{k,t} \leq 1 \\ & \lambda^{k,t} \geq 0. \end{aligned}$$

In this model each $A^{k,t}$ is the incidence matrix of some of the feasible subtrees rooted at node k with an associated cost $c^{k,t}$, and so $A^{k,t}$ is a sub-

matrix of A^k. $\lambda^{k,t}$ denotes the corresponding subvector of the vector λ. Let π^t be optimal dual variables for Restricted Master linear program. Since the Restricted Master Problem contains only a subset of the feasible trees, we would like to know whether we need to consider other subtrees or not.

Let x^k denote the incidence vector of a generic column of A^k corresponding to a feasible subtree rooted at node k. The subproblem for each k is:

$$\mu_k^t = \max\{e^k x^k + f^k w^k - \pi^t x^k, (x^k, w^k) \in W^k\}.$$

If $\mu_k^t \leq 0$ for all $k \in V$, linear programming theory tells us that we have an optimal solution of the Master Problem.

However, if $\mu_k^t > 0$ for some k, then we add one or more subtrees to the Restricted Master Problem, update the matrices giving $A^{k,t+1}, c^{k,t+1}$, and pass to iteration $t+1$. ∎

Because of Theorem 7.1, we observe that

i) the Restricted Master Problem produces an integer feasible solution at each iteration (that is, each $\lambda^{k,t}$ is a 0-1 vector).

ii) the Restricted Master Problem is an (SOSP) problem, so we can solve it using the dynamic programming algorithm presented at the beginning of this section, rather than using the Simplex algorithm.

Since, as we have already noted, Theorem 7.1 implies that the Master Problem (as a linear program) is equivalent to OCSP (an integer program), we have the following result:

iii) the algorithm terminates with an optimal solution of OCSP.

Algorithm B. Lagrangian Relaxation.

As we have seen in Section 3, Lagrangian relaxation is an algorithmic approach for finding a good upper bound (a lower bound in that discussion since we were minimizing) for a maximization problem by introducing some of the problem constraints into the objective function with associated prices (also called dual variables or Lagrange multipliers). Specifically, we start

with the formulation (7.6)-(7.8) of OCSP. Dualizing the packing constraints (7.7) leads to the so-called Lagrangian subproblem:

$$L(\pi) = \max \sum_{k \in V} (e^k x^k \quad + \quad f^k w^k - \pi x^k) + \sum_{j \in V} \pi_j$$
$$\text{subject to } (x^k, w^k) \quad \in \quad W^k \text{ for all } k.$$

Observe that the Lagrangian subproblem separates into independent subproblems for each k, namely,

$$\mu_k = \max\{e^k x^k + f^k w^k - \pi x^k, (x^k, w^k) \in W^k\}.$$

Thus,

$$L(\pi) = \sum_{j \in V} \pi_j + \sum_k \mu_k.$$

To obtain a "best" upper bound, we solve the Lagrangian dual problem:

$$z^B = \min_{\pi \geq 0} L(\pi).$$

We can find an optimal value for π by using standard algorithms from the theory of Lagrangian relaxation (e.g., subgradient optimization) that generate values π^t for π at each iteration. ∎

What can be said about this algorithm? To resolve this question, we first observe that we can rewrite the Master Problem (7.9) as

$$z^A = \max\{\sum_k c^k \lambda^k : \sum_k A^k \lambda^k \leq \mathbf{1}, \mathbf{1}\lambda^k = 1 \text{ for } k \in V, \lambda^k \geq 0 \text{ for } k \in V\}.$$

This model contains additional constraints $\mathbf{1}\lambda^k = 1$ for $k \in V$. Since $0 \in X^k$ with cost 0, and since every nonempty tree $x^k \in X^k$ contains node k, the kth constraint of $\sum_k A^k \lambda^k \leq \mathbf{1}$ implies that $\mathbf{1}\lambda^k = 1$ for $k \in V$ is redundant. More precisely, the row of the node-subtree incidence matrix A^k corresponding to node k has $+1$ for each subtree in X^k and so the row corresponding to this constraint implies that $\sum_j \lambda^{k,j} \leq 1$. Note that $1 - \sum_j \lambda^{k,j}$ is the weight placed on the null tree in X^k.

Linear programming theory shows that the optimal value of the linear programming relaxation of this modified Master Problem, equals the value of its linear program dual, that is,

$$z^A = \min\{\sum_i \pi_i + \sum_k \mu_k : \pi A^k + \mu_k \mathbf{1} \geq c^k \text{ for all } k, \pi \geq 0\}$$
$$= \min_{\pi \geq 0, \mu} \{\sum_i \pi_i + \sum_k \mu_k : \mu_k \geq c(x^k) - \pi x^k \text{ for all } x^k \in X^k\}$$
$$= \min_{\pi \geq 0} L(\pi).$$

The final equality is due to the fact that for a given value of π, the optimal choice of each μ_k is given by $\mu_k = \max_{x^k \in X^k}\{c(x^k) - \pi x^k\} = \max\{e^k x^k + f^k w^k - \pi x^k : (x^k, w^k) \in W^k\}$ and so the objective function $\sum_{i \in V} \pi_i + \sum_k \mu_k$ is equal to $L(\pi)$.

This discussion shows that $z^A = z^B$, and so

i) the optimal value of the Lagrangian dual gives the optimal value of the OCSP problem, and

ii) the optimal solution π of the Lagrangian dual is an optimal dual solution for the Master Problem (7.9).

Algorithm C. Dual Cutting Plane Algorithm using Subtree Separation.

The goal in this approach is to implicitly solve the linear program

$$z^C = \max \sum_k e^k x^k + \sum_k f^k w^k \tag{7.10}$$
$$\text{subject to}$$
$$\sum_k x^k \leq \mathbf{1} \tag{7.11}$$
$$(x^k, w^k) \in \text{conv}(W^k) \text{ for all } k \tag{7.12}$$

by a cutting plane algorithm.

Let $\{(x^k, w^k) : G^k x^k + H^k w^k \leq b^k, 0 \leq x^k_j \leq 1, w^k \geq 0 \text{ for all } k, j \in V\}$ be a representation of $\text{conv}(W^k)$ by a system of linear inequalities. Since it is impractical to write out all the constraints explicitly, we work with a subset of them, and generate missing ones if and when needed.

At iteration t, we therefore have a relaxed linear program:

$$\begin{aligned} \max \sum_{k \in V} (e^k x^k &+ f^k w^k) \\ \text{subject to} & \\ \sum_{k \in V} x^k &\leq 1 \\ G^{k,t} x^k + H^{k,t} w^k &\leq b^{k,t} \text{ for } k \in V \\ 0 \leq x_j^k \leq 1, w^k &\geq 0 \text{ for } k, j \in V \end{aligned}$$

After finding a solution $(x^{k,t}, w^{k,t})$ to the linear programming relaxation, we then solve the separation problem for each k, i.e., we check to see if $(x^{k,t}, w^{k,t}) \in \text{conv}(W^k)$ or not.. If $(x^{k,t}, w^{k,t}) \in \text{conv}(W^k)$ for all k, the algorithm terminates. If not, we obtain one or several violated inequalities that we add to the formulation. We then update the matrices giving $G^{k,t+1}, H^{k,t+1}, b^{k,t+1}$, and pass to iteration $t+1$. ∎

Note that if we always generate a violated facet-defining inequality of $conv(W^k)$, the cutting plane algorithm will terminate in a finite number of iterations having satisfied all the constraints in $G^k x^k + H^k w^k \leq b^k$, even though we have not written all the constraints explicitly.

How does the optimal objective function for this approach compare with the values we obtain from the column generation and Lagrangian relaxation procedures? Linear programming duality (results on "partial duality") implies that

$$z^C = \min_{\pi \geq 0} \{ \sum_i \pi_i + \sum_k \max_{x^k, w^k} \{ e^k x^k + f^k w^k - \pi x^k : (x^k, w^k) \in conv(W^k) \} \}$$

But since optimal solutions of linear programs occur at extreme points, this optimization problem has the same value as:

$$\min_{\pi \geq 0} \{ \sum_i \pi_i + \sum_k \max_{x^k, w^k} \{ e^k x^k + f^k w^k - \pi x^k : (x^k, w^k) \in W^k \} \}$$

which is just the Lagrangian dual problem.

Thus, we have shown that $z^A = z^B = z^C$. Stated differently, we have shown that for any objective function coefficients $(e^1, f^1, \ldots, e^n, f^n)$, the linear programming value z^C of (7.10)-(7.12) equals the optimal value of the corresponding (OCSP) integer program (7.6)-(7.8). This implies that

i) the polyhedron (7.11)-(7.12) is integral (i.e. in all its extreme points $(x^k, w^k) \in W^k$), and thus

ii) Algorithm C terminates with such an extreme point, and thus it gives an optimal solution to (OCSP).

In summary, this discussion shows that if we (i) solve the linear programming relaxation of the column generation model, (ii) solve the Lagrangian dual problem, or (iii) use the cutting algorithm until we have a point lying in $\text{conv}(W^k)$ for all k, then we obtain the same optimal objective function values: $z^A = z^B = z^C$. Algorithms A and C terminate with an optimal solution to (OCSP) since they work in the space of x and w variables. Lagrangian relaxation provides us with an optimal set of dual prices π, which we can then use to find an optimal solution x and w by solving a linear programming feasibility problem (we will not provide the details.)

Example 7.2. We solve an instance of the OCSP problem for the graph

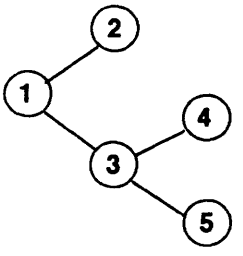


Figure 18: Tree for the OCSP problem instance

shown in Figure 18. The initial profit matrix is

$$\begin{matrix} 6 & 2 & 3 & 4 & 1 \\ 9 & 2 & 7 & 6 & 0 \\ 8 & 2 & 1 & 4 & 9 \\ 2 & 1 & 3 & 4 & 4 \\ 1 & 6 & 2 & 9 & 1. \end{matrix}$$

The entry c_j^k in row k and column j is the value if node j is in a subtree rooted at node k. All subgraphs are feasible.

We use the column generation algorithm starting with an initial set of

five single node subtrees. Thus, the initial Master Problem is of the form:

$$\begin{aligned}
&\max 6\lambda_1^1 + \quad 2\lambda_1^2 + 1\lambda_1^3 + 4\lambda_1^4 + 1\lambda_1^5 \\
&\text{subject to} \\
&\quad 1\lambda_1^1 + \quad 0\lambda_1^2 + 0\lambda_1^3 + 0\lambda_1^4 + 0\lambda_1^5 \quad \leq 1 \\
&\quad 0\lambda_1^1 + \quad 1\lambda_1^2 + 0\lambda_1^3 + 0\lambda_1^4 + 0\lambda_1^5 \quad \leq 1 \\
&\quad 0\lambda_1^1 + \quad 0\lambda_1^2 + 1\lambda_1^3 + 0\lambda_1^4 + 0\lambda_1^5 \quad \leq 1 \\
&\quad 0\lambda_1^1 + \quad 0\lambda_1^2 + 0\lambda_1^3 + 1\lambda_1^4 + 0\lambda_1^5 \quad \leq 1 \\
&\quad 0\lambda_1^1 + \quad 0\lambda_1^2 + 0\lambda_1^3 + 0\lambda_1^4 + 1\lambda_1^5 \quad \leq 1 \\
&\qquad\qquad\qquad\qquad \lambda \qquad\qquad\quad \geq 0.
\end{aligned}$$

Solving the Restricted Master by linear programming (or the dynamic programming algorithm for (SOSP), we obtain

$$H(5) = 1, H(4) = 4, H(3) = 6, H(2) = 2, H(1) = 14,$$

and dual variables $\pi^1 = (6, 2, 1, 4, 1)$.

The subproblem profit matrix $\bar{c}^1 = (c_j^k - \pi_j^1)$ is now

$$\begin{array}{ccccc} 0 & 0 & 2 & 0 & 0 \\ 3 & 0 & 6 & 2 & -1 \\ 2 & 0 & 0 & 0 & 8 \\ -4 & -1 & 2 & 0 & 3 \\ -5 & 4 & 1 & 5 & 0. \end{array}$$

Thus, we have modified the original values c_j^k by a cost π_j if node j appears in the subtree. For instance $\bar{c}_{43}^1 = c_{43} - \pi_3 = 3 - 1 = 2$. Solving the $k = 1, \ldots, 5$ subproblems, we obtain $\mu_1^1 = 2, \mu_2^1 = 11, \mu_3^1 = 10, \mu_4^1 = 5$, and $\mu_5^1 = 6$. Introducing the column with the greatest reduced price of 11, namely $F^2 = \{1, 2, 3, 4\}$ rooted at 2 of value 24, we update the Restricted Master problem, obtaining

$$\begin{aligned}
&\max 6\lambda_1^1 + \quad 2\lambda_1^2 + 1\lambda_1^3 + 4\lambda_1^4 + 1\lambda_1^5 + 24\lambda_2^2 \\
&\text{subject to} \\
&\quad 1\lambda_1^1 + \quad 0\lambda_1^2 + 0\lambda_1^3 + 0\lambda_1^4 + 0\lambda_1^5 + 1\lambda_2^2 \quad \leq 1 \\
&\quad 0\lambda_1^1 + \quad 1\lambda_1^2 + 0\lambda_1^3 + 0\lambda_1^4 + 0\lambda_1^5 + 1\lambda_2^2 \quad \leq 1 \\
&\quad 0\lambda_1^1 + \quad 0\lambda_1^2 + 1\lambda_1^3 + 0\lambda_1^4 + 0\lambda_1^5 + 1\lambda_2^2 \quad \leq 1 \\
&\quad 0\lambda_1^1 + \quad 0\lambda_1^2 + 0\lambda_1^3 + 1\lambda_1^4 + 0\lambda_1^5 + 1\lambda_2^2 \quad \leq 1 \\
&\quad 0\lambda_1^1 + \quad 0\lambda_1^2 + 0\lambda_1^3 + 0\lambda_1^4 + 1\lambda_1^5 + 0\lambda_2^2 \quad \leq 1 \\
&\qquad\qquad\qquad\qquad \lambda \qquad\qquad\quad \geq 0.
\end{aligned}$$

For this problem,

$$H = (25, 2, 6, 4, 1) \text{ and } \pi^2 = (17, 2, 1, 4, 1).$$

Returning to the subproblems, $\mu_5^2 = 12$ is the maximum violation, so we next introduce the subtree $F^5 = \{3, 4, 5\}$ with value 12.

On the third iteration, the Restricted Master gives

$$H = (25, 2, 12, 4, 1) \text{ and } \pi^3 = (11, 2, 7, 4, 1).$$

The subproblem cost matrix $\bar{c}^3 = (c_j^k - \pi_j^3)$ is now

-5	0	-9	0	0	
-2	0	-5	2	-1	
-3	0	-11	0	8	All
-9	-1	-9	0	3	
-10	4	-10	5	0.	

the subproblems have an optimal value of 0, so the corresponding primal solution $\{1, 2, 3, 4\}$ rooted at 2 and $\{5\}$ rooted at 5 with value $H(1) = 25$ is optimal. ∎

Algorithm D. Dynamic Programming.

Dynamic programming provides another algorithmic approach for solving (OSP) with linear costs (that is, the model (7.1)-(7.3)). The algorithm is a more complicated version of the dynamic program for solving (SOSP) in that it implicitly treats an exponential number of subtrees. To develop one such algorithm, let u be any node of the given tree T and let $T(u)$ be the subtree of T rooted at node u. Any feasible solution to (OSP) contains (i) subtrees that are disjoint from $T(u)$, (ii) subtrees contained in $T(u)$, and (iii) at most one subtree T^k, rooted as some node k, that has nodes both in and out of $T(u)$. Note that in case (iii), $T(u) \cap T^k$ is a subtree of $T(u)$ that contains node u. Let $C(u)$ denote the assignment values of the nodes in the subtree $T(u)$, that is, $C(u) = \sum_{j \in T(u)} c_j^{k(j)}$. In this expression, $k(j)$ denotes the node to which the solution assigns node j. If the solution does not assign node j to any node, we set $k(j) = 0$. In this case, $c_j^0 = 0$. Among all feasible solutions to the problem, let $H(u, k)$ denote maximum value of $C(u)$, given that the solution assigns node u to node k (note that node k might or might not belong to $T(u)$).

Let $H(u)$ denote the value of the maximum packing of the tree $T(u)$, that is, the value of (OSP) restricted to the subtree $T(u)$. Let r be a root node of the given tree T. We wish to find $H(r)$.

As before, $S(u)$ denotes the successors of node u in T. We can find $H(u,k)$ for all nodes u and k by working towards the root from the leaves of T using the following recursion:

$$
\begin{aligned}
H(u,k) &= c_u^k + \sum_{w \in S(u)} \max\{H(w,k), H(w)\} \text{ if } k = u \text{ or } k \notin T(u) \\
H(u,k) &= c_u^k + H(\bar{w},k) + \sum_{w \in S(u), w \neq \bar{w}} \max\{H(w,k), H(w)\} \\
&\qquad\qquad \text{if } k \in T(\bar{w}) \text{ and } \bar{w} \in S(u) \\
H(u,k) &= \sum_{w \in S(u)} H(w) \text{ if } k = 0 \text{ (that is, node } u \text{ belongs to no subtree)} \\
H(u) &= \min_{k \in T(u)} \{H(u,k)\}.
\end{aligned}
$$

Note that this recursion uses the fact that if node w is any successor of node u and a solution assigns node u to itself ($k = u$), to no node, or to a node k not in $T(w)$, then the solution must assign node w to node k, to no node, or to a node in $T(w)$. $H(w,k)$ gives the optimal cost of the first case and $H(w)$ gives the optimal cost of the last two cases. If the solution assigns node u to a node k in $T(\bar{w})$ for one of is successor nodes $barw$, then it must assign node $\bar{w}$ to node k as well.

Note that since this recursion compares $H(w,k)$ to $H(w)$ for any two nodes w and k of T once, for a tree with n nodes, it requires $O(n^2)$ computations.

When each node v has an associated demand $d(v)$ and each subtree is restricted to contain nodes with a total demand of at most C for some positive integer capacity C, we can use an extension of the dynamic programming argument that we introduced in Section 4 for the rooted subtree of a tree problem. In this case, we let $H(u,k,t)$ be the maximum value $C(u)$ of a packing on the subtree $T(u)$, given a capacity of t for the packing, and given that the solution assigns node u to node k. If we use the dynamic programming approach suggested in Section 4, the overall algorithm will require $O(n^2C)$ computations. In the special case in which each demand $d(v)$ is one, and so C is a restriction on the number of nodes in any subtree, the algorithm requires $O(n^3)$ computations. The well-known p-median problem is a variant of this problem; recall that in this case, the solution can contain at most p subtrees. The dynamic programming algorithm for this case is

similar to those for these other cases, using, for example, a recursion with the definition of $H(u, k, q)$ as the maximum value $C(u)$ of the subtree $T(u)$, given that the solution assigns node u to node k and that the packing of $T(u)$ contains (or intersects) at most q subtrees. The resulting algorithm requires $O(n^2p)$ computations.

7.4 Polyhedral Results

Our discussion of algorithms in the previous subsection has shown that when the costs are linear, the linear programming relaxation of this integer program solves (OSP) with linear costs.

Theorem 7.2 *If X^k is the set of all subtrees rooted at node k, and $X = \{(x^1, \ldots, x^n) : \sum_k x^k \leq 1, x^k \in X^k$ for all $k \in V\}$, then conv(X) is described by the constraints (7.2)-(7.4).*

This is an apparently surprising result because it is very unusual that, when a set of integral polyhedra (the *conv*(X^k)) are combined with a set of additional constraints (in this case, the packing constraints $\sum_{k \in V} x_j^k \leq 1$), the resulting polyhedron is integral. As we have seen in our discussion of the column generation algorithm, even though we are dealing with an exponential number of subtrees, the essential reason is again Theorem 7.1.

Our discussion has also established the following more general result.

Theorem 7.3 *If*

$$W = \{(x^1, w^1, \ldots, x^n, w^n) : (x^k, w^k) \in W^k \text{ for all } k \in V, \sum_{k \in V} x_j^k \leq \mathbf{1} \text{ for all } j \in V\},$$

then

$$\text{conv}(W) = \{(x^1, w^1, \ldots, x^n, w^n) : (x^k, w^k) \in \text{conv}(W^k) \text{ for all } k \in V, \sum_{k \in V} x_j^k \leq 1 \text{ for all } j \in V\}$$

This generalization of Theorem 7.2 tells us that except for the packing inequalities, the polyhedron describing the convex hull of solutions to (OCSP)

has no facet-defining inequalities that link the subtrees associated with different nodes k. Thus, it suffices to study each set W^k independently.

Finally, we obtain the important Corollary to Theorem 7.3.

Corollary *There is a polynomial algorithm to optimize over W if and only if there is a polynomial algorithm to optimize over W^k for each k.*

In this section we have shown how the structure of the problem of packing subtrees of a tree leads not only to surprising integrality results, but also allows us to successfully adapt three of the most popular algorithmic strategies from integer programming for solving OCSP. Theorem 7.1 implies that we need not make any distinction between the linear programming and integer programming formulations the tree packing problem when the underlying graph is itself a tree, since the feasible region of the linear program is an integer polyhedron. In the Section 8 we examine the more complicated problem of packing subtrees or subgraphs into a general graph. In this more general setting, the linear programming relaxation of the analogous integer programming model does not have integer extreme points, and so we need to develop extensions of the algorithms presented in this section.

8 Packing Subtrees of a General Graph

In this section we show how to model several of the problems mentioned in Section 2 as problems of packing (or partitioning) subtrees, forests, or subgraphs of an arbitrary graph. Our purpose in this discussion is not to be comprehensive. Rather, we wish to (i) show that packing problems on general graphs arise in a number of important problem settings, (ii) show how to use the models and solution approaches we have considered in the previous sections to begin to address these problems, and (iii) suggest some of the results researchers have obtained for these problems. One lesson will become apparent: the polyhedral representations of these problems can be quite complex.

We again use the basic optimal capacitated subtree formulation

$$z^{PSG} = \max\{\sum_k e^k x^k + \sum_k f^k w^k : \sum_k x_j^k \leq 1, (x^k, w^k) \in W^k \text{ for all } k\}.$$

We refer to this formulation as the *Packing Subtrees in a Graph (PSG)* model since the underlying graph need not be a tree. In this model, x^k again represents the incidence vector of nodes of the kth subgraph (usually a tree or a forest), and w^k typically represents edge or flow variables associated with this subgraph (k no longer necessarily represents a node v). As we show next, several generic application contexts are special cases of this model.

8.1 Applications

1. Multi-Item Lot-Sizing. Suppose we are given demands d_t^k for items $k = 1, \ldots, K$ over a time horizon $t = 1, \ldots, T$. All items must be produced on a single machine, and the machine can produce only one item in each period. Given production, storage and set-up costs for each item in each period, we wish to find a minimum cost production plan. Figure 19 shows the graph for a 13 period problem, as well as the subgraphs of production periods for 3 items (we could choose item K as a dummy item: when the machine is "processing" this item, it is idle). The graph G we consider for this application is a path with K nodes, one for each time period. Each item defines a set of subgraphs on G; each is a set of paths from G. Figure 19 shows the graph and a possible set of subgraphs for a 13 period, 3 item

problem. The production plan corresponding to this solution produces item 1 in periods 1, 2, 9, 12, and 13, item 2 in periods 3, 7, and 8, and item 3 in periods 4-6, 10, and 11. This solution indicates, for example, that in period 2 we produce the demand of item 1 for periods 2 through 8 and carry forward inventory of this item in periods 2 - 7.

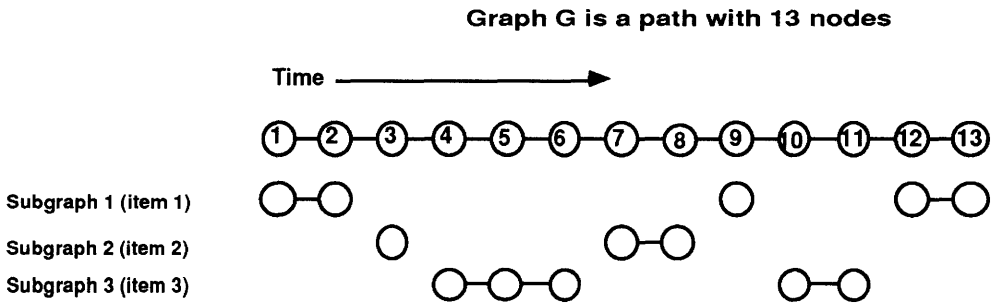


Figure 19: Multi-Item Lot-Sizing as Packing Forests (Paths)

2. Clustering Given a graph $G = (V, E)$, edge costs c_e for $e \in E$, node weights d_i for $i \in V$, positive integers K and C, we wish to find K clusters satisfying the property that the sum of the node weights in each cluster does not exceed C, in a way that minimizes the sum of the weights of edges between clusters (maximizes the sum of weights of edges within clusters). Figure 20 shows a feasible solution with 3 clusters and a capacity of 10.

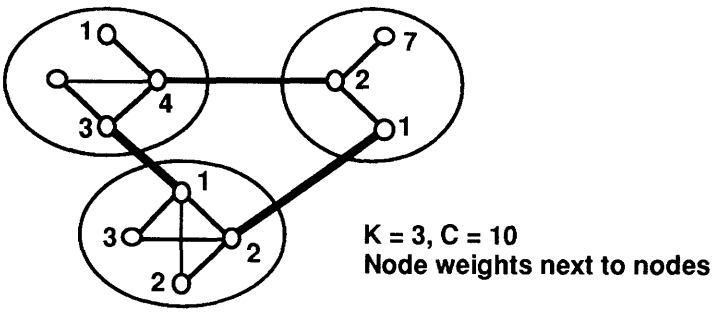


Figure 20: Clustering Solution with Three Clusters

3. The C-capacitated tree problem. Given a graph $G = (V, E)$, a root node 0, edge costs c_e for $e \in E$, find a minimum cost spanning tree in which each subtree on the nodes $V \setminus \{0\}$ contains at most C nodes. That is, if we delete the root node and all its incident edges, the spanning tree decomposes into a forest on the nodes $V \setminus \{0\}$. Each tree in this forest can contain at most C nodes.

4. Capacitated Trees. Given a graph $G = (V, E)$, a root node 0, edge costs c_e for every $e \in E$, positive demands d_i for $i \in V \setminus \{0\}$ and a positive integer C, we wish to find a minimum cost spanning tree satisfying the property that the total demand in each subtree on the nodes $V \setminus \{0\}$ does not exceed C.

5. Capacitated Vehicle Routing. Given a graph $G = (V, E)$, a depot node 0, edge costs c_e for each $e \in E$, K vehicles of capacity C and client orders d_i for $i \in V \setminus \{0\}$, we wish to find a set of tours (cycles) for each vehicle that (i) each contain the depot, (ii) collectively contain all the nodes, (iii) are disjoint on the node set $V \setminus \{0\}$, and (iv) satisfy the property that the total demand on each cycle (that is, total amount delivered by each vehicle) does not exceed C.

6. VLSI Design. The global routing problem in VLSI design can be viewed as a problem of packing Steiner trees (or nets) with packing constraints imposed upon the edges. The problem, defined by a graph $G = (V, E)$, a collection of n terminal sets $T_k \subseteq V$ for $k = 1, 2, \ldots, n$, and edge (channel) capacities u_e, is to find a set of Steiner trees $\{S_k\}_{k=1}^{n}$ so that (i) S_k contains the terminal nodes T_k, and (ii) the number Steiner trees containing edge e does not exceed u_e.

Models 1, 3, and 4 fit into the framework of the general model (PSG), and model 2 fits if we allow general subgraphs. If we remove the depot, model 5 requires the packing of trees (paths) on $G \setminus \{0\}$. Model 6 is somewhat different in that the packing is over the edges rather than the nodes.

8.2 Algorithmic Strategies

How have researchers tackled each of these problems? Before attempting to answer this question, we first briefly discuss three general solution methods

for solving (PSG), each extending the algorithms we considered in the last section.

Algorithm A. Column Generation Algorithm . To correctly model (PSG) we write the Master Problem with the additional "convexity" constraints

$$z^A = \max\{\sum_k c^k\lambda^k : \sum_k A^k\lambda^k \leq \mathbf{1}, \mathbf{1}\lambda^k = 1, \lambda^k \geq 0 \text{ for } k \in V\}.$$

If we associate dual variables (π, σ) with the packing and convexity constraints, the kth subproblem becomes

$$\omega(\pi, \sigma) = \max\{e^k x^k + f^k w^k - \pi^t x^k - \sigma_k, (x^k, w^k) \in W^k\}. \tag{8.1}$$

In this model, A^k is the node-subtree incidence matrix of all feasible edge incidence vectors x^k satisfying the conditions $(x^k, w^k) \in W^k$ for some w^k, and $c^k = c^k(x^k) = \max_{w^k}\{e^k x^k + f^k w^k : (x^k, w^k) \in W^k\}$. When the column generation algorithm terminates, the vector $(\lambda^1, \ldots, \lambda^n)$ might not be integer, and so we might need to use an additional branching phase to solve the problem.

To implement a branching phase, we might wish to form two new problems (branch) by setting some fractional variable λ_i^k to 0 in one problem and to 1 in the other. Typically, however, in doing so we encounter a difficulty: when we set $\lambda_i^k = 0$ and then solve the subproblem (8.1) at some subsequent iteration, we might once again generate the subgraph $S_i \in W^k$ associated with the variable λ_i^k. To avoid this difficulty, we can add a constraint to W^k when solving the subproblem, chosen so that the subtree S_i will not be feasible and so the subproblem will no longer be able generate it. The following inequality will suffice:

$$\sum_{j \in S_i} x_j - \sum_{j \notin S_i} x_j \leq |S_i| - 1$$

since the solution with $x_j = 1$ for all nodes $j \in S_i$ and $x_j = 0$ for all nodes $j \notin S_i$ does not satisfy the inequality.

Unfortunately, this scheme leads to a highly unbalanced enumeration tree (setting the λ_i^k to zero eliminates very few feasible solutions). A better strategy is to choose two subgraphs S_i and S_j whose associated variables λ_i^k and $\lambda_j^{k'}$ are fractional. Consider any pair of nodes u, v satisfying the

conditions $u, v \in S_i, u \in S_j$, but $v \notin S_j$. In an optimal solution either (i) u and v lie in the same subgraph, or (ii) they do not. In the first case, for each subproblem k we can impose the condition $x_u = x_v$; this condition implies that any subgraph S contains either both or neither of u and v. Since $v \notin S_j$, this constraint will eliminate the variable $\lambda_j^{k'}$ corresponding to the set S_j from the formulation. In the second case (ii), all subgraphs satisfy the condition $x_u + x_v \leq 1$; since S_i contains both nodes u and v, this constraint will eliminate the variable λ_i^k corresponding to the set S_i from the formulation. So, imposing the constraints $x_u = x_v$ or $x_u + x_v \leq 1$ on each subproblem permits us to branch as shown in Figure 21.

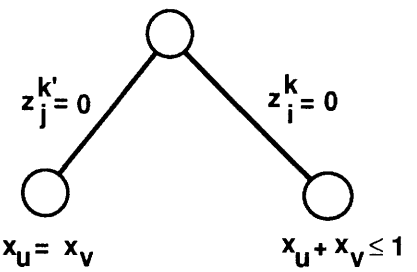


Figure 21: A Branching Scheme

A third, related approach is to branch directly on the original problem variables, i.e., node or edge variables x^k or w^k.

Algorithm B. Lagrangian Relaxation of the Packing Constraints.

As we saw in Section 7, if we attach a Lagrange multiplier π_j to each packing constraint $\sum_k x_j^k \leq 1$, and bring these constraints into the objective function, we obtain a Lagrangian subproblem that decomposes into a separate problem for each k (since the packing constraint was the only constraint coupling the sets W^k). The resulting Lagrangian subproblem becomes $L(\pi) = \sum_k \mu^k(\pi) + \sum_i \pi_i$, and $\mu^k(\pi) = \{\max(e^k - \pi)x^k + f^k w^k : (x^k, w^k) \in W^k\}$.

As before, for each value of the Lagrange multipliers π, the optimal objective value $L(\pi)$ of the Lagrangian subproblem is an upper bound an the optimal objective value of (PSG). To find the multiplier value providing the sharpest upper bound on the optimal objective value, we would solve the

conditions $u, v \in S_i, u \in S_j$, but $v \notin S_j$. In an optimal solution either (i) u and v lie in the same subgraph, or (ii) they do not. In the first case, for each subproblem k we can impose the condition $x_u = x_v$; this condition implies that any subgraph S contains either both or neither of u and v. Since $v \notin S_j$, this constraint will eliminate the variable $\lambda_j^{k'}$ corresponding to the set S_j from the formulation. In the second case (ii), all subgraphs satisfy the condition $x_u + x_v \leq 1$; since S_i contains both nodes u and v, this constraint will eliminate the variable λ_i^k corresponding to the set S_i from the formulation. So, imposing the constraints $x_u = x_v$ or $x_u + x_v \leq 1$ on each subproblem permits us to branch as shown in Figure 21.

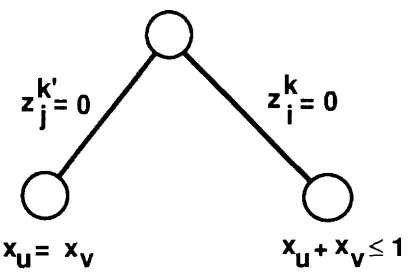


Figure 21: A Branching Scheme

A third, related approach is to branch directly on the original problem variables, i.e., node or edge variables x^k or w^k.

Algorithm B. Lagrangian Relaxation of the Packing Constraints.

As we saw in Section 7, if we attach a Lagrange multiplier π_j to each packing constraint $\sum_k x_j^k \leq 1$, and bring these constraints into the objective function, we obtain a Lagrangian subproblem that decomposes into a separate problem for each k (since the packing constraint was the only constraint coupling the sets W^k). The resulting Lagrangian subproblem becomes $L(\pi) = \sum_k \mu^k(\pi) + \sum_i \pi_i$, and $\mu^k(\pi) = \{\max(e^k - \pi)x^k + f^k w^k : (x^k, w^k) \in W^k\}$.

As before, for each value of the Lagrange multipliers π, the optimal objective value $L(\pi)$ of the Lagrangian subproblem is an upper bound an the optimal objective value of (PSG). To find the multiplier value providing the sharpest upper bound on the optimal objective value, we would solve the

Lagrangian Dual problem stated earlier:

$$z^B = \min_{\pi \geq 0} L(\pi).$$

To implement this approach we need an exact optimization algorithm for solving a linear optimization problem over W^k. We would use standard procedures to solve the Lagrangian dual problem and to continue from its solution using branch and bound.

Algorithm C. A Cutting Plane Algorithm plus Branch and Bound.

One way to approach this problem could be to apply the cutting plane algorithm from the previous section, that is, start with a partial polyhedral representation of each polyhedral set W^k, and solve the linear programming relaxation of the problem. We then check to see if the solution to this problem satisfies all of the constraints defining each set W^k. If not, we determine a violated constraint for some W^k (i.e., solve the separation problem) and add a new constraint to the partial polyhedral representation of W^k.

Assuming the availability of an exact separation algorithm for W^k, the cutting plane algorithm C described in the last section will terminate with value:

$$z^C = \max\{\sum_k e^k x^k + \sum_k f^k w^k : \sum_{k \in V} x_j^k \leq 1, (x^k, w^k) \in conv(W^k) \text{ for all } k\}$$

However, in contrast to the earlier case, the final solution (x^k, w^k) might not be integer, and a further branch and bound, or branch and cut, phase might be necessary. To implement this approach, we could branch on the variables (x^k, w^k), and add other global cuts in standard fashion. (We describe several such cuts later in this section.)

As we have shown in Section 7, each of these three algorithms provides the same upper bound at the initial node of a branch and bound tree.

Theorem 8.1 *For problem (PSG), the bounds satisfy* $z^{PSG} \leq z^A = z^B = z^C$.

In practice, the effectiveness of these algorithms depends in part on how good an approximation

$$\{(x, w) : \sum_{k \in V} x_j^k \leq 1, (x^k, w^k) \in conv(W^k) \text{ for all } k\}$$

provides to conv(W). Put somewhat differently, if z^A is a tight upper bound on z^{PSG}, the branch and bound tree might permit us to rapidly find an optimal solution and prove its optimality. To tackle the more difficult problems, it is usually necessary to find "strong" valid inequalities (e.g., facets) for conv(W), linking together the different sets W^k. We would also need to integrate these inequalities into the Lagrangian or cutting plane approaches, thereby also improving the branching phase of these algorithms.

We now discuss some approaches that researchers have used for solving the six example problems we introduced at the outset of this section, and comment on the usefulness of the model (PSG).

1. Multi-Item Lot-Sizing. In this context, the graph G is a path from $1, \ldots, n$. For each item k, we need to find a set of intervals (or subpaths) in which item k is produced. The sets W^k assume the form:

$$\begin{aligned} W^k = \{(x^k, s^k, v^k) : s^k_{t-1} + v^k_t &= d^k_t + s^k_t \text{ for all } t \\ v^k_t &\leq M x^k_t \text{ for all } t \\ s^k, v^k \geq 0, x^k_j &\in \{0,1\} \text{ for all } j \text{ and } k\} \end{aligned}$$

with $x^k_t = 1$ if item k is produced in period t; $w^k = (s^k, v^k)$ are the unit stock and production variables, and

$$c^k(x^k) = \min_{s,v} \{\sum_t (p^k_t v^k_t + h^k_t s^k_t + f^k_t x^k_t) : (s^k, v^k, x^k) \in W^k\}.$$

For this problem, both optimization and separation over W^k are well understood and can be implemented rapidly. In particular, in the last section we obtained an extended formulation for the uncapacitated lot-sizing problem (ULS). Researchers have successfully used both cutting plane and Lagrangian relaxation based algorithms, as well as some heuristic algorithms based on column generation, in addressing these problems. Little or nothing is known about facet-defining inequalities linking the items (and so the sets W^k).

2. Clustering. For this problem class, each subgraph in the partitioning is totally unstructured. The set W^k is of the form:

$$W^k = \{(x^k, y^k) : \sum_{i \in V} d_i x^k_i \leq C$$

$$y_e^k \leq x_i^k \text{ for } e = (i,j) \in E$$
$$y_e^k \leq x_j^k \text{ for } e = (i,j) \in E$$
$$x_j^k, y_e^k \in \{0,1\} \text{ for all } j, e, \text{ and } k.$$

The variables x_i^k and y_e^k indicate whether node i or edge e belongs to the kth cluster. The constraints impose a common capacity C on each cluster, and state that any edge e can belong to a cluster only if both its endpoints do. One approach for solving this problem has been to use column generation (Algorithm A), using branch and cut with valid inequalities for W^k to solve the subproblem to optimality. We will describe one family of valid inequalities that can be used to improve the linear programmimg approximation of W^k in the subproblem. It is important to note that for the model we are considering (with an unlimited number of clusters), it is necessary to solve only one subproblem (they are all the same).

Proposition 8.2 *Let $T = (V', E')$ be a subtree of G whose node set V' forms a minimal cover, i.e., $\sum_{i \in V'} d_i > C$ and no subset of V' satisfies this property. Let $deg(i)$ be the degree of node i in T. Then the inequality:*

$$\sum_{e \in E'} y_e^k \leq \sum_{i \in V'} (deg(i) - 1) x_i^k$$

is valid for W^k.

Because V' forms a cover, some node $r \in V'$ must satisfy the condition $x_r^k = 0$. Suppose we root the tree arbitrarily at node r, and sum the inequalities $y_e^k \leq x_i^k$ over all edges $e \in E'$ with i as the endpoint of edge e closest to node r: we obtain $\sum_{e \in E'} y_e^k \leq \sum_{i \in V'} (deg(i) - 1) x_i^k + x_r^k$. (Note that if we collect terms, the coefficient of node r is $deg(r)$.) Since $x_r^k = 0$, the inequality is valid. But since the inequality is independent of the node r satisfying the condition $x_r^k = 0$, all feasible solutions satisfy it.

Limited computational experience has shown that the addition of these inequalities to the linear programming relaxation of the subproblem can be effective in practice. One study, modeling a problem in compiler design, found that on 7 out of 10 problems, the final Restricted Master linear program in the column generation approach gave an integer solution; for the other three problems, the column generation approach found a good feasible solution with a small amount of branching.

Some models use a specialized objective function: they wish to minimize a weighted sum of the edges between the clusters. For these problems, we might use an alternative approach, using the node variables x^k and edge variables w_e $(= 1 - \sum_k y_e^k)$. That is, we no longer keep track of which cluster each edge belongs to, but simply keep track of the edges in the cutsets between the clusters. The model would be similar to the one we have given. One advantage of this approach is that it would permit us to create branch and cut algorithms by drawing upon the extensive work conducted on facets of cut polytopes.

The next three problems are all closely related. The C-Capacitated tree problem is the special case of the general capacitated tree problem with $d_i = 1$ for all $i \in V \setminus \{0\}$. The vehicle routing problem is the restriction of the general capacitated tree problem in which each subtree must essentially be a path (whose endpoints are joined to the depot node 0 to form a tour).

The most successful approaches to date for these problems have all avoided an explicit representation using distinct vehicle variables x^k. Researchers have used column generation and dynamic programming to treat vehicle routing problems with tightly constrained schedules (see Chapter xxx). Other researchers have worked entirely in the space of the edge variables y_e for $e \in E$. This approach implicitly treats the weights and capacities using "generalized subtour inequalities" of the form:

$$\sum_{e \in E(S)} y_e \leq |S| - f(S).$$

In this expression, $f(S)$ is the minimum number of trees, or vehicles, needed to satisfy the demand of all nodes in S. Thus $f(S) = 1$ for the tree problem of Section 3, and $f(S) = \lceil \sum_{i \in S} d_i / C \rceil$ for the capacitated tree problems and vehicle routing problems since in any feasible solution must allocate at least $f(S)$ vehicles (subtrees for the capacitated tree problem) to the node in S, implying that the edges in $E(S)$ must contain at least $f(S)$ components. Researchers have obtained even sharper inequalities (that is, with smaller values for $f(S)$) by solving the NP-hard bin-packing problem of finding the minimum number of bins of capacity C needed to contain the set of weights $\{d_i\}_{i \in S}$.

3. The C-capacitated tree problem. The starting formulation is:

$$\begin{aligned}
&\min \sum_{e \in E} c_e y_e \\
&\text{subject to} \\
&\sum_{e \in E(S)} y_e \leq |S| - \lceil |S|/C \rceil \text{ for all } S \subset V \setminus \{0\} \\
&\sum_{e \in E(S)} y_e \leq |S| - 1 \text{ for all } S \subset V \text{ with } 0 \in S \\
&\sum_{e \in E} y_e = n - 1 \\
&y_e \in \{0,1\} \text{ for all } e \in E.
\end{aligned}$$

We refer to the convex hull of the (integer solutions) of this problem as the capacitated tree polyhedron.

Researchers have tackled this problem using a set of facet-defining inequalities as cutting planes. The inequalities are numerous and fairly complex, so we simply illustrate a few of them with examples. Each of the inequalities is determined by relationships between edges in particular subgraphs (so called supporting subgraphs) of the underlying network. This is an approach we have used before; for example, in studying the minimum spanning tree problem, we considered subtour inequalities. In this case, the subgraphs all edges $E(S)$ with both endpoints in any node set S and the inequality stated that no feasible solution could use more than $|S| - 1$ of these edges.

Figure 22 shows an example of the C-capacitated tree problem with 7 nodes and with $C = 3$. The supporting *multistar* graph divides the nodes into two classes: a set of nucleus nodes N all connected to each other and a set of satellite nodes S, each connected to every node in the nucleus. If $0 \notin N$ and $0 \notin S$, the only feasible solutions are those shown in Figure 22(b), 22(c), and 22(d) as well as subsets of these solutions. Note that if we let $E(N,S)$ denote the set of edges with one endpoint in N and the other in S, then every feasible solution satisfies the *multistar inequality*

$$3 \sum_{e \in E(N)} y_e + \sum_{e \in E(N,S)} y_e \leq 6.$$

The general version of this inequality is

$$K \sum_{e \in E(N)} y_e + \sum_{e \in E(N,S)} y_e \leq (C-1)|N|.$$

The constant K in this inequality is required to be less than C and its value depends upon specific problem data (for our example, $K = C$). The fractional solution shown in Figure 22(e) satisfies all the constraints of the starting formulation, but not the multistar inequality.

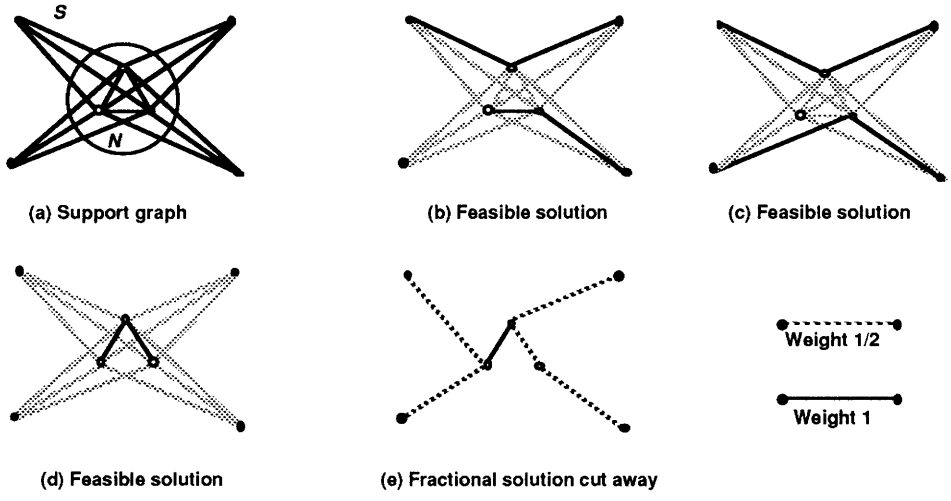


Figure 22: Multistar with C = 3

Figure 23 shows a second example: in this case $C = 5$ and the supporting *clique cluster* graph has three cliques (complete graphs) C_1, C_2, and C_3, all sharing exactly one common node and none containing node 0. The figure shows feasible solutions that satisfy and a fractional solution that does not satisfy the valid *clique cluster inequality*

$$\sum_{e \in C_1} y_e + \sum_{e \in C_2} y_e + \sum_{e \in C_3} y_e \leq 6.$$

The general inequality for t cliques is

$$\sum_{1 \leq j \leq t} \sum_{e \in C_j} y_e \leq constant.$$

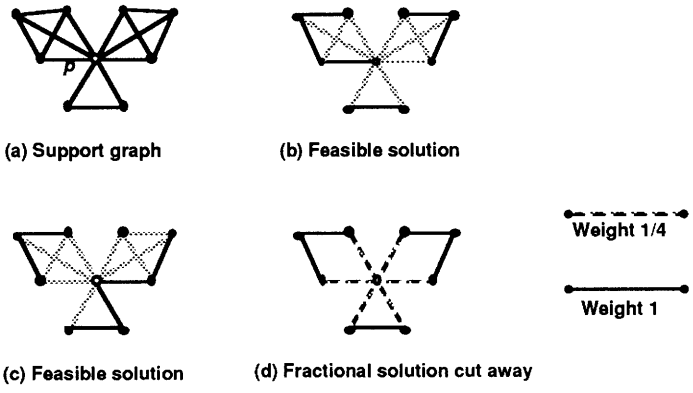


Figure 23: Clique Cluster with C = 5

The constant is determined by the structure of the clique cluster and the value of C.

Figure 24 shows a third example: in this case $C = 3$ and the supporting *ladybug* graph has a set B of body nodes all connected to each other, a set H of head nodes and a set A of antenna nodes. The head and antenna nodes form a multistar and each body node is connected to each head node. We assume that the ladybug graph does not contain node 0. The figure shows several feasible solutions and a fractional solution that satisfies all the other constraints we have considered but not the valid *ladybug* inequality which is

$$2 \sum_{e \in E(B)} y_e + 2 \sum_{e \in E(B,H)} y_e + 3 \sum_{e \in E(H)} y_e + \sum_{e \in E(H,A)} y_e \leq 6.$$

The ladybug inequality in general is similar: the edges in $E(H)$ have a weight of C, the edges in $E(H, A)$ have a weight of 1. The edges $E(B)$ and $E(B, H)$ have the same weight d. The value of d and the right-hand side of the inequality all depend upon the values of C, $|B|$, and $|H|$.

All three classes of these inequalities define facets of the C-capacitated tree polyhedron, assuming mild restrictions on C and the sizes of the support graphs. Because these conditions are complicated, we will not present them in this discussion.

Computational experience has shown that multistar and certain partial mulitstar extensions of them are useful in a cutting plane approach for solving the C-capacitated tree problem.

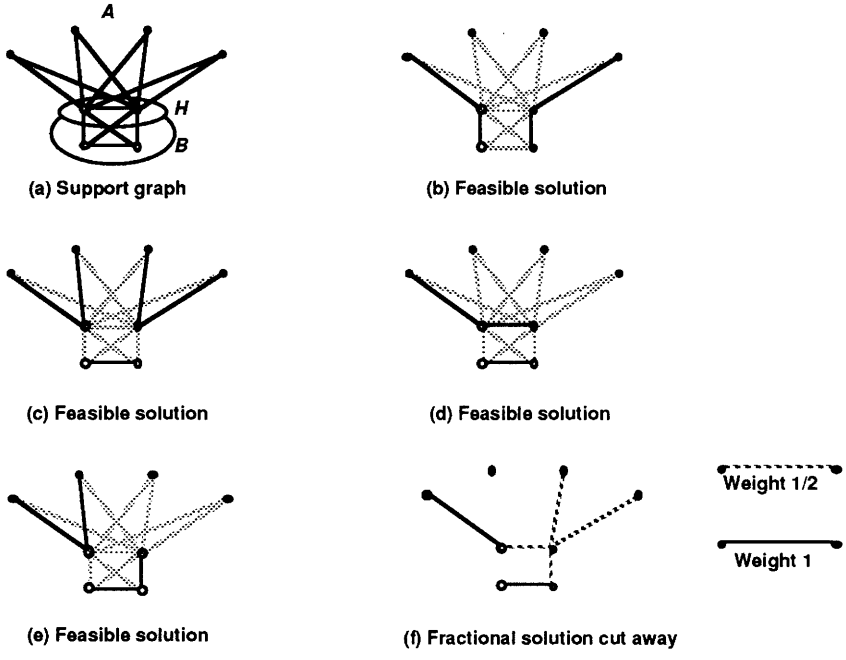


Figure 24: Ladybug with C = 3

4. The Capacitated tree problem. The starting formulation is:

$$
\begin{aligned}
&\min \sum_{e \in E} c_e y_e \\
&\text{subject to} \\
&\sum_{e \in E(S)} y_e \leq |S| - f(S) \text{ for all } S \subset V \setminus \{0\} \\
&\sum_{e \in E(S)} y_e \leq |S| - 1 \text{ for all } S \subset V \text{ with } 0 \in S \\
&\sum_{e \in \delta(i)} y_e \geq 1 \\
&y_e \in \{0,1\} \text{ for all } e \in E.
\end{aligned}
$$

The function $f(S)$ can be any of the functions we introduced before, for example, $f(S) = \lceil \sum_{i \in S} d_i / C \rceil$. In this model $\delta(i)$ denotes the edges with one endpoint at node i. We can view one approach for solving this problem as a combination of a cutting plane algorithm and Lagrangian relaxation. The first difficulty is the exponential number of constraints. Unlike the tree problem, no polynomial separation algorithm is known for the generalized subtour inequalities, so researchers have used heuristics to find violated inequalities. Suppose we have added constraints corresponding to the set $S^1, \ldots, S^t$ to the formulation. One solution approach would be to remove these constraints by incorporating them into the objective function with Lagrange multipliers giving, as a Lagrangian subproblem, a branching problem:

$$
\begin{aligned}
&\min \sum_{e} c_e y_e - \sum_{t=1}^{r} [\sum_{e \in E(S^r)} y_e - (|S^r| - f(S^r))] \lambda_t \\
&\text{subject to} \\
&\sum_{e \in \delta(i)} y_e \geq 1 \text{ for } i \in V \setminus \{0\} \\
&\sum_{e \in E} y_e = n - 1 \\
&y_e \in \{0,1\} \text{ for } e \in E
\end{aligned}
$$

which we could then solve using the algorithm we presented in Section 5.

5. Capacitated Vehicle Routing. The starting formulation is very similar to that of the previous model, and that of the travelling salesman problem. Let 0 be the depot and $V_0 = V \setminus \{0\}$. With a fixed number K of vehicles, we have:

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e y_e && (8.2)\\
\text{subject to} & \\
\sum_{e \in \delta(0)} x_e &= 2K && (8.3)\\
\sum_{e \in \delta(i)} x_e &= 2 \text{ for } i \in V_0 && (8.4)\\
\sum_{e \in \delta(S)} x_e &\geq 2\alpha(S) \text{ for } S \subseteq V_0, S \neq \phi && (8.5)\\
x_e \in \{0,1\} \text{ for } e \in E(V_0), x_e &\in \{0,1,2\} \text{ for } e \in \delta(0). && (8.6)
\end{aligned}$$

In this model $\alpha(S) \in \{\lceil \sum_{i \in S} d_i/C \rceil, r(S), R(S)\}$ with $\lceil \sum_{i \in S} d_i/C \rceil \leq r(S) \leq R(S)$. The first term in this expression for $\alpha(S)$ is the basic capacity bound; $r(S)$ is the bin packing bound, i.e., the minimum number of bins needed to pack the set of demands $\{d_i\}_{i \in S}$; and $R(S)$ is the same bound taking into account the requirement that we must pack all the demands into K bins (this bound accounts for the demand $\{d_i\}_{i \notin S}$). Note that for the travelling salesman problem, $C = \infty$, and then (8.5) becomes $\sum_{e \in \delta(S)} x_e \geq 2$, the basic cut (or, equivalently, subtour elimination) constraint.

A generalization of the so-called *comb inequalities* from the TSP problem applies to this problem. The support graph for these inequalities contains a *handle* H and a set $\{T_j\}_{j=1}^{s}$ of *teeth* satisfying the conditions:

$$\begin{aligned}
T_i \cap T_j &= \phi\\
T_j \cap H &\neq \phi\\
T_j, H &\subseteq V_0\\
\sum_{i \in T_j} d_i &\leq C.
\end{aligned}$$

See Figure 25.

Let $2P(H)$ be the minimum number of times any set of subpaths must intersect H in order for any solution to satisfy all the demands in H given

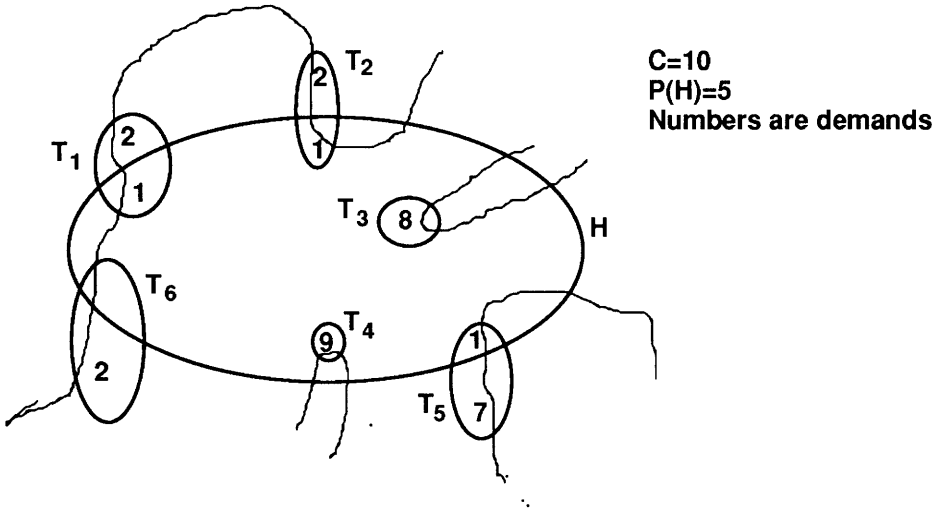


Figure 25: A Feasible Solution and a Comb

that the demands in each tooth T_j are consecutive on one of the subpaths, see Figure 25.

Proposition 8.3 *The generalized comb inequality*

$$\sum_{e \in \delta(H)} x_e \geq 2P(H) - \sum_{j=1}^{s} \Big(\sum_{e \in \delta(T_j)} x_e - 2 \Big)$$

is valid for CVRP.

Note that if a vehicle visits the clients in a tooth one after the other, $\sum_{e \in \delta(T_j)} x_e = 2$ for each j, and the inequality is valid by definition of $P(H)$. If one or more vehicles visits the tooth T_j more than once, then $\sum_{e \in \delta(T_j)} x_e = 2 + k_j$ for some integer $k_j \geq 1$. It is then necessary to verify that this solution never reduces the intersections with the boundary of H by more than $\sum_{j=1}^{s} 2k_j$.

Observe that for the TSP, when $T_j \setminus H \neq \phi$ for $j = 1, \ldots, s$ and s is odd, then $2P(H) = s + 1$, and so this inequality becomes the usual comb inequality.

6. Packing Steiner Trees.

Packing Steiner trees is important in VLSI design, and recently researchers have solved some previously unsolved problems to optimality using a polyhedral approach. Suppose that the edge capacities c_e all equal 1, so we are seeking a minimum cost packing of edge disjoint Steiner trees. We present two results concerning valid inequalities that have proven to be useful in recent compuational sucesses.

Proposition 8.4 *Every nontrivial facet-defining inequality of the Steiner tree polyhedron yields a facet-defining inequality of the Steiner tree packing polyhedron.*

This result implies in particular that the Steiner partition inequalities (Proposition 6.3) provide facets for this problem.

The next class of inequalities involve more than a single Steiner tree. Consider two terminal disjoint sets T_1 and T_2. We refer to a cycle $F \subseteq E$ as an *alternating cycle* with respect to T_1 and T_2 if $F \subseteq E(T_1, T_2)$. We refer to an edge (u, v) as a *diagonal edge* if $u, v \in V(F)$, but $(u, v) \notin F$. We let $y_e^k = 1$ if edge e is in Steiner tree k.

Proposition 8.5 *Let F be an alternating cycle with respect to T_1 and T_2, and $F_1 \subseteq E(T_2), F_2 \subseteq E(T_1)$ be two sets of diagonal edges. Then*

$$\sum_{e \in E \setminus (F \cup F_1)} y_e^1 + \sum_{e \in E \setminus (F \cup F_2)} y_e^2 \geq |V(F)|/2 - 1$$

is a valid inequality for the Steiner tree packing polyhedron.

Figure 26 shows an alternating cycle F of length 6, as well as two tight feasible solutions in which S_1 and S_2 are the Steiner trees spanning T_1 and T_2.

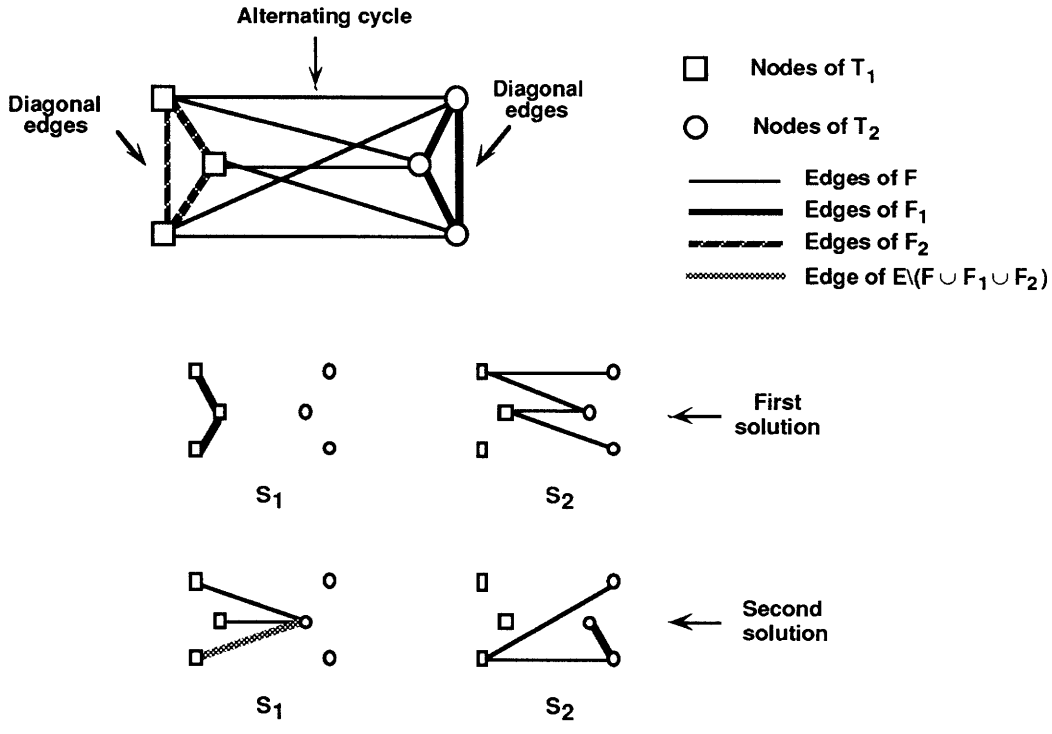


Figure 26: Alternating Cycle for Packing Steiner Trees

9 Trees-on-trees

Each of the models we have examined so far uses a single type of facility (edge) to construct a tree, or a packing of trees, in a given graph. Some applications need to distinguish between different types of edges. For example, electrical power systems often must connect major users with high voltage (or highly reliable) transmission lines, but can use cheaper, low voltage (or less reliable) lines to connect other users. Roadway systems often need to use highways to connect major cities, but can use secondary roads to connect smaller cities.

These applications give rise to a set of hierarchical models that are generalizations of the models we have considered earlier in this chapter. To illustrate the analysis of this class of models, we will consider a particular type of hierarchical problem.

9.1 Tree-on-tree Model

Suppose we are given an undirected graph $G = (V, E)$ with two types of nodes, primary P and secondary S: $P \cup S = V$ and $P \cap S = \phi$. We wish to find a minimum cost spanning tree in G. The problem differs from the usual spanning tree problem, however, because we need to designate any edge in the spanning tree either as a primary (high capacity, more reliable) edge or as a secondary (low capacity, less reliable) edge. Designating edge $\{i, j\}$ as a primary edge costs $a_{ij} \geq 0$ and as a secondary edge costs $b_{ij} \geq 0$; we assume $b_{ij} \leq a_{ij}$. The spanning tree we choose must satisfy the property that the unique path joining every pair of primary nodes contains only primary edges.

As shown in Figure 12, we can interpret the solution to this "tree-on-tree" problem as a Steiner tree with primary edges superimposed on top of a spanning tree. The Steiner tree must contain all the primary nodes (as well, perhaps, as some secondary nodes).

Note that if the costs of the secondary edges are zero, then the problem essentially reduces to a Steiner tree problem with edge costs a_{ij} (the optimal solution to the tree-on-tree problem will be a Steiner tree connected to the other nodes of the network with zero-cost secondary edges). Therefore, the tree-on-tree problem is at least as hard as the Steiner tree problem and so we can expect that solving it will be difficult (at least from a complexity perspective) and that its polyhedral structure will be complicated. If the

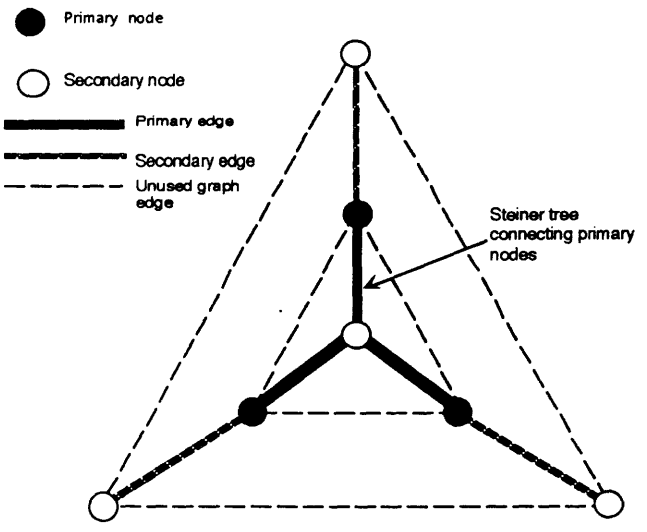


Figure 27: Steiner tree on a spanning tree

costs a and b are the same, the problem reduces to a minimum spanning tree problem. Therefore, the tree-on-tree problem encompasses, as special cases, two of the problems we have consider previously in this chapter.

In this section, we develop and analyze a heuristic procedure for solving this tree-on-tree problem; we also analyze a linear programming representation of the problem. In the context of this development, we show how to use some of the results developed earlier in this chapter to analyze more complex models.

To model the tree-on-tree problem, we let x_{ij} and y_{ij} be 0-1 variables indicating whether or not we designate edge $\{i, j\}$ as a primary or secondary edge in our chosen spanning tree; both these variables will be zero if the spanning tree does not include edge $\{i, j\}$. Let S denote the set of incidence vectors of spanning trees on the given graph and let ST denote the set of incidence vectors of feasible Steiner trees on the graph (with primary nodes as terminal nodes and secondary nodes as Steiner nodes). Let $x = (x_{ij})$ and $y = (y_{ij})$ denote the vectors of decision variables. In addition, let $c_{ij} = a_{ij} - b_{ij}$ denote the incremental cost of upgrading a secondary edge to a primary edge. With this notation, we can formulate the tree-on-tree problem as the following integer program:

$$
\begin{aligned}
z^{ip} &= \min cx + by \\
\text{subject to} & \\
x &\leq y \\
x &\in ST \\
y &\in S.
\end{aligned}
$$

The forcing constraint $x_{ij} \leq y_{ij}$ states that if the chosen Steiner tree contains edge $\{ij\}$, that is, $x_{ij} = 1$, then the chosen spanning tree must also contain this edge. Thus, if edge $\{ij\}$ belongs to the Steiner tree, then $x_{ij} = y_{ij} = 1$ and so, as required, edge $\{i, j\}$ contributes $a_{ij} = c_{ij} + b_{ij}$ to the solution's cost.

9.2 Heuristic Analysis

In an attempt to solve the tree-on-tree problem, we consider a heuristic that chooses the better solution obtained from two other heuristics:

Spanning tree heuristic: Solve a minimum spanning tree problem with respect to the costs a_{ij} and designate each edge in the minimum spanning tree as a primary edge.

Steiner tree completion heuristic: Solve a Steiner tree problem (optimally or approximately) with the primary nodes P as the terminal (required) nodes and designate every edge on this tree T as a primary edge; using the costs b_{ij}, find a least cost spanning tree T' that contains T; designate each edge in $T \setminus T'$ as a secondary edge.

Composite heuristic: Choose the smaller cost solution found by the spanning tree and Steiner tree completion heuristics.

A few comments are in order concerning these heuristics. After using the spanning tree heuristic, we might try to improve upon the solution by downgrading some primary edges into secondary edges: we can downgrade every edge not on a path joining any two primary nodes. Even though we would use this procedure in practice, examples show that this improvement does not affect the worst-case analysis we will be considering, and so we use a simpler procedure without this downgrading phase. Second, to complete

the Steiner tree into a minimum spanning tree in the Steiner tree completion heuristic, we can use a version of the spanning tree greedy algorithm: starting with the edges in the Steiner tree as members of the spanning tree, add cheapest cost secondary edges one at a time that do not create cycles with the edges already chosen.

We might consider two versions of the problem, an *unrelated cost model* in which the costs a and b are arbitrary, and a *proportional cost model* in which the costs a_{ij} the costs of primary to secondary edge are proportional to each other; that is, for some constant r, $a_{ij} = rb_{ij}$ for all edges $\{i, j\}$. First, let us consider the proportional cost model.

Let z^S, z^{ST}, z^{CH} be the objective values of the solutions produced by the spanning tree, the Steiner tree completion, and the composite heuristics. To streamline our notation, assume by scaling that $z^S = r$. Since $z^S = r$ is the cost of a minimum spanning tree containing primary edges, the cost of a minimum spanning tree with secondary edges is 1. We also let s, an unknown, denote the cost of an optimal Steiner tree connecting the primary nodes with *secondary edges.*

In terms of this notation, if we solve the Steiner tree to optimality, we have,

$$\begin{aligned} z^S &= r \\ z^{ST} &\leq rs + 1. \end{aligned}$$

The specified upper bound on z^{ST} is valid because the incremental cost of completing an optimal Steiner tree T at least cost using secondary edges can be no more than the cost of a minimum spanning tree with secondary edges. (To establish this result, note that if we set the cost of every edge $\{i, j\}$ in T from $b_{ij} \geq 0$ to zero, the greedy algorithm on the entire graph could generate the tree produced by the completion procedure as a minimum spanning tree. The assertion is true since reducing some edge costs from b_{ij} to 0 cannot increase the length of a minumim spanning tree.)

If we eliminate the forcing constraints $x \leq y$ from the integer programming formulation of the tree-on-tree problem, the problem decomposes into a minimum spanning tree problem with secondary costs b and a Steiner tree problem with incremental costs c. Since we are considering the proportional cost model, $c = a - b = rb - b = b(r - 1)$. Therefore, the cost of an optimal

Steiner tree with respect the incremental costs c is $(r-1)s$. Since removing constraints cannot increase the optimal costs, we obtain a lower bound on z^{ip}:

$$z^{ip} \geq 1 + (r-1)s.$$

We next use the previous upper and lowerbounds to analyze the composite heuristic.

Theorem 9.1 *For the tree-on-tree problem with proportional costs, if we solve the Steiner tree problem in the Steiner tree completion heuristic to optimality, then*

$$\frac{z^{CH}}{z^{ip}} \leq 4/3.$$

Proof. Combining the upper bounds on z^S and z^{ST} and the lower bound on z^{ip} shows that

$$\frac{z^{CH}}{z^{ip}} \leq \frac{min\{r, rs+1\}}{1+(r-1)s}.$$

For a given value of r, the first term on the right-hand side of this expression decreases with s and the second term increases with s. Therefore, we maximize the right-hand side of this expression by setting $r = rs + 1$ or $s = (r-1)/r$. With this choice of s, the bound on z^{CH}/z^{ip} becomes

$$\frac{z^{CH}}{z^{ip}} \leq \frac{r}{1+(r-1)s} = \frac{r^2}{r+(r-1)^2}.$$

To maximize the right-hand side over r, we set the derivative of the right-hand side to zero, giving $r = 2$ and so $z^{CH}/z^{ip} \leq 4/3$.

Note that when $|P| = 2$, the Steiner tree problem becomes a shortest path problem and so we can solve it to optimality. We can also solve the Steiner tree problem to optimality for specialized classes of network, in particular so-called series-parallel networks. Therefore, the 4/3 bound applies to these situations.

In general, we won't be able to solve the Steiner tree problem to optimality, but will instead use an approximation procedure to solve the problem.

Let us suppose that for the problem class that we wish to investigate, we can obtain a heuristic solution to the Steiner tree problem with a guaranteed performance bound of ρ; that is, the cost of the solution we generate is never more than $\rho \geq 1$ times the cost of an optimal solution. For example, as we have seen in Section 6, for problems satisfying the triangle inequality, we can use an heuristic with a performance guarantee of $\rho = 2$. For Euclidean graphs $\rho = 2$ and, as we have noted already, for series-parallel graphs, $\rho = 1$.

In this case, we obtain the following upper bound on the cost z^{ST} of the Steiner tree completion heuristic:

$$z^{ST} \leq \rho rs + 1.$$

An analysis similar to the one we used to analyze the situation when we could solve the Steiner tree problem optimally permits us to establish the following result.

Theorem 9.2 *For the tree-on-tree problem with proportional costs, if we solve the Steiner tree problem in the Steiner tree completion heuristic using a heuristic with a performance guarantee of ρ, then*

$$\frac{z^{CH}}{z^{ip}} \leq \frac{4}{4-\rho} \text{ if } \rho \leq 2$$

and

$$\frac{z^{CH}}{z^{ip}} \leq \rho \text{ if } \rho > 2$$

.

For the unrelated cost model, a similar analysis permits us to obtain the following result.

Theorem 9.3 *For the tree-on-tree problem with unrelated costs, if we solve the Steiner tree problem in the Steiner tree completion heuristic using a heuristic with a performance guarantee of ρ, then*

$$\frac{z^{CH}}{z^{ip}} \leq \rho + 1$$

.

Although we will not establish this fact in this discussion, examples show that the bound in Theorems 9.1, 9.2, and 9.3 are tight—that is, some examples achieve the worst-case bounds.

9.3 Linear Programming Bounds

Let P_{ST} and P_S be any polyhedral approximations to the spanning tree and Steiner tree polytopes in the sense that $ST \subseteq P_{ST}$ and $S \subseteq P_S$. For example, these polyhedra can be any of the possibilities we have considered in Sections 3 and 6. With respect to these polyhedra, we can consider the following linear programming relaxation of the tree-on-tree problem:

$$\begin{aligned} z^{lp} &= \min\ cx + by \\ \text{subject to} & \\ x &\leq y \\ x &\in P_{ST} \\ y &\in P_S. \end{aligned}$$

Note that the polyhedra P_{ST} and P_S contain the constraints $0 \leq x_{ij} \leq 1$, and $0 \leq y_{ij} \leq 1$ for all edges $\{ij\}$.

To see how well this linear program represents the tree-on-tree problem, we would like to bound z^{ip}/z^{lp}. The bound we will obtain depends upon how well the polyhedra P_{ST} and P_S represent ST and S. As we have seen in Section 3, we can choose several equivalent polyhedra so that P_S is the convex hull of S. Suppose we choose one of these polyhedra. Moreover, suppose that our choice P_{ST} permits us to obtain a performance guarantee of $\theta \geq 1$ whenever we optimize any linear function over P_{ST}, that is for all choices γ of objective functions coefficients,

$$min\ \{\gamma x : x \in ST\}\ \leq \theta\ min\ \{\gamma x : x \in P_{ST}\}.$$

Note that if we eliminate the forcing constraints from the linear programming relaxation, then the problem separates into two independent linear programming subproblems, one defined over P_S with cost coefficients b and one defined over P_{ST} with cost coefficients c. Since P_S equals the convex hull of spanning tree solutions, the first linear program has an optimal objective value equal to 1, the value of a minimum spanning tree using secondary edges. Our performance guarantee implies that the optimal objective value equal of the second linear program is no less than $(r-1)s/\theta$. (Recall that

$(r-1)s$ is the cost of an optimal Steiner tree connecting the primary nodes using incremental costs c.)

As before, eliminating the forcing constraints $x \leq y$ cannot increase the optimal objective value, so we obtain the following lower bound on the objective value of the linear programming relaxation:

$$z^{lp} \geq 1 + [(r-1)s/\theta].$$

As we have noted before, if we solve the Steiner tree problem to optimality, $z^{ip} \leq z^{ST} \leq rs + 1$; moreover, $z^{ip} \leq r$, the cost of a minimum spanning tree with primary edges. Combining these results gives us the bound

$$\frac{z^{ip}}{z^{lp}} \leq \frac{\theta \ min\{r, rs+1\}}{\theta + (r-1)s}.$$

Using an analysis similar to that used in the development of Theorems 9.1 and 9.2 permits us to establish the following result.

Theorem 9.4 *Suppose that for any vector γ, the optimal value of the linear program min $\{\gamma y : y \in P_{ST}\}$ defined over the polyhedron P_{ST} is at least θ times the cost of an optimal Steiner tree with edge costs γ. Then for the tree-on-tree problem with proportional costs,*

$$\frac{z^{ip}}{z^{lp}} \leq \frac{4}{4-\theta} \text{ if } \theta \leq 2$$

and

$$\frac{z^{ip}}{z^{lp}} \leq \theta \text{ if } \theta > 2.$$

Similarly, we can obtain the following result for the unrelated cost model.

Theorem 9.5 *Suppose that for any vector γ, the optimal value of the linear program min $\{\gamma y : y \in P_{ST}\}$ defined over the polyhedron P_{ST} is at least θ times the cost of an optimal Steiner tree with edge costs γ. Then for the tree-on-tree problem with unrelated costs,*

$$\frac{z^{ip}}{z^{lp}} \leq \theta + 1.$$

In Section 6, we analyzed one formulation of the Steiner tree problem with $\theta = 2$. Theorems 9.4 and 9.5 show that by using this same formulation for the tree-on-tree problem, we obtain the same worst-case bound $4/(4-2) = 2$ for the proportional cost tree-on-tree problem and a bound of 3 for the unrelated cost model.

Using flow models to represent P_{ST} and P_S and a dual ascent procedure to approximately solve the resulting linear programming relaxation of the tree-on-tree problem combined with an associated linear programming-based heuristic, researchers have been able to solve large-scale problems (with up to 500 nodes and 5000 arcs) to near optimality (guaranteed within 2% of optimality). This computational experience is comparable (in problem size, performance guarantee, and algorithm execution time) to the computational experience for solving the Steiner tree subproblem itself.

10 Summary

In this chapter, we have considered a variety of tree optimization problems. Stimulated by applications in telecommunications, production planning, routing, and VLSI design introduced in Section 2, we set out to examine a variety of issues in modeling and algorithm design. Rather than attempting to summarize all the results we have presented, in these concluding remarks, we will focus on a few lessons to be learned from our discussion—about trees, about modeling, or about algorithm design.

The algorithms we have considered either directly exploit a problem's underlying combinatorial structure and/or build upon insights derived from the problem's representation(s) as mathematical programs. For example, typically, when a problem is defined on a tree, a dynamic program will provide an efficient solution procedure. Algorithms in this category include dynamic programs for the optimal rooted subtree problem (Section 4.1) and for packing subtrees in a tree (Section 7). Greedy algorithms traditionally exploit a problem's underlying combinatorial structure: for example, the fact that whenever we add an edge to a spanning tree, we create a new tree by deleting any edge in the cycle that this edge produces. The basic greedy algorithm for the minimum spanning tree problem (Section 3) exploits this property and the modified greedy algorithm (with node shrinking) for the optimal branching problem (Section 5.2) exploits the combinatorial property that if the node greedy solution contains a cycle C, then the problem has an optimal solution containing $|C| - 1$ arcs from C.

Often in combinatorial optimization, whenever we can solve a problem efficiently (using a number of computations that are polynomial in the problems size—e.g., the number of nodes and edges of the associated graph), we are able to completely describe the underlying integer polyhedron, for example, by showing that the algorithm that generates an integer solution solves a linear programming formulation of the problem. This is the case for each of the problems mentioned in the last paragraph.

For other problems, such as the capacitated version of the optimal rooted subtree of a tree problem (Section 4.2), dynamic programming algorithms might require more than a polynomial number of computations. The number of computations required for the dynamic programming algorithm for the capacitated rooted subtree of a tree problem we have given grows as a polynomial in the capacity C, which is exponential in $log(C)$, the number of bits

necessary to store the capacity (and therefore, in the sense of computational complexity theory, the dynamic program is not a polynomial algorithm). As a general rule, in these situations, the underlying integer polyhedra will be quite complex; our discussion of several valid inequalities for this capacitated rooted subtree problem illustrates this point.

Our analysis of the core minimum spanning tree problem in Section 3 provides useful lessons concerning the use of a mathematical programming model to analyze an algorithm even when the mathematical program itself is not required for creating or even stating an algorithm. In particular, the use of linear programming lower bounds (assuming a minimization form of the problem) and linear programming dual variables has permitted us to show that the algorithm finds an optimal solution. The same type of linear programming bounding argument applies to many other problem situations, as illustrated, for example, by our discussion of the rooted subtree of a tree problem.

For more complex models such as those we considered in Sections 6, 8, and 9, the underlying integer polyhedra are generally quite complicated. The problems are also generally difficult to solve, at least in the theoretical sense of computational complexity worst-case analysis. One major stream of research for addressing such situations has been to develop "good" linear programming representations of these problems, typically by adding new valid inequalities to "natural" starting formulations. As we have noted, often by developing good linear programming representations, empirically we are able to obtain optimal or near optimal solutions fairly efficiently.

For two classes of models, Steiner tree models (with costs satisfying the triangle inequality) and tree-on-tree models, we have been able to establish worst-case bounds on the ratio between the objective values of the underlying integer program and its linear programming relaxation and between the optimal objective value of the problem and the optimal objective value of certain heuristic solution methods. In each case, we were able to do so by using a common technique in combinatorial optimization: relating the optimal objective value of the problem to some convenient and more easily analyzed (linear programming or Lagrangian) relaxation of it.

Our development has frequently introduced and compared alternate modeling and algorithmic approaches. In the context of packing subtrees on a tree, we showed the equivalence between three popular general modeling approaches—column generation, Lagrangian relaxation, and cutting planes.

As we have seen, these three approaches all are capable of solving this class of problems: they all find the optimal objective value. When applied to the more general problem of packing subtrees on general graphs, the tree solution methods also provide identical initial bounds on the optimal objective function value of problem. In this broader context, we typically need to embed the starting solution into an enumeration procedure or in a branch and cut method that adds valid inequalities to improve the initial bounds. Lagrangian relaxation and column generation are attractive solution methods whenever we can identify an easily solvable subproblem (for example, if the subproblems are any of the polynomially solvable problems we have mentioned above). Column generation, like cutting plane methods, has the advantage of working directly in the space of decision variables, whereas Lagrangian relaxation works in a dual space of Lagrange multipliers. Column generation has the further advantage of possibly providing a feasible solution to the problem before optimality is attained. Lagrangian relaxation, on the other hand, has the advantage of not having to solve a complex master problem, but rather typically uses simple multiplier adjustment methods (subgradient optimization, simple heuristic methods) to find the optimal Lagrange multipliers. Cutting planes require the solution of comparatively expensive linear programs at each stage (reoptimization from stage to stage is, however, much easier than solving the linear program form scratch); cutting planes have the advantage of being rather universal, however, (not requiring easily solvable subproblems) as a long as we can solve the separation problem of finding a violated inequality at each iteration, either optimally or heuristically.

As this discussion shows, having different solution procedures at our disposal offers us the flexibility of exploiting various characteristics of the problem we are solving.

Alternate models can be valuable for several reasons. First, as we have just seen, some models (those that better identify special underlying substructure) are better suited for use with different algorithms. Second, alternate models can offer different theoretical or applied insight; for example, the fact that the number of variables and constraints in the multicommodity flow formulation of the minimum spanning tree is polynomial in the number of nodes and edges of the associated graph immediately implies, from the theory of linear programming, that the problem is solvable in polynomial time without any insight into the problem's combinatorial structure. In addition,

alternate, but equivalent models, often can work in concert with each other. For example, as we saw in our discussion of the minimum spanning tree problem, the multicommodity flow formulation permits us to efficiently solve the separation problem that arises when we apply a cutting plane algorithm to the subtour formulation.

Our development has also highlighted one other fact; we have seen that tree optimization problems provide a concrete problem setting for introducing a number of important methodologies and proof (and solution) techniques from combinatorial optimization that are applicable more widely. We have seen, for example, how to use dynamic programming or combinatorial methods (the greedy algorithm) to define dual variables for underlying linear programming representations of integer programs. We have introduced the optimal inequality argument from the field of polyhedral combinatorics. We have used linear programming and Lagrangian relaxation lower bounds to show that certain polyhedra are integer. In our study of the Steiner problem, we have seen how to develop worst-case bounds by relaxing some of the constraints of a linear programming model and we have seen how to use the "parsimonious propert" to establish worst-case bounds. For the tree-on-tree problem, we have seen how to combine two heuristics to develop a composite heuristic with attractive worst-case error bounds.

Tree optimization problems and their variants are conceptually simple. As we have seen, simple yet elegant solution methods are able to solve some versions of this general problem class and very good mathematical (linear programming) representations are available for many of these problems. In this sense, many tree optimization problems are well solved. And yet, new insights about tree optimization problems continue to surface; moreover, this deceptively simple problem setting also poses significant algorithmic and modeling challenges that have the potential, as in the past, to not only draw upon a wealth of knowledge from the general field of combinatorial optimization, but to also stimulate new results and new methods of analysis.

11 Notes and References

Section 1. No previous source has dealt with the range of topics we have considered in this paper. Many books in the fields of network flows, graph theory, and combinatorial optimization, as well as several survey articles that we cite below, treat a number of particular topics though.

Section 2. For general background on the application domains we have considered in this discussion, see the following sources: clustering (Hartigan (1975)), computer and communications networks (Bertsekas and Gallager (1992), Schwartz (1977), and Tanenbaum (1985)), facility location (Francis, McGinnis and White (1992) and Mirchandani and Francis (1990)), production planning (Graves et al. (1993)), routing (Bodin et al. 1983 and Lawler et al. (1985)), VLSI design (Leighton (1983), Hu and Kuh (1985), and Lengauer (1990)). Wagner and Whitin (1958) proposed the dynamic programming recursion for the production planning problem. For a recent account of network flows, see Ahuja, Magnanti and Orlin (1993).

Section 3. The greedy algorithm and the combinatorial proof for the spanning tree problem are due to Kruskal (1956). This result can also be found earlier in the Russian literature (Boruvka (1926) and Jarník (1930)). Prim (1957) and Sollin (see Berge and Ghouila-Houri (1962)) have developed other efficient algorithms for the spanning tree problem (see Ahuja et al. (1993) for a discussion of these methods). Edmonds (1971) has described the matroid polytope of which the tree polytope is a special case, and used the primal-dual proof to prove integrality of the tree polyhedron. His idea of a certificate of optimality has been of crucial importance in combinatorial optimization that predates the development of NP-completeness (Cook (1971) and Karp (1972)).

Wong (1980) first spurred interest in alternative formulations for the tree polyhedron. In the context of the travelling salesman problem, he showed the equivalence of the subtour model and the multicommodity flow model. In Section 6 in our investigation of formulations for the Steiner tree problem, we consider generalizations of this work. The study of alternative formulations has become an important topic in combinatorial optimization; see Martin

(1987) and Nemhauser and Wolsey (1988).

The mimimum spanning tree is just one of many problems defined on trees; for example, we could choose a "bottleneck" objective function of maximizing the minimum edge weight in the chosen tree. Camerini, Galbiati and Maffioli (1984) have provided a survey of the computational status of many such alternative tree optimization problems

Section 4. In our discussion, we have introduced the core tree problem as a prototype for a nonserial (non shortest path) dynamic program. Groeflin and Liebling (1981) have presented a more general model and used the network flow argument to prove integrality. The dynamic programming argument is again of the primal-dual type due to Edmonds that we cited above. Lovász (1979) first used the so-called optimal inequality argument which other researchers have recently rediscovered and used extensively.

Section 5. The degree constrained spanning tree problem and the optimal branching problem are special cases of the matroid intersection problem and are thus covered in Edmonds (1969). The algorithm for the degree constrained spanning tree problem is due to Volgenant (1989). Edmonds (1967) first treated optimal branchings. The branching algorithm, though not our analysis of it, is based on a presentation in Lawler (1976). The proof of integrality we have given uses ideas of an optimal inequality proof due to Goemans (1992).

Section 6. The Steiner problem has received considerable attention in recent years; Maculan (1987) and Winter (1987) have presented surveys on exact and heuristic algorithms, respectively. See also, Hwang and Richards (1992) and Hwang et al. (1992). Very recently several researchers—Goemans (1994), Myung and Goemans (1993), Lucena and Beasley (1992) and Margot et al. (1991)—have modeled and analyzed the node weighted Steiner tree problem. The first two of these papers shows the equivalence of a large number of formulations.

The polyhedral structure of the Steiner problem is treated in Chopra and Rao (1988a, 1988b) who developed various families of facet-defining inequal-

ities, including both Steiner Partition and odd hole inequalities. Goemans (1994) introduced combinatorial design facets.

As was suggested in Section 4, most optimization problems on trees turn out to be easy, so it is natural to ask whether there is a larger class of graphs on which problems that are NP-hard on general graphs remain polynomially solvable. Series-parallel graphs (also known as two-trees) and more generally k-trees often have this property: see, for example, Takamizawa et al. (1982), Arnborg et al. (1991). The critical point in analyzing these problems is the fact that by eliminating a k-clique in a k-tree it is possible to decompose the graph, and thereby derive a recursive algorithm. Given that there are efficient algorithms on such graphs, we might also expect polyhedral results. For instance, Goemans (1994) and Margot et al. (1994) show that the formulation P_{sub} is integral for the node-weighted Steiner problem on series parallel graphs. Prodon et al. (1985), Goemans (to appear), and Schaffers (1991) contain related polyhedral results on such graphs.

Lucena and Beasley (1992) and Chopra and Gorres (1990) have conducted computational studies based on the formulation P_{sub}, Chopra, Gorres and Rao (1992) have used formulation P_{dcut} and Balakrishnan et al. (1992) have used a directed flow formulation. Wong (1984) earlier developed a dual ascent algorithm for the directed Steiner problem using P_{dcut}, and Beasley(1989) developed a Lagrangian approach with spanning tree subproblems. Beasley has solved random generated sparse problems with up to 2500 nodes and 62500 edges, Chopra et al. have handled graphs with up to 300 nodes and average degrees of 2,5 and 10, as well as complete Euclidean graphs with between 100 and 500 nodes, and Balakrishnan et al. have solved problems with up to 500 nodes and 5000 edges.

Much work has been done on heuristics for the Steiner problem. One motivation comes from VLSI design and the routing of nets, see Korte et al. (1990). The worst case bound of 2 for the tree heuristic has been known for years. The parsimonious property and our proof of the bound is due to Goemans and Bertsimas (1990). The Held and Karp relaxation for the TSP first appeared in Held and Karp (1971). Recently, researchers have developed improved worst case heuristics for the Steiner problem. Zelikovsky (1993) derives a bound of 11/6, and Berman and Ramaiyer (1992) show how to reduce it to about 1.75.

Theorem 6.12 is due to Bienstock et al. (1993). Geomans and Williamson (1992) have developed a heuristic with a worst-case bound of 2. When we

must pay to include a node in the Steiner tree,i.e., the objective function is of the form $\sum_{e \in E} w_e + \sum_{i \in V} \pi_i z_i$ with each $\pi \geq 0$, there is little hope of finding a heuristic with a constant worst-case bound. Klein and Ravi (1993) have presented a heuristic for which $z^H / z^{NWST} \leq 2 \log |T|$.

Section 7. The dynamic programming algorithm for the OSP and the polyhedral proofs of Theorems 1 and 2 appear in Barany et al. (1986). The relationship between subtree of tree incidence matrices and clique matrices of chordal graphs appears in Golumbic (1980). Since chordal graphs are perfect, this analysis provides an alternative proof of Theorem 1. For the extensive literature on facility location on trees, see Mirchandani and Francis (1990). Results for the Constrained Subtree Packing problem OCSP are based on Aghezzaf et al. (1992). Balakrishnan et al. (1991) report on computational experience, using Lagrangian relaxation, with the telecommunications model cited in Section 1.

Section 8. The equality of the objective values of the three algorithms can be derived by consulting standard texts. The important question of how to continue when the column generation approach gives a fractional solution has received some attention in Vance et al. (1992) and in Hansen et al. (1992).

During the last decade, researchers have intensively studied the multi-item lotsizing problem. Work on the polyhedral structure of the single-item problem can be found in Bárány et al. (1984), Van Hoesel et al. (1991), and Leung et al. (1989). Thizy and Van Wassenhove (1986) have used the Lagrangian relaxation approach for a closely related multi-item problem. Cattryse et al. (1990) have examined heuristics based on a column generation approach, and Pochet and Wolsey (1991) have reported computational results with the cutting plane approach.

Wagner and Whiten (1958) proposed a dynamic programming algorithm for solving the single-item dynamic lotsizing problem. Aggarwal and Park (1993), Federgrun and Tsur (1991), Wagelmans et al. (1992) have proposed very efficient algorithms for this problem. Their analysis shows how to exploit special structure to develop algorithms that are more efficient than those that apply to general trees.

The valid inequalities and the column generation approach described for

the clustering problem can be found in Johnson et al. (1991). The literature on the polyhedral structure of cut polytopes is extensive. Work closely related to the clustering problem includes Chopra and Rao (1993), Groetschel and Wakayabashi (1989), and de Souza et al. (1993).

Araque et al. (1990) have studied the polyhedral structure of the capacitated tree problem and the unit demand vehicle routing problem including the inequalities we have presented. Araque (1989) has studied the use of these inequalities for solving unit demand capacitated vehicle routing problems, but little or nothing has been reported for non-unit demands. However, several papers contain results on the polyhedral structure of the VRP polyhedron with constant capacity but arbitrary demands, including Cornuéjols and Harche (1993). Our presentation of the comb inequalities is taken from Pochet (1992). Gavish (1983,1984) has designed several algorithms for the capacitated tree problem including the approach we described.

Work on the Steiner tree packing polyhedron is due to Groetschel at al.(1992a, 1992b). These authors have developing a branch and cut approach based on valid inequalities and separation heuristics including both Steiner tree inequalities and the cycle inequalities we have presented; they solve seven unsolved problems from the literature all to within 0.7% of optimality (four to optimality).

Section 9. The results in this section are drawn from Balakrishnan et al. (1992a, 1992b, 1993) who treat not only the tree-on-tree problem, but also more general problem of "overlaying" solutions to one problem on top of another other. Previously, Iwainsky (1985) and Current et al. (1986) had introduced and treated the tree-on tree problem and a more specialized path on tree problem. Duin and Volgenant (1989) have shown how to convert the tree-on-tree problem into an equivalent Steiner tree problem. Therefore, in principle, any Steiner tree algorithm is capable of solving these problems. The computational experience we have cited in Section 9 (see Balakrishnan et al. (1993)) uses a specialized dual-ascent approach directly on the tree-on-tree formulation.

Acknowledgments. We are grateful to Michel Goemans, Leslie Hall, Prakash Mirchandani, and S. Raghavan for constructive feedback on an earlier version of this paper.

References

A. Aggarwal and J. Park (1993), Improved Algorithms for Economic Lot-Size Problems, Operations Research 41, 549-571.

S. Arnborg, J. Lagergren and D. Seese (1991), Easy problems for tree decomposable graphs, J. of Algorithms 12, 308-340

R. Ahuja. T. Magnanti, and J. Orlin (1993), Network Flows: Theory, Algorithms, and Applications, Prentice Hall.

J.R. Araque (1989), Solution of a 48-city Routing problem by Branch-and-cut, Unpublished manuscript, Department of Applied Mathematics and Statistics, SUNY at Stony Brook (Stony Brook, New York).

J.R. Araque, L.A. Hall and T.L. Magnanti (1990), Capacitated Trees, Capacitated Routing and Associated, Polyhedra, Core DP 9061, Louvain-la-Neuve, Belgium

A. Balakrishnan, T. Magnanti and P. Mirchandani (1992a), Modeling and Heuristic Worst-case Performance Analysis of the Two-level Network Design Problem, Working Paper # 3498-92-MSA, MIT Sloan School of Management, (to appear in Management Science).

A. Balakrishnan, T. Magnanti and P. Mirchandani (1992b), A Dual-based Algorithm for Multi-level Network Design, Working Paper # 3365-91-MSA, Sloan School of Management, (to appear in Management Science).

A. Balakrishnan, T. Magnanti and P. Mirchandani (1993), Heuristics, LPs, and Generalizations of Trees on Trees, Working Paper OR-275-93, Operations Research Center.

A. Balakrishnan, T.L. Magnanti and R.T. Wong (1991), A Decomposition Algorithm for Expanding Local Access Telecommunications Networks, Report OR 244-91, MIT (to appear in Operations Research).

I. Bárány, J. Edmonds and L.A. Wolsey (1986), Packing and Covering a Tree by Subtrees, Combinatorica 6, 245-257.

I. Bárány, T.J. Van Roy and L.A. Wolsey (1984), Uncapacitated Lot-Sizing: The Convex Hull of Solutions, Mathematical Programming Study 22, 32-43.

J.E. Beasley (1989), An SST-based Algorithm for the Steiner problem on graphs, Networks 19 1-16

P. Berman and V. Ramaiyer (1992), Improved Approximation of the Steiner Tree Problem, Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, 325-334.

C. Berge and A. Ghouila-Houri (1962), Programming, Games, and Transportation Networks, John Wiley and Sons.

D. Bertsekas and R. Gallager (1992), Data Networks (2nd ed.), Prentice-Hall.

D.Bienstock, M. Goemans, D. Simchi-Levi and D. Willianson, A Note on the Prize-Collecting Travelling Salesman Problem (1993), Mathematical Programming, 59, 413-420.

L. Bodin, B. Golden, A. Assad and M. Ball, Routing and Scheduling of Vehicles and Crews: The State of the Art, Computers and Operations Research 10, 69-211.

O. Boruvka (1926), Příspekěv k řešení otázky ekonomické stavby elektrovodních sítí, Elektrotechnicky Obzor 15, 153-154.

P.M. Camerini, G. Galbiati and F. Maffioli (1984), The Complexity of Weighted Multi-constrained Spanning Tree Problems, Colloquia Mathematica Societatis János Boltai 44, Theory of Algorithms Pécs, (Hungary).

D. Cattryse, J. Maes and L.N. Van Wassenhove (1990), Set Partitioning and Column Generation Heuristics for Capacitated Dynamic Lot-Sizing, European J. of Operations Research 46, 38-47.

S. Chopra and E. Gorres (1990), On the Node Weighted Steiner Tree Problem, Department of Managerial Economics and Decision Sciences, J. Kellogg School of Management, Northwestern University.

S. Chopra and M.R. Rao (1988a), The Steiner Tree Problem I: Formulations, Compositions and Extensions of Facets, New York University, to appear in Mathematical Programming.

S. Chopra and M.R. Rao (1988b), The Steiner Tree Problem II: Properties and Classes of Facets, New York University, to appear in Mathematical Programming.

S. Chopra and M.R. Rao (1993), The Partition Problem, Mathematical Programming 59, 87-116.

S. Chopra, E.R. Gorres and M.R. Rao, Solving the Steiner Tree problem on a graph using Branch and Cut, ORSA J. of Computing 4, 320-335 (1992).

S.A. Cook (1971), The Complexity of Theorem-Proving Procedures, Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, 151-158 ACM.

G. Cornuéjols and F. Harche (1993), Polyhedral Study of the Capacitated Vehicle Routing Problem, Mathematical Programming 60, 21-52.

J.R. Current, C.S. Revelle and J.L. Cohon (1986), The Hierarchical Network Design Problem, European Journal of Operations Research 27, 57-66.

C. de Souza, C. Ferreira, A. Martin, R. Weismantel and L.A. Wolsey (1993), Valid Inequalities and Separation for Node-Weighted Partitioning Problems, in preparation.

C. Duin and A. Volgenant (1989), Reducing the Hierarchical Network Design Problem, European Journal of Operations Research 39, 332-344.

J. Edmonds (1967), Optimum Branchings, J. of Research of the National

Bureau of Standards 71B, 233-240

J. Edmonds (1970), Submodular Functions, Matroids and Certain Polyhedra, in Combinatorial Structures and their Applications, R. Guy et al. eds. Gordon and Breach, 69-87.

J. Edmonds (1971), Matroids and the Greedy Algorithm, Mathematical Programming 1, 127-136.

G.D. Eppen and R.K. Martin (1987), Solving Multi-Item Lot-Sizing Problems using Variable Definition, Operations Research 35, 832-848.

A. Federgrun and M. Tsur (1991), A Simple Forward algorithm to Solve General Dynamic Lot-Size Models with n Periods in $O(nlogn)$ or $O(n)$ time, Management Science 37, 909-925.

R.L. Francis, L.F. McGinnis and J.A. White (1992), Facility Layout and Location: An Analytic Approach, Prentice-Hall.

B. Gavish (1983), Formulations and Algorithms for the Capacitated Minimal Directed Tree Problem, J. of the ACM 30 , 118-132

B. Gavish (1984), Augmented Lagrangean Based Algorithms for Centralized Network Design, IEEE Transactions Comm. 33, 1247-1275

M.X. Goemans (1994), Arborescence Polytopes for Series-Parallel Graphs, Discrete Applied Mathematics.

M.X. Goemans (to appear), The Steiner Polytope and Related Polyhedra, Mathematical Programming, 63, 157-182.

M.X. Goemans (1992), Personal Communication.

M.X. Goemans and D. Bertsimas (1993), Survivable Networks, Linear Programming Relaxations and the Parsimonious Property, Mathematical Programming, 060, 145-166.

M. Goemans and D. Willianson (1992), A General Approximation Technique for Constrained Forest Problems, Proceedings of the 3rd ACM-SIAM Symposium on Discrete Algorithms, to appear in SIAM J. on Computing.

M.X. Goemans and Y-S. Myung (1993), A Catalog of Steiner Tree Formulations, Networks 23, 19-28

M.C. Golumbic (1980), Algorithmic Graph Theory and Perfect Graphs, Academic Press.

S. Graves, A.H.G. Rinnooy Kan, and P. Zipkin, eds. (1993), Logistics of Production and Inventory, Vol. 4 of the Handbooks of Operations Research and Management Science, North-Holland.

H. Groeflin and T.M. Liebling (1981), Connected and Alternating Vectors: Polyhedra and Algorithms, Mathematical Programming 20, 233-244.

M. Groetschel, A. Martin and R. Weismantel (1992), Packing Steiner Trees: Polyhedral Investigations, Preprint SC 92-8 Konrad-Zuse Zentrum, Berlin.

M. Groetschel, A. Martin and R. Weismantel (1992), Packing Steiner Trees: a Cutting Plane Algorithm and Computational Results, Preprint SC 92-9 Konrad-Zuse Zentrum, Berlin.

M. Groetschel and Y. Wakayabashi (1989), A Cutting Plane Algorithm for a Clustering Problem, Mathematical Programming 45, 59-96.

P. Hansen, B. Jaumard and M.P. de Aragao (1992), Mixed Integer Column Generation Algorithms and the Probabilistic Maximum Satisfiability Problem, Proceedings of 2nd IPCO Conference, eds., E. Balas, G. Cornuéjols and R. Kannan, Carnegie Mellon University, 165-180.

J.A. Hartigan (1975), Clustering Algorithms, John Wiley and Sons.

M. Held and R.M. Karp (1971), The Travelling Salesman Problem and Minimum Spanning Trees: Part II, Mathematical Programming 1, 6-25.

T.C. Hu and E.S. Kuh, eds. (1985), VLSI Layout: Theory and Design, IEEE Press, New York.

F.W. Hwang and D.S. Richards (1992), Steiner Tree Problems, Networks, 22, 55-90.

F.W. Hwang, D.S. Richards and P. Winter (1992), The Steiner Tree Problem, Annals of Discrete Mathematics, 53, North-Holland.

A. Iwainsky (1985), Optimal Trees–A Short Overview on Problem Formulations, in Optimization of Connection Structures in Graphs, Central Institute of Cybernetics and Information Processes, Berlin.

V. Jarník (1930), Ojistém problému minimálním, Axta Societatis Natur. Moravicae 6, 57-63.

E.L. Johnson, A. Mehrotra and G.L. Nemhauser (1993), Min-Cut Clustering, Mathematical Programming 62, 133-151

R. M. Karp (1972), Reducibility among Combinatorial Problems, in Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, eds., Plenum Press 85-103.

P. Klein and R. Ravi (1993), A nearly best possible approximation for node weighted Steiner trees, in Proceedings of 3rd IPCO Conference, G. Rinaldi and L.A. Wolsey, eds. 323-331.

B. Korte, H.J. Promel and A. Steger, Steiner Trees in VLSI Layout, in Paths, Flows and VLSI Layout, B. Korte, L. Lovász, H.J. Promel, A. Schijver eds., Springer 1990, 185-214.

J.B. Kruskal (1956), On the Shortest Spanning Tree of a Graph and the Travelling Salesman Problem, Proceedings of the American Mathematical Society 7, 48-50.

E.L. Lawler. J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, eds.

(1985), The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, John Wiley and Sons.

E.L. Lawler (1976), Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston.

T.E. Leighton (1983), Complexity Issues in VSLI, M.I.T. Press.

T. Lengauer (1990), Combinatorial Algorithms for Integrated Circuit Layout, John Wiley and Sons.

J. Leung, T. Magnanti and R. Vachani (1989), Facets and Algorithms for Capacitated Lot Sizing, Mathematical Programming 45, 331-360

L. Lovász (1979), Graph Theory and Integer Programming, Annals of Discrete Mathematics 4, 141-158.

A. Lucena and J.E. Beasley (1992), A Branch and Cut Algorithm for the Steiner Problem in Graphs, Report, The Management School, Imperial College.

N. Maculan (1987), The Steiner Problem in Graphs, Annals of Discrete Mathematics 31, 185-212.

T. Magnanti and R. Vachani (1990), A Strong Cutting Plane Algorithm for Production Scheduling with Changeover Costs, Operations Research 38, 456-473.

F. Margot, A. Prodon, Th. M. Liebling (1994), Tree Polytope on 2-trees, Mathematical Programming, 63, 183-192.

R.K. Martin (1987), Generating Alternative Mixed-Integer Programming Models using Variable Redefinition, Operations Research 35, 331-359.

R.K. Martin (1991), Using Separation Algorithms to Generate Mixed Integer Model Reformulations, Operations Research Letters 10, 119-128

Mirchandani P.B. and R.L. Francis (1990), Discrete Location Theory, John Wiley and Sons.

Nemhauser, G.L., and L.A. Wolsey (1988), Integer and Combinatorial Optimization, John Wiley and Sons.

Y. Pochet (1992), A Common Derivation of TSP and VRP Inequalities, Talk presented at the 3rd Cycle FNRS day on Combinatorial Optimization, Core, Louvain-la-Neuve, December 11th

Y. Pochet and L.A. Wolsey (1991), Solving Multi-Item Lot-Sizing Problems with Strong Cutting Planes, Management Science 37,53-67.

R.C. Prim (1957), Shortest Connection Networks and some Generalizations, Bell System Technological Journal 36, 1389-1401.

A. Prodon, T.M. Liebling and H. Groeflin (1985), Steiner's Problem in Two Trees, RO 850315, Department of Mathematics, Ecole Polytechnique Fédérale de Lausanne.

M. Schwartz (1977), Computer and Communication Network Design and Analysis, Prentice-Hall.

K. Takamizawa, T. Nishizeki and N. Saato (1982), Linear Time Computability of Combinatorial Problems on Series-parallel Graphs, J. ACM 29, 623-641.

A.S. Tanenbaum (1989), Computer Networks (2nd ed.), Prentice-Hall.

J.M. Thizy and L.N. Van Wassenhove (1986), A Subgradient Algorithm for the Multi-item Capacitated Lot-Sizing Problem, IIE Transactions 18, 114-123.

P.H. Vance, C. Barnhart, E.L. Johnson and G.L. Nemhauser (1992), Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound, Computational Optimization Center COC-92-09, Georgia Institute of Technology.

C.P.M. van Hoesel, A.P.M. Wagelmans and L.A. Wolsey (1991), Economic Lot-Sizing with Start-Up Costs: The Convex Hull, Core Discussion Paper 9109, Université Catholique de Louvain (to appear in SIAM J. of Discrete Mathematics).

A. Volgenant (1989), A Lagrangian Approach to the Degree-constrained Minimum Spanning Tree Problem, European Journal of Operations Research 39, 325-331.

H.M. Wagner and T.M. Whitin (1958), A Dynamic Version of the Economic Lot Size Model, Management Science 5, 89-96.

A.P.M. Wagelmans, C.P.M. van Hoesel and A.W.J. Kolen (1992), Economic Lot-Sizing: an $O(nlogn)$ Algorithm that runs in Linear Time in the Wagner-Whitin Case, Operations Research 40, Supplement 1, 145-156

P. Winter (1987) Steiner Problem in Networks: a Survey, Networks 17, 129-167.

R.T. Wong (1980), Integer Programming Formulations of the Travelling Salesman Problem, Proceedings of IEEE Conference on Circuits and Computers, 149-152.

R.T. Wong (1984), A Dual Ascent Approach for Steiner Tree Problems on a Directed Graph, Mathematical Programming 28, 271-287

A.Z. Zelikovsky (1993), An 11/6 Approximation Algorithm for the Network Steiner Problem, Algorithmica, 9, 463-470.