Approximation algorithms for network problems

J. Cheriyan [1] R. Ravi [2]

September 1998

[1]C & O Dept., Univ. of Waterloo, Waterloo, CANADA, N2L3G1, jcheriyan@dragon.uwaterloo.ca
[2]GSIA, Carnegie Mellon University, Pittsburgh, PA 15213-3890, ravi@cmu.edu

Contents

List of Figures

List of Tables

Chapter 1

Basic Graph Theory and Shortest-Paths Algorithms

This chapter gives some basic information from graph theory and graph algorithms, followed by a discussion on the shortest paths problem. Since this chapter is based on well-known, standard results, we do not give the original references, instead we use standard texts in the area.

For graph theory, we refer to Bollobas [2] and Bondy & Murty [3], and readers interested in more informatiion are refered to those texts. For the design and analysis of algorithms, we refer to the text by Cormen, Leiserson and Rivest [4].

Our discussion on the shortest paths problem is brief but self-contained. First, Bellman's inequalities, node potentials, and shortest paths arborescences (directed trees) are discussed, followed by descriptions of Dijkstra's algorithm and the Floyd-Warshall algorithm. Our treatment is based on several texts, namely, Ahuja, Magnanti & Orlin [1, Chapter 4], Cormen et al [4, Chapters 25 & 26], Lawler [8], and Larson & Odoni [6, Chapter 6.2]. Readers interested in more informatiion are referred to one of the above texts. Readers interested in a compact and accessible introduction to network algorithms are referred to the chapter by Larson & Odoni.

1.1 Graph theory terminology

A graph $G = (V(G), E(G))$ consists of a finite set $V(G)$ of *nodes* and a finite set $E(G)$ of unordered pairs of (distinct) nodes called *edges*. (For the most part, we study simple graphs having neither loops nor multiple edges, so our definition does not allow these.) Usually, there is only one graph G under consideration, and since there is no ambiguity, we use V instead of $V(G)$ and E instead of $E(G)$. The number of nodes, $|V|$, is usually denoted by n, and the number of edges, $|E|$, is usually denoted by m. For an edge $e = \{x, y\}$, nodes x and y are said to be *incident* to e, and x and y are called the *ends* of e. We also denote the edge as xy. Nodes x and y are called *adjacent* and are said to be *joined* by the edge e. Figure 1.1 shows an example of a graph; here $n = 7$ and $m = 9$.

The *degree* of a node v, denoted $\deg(v)$, is the number of edges incident to v. Here are the degrees of the nodes for the example in Figure 1.1.

node v	s	a	b	c	d	e	t
$\deg(v)$	2	4	2	3	2	3	2

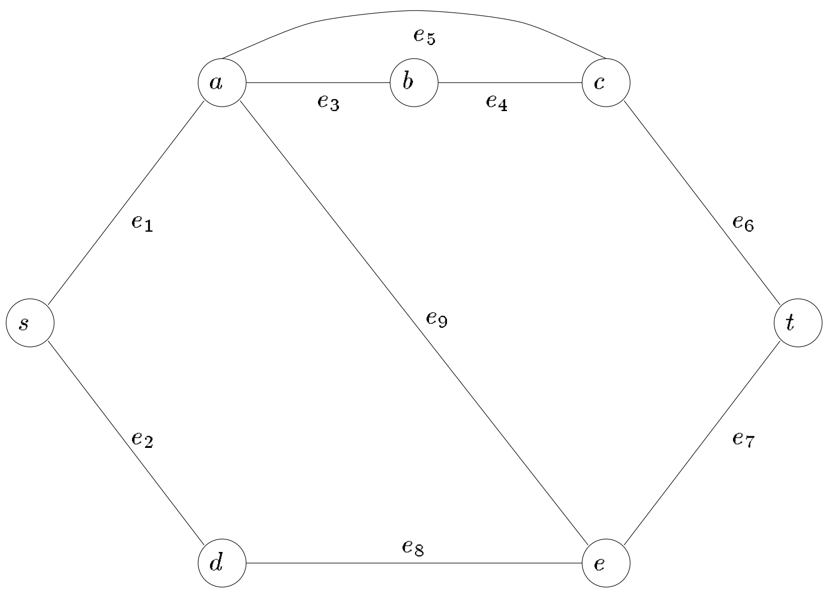


Figure 1.1: An example of a graph.

Graphs may be represented by matrices. The *adjacency* matrix $= (a_{ij})$ of a graph G has its rows and columns indexed by $V(G)$ and is defined

$$a_{ij} = \begin{cases} 1 & \text{if node } i \text{ and node } j \text{ are adjacent,} \\ 0 & \text{otherwise.} \end{cases}$$

Here is the adjacency matrix for the example in Figure 1.1.

	s	a	b	c	d	e	t
s	0	1	0	0	1	0	0
a	1	0	1	1	0	1	0
b	0	1	0	1	0	0	0
c	0	1	1	0	0	0	1
d	1	0	0	0	0	1	0
e	0	1	0	0	1	0	1
t	0	0	0	1	0	1	0

The *incidence* matrix $B = (b_{ij})$ of a graph G has its rows indexed by $V(G)$ and its columns indexed by $E(G)$ and is defined

$$b_{ij} = \begin{cases} 1 & \text{if node } i \text{ is incident to edge } j, \\ 0 & \text{otherwise.} \end{cases}$$

Here is the incidence matrix for the example in Figure 1.1.

$$\begin{array}{c|ccccccccc|}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 \\ \hline
s & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
b & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
c & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
d & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
e & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
t & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline
\end{array}$$

A *subgraph* $H = (V(H), E(H))$ of a graph G is a graph with node set $V(H) \subseteq V(G)$ and edge set $E(H) \subseteq E(G)$. Since H is a graph, every element of $E(H)$ is an unordered pair of elements of $V(H)$; in other words, both ends of every edge of H are nodes of H.

A subgraph H of a graph G is called *spanning* if $V(H) = V(G)$.

A subgraph H of a graph G is called an *induced* subgraph if $E(H)$ contains every edge of $E(G)$ such that both ends of the edge are in $V(H)$. For a node set $S \subseteq V(G)$, the subgraph of G *induced by* S means the induced subgraph H of G such that $V(H) = S$.

Given a graph $G = (V(G), E(G))$, a *walk* W is a sequence of nodes and edges

$$v_0 e_1 v_1 e_2 v_2 \cdots e_\ell v_\ell$$

where edge e_i, $1 \leq i \leq \ell$, is incident to nodes v_{i-1} and v_i. A walk is also denoted by its node sequence, e.g., $W = v_0 v_1 \ldots v_\ell$. The node v_0 is called the *start node* of the walk W, and v_ℓ is called the *end node* of W. Also, the walk W is said to *join* v_0 and v_ℓ, and W is called a v_0-v_ℓ walk.

A *trail* is a walk with no repeated edges.

A *path* is a walk with no repeated nodes.

The *length* of a walk (or trail or path) is defined to be the number of edges in it, e.g., the length of W is ℓ.

A *cycle* is a trail of positive length whose start node and end node are the same but all other nodes are distinct.

For example, in Figure 1.1, $s\ e_1\ a\ e_5\ c\ e_4\ b\ e_3\ a\ e_5\ c\ e_6\ t$ is a walk of length 6, $s\ e_1\ a\ e_5\ c\ e_4\ b\ e_3\ a\ e_9\ e\ e_7\ t$ is a trail of length 6, $s\ e_1\ a\ e_5\ c\ e_6\ t$ is a path of length 3, and $a\ e_5\ c\ e_4\ b\ e_3\ a$ is a cycle of length 3.

A graph is said to be *connected* if for every pair of nodes v, w there is a walk joining v and w.

A (connected) *component* of a graph is defined to be a subgraph induced by a maximal set S of nodes such that there is a walk joining every pair of nodes in S.

A graph that has no cycles is called a *forest.*

A connected graph that has no cycles is called a *tree.*

Given a node set $Q \subseteq V$, $\delta(Q)$ denotes the set of all edges with one end in Q and the other end in $V \backslash Q$. (Informally, $\delta(Q)$ is the "boundary" of the node set Q in G.) For example, in Figure 1.1, taking $Q = \{a, c, d, e\}$ we have $\delta(Q) = \{e_1, e_2, e_3, e_4, e_6, e_7\}$.

A cut consists of all edges that have one end in Q and the other end in $V \backslash Q$, where Q is a node set such that $Q \neq \emptyset$ and $Q \neq V$; this cut is denoted $(Q, V \backslash Q)$. Clearly, if $\emptyset \neq Q \neq V$, then $\delta(Q) = (Q, V \backslash Q)$. Another example for Figure 1.1 is $(\{s, a, b, c\},\ \{d, e, t\}) = \{e_2, e_6, e_9\}$.

A *network* is a graph (or directed graph), together with zero or more functions from the nodes to the real numbers, and zero or more functions from the edges to the real numbers. For the most part, we study networks consisting of a graph $G = (V(G), E(G))$ and a single real-valued function on the edges $c : E(G) \rightarrow \Re$ called the cost. The cost of an edge $e = xy$ is denoted by either $c(xy)$ or c_{xy}. For a subgraph H of G, the cost of H is defined to be the sum of the costs of all the edges in H and is denoted $c(H)$; so, $c(H) = \sum_{e \in E(H)} c(e)$.

1.2 The running time of algorithms

The number of steps executed by an algorithm is called the runing time. This is justified since each step takes roughly the same time to execute.
Big-Oh notation: Given two functions f and g from the natural numbers to the real numbers, f is said to be $O(g)$ if there exist positive numbers n_0 and k such that

$$\forall n \geq n_0 \ : \ f(n) \leq k \cdot g(n).$$

Informally, when we use this notation, we keep only the asymptotically largest terms, and then drop their coefficients.

$$\begin{aligned} & Examples: \\ 2n^3 - 5n^2 - 2n + 100 &= O(n^3). \\ 10n^2 - 1000n + 10^6 n^{1/2} &= O(n^2). \\ \frac{\log n}{1000} + 10^6 &= O(\log n). \\ \frac{n^2 \log n}{1000} + 10^6 n^2 + 10^{12} n^{1.99} &= O(n^2 \lg n). \end{aligned}$$

1.3 Shortest Paths Problems

Consider a network consisting of a graph $G = (V, E)$ and a nonnegative cost function c on the edges. That is, each edge e has a nonnegative real number $c(e)$ assigned to it. Assume that G is connected; otherwise, we focus on a connected component of G. The *cost* of a path (or walk) $P = v_0 e_1 v_1 e_2 v_2 \ldots v_{\ell-1} e_\ell v_\ell$ is defined to be

$$c(P) = \sum_{i=1}^{\ell} c(e_i).$$

The general problem we consider is to find a path P joining specified nodes such that $c(P)$ is minimum among all such paths. Such a path is called a *shortest path.*

Two specific problems of interest are:

SINGLE SOURCE:
Given a specified node s, find a shortest path from s to every other node of G.

ALL PAIRS:
Find shortest paths joining every pair of nodes in G.

1.3.1 Bellman's inequalities, node potentials and reduced costs

The basic idea behind all methods for solving shortest paths problems is this. Let s be a specified start node. Suppose we know that there is a path from s to v of cost $d(v)$ for each node v, and we find an edge vt satisfying $d(v) + c(vt) < d(t)$. Since appending vt to the path from s to v gives a path from s to t, we know that there is a shorter path to t, of cost $d(v) + c(vt)$. On the other hand, if each edge vw satisfies $d(v) + c(vw) \geq d(w)$, then the method of constructing a path to t by appending an edge vt to the path to v does not improve on the cost $d(t)$.

To study the properties of the values $d(v)$ $(v \in V)$, we focus on *Bellman's inequalities*:

$$d(s) = 0 \tag{1.1}$$

$$d(w) \leq d(v) + c(vw) \text{ for all } vw \in E. \tag{1.2}$$

The values $d(v)$ $(v \in V)$ satisfying Bellman's inequalities are called *node potentials.*

These inequalities and their solutions are fundamental for shortest path problems. We give some elementary but useful results about node potentials.

Proposition 1.1 *If there exists a solution to Bellman's inequalities, then G, c has no edges of negative cost.*

Lemma 1.2 *If $d : V \to \Re$ is a solution to Bellman's inequalities, then for each node v, $d(v)$ is less than or equal to the cost of any walk W from s to v.*

Proof: Let $s = v_0, v_1, \ldots, v_k = v$ be the sequence of nodes in W. We add the inequalities (1.2) for the edges of W:

$$\begin{array}{rcl} d(v_1) - d(v_0) & \leq & c(v_0 v_1) \\ d(v_2) - d(v_1) & \leq & c(v_1 v_2) \\ \cdots & . & . \\ \cdots & . & . \\ \cdots & . & . \\ d(v_k) - d(v_{k-1}) & \leq & c(v_{k-1} v_k) \\ \hline d(v_k) - d(s) & \leq & \sum_{i=1}^{k} c(v_{i-1} v_i) = c(W). \end{array}$$

We obtain $c(W) \geq d(v) - d(s)$, and hence by equation (1.1), we find that $d(v) \leq c(W)$. ∎

Proposition 1.3 *Let $d : V \to \Re$ be a solution to Bellman's inequalities and let E' be the set of edges satisfying (1.2) with equality, i.e., $E' = \{uv \in E \ : \ d(v) - d(u) = c(uv)\}$. Let E^* be obtained by orienting each edge $e \in E'$ from the end with lower d-value to the end with higher d-value, $E^* = \{(u,v) \ : \ uv \in E'$ and $d(u) \leq d(v)\}$. Let G^* be the directed graph (V, E^*). For each node v, every path from s to v in G^* is a shortest s-v path of G, c; if there is at least one such path, then $d(v)$ is the cost of a shortest s-v path.*

Proof: We add the inequalities (1.2) for the edges of a (directed) path P from s to v such that all of P's edges are in E^*, and obtain $d(v) = c(P)$. By Lemma 1.2, the cost of a shortest path must be at least $d(v)$, so P is a shortest path from s to v. ∎

Let $d : V \to \Re$ be a solution to Bellman's inequalities. Define the *reduced cost* of an edge vw, denoted $\overline{c}(vw)$, with respect to the node potentials d to be

$$\overline{c}(vw) = c(vw) + d(v) - d(w).$$

The reduced cost of a path $P = v_0 v_1 \ldots v_\ell$, $\overline{c}(P)$, and the reduced cost of a cycle $Q = v_0 v_1 \ldots v_\ell v_0$ ($v_{\ell+1} = v_0$), $\overline{c}(Q)$, are defined similarly to the cost of a path and of a cycle, namely, $\overline{c}(P) = \sum_{i=1}^{\ell} \overline{c}(v_{i-1} v_i)$ and $\overline{c}(Q) = \sum_{i=1}^{\ell+1} \overline{c}(v_{i-1} v_i)$.

Proposition 1.4 *Let $d : V \to \Re$ be a set of node potentials, and for each edge vw, let its reduced cost be $\overline{c}(vw) = c(vw) + d(v) - d(w)$. The reduced costs satisfy the following properties.*

(1) $\overline{c}(vw)$ is nonnegative for each edge vw.

(2) For any cycle Q, $\overline{c}(Q) = c(Q)$.

(3) Let P be a path with start node v_0 and end node v_ℓ. Then $\overline{c}(P) = c(P) + d(v_0) - d(v_\ell)$.

Proof: The proof is straightforward, using the fact that d satisfies Bellman's inequalities. For parts (2) and (3) of the lemma, add the reduced costs of the edges in the cycle Q or the path P as in Lemma 1.2. ∎

1.3.2 Shortest paths arborescence

Before presenting an algorithm for the single-source shortest paths problem, we state a few more results.

Proposition 1.5 *Suppose that $G,\ c$ has no edges of negative cost. Let W be a walk from v_0 to v_ℓ. Then there exists a path P from v_0 to v_ℓ that is a subsequence of W such that $c(P) \leq c(W)$.*

Proposition 1.6 *Suppose that $G,\ c$ has no edges of negative cost. Let $P = v_0 v_1 \cdots v_\ell$ be a shortest path from v_0 to v_ℓ. Then for any i and j, $0 \leq i < j \leq \ell$, $P' = v_i v_{i+1} \ldots v_j$ is a shortest path from v_i to v_j.*

The optimal solutions to the single-source shortest paths problem have a particularly simple structure. An *s-arborescence* T of a graph $G = (V, E)$, where $s \in V(T)$, is a directed graph such that the node set $V(T)$ equals $V(G)$, the number of edges is $|E(T)| = |V(T)| - 1$, corresponding to every (directed) edge $(v, w) \in E(T)$ there is an edge $vw \in E(G)$, and for each node $v \in V(T) \backslash \{s\}$ there is exactly one directed path in T from s to v. Alternatively, T is a directed graph with node set $V(G)$ that has no directed cycles such that corresponding to every (directed) edge $(v, w) \in E(T)$ there is an edge $vw \in E(G)$, there are no edges of T entering node s, and for each node $v \in V \backslash \{s\}$, there is exactly one edge in T entering v. Roughly speaking, T is a spanning tree such that all its edges are directed "away" from s. See Figure 1.2. An s-arborescence T of a network $G,\ c$ is called a *shortest paths s-arborescence* of $G,\ c$ if for each node $v \in V(T)$, the path in T from s to v is a shortest s-v path of $G,\ c$. For each node v, if $v \neq s$, we take $p(v)$ to be the unique node such that $(p(v), v)$ is an edge of T, and we take $p(s) = \emptyset$. We call $p(v)$ the *predecessor* of v. If we have a shortest paths s-arborescence T of $G,\ c$, then a shortest path from s to any node v can be obtained by starting at v and repeatedly moving to the predecessor until we get to node s, i.e., the path $s\ \ldots\ p(p(p(v)))\ p(p(v))\ p(v)\ v$ is a shortest s-v path of $G,\ c$.

Proposition 1.7 *Suppose that $G,\ c$ has no edges of negative cost, and that G is connected. Then there exists a shortest paths s-arborescence for $G,\ c$.*

A shortest paths s-arborescence can be used to find a "tight" solution to Bellman's inequalities; conversely, Bellman's inequalities can be used to check whether a given s-arborescence is a shortest paths s-arborescence.

Theorem 1.8 *Suppose that G is connected and that $G,\ c$ has no edges of negative cost. Then there exists a solution $d : V \rightarrow \Re$ to Bellman's inequalities such that for each node v, $d(v)$ is the cost of a shortest path from s to v. Moreover, an s-arborescence T of G is a shortest paths s-arborescence if and only if each edge (v, w) of T satisfies (1.2) with equality, i.e., $d(w) = d(v) + c(vw)$.*

Using the theorem, it is easy to verify whether an s-arborescence T is a shortest paths s-arborescence. For each node v, we take $d'(v)$ to be the cost of the path in T from s to v (we take $d'(s) = 0$). We check whether $d' : V \rightarrow \Re$ satisfies Bellman's inequalities. If yes, then T is a shortest paths s-arborescence.

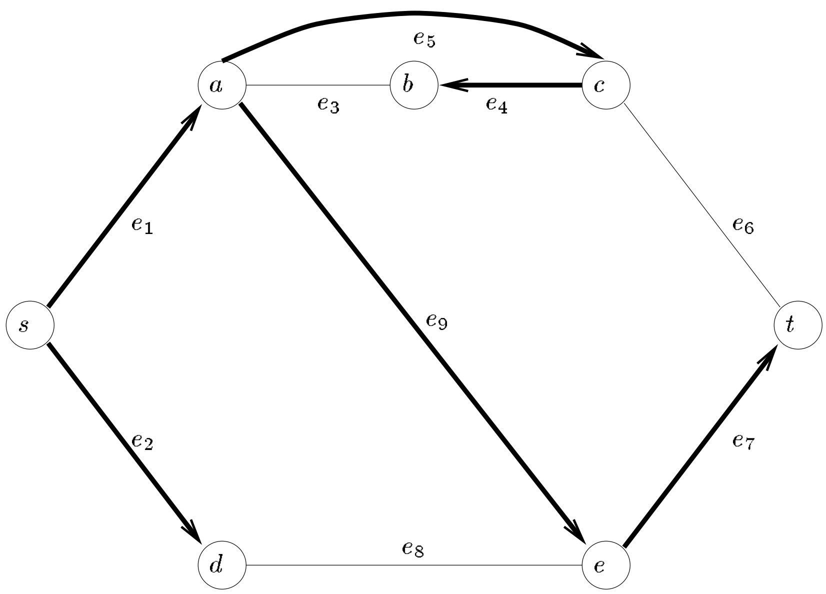


Figure 1.2: An s-arborescence for the graph in Figure 1.1 is indicated by thick lines.

1.3.3 Dijkstra's algorithm

Dijkstra's algorithm efficiently solves the single-source shortest paths problem, assuming that the edge costs are nonnegative. For each node u, let $d(u)$ denote the cost of a shortest path of $G,\ c$ from s to u. The main idea of the algorithm is this: Suppose that for some node set $S \subseteq V$ we know the value $d(u)$ for each $u \in S$. Of course, we have $s \in S$ and $d(s) = 0$. For each node $v \in V \backslash S$, we compute a "temporary label" $\ell(v)$, where

$$\ell(v) = \min\ \{d(u) + c(uv)\ :\ u \in S, \quad uv \in E\}.$$

(If no edge exists joining a node of S to v, then we define $\ell(v) = \infty$.)

Lemma 1.9 *If v^* is a node in $V\backslash S$ such that $\ell(v^*)$ is minimized over all nodes of $V\backslash S$, then $\ell(v^*) = d(v^*)$, i.e., $\ell(v^*)$ is the cost of a shortest s-v^* path of $G,\ c$.*

Proof: First, note that a shortest path from s to a node $u \in S$ such that $d(u) + c(uv^*) = \ell(v^*)$, plus the edge uv^* gives a walk from s to v^* of cost $\ell(v^*)$. Now let $P = v_0v_1v_2 \dots v_q$ ($s = v_0$ and $v^* = v_q$) be *any* path from s to v^*. To complete the proof, we show that its cost is at least $\ell(v^*)$. Let v_k be the first node of this path not belonging to S. Since the edge costs c are nonnegative, and $\ell(v^*) = \min_{v \in V\backslash S} \ell(v)$, we have

$$c(P)\ \geq\ c(v_0v_1v_2\dots v_k)\ \geq\ d(v_{k-1}) + c(v_{k-1}v_k)\ \geq\ \ell(v_k)\ \geq\ \ell(v^*).$$

In other words, the cost of every s-v^* path of $G,\ c$ is at least $\ell(v^*)$. ∎

Algorithm Dijkstra's Algorithm
input: Graph $G = (V, E)$, nonnegative edge costs c, and start node s; G is connected.
output: A shortest paths s-arborescence of G, c given by $p : V \backslash \{s\} \to V$, and for each node v, the cost of a shortest path from s to v, $d(v)$.

STEP 0: $S := \{s\}$;
$d(s) := 0$; $p(s) := \emptyset$;
for each node $v \in V \backslash S$ **do**
  $\ell(v) := \infty$; $p(v) := \emptyset$;
end; (for)
$v^* := s$; (v^* is last node added to S)
while $S \neq V$ **do**
STEP 1: **for** each edge v^*x **do**
  if $x \in V \backslash S$ and $\ell(x) > d(v^*) + c(v^*x)$
  then $\ell(x) := d(v^*) + c(v^*x)$;
    $p(x) := v^*$;
  end; (if)
 end; (for)
STEP 2: (add node v in $V \backslash S$ with smallest $\ell(v)$ to S)
 find a node $y \in V \backslash S$ with $\ell(y) = \min\{\ell(v) \ : \ v \in V \backslash S\}$;
 $v^* := y$; $S := S \cup \{v^*\}$;
 $d(v^*) := \ell(v^*)$;
end; (while)

Theorem 1.10 *Given a network $G = (V, E)$, c, where c is nonnegative, and a start node s, Dijkstra's algorithm correctly computes a shortest paths s-arborescence. The running time is $O(|V|^2)$.*

Remark: Using the Fibonacci heaps data structure, the running time can be improved to $O(|E| + |V| \log |V|)$.

1.3.4 The Floyd-Warshall algorithm

The Floyd-Warshall algorithm efficiently solves the all-pairs shortest paths problem, assuming that the edge costs are nonnegative. First we order the nodes (arbitrarily) as $v_1, v_2, \ldots, v_n$, where $n = |V|$. An *intermediate* node of a path is any node different from the start node and the end node. For $k = 0, 1, 2, \ldots, n$, we define $d^k(u, v)$ to be the cost of a shortest path from u to v, with the added condition that only nodes $\{v_1, v_2, \ldots, v_k\}$ can be used as intermediate nodes. (If $k = 0$, we say that no nodes can be intermediates, i.e., the paths consist of single edges.)

The idea of the algorithm is that the values $d^k(u, v)$ can be computed from the values of $d^{k-1}(u, v)$: A shortest path from u to v that uses only $\{v_1, v_2, \ldots v_k\}$ as intermediate nodes can either use or not use the node v_k. In the former case, its cost is $d^{k-1}(u, v)$. In the latter, its cost is $d^{k-1}(u, v_k) + d^{k-1}(v_k, v)$.

The algorithm also computes the values $d^k(u,u)$ for all u. All these values remain equal to zero always if and only if $G,\ c$ has no edges of negative cost.

Algorithm Floyd-Warshall Algorithm
input: Graph $G = (V, E)$, and nonnegative costs c on the edges.
output: For each pair of nodes $u,\ v$, finds the minimum cost of a path from u to v.

STEP 0: let $d^0(u,v) = c(uv)$ for all $uv \in E$;
let $d^0(u,u) = 0$ for all $u \in V$;
let $d^0(u,v) = \infty$ for all distinct $u,\ v \in V \times V$ such that $uv \notin E$;
STEP 1: **for** $k = 1, 2, \ldots, |V|$ **do**
  for all $u,\ v \in V \times V$ **do**
    $d^k(u,v) := \min\{d^{k-1}(u,v)\ ,\ d^{k-1}(u,v_k) + d^{k-1}(v_k,v)\}$;
  end; (for)
end; (for)

At termination, if $d^n(u,u) = 0$ for all $u \in V$, then the values $d^n(u,v)$ are the required shortest paths costs.

Theorem 1.11 *Given a network $G = (V, E)$, c, where c is nonnegative, the Floyd-Warshall algorithm correctly computes the cost of a shortest path between every pair of nodes. The running time is $O(|V|^3)$.*

If desired, we can modify the algorithm so that it constructs a shortest paths u-arborescence for each node $u \in V$. This requires each node v to have $|V|$ predecessor pointers $p_u(v)$, each representing the previous node in a shortest path from u to v, for some $u \in V$.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications.* Prentice-Hall, Englewood Cliffs, N. J., 1993.

[2] B. Bollobas, *Graph Theory: An Introductory Course.* Springer-Verlag, New York, 1979.

[3] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications.* Macmillan, London, 1976.

[4] N. Christofides, *Graph Theory: An Algorithmic Approach.* Academic Press, London, 1975.

[5] V. Chvátal, *Linear Programming*, (1983) W. H. Freeman and Company.

[6] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms.* The MIT Press, Cambridge, MA, 1992.

[7] R. C. Larson and A. R. Odoni, *Urban Operations Research.* Prentice-Hall, Englewood Cliffs, N. J., 1981.

[8] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

Chapter 2

The Set Covering Problem

In this note, we look at the classical set covering problem and analyze the performance of the greedy approximation algorithm in three different ways.

2.1 The problem and its complexity

The set covering problem is a fundamental problem in the class of covering problems. Given a finite set X and a family $\mathcal{F} = S_1, S_2, \ldots, S_n$ of subsets of X (i.e., $S_j \subseteq X, \quad j = 1, \ldots, n$) the set covering problem is to find a minimum cardinality $J \subseteq \{1, \ldots, n\}$ such that $\bigcup_{j \in J} S_j = X$. The elements of X are called *points*. Given a $J \subseteq \{1, \ldots, n\}$, a point is said to be *covered* if it belongs to $\bigcup_{j \in J} S_j$. In the minimum-cost set covering problem, each set S_j, $1 \leq j \leq n$, has a cost c_j, and the problem is to find a $J \subseteq \{1, \ldots, n\}$ such that each point is covered and $\sum_{j \in J} c_j$ is minimum.

Define the incidence matrix A of a set covering problem as follows. There are $|X|$ rows in A, one for each point $x_i \in X$, and n columns in A, one for each set S_j. The entry a_{ij} of A (the entry at the intersection of the ith row and the jth column) is one if point x_i is in set S_j, otherwise a_{ij} is zero. Figure 2.1 shows two examples of the set covering problem, along with their incidence matrices.

An integer linear programming formulation of the set covering problem is as follows. For each set S_j, we have a zero-one variable s_j. The intention is that $s_j = 1$ iff set S_j is chosen in the optimal solution. Let $\mathbf{1}^m$ denote a vector with m entries, each of which is 1; it will be clear from the context whether the vector is a row vector or a column vector (either is possible). Let s denote the n-element column vector $\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}$.

$$A = \begin{array}{c|ccc|} & S_1 & S_2 & S_3 \\ \hline x_1 & 1 & 0 & 1 \\ x_2 & 1 & 1 & 0 \\ x_3 & 0 & 1 & 1 \\ x_4 & 0 & 0 & 1 \\ \hline \end{array}$$

$$A = \begin{array}{c|cccccc|} & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 \\ \hline x_1 & 1 & 0 & 1 & 0 & 0 & 0 \\ x_2 & 1 & 0 & 1 & 0 & 0 & 0 \\ x_3 & 0 & 1 & 1 & 0 & 0 & 0 \\ x_4 & 0 & 0 & 1 & 0 & 0 & 1 \\ x_5 & 1 & 0 & 0 & 1 & 0 & 0 \\ x_6 & 1 & 1 & 0 & 1 & 0 & 0 \\ x_7 & 0 & 1 & 0 & 1 & 0 & 0 \\ x_8 & 0 & 0 & 0 & 1 & 0 & 1 \\ x_9 & 1 & 0 & 0 & 0 & 1 & 0 \\ x_{10} & 1 & 1 & 0 & 0 & 1 & 0 \\ x_{11} & 0 & 1 & 0 & 0 & 1 & 0 \\ x_{12} & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Figure 2.1: Two examples of the set covering problem.

$$\boxed{\begin{array}{llrcll} (IP) & \text{minimize} & \sum_{j=1}^{n} s_j & & & \\ & \text{subject to} & As & \geq & \mathbf{1}^m & \\ & & s_j & \in & \{0,1\}, & (j = 1, 2, \ldots, n). \end{array}}$$

For the minimum-cost set covering problem the objective function becomes

$$\sum_{j=1}^{n} c_j s_j.$$

Proposition 2.1 *The set covering problem is NP-hard.*

A rough translation of the proposition is that if there exists a polynomial-time algorithm for solving the set covering problem, then there exist polynomial-time algorithms for solving each of the following "hard problems" as well as many other hard problems:

1. integer linear programming,

2. finding Steiner minimal trees,

3. finding a cut that has the maximum number of edges, and

4. the traveling salesman problem.

In other words, showing that a problem is NP-hard gives evidence that it has no polynomial-time algorithm, but no lower bound on the running time required for solving the problem is obtained. The formal proof of the above proposition starts with a problem already known to be NP-hard and shows that each instance of the problem can be transformed in polynomial running time to an instance of the set covering problem such that an optimal solution to the latter instance can be used to find an optimal solution to the former instance. In other words, to prove that the set covering problem is NP-hard, we show that a problem already known to be NP-hard is *polynomially transformable* to the set covering problem. Consequently, if a polynomial-time algorithm were available for solving set covering problems, then it could be used to solve the original NP-hard problem in polynomial time. For many NP-hard problems, the best algorithms known so far are rather inefficient, and amount to searching through all possible solutions of the input instance. Hence, it is commonly assumed that no efficient (i.e., polynomial-time) algorithm will ever be found for an NP-hard problem.

2.2 Applications

The set covering problem has many applications. Three are listed here.

(a) Information retrieval:
There are n files $S_1, \ldots, S_n$, where S_j has length c_j, and there are m requests for information. Each unit of information is stored in at least one file. Find a subset of the files of minimum total length such that searching these will retrieve all the requested information.

(b) Airline crew scheduling:
An airline has m flights $x_1, \ldots, x_m$. These flights can be combined into "flight legs" $S_1, \ldots, S_n$ such that the same crew can service all the flights in leg S_j. Find the minimum number of crews required to service all flights. Note that the number of flight legs may be much larger than the number of crews.

EXAMPLE

	Leg 1	Leg 2	Leg 3	Leg 4
Toronto Paris	1	0	0	0
Toronto New York	0	1	0	0
Toronto Chicago	0	0	1	1
Chicago Vancouver	0	0	1	0
Chicago New York	0	0	0	1
New York Miami	0	1	0	1

(c) Minimum path separator:
An undirected graph $G = (V, E)$ is given, together with m special paths $P_1, P_2, \ldots, P_m$. Here, m may be much greater than $|V|$ or $|E|$. For each edge e_j of G, we denote the set of paths

$$
\begin{array}{ll}
P_1 = v_2v_3v_4 & S_1 = \{P_2\} \\
P_2 = v_2v_1v_4 & S_2 = \{P_5\} \\
P_3 = v_2v_5v_4 & S_3 = \{P_2\} \\
P_4 = v_3v_4v_5 & S_4 = \{P_5\} \\
P_5 = v_3v_1v_5 & S_5 = \{P_1, P_6\} \\
P_6 = v_3v_2v_5 & S_6 = \{P_1, P_4\} \\
 & S_7 = \{P_3, P_4\} \\
 & S_8 = \{P_3, P_6\}
\end{array}
$$

Figure 2.2: Modeling the minimum path separator problem as a set covering problem.

P_i, $1 \leq i \leq m$, such that e_j is in P_i by S_j. The problem is to find a minimum cardinality set of edges whose deletion from G disconnects each of the paths $P_1, P_2, \ldots, P_m$.

2.3 The matrix reduction algorithm for set covering

Since no polynomial-time algorithm is available for finding the optimal solution of set covering problems, it is worthwhile to study heuristics for solving the problem with the goal of obtaining a performance guarantee (or approximation guarantee) on the heuristic. That is, given a simple method that finds feasible but not necessarily optimal solutions, the goal is to find whether the solution returned by the method is within a fixed factor of optimal, for every instance of the problem.

The matrix reduction algorithm is a heuristic for the set covering problem. Given an instance of the problem in the form of the incidence matrix, the algorithm "reduces" the matrix by repeatedly attempting to eliminate rows and columns and in the process the algorithm constructs a $J \subseteq \{1, \ldots, n\}$ such that if $J' \subseteq \{1, \ldots, n\}$ is an optimal solution of the reduced instance, then $J \cup J'$ is an optimal solution of the original instance. The algorithm may find the optimal solution on some instances, since the reduced instance may be trivial and so an optimal solution J' for it is easily found. The algorithm may fail completely on other instances, since it may not succeed in eliminating even one row or column from the original incidence matrix. Thus there is *no* performance guarantee for the matrix reduction algorithm.

Algorithm Matrix Reduction Algorithm for Set Covering
input: Incidence matrix A (with one row for each point x_i in X and one column for each set S_j in $\mathcal{F}$).
output: Partial solution J and "reduced" incidence matrix A' such that for any optimal solution J' of A', $J' \cup J$ is an optimal solution of A.

STEP 0: (feasibility check)
    if A has a row with all entries zero
    then stop, no feasible solution exists;
    end;

    $J := \emptyset$;

repeat
    STEP 1:
        if row i has exactly one nonzero entry, say, in column j
        then $J := J \cup \{j\}$;   (S_j must be chosen in the optimal solution)
            eliminate column j, and all rows having a "one" in that column;
        end;
    STEP 2:
        if rows i "dominates" row i', i.e., for all columns j, $a_{ij} \geq a_{i'j}$
        then eliminate row i;   (every solution that covers i' will cover i too)
        end;
    STEP 3:
        if column j is "dominated by" column j', i.e., for all rows i, $a_{ij} \leq a_{ij'}$
        then eliminate column j;   (j' can replace j in every solution)
        end;
until (matrix A is empty) **or** (no rows nor columns are eliminated in the last iteration);

output J and the current matrix A' obtained by reducing A;

2.4 The greedy algorithm for set covering

The greedy algorithm is a simple method that has a performance guarantee of roughly $\ln(|X|)$. This section describes the greedy algorithm, and the performance guarantee is proved in the next section.

$$A = \begin{array}{c|ccccc|} & S_1 & S_2 & S_3 & S_4 & S_5 \\ \hline x_1 & 1 & 0 & 1 & 1 & 0 \\ x_2 & 1 & 0 & 0 & 1 & 0 \\ x_3 & 1 & 1 & 0 & 1 & 0 \\ x_4 & 0 & 1 & 1 & 1 & 0 \\ x_5 & 1 & 0 & 0 & 0 & 1 \\ x_6 & 0 & 0 & 1 & 0 & 0 \\ x_7 & 0 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

Initially, $J = \emptyset$.
Step 1 adds 3 to J, and eliminates column S_3 and rows x_1, x_4 and x_6,
Step 2 eliminates row x_3 since it dominates row x_2, and
Step 3 eliminates columns S_2 and S_5 since they are dominated by columns S_4 and S_1, respectively.
Step 1 adds 1 to J, and eliminates column S_1 and rows x_2 and x_5, and then it adds 4 to J, and eliminates column S_4 and row x_7.
The reduced matrix is empty, so the algorithm stops and outputs an optimal solution $J = \{1, 3, 4\}$, i.e., S_1, S_3, S_4 form a minimum set covering.

Figure 2.3: The working of the matrix reduction algorithm is shown on an example (from Chapter 6, "Urban Operations Research", by Larson and Odoni).

Algorithm Greedy Algorithm for Set Covering
input: Family $\mathcal{F} = S_1, S_2, \ldots, S_n$ of subsets of a finite set X.
output: $J \subseteq \{1, 2, \ldots, n\}$ such that $\bigcup_{j \in J} S_j = X$.

$U := X$; (U is the set of points that are currently uncovered)
$J := \emptyset$;

while $U \neq \emptyset$ **do**
    choose a subset S_{j^*} in $\mathcal{F}$ such that $|S_{j^*} \cap U|$ is maximum among all S_j in $\mathcal{F}$;
    $U := U \backslash S_{j^*}$;
    $J := J \cup \{j^*\}$;
end (while);

output J;

2.5 A performance guarantee for the greedy algorithm

For a nonnegative integer d, let the dth harmonic number $\sum_{i=1}^{d} \frac{1}{i}$ be denoted by $H(d)$ (take $H(0) = 0$).

Fact 2.2 *For d a positive integer, $\ln(d+1) \leq H(d) \leq 1 + \ln(d)$.*

For sufficiently large integers d, $H(d)$ can be approximated by $\ln(d) = \log_e(d)$, where e denotes the base of the natural logarithm.

EXAMPLE

d	$H(d)$	$\ln(d)$	$\ln(d+1)$
5	$1 + \cdots + \frac{1}{5} = \frac{137}{60} \approx 2.28333$	$1.60943\ldots$	$1.79175\ldots$
10	$1 + \cdots + \frac{1}{10} = \frac{7381}{2520} \approx 2.92897$	$2.30258\ldots$	$2.39789\ldots$
25	$3.81595\ldots$	$3.21887\ldots$	$3.25809\ldots$
100	$5.18737\ldots$	$4.60517\ldots$	$4.61512\ldots$
1000	$7.48547\ldots$	$6.90775\ldots$	$6.90875\ldots$
10000	$9.78760\ldots$	$9.21034\ldots$	$9.21044\ldots$

Theorem 2.3 *Consider an instance of the set covering problem given by $\mathcal{F} = \{S_1, \ldots, S_n\}$, where each $S_j \subseteq X$, and $X = \bigcup_{j=1}^{n} S_j$. Let J_{opt} denote an optimal solution, and let J_{greedy} denote a solution found by the greedy algorithm. Then*

$$|J_{greedy}| \leq H(\max_{j=1,\ldots,n} |S_j|) \cdot |J_{opt}|.$$

Proof: Let r denote $\max_{j=1,\ldots,n}(|S_j|)$.

Let A be the $m \times n$ incidence matrix. Consider the linear programming (LP) relaxation (P) of the integer program (IP) for set covering, and the dual LP (D) of (P). Let y denote the row vector $[y_1, \ldots, y_m]$ of the dual variables.

(P) minimize $\sum_{j=1}^{n} s_j$ subject to $As \geq \mathbf{1}^m$, $s_j \geq 0 \quad (j = 1, \ldots, n)$	(D) maximize $\sum_{i=1}^{m} y_i$ subject to $yA \leq \mathbf{1}^n$, $y_i \geq 0 \quad (i = 1, \ldots, m)$

The set covering problem has a feasible solution (since $X = \bigcup_{j=1}^{n} S_j$), hence, both (IP) and (P) have feasible solutions. Let z_{IP} and z_{LP} denote the optimal values of (IP) and (P), respectively; so $z_{IP} = |J_{opt}|$.

The key point of the proof is this:

> there is a way to construct a feasible solution $y = [y_1, \ldots, y_m]$ of the dual LP (D) such that the corresponding objective value $\sum_{i=1}^{m} y_i$ is $\geq |J_{greedy}|/H(r)$.

The theorem immediately follows because

$$|J_{opt}| = z_{IP} \geq z_{LP} \geq \sum_{i=1}^{m} y_i = |J_{greedy}|/H(r).$$

This sequence of inequalities is worth a detailed study. The first inequality ($z_{IP} \geq z_{LP}$) follows because (P) is a relaxation of (IP). In other words, every feasible solution of (IP), including the optimal (IP) solution, is a feasible solution of (P), hence, the optimal solution of (P) has objective value at most z_{IP}. The second inequality ($z_{LP} \geq \sum_{i=1}^{m} y_i$) is a direct consequence of linear programming weak duality applied to (P) and (D). The third inequality is guaranteed by the construction of y.

Here is the construction for y. For every iteration of the greedy algorithm, let U denote the set of uncovered points at the start of that iteration. Consider an arbitrary iteration, and let the set chosen in that iteration be $\hat{S}$. We say that each point in $\hat{S} \cap U$ is *newly covered* by $\hat{S}$. In other words, a point x is said to be newly covered by the set S if S is the first set in the sequence of sets chosen by the greedy algorithm such that $x \in S$. For each point $x \in \hat{S} \cap U$, define $wgt(x) = 1/|\hat{S} \cap U|$. Observe that for each point $x \in X$, the quantity $wgt(x)$ is well defined, because for each point $x \in X$, there exists a set S in our family $\mathcal{F}$ such that x is newly covered by S. To complete the construction, for $1 \leq i \leq n$, the dual variable y_i of the point x_i is assigned the value $wgt(x_i)/H(r)$. The next three claims show that $y = [y_1, \ldots, y_m]$ satisfies all the constraints of the dual LP (D) and that $\sum_{i=1}^{m} y_i \geq |J_{greedy}|/H(r)$. The first claim is the crucial step in the proof.

Claim 1: Consider an arbitrary set $S \in \mathcal{F}$. Let q denote $|S|$. Order the points in S in the reverse order in which they are covered by the greedy algorithm, and let this ordering be $x'_1, x'_2, \ldots, x'_q$. Then

$$wgt(x'_\ell) \leq 1/\ell, \quad \forall \ell = 1, 2, \ldots, q.$$

To elaborate on the ordering $x'_1, x'_2, \ldots, x'_q$, focus on the last iteration of the greedy algorithm such that $S \cap U$ is nonempty; then we order the points in $S \cap U$ arbitrarily, and take them to be $x'_1, x'_2, \ldots, x'_{|S\cap U|}$. (Similarly, in an arbitrary iteration of the greedy algorithm such that the chosen set is, say, S_j (where $1 \leq j \leq n$) we take the points in $S \cap U \cap S_j$ to be $x'_{|S\cap U|}, x'_{|S\cap U|-1}, \ldots, x'_{|S\cap U|-|S\cap U \cap S_j|+1}$.) Now, consider an arbitrary point $x'_\ell \in S$, $1 \leq \ell \leq q$. Let x'_ℓ be newly covered by the set $\hat{S}$. At the start of the iteration that chooses $\hat{S}$, all the points $x'_1, x'_2, \ldots, x'_\ell$ are uncovered, i.e., $\{x'_1, x'_2, \ldots, x'_\ell\} \subseteq S \cap U$. Hence, $|S \cap U| \geq \ell$. By the definition of the greedy strategy, the set $\hat{S}$ chosen in this iteration must have

$$|\hat{S} \cap U| \geq |S \cap U| \geq \ell.$$

Since the number of newly covered points in $\hat{S}$ is $\geq \ell$, $wgt(x'_\ell) = 1/|\hat{S} \cap U| \leq 1/\ell$.

Claim 2: For every set $S \in \mathcal{F}$, $\sum\limits_{x \in S} wgt(x) \leq H(r)$.

Let $q = |S|$, and let $x'_1, x'_2, \ldots, x'_q$ be the ordering of the points in S defined in Claim 1. Then

$$\sum_{x\in S} wgt(x) = wgt(x'_1) + wgt(x'_2) + \ldots + wgt(x'_i) + \ldots + wgt(x'_q) \leq 1 + \frac{1}{2} + \ldots + \frac{1}{i} + \ldots + \frac{1}{q} = H(q) \leq H(r),$$

where the first inequality follows directly from Claim 1, and the last inequality follows since $q = |S| \leq \max_{j=1,\ldots,n}(|S_j|) = r$.

Claim 3: $\sum_{i=1}^{m} wgt(x_i) = |J_{greedy}|$.

To see this, focus on a set $\hat{S}$ chosen by the greedy algorithm and the corresponding set U as defined above. Then $\sum_{x \in \hat{S} \cap U} wgt(x) = |\hat{S} \cap U| \cdot wgt(x \mid x \in \hat{S} \cap U) = |\hat{S} \cap U| / |\hat{S} \cap U| = 1$. In other words, for each set $\hat{S}$ chosen by the greedy algorithm, the newly covered points in $\hat{S}$ contribute a total weight of one. Summing over all sets chosen by the greedy algorithm gives Claim 3.

Consider $y = [y_1, \ldots, y_m]$, where $y_i = wgt(x_i)/H(r), \forall i = 1, \ldots, m$. Clearly, y is ≥ 0, and by Claim 2, for each set $S \in \mathcal{F}$, we have $\sum_{x_i \in S} y_i = \frac{1}{H(r)} \sum_{x_i \in S} wgt(x_i) \leq \frac{1}{H(r)} \cdot H(r) = 1$. This shows that y is a feasible solution of (D). Moreover, Claim 3 shows that $\sum_{i=1}^{m} y_i = \frac{1}{H(r)} \sum_{i=1}^{m} wgt(x_i) = \frac{|J_{greedy}|}{H(r)}$. This completes the construction of y, and the proof of the theorem. ▌

2.6 A direct analysis

In this section, we look at an alternate analysis of the greedy heuristic for set cover that does not use linear programming duality. Recall that the greedy algorithm chooses sets in iterations, where in every iteration, the set chosen in one newly covering the maximum number of points.

The key point of the alternate proof is this:

> When set S_{j^*} is chosen by the greedy algorithm, $|S_{j^*} \cap U| \geq \frac{|U|}{|J_{opt}|}$.

Note that U represents the current set of uncovered points in the above statement. This observation is a direct consequence of the greedy choice, and the definition of the optimal solution: consider the optimal cover J_{opt} which covers all the points in X, and hence also the points in U. By averaging among the sets in J_{opt}, the one which covers the maximum number of points in U must cover at least $\frac{|U|}{|J_{opt}|}$. Since the greedy algorithm chooses among all the sets the one with the maximum new coverage, this coverage must be at least as much as claimed.

Let the indices of the sets picked by the greedy algorithm in the order they were picked be denoted by $j(1), \ldots, j(g)$. Furthermore, for $t = 1, \ldots, g$, let U_t denote the set U just before the set $J_{j(t)}$ was picked. Thus, for example, $U_1 = X$. Define $U_{g+1} = \emptyset$. We can write a simple recurrence for $|U_t|$.

$$|U_{t+1}| = |U_t| - |S_{j(t)} \cap U_t|$$

By the observation above, we have $|S_{j(t)} \cap U_t| \geq \frac{|U_t|}{|J_{opt}|}$. Substituting and simplifying, we get

$$|U_{t+1}| \leq |U_t|(1 - \frac{1}{|J_{opt}|})$$

Expanding out the recurrence, we get

$$|U_t| \le |X|(1 - \frac{1}{|J_{opt}|})^t$$

Taking natural logarithm and simplifying, we finally get

$$t \le |J_{opt}| \ln \frac{|X|}{U_t} \tag{2.1}$$

Since $U_g \neq \emptyset$, this immediately implies that $|J_{greedy}| = g \le |J_{opt}| \ln |X|$. Note that the performance ratio is slightly weaker, namely, $\ln |X|$ versus the earlier derived $\ln \max_{j=1,\ldots,n}(|S_j|)$. Despite this shortcoming, there are a couple of corollaries of this analysis that are simpler to derive than from the earlier proof.

Corollary 2.4 *Suppose we run the greedy algorithm only until a constant fraction, say $0 < \alpha < 1$ of the points are covered. Then the number of sets used is at most $1 + |J_{opt}| \ln \frac{1}{\alpha}$.*

To prove the corollary, we use the first term of one to account for the last iteration before stopping. At the iteration just preceding this, we have $|U| > \alpha \cdot |X|$ by the stopping rule, and the claim follows from equation 2.1.

Corollary 2.5 *Suppose at every step of the greedy algorithm, we are only able to choose a near-optimal greedy set, i.e., we choose a set S_{j^*} whose new coverage is at least β times the maximum among that of all S_j in $\mathcal{F}$ for some $0 < \beta \le 1$. Then the size of the cover output is at most $\beta |J_{opt}| \ln |X|$.*

This corollary is a direct consequence of substituting the weaker guarantee into the analysis and simplification of the recurrence relation for $|U|$.

2.7 A randomized algorithm

The greedy algorithm may be viewed as a derandomization of a simple randomized algorithm for choosing a set cover. The starting point of this algorithm is a solution to the linear program (P) for set cover. This solution assigns values s_j between zero and one to the sets, that sum to z_{LP}. We can interpret these values as probabilities and choose independently, every set S_j with probability s_j. By linearity of expectation, the expected number of sets chosen is z_{LP}.

Next we bound the probability that any point $x \in X$ is not covered by the sets chosen in one round of this randomized selection of sets. The point x is not covered when none of the sets containing it is chosen. The probability of this event is $\prod_{j:S_j \ni x}(1 - s_j) \le e^{-\sum_{j:S_j \ni x} s_j} \le \frac{1}{e}$.

Suppose we repeat the randomoized selection for $\gamma \ln |X|$ rounds. Then the probability of not covering any given point $x \in X$ drops to at most $(\frac{1}{e})^{\gamma \ln |X|} \le |X|^{-\gamma}$. By the union bound, the probability that some point in X is not covered is at most $\sum_{x \in X} |X|^{-\gamma} \le |X|^{1-\gamma}$. By choosing a sufficiently large constant γ, we can drive down this probability to any polynomially small factor.

An alternate oblivious rounding algorithm and its derandomization leading to the greedy algorithm is discussed by Young in [8]. The analysis of the greedy heuristic for set covering is due to

Johnson [3], Lóvasz [5], and Chvátal [1]. Our analysis using duality follows the treatment of Chvátal closely. The direct analysis to the best of our knowledge appeared first in a paper of Leighton and Rao [4] on approximating graph separators. A tight analysis of the greedy heuristic, with more on the second order terms is due to Slavik [7].

What will not discuss but is nevertheless another exciting aspect of research on the set covering problem is its non-approximability. The first result in this area due to Lund and Yannakakis [6] showed that there is no polynomial time approximation algorithm with performance ration better than $\frac{\ln |X|}{8}$ unless $NP \subseteq DTIME[n^{\text{polylog } n}]$, an unlikely complexity theoretic assumption akin to $NP \subseteq P$. After a series of strengthenings, the best known result [2] essentially matches the performance of the greedy algorithm and shows that there is no polynomial time approximation algorithm with performance ration better than $(1 - o(1)) \ln |X|$ unless $NP \subseteq DTIME[n^{\log \log n}]$.

2.8 Exercises

1. Given an undirected graph $G = (V, E)$ with a subset $T \subseteq V$ of nodes specified as *terminals*, and nonnegative costs $c : V \rightarrow \Re_+$ on the nodes, the Node Steiner Tree problem is to find a tree of minimum total cost containing all the terminals. Transform set covering to this problem to show that it is NP-hard.

   (Extra credit) Derive an $O(\log |V|)$ approximation algorithm for the Node Steiner Tree problem. Hint: Formulate this problem as a set cover problem in such a way that it would be possible to implement a single step of the greedy algorithm in polynomial time.

   Is there an approximation algorithm for this problem with performance ratio $o(\log |V|)$?

2. Construct an example of the set covering problem (one for each $|X|$) such that

   (P1) $|J_{greedy}| \geq \alpha \cdot \log_2 |X| \cdot |J_{opt}|$, where α is a constant,

   (P2) no two rows of the incidence matrix A are identical and

   (P3) in the set family $\mathcal{F}$, no set S_j is a subset of another set S_k.

3. Repeat the above replacing (P1) with $|J_{opt}| \geq \alpha \cdot \log_2 |X| \cdot z_{LP}$ where α is a constant.

4. The fixed-charge median problem is defined as follows: we are given a complete undirected graph $G = (V, E)$, with nonnegative fixed location costs $f : V \rightarrow \Re_+$ on the nodes and nonnegative distances $d : E \rightarrow \Re_+$ on the edges. The objective is to locate median nodes at a subset $M \subseteq V$ of the nodes. Every non-median node is defined to be serviced by the median node closest to it under the distances d. The total cost of this median placement is the sum of the fixed location costs of all the median nodes, and the sum over non-median nodes, of the distance of the node to its closest median. Formally, the cost of a placement of medians in M is $c(M) = \sum_{v \in M} f_v + \sum_{v \in V-M} \min_{u \in M} d_{uv}$. The goal is to find a placement M with minimum total cost.

   Derive an $O(\log |V|)$ approximation algorithm for this problem.

5. (Extra credit) In showing a 2-approximation algorithm for node-multiway cuts, we showed half-integrality of optimal solutions for a certain linear relaxation of the problem. The dual to this relaxation is a linear program whose objective is to maximize the total concurrent flow between terminals specified in the problem. Prove or disprove: for any specification of integral capacities, there is a maximum flow solution that is also half-integral.

6. Transform set covering to an instance of the Directed Steiner Tree problem: given a digraph $G = (V, A)$, with a root node r and a subset $T \subseteq V$ of *terminals*, and nonnegative costs $c : A \rightarrow \Re_+$ on the arcs, find an outward-directed spanning tree of minimum total cost with r as the root and containing all the nodes in T.

Bibliography

[1] V. Chvátal, *A greedy heuristic for the set-covering problem*, Math. of OR **4:3** (1979), pp. 233-235.

[2] U. Feige, *A threshold of ln n for approximating set cover*, Proc. 28th Annual Symp. on Theory of Computing (1996), pp. 314-318.

[3] D. S. Johnson, *Approximation algorithms for Combinatorial Problems*, J. Computer System. Sci. **9** (1974), pp. 256-278.

[4] F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms*," Proc. of the 29th Annual IEEE Conference on Foundations of Computer Science (1988), pp. 422-431.

[5] L. Lovàsz, *On the ratio of optimal integral and fractional covers*, Disc. Math. **13** (1975), pp. 383-390.

[6] C . Lund and M. Yannakakis, *On the Hardness of Approximating Minimization Problems*, Proc., 25th Annual ACM Symp. on Theory of Computing (1993), pp. 286-293.

[7] P. Slavik, *A tight analysis of the greedy algorithm for set cover*, Proc. 28th Annual Symp. on Theory of Computing (1996), pp. 435-441.

[8] N. Young, *Randomized rounding without solving the linear program*, Proc. of the 6th Annual ACM-SIAM Symp. on Disc. Algo. (1995), pp. 170-178.

Chapter 3

Center Problems and Median Problems

This chapter focuses on discrete optimization problems within a spatial context. Two types of objective functions are of interest: (i) minimizing the maximum cost of serving one of several clients, the so-called center problem, and (ii) minimizing the sum of the costs of serving several clients, the so-called median problem. The first few sections discuss well-known results and algorithms from the area. This part is based on Christofides [2, Chapters 5 & 6], Larson & Odoni [6, Chapter 6.5], and on two surveys from Mirchandani & Francis [5], namely, [5, Chapter 1] by J. Krarup and P. M. Pruzan, and [5, Chapter 2] by P. B. Mirchandani. We do not give original references for this part, and instead refer the reader to the above texts. The last section discusses a seminal paper of Lin & Vitter [3] that gives a bicriteria approximation algorithm for the k-median problem. Lin & Vitter introduce their filtering-and-rounding method. This method first solves an LP relaxation, then uses the LP solution to construct a filtered problem by fixing some variables to zero in the integer programming formulation. A near-optimal solution to the filtered problem gives a solution to the original problem such that the objective value is nearly optimal but some of the original constraints may be violated. Lin & Vitter's analysis shows that the objective value is within a guaranteed factor of optimal, and also the "constraint violation ratio" is within a guaranteed factor. Lin & Vitter's work is based on earlier results due to Raghavan and Thompson [8], and Raghavan [7].

3.1 Introduction

In this chapter, we would like to find optimal (or approximately optimal) locations for routine services, such as airports, factories, warehouses, schools, garbage incinerators, prisons, etc., as well as for emergency services such as ambulances, fire stations and snow plows. Locational decisions are based on many factors; some of these factors are physical, economic, social, environmental or political. Our focus will be on the quantitative analysis of discrete problems. That is, the locations must be chosen from a finite number of potential sites selected in a preliminary stage.

We focus on three types of problems:

1. Center problems (also called minimax problems):

A given number of facilities (or services) must be located so as to minimize the maximum travel cost (or maximum travel time) of the users to/from the facilities. These problems usually arise in applications involving the location of emergency services.

2. Median problems:
   A given number of facilities (or services) must be located so as to minimize the average travel cost (or average travel time) of the users to/from the facilities. These problems usually come from applications involving the location of routine, nonemergency services.

3. Uncapacitated facility location problems:
   Given a number of potential sites, fixed costs for opening a facility at each site, and profits made by serving certain clients from certain sites, the problem is to choose sites where facilities should be located and to assign clients to facilities such that the net earnings are maximized.

The uncapacitated facility location problem is studied in the next chapter.

3.2 The basic model for median problems and center problems

A weighted undirected graph (or network) $G = (V, E)$ is used to model the application. Demands for service originate at the nodes of G. Each node $v \in V$ has a real-valued demand $h(v)$; this indicates the rate (or intensity) at which demands for service originate from node v. Each edge $vw \in E$ has a real-valued cost $c(vw)$; this represents the cost of providing a service (or sending a commodity) from node v to node w (or from node w to node v). The goal is to locate one or more facilities to serve the node demands such that a specified objective function is maximized or minimized. A facility can be located at any *point* of the network G, where a point x is either a node or a location on an edge vw, and in the latter case the point is specified by giving $d(x, v)$ (or $d(x, w)$) such that $d(x, v) + d(x, w) = c(vw)$ (i.e., we add a "new node" x and "new edges" xv, xw with costs $d(xv), d(xw)$ such that $d(xv) + d(xw) = c(vw)$). For a point x and a node v, $d(x, v)$ denotes the minimum cost of a path between x and v. In other words, if x is a node, then $d(x, v)$ is the minimum over all paths $v_0v_1v_2\cdots v_{\ell-1}v_\ell$ such that $v_0 = x$ and $v_\ell = v$ of $\sum_{i=0}^{\ell-1} c(v_i, v_{i+1})$, otherwise, if x is a point on an edge $v'w'$ then $d(x, v)$ is the minimum over all paths $v_0v_1v_2\cdots v_{\ell-1}v_\ell$ such that $v_0 = x$, $v_1 = v'$ or $v_1 = w'$, and $v_\ell = v$ of $d(x, v_1) + \sum_{i=1}^{\ell-1} c(v_i, v_{i+1})$. For a set of points $x = \{x_1, \cdots, x_k\}$ and a node $v \in V$, $d(X, v)$ denotes $\min_{x_i \in X} d(x_i, v)$, that is, $d(X, v)$ is the minimum cost of a path between any one of the points x_i in X and v.

3.3 Center problems (minimax problems)

The problem is to locate k facilities on the points of a given network G so as to minimize the maximum cost of a path between a demand node and its "closest" facility. The main applications are to the location of emergency services such as fire stations, ambulances, etc.

For a point x of the network G let $m(x)$ denote

$$\max_{v \in V} d(x, v),$$

the cost of a "shortest" path between x and a "farthest" demand node. Recall that a point x is either a node of G or a location on an edge of G.

The *vertex center* (or node center) is defined to be a node $x^* \in V$ such that for every node $y \in V$,

$$m(x^*) \leq m(y).$$

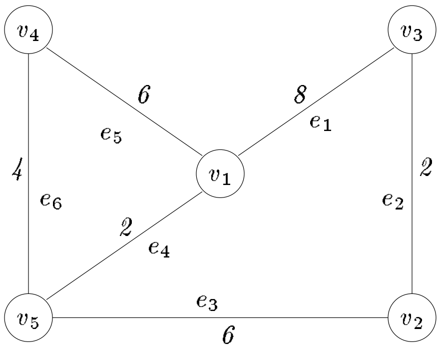
The *local center* of an edge pq is a point x_ℓ on pq such that for every point y on pq,

$$m(x_\ell) \leq m(y).$$

The *absolute center* of the network G is a point x^* in G such that for every point y in G (y may be on an edge of G)

$$m(x^*) \leq m(y).$$

To find a single node center, we compute the matrix of shortest paths costs for all pairs of nodes, and then choose a node such that the maximum entry in its row in the matrix is smallest among the maximum entries of all rows. See Figure 3.1 for an example.



$d(v, w)$

	v_1	v_2	v_3	v_4	v_5
v_1	0	8	8	6	2
v_2	8	0	2	10	6
v_3	8	2	0	12	8
v_4	6	10	12	0	4
v_5	2	6	8	4	0

Figure 3.1: Finding a single node center on an example (from Chapter 5, "Graph Theory: An Algorithmic Approach", by Christofides). The matrix d of shortest paths costs is shown. Either v_1 or v_5 may be taken as the node center.

Finding a single absolute center is more involved. We start with a simple method, then develop a more efficient method. Figure 3.2 shows the working on an example.

Algorithm Simple Algorithm for Absolute Center
input: Graph $G = (V, E)$, nonnegative edge costs c.
output: Absolute center of G, c.

for each edge pq of G, find the local center x_{pq} of the edge;

choose the local center x^* (among those found in the previous step) that minimizes $m(x^*)$;
output x^*, since it is an absolute center of G;

The absolute center of G may not coincide with the node center of G. Moreover, the absolute center of G may be located on an edge that is *not* incident to the node center of G.

To develop a more efficient method for finding absolute centers we state two results.

Proposition 3.1 *For the set of all points x on a fixed edge of G, the maximum distance function $m(x)$ is piecewise linear and its slope is always $+1$ or -1.*

Proposition 3.2 *For an edge pq, the local center x_ℓ satisfies*

$$m(x_\ell) \geq \frac{m(p) + m(q) - c(pq)}{2} \ ,$$

where $c(pq)$ denotes the cost of edge pq.

Proof: Consider any point on the edge pq. Let x, $0 \leq x \leq c(pq)$, specify the point, where we take the point with $x = 0$ to be p and the point with $x = c(pq)$ to be q. We take $d(x, p)$ to be x and $d(x, q)$ to be $c(p, q) - x$. The cost of a shortest path between x and a farthest demand node, $m(x)$, is piecewise linear with a slope of $+1$ or -1, its value at $x = 0$ is $m(p)$, and its value at $x = c(pq)$ is $m(q)$. Hence,

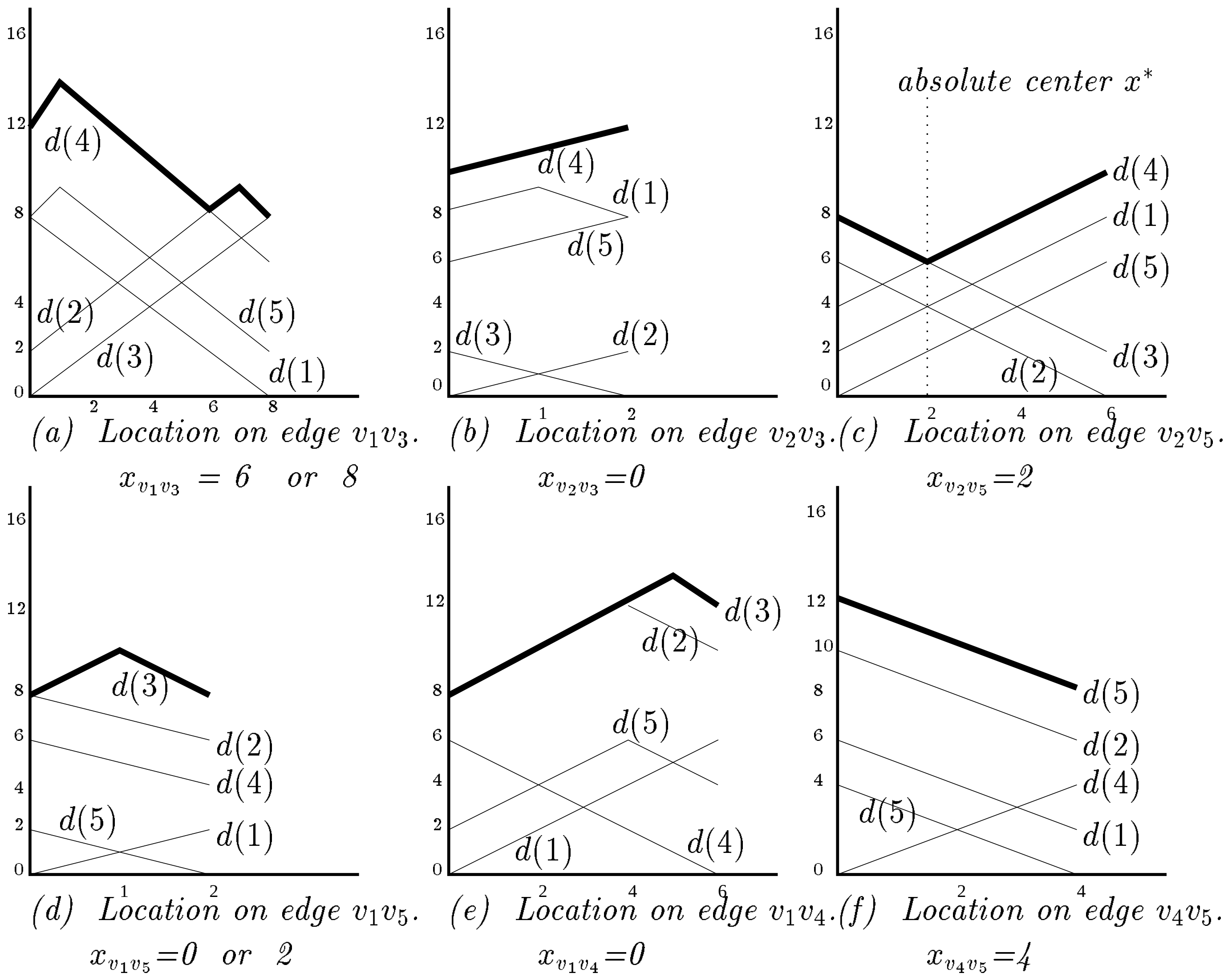
$$\begin{aligned} m(x) &\geq m(p) - x \qquad (\forall x:\ 0 \leq x \leq c(pq)), \ \text{and} \\ m(x) &\geq m(q) - (c(pq) - x) \qquad (\forall x:\ 0 \leq x \leq c(pq)). \end{aligned}$$

Adding the two inequalities above and then dividing by 2, we obtain the result. ∎

Algorithm Improved Algorithm for Absolute Center
input: Graph $G = (V, E)$, nonnegative edge costs c.
output: Absolute center of G, c.

find a node center $x \in V$, and let $t = m(x)$;
for each edge pq in sequence **do**
    if $(m(p) + m(q) - c(pq))/2 < t$
    then find the local center x_{pq} of pq and let $t = \min\{t, m(x_{pq})\}$;
    end (if);
end (for);
output x^* and t, where x^* is the last point that assigned a value to t, and $t = m(x^*)$;

At termination of the algorithm, the absolute center is the point x^* (node center x or local center x_{pq}) that assigned the last value to t.



(a) Location on edge v_1v_3. $x_{v_1v_3} = 6$ *or* 8

(b) Location on edge v_2v_3. $x_{v_2v_3}=0$

(c) Location on edge v_2v_5. $x_{v_2v_5}=2$

(d) Location on edge v_1v_5. $x_{v_1v_5}=0$ *or* 2

(e) Location on edge v_1v_4. $x_{v_1v_4}=0$

(f) Location on edge v_4v_5. $x_{v_4v_5}=4$

Figure 3.2: Finding a single absolute center for the example in Figure 3.1 (from Chapter 5, "Graph Theory: An Algorithmic Approach", by Christofides). For each edge pq, the local center is found by plotting $d(x, v_i)$ for each node $v_i \in V$, where $0 \leq x \leq c(pq)$ – in the figure, $d(x, v_i)$ is abbreviated to $d(i)$. The "upper envelope" of the five $d(x, v_i)$'s is indicated by thick lines: for each point x in pq, the upper envelope gives the cost of a shortest path from x to a farthest node. The local center x_{pq} is the point that minimizes the upper envelope. The absolute center is the point x^* on the edge v_5v_2 with $d(x^*, v_5) = 2$ and $d(x^*, v_2) = 4$; $m(x^*) = 6$.

3.3.1 Multiple centers

For a set $X = \{x_1, \cdots, x_k\}$ of points of G, let $m(x)$ denote $\max_{v \in V} d(X, v)$, where $d(X, v)$ as before denotes the minimum cost of a path between v and any one of the points x_i in X. A set X^* of points (nodes) of G with $|X| = k$ is called a set of absolute (node) k-centers if $m(X^*) \leq m(X)$ for every set X of k points (nodes) of G.

Proposition 3.3 *The problem of finding absolute (or, node) k-centers is NP-hard.*

See "p-Center Problems" by G.Y. Handler in "Discrete Location Theory", Ed., Mirchandani and Francis, and Christofides and Viola (1971) for approximation algorithms for multiple center problems.

3.4 Median problems (minisum problems)

Consider the single facility location problem of Figure 3.3. The network represents an urban area; the nodes represent locations where demands for services are generated, and the edges represent the roads between these locations. A single facility is to be located in the area (possibly on one of the roads), and the users need to travel to the facility. The daily demand for the service at each node v is shown next to the node; this number is denoted $h(v)$. The length of each road vw is also given; this number is the cost of vw, $c(vw)$. The problem is to choose a location for the facility such that the average travel distance to it is minimized. The key result in this section, Hakimi's theorem (Theorem 3.4), assures us that rather than considering the infinite number of points in the network, we need to focus only on the $|V|$ nodes: one of the nodes is an optimal location.

For a set of points X (the points in X may be nodes or locations on edges), let $J(X)$ denote $\sum_{v \in V} h(v)d(X, v)$. In other words, $J(X)$ is the total cost of serving all the node demands, assuming that a facility is located at each point $x_i \in X$ and that each node $v \in V$ is served from the "closest" point $x_i \in X$, i.e., all the demand of v is served by a point $x_i \in X$ with $d(x_i, v) = d(X, v)$.

A *median* of network G is defined to be a point $x^* \in G$ such that for every point $x \in G$,

$$J(x^*) \leq J(x),$$

i.e., a median is a point that minimizes the total cost of serving all node demands.

A *k-median* is defined to be a set X_k^* of k points $x_1^*, x_2^*, \cdots, x_k^* \in G$ such that for every set X_k of k points,

$$J(X_k^*) \leq J(X_k).$$

If k facilities are to be located at the k points in X_k, then finding the k-median X_k^* amounts to finding the set of k locations that minimizes the total cost of serving all node demands. If each $h(v)$ is fixed to be the fraction of all calls for service that originate from node v, then $J(X_k)$ gives the average cost of providing the service to all users, and finding the k-median X_k^* amounts to finding the set of k locations that minimizes the average cost of serving all node demands. The proof of Hakimi's theorem for a single median follows; the general case is left as an exercise for the reader.

Theorem 3.4 (Hakimi's theorem) *At least one set of k-medians exists solely on the nodes of G.*

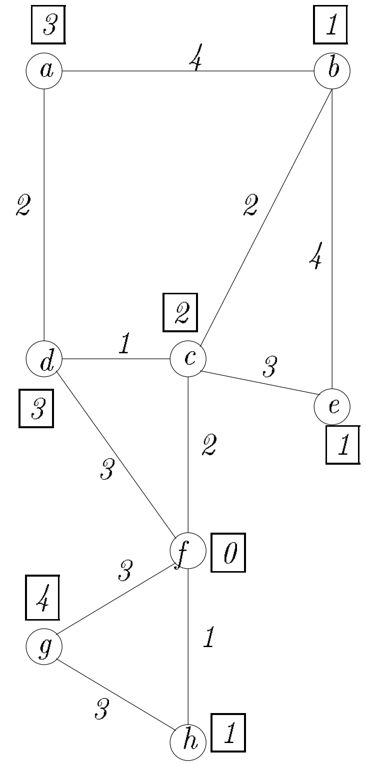


Figure 3.3: An example of a single median problem (from Chapter 6, "Urban Operations Research", by Larson and Odoni). Node c is a median of the network.

Proof: (for a single median) Let x^* be a median of G. If x^* is a node, then the proof is done, so suppose that x^* lies strictly inside an edge uw. Partition the node set V into sets U and W (so $U \cap W = \emptyset$, and $U \cup W = V$), where node v is in U if and only if there is a minimum-cost path between v and x^* that contains u. Without loss of generality, suppose that $\sum\limits_{v \in U} h(v) \geq \sum\limits_{v \in W} h(v)$. Now,

$$\begin{aligned}
J(x^*) &= \sum_{v \in V} h(v) d(v, x^*) \\
&= \sum_{v \in U} h(v)[d(v,u) + d(u,x^*)] \quad + \quad \sum_{v \in W} h(v) d(v, x^*) \\
&= \sum_{v \in U} h(v) d(v,u) \quad + \quad \sum_{v \in U} h(v) d(u, x^*) \quad + \quad \sum_{v \in W} h(v) d(v, x^*) \\
&\geq \sum_{v \in U} h(v) d(v,u) \quad + \quad \sum_{v \in W} h(v)[d(u,x^*) + d(v,x^*)] \\
&\quad \left(\text{because } \sum_{v \in U} h(v) \geq \sum_{v \in W} h(v) \right) \\
&\geq \sum_{v \in U} h(v) d(v,u) \quad + \quad \sum_{v \in W} h(v) d(v,u) \\
&\quad (\text{ using the triangle inequality: } d(v,x^*) + d(x^*,u) \geq d(v,u)) \\
&= \sum_{v \in V} h(v) d(v,u) \\
&= J(u).
\end{aligned}$$

This completes the proof: the 1-median can always be moved from a point strictly inside an edge to an end node of the edge without increasing the objective value. ∎

3.4.1 A single-median algorithm

To find a single median in a network, which consists of a graph $G = (V, E)$ and edge costs $c : E \rightarrow \Re_+$, we compute the matrix d of shortest paths costs between all pairs of nodes. This computation is done either using inspection or one of the all-pairs shortest-paths algorithms such as the Floyd-Warshall algorithm or $|V|$ applications of Dijkstra's algorithm. Next, we compute the terms $h(v)d_{wv}$ by multiplying each column of the matrix d by the demand of node v. Each of these terms gives the cost of satisfying the demands originating at node v assuming that the facility is located at node w. The optimum location for the facility can now be found by summing across the entries for each row w of the $[h(v)d_{wv}]$ matrix; this gives the total cost of serving all the node demands if the facility is located at node w. Normalizing the node demands $h(v)$ by dividing by the total demand $\sum\limits_{v \in V} h(v)$, we can find the average cost associated with each of the $|V|$ candidate locations. See the top part of Figure 3.4 for an example.

Algorithm Single-Median Algorithm

input: Graph $G = (V, E)$, nonnegative edge costs c, and node demands $h : V \to \Re$.
output: A median node.

obtain the matrix d of shortest paths costs;
multiply the vth column of d by the node demand $h(v)$ to obtain the matrix $[h(v)d_{wv}]$;
for each row w of the matrix $[h(v)d_{wv}]$, compute the sum of all terms in the row;
take the median x^* to be a node whose row sum is minimum;
output x^*;

3.4.2 A multimedian heuristic

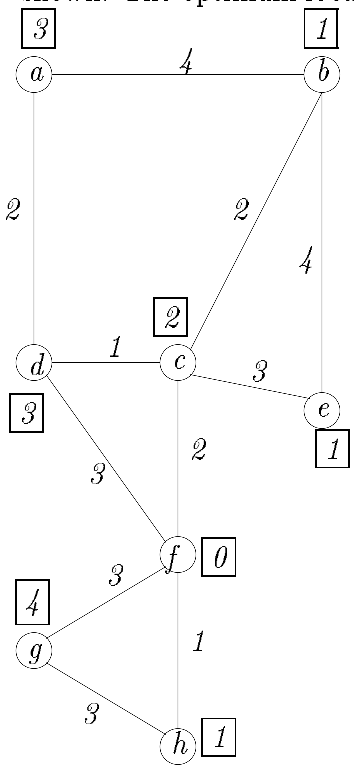
For small k (or small $|V| - k$), the k-median can be found by a straightforward extension of the single-median algorithm. By Hakimi's theorem, we only need to consider sets of points consisting of k nodes. With a total of $n = |V|$ nodes, the number of sets of k nodes to be examined is $\binom{n}{k} \approx \frac{n^n}{(n-k)^{n-k}k^k}$. If both k and $n-k$ are moderately large, then the number of sets to be examined becomes prohibitively large. For small k, total or average costs for each of the $\binom{n}{k}$ sets of locations can be obtained from the $[h(v)d_{wv}]$ matrix: all the demand from each node v is "assigned" to the facility "closest" to it, i.e., the facility w that minimizes d_{wv} among the k facilities.

The multimedian heuristic algorithm below finds a set S of k nodes that is not guaranteed to be optimal but is locally optimal in the sense that for any set X of k nodes if X and S have $k-1$ nodes in common, then $J(S) \leq J(X)$, i.e., S has as good an objective value as any set X obtained by replacing one node of S. The algorithm begins by finding a single median and then increases the number of selected points in steps of one at a time until this becomes equal to the required number, k. By Hakimi's theorem, only the nodes need to be considered for inclusion in the set S. See Figure 3.4 for a worked example.

Algorithm Multimedian Heuristic Algorithm
input: Graph $G = (V, E)$, nonnegative edge costs c, node demands $h : V \to \Re$, and
a number $k \geq 1$.
output: "Locally optimal" set S with $|S| = k$; S may not be a k-median,
but for any set X with $|X| = k$ and $|S \cap X| = k-1$, $J(S) \leq J(X)$.

find a single median of G, and suppose that it is node x;
$S := \{x\}$;
while $|S| < k$ **do**
(facility addition step)
find a node $y \in V \backslash S$ that maximizes $J(S) - J(S \cup \{y\})$;
(adding y to S gives the maximum improvement in the objective value)
$S := S \cup \{y\}$;

Figure 3.4: Finding a single median and an (approximate) 2-median on the example in Figure 3.3 (from Chapter 6, "Urban Operations Research", by Larson and Odoni). The matrix $[h(v)\ d_{wv}]$ is shown. The optimum location for the facility (single median) is at node c.



$[h(v)\ d_{wv}]$

a	b	c	d	e	f	g	h		row sums
0	4	6	6	6	0	32	6	a	60
12	0	4	9	4	0	28	5	b	62
9	2	0	3	3	0	20	3	c	40
6	3	2	0	4	0	24	4	d	43
18	4	6	12	0	0	32	6	e	78
15	4	4	9	5	0	12	1	f	50
24	7	10	18	8	0	0	3	g	70
18	5	6	12	6	0	12	0	h	59

single median: node c, with $J(\{c\}) = 40$.

2-median heuristic:

facility addition: find $\min_{y \in V} J(\{y, c\})$.

$y \rightarrow$	a	b	c	d	e	f	g	h
J({y,c})	31	38	–	34	37	30	20	29

let $S = \{g, c\}$, with $J(S) = 20$.

solution improvement:

try swapping c with each of $a,\ b,\ d,\ e,\ f,\ h$ in turn:

$S = \{d, g\}$ with $J(S) = 18$ gives improvement;

try swapping d with each of $a,\ b,\ c,\ e,\ f,\ h$ in turn: no improvement;

try swapping g with each of $a,\ b,\ c,\ e,\ f,\ h$ in turn: no improvement;

stop and **output** $S = \{d, g\}$ with $J(S) = 18$.

```
          (solution improvement step)
SWAP:     for each x ∈ S do
              for each y ∈ V\S do
                  if J({y} ∪ (S\{x})) < J(S)
                  then S := (S\{x}) ∪ {y};
                       go to SWAP;
                  end (if);
              end (for);
          end (for);
end (while);
output S;
```

3.5 Bicriteria Approximations for the k-median Problem

In this section, we sketch a bicriteria approximation algorithm for an important facility location problem, the k-median problem. The performance guarantees are nearly best-possible.

3.5.1 Introduction

The k-median problem is an important problem that arises in a variety of contexts from facility location, clustering and data compression: In the most general version of the problem, we are given a complete directed graph with nonnegative costs c_{ij} in its arcs (ij). The objective is to choose k vertices as medians so that the sum of the costs from every vertex to its closest median is minimized. The *symmetric* version of the problem is the undirected counterpart of the problem obeying $c_{ij} = c_{ji}$. Another special case arises when the (symmetric or asymmetric) costs obey the triangle inequality.

The k-median problem can be viewed as the bicriteria problem of minimizing simultaneously the number of medians used and the sum of costs to the nearest medians. Call the latter quantity "median cost." In our terminology, this is the problem (Number of medians, Median cost, Empty subgraph) or simply (Number of medians, Median cost). Note that following our convention, we have imposed a budget k on the first objective in the above formulation.

We shall study a rounding method due to Lin and Vitter [3] that works on the linear programming relaxation of a natural IP formulation for the problem. First, we review hardness results on approximating this problem.

3.5.2 Hardness of approximation

The k-median problem is very closely related to the dominating set problem in an undirected graph. In the latter problem, we are given an undirected graph $G = (V, E)$ and a positive integer k, and the task is to determine if there exist a dominating set of G of size at most k. In other words, does there exist k nodes in G such that every vertex is either one among these k or is adjacent to one among these k vertices.

Even when the cost matrix for the k-median problem is symmetric and obeys the triangle inequality (i.e., metric), it is hard to obtain a $(o(\log n), 1)$ approximation in an n-node problem.

Theorem 3.5 *Let $f(n) = o(\log n)$, and suppose there exists a $(f(n), 1)$ approximation algorithm for the k-median problem on an n-node graph with metric costs (i.e., if V_k is an optimal median set, the approximation algorithm outputs a set U of medians with $|U| \leq f(n)k$ and $\sum_{i \in V} \min_{j \in U}\{c_{ij}\} \leq \sum_{i \in V} \min_{j \in V_k}\{c_{ij}\}$). Then both the dominating set and set covering problems can be approximated within a factor of $O(f(n))$.*

Proof: We use a reduction from the dominating set problem. Given an unweighted graph G, we use its edges to define a metric on its nodes, namely the shortest path metric using edges in G. Thus c_{ij} is the number of edges in a shortest path between i and j in G. Suppose a minimum dominating set in G has size k. Then there is a solution to the corresponding k-median problem with median cost at most $n - k$.

Conversely, suppose the approximation algorithm for the median problem outputs a median set U with at most $f(n)k$ vertices and median cost at most $n-k$. We bound the number of vertices non-dominated by U, i.e., not in or adjacent to U. We claim there can be at most $f(n)k$ such vertices. For otherwise, the median cost of the solution U would be at least $2f(n)k + (n - f(n)k - f(n)k)$. The first term is a lower bound on the median cost of all the nondominated vertices, while the second term is a lower bound on the cost of all the remaining vertices other than those in U, and those that are nondominated. The resulting median cost is greater than $n - k$, a contradiction. Now, it is easy to form a dominating set out of U: simply include all the nondominated vertices in the set to obtain a dominating set of size at most $2f(n)k$. This implies the theorem. ▌

For the same problem, without the triangle inequality, it is easy to show that there is no $(1, \rho)$ approximation for any ρ.

Theorem 3.6 *Let $\rho > 1$ and suppose there exists a $(1, \rho)$ approximation algorithm for the k-median problem on an n-node graph with symmetric costs. Then there is a polynomial time algorithm for the dominating set problem.*

Proof: We use a similar reduction from the dominating set problem as before. Given an unweighted graph G, we now define c_{ij} to be one if the edge ij is in G, and to be $\rho(n-k)+1$ if ij is not an edge. Suppose a minimum dominating set in G has size k. Then there is a solution to the corresponding k-median problem with median cost at most $n - k$.

Conversely, a $(1, \rho)$ approximation algorithm will return a median set of size at most k such that its median cost is at most $\rho(n - k)$. But by our cost function, every node in the graph must be adjacent to this median set since the cost of every non-edge is more than $\rho(n - k)$. Thus the returned median set is dominating, giving us an algorithm for the dominating set problem. ▌

In fact the proof of the above theorem is approximation preserving and gives the following stronger theorem.

Theorem 3.7 *Let $\rho > 1$ and suppose there exists a $(f(n), \rho)$ approximation algorithm for the k-median problem on an n-node graph with symmetric costs. Then there is a $f(n)$-approximation algorithm for the dominating set problem.*

The fact below is a consequence of a straightforward approximation-preserving reduction (do it!) of the set covering problem to a dominating set problem in an undirected graph.

Fact 3.8 *A $f(n)$-approximation algorithm for the dominating set problem implies an $O(f(n))$-approximation for the unweighted set cover problem.*

By the known hardness results on the set covering problem [3], we can infer that $f(n) = \Omega(\ln n)$.

3.5.3 Rounding by filtering

In this section, we present a $((1+\frac{1}{\epsilon})H(n), 1+\epsilon)$-approximation for the general k-median problem for any $\epsilon > 0$. To do this, we first formulate the problem as an integer program.

Formulation

The choice variables in our IP formulation correspond to deciding whether a median is located at any given node. Let y_j denote this choice, i.e., $y_j = 1$ exactly when vertex j is chosen as a median node. To formulate the objective function, it is convenient to have a variable that denotes for every vertex i, the closest median node to this vertex that it is assigned to. Let x_{ij} denote the assignment of node i to median j. thus $x_{ij} = 1$ exactly when $y_j = 1$ (node j is a median) and vertex i is assigned to j, where node j is the closest median node to i. Now we are ready to write down the integer programming formulation.

$$
\boxed{
\begin{array}{llrcll}
(IP) & \text{minimize} & \displaystyle\sum_{i\in V}\sum_{j\in V} c_{ij}x_{ij} & & & \\
& \text{subject to} & \displaystyle\sum_{j\in V} x_{ij} & \geq & 1 & \forall\, i \in V \\
& & \displaystyle\sum_{j\in V} y_j & \leq & k & \\
& & x_{ij} & \leq & y_j & \forall\, i,j \in V \\
& & x_{ij}, y_j & \in & \{0,1\} & \forall\, i,j \in V
\end{array}
}
$$

The LP relaxation of the above formulation relaxes the last set of integrality constraints to allow the variables x_{ij} and y_j to take rational values between 0 and 1.

A solution to the linear programming relaxation is completely characterized by a fractional assignment $\hat{y_j}$ to the variables y_j. The remaining fractional variables x_{ij} can be set "optimally" as follows [1].

We assign each vertex to all its nearest fractional medians whose $\hat{y_j}$-values sum to at least 1. Formally, sort the values c_{ij} for $j \in V$ so that $c_{ij_1(i)} \leq c_{ij_2(i)} \leq ...c_{ij_n(i)}$. Let p be the index such that $\sum_{l=1}^{p-1} \hat{y_l} < 1 \leq \sum_{l=1}^{p} \hat{y_l}$. Then set $\hat{x_{ij}} = \hat{y_j}$ for $j = j_1(i), \ldots, j_{p-1}(i)$, and $x_{i\hat{j_p(i)}} = 1 - \sum_{l=1}^{p-1} \hat{y_l}$. For all other j, set $\hat{x_{ij}} = 0$.

Filtering

The key idea in rounding the LP solution $\hat{y_j}$ is to filter out (set to zero) some of the integral assignments x_{ij} by using information from $\hat{y_j}$. In this case, this corresponds to allowing every vertex i to be assigned only to the remaining (non-filtered) centers. We must do the filtering so that there is still a good median solution where every vertex i can be assigned to some non-filtered vertex that is chosen in the solution. In our case, we can view this assignment as a fractional set covering problem and hence derive a good filtering rule.

Denote by $\hat{C}_i$ the cost contribution of vertex i to the fractional median cost, i.e. $\hat{C}_i = \sum_{j \in V} c_{ij} \hat{x_{ij}}$. For a given $\epsilon > 0$, define the neighborhood of i to be

$$S_i = \{j \in V | c_{ij} \leq (1+\epsilon)\hat{C}_i\}.$$

The filtering now disallows the choices x_{ij} for those j that are not in the neighborhood S_i. Despite this, if we regard y_j as fractional choices for the medians, by our filtering rule, we ensure that every vertex i still has enough fractional medians that it can be assigned to.

Lemma 3.9 *For every $i \in V$ and $\epsilon > 0$,*

$$\sum_{j \in S_i} \hat{y}_j > \frac{\epsilon}{1+\epsilon}.$$

Proof: We prove by contradiction. Suppose $\sum_{j \in S_i} \hat{y}_j \leq \frac{\epsilon}{1+\epsilon}$. Then

$$\begin{aligned}
\hat{C}_i &= \sum_{j \in V} c_{ij} \hat{x_{ij}} \\
&\geq \sum_{j \notin S_i} c_{ij} \hat{x_{ij}} \\
&> (1+\epsilon)\hat{C}_i \sum_{j \notin S_i} \hat{x_{ij}} \\
&\geq (1+\epsilon)\hat{C}_i (1 - \frac{\epsilon}{1+\epsilon}) \\
&= \hat{C}_i
\end{aligned}$$

The contradiction proves the lemma. ∎

Transformation to set covering

The above lemma allows us to look upon the problem of assigning medians to cover the neighborhoods S_i of all the vertices in the graph as a set covering problem. We must now choose a small number of medians to cover all the sets S_i. Note that this restriction immediately implies a bound on the total median cost of our choice since every vertex i is assigned to some median within distance at most $(1+\epsilon)\hat{C}_i = (1+\epsilon)\sum j \in V\hat{x_{ij}}$.

Lemma 3.10 *Suppose every vertex i is assigned to a median within its neighborhood S_i. Then the median cost of the resulting solution is at most $(1+\epsilon)\sum_{i \in V}\sum_{j \in V} c_{ij}\hat{x_{ij}}$, which in turn is at most $(1+\epsilon)$ times the the optimal k-median cost.*

It remains to solve the resulting set cover problem to find an integral solution with a minimum number of medians. To this end, we can use the several algorithms we studied earlier for rounding a fractional solution to the set cover problem. However, we still do not have a fractional feasible solution for the set cover problem of assigning medians to cover all the neighborhoods S_i. However, Lemma 3.9 is useful in producing such a solution.

Set $\bar{y_j} = (1+\frac{1}{\epsilon})\hat{y_j}$ for every vertex $j \in V$. By lemma 3.9, for any vertex i, we have

$$\sum_{j\in S_i} \bar{y_j} \geq \sum_{j\in S_i}(1+\frac{1}{\epsilon})\hat{y_j} > 1.$$

In other words, $\bar{y_j}$ s a fractional solution to the set covering problem and the total fractional value of this solution is

$$\sum_{j\in V} \bar{y_j} \leq \sum_{j\in V}(1+\frac{1}{\epsilon})\hat{y_j} \leq (1+\frac{1}{\epsilon})k.$$

Using any of numerous approximation algorithms for the set covering problems we saw earlier, we can find an integral solution of size at most the fractional cover times $H(n)$ where n is the size of the largest set and $H(n) = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \leq \ln n + 1$, the n^{th} harmonic number. In our case n is at most the number of nodes in the graph since every set is a neighborhood of nodes. Thus the value of the approximate set cover we find is at most $(1+\frac{1}{\epsilon})H(n)k$. Thus, we have proved the following theorem.

Theorem 3.11 *For any $\epsilon > 0$, there is a $((1+\frac{1}{\epsilon})H(n), 1+\epsilon)$-approximation for the general k-median problem on an n-node graph.*

3.6 Exercises

1. Find (i) the node center, and (ii) the absolute center of the network G, c in Figure 3.5 using any method you like. First, give a precise explanation of your method.

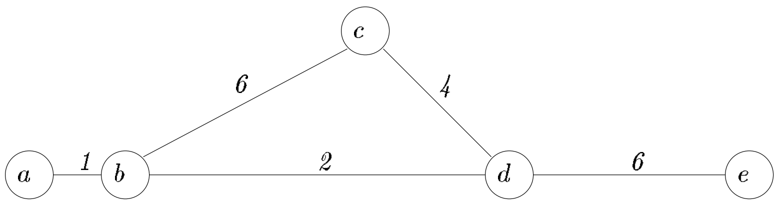


Figure 3.5: Network for exercise

2. Construct a network G, c such that the absolute center is located on an edge that is *not* incident to the node center.

3. The goal is to show that the following algorithm finds a single absolute center of a tree.

Algorithm Single Absolute Center of a Tree
input: Tree $G = (V, E)$, and nonnegative edge costs c.
output: Absolute center x^*.

choose an arbitrary point x in G;
find an (end) node v_s that is farthest from x, i.e., $d(x, v_s) = \max_{v \in V} d(x, v)$;
find an (end) node v_t that is farthest from v_s, i.e., $d(v_s, v_t) = \max_{v \in V} d(v_s, v)$;
let x^* be the midpoint of the path from v_s to v_t;
output x^*; (x^* is the absolute center of G, c)

Show that for all points $x \in G$ we must have

$$m(x) = m(x^*) + d(x, x^*).$$

Hence, argue that

$$m(v_s) = 2\, m(x^*)$$

and, therefore, that x^* must lie on the path associated with $m(v_s)$ and must be at the halfway point between v_s and v_t.

4. Explain whether or not the following method finds the absolute center of a graph $G = (V, E)$ with respect to edge costs $c : E \to \Re_+$:
Find a pair of nodes x, y such that the cost of a shortest path from x to y, $d(x, y)$, is **maximum** among all pairs of nodes. Take the absolute center x^* to be a point at the middle of a shortest path from x to y.

5. Either prove the following statement or give a counterexample: For every network G, c (G is any graph, c is any nonnegative cost function on its edges) and a single absolute center x^* of G, c there exists a pair of nodes v_s and v_t such that x^* is the absolute center of a shortest path from v_s to v_t (i.e., x^* is at the halfway point between v_s and v_t).

6. (More improved algorithm for absolute center; from Chapter 6, "Urban Operations Research", by Larson and Odoni.) The goal is to obtain an algorithm for the absolute center that is better than the improved algorithm (page 28), by using a better lower bound $L_2(pq)$ given below for the value $m(x_\ell)$ of the local center x_ℓ of an edge pq. For the edge pq, let $v_p \in V$ and $v_q \in V$ denote the farthest nodes from p and q, respectively, i.e., $m(p) = d(p, v_p)$ and $m(q) = d(q, v_q)$. Clearly, the local center x_ℓ of pq satisfies $m(x_\ell) \geq \max(d(x_\ell, v_p),\ d(x_\ell, v_q))$. Prove:

   (a) For a point x in the interior of edge pq (i.e., $x \in pq$, $x \neq p$ and $x \neq q$), $\max(d(x, v_p),\ d(x, v_q))$ may attain at most a single local minimum.

   (b) Such a local minimum, if it exists, is attained at the point $x \in pq$ whose distance from p along edge pq is

   $$\frac{c(pq) + d(q, v_p) - d(p, v_q)}{2},$$

   and at this point $\max(d(x, v_p),\ d(x, v_q))$ attains the value

   $$L_2(pq) = \frac{d(p, v_q) + d(q, v_p) + c(pq)}{2}.$$

(c) $L_2(p,q) \geq \dfrac{m(p)+m(q)-c(pq)}{2}$, i.e., $L_2(p,q)$ provides a better lower bound on $m(x_\ell)$ than the one used in the algorithm on page 28.

7. (Location of a "supporting facility"; from Chapter 6, "Urban Operations Research", by Larson and Odoni.) Consider the network of Figure 3.6 and suppose that the nodes v represent five cities, the numbers next to the nodes give the demands $h(v)$ of the cities, and the numbers next to the links vw give the mile length c_{vw} of roadways connecting the cities. Cars travel on the roadways at an effective speed of 30 m.p.h. Assume that a major facility, say an airport, has been located at some point on this network. A regional planning group now wishes to install a high-speed transportation link to the airport with a single station. The high-speed vehicles will travel on the network at twice the speed of cars and their route will be the shortest route to the airport. It is assumed that travelers to the airport will choose the combination of transportation modes which minimizes their access time to the airport (ignoring transfer times). To clarify the description above, assume that the airport is at node b and that the single station of the high-speed link is located at node e. Then, the access time to the airport of travelers from node e is 25 minutes. However, the access time of travelers from node a is still 40 minutes. (It would take travelers from node a exactly 20 minutes to get to node e by car and then another 25 minutes to get from node e to node b, so that it is better to go directly to the airport by car.)

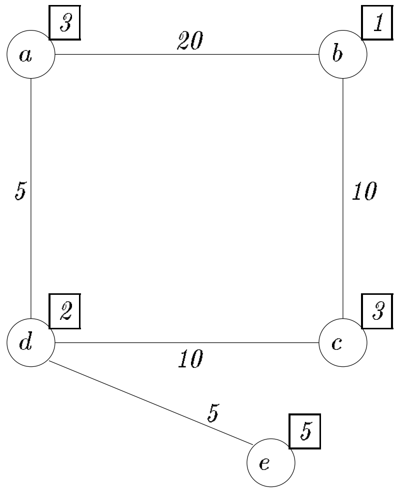


Figure 3.6: Network for exercise

(a) Show that no matter where the airport is located, an "optimal location" for the station of the high-speed vehicles must be on a node of the graph, where an optimal location is one that minimizes the total weighted travel distance to the airport for travelers from the five cities. Note that the airport is not restricted to be at one of the nodes of the network. Alternatively, show this for a general network rather than for this specific case only.

(b) Assuming that the airport is located at node b, where should the station be located? Devise an algorithm for solving this type of problem.

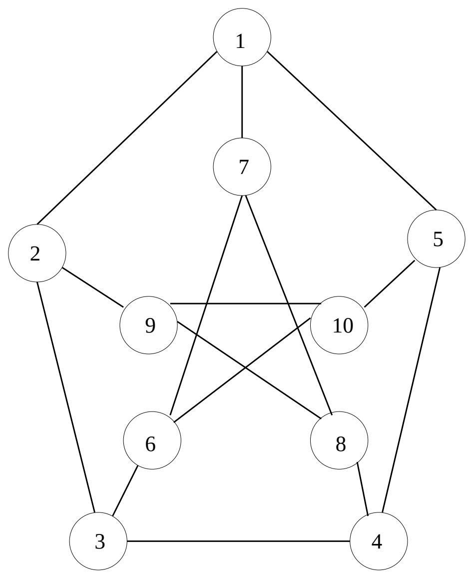
(c) Assume now that travel time, in minutes, by car between any two points x and y on the network is given by $f(d_{xy}) = \left(\frac{d_{xy}}{5}\right)^2$, where d_{xy} is the minimum cost of a path between the two points, and that travel time through the high-speed link between the same two points is given by $g(d_{xy}) = \frac{1}{2}\left(\frac{d_{xy}}{5}\right)^2$. The optimal location may not be on a node. How would you answer part (b) now?

(d) Show that the result you proved in part (a) holds as long as the functions $f(d_{xy})$ and $g(d_{xy})$ are both concave in d_{xy} (Mirchandani 1979).

8. Prove Hakimi's k-median theorem, Theorem 3.4.

(Hint: Suppose X_k is a k-median, and $x \in X_k$ is not a node. Focus on the nodes $v \in V$ such that $d(x, v) < d(X_k \backslash \{x\}, v)$.)

9. Consider the k-median problem on the graph $G = (V, E)$ in Figure 3.7. This is the Petersen graph. Note that there are 10 nodes, 15 edges, and every node $v \in V$ has degree equal to 3. Take the cost of each edge to be 1. Take the node set V to be $\{1, 2, \ldots, 10\}$. The matrix of shortest paths costs $[c_{vw}]$ is given in Figure 3.7.



$$[c_{vw}] = \begin{bmatrix} 0 & 1 & 2 & 2 & 1 & 2 & 1 & 2 & 2 & 2 \\ 1 & 0 & 1 & 2 & 2 & 2 & 2 & 2 & 1 & 2 \\ 2 & 1 & 0 & 1 & 2 & 1 & 2 & 2 & 2 & 2 \\ 2 & 2 & 1 & 0 & 1 & 2 & 2 & 1 & 2 & 2 \\ 1 & 2 & 2 & 1 & 0 & 2 & 2 & 2 & 2 & 1 \\ 2 & 2 & 1 & 2 & 2 & 0 & 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 1 & 0 & 1 & 2 & 2 \\ 2 & 2 & 2 & 1 & 2 & 2 & 1 & 0 & 1 & 2 \\ 2 & 1 & 2 & 2 & 2 & 2 & 2 & 1 & 0 & 1 \\ 2 & 2 & 2 & 2 & 1 & 1 & 2 & 2 & 1 & 0 \end{bmatrix}$$

Figure 3.7: The Petersen graph $G = (V, E)$ and cost matrix $[c_{vw}]$ for Exercise 9

(a) Prove or disprove:
The objective value of the 2-median problem is at least 11, i.e., the optimal value of

(IP) is at least 11, where (IP) is the integer programming formulation of the 2-median problem. Recall that (IP) has a variable y_j for each node $j \in V$, and has a variable x_{ij} for each ordered node pair $i, j \in V \times V$.

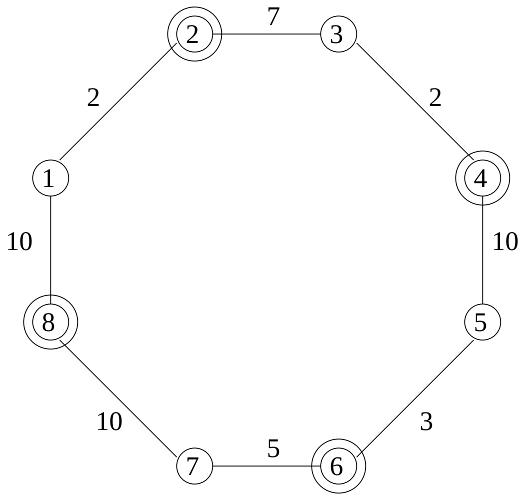
(b) Consider the LP relaxation (P) of (IP). Let $\widehat{y}$ be given by

$$\widehat{y}_j = 1/5, \ \forall j \in V.$$

Compute the optimal $\widehat{x}$ with respect to $\widehat{y}$, and find the objective value of (P) corresponding to the feasible solution $\widehat{x}, \widehat{y}$.

10. The aim is to apply Lin & Vitter's Filtering & Rounding method to an example of the k-median problem, for $k = 1$. Recall the integer linear programming formulation (IP) of the k-median problem, and the LP relaxation (P) of (IP).

Consider the k-median problem in Figure 3.8 below. Note that $k = 1$. The "cost matrix" $[c_{vw}]$ for all node pairs $v, w \in V \times V$ is given in Figure 3.8.



$$[c_{vw}] = \begin{bmatrix} 0 & 2 & 9 & 11 & 21 & 24 & 20 & 10 \\ 2 & 0 & 7 & 9 & 19 & 22 & 22 & 12 \\ 9 & 7 & 0 & 2 & 12 & 15 & 20 & 19 \\ 11 & 9 & 2 & 0 & 10 & 13 & 18 & 21 \\ 21 & 19 & 12 & 10 & 0 & 3 & 8 & 18 \\ 24 & 22 & 15 & 13 & 3 & 0 & 5 & 15 \\ 20 & 22 & 20 & 18 & 8 & 5 & 0 & 10 \\ 10 & 12 & 19 & 21 & 18 & 15 & 10 & 0 \end{bmatrix}$$

Figure 3.8: Graph $G = (V, E)$ and cost matrix $[c_{vw}]$ for Exercise 10.

Let $\widehat{y}$ give a feasible solution to (P), where $\widehat{y}$ is:

$$\widehat{y}_2 = 1/4, \quad \widehat{y}_4 = 1/4, \quad \widehat{y}_6 = 1/4, \quad \widehat{y}_8 = 1/4, \qquad \text{and} \qquad \widehat{y}_j = 0, \ \forall j \in \{1, 3, 5, 7\}.$$

- Find $\widehat{x}$ that minimizes the objective function with respect to $\widehat{y}$.
- Take $\epsilon = 1/2$ and apply the Filtering & Rounding method as follows.
- For each node i, compute the cost $\widehat{C}_i = \sum_{j \in V} c_{ij} \widehat{x}_{ij}$ and compute the neighborhood V_i.
- Write down the set covering problem for the filtered problem, by making use of the neighborhoods V_i.
- Solve the set covering problem, using <u>either</u> inspection <u>or</u> the greedy heuristic.

11. Recall the bicriteria k-median problem (number of medians, median cost). We discussed a $((1+\frac{1}{\epsilon})H(n), 1+\epsilon)$-approximation for any $\epsilon > 0$ in an n-node graph. The goal here is to give an $(O(1), O(1))$-approximation algorithm for the special case of metric costs. In detail:
Suppose that the cost matrix $[c_{vw}]$ $(\forall v, w \in V \times V)$ is symmetric and satisfies the triangle inequality. Let $\epsilon > 0$ be the parameter. Then modify the filtering and rounding method to find a set of columns J^* such that

$$|J^*| \leq (1 + \frac{1}{\epsilon})k,$$

and

$$\sum_{i \in V} \min_{j \in J^*} c_{ij} \leq 3(1+\epsilon)z_{LP},$$

where z_{LP} is the optimal value of the LP relaxation. (In other words, the modified method may increase the number of medians by a factor of $(1+\frac{1}{\epsilon})$ and must achieve an objective value within a factor of $3(1+\epsilon)$ of z_{LP}.)

Bibliography

[1] S. Ahn, C. Cooper, G. Cornuejols, and A. Frieze, *Probabilistic analysis of a relaxation for the k-median problem*, Math. of Oper. Res. **13**, pp. 1–31 (1988).

[2] N. Christofides, *Graph Theory: An Algorithmic Approach.* Academic Press, London, 1975.

[3] U. Feige, *A threshold of ln n for approximating set cover*, Proc. 28th Annual ACM Symp. on Theory of Computing (1996), pp. 314–318.

[4] D. S. Hochbaum and D. B. Shmoys, *A unified approach to approximation algorithms for bottleneck problems*, J. Assoc. Comput. Mach. **33** (1986), pp. 533–550.

[5] R. C. Larson and A. R. Odoni, *Urban Operations Research.* Prentice-Hall, Englewood Cliffs, N. J., 1981.

[6] J.-H. Lin and J. S. Vitter, *ϵ-approximations with minimum packing constraint violation*, Proc. 24th Annual ACM Symp. on Theory of Computing (1992), pp. 771–782.

[7] J.-H. Lin and J. S. Vitter, *Approximation algorithms for geometric location problems*, Inf. Proc. Lett. **44** (1992), pp. 245–249.

[8] P. B. Mirchandani and R. L. Francis, *Discrete Location Theory.* John Wiley & Sons, New York, 1990.

[9] P. Raghavan, *Probabilistic construction of deterministic algorithms: approximating packing integer programs*, J. Comp. Sys. Sci. **37** (1988), pp. 130–143.

[10] P. Raghavan and C. D. Thompson, *Randomized rounding: a technique for provably good algorithms and algorithmic proofs*, Combinatorica **7** (1987), pp. 365–374.

Chapter 4

The Uncapacitated Facility Location Problem

This chapter is based on two sources, namely, the survey by Cornuejols, Nemhauser and Wolsey [5, Chapter 3], and a recent paper by Aardal, Shmoys and Tardos [9]. The book by Nemhauser and Wolsey [6] has a detailed discussion on the topic of this chapter.

The results of Aardal et al are based on earlier results of Lin & Vitter [3, 4], and Raghavan & Thompson [8, 7].

4.1 The problem

Given a set of potential sites, a set of clients, and relevant profit and cost data, the goal is to find a maximum-profit plan giving the number of facilities to open, their locations and an allocation of each client to an open facility.

In detail, the input to an *uncapacitated facility location* (UFL) problem consists of

- a set $J = \{1, \ldots, n\}$ of potential sites for locating facilities,
- a set $I = \{1, \ldots, m\}$ of clients whose demands need to be served by the facilities,
- a profit c_{ij} for each $i \in I$ and $j \in J$; this is the profit made by satisfying the demand of client i from a facility located at site j, and
- a fixed nonnegative cost f_j for each $j \in J$; this is the (one time) cost of opening a facility at site j.

The problem is to select a subset Q ($Q \subseteq J$) of sites, to open facilities at these sites, and to assign each client to exactly one facility such that the difference of the variable profits and the fixed costs is maximized. The number of facilities to be opened, $|Q|$, is not prespecified, rather it is determined by an optimal solution. The profits c_{ij} usually depend on several factors such as the per unit production cost of a facility at site j, the per unit transportation cost from j to i, and the selling price to client i. An example problem and its optimal solution are shown in Figure 4.1.

$$
\begin{aligned}
m &= |I| = 3, \qquad n = |J| = 3 \\
f_j &= 1 \quad \text{for } (j = 1, \ldots, n) \\
C &= \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}
\end{aligned}
$$

Optimal solution: $S = \{1, 2\}$, i.e., open facilities at sites 1 and 2. Assign client 1 to site 2, client 2 to site 1, and client 3 to site 1 to obtain an optimal value of 1.

Figure 4.1: An example of the uncapacitated facility location problem together with an optimal solution.

Proposition 4.1 *The UFL problem is NP-hard.*

Proof: Given an instance of the set covering problem, we can construct an instance of the UFL problem such that an optimal solution to the UFL problem gives an optimal solution to the set covering problem. First, we construct a bipartite graph based on the set covering instance: for each point x_i there is a node x_i in the "left side" of the bipartite graph, and for each set S_j, $S_j \subseteq \{x_1, x_2, \ldots, x_m\}$, there is a node S_j in the "right side" of the bipartite graph. There is an edge x_iS_j if and only if point x_i belongs to set S_j. The instance of the UFL problem is as follows: the set of clients is $\{x_1, \ldots, x_m\}$, the set of potential sites is $\{S_1, S_2, \ldots, S_n\}$, the profit c_{ij} is taken to be either zero if edge x_iS_j is present or $-\infty$ otherwise, and the fixed cost f_j (for each $j = 1, \ldots, n$) is taken to be 1.

Thus, the problem is to open the minimum number of facilities such that each client (point) x_i can be assigned to a facility (set) S_j adjacent to it (that contains x_i). It is easily seen that a solution of the UFL instance (set of sites to be opened) is optimal if and only if the corresponding solution of the set covering instance is optimal. ∎

4.2 Applications

The UFL problem is used to model many applications. Some of these applications are: bank account allocation, clustering analysis, lock-box location, location of offshore drilling platforms, economic lot sizing, machine scheduling and inventory management, portfolio management and the design of communication networks. We describe the first two applications. The *bank account location problem* arises from the fact that the clearing time for a check depends on the city i where it is cashed, and the city j where the paying bank is located. A company that pays bills by cheque to clients in several locations finds it useful to open accounts in several strategically located banks. It pays the bill to the client in city i from a bank in city j that maximizes the clearing time. Here I is the set of cities where clients are located, J is the set of potential bank locations, f_j is the cost of maintaining an account in city j, and c_{ij} is the monetary value of the clearing time between cities i and j. In *clustering analysis*, we are given a set I of objects, and the problem is to partition them into clusters such that objects in the same cluster are similar. Here, J is a subset of I, and consists of potential cluster representatives; the c_{ij}'s give the similarity between objects i and j; the f_j's may be either zero, in which case the clustering is entirely based on the similarity between objects, or the f_j's may be large, in which case the number of clusters tends to be minimized.

4.3 Linear programming formulations of the UFL problem

We start with the an integer linear programming formulation of the problem. For each potential site $j \in J$ we have a zero-one variable x_j. The intention is that a facility is opened at site j iff $x_j = 1$. For each client $i \in I$ and site $j \in J$, we have a zero-one variable y_{ij}. The intention is that the demand of client i is served by the facility at site j iff $y_{ij} = 1$.

$$
\boxed{\begin{array}{lllll}
\text{(IP)} \quad \text{maximize} & z_{IP} = & \sum_{i \in I} \sum_{j \in J} c_{ij}\, y_{ij} - \sum_{j \in J} f_j\, x_j & & \\
\quad\quad\;\; \text{subject to} & \sum_{j \in J} y_{ij} = & 1 & \forall i \in I & \\
 & y_{ij} \leq & x_j & \forall i \in I, \forall j \in J & (I1) \\
 & x_j \in & \{0,1\} & \forall j \in J & \\
 & y_{ij} \in & \{0,1\} & \forall i \in I, \forall j \in J & (I2)
\end{array}}
$$

Even if the integrality constraints on the y_{ij} are relaxed, i.e., even if the constraints (I2) above are replaced by

$$y_{ij} \geq 0 \qquad \forall i \in I,\ \forall j \in J,$$

the resulting mixed integer program is equivalent to (IP) above.
Exercise: Prove the above claim.

Another integer programming formulation, (WIP), is obtained from (IP) by replacing the mn constraints $(I1)$ by the n constraints

$$\sum_{i \in I} y_{ij} \leq m\, x_j \qquad \forall j \in J. \qquad (I1')$$

Each of the constraints $(I1')$ is obtained by summing the m constraints $y_{ij} \leq x_j$ in $(I1)$ for a fixed $j \in J$.
Fact: A set of numbers x_j $(j \in J)$, $y_{ij} (i \in I,\ j \in J)$ is a feasible solution of (WIP) if and only if it is a feasible solution of (IP).

Proof: Clearly, every feasible solution of (IP) is a feasible solution of (WIP). Consider a solution x_j, y_{ij} of (WIP). Suppose that for some $k \in J$, $x_k = 1$. Then the constraints $\sum_{j \in J} y_{ij} = 1$ (for each $i \in I$) ensure that each y_{ik} is at most 1. Otherwise, if $x_k = 0$, then $y_{ik} = 0$, $\forall i \in I$. Since all the constraints of (IP) are satisfied, the result follows. ∎

The linear programming relaxation of (IP) (respectively, (WIP)), called the strong (respectively, weak) LP relaxation and abbreviated (SLPR) (respectively, (WLPR)), is obtained by replacing the integrality restrictions on the x_j's by linear constraints, i.e., the constraints

$$x_j \in \{0,1\}, \qquad \forall j \in J$$

are replaced by the constraints

$$0 \le x_j \le 1, \qquad \forall j \in J.$$

$$\boxed{\begin{array}{llrcll}
\text{(SLPR)} & \text{maximize} & z & = & \sum_{i\in I}\sum_{j\in J} c_{ij}\, y_{ij} - \sum_{j\in J} f_j\, x_j & \\
 & \text{subject to} & \sum_{j\in J} y_{ij} & = & 1 & \forall i \in I \\
 & & y_{ij} & \le & x_j & \forall i \in I, \forall j \in J \\
 & & x_j & \le & 1 & \forall j \in J \\
 & & x_j & \ge & 0 & \forall j \in J \\
 & & y_{ij} & \ge & 0 & \forall i \in I, \forall j \in J
\end{array}}$$

Fact: Every feasible solution of (SLPR) is a feasible solution of (WLPR), but there may be feasible solutions of (WLPR) that are not feasible solutions of (SLPR).

In practice, the strong LP relaxation performs remarkably well, giving integer optimal solutions on many instances of the UFL problem. The performance of (WLPR) is far poorer. Unfortunately, the computational cost of solving (SLPR) for large instances is high, since the number of constraints is $n + m + nm$. Specialized methods for solving (SLPR) are being developed by researchers.

4.4 Duality

To write down the dual of the strong LP relaxation, we introduce dual variables u_i, $i \in I$, w_{ij}, $i \in I$, $j \in J$, and t_j, $j \in J$, corresponding to the (SLPR) constraints $\sum_{j\in J} y_{ij} = 1$ $(\forall i \in I)$, $y_{ij} - x_j \le 0$ $(\forall i \in I,\ \forall j \in J)$, and $x_j \le 1$ $(\forall_j \in J)$, respectively. The dual LP is as follows:

$$\boxed{\begin{array}{llrcll}
\text{(Dual SLPR)} & \text{minimize} & w & = & \sum_{i\in I} u_i + \sum_{j\in J} t_j & \\
 & \text{subject to} & t_j - \sum_{i\in I} w_{ij} & \ge & -f_j & \forall j \in J \\
 & & u_i + w_{ij} & \ge & c_{ij} & \forall i \in I,\ \forall j \in J \\
 & & u_i & & \text{free} & \forall i \in I \\
 & & w_{ij} & \ge & 0 & \forall i \in I,\ \forall j \in J \\
 & & t_j & \ge & 0 & \forall j \in J
\end{array}}$$

It is possible to write two condensed forms for (Dual SLPR), though the resulting duals are *not* linear programs.

4.4.1 First condensed dual

Suppose that all the variables u_i in (Dual SLPR) have fixed values. Then, to minimize the objective function, we must assign each w_{ij} the minimum value such that the constraints $u_i + w_{ij} \ge$

c_{ij} $(\forall i \in I, \forall j \in J)$ and $w_{ij} \geq 0$ $(\forall i \in I, \forall j \in J)$ are satisfied. This gives

$$w_{ij} = (c_{ij} - u_i)^+, \qquad \forall i \in I, \forall j \in J,$$

where for an expression α, $(\alpha)^+$ means $\max(\alpha, 0)$. Now, consider the variables t_j $(\forall j \in J)$. To minimize the objective function, we must assign each t_j the minimum value such that the constraints $t_j - \sum_{i \in I} w_{ij} \geq -f_j$ $(\forall j \in J)$ and $t_j \geq 0$ $(\forall j \in J)$ are satisfied. So, let

$$t_j = \left(\sum_{i \in I} w_{ij} - f_j \right)^+, \qquad \forall j \in J.$$

Substituting the formula for w_{ij} above we get

$$t_j = \left(\sum_{i \in I} (c_{ij} - u_i)^+ - f_j \right)^+, \qquad \forall j \in J.$$

This gives the first condensed dual:

$$\boxed{\text{(CD1)} \qquad w = \underset{u_1, \ldots, u_m}{\text{minimize}} \left\{ \textstyle\sum_{i \in I} u_i + \sum_{j \in J} \left(\sum_{i \in I} (c_{ij} - u_i)^+ - f_j \right)^+ \right\}.}$$

4.4.2 Second condensed dual

In a feasible solution of the dual LP (Dual SLPR), suppose that there is a $k \in J$ such that t_k is positive, that is, $\left(\sum_{i \in I} (c_{ik} - u_i)^+ - f_k \right)^+$ is positive. Then there exists a $u_\ell (\ell \in I)$, such that $c_{\ell k} - u_\ell > 0$. If we increase u_ℓ by an amount $\epsilon (\epsilon > 0)$, then t_k will decrease by ϵ, therefore the objective value stays the same; also, all dual constraints will continue to hold. It follows that there always exists an optimal solution to the dual LP with

$$t_j \leq 0 \qquad \forall j \in J.$$

To see this, use the above procedure repeatedly until each t_j is at most zero. We also add the constraints $u_i \leq \max_{j \in J}(c_{ij})$ $(\forall i \in I)$; clearly, all optimal solutions of the dual LP satisfy these constraints. This gives the second condensed dual:

$$\boxed{\begin{array}{llrll} \text{(CD2)} & \text{minimize} & w & = \sum_{i \in I} u_i & \\ & \text{subject to} & \sum_{i \in I} (c_{ij} - u_i)^+ - f_j & \leq 0 & \forall j \in J \\ & & u_i & \leq \max_{j \in J}(c_{ij}) & \forall i \in I. \end{array}}$$

4.5 Heuristics for solving the UFL problem

This section develops two well known heuristics for solving the UFL problem, namely, the greedy heuristic and the dual descent procedure. These heuristics simultaneously find a candidate solution $x_j \in \{0,1\}, y_{ij} \in \{0,1\} \quad (\forall j \in J,\ \forall i \in I)$ to the UFL problem instance as well as a feasible solution for the dual of the strong LP relaxation. By linear programming weak duality, the objective value of every feasible solution of (DUAL SLPR) gives an upper bound on the optimal value of (SLPR), and hence it gives an upper bound on the optimal value of the UFL instance. So, not only do we obtain a candidate solution to the UFL instance, but also an indication of how "far" this solution is from being optimal.

4.5.1 The greedy heuristic

We start with an empty set S of open facilities, and at each step we add to S a site $j \in J \backslash S$ that yields the maximum improvement in the objective value. For a set S, $S \subseteq J$, of open facilities, the objective value is given by

$$z(S) = \sum_{i \in I} \max_{j \in S}\{c_{ij}\} - \sum_{j \in S} f_j.$$

For a site $j \in J \backslash S$, let $p_j(S) = z(S \cup \{j\}) - z(S)$ denote the change in the objective value when a new facility is opened at j. For the currently open set of facilities S and for each client $i \in I$, define $u_i(S)$ to be $\max_{j \in S}\{c_{ij}\}$; define $u_i(\emptyset)$ to be 0. That is, $u_i(S)$ is the maximum profit obtained from serving client i using only the facilities in S. Then

$$z(S) = \sum_{i \in I} u_i(S) - \sum_{j \in S} f_j$$

and further

$$p_j(S) = z(S \cup \{j\}) - z(S) = \sum_{i \in I}(c_{ij} - u_i(S))^+ - f_j.$$

In each iteration of the greedy heuristic, we compute $p_j(S)$ for each $j \in J \backslash S$. If either $J \backslash S$ is empty or $p_j(S) \leq 0$ for each $j \in J \backslash S$, then we terminate the heuristic. Otherwise, we add to S a $j \in J \backslash S$ whose incremental value $p_j(S)$ is maximum. See Figure 4.2 for an example.

Consider the first condensed dual of (SLPR), (CD1), and for each $i \in I$ let the ith dual variable u_i be assigned the value $u_i(S)$ defined above. Then the dual objective value corresponding to S is

$$\begin{aligned} w(u(S)) &= \sum_{i \in I} u_i(S) + \sum_{j \in J} \left(\sum_{i \in I}(c_{ij} - u_i(S))^+ - f_j \right)^+ \\ &= \sum_{i \in I} u_i(S) + \sum_{j \in S} (p_j(S))^+ , \end{aligned}$$

because each $j \in S$ has $\sum_{i \in I}(c_{ij} - u_i(S))^+ - f_j = p_j(S)$.

For the final solution S^G found by the greedy heuristic, each $j \in J \backslash S^G$ has $p_j(S^G) \leq 0$, hence the dual objective value is

$$w(u(S^G)) = \sum_{i \in I} u_i(S^G).$$

Figure 4.2: Solving an example of the uncapacitated facility location problem by the greedy heuristic (from Chapter 3, "Discrete Location Theory", Ed., Mirchandani & Francis).

$$
\begin{aligned}
m &= 4, \quad n = 6 \\
f &= [\ 3\ \ 2\ \ 2\ \ 2\ \ 3\ \ 3\ ] \\
C &= \begin{bmatrix} 6 & 6 & 8 & 6 & 0 & 6 \\ 6 & 8 & 6 & 0 & 6 & 6 \\ 5 & 0 & 3 & 6 & 3 & 0 \\ 2 & 3 & 0 & 2 & 4 & 4 \end{bmatrix}.
\end{aligned}
$$

Iteration 1:

$$S_0 = \emptyset, \quad Z(S_0) = 0, \quad u(S_0) = [0\ 0\ 0\ 0].$$

$p_1(S_0)$	$p_2(S_0)$	$p_3(S_0)$	$p_4(S_0)$	$p_5(S_0)$	$p_6(S_0)$
16	15	15	12	10	13

$$w(u(S_0)) = (0) + (81) = 81.$$
$$S_1 := \{1\}, \quad \text{since } p_1(S_0) > 0 \text{ is maximum.}$$

Iteration 2:

$$S_1 = \{1\}, \quad z(S_1) = 16, \quad u(S_1) = [6\ 6\ 5\ 2].$$

$p_2(S_1)$	$p_3(S_1)$	$p_4(S_1)$	$p_5(S_1)$	$p_6(S_1)$
$(2+1)-2=1$	$(2)-2=0$	$(1)-2=-1$	$(2)-3=-1$	$(2)-3=-1$

$$w(u(S_1)) = (19) + (1) = 20.$$
$$S_2 := \{1,2\}, \quad \text{since } p_2(S_1) > 0 \text{ is maximum.}$$

Iteration 3:

$$S_2 = \{1,2\}, \quad z(S_2) = 16 + 1 = 17, \quad u(S_2) = [6\ 8\ 5\ 3].$$

$p_3(S_2)$	$p_4(S_2)$	$p_5(S_2)$	$p_6(S_2)$
$(2)-2=0$	$(1)-2=-1$	$(1)-3=-2$	$(1)-3=-2$

$$w(u(S_2)) = (22) + (0) = 22.$$
$$\textbf{stop}, \text{ since } \ p_j(S_2) \le 0, \quad \forall j \in J \backslash S_2.$$

The greedy solution is $S^G = \{1,2\}$, with objective value $Z^G = 17$.
The dual greedy value W^G is the best upper bound computed on the optimal value: $W^G = \min_{t=1,2,\ldots} w(u(S_t)) = w(u(S_1)) = 20$

The greedy heuristic actually computes the dual objective value $w(u(S))$ for the set S in each iteration, and takes the smallest of these values to be the upper bound W^G that is returned at the end, since this value is the least upper bound computed on the optimal value of (SLPR). See the example in Figure 4.2.

The proofs of the next two results may be found in Cornuejols, Fisher and Nemhauser (1977). Below, $e \approx 2.71828$ denotes the base of the natural logarithm.

Theorem 4.2 *For the UFL problem, the objective value Z^G of the candidate solution found by the greedy heuristic is at least*

$$\frac{e-1}{e}W^G + \frac{1}{e}\left(\sum_{i\in I}\min_{j\in J} c_{ij} - \sum_{j\in J} f_j\right).$$

Theorem 4.3 *For the k-median problem with $c_{ij} \geq 0$ for all i and j (here, $f_j = 0$ for all j), the objective value Z^G of the candidate solution found by the greedy heuristic is at least*

$$\frac{e-1}{e}W^G.$$

4.5.2 The dual descent procedure

The dual descent procedure for solving the UFL problem works well on most instances, but it may perform poorly on hard instances. This procedure is used by the program DUALOC (by Erlenkotter (1978)), which is one of the best programs available for solving UFL problems.

The procedure attempts to find a good solution $u_1, \ldots, u_m$ to the second condensed dual, (CD2), and then uses the complementary slackness conditions to find a candidate solution for the UFL problem instance. The dual variables u_i ($\forall i \in I$) are initialized to $\max_{j\in J}\{c_{ij}\}$. This gives a feasible solution to the constraints

$$\sum_{i\in I}(c_{ij} - u_i)^+ - f_j \leq 0 \qquad \forall j \in J \tag{$*$}$$

of (CD2). Then the procedure repeatedly steps through all the indices $i \in I$ in an arbitrary but fixed order and attempts to decrease u_i as follows: if u_i can be decreased to

$$\max_{j\in J}\ \{c_{ij} : c_{ij} < u_i\}$$

(i.e., the largest profit c_{ij} for client i that is strictly less than u_i) without violating the constraint ($*$) above, then this is done, otherwise u_i is decreased to the smallest value such that the constraint ($*$) continues to be satisfied. The procedure terminates when there is an iteration such that no u_i ($\forall i \in I$) is decreased. The complementary slackness conditions for finding a candidate solution to the UFL instance are as follows: A pair of feasible solutions x_j, y_{ij} ($\forall i \in I$, $\forall j \in J$) and u_i ($\forall i \in I$) to (SLPR) and it dual, respectively, forms optimal solutions only if

$$x_j = 0 \qquad \text{or} \qquad t_j - \sum_{i\in I} w_{ij} = -f_j \qquad (\forall j \in J),$$

or, in term of (CD2), only if

$$x_j = 0 \qquad \text{or} \qquad \sum_{i\in I}(c_{ij} - u_i)^+ - f_j = 0 \qquad (\forall j \in J).$$

This gives us the following complementarity conditions:

$$x_j \left(\sum_{i\in I}(c_{ij} - u_i)^+ - f_j\right) = 0 \qquad \forall j \in J.$$

Given a feasible solution $(u_1, \ldots, u_m)$ of (CD2), if $\sum_{i\in I}(c_{ij} - u_i)^+ - f_j < 0$, then we fix x_j at zero. Let $J(u)$, $J(u) \subseteq J$, be the set of all sites j such that the constraint $(*)$ holds with equality, i.e.,

$$J(u) = \left\{ j \in J : \sum_{i\in I}(c_{ij} - u_i)^+ - f_j = 0 \right\}.$$

Then, we find a minimal subset $K(u)$ of $J(u)$ such that for all $i \in I$, $\max_{j\in K(u)}\{c_{ij}\} = \max_{j\in J(u)} c_{ij}$ (possibly, $K(u) = J(u)$). The set $K(u)$ is the set of sites where facilities are opened. The objective value of the UFL instance corresponding to $K(u)$ is given by

$$Z^{DD} = \sum_{i\in I} \max_{j\in K(u)}\{c_{ij}\} - \sum_{j\in K(u)} f_j.$$

An example is shown in Figure 4.3.

The dual descent procedure cannot find the optimal solution to every instance of the UFL problem, simply because the optimal value of the UFL instance may be less than the optimal value of the strong LP relaxation. Moreover, the procedure is *not* guaranteed to find an optimal solution to the condensed dual (CD2). The example in Figure 4.4 has an optimal dual solution $u^* = (1, 1, 1)$ with objective value $w^* = 3$, however, the solution found by the dual descent procedure is $u^{DD} = (0, 2, 2)$ with objective value $w^{DD} = 4$.

Proposition 4.4 *Let $K(u)$ be defined as above. For each client i, let k_i be the number of sites in $K(u)$ whose profit c_{ij} is greater than u_i, $k_i = |\{j \in K(u) : c_{ij} > u_i\}|$. If k_i is at most one for each $i \in I$, then $K(u)$ is an optimal set of open facilities for the UFL problem.*

Proof: Focus on a client i. If $k_i = 0$, then we have

$$\max_{j\in K(u)}\{c_{ij}\} = u_i = u_i + \sum_{j\in K(u)}(c_{ij} - u_i)^+,$$

and if $k_i = 1$, then we have

$$\max_{j\in K(u)}\{c_{ij}\} = u_i + \sum_{j\in K(u)}(c_{ij} - u_i)^+.$$

Figure 4.3: Solving an example of the uncapacitated facility location problem by the dual descent procedure (from Chapter 3, "Discrete Location Theory", Ed., Mirchandani & Francis).

$$
\begin{aligned}
m &= 4, \quad n = 6 \\
f &= [\ 3 \ \ 2 \ \ 2 \ \ 2 \ \ 3 \ \ 3\ ] \\
C &= \begin{bmatrix} 6 & 6 & 8 & 6 & 0 & 6 \\ 6 & 8 & 6 & 0 & 6 & 6 \\ 5 & 0 & 3 & 6 & 3 & 0 \\ 2 & 3 & 0 & 2 & 4 & 4 \end{bmatrix}.
\end{aligned}
$$

Step no.	u_1	u_2	u_3	u_4	$\sum_{i\in I} (c_{ij} - u_i)^+ - f_j$					
					$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$
0	8	8	6	4	-3	-2	-2	-2	-3	-3
1	6	8	6	4	-3	-2	0	-2	-3	-3
2	6	6	6	4	-3	0	0	-2	-3	-3
3	6	6	5	4	-3	0	0	-1	-3	-3
4	6	6	5	3	-3	0	0	-1	-2	-2
5	6	6	4	3	-2	0	0	0	-2	-2

In the last iteration, u_1 cannot decrease otherwise the constraint

$$\sum_{i\in I}(c_{ij} - u_i)^+ - f_j \leq 0 \qquad (*)$$

is violated for $j = 3$, and u_2 cannot decrease otherwise the constraint $(*)$ for $j = 2$ is violated; u_3 is decreased from 5 to 4 (not 3, since that violates the constraint $(*)$ for $j = 4$); u_4 cannot decrease otherwise the constraint $(*)$ for $j = 2$ is violated.

The dual objective value is $w^{DD}(u) = \sum_{i\in I} u_i = 19$, $J(u) = \{2, 3, 4\}$, and $K(u) = \{2, 3, 4\}$.

Opening facilities at sites 2, 3, and 4 gives an objective value of $Z^{DD} = (8+8+6+3)-(2+2+2) = 19$. This solution is optimal, since $Z^{DD} = w^{DD}(u)$. Optimality also follows from Proposition 4.4.

Figure 4.4: An example of the uncapacitated facility location problem on which the dual descent procedure fails to find an optimal solution for the dual (CD2) (from Chapter 3, "Discrete Location Theory", Ed., Mirchandani & Francis).

$$\begin{aligned} m &= 3, \quad n = 3 \\ f &= [\ 2 \quad 2 \quad 2\ ] \\ C &= \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}. \end{aligned}$$

Step no.	u_1	u_2	u_3	$\sum_{i\in I}(c_{ij}-u_i)^+ - f_j$, $j=1$	$j=2$	$j=3$
0	2	2	2	-2	-2	-2
1	0	2	2	-2	0	0

In the last iteration, u_1, u_2 and u_3 are prevented from decreasing further by the constraints $(*)$ for $j = 2$, 3 and 2, respectively.

$$w^{DD}(u) = \sum_{i\in I} u_i = 4 > w^* = 3.$$

Now, consider the objective value of the UFL instance corresponding to the given solution $K(u)$,

$$\begin{aligned} z(K(u)) &= \sum_{i\in I} \max_{j\in K(u)} \{c_{ij}\} - \sum_{j\in K(u)} f_j \\ &= \sum_{i\in I} \left(u_i + \sum_{j\in K(u)} (c_{ij}-u_i)^+ \right) - \sum_{j\in K(u)} f_j, \\ &= \sum_{i\in I} u_i + \sum_{j\in K(u)} \left(\sum_{i\in I} (c_{ij}-u_i)^+ - f_j \right) \\ &= \sum_{i\in I} u_i, \quad \text{(since complementarity ensures that for each } j \in K(u), \\ &\qquad\qquad \sum_{i\in I}(c_{ij}-u_i)^+ - f_j \text{ is at zero)} \\ &= w(u). \end{aligned}$$

Since the dual objective value equals $z(K(u))$, $K(u)$ must be an optimal solution. ∎

4.6 The filtering and rounding method for location problems

4.6.1 Introduction

This section describes the filtering and rounding method, a recently developed method for finding approximately optimal solutions to NP-hard location problems. The method starts from an optimal (or near-optimal) solution to an appropriately formulated LP (linear programming) relaxation, and obtains a near-optimal solution to the original location problem. There are two major steps. The filtering step uses the optimal LP solution to construct a "filtered problem" by fixing some variables to zero in the integer programming formulation. Thus the filtered problem is a restricted version of the original location problem. The critical point is this:

> *every integral solution* of the filtered problem is guaranteed to have its objective value "near" (i.e., within a $(1+\epsilon)$-factor of) the optimal value of the original LP.

The rounding step produces a feasible integral solution to the filtered problem, either by using a simple randomized heuristic for "rounding" a "fractional solution" (i.e., a solution to the LP relaxation) to the filtered problem, or by using a simple greedy heuristic (nonrandomized) for the same task.

In our application, the "minimize" UFL problem with symmetric costs that satisfy the triangle inequality, the method does give a feasible integral solution whose objective value is "near" the optimal value.

The filtering and rounding method originated from the work of Lin & Vitter (1992). They apply it to the k-median problem. The application to the "minimize" UFL problem with restricted costs is due to Aardal, Shmoys and Tardos (1997).

4.6.2 An integer programming formulation for "minimize" UFL problems and its LP relaxation

Let V be a set of nodes. Let $I \subseteq V$ be a set of *client nodes*, and let $J \subseteq V$ be a set of *site nodes.* For each pair of nodes $v, w \in V$, let c_{vw} be the *service cost* for v, w. For each site node $j \in J$, let f_j be the *fixed cost* for opening a facility at j. The "minimize" UFL problem is to open facilities at a subset of sites $J^* \subseteq J$ and to assign each client $i \in I$ to a facility $j \in J^*$ so as to minimize the sum of the "service costs", namely, $\sum_{i\in I}\{c_{ij} \mid j \in J^* \text{ is the unique facility assigned to serve client } i\}$, and the "fixed costs", namely, $\sum_{j\in J^*} f_j$.

Here is an integer linear programming formulation of the "minimize" UFL problem. Let (IP) denote this integer linear program, and let z_{IP} denote the optimal value of (IP). For each site $j \in J$ we have a zero-one variable y_j. The intention is that a facility is opened at site j iff $y_j = 1$. For each client $i \in I$ and site $j \in J$, we have a zero-one variable x_{ij}. The intention is that the demand of client i is served by the facility at site j iff $x_{ij} = 1$.

$$
\begin{array}{|lrcll|}
\hline
\text{(IP)} & z_{IP} & = & \text{minimize } \sum_{i\in I}\sum_{j\in J} c_{ij}\, x_{ij} + \sum_{j\in J} f_j\, y_j & \\
 & \text{subject to } \sum_{j\in J} x_{ij} & = & 1 & \forall i \in I \\
 & x_{ij} & \leq & y_j & \forall i \in I, \forall j \in J \\
 & y_j & \in & \{0,1\} & \forall j \in J \\
 & x_{ij} & \in & \{0,1\} & \forall i \in I, \forall j \in J \\
\hline
\end{array}
$$

The linear programming relaxation of (IP) is obtained by replacing the integrality restrictions on the y_j's by linear constraints, i.e., the constraints

$$y_j \in \{0,1\}, \qquad \forall j \in J$$

are replaced by the constraints

$$0 \leq y_j \leq 1, \qquad \forall j \in J.$$

Also, the integrality restrictions on the x_{ij}'s are replaced by nonnegativity constraints

$$0 \leq x_{ij}, \qquad \forall i \in I, \forall j \in J.$$

Let (LP) denote this relaxation, and let z_{LP} denote the optimal value of (LP).

4.6.3 The Aardal-Shmoys-Tardos algorithm for "minimize" UFL problems

Recall that a function $c : V \times V \rightarrow \Re$ is said to satisfy the *triangle inequality* if the following holds

$$\text{for all triples of nodes } u, v, w, \quad c(v,w) \leq c(v,u) + c(u,w).$$

Theorem 4.5 (Aardal, Shmoys & Tardos (1997)) *Suppose that the service costs c and the fixed costs f are nonnegative. If the costs matrix $[c_{vw}]$ $(\forall v, w \in V)$ is symmetric and satisfies the triangle inequality, then there is a 3-approximation algorithm for the "minimize" UFL problem. In fact, the approximation algorithm delivers a feasible integral solution $\widetilde{x}, \widetilde{y}$ to (IP) whose objective value is at most $3z_{LP}$.*

We will prove a weaker version of this theorem. The proof will describe an approximation algorithm, and we will prove that it achieves an approximation guarantee of 4. Let ϵ be a fixed parameter. (It will turn out that the best choice is $\epsilon = 1/3$.) Focus on the objective function of (IP) or (LP), and note that it consists of a "service cost" part $(\sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij})$ and a "fixed cost" part $(\sum_{j\in J} f_j y_j)$. Let z^s_{LP} and z^f_{LP} denote the values of the "service cost" part and of the "fixed cost" part in an optimal solution to (LP).

Let $\widehat{x}, \widehat{y}$ be an optimal solution to (LP). For each client $i \in I$, define $\widehat{C}_i = \sum_{j\in J} c_{ij}\widehat{x}_{ij}$. Note that $z^s_{LP} = \sum_{i\in I} \widehat{C}_i$. As in the Lin & Vitter algorithm for the k-median problem, take $V_i = \{j \in J \mid c_{ij} \leq (1+\epsilon)\widehat{C}_i\}$.

Recall the main lemma in Lin & Vitter's analysis of their algorithm.

Lemma 4.6 *For all $i \in I$,*

$$\sum_{j \in V_i} \widehat{y}_j > \frac{\epsilon}{1+\epsilon}.$$

We construct a "near optimal" feasible solution $\widetilde{x}, \widetilde{y}$ to the "filtered" (LP), by taking

$$\widetilde{y}_j = \min\{1,\ (\frac{1+\epsilon}{\epsilon})\widehat{y}_j\}, \quad \forall j \in J.$$

By the above lemma, for all $i \in I$, we have $\sum_{j \in V_i} \widetilde{y}_j \geq 1$.

We take $[\widetilde{x}_{ij}]$ $(\forall i \in I, \forall j \in J)$ to be optimal with respect to $\widetilde{y}_j$ $(\forall j \in J)$. (That is, we take $\widetilde{y}$ to be fixed, and use the greedy procedure to find $\widetilde{x}$ such that the service costs are minimized.)

The second part of the algorithm starts with the "fractional" solution $\widetilde{x}, \widetilde{y}$ and executes several iterations. In each iteration, at least two fractional variables $\widetilde{y}_j$ are assigned integer values. In more detail, one fractional variable $\widetilde{y}_j$ is assigned the value 1, and one or more fractional variables $\widetilde{y}_j$ are assigned the value 0. This immediately implies that several of the fractional variables $\widetilde{x}_{ij}$ are assigned integer values of 0 or 1. We use $\widetilde{x}, \widetilde{y}$ to denote the current solution, through all iterations. At the start of the second part, $\widetilde{x}, \widetilde{y}$ is obtained from the optimal LP solution as described above. At termination, $\widetilde{x}, \widetilde{y}$ will be a feasible integral solution to (IP), and its objective value will be proved to be $\leq 4z_{LP}$.

Let $\widetilde{J} = \{j \in J \mid 0 < \widetilde{y}_j < 1\}$ denote the set of sites whose variables have fractional values in the current solution $\widetilde{y}$. Let $\widetilde{I} = \{i \in I \mid \exists j \in J \text{ such that } 0 < \widetilde{x}_{ij} < 1\}$ denote the set of clients that are partially assigned to a "fractional site."

Fact 4.7 *For each client i in $\widetilde{I}$, there is no site $j \in J$ such that $\widetilde{x}_{ij} > 0$ and $\widetilde{y}_j = 1$, and moreover, $\{j \in J \mid \widetilde{x}_{ij} > 0\}$ must contain at least two "fractional sites" $j \in \widetilde{J}$.*

The goal in each iteration is to maintain the following:

Induction hypothesis:

(1) $\widetilde{x}, \widetilde{y}$ is a feasible solution to (LP).

(2) $\displaystyle\sum_{j \in J} f_j \widetilde{y}_j \quad \leq \quad (\frac{1+\epsilon}{\epsilon}) z^f_{LP} \quad = \quad (\frac{1+\epsilon}{\epsilon}) \sum_{j \in J} f_j \widehat{y}_j.$

(3) $\forall i \in \widetilde{I}\ :\ \widetilde{x}_{ij} > 0 \Longrightarrow c_{ij} \leq (1+\epsilon)\widehat{C}_i.$

(4) $\forall i \in I\ :\ \displaystyle\sum_{j \in J} c_{ij}\widetilde{x}_{ij} \leq 3(1+\epsilon)\widehat{C}_i.$

The induction basis is that parts (1)–(4) of the induction hypothesis hold at the start (of the second part of the algorithm). Part (1) follows from Lin & Vitter's main lemma; part (2) holds by our choice of $\widetilde{y}_j \leq (1+\epsilon)\widehat{y}_j/\epsilon$, $\forall j \in J$; parts (3) and (4) hold by our choice of V_i, $\forall i \in I$.

An iteration starts by choosing a client $h \in \widetilde{I}$ that minimizes $\widehat{C}_h$, i.e., $\widehat{C}_h \leq \widehat{C}_i, \forall i \in \widetilde{I}$.

Let $L = \{j \in J \mid \widetilde{x}_{hj} > 0\}$, and note that $L \subseteq \widetilde{J}$. That is, L consists of the sites assigned to serve the client h by the current solution $\widetilde{x}, \widetilde{y}$. By the fact above, every site in L is "fractional", and L must have ≥ 2 sites.

Let $H = \{i \in I \mid \exists j \in L \text{ such that } \widetilde{x}_{ij} > 0\}$, and note that $H \subseteq \widetilde{I}$. That is, H consists of those clients who are served by at least one "fractional site" in L according to the current solution $\widetilde{x}, \widetilde{y}$.

Choose $\ell \in L$ that minimizes f_ℓ, i.e., $f_\ell \leq f_j, \forall j \in L$. In words, ℓ is the "fractional site" serving our chosen client h that has the smallest fixed cost.

We now change the current solution $\widetilde{x}, \widetilde{y}$ to $\widetilde{x}', \widetilde{y}'$:
we take $\widetilde{y}'_\ell = 1$ and take $\widetilde{y}'_j = 0, \forall j \in L - \{\ell\}$;
for all clients $i \in H$, we take $\widetilde{x}'_{i\ell} = 1$ and take $\widetilde{x}'_{ij} = 0, \forall j \in J - \{\ell\}$;
for all the remaining variables, we take $\widetilde{y}'_j = \widetilde{y}_j$ and $\widetilde{x}'_{ij} = \widetilde{x}_{ij}$.

Finally, we must prove that the new solution $\widetilde{x}', \widetilde{y}'$ satisfies the induction hypothesis, assuming that the previous solution $\widetilde{x}, \widetilde{y}$ did so.

Part(1) obviously holds.

For part(2), note that $\sum_{j \in J} f_j \widetilde{y}_j = \sum_{j \in L} f_j \widetilde{y}_j + \sum_{j \in J-L} f_j \widetilde{y}_j$. Focus on the first term, $\sum_{j \in L} f_j \widetilde{y}_j$. We claim that it is at least $f_\ell = \sum_{j \in L} f_j \widetilde{y}'_j$. To see this, firstly note that $\sum_{j \in L} \widetilde{y}_j \geq 1$ (since $\widetilde{x}, \widetilde{y}$ is feasible for (LP), we have $\sum_{j \in J} \widetilde{x}_{hj} = \sum_{j \in L} \widetilde{x}_{hj} = 1$ and $\widetilde{x}_{hj} \leq \widetilde{y}_j, \forall j \in J$). Secondly, note that $f_\ell \leq f_j, \forall j \in L$, by our choice of ℓ.

Part(3) obviously holds because all clients i in the new $\widetilde{I}$ were in $\widetilde{I} - H$, and so have the same values before and after the iteration for their variables $\widetilde{x}_{ij} (\forall j \in J)$.

For part(4), consider any client $i \in H$. We claim that $c_{i\ell} \leq 3(1+\epsilon)\widehat{C}_i$. To see this, firstly note that by the definition of H, there must be a site $j \in L$ with $\widetilde{x}_{ij} > 0$, hence $c_{ij} \leq (1+\epsilon)\widehat{C}_i$. Secondly, note that by the definition of L, both $\widetilde{x}_{hj} > 0$ and $\widetilde{x}_{h\ell} > 0$, hence $c_{hj} \leq (1+\epsilon)\widehat{C}_h$ and $c_{h\ell} \leq (1+\epsilon)\widehat{C}_h$. Since $[c_{vw}]$ is symmetric and satisfies the triangle inequality, and moreover, we chose client h to minimize $\widehat{C}_h$ over all $i \in \widetilde{I}$, we have

$$c_{i\ell} \leq c_{ij} + c_{jh} + c_{h\ell} \leq c_{ij} + c_{hj} + c_{h\ell} \leq (1+\epsilon)(\widehat{C}_i + \widehat{C}_h + \widehat{C}_h) \leq (1+\epsilon)(3\widehat{C}_i).$$

To derive the approximation guarantee of 4, we take $\epsilon = 1/3$ and apply the induction hypothesis to the final solution $\widetilde{x}, \widetilde{y}$, which is guaranteed to be integral. Then we see that the final solution $\widetilde{x}, \widetilde{y}$ has service cost

$$\leq 3(1+\epsilon)\sum_{i \in I} \widehat{C}_i = 3(1+\epsilon) z^s_{LP} \leq 4 z^s_{LP},$$

(by part(3) of the induction hypothesis), and has fixed cost

$$\leq (\frac{1+\epsilon}{\epsilon}) z^f_{LP} \leq 4 z^f_{LP},$$

(by part(4) of the induction hypothesis).

4.7 Exercises

1. (From Chapter 3, "Discrete Location Theory", Ed., Mirchandani and Francis.) Consider the instance of the UFL problem defined by $m = 5,\ n = 8,\ f_j = 2$ for $j = 1, \ldots, 8$, and

$$C = \begin{bmatrix} 3 & 6 & 3 & 5 & 6 & 4 & 3 & 0 \\ 4 & 4 & 5 & 3 & 5 & 2 & 0 & 5 \\ 4 & 3 & 4 & 3 & 4 & 0 & 4 & 5 \\ 5 & 3 & 4 & 6 & 0 & 4 & 6 & 3 \\ 5 & 5 & 4 & 0 & 3 & 6 & 5 & 3 \end{bmatrix}.$$

   (a) Use the greedy heuristic to find a solution. Give the greedy value Z^G and the dual greedy value W^G.

   (b) Give the optimal value W' of the weak linear programming relaxation (use the CPLEX optimizer).

   (c) Apply the dual descent procedure, cycling through the indices i. Give the value $W^{DD}(u)$ found by this procedure. Is the set $K(u)$ an optimal set of open facilities for this problem instance?

2. Repeat parts (a)–(c) of the previous problem for another instance of the UFL problem defined by $m = 4,\ n = 5,\ f = [1\ 2\ 3\ 4\ 5]$, and

$$C = \begin{bmatrix} 9 & 7 & 6 & 7 & 8 \\ 7 & 9 & 8 & 5 & 6 \\ 8 & 7 & 10 & 9 & 8 \\ 5 & 9 & 8 & 11 & 10 \end{bmatrix}.$$

3. (From Chapter 3, "Discrete Location Theory", Ed., Mirchandani and Francis.) Let P denote the set of nine integral points in the square $0 \le x \le 2,\ 0 \le y \le 2$. For any point $i \in P$, let x_i and y_i be its coordinates. Consider the instance of the UFL problem defined by $m = n = 9$, $f_j = 2$ for all $j \in P$, and $c_{ij} = -|x_i - x_j| - |y_i - y_j|$ for all $i,\ j \in P$.

   (a) Use the greedy heuristic to find the values Z^G and W^G.

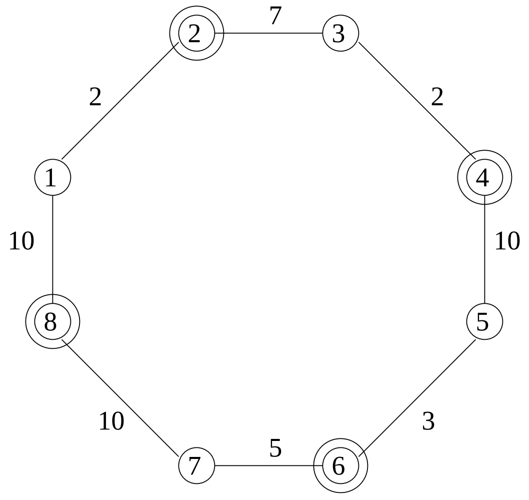
   (b) Apply the the dual descent procedure and find the value $W^{DD}(u)$. Is the set $K(u)$ an optimal set of open facilities for this problem instance?

4. Apply the Aardal-Shmoys-Tardos algorithm (based on the Filtering & Rounding method) to the following "minimize" UFL problem:

$$V = \{1, 2, 3, 4, 5, 6, 7, 8\}, \quad I = \{1, 3, 5, 7\}, \quad J = \{2, 4, 6, 8\}.$$

   The fixed costs f_j for the sites $j \in J$ are

$$f_2 = 400, \quad f_4 = 800, \quad f_6 = 400, \quad f_8 = 200.$$



$$[c_{vw}] = \begin{bmatrix} 0 & 2 & 9 & 11 & 21 & 24 & 20 & 10 \\ 2 & 0 & 7 & 9 & 19 & 22 & 22 & 12 \\ 9 & 7 & 0 & 2 & 12 & 15 & 20 & 19 \\ 11 & 9 & 2 & 0 & 10 & 13 & 18 & 21 \\ 21 & 19 & 12 & 10 & 0 & 3 & 8 & 18 \\ 24 & 22 & 15 & 13 & 3 & 0 & 5 & 15 \\ 20 & 22 & 20 & 18 & 8 & 5 & 0 & 10 \\ 10 & 12 & 19 & 21 & 18 & 15 & 10 & 0 \end{bmatrix}$$

Figure 4.5: Graph $G = (V, E)$ and cost matrix $[c_{vw}]$ for Exercise 4.

The service costs are given by the matrix $[c_{vw}]$ $(\forall v, w \in V \times V)$ in Figure 4.5. Note that the matrix $[c_{vw}]$ is symmetric and satisfies the triangle inequality.

Start with the fractional solution $\widehat{y}$ given by

$$\widehat{y}_2 = 1/4, \quad \widehat{y}_4 = 1/4, \quad \widehat{y}_6 = 1/4, \quad \widehat{y}_8 = 1/4.$$

Take the parameter $\epsilon = 1/2$.

5. The aim here is to consider a generalization of the "minimize" UFL problem and to develop a 4-approximation algorithm.

   In the "minimize" UFL problem, suppose that each client $i \in I$ has a nonnegative real-valued demand h_i (different clients may have different demands). Consider the modified objective function

   $$\text{minimize} \quad \sum_{i \in I} \sum_{j \in J} h_i c_{ij} x_{ij} + \sum_{j \in J} f_j y_j.$$

   Modify the Aardal-Shmoys-Tardos algorithm so that it finds a feasible integral solution $\widetilde{x}, \widetilde{y}$ whose objective value is $\leq 4 z_{LP}$, where z_{LP} is the objective value of the LP relaxation using the above objective function.

Bibliography

[1] G. Cornuejols, G. L. Nemhauser and L. A. Wolsey, *The uncapacitated facility location problem.* in P. Mirchandani and R. Francis, editors, *Discrete Location Theory*, John Wiley and Sons Inc., New York, 1990, pp. 119–171.

[2] R. C. Larson and A. R. Odoni, *Urban Operations Research.* Prentice-Hall, Englewood Cliffs, N. J., 1981.

[3] J.-H. Lin and J. S. Vitter, *ϵ-approximations with minimum packing constraint violation*, Proc. 24th Annual ACM Symp. on Theory of Computing (1992), pp. 771–782.

[4] J.-H. Lin and J. S. Vitter, *Approximation algorithms for geometric location problems*, Inf. Proc. Lett. **44** (1992), pp. 245–249.

[5] P. B. Mirchandani and R. L. Francis, *Discrete Location Theory.* John Wiley & Sons, New York, 1990.

[6] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley and Sons Inc., New York, 1988.

[7] P. Raghavan, *Probabilistic construction of deterministic algorithms: approximating packing integer programs*, J. Comp. Sys. Sci. **37** (1988), pp. 130–143.

[8] P. Raghavan and C. D. Thompson, *Randomized rounding: a technique for provably good algorithms and algorithmic proofs*, Combinatorica **7** (1987), pp. 365–374.

[9] D. B. Shmoys, E. Tardos and K. Aardal, *Approximation algorithms for facility location problems.* Proc. 29th ACM Sympos. on Theory of Computing (1997), pp. 265–274.

Chapter 5

Minimum Spanning Trees

This chapter is based on well-known results. Our discussion follows Ahuja, Magnanti & Orlin [1, Chapter 13] and the survey by Magnanti and Wolsey [8].

The focus of this chapter is on the linear programming formulation of the minimum spanning tree problem due to Edmonds. Also, we describe the well-known algorithms of Kruskal and Prim for finding minimum spanning trees. For further details and data structures for efficiently implementing these algorithms, we refer the reader to Cormen et al [4].

5.1 Applications

We start with some applications of the minimum spanning tree problem.

(a) Minimum-cost road interconnection network:
There are n towns in a region. For certain pairs i and j it is feasible to build a direct road between i and j, and there is a cost c_{ij} incurred if the road ij is built. The problem is to construct enough roads so that every pair of towns can communicate (perhaps indirectly), and the total construction cost must be minimized.

(b) Finding routes with maximum bottleneck capacity:
There are n computers connected by a network such that for certain pairs i and j there is a direct link with a capacity of c_{ij} bits/second. For each pair of computers, the problem is to find a path between them (i.e., a sequence of direct links) such that the bottleneck capacity (i.e., the smallest link capacity in the path) is as large as possible. (See Problem 3 for more details.)

(c) Reducing data storage:
There is a 2-dimensional array such that the rows have similar entries and differ only at a few places. Let c_{ij} denote the number of different entries in rows i and j. The problem is to store the array using a small amount of storage space. One solution is to store a reference row i completely, and for the remaining rows j to store only the positions and entries where rows i and j differ.

(d) Cluster analysis:
Given a set of n data points, the problem is to partition it into "clusters" such that data points within a cluster are "closely related" to each other. Kruskal's minimum spanning tree algorithm (see Section 5.4) maintains "clusters" and at each iteration "merges" the "closest" two clusters. The algorithm starts with n clusters, and ends with one. Each stage of the algorithm gives a partition of the data points into clusters. Consequently, several solutions to the clustering analysis problem can be found by running Kruskal's algorithm on the given data points.

5.2 Trees and cuts

This section has some fundamental results on trees and cuts that are used in this chapter and the later chapters. We recall a few definitions from Chapter 1. A tree is a connected graph that has no cycles. Given a node set $Q \subseteq V$, $\delta(Q)$ denotes the set of all edges with one end in Q and the other end in $V \backslash Q$. A *cut* consists of all edges that have one end in Q and the other end in $V \backslash Q$, where Q is a node set. this cut is denoted $(Q, V \backslash Q)$. Clearly, if $\emptyset \neq Q \neq V$, then $\delta(Q) = (Q, V \backslash Q)$.

Proposition 5.1 *A graph $G = (V, E)$ is connected if and only if for every node set $Q \subset V$, $\emptyset \neq Q \neq V$, $\delta(Q) \neq \emptyset$.*

Proof: Suppose that there is a node set Q, $\emptyset \neq Q \neq V$, with $\delta(Q) = \emptyset$. Then G is not connected because for any node $v \in Q$ and any node $w \in V \backslash Q$ (both v and w exist) there is no path from v to w in G. For the other direction, start with any node v; let $Q = \{v\}$ and let F be an edge set that is initially empty. As long as $Q \neq V$, there is an edge qz in $\delta(Q)$ with $q \in Q$ and $z \notin Q$. Repeatedly, add the edge qz to F and add z to Q. By induction on the number of steps, observe that the subgraph (Q, F) contains a path from the start node v to each node $q \in Q$. Hence, when $Q = V$, then (Q, F) is a connected spanning subgraph of G, implying that G is connected. (In fact, the final F is (the edge set of) a spanning tree of G.) ∎

Theorem 5.2 *Let $T = (V, F)$ be a graph. The following statements are equivalent:*

(a) T is a tree, or equivalently, T is connected and has no cycles.

(b) There is exactly one path between each pair of nodes in T.

(c) T contains $|V| - 1$ edges and is connected.

(d) T contains $|V| - 1$ edges and has no cycles.

Theorem 5.3 *Let $G = (V, E)$ be a graph, and let $F \subseteq E$ be an edge set such that $T = (V, F)$ is a spanning tree of G.*

(a) For every edge $e \in F$ (e is a tree edge) the subgraph $T \backslash \{e\} = (V, F \backslash \{e\})$ is a forest consisting of two trees, say, $T_1 = (V_1, F_1)$ and $T_2 = (V_2, F_2)$ (i.e., removing e disconnects T into two trees). The node sets V_1, V_2 form a partition of V. For every edge $\bar{e}$ in the cut (V_1, V_2), the subgraph $(T \backslash \{e\}) \cup \{\bar{e}\} = (V, (F \backslash \{e\}) \cup \{\bar{e}\})$ is a spanning tree of G (i.e., adding $\bar{e}$ joins the two trees into one).

(b) If e' is an edge in $E \backslash F$ (e' is a nontree edge), then the subgraph $T' = (V, F \cup \{e'\})$ has exactly one cycle (i.e., adding e' to T gives one cycle). The cycle consists of e' and the unique path in T between the ends of e'.

(c) Let T' be the subgraph in part (b). For every edge e in the unique cycle of T' the subgraph $T' \backslash \{e\} = (V, (F \cup \{e'\}) \backslash \{e\})$ is a spanning tree of G (i.e., removing e from T' gives a spanning tree).

5.3 Minimum spanning trees

Let $G = (V, E)$ be a graph, and let each edge $ij \in E$ have a real-valued cost $c(ij)$. We also use c_{ij} to denote the cost. The cost of a subgraph $G' = (V', E')$, denoted $c(G')$ or $c(E')$, is defined to be the sum of the costs of all edges in G', $\sum_{e \in E'} c(e)$. A *minimum spanning tree* of G is defined to be a spanning tree of G whose cost is minimum. Recall that a spanning tree is a connected subgraph of G that has no cycles.

The next two results give two alternative characterizations of minimum spanning trees.

Proposition 5.4 (**Cut optimality condition**) *A spanning tree $T = (V, F)$ of a graph $G = (V, E)$, c is a minimum spanning tree w. r. t. c if and only if for every edge $ij \in F$ and every edge $k\ell$ in the cut (V_1, V_2), where V_1 and V_2 are the node sets of the two trees in $T \backslash \{ij\}$,*

$$c_{ij} \leq c_{k\ell}.$$

Proof: Suppose that T is a minimum spanning tree of G. Let ij be any edge in F, and let $k\ell$ be any edge such that nodes k and ℓ are in different trees of the subgraph $T \backslash \{ij\}$ (i.e., $k\ell$ is as in the proposition). By Theorem 5.3 the subgraph $H = (T \backslash \{ij\}) \cup \{k\ell\}$ is a spanning tree of G, so $c(H) \geq c(T)$. This implies that $c_{ij} \leq c_{k\ell}$.

For the other direction, suppose that T satisfies the cut optimality condition but T is not a minimum spanning tree of G. Let T^* be a minimum spanning tree of G that has as many edges as possible in common with T, i.e., T^* maximizes $|E(T^*) \cap E(T)|$ among all minimum spanning trees of G. Let ij be an edge of T that is not in T^* (ij exists since $T \neq T^*$). Let (V_1, V_2) be the cut formed by removing ij from T, where V_1 and V_2 are the node sets of the two trees in $T \backslash \{ij\}$, and let i be in V_1 and let j be in V_2. Consider the unique path in T^* from i to j. At least one edge $k\ell$ in this path is in the cut (V_1, V_2) (otherwise all nodes in the path would be in V_1). Since T satisfies the cut optimality condition, $c_{ij} \leq c_{k\ell}$. Hence, removing the edge $k\ell$ from T^* and adding the edge

ij gives a new subgraph $T^{**} = (T^*\backslash\{k\ell\}) \cup \{ij\}$ that has $c(T^{**}) \leq c(T^*)$. T^{**} is a spanning tree by Theorem 5.3. This contradicts our definition of T^*, since T^{**} is a minimum spanning tree of G that has one more edge in common with T than T^*. ∎

Proposition 5.5 *(***Path optimality condition***) A spanning tree $T = (V, F)$ of a graph $G = (V, E)$, c is a minimum spanning tree w. r. t. c if and only if for every edge $ij \in E\backslash F$ (ij is a nontree edge) and every edge $k\ell$ in the unique path of T between i and j*

$$c_{ij} \geq c_{k\ell} \ .$$

Proof: Suppose that T is a minimum spanning tree of G. Let ij be any edge in $E\backslash F$, and let $k\ell$ be any edge in the unique path of T between i and j. By Theorem 5.3 the subgraph $H = (T \cup \{ij\})\backslash\{k\ell\}$ is a spanning tree of G, so $c(H) \geq c(T)$. This implies that $c_{ij} \geq c_{k\ell}$.

The other direction can be proved in one of two ways:
(i) Similarly to the previous proposition, assume that T satisfies the path optimality condition but T is not a minimum spanning tree of G. Let T^* be a minimum spanning tree of G that has as many edges as possible in common with T, let ij be an edge of T^* that is not in T, and let $k\ell$ be an edge of the unique i-j path of T that is in the cut given by $T^*\backslash\{ij\}$ (i.e., $k\ell \in F$ and $k\ell \in (V_1, V_2)$, where V_1 and V_2 are the node sets of the two trees in $T^*\backslash\{ij\}$.) Then the spanning tree $(T^*\backslash\{ij\}) \cup \{k\ell\}$ gives the desired contradiction.
(ii) Alternatively, show that if T satisfies the path optimality condition, then it satisfies the cut optimality condition (we skip the details). Then the proof is completed by applying the previous proposition. ∎

Proposition 5.6 *Let $T = (V, F)$ be a minimum spanning tree of $G = (V, E)$, and let F' be a subset of F. Let $(S, V\backslash S)$ be a cut that contains no edge of F', and let ij be an edge of minimum cost in this cut. Then there exists a minimum spanning tree of G that contains all edges in $F' \cup \{ij\}$.*

Proof: If the edge ij is in F, then the proof is done. Otherwise, T contains at least one edge pq such that pq is in the cut $(S, V\backslash S)$ (since T is connected) and $pq \neq ij$. By the proposition $c_{ij} \leq c_{pq}$. Then the subgraph $H = (T\backslash\{pq\}) \cup \{ij\}$ is a spanning tree (by Theorem 5.3) and $c(H) \leq c(T)$. That is, H is a minimum spanning tree whose edge set contains $F' \cup \{ij\}$. ∎

5.4 Algorithms for minimum spanning trees

This section presents two efficient algorithms for constructing minimum spanning trees, namely, Kruskal's algorithm and Prim's algorithm. Kruskal's algorithm is also called the greedy algorithm. Given a set of objects, the greedy algorithm attempts to find a feasible subset with minimum (or maximum) objective value by repeatedly choosing an object of minimum (maximum) cost from among the unchosen ones and adding it to the current subset provided the resulting subset is feasible. In particular, Kruskal's algorithm works by repeatedly choosing an edge of minimum cost among the edges not chosen so far, and adding this edge to the "current spanning forest" provided this does not create a cycle. The algorithm terminates when the current spanning forest becomes connected.

Algorithm Kruskal's Minimum Spanning Tree Algorithm
input: Connected graph $G = (V, E)$ and edge costs $c : E \to \Re$.
output: Edge set $F \subseteq E$ of minimum spanning tree of G.

$F := \emptyset$; (F is the edge set of the current spanning forest)
linearly order the edges in E according to nondecreasing cost;
let the ordering be $e_1, e_2, \ldots, e_{|E|}$;
for each edge e_i, $i = 1, 2, \ldots, |E|$, **do**
    if $F \cup \{e_i\}$ has no cycle
    then $F := F \cup \{e_i\}$; (add the edge to the current forest)
        if $|F| = |V| - 1$ **then stop** and **output** F; **end**;
    end; (if)
end; (for)

Theorem 5.7 *Kruskal's algorithm is correct and it finds a minimum spanning tree. Its running time is $O(|V||E|)$.*

Proof: At termination of the algorithm, F is (the edge set of) a spanning tree since it has $|V| - 1$ edges and contains no cycle (see Theorem 5.2). Further, F satisfies the path optimality condition: Consider any edge ij that is not in F. At the step when the algorithm examined ij, F contained the edges of a path between i and j. Each edge kl in this path has $c_{kl} \leq c_{ij}$, since kl is examined before ij. This shows that the path optimality condition holds. Then by Proposition 5.5, F is a minimum spanning tree. ∎

Remark: By using appropriate data structures, the running time of Kruskal's algorithm can be improved to $O(|E| \log |V|)$.

Prim's algorithm starts with a "current tree" T that consists of a single node. In each iteration, a minimum-cost edge in the "boundary" $\delta(V(T))$ of T is added to T, and this is repeated till T is a spanning tree.

Algorithm Prim's Minimum Spanning Tree Algorithm
input: Connected graph $G = (V, E)$ and edge costs $c : E \to \Re$.
output: Minimum spanning tree $T = (S, F)$ of G.

$F := \emptyset$; (F is the edge set of the current tree T)
$S := \{v\}$, where v is an arbitrary node; (S is the node set of T)
while $S \neq V$ **do**
    among the edges having exactly one end in S, find an edge ij of minimum cost;
    $F := F \cup \{ij\}$;
    $S := S \cup (\{i, j\} \backslash S)$; (add the end node in $V \backslash S$)
end; (while)

Theorem 5.8 *Prim's algorithm is correct and it finds a minimum spanning tree. Its running time is* $O(|V|^2)$.

Remark: By using the Fibonacci heaps data structure, the running time of Prim's algorithm can be improved to $O(|E| + |V| \log |V|)$.

5.5 LP formulation of the minimum spanning tree problem

We start with an integer linear programming formulation, called (IP), of the minimum spanning tree problem, and then study the linear programming (LP) relaxation of (IP). Usually, when the integrality constraints of an integer program are relaxed to give a linear program, then the feasible region becomes larger, and so the optimal solution of the linear program may be considerably better than that of the integer program. Surprisingly, Theorem 5.9 below shows that an optimal solution of (IP) is also an optimal solution of the LP relaxation. In other words, the LP relaxation exactly formulates the minimum spanning tree problem.

First, introduce a zero-one variable x_{ij} for each edge ij in the given graph $G = (V, E)$. The intention is that the set of edges whose variables take on the value one, $F = \{ij \in E : x_{ij} = 1\}$, should form (the edge set of) a spanning tree. By Theorem 5.2, this can be achieved by ensuring that $|F| = \sum_{ij \in E} x_{ij} = |V| - 1$ and F contains no cycle. To impose the latter condition, we use the so-called *subtour-elimination constraints* :

$$\sum_{ij \in E: i \in S, j \in S} x_{ij} \leq |S| - 1, \qquad \forall S \subseteq V.$$

To see that these constraints "eliminate" all cycles in $F = \{ij \in E : x_{ij} = 1\}$, suppose that there is a cycle $D = v_1 \ldots v_\ell v_1$ $(v_{\ell+1} = v_1)$ such that for each edge $e = v_i v_{i+1}$, $1 \leq i \leq \ell$, in the cycle $x_e = 1$; then for the set $S = V(D) = \{v_1, \ldots, v_\ell\}$, we have $\sum_{ij \in E:\, i \in S, j \in S} x_{ij} \geq \ell = |S|$, i.e., the subtour-elimination constraint for $S = V(D)$ is violated.

The constraint $\sum_{ij \in E} x_{ij} = |V| - 1$ is implied by the first constraint in (IP) together with the subtour-elimination constraint for $S = V$.

$$\boxed{\begin{array}{llrcll}
\text{(IP)} & \text{minimize} & \sum_{ij \in E} c_{ij} x_{ij} & & & \\
& \text{subject to} & \sum_{ij \in E} x_{ij} & \geq & |V| - 1 & \\
& & \sum_{ij \in E:\, i \in S, j \in S} x_{ij} & \leq & |S| - 1, & \forall S \subseteq V \\
& & x_{ij} & \in & \{0, 1\}, & \forall ij \in E
\end{array}}$$

The LP relaxation of (IP) is obtained by replacing the integrality constraints $x_{ij} \in \{0, 1\}$ $(\forall ij \in E)$ by the constraints $x_{ij} \geq 0$ $(\forall ij \in E)$. The constraints $x_{ij} \leq 1$ need not be added explicitly, since they are implied by the subtour-elimination constraints for node sets S of size two. The LP and its dual LP (in standard form) are given below; n denotes $|V|$.

$$
\begin{array}{|llrcll|}
\hline
\text{(LP)} & \text{minimize} & \displaystyle\sum_{ij \in E} c_{ij} x_{ij} & & & \\
 & \text{subject to} & \displaystyle\sum_{ij \in E} x_{ij} & \geq & n-1 & \\
 & & \displaystyle\sum_{ij \in E:\ i \in S, j \in S} -x_{ij} & \geq & -|S|+1, & \forall S \subseteq V \\
 & & x_{ij} & \geq & 0, & \forall ij \in E \\
\hline
\end{array}
$$

$$
\begin{array}{|llrcll|}
\hline
\text{(Dual LP)} & \text{maximize} & (n-1)y_1 - \displaystyle\sum_{S \subseteq V} (|S|-1) y_S & & & \\
 & \text{subject to} & y_1 - \displaystyle\sum_{S \subseteq V:\ S \ni i, S \ni j} y_S & \leq & c_{ij}, & \forall ij \in E \\
 & & y_1 & \geq & 0 & \\
 & & y_S & \geq & 0, & \forall S \subseteq V \\
\hline
\end{array}
$$

Recall from linear programming duality that given a feasible solution $x^* = [x^*_{ij}\ (\forall ij \in E)]$ of (P) and a feasible solution $y^* = [y_1, y_S\ (\forall S \subseteq V)]$ of (D), both are optimal solutions of the respective LPs if and only if the complementary slackness conditions hold. The complementary slackness conditions are as follows:

$$
\begin{array}{lllcl}
\text{[primal]} & x_{ij} > 0 \implies & y_1 - \displaystyle\sum_{S \subseteq V:\ S \ni i, S \ni j} y_S & = & c_{ij}, \qquad \forall ij \in E \\
\text{[dual]} & \left\{ \begin{array}{l} y_1 > 0 \implies \\ \\ y_S > 0 \implies \end{array} \right. & \begin{array}{l} \displaystyle\sum_{ij \in E} x_{ij} \\ \displaystyle\sum_{ij \in E:\ i \in S, j \in S} x_{ij} \end{array} & \begin{array}{c} = \\ \\ = \end{array} & \begin{array}{l} |V|-1 \\ \\ |S|-1, \qquad \forall S \subseteq V. \end{array}
\end{array}
$$

Theorem 5.9 *Let T^* be a minimum spanning tree constructed by Kruskal's algorithm, and let x^* be the incidence vector of the edges in T^*, i.e.,*

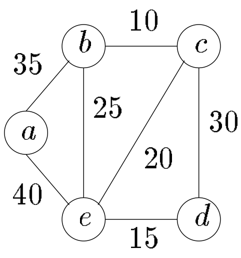
$$
x^*_{ij} = \begin{cases} 1 & \text{if } ij \in E(T^*) \\ 0 & \text{otherwise.} \end{cases}
$$

Then x^* is an optimal solution of the LP relaxation of (IP).

Proof: We construct a feasible solution y^* of the dual LP that satisfies the complementary slackness conditions with respect to x^*. It follows from linear programming duality that x^* and y^* are optimal solutions of the LP relaxation of (IP) and its dual, respectively.

The procedure for constructing y^* is called the *dual greedy algorithm*. Let m denote the number of edges, $|E|$; since G is a connected graph, $m \geq |V|-1$. Order the edges in E in nondecreasing order of the costs, and let the ordering be $e_1, e_2, \ldots, e_m$, where $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$. For each i, $1 \leq i \leq m$, let E_i denote the edge set $\{e_1, \ldots, e_i\}$. For each i, $1 \leq i \leq m$, let S_i denote the node set of the connected component that contains e_i in the subgraph (V, E_i); so $S_m = V$. In general, several indices $i, j, k, \ldots$ may have $S_i = S_j = S_k = \cdots$. Observe that the node sets S_i are explicitly "constructed" during the execution of the greedy algorithm (Kruskal's algorithm):

$G = (V, E), \quad c$

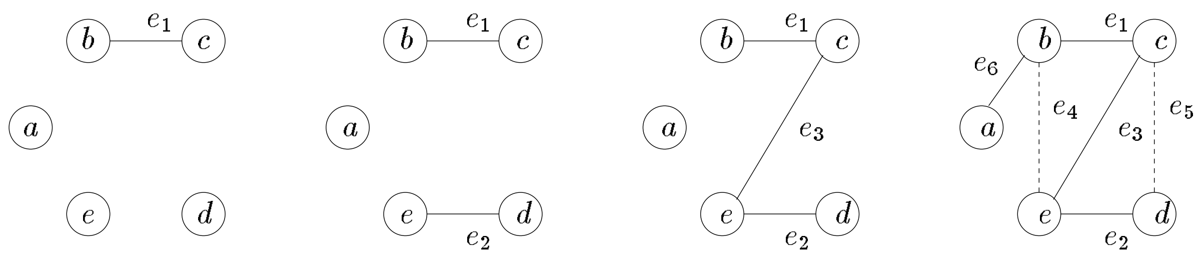


Edges ordered by cost:

$e_1, \quad e_2, \quad e_3, \quad e_4, \quad e_5, \quad e_6, \quad e_7$

$bc, \quad de, \quad ce, \quad be, \quad cd, \quad ab, \quad ae$

Execution of Kruskal's algorithm: current spanning forest (V, F)



	S_i : node sets of components of (V, F)			
	$S_1 = \{b, c\}$	$S_2 = \{d, e\}$	$S_3 = \{b, c, d, e\} = S_4 = S_5$	$S_6 = V = \{a, b, c, d, e\}$
$e(S_i)$ last edge of MST with both ends in S_i	bc	de	ce	ab
$f(S_i)$ first edge of MST with exactly one end in S_i	ce	ce	ab	—
$y_{S_i} = c(f(S_i)) - c(e(S_i))$	10	5	15	0
y_1	35, since $c(e(V)) = c(ab) = 35$ is positive			

Figure 5.0: Illustrating the LP formulation of the minimum spanning tree problem on an example. The optimal values of the dual variables y_1 and y_{S_i} are computed from the execution of Kruskal's algorithm.

these node sets are precisely the node sets of the connected components of the current spanning forest (V, F). For each node set $S \subset V$, $S \neq V$, if $S = S_i$ for some i, $1 \leq i \leq m$, then define $e(S)$ to be the edge e_{i^*} where $i^* = \min_{i:\ 1 \leq i \leq m} S_i = S$, and define $f(S)$ to be the edge e_{j^*} where $j^* = \min_{j:\ 1 \leq j \leq m} S_j \supset S, S_j \neq S$. Let $f(V)$ be undefined, and define $e(V)$ to be the edge e_{k^*} where $k^* = \min_{k:\ 1 \leq k \leq m} S_k = V$, i.e., $e(V)$ is the last edge added to the minimum spanning tree T^* by the greedy algorithm. If there is no i, $1 \leq i \leq m$, such that $S = S_i$, then $e(S)$ and $f(S)$ are undefined. Again, $e(S)$ and $f(S)$ have specific meanings in the execution of the greedy algorithm: if at some step of the execution, S is the node set of a connected component of the current spanning forest (V, F), then $e(S) = vw$ denotes the *last* edge added to the current F that has both end nodes v and w in S, and (assuming $S \neq V$) $f(S)$ denotes the *first* edge added to the minimum spanning tree T^* (i.e., the final F) with exactly one end node in S. Fix the dual solution y^* to be

$$
\begin{aligned}
y^*_S &= \begin{cases} c(f(S)) - c(e(S)) & \text{if } S = S_i \ (1 \leq i \leq m) \text{ and } S \neq V \\ -c(e(V)) & \text{if } S = V \text{ and } c(e(V)) \leq 0 \\ 0 & \text{otherwise} \end{cases} \\
y^*_1 &= \begin{cases} 0 & \text{if } c(e(V)) \leq 0 \\ c(e(V)) & \text{otherwise.} \end{cases}
\end{aligned}
$$

Clearly, y^* is nonnegative. To prove that y^* satisfies the remaining constraints of the dual LP, consider an arbitrary edge $e_i = vw$. The constraint for vw may be written as

$$\sum_{S \subseteq V:\ S \ni v, S \ni w} y^*_S \ \geq \ -c(vw) + y^*_1. \tag{$*$}$$

Focus on the left-hand side of inequality $(*)$ above. Assume that the edge $e(V)$ has $c(e(V)) \leq 0$; the other case $(c(e(V)) > 0)$ is handled similarly. Then $y^*_1 = 0$ and $y^*_V = -c(e(V))$. Define $U_1, U_2, \ldots, U_\ell$ to be the sequence of node sets

$$
\begin{aligned}
U_1 &\equiv S_i \\
U_2 &\equiv S_{i'}, \qquad \text{where } e_{i'} = f(U_1) \\
&\cdots \\
U_{j+1} &\equiv S_k, \qquad \text{where } e_k = f(U_j) \\
&\cdots \\
U_\ell &\equiv V \equiv S_{(k:\ e_k = f(U_{\ell-1}))}.
\end{aligned}
$$

In the execution of the greedy algorithm, $U_1, U_2, \ldots, U_\ell$ $(U_1 = S_i)$ are the node sets of the successive connected components of (V, F) that contain the edge $e_i = vw$; in other words, the step of the execution that adds the edge $f(U_j)$, $1 \leq j < \ell$, to the current spanning subgraph (V, F) "constructs" the node set U_{j+1} by merging U_j with the node set of another connected component of (V, F). Now observe that

$$\sum_{S \subseteq V:\ S \ni v, S \ni w} y^*_S = \sum_{j=1}^{\ell} y^*_{U_j},$$

because for every node set S, if $S \neq U_j$ (for all j, $1 \leq j \leq \ell$), then either $y^*_S = 0$ (since $S \neq S_i$, for all i, $1 \leq i \leq m$) or S does not contain both the ends v and w of the edge e_i. Inequality (*) follows because

$$\begin{aligned}\sum_{j=1}^{\ell} y^*_{U_j} &= [c(f(U_1)) - c(e(U_1))] + [c(f(U_2)) - c(e(U_2))] + \cdots + \\ &\quad [c(f(U_{\ell-1})) - c(e(U_{\ell-1}))] + [-c(e(V))] = \\ &= -c(e(U_1)) \geq -c(vw),\end{aligned}$$

since for $j = 1, \ldots, \ell-1$, $f(U_j) = e(U_{j+1})$, and $c(e(S_i)) \leq c(e_i)$.

The primal complementary slackness conditions hold because for every edge $e_i = vw$ in the minimum spanning tree T^*, $c(e(S_i)) = c(e_i)$, therefore y^* satisfies the constraint $(*)$ of the dual LP with equality.

Observe that the dual complementary slackness conditions hold, because for every node set S_i, $1 \leq i \leq m$, there are exactly $|S_i| - 1$ edges of T^* that have both ends in S_i (since for each S_i, the subgraph of T^* induced by S_i is connected); the remaining node sets S have $y^*_S = 0$. This completes the proof of the theorem. ▌

5.6 More LP formulations of the minimum spanning tree problem

We give two more linear programming relaxations of the minimum spanning tree problem that are obtained by relaxing the integrality constraints in two natural integer programming formulations. The second relaxation here, but not the first, is an exact formulation of the minimum spanning tree problem.

Our first integer programming formulation, (IP$_{cut}$), is based on Theorem 5.2(c) and Proposition 5.1: A spanning tree is a connected graph with $|V| - 1$ edges, where a graph is connected iff every node set $Q \subset V$, $\emptyset \neq Q \neq V$, has $|\delta(Q)| \geq 1$.

$$\begin{array}{llrcll}
(\mathrm{IP}_{cut}) & \text{minimize} & \displaystyle\sum_{ij \in E} c_{ij} x_{ij} & & & \\
& \text{subject to} & \displaystyle\sum_{ij \in E} x_{ij} & = & |V| - 1 & \\
& & \displaystyle\sum_{ij \in E:\, ij \in \delta(S)} x_{ij} & \geq & 1, & \forall S \subseteq V,\ \emptyset \neq S \neq V \\
& & x_{ij} & \in & \{0,1\}, & \forall ij \in E
\end{array}$$

The LP relaxation (LP$_{cut}$) of (IP$_{cut}$) is obtained by replacing the integrality constraints $x_{ij} \in \{0,1\}$ ($\forall ij \in E$) by the constraints $x_{ij} \geq 0$ ($\forall ij \in E$). In general, the optimal solution x^* of (LP$_{cut}$) may not correspond to a spanning tree, because some of the values x^*_{ij} may be fractional (not integral). In other words, the feasible region of (LP$_{cut}$) may have fractional extreme points.

Our second integer programming formulation, (IP$_{dcut}$), is a "directed" version of (IP$_{cut}$). We obtain a directed graph $D = (V, A)$ from the original graph $G = (V, E)$, by replacing each edge $ij \in E$ by the arcs (directed edges) (i,j) and (j,i); so, $A = \{(i,j),\ (j,i)\ :\ ij \in E\}$. We choose an arbitrary node $r \in V$ to be the "root node". The goal is to find a directed spanning tree $T = (V, F)$

of D rooted at r, that is, to find $F \subseteq A$ such that $|F| = |V| - 1$, and for each node $v \in V \backslash \{r\}$, F contains the arc set of a directed path from r to v. In other words, $|F| = |V| - 1$ and for every node set $S \subseteq V$ ($S \neq V$) with $r \in S$, F must contain at least one arc from the directed cut $(S, V \backslash S)$. Here, a directed cut $(S, V \backslash S)$ consists of all the arcs $(i,j) \in A$ such that $i \in S$ and $j \in V \backslash S$. Such an arc set F is necessarily the arc set of a directed spanning tree rooted at r, and its undirected version gives the edge set of a spanning tree of G.

$$
\begin{array}{llrcll}
(\mathrm{IP}_{dcut}) & \text{minimize} & \sum_{ij \in E} c_{ij} x_{ij} & & & \\
& \text{subject to} & x_{ij} & = & y_{(i,j)} + y_{(j,i)}, & \forall ij \in E \\
& & \sum_{(i,j) \in A} y_{(i,j)} & = & |V| - 1 & \\
& & \sum_{(i,j) \in A:\ (i,j) \in (S, V \backslash S)} y_{(i,j)} & \geq & 1, & \forall S \subseteq V,\ S \neq V,\ r \in S \\
& & x_{ij} & \geq & 0, & \forall ij \in E \\
& & y_{(i,j)} & \in & \{0,1\}, & \forall (i,j) \in A
\end{array}
$$

The LP relaxation (LP_{dcut}) of (IP_{dcut}), obtained by replacing the integrality constraints $y_{(i,j)} \in \{0,1\}$ ($\forall (i,j) \in A$) by the constraints $y_{(i,j)} \geq 0$ ($\forall (i,j) \in A$), gives an exact formulation of the minimum spanning tree problem.

5.7 Exercises

1. Let $G = (V, E)$ be a graph, and let the edges be partitioned into two sets R and B (i.e., the edges are colored either red or blue). Suppose that there exists a spanning tree T_R with $E(T_R) \subseteq R$ (i.e., every edge in T_R is red), and another spanning tree T_B with $E(T_B) \subseteq B$ (i.e., every edge in T_B is blue). Let k be an integer between 0 and $|V| - 1$. Use induction to show that G has a spanning tree T with $|E(T) \cap R| = k$ (i.e., the number of red edges in T is k).

2. Let G be a graph, and let each of its edges have distinct cost, i.e, for any two different edges ij and kl, $c_{ij} \neq c_{kl}$. Let r be a given node of G. Prove or disprove:

   (a) G has a unique minimum spanning tree.

   (b) G has a unique shortest paths tree with root node r.

3. (Bottleneck spanning trees.) Let $G = (V, E)$ be a graph, and let c be a cost function on the edges. A spanning tree such that the maximum cost of an edge in it is as small as possible among all spanning trees of G, c is called a *bottleneck spanning tree.*
   (a) Show that a minimum spanning tree of G, c is also a bottleneck spanning tree.
   (b) Prove or disprove the converse:
   A bottleneck spanning tree of G, c is a minimum spanning tree.

4. (Most vital edge.) Let $G = (V, E)$ be a graph, and let c be a cost function on the edges. Let $mst(G, c)$ denote the cost of a minimum spanning tree of G, c. An edge $e \in E$ is called *vital* if $mst(G', c') > mst(G, c)$, where G', c' is obtained by deleting edge e from G, c (i.e., e is vital

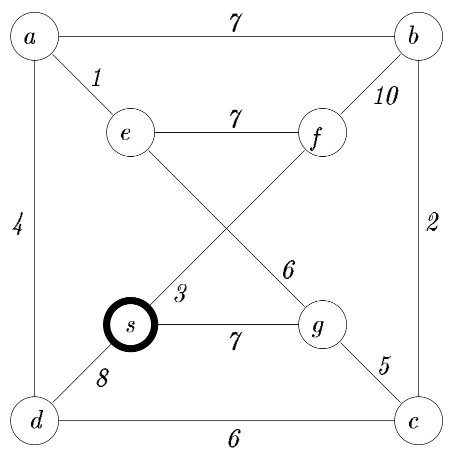


Figure 5.1: Network $G,\ c$ for Problem 5.

if its deletion strictly increases the cost of a minimum spanning tree). A *most vital edge* is a vital edge whose deletion increases the cost of the minimum spanning tree by the maximum amount. Does every network $G,\ c$ contain a vital edge? Suppose that a network contains a vital edge. Describe an efficient algorithm for finding a most vital edge.
(Hint: Use the cut optimality conditions.)

5. Consider the graph and edge costs in Figure 5.1.

   (a) Find a minimum spanning tree using Kruskal's algorithm.

   (b) Find a minimum spanning tree using Prim's algorithm.

   (c) Find a shortest paths tree with root node s using Dijkstra's algorithm.

6. (a) For the graph $G = (V, E)$ and edge costs c given in Figure 5.2, formulate the problem of finding a spanning tree of maximum cost as a linear program.

   (b) Write the dual LP and the complementary slackness conditions for the example in part (a).

   (c) For $G,\ c$ and the LP you wrote in part (a), use the dual greedy method to write down an optimal solution to the dual LP. Verify the complementary slackness conditions.

   (d) Generalize your solution of the previous part to an arbitrary graph G, and arbitrary edge costs c.

   (e) Solve the LP in part (a) using the CPLEX optimizer.

7. (Sensitivity analysis.) Let $G = (V, E)$ be a graph, and let c be a cost function on the edges. Let T^* be a minimum spanning tree of $G,\ c$. For an edge $ij \in E$, define its *cost interval* to be the set of all real numbers γ such that if the cost c_{ij} is changed to γ (but all other edge costs stay the same), then T^* continues to be a minimum spanning tree w.r.t. the new costs.
(a) Describe an efficient method for determining the cost interval of a given edge ij.

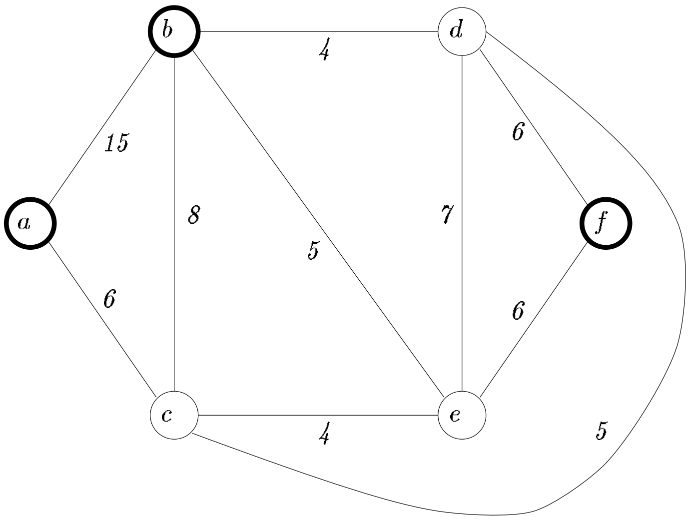


Figure 5.2: Network $G,\ c$ for Problem 6.

(Hint: Consider two cases: when $ij \in T^*$ and when $ij \notin T^*$, and use the cut and path optimality conditions.)
(b) Determine the cost interval of every edge for $G,\ c$ in Figure 5.2.

8. Construct an example to show that the feasible region of the linear programming relaxation (LP_{cut}) (page 73) may have fractional extreme points.
   Prove that the feasible region of (LP_{cut}) contains the feasible region of the linear programming relaxation (LP) in Section 5.5, by showing that every feasible solution $x : E \to \Re_+$ of (LP) satisfies all the constraints of (LP_{cut}).

9. Prove that the linear programming relaxation (LP_{dcut}) of (IP_{dcut}) (page 74) gives an exact formulation of the minimum spanning tree problem.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, Englewood Cliffs, N. J., 1993.

[2] B. Bollobas, *Graph Theory: An Introductory Course*. Springer-Verlag, New York, 1979.

[3] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*. Macmillan, London, 1976.

[4] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1992.

[5] M. Grötschel, L. Lovász and A. Schrijver, *Geometric algorithms and combinatorial optimization*, (1988) Springer-Verlag, Berlin.

[6] R. C. Larson and A. R. Odoni, *Urban Operations Research*. Prentice-Hall, Englewood Cliffs, N. J., 1981.

[7] T. L. Magnanti and L. A. Wolsey, *Optimal Trees*. CORE discussion paper 9426, C. O. R. E., Universite Catholique de Louvain, Louvain-la-neuve, Belgium, May 1994.

Chapter 6

Light Approximate Shortest Paths Trees

This chapter is based on a paper by Khuller, Raghavachari and Young [4]. Awerbuch, Baratz and Peleg have related results on so-called shallow light trees [3].

There are some other recent papers on related topics. Althofer et al [1] study spanners of weighted graphs. Mansour and Peleg [5], Salman et al [6], and Awerbuch and Azar [2] give approximation algorithms for related problems in network design where the objective function is nonlinear.

6.1 Introduction

Given a graph $G = (V, E)$, nonnegative costs $c(e)$ on the edges $e \in E$, and a specified root node s, can we always find a spanning tree that has both approximately minimum cost and that approximately preserves the cost of shortest s-v paths for all nodes v? Does a shortest paths tree (with root s) or a minimum spanning tree always achieve these goals? Unfortunately, the answer to the second question is no – demonstrating this is left as an exercise for the reader.

For any tree T of G, recall that $c(T)$ denotes the cost of the tree $c(T) = \sum_{vw \in T} c(vw)$. For any node v, let $d(v)$ denote the cost of a shortest s-v path. Let T_s denote a shortest paths tree of G with root s, and let T_M denote a minimum spanning tree of G. Let α and β be numbers ≥ 1. A spanning tree T of G is called an (α, β) light approximate shortest-paths tree (LAST) if

1. for all nodes $v \in V$, the cost of the unique s-v path in T is at most $\alpha \cdot d(v)$, and
2. $c(T) \leq \beta \cdot c(T^*_M)$, where T_M is a minimum spanning tree of G.

6.2 The preorder traversal of a tree

Before developing the LAST algorithm, we need to define a preorder traversal of a tree and to prove a lemma on preorder traversals. Let T be a tree, and take s to be the root node. Consider any node v. A node w adjacent to v in T is called the *parent* of v if w lies on the unique v-s path in

T, otherwise, w is called a *child* of v. Every node except s has one parent (s has no parent), and has zero or more children. For a node $v \in V$, the subtree of T with root v is the subgraph of T induced by all nodes x such that v lies in the unique x-s path of T.

Let the number of nodes in T be n. A *preorder numbering* (or depth-first-search numbering) of T assigns the numbers $1, \ldots, n$ to the nodes of T. More precisely, a preorder numbering is a bijection from $V(T)$ to $\{1, 2, 3, \ldots, n\}$ such that the root s is assigned the number 1, and for each child v_j of s, the subtree T_j with root v_j is assigned a preorder number using the numbers

$$2 + \sum_{i=1}^{j-1} |V(T_i)|, \ \ldots, \ 1 + \sum_{i=1}^{j} |V(T_i)|.$$

The following algorithm computes a preorder numbering of T using the recursive procedure PREORDER.

Algorithm Preorder Numbering of a Tree
input: Tree T and root node s.
output: Preorder numbering num(v), $\forall v \in V(T)$.

$i := 1$; (initialize global variable)
PREORDER(s); (call recursive procedure to do the numbering)
for each node v **do**
  output num(v) **end**;

procedure PREORDER(node v);
num(v) $:= i$;
$i := i + 1$;
for each child w of v **do**
  PREORDER(w); (call recursive procedure)
end; (for)
end; (procedure)

The next lemma is used in the analysis of the LAST algorithm. For any two nodes v and w in a tree T, let $d_T(v, w)$ denote the cost of the unique path in T between v and w.

Lemma 6.1 *Let T be a spanning tree with root s. Let $z_0, z_1, \ldots, z_k$ be any k nodes of T, arranged according to a preorder numbering of T. Then*

$$\sum_{i=1}^{k} d_T(z_{1-1}, z_i) \quad \leq \quad 2\, c(T).$$

Proof: Focus on the walk W formed by "doubling" every edge of T and making a preorder traversal of T; if we draw T on the plane such that each edge is drawn as a thin strip, then W

corresponds to the boundary of the drawing of T. Since each edge occurs exactly two times in W, the cost of W, $c(W)$, equals $2\,c(T) = 2 \sum_{e \in E(T)} c(e)$ (recall that the cost of a walk $W = v_0 v_1 v_2 \ldots v_\ell$ is $\sum_{i=1}^{\ell} c(v_{i-1}, v_i)$).

Note that the first occurrence of node $z_i (0 \leq i < k)$ on W precedes the first occurrence of node z_{i+1} on W, because in the preorder numbering z_0 precedes z_1 precedes ... precedes z_k. Hence, the edges of W can be partitioned into edge-disjoint sequences $W(z_i, z_{i+1})$ $(0 \leq i < k)$ and $W(z_k, z_0)$. The lemma follows since for each of these subwalks $W(z_i, z_{i+1})$, the cost, $c(W(z_i, z_{i+1}))$ is $\geq d_T(z_i, z_{i+1})$. ∎

6.3 An algorithm for finding a light approximate shortest-paths tree

This section presents the LAST algorithm, and proves that for any given numbers α, $1 < \alpha$, and β, $\beta \geq 1 + \frac{2}{\alpha - 1}$, the algorithm correctly finds an (α, β) LAST. Let T_M denote a minimum spanning tree of the given network. Step 3 of the following algorithm constructs a spanning subgraph $H = (V, E')$ of G that contains T_M such that for each node v, the cost of a shortest s-v path in H is at most $\alpha \cdot d(v)$. Theorem 6.2 below proves that the cost of H, $\sum_{vw \in E'} c(vw)$, is at most $\beta \cdot c(T_M)$.

Algorithm (α, β) Light Approximate Shortest-paths Tree
input: Graph $G = (V, E)$, root node s, nonnegative edge costs c,
  numbers α and β such that $\alpha > 1$, $\beta \geq 1 + \frac{2}{\alpha-1}$.
output: Spanning tree T^* of G which is an (α, β) LAST.

STEP 1: find a minimum spanning tree T_M of G, c;
  find a shortest-paths tree T_S of G, c with start node s;
STEP 2: find a preorder numbering of T_M using s as the start node;
STEP 3: $H := T_M$;
  for each node v in the preorder sequence of T_M **do**
    find a shortest s-v path P in H;
    if $c(P) > \alpha \cdot d(v)$
    then add all the edges in a shortest s-v path in G to H;
    end; (if)
  end; (for)
STEP 4: find a shortest-paths tree T^* of H with start node s;
  output T^*;

Theorem 6.2 *Let $\alpha > 1$ and $\beta = 1 + \frac{2}{\alpha-1}$ be two numbers. In the subgraph H,*

(i) for each node v, the cost of a shortest s-v path is at most $\alpha \cdot d(v)$, and

(ii) the total cost $c(H)$ is at most $\beta \cdot c(T_M)$.

Proof: Step 3 of the algorithm ensures that part (i) holds. We prove part (ii). Let $z_0 = s$, and let $z_1, z_2, \ldots, z_k$ be the preorder sequence of vertices that caused the edges of shortest paths to be added to H in Step 3. When z_i $(i = 1, \ldots, k)$ is examined in Step 3, H contains the s-z_i walk formed by taking the last path P_{i-1} (from s to z_{i-1}) added to H followed by the path in T_M from z_{i-1} to z_i, and the cost of this walk is

$$d(z_{i-1}) + d_{T_M}(z_{i-1}, z_i)$$

Since the shortest s-z_i path P_i in H has at most this cost, and since $c(P_i) > \alpha \cdot d(z_i)$ (from Step 3), we have

$$d(z_{i-1}) + d_{T_M}(z_{i-1}, z_i) > \alpha \cdot d(z_i)$$

or

$$\alpha \cdot d(z_i) - d(z_{i-1}) < d_{T_M}(z_{i-1}, z_i).$$

Summing over all $i = 1, \ldots, k$, and noting that $d(z_0) = 0$, we have

$$\begin{array}{lcr} \alpha \cdot d(z_1) - d(z_0) & < & d_{T_M}(z_0, z_1) \\ \alpha \cdot d(z_2) - d(z_1) & < & d_{T_M}(z_1, z_2) \\ \alpha \cdot d(z_k) - d(z_{k-1}) & < & d_{T_M}(z_{k-1}, z_k) \\ \hline \alpha \cdot d(z_k) + \sum_{i=1}^{k-1}(\alpha - 1)d(z_i) & < & \sum_{i=1}^{k} d_{T_M}(z_{i-1}, z_i), \end{array}$$

or $\displaystyle\sum_{i=1}^{k}(\alpha - 1)d(z_i) < \sum_{i=1}^{k} d_{T_M}(z_{i-1}, z_i)$. From the previous lemma, the right-hand side is at most $2\ c(T_M)$, hence, $\displaystyle\sum_{i=1}^{k} d(z_i) < \frac{2\ c(T_M)}{\alpha - 1}$. The proof is complete since $c(H) = c(T_M) + \displaystyle\sum_{i=1}^{k} d(z_i) < \left(1 + \frac{2}{\alpha - 1}\right) c(T_M)$. ∎

6.4 Exercises

1. Construct a family of networks G, c (one for each $|V(G)|$) and a root node $s \in V(G)$ such that given numbers α and β as in the algorithm, and any positive number ϵ, the cost of the spanning tree output by the (α, β) LAST algorithm is at least

$$(1 + \frac{2}{\alpha - 1} - \epsilon)c(T_M),$$

where $c(T_M)$ is the cost of a minimum spanning tree of G, c.

2. A breadth-first-search (BFS) ordering of a tree with root s is obtained by doing a BFS traversal of the tree, and numbering the nodes in the order in which they are visited. Alternatively, it is the numbering obtained by setting the number of s to 1, then ordering the nodes in a "level" of the tree according to the number of their parents, and then numbering the nodes

in sequence. Prove or disprove:
Suppose that Step 3 of the (α, β) LAST algorithm traverses the minimum spanning tree T_M in BFS order (instead of using preorder), then the subgraph H has cost $c(H)$ at most $(1 + \frac{2}{\alpha-1})c(T_M)$.

Bibliography

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph and J. Soares, *On sparse spanners of weighted graphs.* Discrete and Computational Geometry, (1992).

[2] B. Awerbuch, and Y. Azar, *Buy-at-bulk network design approximation algorithms.* Proc. 38th Annual IEEE Sympos. on Foundations of Comput. Sci. (FOCS) 1997.

[3] B. Awerbuch, A. Baratz, and D. Peleg, *Cost-sensitive analysis of communication protocols.* Proc. 9th Sympos. on Distributed Computing (PODC), (1990), pp. 177–187.

[4] S. Khuller, B. Raghavachari and N. E. Young, *Balancing minimum spanning and shortest path trees.* Algorithmica **14** (1995), pp. 305–322.

[5] Y. Mansour and D. Peleg, *An approximation algorithm for minimum-cost network design.* The Weizman Institute of Science, Rehovot, 76100 Israel, Tech. Report CS94-22, 10 pages.

[6] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian, *Buy-at-bulk network design: approximating the single-sink edge installation problem.* Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-1997).

Chapter 7

Approximation Algorithms for Steiner Trees

7.1 The problem and its complexity

Let $G = (V, E)$ be an undirected graph, let each edge $ij \in E$ have a nonnegative real-valued cost, and let N be a set of terminal nodes, $N \subseteq V$. A tree T of G (that is, the node set $V(T)$ is a subset of V and the edge set $E(T)$ is a subset of E) is called a *Steiner tree* if it contains all the terminal nodes, $V(T) \supseteq N$. Nonterminal nodes, nodes in $V \backslash N$, may be present in T; nodes in $V(T) \backslash N$ are called *Steiner nodes*. The problem is to find a Steiner tree T whose cost $c(T) = \sum_{ij \in E(T)} c(ij)$ is minimum among all Steiner trees; an optimal solution T is called a *Steiner minimal tree* (SMT for short). See Figure 7.1 for an example. The problem, even with unit costs on the edges, is NP-hard, so an efficient algorithm (i.e., one that runs in time polynomial in $|V|$) for finding an optimal solution would give efficient algorithms for solving all NP-complete problems. So it is likely that any algorithm ever designed for finding Steiner minimal trees will be inefficient.

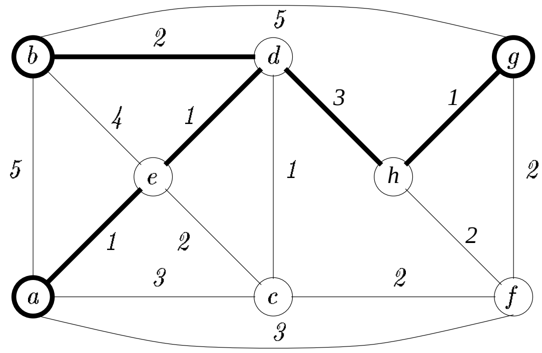


Figure 7.1: An example of the Steiner tree problem; the terminal node set is $N = \{a, b, g\}$. The terminal nodes (nodes in N) are circled in bold. A Steiner minimal tree is indicated by thick lines; it has cost 8.

Nevertheless, the problem has many practical applications such as routing in VLSI layout, the design of communication networks, accounting and billing for the use of telephone networks, the phylogeny problem in biology, etc. Moreover, it has generated interesting mathematical questions and conjectures, from the days of Fermat (1601–1665) to the present.

Two special cases of the Steiner tree problem are solvable by efficient algorithms: If all nodes are terminals, that is, if $N = V$, then we have the minimum spanning tree problem, and an optimal solution can be found using, e.g., the greedy algorithm (Kruskal's algorithm). If there are exactly two terminal nodes, say, $N = \{s, t\}$, then we have the shortest s-t path problem, and an optimal solution can be found using Dijkstra's algorithm.

Our main focus in this chapter is on heuristics for the Steiner tree problem and on deriving their performance guarantees. First we present a simple heuristic called the distance network heuristic with performance ratio 2 and an efficient variant due to Mehlhorn. Then we present Zelikovsky's recent algorithm with a better performance ratio of $\frac{11}{6}$.

7.2 Distance network heuristics for Steiner trees

7.2.1 Introduction

An instance of the Steiner tree problem consists of a graph $G = (V, E)$, a nonnegative real-valued cost $c : E \to \Re_+$ on the edges, and a set N of terminal nodes. Let $T_G(N)$ and $c(T_G(N))$ denote a Steiner minimal tree and its cost, respectively. For any two nodes v and w of G, let $d_G(v, w)$ denote the cost of a shortest path between v and w in G, i.e., $d_G(v, w) = \min\ \sum_{i=1}^{\ell} c(v_{i-1}v_i)$, where the minimization is over all paths $P = v_0v_1 \ldots v_\ell$ in G from $v = v_0$ to $w = v_\ell$. For a node v and a set of nodes S, $d_G(S, v)$ denotes $\min\limits_{w \in S} d_G(w, v)$, i.e., the cost of a shortest path from any node in S to v. Given any subset V' of the nodes of G, the distance network $D_G(V')$ is defined to be a graph whose node set is V' and such that for every pair of nodes $v \in V'$, $w \in V'$, the edge vw is present; the cost of an edge vw, denoted $c_D(vw)$, is defined to be $d_G(v, w)$. In other words, $D_G(V')$ is the complete graph on the node set V' together with the edge costs $c_D(vw) = d_G(v, w)$ (for all edges vw). For every edge vw in G, observe that $d_G(v, w) \le c(vw)$. This gives us the following.

Fact: For every edge vw of the distance network $D_G(V')$, if vw is an edge of G, then $c_D(vw) \le c(vw)$.

A real-valued nonnegative function on pairs of nodes $\ell : V \times V \to \Re_+$ is said to satisfy the triangle inequality if for every three nodes v, w and x, $\ell(v, w) \le \ell(v, x) + \ell(x, w)$. Clearly, the shortest paths costs $d : V \times V \to \Re_+$ satisfy the triangle inequality, and this gives the following.

Fact: In the distance network $D_G(V')$, the edge costs c_D satisfy the triangle inequality.

The next few results consider the distance network $D = D_G(V)$, i.e., all nodes of G are present in D.

Proposition 7.1 *Let N be the set of terminal nodes, let $T_G(N)$ be a Steiner minimal tree of G with terminal node set N, and let $T_D(N)$ be a Steiner minimal tree of the distance network $D = D_G(V)$ with terminal node set N. Then the cost of $T_D(N)$ equals the cost of $T_G(N)$.*

Proof: We show first that $c_D(T_D(N)) \leq c(T_G(N))$, and then that $c(T_G(N)) \leq c_D(T_D(N))$. For every edge vw of $T_G(N)$, note that it is present in D and that its cost in D, $c_D(vw)$, is at most $c(vw)$. Hence, we obtain the first inequality. To prove the second inequality, replace each edge vw of $T_D(N)$ by a shortest v-w path of G, and call the resulting graph T'. Clearly, T' is a connected subgraph of G that contains all nodes in N, and moreover its cost $c(T') = \sum_{e \in E(T')} c(e)$ equals $c_D(T_D(N))$. Let T'' be a spanning tree of T'. Then T'' is a Steiner tree of G for the terminal node set N, and its cost $c(T'')$ is at most $c(T') = c_D(T_D(N))$. ∎

Lemma 7.2 *Let N be a set of terminal nodes. There exists a Steiner minimal tree $T_D(N)$ in the distance network $D = D_G(V)$ such that each Steiner node has degree at least 3.*

Proof: If $T_D(N)$ has any Steiner nodes having degree one, then these may be removed; the resulting tree is still a Steiner tree for the terminal set N. So suppose that every Steiner node of $T_D(N)$ has degree ≥ 2. Consider any Steiner node v of degree two, and let u and w be the nodes adjacent to v in $T_D(N)$. Clearly, the distance network D contains all three edges uw, uv and vw, and by the fact above, the edge costs satisfy the triangle inequality:

$$c_D(uw) \leq c_D(uv) + c_D(vw).$$

By replacing the node v and its two incident edges in $T_D(N)$ by the edge uw, we obtain a Steiner tree for N whose cost is at most $c_D(T_D(N))$. We repeat this procedure until there are no Steiner nodes of degree less than three. ∎

Proposition 7.3 *Let N be a set of terminal nodes. There exists a Steiner minimal tree $T_D(N)$ in the distance network $D = D_G(V)$ such that the number of Steiner nodes is at most $|N| - 2$.*

Proof: Let s be the number of Steiner nodes in $T = T_D(N)$. By the previous lemma, we may assume that each Steiner node has degree at least 3 in T. Hence, the sum of the degrees of all nodes in T is at least $|N| + 3s$. Also, the sum of the node degrees equals $2|E(T)| = 2|V(T)| - 2 = 2|N| + 2s - 2$. Since $|N| + 3s \leq 2|N| + 2s - 2$, we see that $s \leq |N| - 2$. ∎

7.2.2 The basic distance network heuristic

We construct the distance network $D_G(N)$, where N is a specified set of terminal nodes, and then find a minimum spanning tree M of $D_G(N)$. By replacing each edge vw of M by a shortest v-w path in G we obtain a new subgraph T' of G. Finally, we find a minimum spanning tree of T', repeatedly remove any Steiner nodes of degree one, and output the resulting tree T_{DNH}. Clearly, T_{DNH} contains all the terminal nodes. See Figure 7.2 for a worked example.

Algorithm Distance Network Heuristic for Steiner Minimal Trees
input: Graph $G = (V, E)$, nonnegative edge costs c, terminal node set N.
output: Tree T_{DNH} that contains all nodes in N; $c(T_{DHN}) \leq (2 - \frac{2}{|N|})c(T_G(N))$.

STEP 1: construct the distance network $D_G(N)$ for N, G and c;
(for each pair of terminal nodes v, w, edge vw has cost $c_D(vw) = d_G(v, w)$)
STEP 2: find a minimum spanning tree M of $D_G(N)$ (w.r.t. costs c_D);
STEP 3: replace each edge in M by a corresponding shortest path in G, c
to get a subgraph T' of G;
STEP 4: find a minimum spanning tree (call it T_{DNH}) for the subgraph of G, c
induced by the nodes in T';
STEP 5: repeatedly delete from T_{DNH} nonterminal nodes of degree one;
output the resulting tree T_{DNH};

Theorem 7.4 *For every graph G, cost function $c : E(G) \to \Re_+$, and terminal node set N, the distance network heuristic finds a Steiner tree T_{DNH} such that $c(T_{DNH}) \leq (2 - \frac{2}{|N|})c(T_G(N))$.*

Proof: Let L be a walk on $T_G(N)$ that uses each edge exactly twice and that visits every node of $T_G(N)$. In other words, L is the "boundary" in a drawing of $T_G(N)$ on the plane. The cost of L is $c(L) = 2\ c(T_G(N))$. Order the terminal nodes according to their first occurrence on L, assuming that L starts with a terminal node z_1, and let the terminal nodes in the ordering be $z_1, z_2, \ldots, z_k$, where we use k for $|N|$ and we take z_{k+1} to be z_1. Let $L(z_i, z_{i+1})$, $1 \leq i \leq k$, denote the subwalk of L between the first occurrence of z_i and the first occurrence of z_{i+1}; note that $L(z_i, z_{i+1})$ may contain several other terminal nodes z_j, $j < i$. By choosing the start node z_1 appropriately, we can ensure that the subwalk $L(z_k, z_{k+1}) = L(z_k, z_1)$ has cost at least $\frac{c(L)}{k} = \frac{2\ c(T_G(N))}{k}$. We remove the edges of this subwalk from L, and call the resulting walk L'. Clearly, L' has cost $c(L') \leq \left(2 - \frac{2}{k}\right) c(T_G(N))$. The proof of the theorem follows from the next result.

Fact: The cost of T_{DNH} is at most $c(L')$.
Proof: To prove the claim, consider any subwalk $L(z_i, z_{i+1})$, $1 \leq i \leq k-1$, of L' and observe that the edge $z_i z_{i+1}$ in the distance network $D_G(N)$ has cost $c_D(z_i z_{i+1}) = d_G(z_i, z_{i+1}) \leq c(L(z_i, z_{i+1}))$, since the minimum cost of a path in G from z_i to z_{i+1} is at most the cost of $L(z_i, z_{i+1})$. Hence, the union of the edges $z_i z_{i+1}$, $1 \leq i \leq k-1$, of $D_G(N)$ gives a spanning tree T of $D_G(N)$ whose cost $c_D(T)$ is at most $\sum_{i=1}^{k-1} c(L(z_i, z_{i+1})) = c(L')$. Since the cost of the Steiner tree T_{DNH} found by the distance network heuristic is at most the cost of a minimum spanning tree of $D_G(N)$, it follows that $c(T_{DNH}) \leq c_D(T) \leq c(L')$. This proves the claim. ∎ ∎

Proposition 7.5 *The time complexity of the distance network heuristic is $O(|N|(|E|+|V|\cdot\log|V|))$.*

Proof: In the first step, in order to construct $D_G(N)$, we have to compute shortest paths costs starting from each terminal node $z \in N$. This needs $|N|$ applications of Dijkstra's shortest-paths algorithm. Each application of Dijkstra's algorithm contributes $O(|E| + |V| \cdot \log|V|)$ to the running time. The above time bound suffices for the remaining steps. ∎

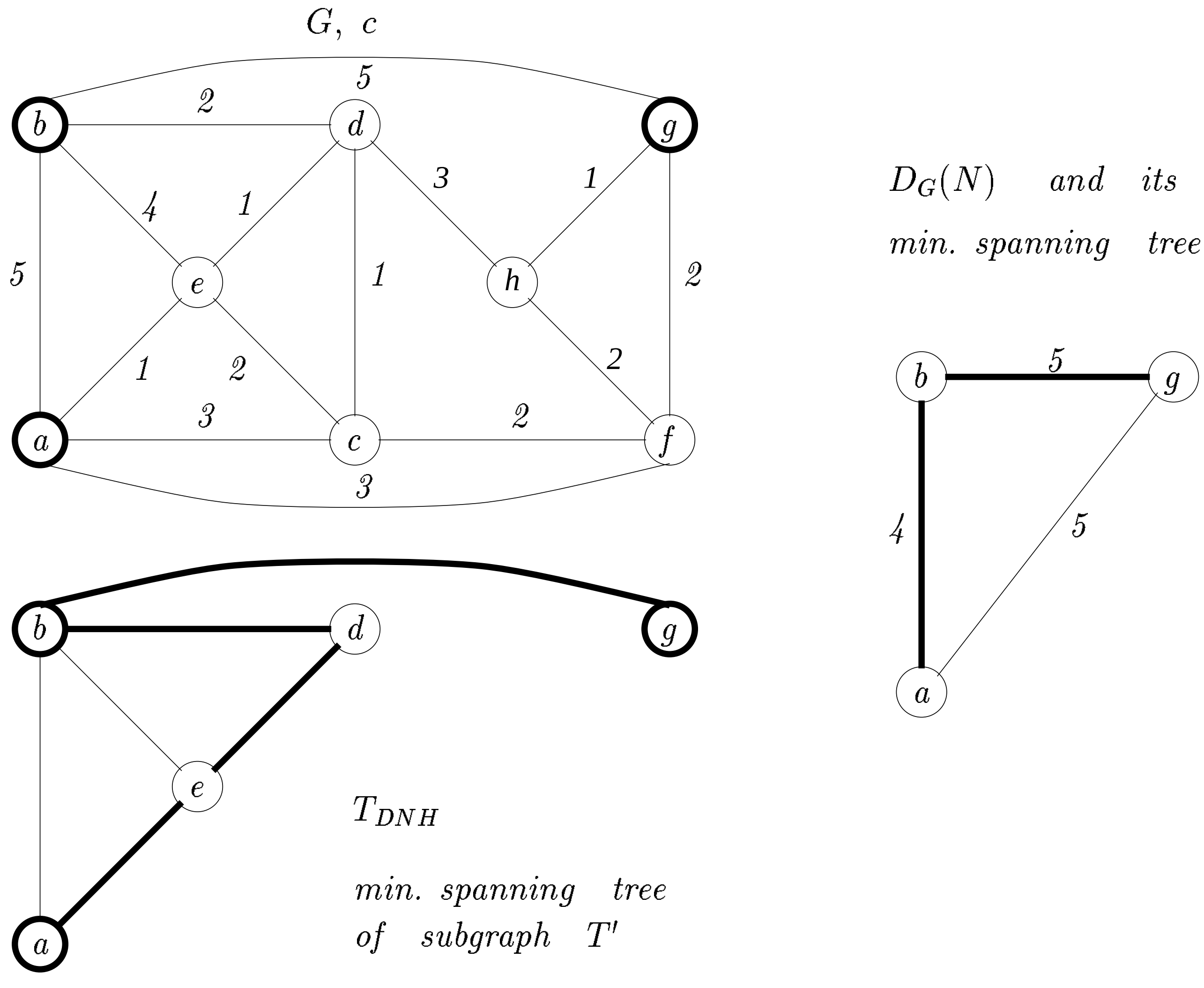


Figure 7.2: The example of the Steiner tree problem in Figure 7.1 is solved using the distance network heuristic. The terminal node set is $N = \{a, b, g\}$. The networks $G,\ c$ and $D_G(N),\ c_D$ are shown, along with subgraph T' and T_{DNH}. Note that T_{DNH} is not an SMT.

7.2.3 Mehlhorn's variant of the distance network heuristic

Mehlhorn [5] modifies the first step of the distance network heuristic so that it computes a graph $D' = (N, E')$ and arc costs $c' : E' \to \Re_+$ instead of the distance network $D_G(N)$. The rest of Mehlhorn's variant (after the construction of D' and c') is the same as in the distance network heuristic. It turns out that D' and c' can be computed by *one* application of Dijkstra's shortest-paths algorithm, hence the overall time complexity for finding T_{DNH} becomes $(|E| + |V| \cdot \log |V|)$. The performance guarantee of Mehlhorn's variant, $\dfrac{c(T_{DNH})}{c(T_G(N))}$, is the same as in the distance network heuristic, namely, $\left(2 - \dfrac{2}{|N|}\right)$.

For each terminal node $z_i \in N$, define $N(z_i)$ to be the set containing z_i and all nonterminal nodes $v \in V \backslash N$ that are "closer" to z_i than to any other terminal node z_j, $z_j \neq z_i$; if a nonterminal node v is "closest" to two or more terminal nodes $z_i, z_j, \ldots$, i.e., if $d_G(v, z_i) = d_G(v, z_j) = \cdots = \min_{z \in N} d_G(v, z)$, then v is included in the set of the terminal node having the smallest index among $i, j, \ldots$. In symbols,

$$N(z_i) \quad = \quad \{z_i\} \cup \left\{ \begin{array}{ll} v \in V \backslash N \ : & d_G(v, z_i) = \min_{z \in N} d_G(v, z) \ \text{ and} \\ & i = \min\{j = 1, \ldots, |N| \ : \ d_G(v, z_j) = \min_{z \in N} d_G(v, z)\} \end{array} \right\}.$$

Thus, the sets $N(z_i)$, $z_i \in N$, form a partition of the node set V. For every pair of terminal nodes z_i, z_j, if G has an edge between a node in $N(z_i)$ and a node in $N(z_j)$, then the edge $z_i z_j$ is present in D', i.e.,

$$E' = \{z_i z_j, \ z_i \in N, \ z_j \in N \ : \ \exists v \in N(z_i), \ \exists w \in N(z_j) \ : \ vw \in E(G)\}.$$

Informally, an edge of D' may "represent" several different edges of G. The cost of an edge $z_i z_j$ in E' is taken to be the minimum cost of a path from z_i to z_j in G that uses exactly one of the edges that "represent" $z_i z_j$. The path may contain several edges that either have both end nodes in $N(z_i)$ or have both end nodes in $N(z_j)$. That is, $c' : E \to \Re_+$ is defined by

$$c'(z_i z_j) = \min_{vw \in E(G), \ v \in N(z_i), \ w \in N(z_j)} \{d_G(z_i, v) + c(vw) + d_G(w, z_j)\}.$$

D' is a subgraph of the distance network $D_G(N)$, and it may be a proper subgraph of $D_G(N)$, i.e., some edges of $D_G(N)$ may be missing from D'. Moreover, even if an edge $z_i z_j$ of $D_G(N)$ is present in D', its cost in D', $c'(z_i z_j)$, may be greater than its cost in $D_G(N)$, $c_D(z_i z_j) = d_G(z_i, z_j)$. Figure 7.3 shows D' and $D_G(N)$ for an example.

D' can be easily constructed using a single application of Dijkstra's shortest-paths algorithm. We add a new start node v_0 to G, and for each terminal node $z_i \in N$, we add the edge $v_0 z_i$ to G and fix the edge cost to be zero, $c(v_0 z_i) = 0$. We assume that none of the original edges of G has zero cost. Then we take the start node to be v_0 and find a shortest-paths tree T_S with root v_0. Removing v_0 and its incident edges from T_S gives a collection of subtrees, each subtree containing exactly one terminal node z_i and zero or more nonterminal nodes (here, we use the assumption on the costs of the original edges of G). For each terminal node z_i, $N(z_i)$ is the set of nodes in the subtree containing z_i. The shortest-paths costs $d(v)$, $v \in V$, computed by Dijkstra's algorithm

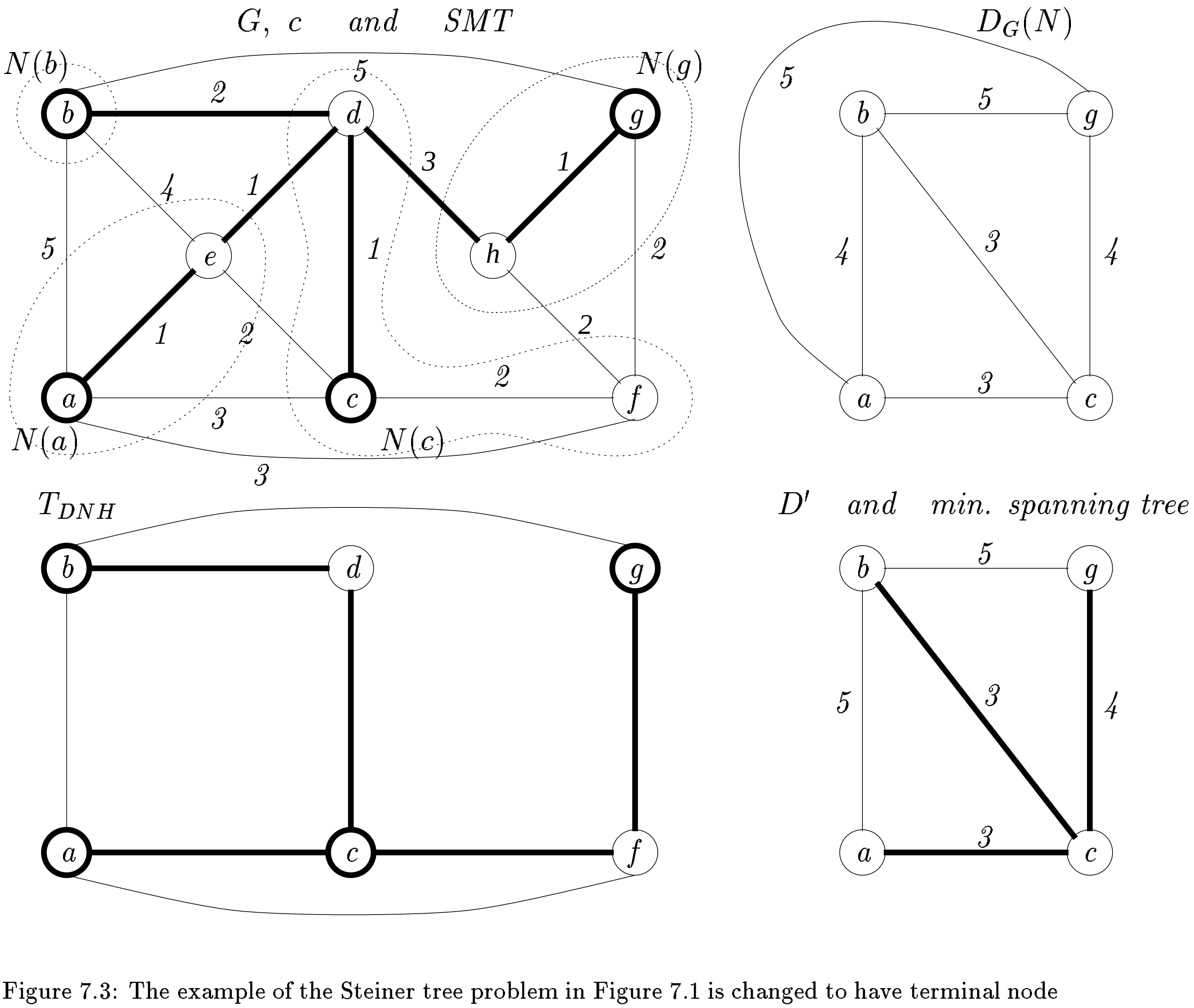


Figure 7.3: The example of the Steiner tree problem in Figure 7.1 is changed to have terminal node set $N = \{a, b, c, g\}$, and then solved using Mehlhorn's variant of the distance network heuristic. The network G, c is shown, along with an SMT and the node sets $N(z)$, $z \in N$. The networks $D_G(N)$, c_D and D', c' are shown for comparison, though Mehlhorn's variant uses only D', c'. Note that T_{DNH} is not an SMT.

are exactly the values $d_G(v, z_i)$, $z_i \in N$ and $v \in N(z_i)$, that we need to compute $c' : E' \to \Re_+$. By examining each edge vw of G such that v and w are in different sets $N(z_i)$ and $N(z_j)$, it is straightforward to compute the edge costs c' for D'. The next lemma, which is the key result on D', shows that every minimum spanning tree of D', c' is a minimum spanning tree of $D_G(N)$, c_D. Based on this lemma, we compute a minimum spanning tree M' of D', c'. Then we replace each edge vw of M' by a shortest path from v to w in G, c to obtain a subgraph T'. T_{DNH} can be obtained from T', by using the last two steps of the basic distance network heuristic. Figure 7.3 shows a worked example.

Theorem 7.6 *Mehlhorn's variant of the distance network heuristic computes T_{DNH} in time $O(|E| + |V| \cdot \log |V|)$.*

Lemma 7.7 *Let M' be a minimum spanning tree of D' with respect to the edge costs c'. Then M' is a minimum spanning tree of $D_G(N)$ with respect to the edge costs c_D.*

Proof: We claim that there exists a minimum spanning tree M^* of $D_G(N)$ such that each edge z_iz_j of M^* is present in D', and the cost of the edge in D', $c'(z_iz_j)$, equals the cost in $D_G(N)$, $c_D(z_iz_j)$. Suppose that the claim is true. Then M^* is obviously a minimum spanning tree of D' w.r.t. c', since any spanning tree of D' has cost at least $c_D(M^*)$. By definition, any other minimum spanning tree M' of D' w.r.t. c' has exactly the same cost as M^*, i.e., $c'(M') = c'(M^*) = c_D(M^*)$, therefore M' must be a minimum spanning tree of $D_G(N)$ w.r.t. c_D.

The claim is proved by contradiction. Suppose that the claim is false. Let M be a minimum spanning tree of $D_G(N)$ that

(i) has the maximum number of edges in D', i.e., $|E(M) \cap E'|$ is maximized over all minimum spanning trees of $D_G(N)$, c_D, and moreover,

(ii) the total cost with respect to c' of the edges in both M and D' is minimum, i.e., $\sum_{e \in E(M) \cap E'} c'(e)$ is minimized over all minimum spanning trees of $D_G(N)$, c_D satisfying (i).

Either an edge z_sz_t of M is not present in D', or there is an edge z_sz_t of M with $c'(z_sz_t) > c_D(z_sz_t) = d_G(z_s, z_t)$. Let P be a shortest path from z_s to z_t in G, with respect to the costs c. Consider any edge v^*w^* of P such that v^* and w^* are in distinct sets, say, $v^* \in N(z_i)$ and $w^* \in N(z_j)$. By the definition of $N(z_i)$, we get $d_G(z_i, v^*) \le d_G(z_s, v^*)$, and by the definition of $N(z_j)$, $d_G(z_j, w^*) \le d_G(z_t, w^*)$. Clearly, D' contains the edge z_iz_j. The cost of this edge w.r.t. c' is

$$\begin{aligned}
c'(z_iz_j) &= \min_{vw \in E(G),\ v \in N(z_i),\ w \in N(z_j)} \{d_G(z_i, v) + c(vw) + d_G(w, z_j)\} \\
&\le d_G(z_i, v^*) + c(v^*w^*) + d_G(w^*, z_j) \\
&\le d_G(z_s, v^*) + c(v^*w^*) + d_G(w^*, z_t) \\
&= c(P) \\
&= d_G(z_s, z_t) \\
&= c_D(z_sz_t).
\end{aligned}$$

That is, for every one of the edges v^*w^* of P with v^* and w^* in distinct sets $N(z_i)$ and $N(z_j)$, the edge z_iz_j is present in D', and $c'(z_iz_j) \leq c_D(z_sz_t)$. These edges form a walk P' from z_s to z_t in D' since P is a z_s-z_t path in G: indeed, if we start with the subgraph P and for each $z_i \in N$ we "contract" all nodes in $N(z_i)$ into a single node z_i', then we obtain P'.

If we remove edge z_sz_t from M, then we obtain a partition N_s, N_t of the nodes of $D_G(N)$, where N_s is the set of nodes in the subtree of $M\backslash\{z_sz_t\}$ containing z_s, and $N_t = N\backslash N_s$. Since P' is a walk from z_s to z_t, at least one of its edges $z_i^*z_j^*$ has one end node in N_s and the other in N_t, i.e., $z_i^*z_j^*$ is in the cut (N_s, N_t). By replacing the edge z_sz_t in M by the edge $z_i^*z_j^*$, we obtain another spanning tree $\overline{M}$ of $D_G(N)$. $\overline{M}$ is a minimum spanning tree of $D_G(N)$ because its cost $c_D(\overline{M}) = c_D(M) - c_D(z_sz_t) + c_D(z_i^*z_j^*)$ is at most $c_D(M)$, since $c_D(z_i^*z_j^*) \leq c'(z_i^*z_j^*) \leq c_D(z_sz_t)$ by the derivation above. Now, by our assumption on M, either edge z_sz_t is not in D' or z_sz_t is in D' but $c'(z_sz_t) > c_D(z_sz_t)$. In the first case, $\overline{M}$ has more edges in D' than M, and in the second case

$$\sum_{e\in E(\overline{M})\cap E'} c'(e) < \sum_{e\in E(M)\cap E'} c'(e) \qquad \text{since} \quad c'(z_i^*z_j^*) \leq c_D(z_sz_t) < c'(z_sz_t).$$

In either case, we have a contradiction to the definition of M, and so our claim is proved. The lemma follows. ∎

7.3 The Dreyfus-Wagner dynamic programming algorithm

The Dreyfus-Wagner algorithm (1972) is one of the oldest exact algorithms for finding a Steiner minimal tree. Its time complexity is exponential in $|N|$. The algorithm is based on dynamic programming. It uses a recursion to derive the cost of a Steiner minimal tree from the costs of Steiner minimal trees for subsets of the given terminal node set. For a node set $D \subseteq V$ and a node $v \in V$, let $S(v, D)$ denote the cost of a Steiner minimal tree for the terminal node set $D \cup \{v\}$. The recursion is:

$$S(v, D) = \min_{w\in V}\left\{d_G(v,w) + \min_{\emptyset\neq D'\subset D}\left(S(w,D') + S(w, D\backslash D')\right)\right\}.$$

Note that D' is a proper subset of D in the recursion. Possibly, $v = w$ in the recursion, in which case $d_G(v, w) = 0$. Informally, in a Steiner minimal tree for $D \cup \{v\}$, node w is a "junction node", i.e., a terminal node or a nonterminal node of degree at least 3, such that all internal nodes in the path from v to w are nonterminal nodes of degree 2. The three main cases of the recursion are shown in Figure 7.4.

Let N be the given set of terminal nodes, and let q be an arbitrary node in N. Initially, the algorithm computes $d_G(v, w)$ for all pairs of nodes $v,\ w \in V$; note that $S(v, \{w\}) = d_G(v, w)$. Then for all integers ℓ between 2 and $|N| - 2$, in sequence, for all $D \subseteq N\backslash\{q\}$ such that $|D| = \ell$ and for all $v \in V$, $S(v, D)$ is computed. Finally, $S(q, N\backslash\{q\})$ is computed, and this gives the cost of a Steiner minimal tree.

Theorem 7.8 *The Dreyfus-Wagner algorithm computes the cost of a Steiner minimal tree. The running time of the algorithm is* $O(3^{|N|}(|V|) + 2^{|N|}(|V|^2) + |V|^3)$.

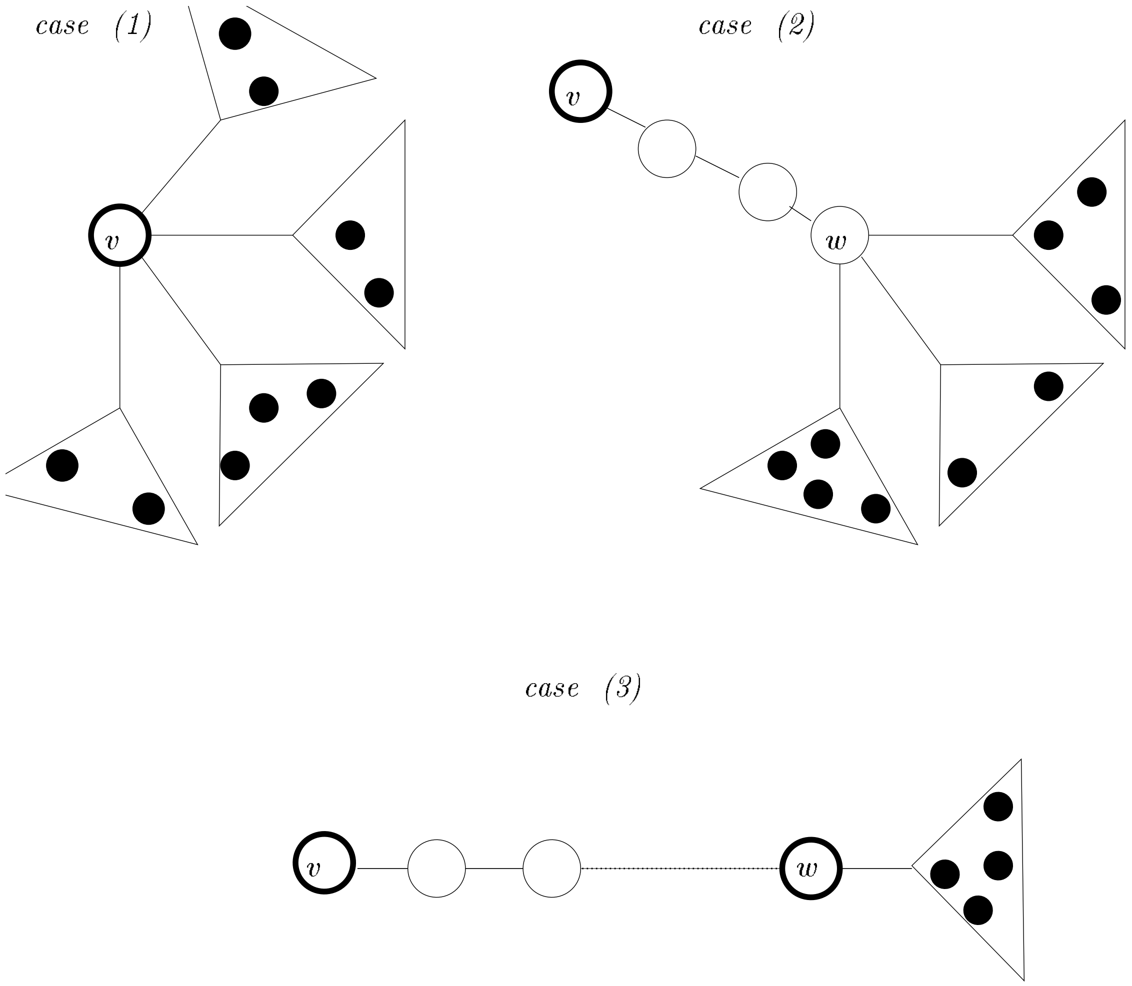


Figure 7.4: An illustration of the three main cases in the Dreyfus-Wagner recursion. The node v and the terminal nodes are indicated by either bold circles or shaded circles.

If $|N|$ is small, say, $|N| = O(\log\ |V|)$, then note that the algorithm runs in polynomial time.

The algorithm has to be supplemented in order to find a Steiner minimal tree, rather than the cost of an SMT. This is done by backtracing, that is, by examining the sets $D \subseteq N$ in the reverse order of the cost-determining computation, assuming that the values $S(v, D)$, $v \in V$, $D \subset N$, have been stored for later use. We start with $D = N \backslash \{q\}$, and find the "junction node" w and the partition $D = D' \cup D''$, $D' \cap D'' = \emptyset$, that determine the value of $S(q, D)$. Then we compute the "junction node" and the optimal partition for each of D' and D'' by recomputing $S(w, D')$ and $S(w, D'')$. Continuing this process, we find the Steiner minimal tree.

Algorithm Dreyfus-Wagner Dynamic Programming Algorithm for Steiner Minimal Trees.
input: Graph $G = (V, E)$, nonnegative edge costs c, terminal node set N.
output: Cost of a Steiner minimal tree.

STEP 0: **for** every pair of nodes $v, w \in V$, **do**
    compute $d_G(v, w)$, the minimum cost of a v-w path;
  end; (for)
  fix q to be an arbitrary node in N;
STEP 1: (singleton sets)
  for each $t \in N \backslash \{q\}$ **do**
    for each $v \in V$ **do** $S(v, \{t\}) := d_G(v, t)$; **end**;
  end; (for)
STEP 2: (subsets of size $2, 3, \ldots, |N| - 2$)
  for $\ell := 2$ to $|N| - 2$ **do**
    for each $D \subset N \backslash \{q\}$ such that $|D| = \ell$ **do**
      $S(v, D) := \infty$, for all $v \in V$; (initialize)
      choose an arbitrary node $z \in D$;
      for each $w \in V$ **do**
        $s := \infty$; (temporary variable)
        for each $D' \subset D$ with $z \in D'$ **do**
          $s := \min(s,\ S(w, D') + S(w, D \backslash D'))$;
        end; (for)
        for each $v \in V$ **do**
          $S(v, D) := \min(S(v, D),\ d_G(v, w) + s)$;
        end; (for)
      end; (for)
    end; (for)
  end; (for)
STEP 3: (find final solution for $N \backslash \{q\}$)
  $C := \infty$; (initialize)
  choose an arbitrary node $z \in N \backslash \{q\}$;
  for each $w \in V$ **do**
    $s := \infty$; (temporary variable)
    for each $D' \subset N \backslash \{q\}$ with $z \in D'$ **do**
      $s := \min(s,\ S(w, D') + S(w, (N \backslash \{q\}) \backslash D'))$;
    end; (for)
    $C := \min(C,\ d_G(q, w) + s)$;
  end; (for)
  output C;

7.4 Zelikovsky's Algorithm

In this section, we present an algorithm with improved performance ratio for the Steiner problem due to Zelikovsky [7]. This version of the proof appeared in two lectures in January 1997 at MPI, Saarbruecken, Germany, by Ulrich Fössmeyer; The following treatment is from a manuscript by Kurt Mehlhorn, in which Naveen Garg filled in some details. The proof of a key lemma is incomplete however. The following description may be useful in presenting a different point of view of looking at Zelikovsky's algorithm.

7.4.1 Definitions

Recalle that we used $T_G(N)$ to denote the optimal Steiner tree in G for the terminals N. In the sequel, whenever we consider optimal Steiner trees for subsets S of nodes, we always do so in the input graph G so for ease of notation we drop the subscript and assume that we always compute the optimal tree in G. Thus we shall henceforth use $T(S)$ to denote an optimal Steiner tree for the terminal set S in G.

7.4.2 The algorithm

The starting point of Zelikovsky's algorithm is the 2-approximate solution from the distance network heuristic. The algorithm maintains a complete graph on the terminal nodes. The edge costs of this graph decrease as the algorithm progresses through several phases. In particular, in each phase, the cost of exactly two edges decreases. The approximate solution maintained by the algorithm is the minimum cost spanning tree of this graph.

Suppose there are f phases in the algorithm. Let G_i represent the graph maintained by the algorithm just before the i^{th} phase. Thus, in our notation, $G_0 = D_G(N)$, the complete graph on node set N with costs denoting shortest distances in G. Let M_i denote the minimum cost spanning tree of G_i. The performance guarantee of the distance network heuristic guarantees that $c(T_0) < 2c(T(S))$.

Before we describe the details of each phase, we need a few more definitions. For terminals x and y we use $Bridge_i(x, y)$ to denote the heaviest edge on the path from x to y in M_i. For a triple $Tr = \{x, y, z\}$ of distinct vertices let

$$Bridge_i(Tr) = \{Bridge_i(x, y), Bridge_i(x, z), Bridge_i(y, z)\}.$$

Proposition 7.9 $|Bridge_i(T)| = 2$.

Proof: Let M_{Tr} be the subtree of M_i spanned by Tr. The tree M_{Tr} contains a unique point c such that the tree-paths p_x, p_y, and p_z from c to x, y, and z respectively are pairwise edge-disjoint[1]. Assume without loss of generality that the heaviest edge of M_{Tr} lies on p_x then $Bridge_i(x, y) = Bridge_i(x, z) \neq Bridge_i(y, z)$. ∎

For ease of notation we assume in the sequel that $Bridge_i(Tr)$ consists of $Bridge_i(x, y)$ and $Bridge_i(x, z)$. In this case, we also say that the triple Tr is x-light with respect to the tree M_i.

[1] c may be equal to one of x or y or z in which case the corresponding path has length zero.

Note that this is equivalent to saying that the heaviest edge in M_{Tr} defined in the proof above lies on the path p_x. Note that $c(Bridge_i(Tr)) = c(Bridge_i(x,y)) + c(Bridge_i(x,z))$. For a triple Tr let

$$gain_i(T) = c(Bridge_i(T)) - c(T(Tr)).$$

Zelikovsky's algorithm terminates when there is no triple of positive gain. If there is a triple of positive gain in the i-th phase let $Tr_i = \{x, y, z\}$ be the triple with maximum gain. We obtain G_{i+1} by adding edges xy and xz to G_i with costs $c(Bridge_i(x,y)) - gain_i(Tr_i)$ and $c(Bridge_i(x,z)) - gain_i(Tr_i)$ respectively. The minimum spanning tree M_{i+1} of G_{i+1} is clearly $M_i - Bridge_i(Tr_i) \cup \{xy, xz\}$. Also, $c(M_{i+1}) = c(M_i) - 2 \cdot gain_i(Tr_i)$, i.e., the cost of the minimum spanning tree decreases by twice the gain. This implies that

$$c(M_f) = c(M_0) - 2 \cdot \sum_i gain_i(Tr_i). \tag{7.1}$$

7.4.3 Performance Guarantee

To analyze the performance guarantee we first observe some important properties of the spanning tree M_i during the course of the algorithm. We do this next.

In each phase the algorithm replaces two edges of the current spanning tree by new ones. It is helpful to view these replacements as taking place consecutively. So, the M_i's evolve by selecting pairs $\{a, b\}$ of terminals and replacing $Bridge_i(a,b)$ by ab. The new edge has lower cost than the old edge, the replaced edge had the largest cost on the cycle closed by the new edge, and the new edge separates exactly the same terminals as the old edge, i.e., for any terminals x and y, the path from x to y went through $Bridge_i(a,b)$ before the replacement if it goes through ab after the replacement.

This allows us to argue that for any two terminals x and y the cost of their bridge never increases.

Lemma 7.10 *For any two terminals x and y and for any $i \in \{0, 1, \ldots, f-1\}$, the following is true: $c(Bridge_i(x,y)) \geq c(Bridge_{i+1}(x,y))$.*

Proof: When moving from M_i to M_{i+1} we replace two edges. A replacement substitutes an edge ab for the edge $Bridge_i(a,b)$. If ab does not lie on the path from x to y then $Bridge_i(x,y)$ does not change. Otherwise, the edge ab closes a cycle $p \circ q$ in M_i and paths p_x and p_y connect x and y to the common endpoints of p and q. In M_i the path from x to y is $p_x \circ q \circ p_y$ and in M_{i+1} the path from x to y is $p_x \circ p \circ p_y$. Since $Bridge_i(a,b)$ is the heaviest edge on the cycle $p \circ q$ the claim follows. ∎

We can now show that every edge whose cost is decreased and hence is included in the minimum spanning tree is retained in the final tree M_f.

Lemma 7.11 *For all j, $Bridge_j(Tr_j)$ consists only of edges of G_0.*

Proof: The proof of this lemma is at the crux of the analysis but is however beyond the scope of these notes. A short version of the proof may be found in the Appendix of [3]. ∎

The Steiner tree finally output when the algorithm terminates is the shortest paths in G corresponding to edges of G_0 retained in M_f, as well as the optimal trees for all the triples used to decrease edge costs in the course of the algorithm. It is now straightforward to bound the cost of this tree.

Lemma 7.12 *The Steiner tree output by Zelikovsky's algorithm has cost at most* $(c(M_0)+c(M_f))/2$.

Proof: Each edge of M_f is either original or new. Lemma 7.11 tells us that once a new edge is added to some M_i it will stay in the minimum spanning tree of the auxiliary graph till the end of the algorithm. In other words, M_f consists of some original edges and the new edges introduced by the triples, $Tr_0, Tr_1, \ldots, Tr_{f-1}$. Construct a subgraph G' of G consisting of the paths corresponding to the edges of $M_f \cap G_0$ and the Steiner minimal trees $T(Tr_0), \ldots, T(Tr_{f-1}))$ and let $\tilde{T}$ be a minimum spanning tree of G'. The cost of $\tilde{T}$ is at most $c(M_f)+\sum_i(c(T(Tr_i))-(c(Bridge_i(Tr_i))-2\cdot gain_i(Tr_i)))=c(M_f)+\sum_i gain_i(Tr_i)$ since any triple Tr_i adds a Steiner tree of cost $c(T(Tr_i)($ to G' in replacement of edges of cost $c(Bridge_i(Tr_i))-2\cdot gain_i(Tr_i)$ of M_f. Hence,

$$\begin{aligned} c(\tilde{T}) &\leq c(M_f)+\sum_i gain_i(Tr_i) \\ &= (c(M_f)+c(M_f)+2\cdot\sum_i gain_i(Tr_i))/2 \\ &= (c(M_f)+c(M_0))/2 \end{aligned}$$

since $c(M_f)=c(M_0)-2\cdot\sum_i gain_i(Tr_i)$ from equation 7.1. ∎

We recall the main result from the previous section.

Lemma 7.13 $c(M_0)<2\cdot c(T(N))$.

To bound the cost of the tree $\tilde{T}$ output by Zelikovsky's algorithm, we still need to bound $c(M_f)$ independent of $C(M_0)$. We shall prove the following result in the next section.

Lemma 7.14 $c(M_f)\leq\frac{5}{3}\cdot c(T(N))$.

With this, we can prove the performance guarantee of Zelikovsky's algorithm.

Theorem 7.15 *Zelikovsky's algorithm finds a Steiner tree* $\tilde{T}$ *with cost at most* $\frac{11}{6}$ *times the minimum.*

Proof: From lemmas 7.12, 7.13 and 7.14, we get

$$\begin{aligned} c(\tilde{T}) &\leq (c(M_0)+c(M_f))/2 \\ &\leq (2+5/3)\cdot c(T(N))/2 \\ &= (11/6)\cdot c(T(N)). \end{aligned}$$

∎

7.4.4 Full Steiner Trees

A Steiner tree for terminal set S is said to be *full* if all nodes in S are leaves of this tree. The *full components* of any Steiner tree for S are its maximal full subtrees. Finally, a Steiner tree of S is said to be *k-restricted* if all its full components have at most k leaves. We shall denote an optimal (minimum cost) k-restricted Steiner tree for S by $T^k(S)$. Note that $T(S) = T^{|S|}(S)$.

We prove Lemma 7.14 in this section by proving the following two results.

Lemma 7.16 $c(M_f) \leq c(T^3(N))$.

Lemma 7.17 $c(T^3(N)) \leq \frac{5}{3} \cdot c(T(N))$.

Note that Zelikovsky's algorithm and the analysis via the above two lemmas can be used to derive even better performance guarantees for the case of special metrics such as rectilinear or ℓ_1 metrics. See [1] for more on this development.

Next we demonstrate that M_f is not costlier than an optimal 3-restricted Steiner tree by examining the natural 3-restricted tree defined by it on the terminals.

Proof: of Lemma 7.16 Let M_f be the minimum spanning tree of the final auxiliary graph. There are no triples of positive gain with respect to M_f. Consider an optimal 3-restricted Steiner tree $T^3(N)$. Each full component of $T^3(N)$ has either two or three leaves. Let Tr be the triples of terminals induced by the full components with three leaves and let Pr be the pairs of terminals induced by the full components with two leaves. Then

$$c(T^3(N)) = \sum_{tr \in Tr} c(T(tr)) + \sum_{pr \in Pr} c(T(pr))$$

We need to show that $c(M_f) \leq c(T^3(N))$. To this end, we define a tree M_a of cost at most $c(T^3(N))$ on N and then show that M_f is a minimum spanning tree of the graph $M_f \cup M_a$. The edges of M_a are defined as follows: For each pair $p = xy \in Pr$ we put the edge xy with cost $c(T(pr))$ into M_a and for each triple $tr = \{x, y, z\}$ we put the edges xy and xz into M_a with costs $c(Bridge_f(x, y))$ and $c(Bridge_f(x, z))$, respectively[2]. Since $gain_f(tr) \leq 0$ we have $c(Bridge_f(tr)) \leq c(T(tr))$. Thus, $c(M_a) \leq c(T^3(N))$. The graph M_a is clearly connected and since the number of edges of M_a is $|Pr| + 2|Tr| = |N| - 1$, M_a is a tree.

It remains to show that M_f is a minimum spanning tree of $M_f \cup M_a$. This follows from the cycle rule for minimum spanning trees. Assume that we start with M_f and add the edges of M_a one by one. Let xy be an edge of M_a. If xy comes from a pair p then the cost $c(T(pr))$ of the edge is at equal to the cost of the edge xy in G_0. Similarly, if xy comes from a triple then it is clearly a heaviest edge in the cycle which it closes. ∎

Finally we bound on the cost of an optimal 3-restricted tree with respect to the optimum solution[3].

Proof: of Lemma 7.17

Recall that $T(N)$ denotes an optimal Steiner tree for the terminals N. We may assume w.l.o.g. that $T(N)$ is full. Otherwise, we apply the following argument to each full component.

[2] Recall that we adopted the convention that $Bridge(T)$ consists of $Bridge(x, y)$ and $Bridge(x, z)$.

[3] The following proof is due to Vijay Vazirani transmitted via Naveen Garg.

We also assume for the moment that $T(N)$ is binary. If not, we replace any internal node of out-degree larger than 2 by a chain of nodes of degree 3, give all new edges cost zero, and apply the argument to the resulting tree. If there are nodes of degree 2 in the original tree, we may shorcut over such nodes by using the triangle inequality on the costs (w.l.o.g.).

Root $T(N)$ at an arbitrary internal vertex, add an edge to the root, and three-color the edges of $T(N)$. We define a 3-restricted Steiner tree for each color class of edges and show that the total cost of the three trees is at most $5c(T(N))$. The lemma follows by averaging.

We may assume w.l.o.g. that $T(N)$ is ordered such that for every vertex v the path to the rightmost leaf $\alpha(v)$ in its subtree is the shortest path to any leaf in v's subtree. Note that this definition is consistent with all the internal nodes: if the internal node u has l as its closest leaf and the path from u to l uses the child a of u, then l is also a closest leaf to a. We define the 3-restricted Steiner tree for a color class by specifying the triples of nodes that are to form full subtrees.

Consider any of the color classes and let $(u, |parent|(u))$ be an edge of the color class.

- If u has children a, b and a sibling c then include the optimal Steiner tree for the triple $\{\alpha(a), \alpha(b), \alpha(c)\}$ into the Steiner tree for the color class and call $\{a, b, c\}$ the core of the triple.

- If u has children a, b but no sibling, i.e., u is the root, add the edge $\{\alpha(a), \alpha(b)\}$ into the Steiner tree for the color class and call $\{a, b\}$ the core of the pair.

- If u has a sibling c but no children, i.e., u is a terminal, add the edge $\{u, \alpha(c)\}$ into the Steiner tree for the color class and call $\{u, c\}$ the core of the pair.

We claim that all the edges added for a color class define a Steiner tree for N, which, by construction, is 3-restricted. This is readily proved by induction. Consider the edges of any color class bottom up and argue that after considering an edge $(u, parent(u))$ all terminals in the subtree rooted at $parent(u)$ are connected by the edges added so far for this color class.

It remains to show that the total cost of the three trees is at most $5 \cdot (T(N))$. Note first that the cost of each tree is the sum of the costs of the minimum Steiner trees of the pairs and triples defining the tree. Note next that the cost of the minimum Steiner tree for a pair of triple is bounded by the cost of the subtree of $T(N)$ spanned by the pair or triple. The cost of the subtree spanned by a pair or triple has two components:

- the cost of subtree spanned by the elements in the core of the pair or triple.

- the cost of the paths connecting v to $\alpha(v)$ for v in the core of the triple or pair and

We bound the total cost of the two contributions separately.

Consider first the subtrees spanned by the cores of pairs and triples. They have the shapes shown in Figure 7.5.

Each edge of T(N) belongs to exactly three such trees, namely once as the edge inducing the pair or triple, once as the sibling of the inducing edge, and once as the child of the inducing edge. Thus, the sum of the costs of the subtrees spanned by the cores is exactly $3 \cdot (T(N))$.

It remains to sum the costs of the paths from v to $\alpha(v)$ over all v over all cores. We use S to denote this sum. Clearly, we need to consider only v's that are not terminals (leaves). Each such

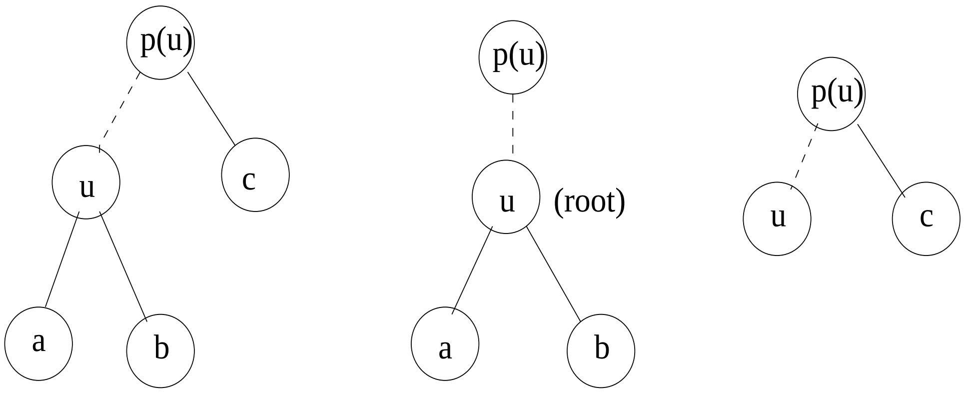


Figure 7.5: Shapes of subtrees spanned by cores of pairs and triples.

v is contained in the boundary of at most two subtrees spanned by cores, namely the core induced by the edge connecting the sibling of v with the parent of v and the core induced by the edge connecting the parent of v with the grand-parent of v. We conclude that S is bounded by twice the cost of the paths from v to $\alpha(v)$ summed over all v. Let $\beta(v)$ be the leaf reached from v by going right once and then always left. The path $v \longrightarrow \beta(v)$ is at least as long as the path $v \longrightarrow \alpha(v)$ and the former paths are disjoint. Thus, the sum of their lengths is bounded by $c(T(N))$ completing the proof. ▌

Further improvements on Zelikovsky's ideas are investigated by Berman and Ramaiyer[2].

7.5 Exercises

1. Consider the Steiner tree problem. Suppose that the set S^* of Steiner nodes in an optimal solution is available. How would you find a Steiner minimal tree?

2. Let N be a set of points in the Euclidean plane. Take the cost of an edge ij to be the Euclidean distance between points i and j.

   (a) Let T^* be a Steiner minimal tree in the Euclidean plane, and let x be a Steiner point in T^*. Explain whether the number of edges of T^* incident to x can be either (i) one? or (ii) two?

   (b) Let $|MST(N)|$ denote the cost of a minimum spanning tree of N, and let $|SMT(N)|$ denote the cost of a Steiner minimal tree. Prove that $|MST(N)| \leq (2 - \frac{2}{N})\ |SMT(N)|$.

3. Consider the following example of the set covering problem consisting of 10 points $x_1, \ \ldots, \ x_{10}$,

and 7 sets $S_1, \ldots, S_7$. The incidence matrix is

$$A = \begin{array}{c|ccccccc|c} & S_1 & S_2 & S_3 & S_4 & S_5 & S_6 & S_7 & \\ \hline & & & & & 1 & & 1 & x_1 \\ & & & & 1 & & 1 & & x_2 \\ & & & 1 & & 1 & & 1 & x_3 \\ & & 1 & & 1 & & 1 & 1 & x_4 \\ & 1 & & 1 & & 1 & 1 & & x_5 \\ & 1 & 1 & & 1 & & & & x_6 \\ & 1 & 1 & 1 & 1 & & & & x_7 \\ & 1 & 1 & 1 & & 1 & & & x_8 \\ & 1 & 1 & 1 & & & 1 & & x_9 \\ & 1 & 1 & 1 & & & & 1 & x_{10} \\ \hline \end{array}$$

(a) Convert the set covering example into an example of the Steiner tree problem. Use the shortest paths heuristic to find a solution T_{SPH} to the Steiner tree problem. What is the theoretical guarantee g on the ratio $\frac{c(T_{SPH})}{c(T_{OPT})}$, where T_{OPT} is a Steiner minimal tree?

(b) Use your solution in part (a) to find a solution to the set covering example. Explain your working.

(c) An optimal solution of the dual (D) of the LP relaxation of the integer programming formulation (IP) of the set covering problem is

$$\begin{array}{lllll} y_1 = 1 & y_2 = \frac{1}{2} & y_3 = 0 & y_4 = 0 & y_5 = 0 \\ y_6 = 0 & y_7 = \frac{1}{2} & y_8 = 0 & y_9 = \frac{1}{2} & y_{10} = 0, \end{array}$$

where y_i is the dual variable for point x_i. Discuss whether your solution in part (b) is optimal, and explain your reasoning.

(d) Consider solving the set covering problem (in general), using the algorithm in parts (a) and (b). Is it true that the set cover J_{algo} found by this algorithm satisfies $|J_{algo}| \leq g \cdot |J_{opt}|$, where J_{opt} is an optimal solution, and g is as in part (a)? Explain your answer.

4. Solve (approximately) the Steiner tree problem defined by the graph $G = (V, E)$ and the edge costs c in Figure 5.2 taking the terminal node set to be $N = \{a, b, f\}$. Use each of the following algorithms:

   (a) the shortest paths heuristic,

   (b) Mehlhorn's variant of the distance network heuristic, and

   (c) Dreyfus and Wagner's dynamic programming algorithm.

5. Repeat parts (a)–(c) of the previous problem for the example of the Steiner tree problem in Figure 7.6; the terminal node set is $N = \{a, c, e, g\}$.

6. For a set of terminal nodes N, let $\ell(N)$ denote the minimum number of leaf nodes in a Steiner minimal tree of N. Clearly, $2 \leq \ell(N) \leq |N|$. Show that the performance guarantee of the distance network heuristic can be improved to

$$\frac{c(T_{DNH})}{c(T_G(N))} \leq \left(2 - \frac{2}{\ell(N)}\right).$$

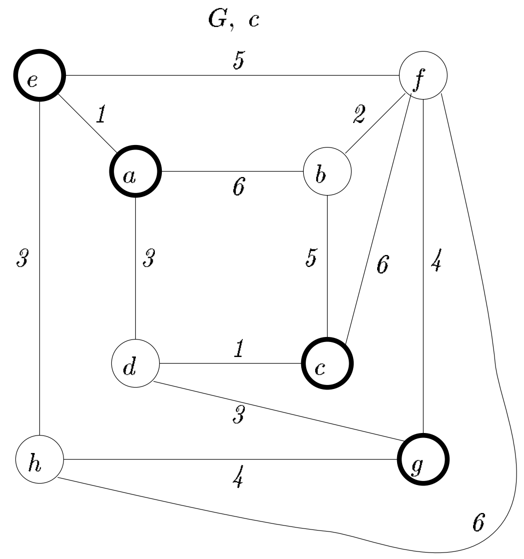


Figure 7.6: An example of the Steiner tree problem.

7. Give a detailed implementation of Mehlhorn's distance network heuristic, focusing on Steps 1, 3 and 5 of the basic distance network heuristic (page 86). Show that Step 4 (finding a minimum spanning tree of the (node-induced) subgraph T' of G) can be avoided by appropriately implementing Step 3 (transforming minimum spanning tree M of $D_G(N)$ into subgraph T' of G) using information from Step 1 (constructing $D_G(N)$).

8. The goal here is to study an integer programming formulation (IP) of the Steiner tree problem that is similar to the integer programming formulation of the minimum spanning tree problem in Chapter 5. Let $G = (V, E)$ be the input graph, let N be the set of terminal nodes, and let the cost of edge ij be denoted by c_{ij} $\quad (c_{ij} \geq 0,\ \forall_{ij} \in E)$. The integer program (IP) has a zero-one variable z_i for each node $i \in V \backslash N$. The intention is that an optimal solution of (IP) has $z_i^* = 1$ iff node i is a Steiner node in the corresponding Steiner minimal tree.

$$\begin{array}{lllll}
\text{(IP)} \quad \text{minimize} & \displaystyle\sum_{ij\in E} c_{ij}x_{ij} & & & \\
\quad\text{subject to} & \displaystyle\sum_{ij\in E} x_{ij} & \geq & (\displaystyle\sum_{i\in V\backslash N} z_i) + |N| - 1 & \\
& \displaystyle\sum_{ij\in E:\ i\in S, j\in S} x_{ij} & \leq & (\displaystyle\sum_{i\in S\backslash N} z_i) + |S\cap N| - 1, & \forall S \subseteq V \\
& x_{ij} & \in & \{0,1\}, & \forall ij \in E \\
& z_i & \in & \{0,1\}, & \forall i \in V\backslash N
\end{array}$$

(a) Consider the Steiner tree problem with G, c as in Figure 5.2 and terminal node set $N = \{a, b, e\}$. For this instance, formulate (IP), the LP relaxation (P), and the dual LP (D) of (P). Solve (IP) and (P) using the CPLEX optimizer.

(b) Discuss the complementary slackness conditions for the LP (P), and use these to design a primal-dual algorithm that (approximately) solves the Steiner tree problem.

(Note: When you submit (IP) to CPLEX, make sure that the right-hand side of each constraint is a *number*, i.e., move all the z_i's to the left-hand side.)

Bibliography

[1] P. Berman, U. Fössmeyer, M. Karpinski, M. Kauffman, and A. Zelikovsky, *Approaching the 5/4 approximation for rectilinear Steiner trees*, Proc. of European Symposium on Algorithms '94, LNCS **855** (1994), pp. 60-71.

[2] P. Berman and V. Ramaiyer, *Improved approximations for the Steiner tree problem*, J. Algorithms, **17(3)** (1994), pp. 381-408.

[3] D. Z. Du, Y. Zhang and Q. Feng, *On better heuristics for Euclidean Steiner minimum trees*, FOCS 91, 431-439.

[4] L. D Kou, G. Markowsky and L. Berman, *A fast algorithm for Steiner trees,* Acta Informatica **15** (1981), pp. 141-145.

[5] K. Mehlhorn, *A faster approximation algorithm for the Steiner problem in graphs*, Information Processing Letters **27(3)** (1988), pp. 125-128.

[6] H. Takahashi and A. Matsuyama, *An approximate solution for the Steiner problem in graphs*, Math. Japonica **24** (1980), pp. 573-577.

[7] A. Z. Zelikovsky, *The 11/6-approximation algorithm for the Steiner problem on networks,* Algorithmica **9** (1993), pp. 463-470.

Chapter 8

A General Approximation Technique for Constrained Forest Problems

This chapter studies an approximation algorithm due to M.X.Goemans and D.P.Williamson for so-called constrained forest problems, see [3] and [4]. The algorithm is based on the primal-dual method for solving a linear programming relaxation of the problem. For further details see these papers. In the context of network design problems, such approximation techniques were first presented by Agrawal, Klein and Ravi [1].

Subsequently this primal-dual method of Goemans and Williamson has been extended to several other network problems. Williamson et al [11] apply it to the generalized Steiner network problem, and Ravi and Williamson [9] apply it to the min-cost k-node connected spanning subgraph problem. Chudak et al [2] give 2-approximation algorithms based on the primal-dual method for the minimum feedback vertex set problem in undirected graphs. The primal-dual method appears to provide a versatile paradigm for the design and analysis of approximation algorithms.

We mention that Jain [7] has recently given a 2-approximation algorithm for the generalized Steiner network problem; this algorithm is based on repeatedly solving LP relaxations of the problem, but does not use the primal-dual method.

8.1 Constrained forest problems

Several problems in network design can be modeled as constrained forest problems. Some examples are: the minimum spanning tree problem, the shortest s-t path problem, the minimum-cost T-join problem, the Steiner minimal tree problem, the generalized Steiner forest problem, and the point-to-point connection problem (the last two are defined below). Moreover, assuming that the edge costs satisfy the triangle inequality, the GW (Goemans-Williamson) approximation algorithm can be applied to find approximately optimal solutions to the minimum-cost perfect matching problem and the exact tree (or cycle or path) partitioning problem. The key result in this chapter is that for (proper) constrained forest problems, the GW algorithm finds a solution whose cost is guaranteed to be within a factor of two of the optimal cost. Thus a variety of NP-hard problems in network design can be approximately solved within a factor of two, using this algorithm. The algorithm finds optimal solutions to the minimum spanning tree problem and the shortest s-t path problem;

for the first problem, the GW algorithm specializes to Kruskal's algorithm, and for the second it resembles a variant of Dijkstra's algorithm.

The GW algorithm may be viewed in several ways: It is a primal-dual algorithm, based on the linear programming relaxation of an integer programming formulation of a constrained forest problem; the algorithm is guided by the complementary slackness conditions and alternately performs updates on the primal LP solution and the dual LP solution. Also, the algorithm may be regarded as an adaptive greedy algorithm that repeatedly selects an edge with minimum reduced cost; the algorithm is adaptive, since the reduced costs are updated throughout the execution. The algorithm runs in time $O(n^2 \log n)$ on a graph with n nodes.

The definition of a proper function is given later, along with the requirements for a set of edges to satisfy a proper function.

Given an undirected graph $G = (V, E)$, nonnegative edge costs $c : E \to \Re_+$, and a proper function $f : 2^V \to \{0, 1\}$ (that is, f assigns a value of 0 or 1 to each subset S of V), the problem is to find an edge set $H \subseteq E$ that satisfies f such that its cost, $c(H) = \sum_{e \in H} c_e$, is minimum.
Recall our notation $\delta(S)$, where S is a node subset: $\delta(S)$ is the set of edges with one end in S and the other end in $V \backslash S$. An edge set H is said to *satisfy* f if

$$|H \cap \delta(S)| \geq f(S), \qquad \forall S \subset V.$$

That is, H satisfies f if the number of edges of H with exactly one end in S is at least $f(S)$, for all node subsets S.

An integer linear programming formulation of a constrained forest problem is easily obtained from the above problem statement:

$$\boxed{\begin{array}{llccll} \text{(IP)} & \text{minimize} & \sum_{e \in E} c_e & x_e & & \\ & \text{subject to} & \sum_{e \in \delta(S)} & x_e \geq & f(S), & \forall S \subset V \\ & & & x_e \in & \{0,1\}, & \forall e \in E. \end{array}}$$

8.2 Proper functions

A function $f : 2^V \to \{0, 1\}$ (assume that $f(V) = f(\emptyset) = 0$) is called *proper* if it satisfies the two properties of symmetry and maximality (also called disjointness):

[symmetry] $f(S) = f(V \backslash S)$, $\forall S \subset V$,

[maximality] $f(A \cup B) \leq f(A) + f(B)$, for any two disjoint subsets A, B of V.

The maximality property may be restated as :
$f(A) = 0,\ f(B) = 0 \implies f(A \cup B) = 0$ for any two disjoint subsets A, B of V.

An example of a proper function is given in Table 8.1; the next proposition shows that the function in the example is proper.

Proposition 8.1 *Given a graph $G = (V, E)$ and a set of "terminal nodes" $N \subseteq V$, the function $f(S)$ given by*

$$f(S) = \begin{cases} 1 & \text{if } \emptyset \neq S \cap N \neq N \\ 0 & \text{otherwise} \end{cases}$$

is a proper function.

Proof: We must show that f satisfies

(i) symmetry:
For any $S \subseteq V$, if $S \cap N \neq \emptyset$ and $S \cap N \neq N$, then $(V \backslash S) \cap N \neq \emptyset$ and $(V \backslash S) \cap N \neq N$. Hence, $f(S) = f(V \backslash S)$.

(ii) maximality:
Consider disjoint subsets A and B of V, and suppose that $f(A \cup B) = 1$. Then there is a terminal node $z \in A \cup B$, say, z is in A, and also a terminal node z' that is *not* in $A \cup B$. Now consider $A \cap N$. Since $z \in A$, $A \cap N \neq \emptyset$. Since $z' \notin A \cup B$, $z' \notin A \cap N$, and so $A \cap N \neq N$. Hence, $f(A) = 1$.

∎

S	$f(S)$	S	$f(S)$
$\emptyset$	0	$\{a,b,c\}$	0
$\{a\}$	1	$\{a,b,d\}$	1
$\{b\}$	1	$\{a,b,e\}$	1
$\{c\}$	1	$\{a,c,d\}$	1
$\{d\}$	0	$\{a,c,e\}$	1
$\{e\}$	0	$\{a,d,e\}$	1
$\{a,b\}$	1	$\{b,c,d\}$	1
$\{a,c\}$	1	$\{b,c,e\}$	1
$\{a,d\}$	1	$\{b,d,e\}$	1
$\{a,e\}$	1	$\{c,d,e\}$	1
$\{b,c\}$	1	$\{a,b,c,d\}$	0
$\{b,d\}$	1	$\{a,b,c,e\}$	0
$\{b,e\}$	1	$\{a,b,d,e\}$	1
$\{c,d\}$	1	$\{a,c,d,e\}$	1
$\{c,e\}$	1	$\{b,c,d,e\}$	1
$\{d,e\}$	0	$\{a,b,c,d,e\}$	0

Table 8.1: An example of a proper function. The graph $G = (V, E)$ has 5 nodes a, b, c, d and e, and has $N = \{a, b, c\}$ as a set of "terminal nodes". The function f is given by $f(S) = 1$ if $\emptyset \neq S \cap N \neq N$, and $f(S) = 0$ otherwise

8.3 Using proper functions to model problems in network design

Four problems in network design are given here. The first three are NP-hard, whereas for the fourth problem, there is a polynomial-time algorithm based on matching theory. Each of the four problems can be modeled as a constrained forest problem such that the function f is proper. Hence, by applying the GW algorithm, an approximately optimal solution whose cost is within a factor of two of the optimal value can be obtained. For each of the four problems, first, the problem is defined, then the appropriate function f is given.

(i) Steiner tree problem:
Given a set $N \subseteq V$ of terminal nodes, find a minimum-cost tree that connects all nodes in N.

$$\forall S \subset V: \qquad f(S) = \begin{cases} 1 & \text{if } \emptyset \neq S \cap N \neq N \\ 0 & \text{otherwise.} \end{cases}$$

(ii) Generalized Steiner forest problem :
Given p sets of terminal nodes $N_1, N_2, \ldots, N_p$ (these need not be disjoint), find a minimum-cost forest that connects all nodes in $N_i, i = 1, 2, \ldots, p$. (The number of trees in the forest may be either $1, 2, \ldots,$ or p.)

$$\forall S \subset V: \qquad f(S) = \begin{cases} 1 \quad \text{if} & \begin{cases} \emptyset \neq S \cap N_1 \neq N_1 \text{ or} \\ \emptyset \neq S \cap N_2 \neq N_2 \text{ or} \\ \cdots \\ \emptyset \neq S \cap N_p \neq N_p \end{cases} \\ 0 \quad \text{otherwise.} \end{cases}$$

(iii) Point-to-point connection problem :
Given a set $C = \{c_1, \ldots, c_p\}$ of source nodes, and a set $D = \{d_1, \ldots d_p\}$ of destination nodes, find a minimum-cost forest such that each connected component of the forest contains the same number of source nodes and destination nodes.

$$\forall S \subset V: \qquad f(S) = \begin{cases} 1 & \text{if} \qquad |S \cap C| \neq |S \cap D|, \\ 0 & \text{otherwise.} \end{cases}$$

(iv) T-join problem :
Given a node set T of even cardinality, find a set of edges F of minimum cost such that every node in T is incident to an odd number of edges of F.

$$\forall S \subset V: \qquad f(S) = \begin{cases} 1 & \text{if} \qquad |S \cap T| \text{ is odd,} \\ 0 & \text{otherwise.} \end{cases}$$

The function f in part (i) has been shown already to be proper; the other three parts are left as an exercise.

8.4 The LP relaxation of (IP)

The linear programming (LP) relaxation of our integer linear programming formulation of a constrained forest problem (IP) and the dual LP are as follows. In the LP relaxation, the constraints

$$x_e \leq 1, \qquad \forall e \in E,$$

are redundant, since for every feasible solution x^*, any $x_e^* > 1$ may be replaced by $x_e^* = 1$ without violating any constraints, and moreover the objective value $\sum c_e x_e^*$ does not increase.

(LP relaxation)			(Dual LP)		
minimize	$z = \sum_{e \in E} c_e x_e$		maximize	$w = \sum_{S \subset V} f(S) y_S$	
subject to	$\sum_{e \in \delta(S)} x_e \geq f(S),$	$\forall S \subset V$	subject to	$\sum_{S : e \in \delta(S)} y_S \leq c_e,$	$\forall e \in E$
	$x_e \geq 0,$	$\forall e \in E$		$y_S \geq 0,$	$\forall S \subset V$

The complementary slackness conditions are as follows:

$$\text{[primal]} \quad x_e > 0 \;\Rightarrow\; \sum_{S : e \in \delta(S)} y_S = c_e, \quad \forall e \in E$$

$$\text{[dual]} \quad y_S > 0 \;\Rightarrow\; \sum_{e \in \delta(S)} x_e = f(S), \quad \forall S \subset V$$

The GW algorithm is driven by the primal complementary slackness conditions; the dual complementary slackness conditions do not always hold, but they are needed in the analysis of the performance guarantee.

8.5 The GW algorithm for the constrained forests problem

The GW algorithm augments a subset F of the edges by one edge in each iteration (initially, $F = \emptyset$), until F satisfies the proper function f. Let F^* denote F at the end of the last iteration. Although F^* satisfies f, it may not be a minimal edge set satisfying f. The last step of the algorithm is a *clean-up* step that constructs a subset F' of F^* consisting of edges that are necessary to satisfy f. That is, F' is obtained from F^* by removing edges e such that $F^* \backslash e$ satisfies f. The clean-up step is essential for obtaining the performance guarantee.

The algorithm starts with the dual feasible solution $y_S = 0, \;\; \forall S \subset V$, and a primal solution that is infeasible, $x_e = 0, \;\; \forall e \in E$. At each iteration, corresponding to the edge set F there is a solution (that may not be feasible) of the primal LP:

$$x_e = \begin{cases} 1 & \text{if } e \in F \\ 0 & \text{otherwise.} \end{cases}$$

At any step, let C be the node set of a connected component of the subgraph formed by the current edge set F, i.e., C induces a maximal connected subgraph of (V, F). Then C is called an *active component* if $f(C) = 1$. Note that if $C \subset V$ is an active component, then the current edge set F does *not* satisfy

$$|\delta(C) \cap F| \geq f(C),$$

but for every proper subset $C' \subset C$ the current edge set F does satisfy

$$|\delta(C') \cap F| \geq 1 \geq f(C').$$

At each iteration, the algorithm finds the maximum ϵ such that for each active component C, the dual variable y_C can be increased by ϵ without violating any constraint of the dual LP, and then increases y_C for all active components C by ϵ. After this increase, one or more edges $e^* \in \delta(C)$, C an active component (clearly, $e^* \in E \backslash F$), satisfy

$$\sum_{S \subset V : e^* \in \delta(S)} y_S = c_{e^*},$$

i.e., the edge e^* becomes "tight" in the sense that the corresponding constraint in the dual LP now holds with equality. One such edge $e^* = ij$ is added to F, that is, in the primal LP, x_{e^*} is increased from zero to one. Clearly, e^* satisfies the primal complementary slackness condition when it is added to F; this fact is useful for the analysis. Let $C(i)$ and $C(j)$ denote the node sets of the components containing i and j respectively, before the edge $e^* = ij$ is added to F. After ij is added to F, both $C(i)$ and $C(j)$ will satisfy the constraint $|\delta(S) \cap F| \geq f(S)$, since $|\delta(C(i)) \cap F| = 1 \geq f(C(i))$ (similarly for $C(j)$). However, the new connected component that contains the edge $e^* = ij$ and has node set $C = C(i) \cup C(j)$ may have $f(C) = 1$ and so C may now be an active component. The algorithm repeats the above iteration, augmenting edges to F, until F satisfies the proper function f. A sketch of the algorithm follows.

Initialize:
$y_S := 0, \qquad \forall S \subset V$; ($y$ satisfies all constraints of the dual LP)
$F := \emptyset$;
while F does not satisfy the given proper function f **do**
let ϵ be the maximum number such that the dual variables y_C of all active components C can be increased to $y_C + \epsilon$ without violating any constraint of the dual LP;
for each active component C, increase y_C by ϵ;
at least one edge $e^* = ij \in \delta(C)$, C an active component, becomes tight, i.e.,
$\qquad \sum_{S : e^* \in \delta(S)} y_S$ becomes equal to c_{e^*};
add e^* to F;
end; (while)

perform the clean-up step, and output the resulting forest F';

The key result on the GW algorithm is the following performance guarantee.

Theorem 8.2 *Let $F' \subseteq E$ be the forest output by the GW algorithm, let $y_S (S \subset V)$ be the dual solution at termination of the algorithm, and let Z^*_{IP} be the cost of an optimal solution. Then*

$$\sum_{e \in F'} c_e \leq 2 \sum_{S \subset V} y_S \leq 2 Z^*_{IP}.$$

In other words, the GW algorithm finds a feasible solution whose cost is at most two times the optimal cost. The second inequality in the theorem follows because the objective value of every feasible solution of the dual LP is at most the optimal value of the LP, Z^*_{LP}, by linear programming weak duality, and Z^*_{LP} is at most Z^*_{IP}, since (LP) is a relaxation of (IP).

Before presenting the proof of the first inequality in the theorem, a detailed implementation of the GW algorithm is given, followed by two examples of the working of the algorithm, see Figures 8.1 and 8.2. To obtain a reasonably efficient implementation of the algorithm, two issues need to be examined, namely, how to maintain the dual variables $y_S, S \subset V$, and how to discover the active components in each iteration. The algorithm maintains a list (or set) Γ of the node sets C of the connected components of the current subgraph (V, F). Initially, Γ consists of $|V|$ singleton sets $\{v\}$, one for each $v \in V$. In each iteration, when edge ij is added to F, the components containing the end nodes i and j, namely, $C(i)$ and $C(j)$, "merge", that is, $C(i)$ and $C(j)$ are removed from Γ and are replaced by $C(i) \cup C(j)$. The active components are precisely the node sets C in Γ such that $f(C) = 1$, and these are easily discovered by the algorithm. Let Γ_1 be the set of active components, $\Gamma_1 = \{C \subset V : f(C) = 1 \text{ and } C \text{ is in } \Gamma\}$. Since the algorithm raises the dual variables y_C for the node sets $C \in \Gamma_1$ by ϵ, the dual objective value, $\sum_{S \subset V} f(S) y_S$, increases by $\epsilon \cdot |\Gamma_1|$. In the detailed implementation, there is a variable LB which gives the current dual objective value. Rather than explicitly maintaining all the dual variables, $y_S, S \subset V$, the algorithm maintains a value $d(v)$ for each node v such that

$$d(v) = \sum_{S: v \in S} y_S, \qquad \forall v \in V.$$

Initially, $d(v) = 0$ for each node v, and in each iteration, if the node v is in an active component, then $d(v)$ is increased by ϵ, otherwise $d(v)$ stays unchanged. Clearly, this ensures that the above assertion on $d : V \to \Re_+$ holds at the start of each iteration.

Algorithm GW Constrained Forest Algorithm.
input: Graph $G = (V, E)$, edge costs $c : E \to \Re_+$, proper function $f : 2^V \to \{0, 1\}$.
output: Forest $F' \subseteq E$ satisfying f, and value LB such that LB $\leq Z^*_{IP} \leq c(F') \leq 2 \cdot$ LB ,
where Z^*_{IP} is the cost of an optimal solution.

$F := \emptyset$;
LB $:= 0$;
$\Gamma := \{\{v\} : v \in V\}$;
for each $v \in V$ **do** $d(v) := 0$; **end**;

while $\exists C \in \Gamma$ such that $f(C) = 1$ **do**

(there exists an active component C)
find an edge $e^* = ij$ with $i \in C(i) \in \Gamma$ and $j \in C(j) \in \Gamma$, $C(i) \neq C(j)$,
that minimizes $\epsilon = \dfrac{c_{ij} - d(i) - d(j)}{f(C(i)) + f(C(j))}$;
for each $v \in C \in \Gamma$ **do if** $f(C) = 1$ **then** $d(v) := d(v) + \epsilon$; **end**; **end**;
$\gamma_1 := \sum_{C \in \Gamma} f(C)$; ($\gamma_1$ is the number of active components)
LB := LB $+ \epsilon \cdot \gamma_1$;
$F := F \cup \{ij\}$;
$\Gamma := (\Gamma \backslash \{C(i), C(j)\}) \cup \{(C(i) \cup C(j))\}$;
end (while);

CLEAN-UP:
$F' := \{e \in F : \exists D \subset V : D$ is a connected component of $(V, F \backslash \{e\})$ and $f(D) = 1\}$;

8.6 A performance guarantee for the GW algorithm

The proof of the approximation guarantee for the GW algorithm, Theorem 8.2, is presented here. Recall the statement of the theorem:
Let $F' \subseteq E$ be the forest output by the GW algorithm, let $y_S(S \subset V)$ be the dual solution at termination of the algorithm, and let Z^*_{IP} be the cost of an optimal solution. Then

$$\sum_{e \in F'} c_e \leq 2 \sum_{S \subset V} y_S \leq 2 Z^*_{IP}.$$

The proof hinges on the next lemma. The lemma characterizes the number of edges of the final output F' (after the CLEAN-UP step) in the "boundary" $\delta(C)$ of a node set C such that $f(C) = 0$. The lemma fails if F' is replaced by F^*, where F^* denotes the forest F just before the CLEAN-UP step.

In the proofs below, a connected component of a subgraph of G (such as the subgraph (V, F)) is sometimes identified with its node set; this should not cause any confusion.

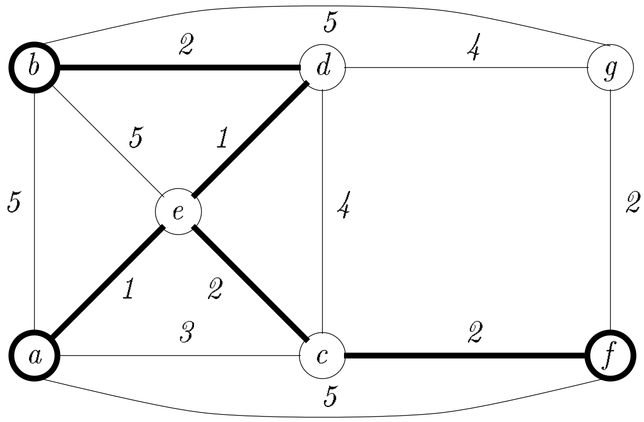
Lemma 8.3 *For every node set $C \subset V$, if $f(C) = 0$, then the final output F' has either zero or two or more edges with exactly one end in C, i.e.,*

$$f(C) = 0 \quad \Longrightarrow \quad |\delta(C) \cap F'| = 0 \quad or \quad |\delta(C) \cap F'| \geq 2.$$

Proof: The following fact is needed to prove the lemma.

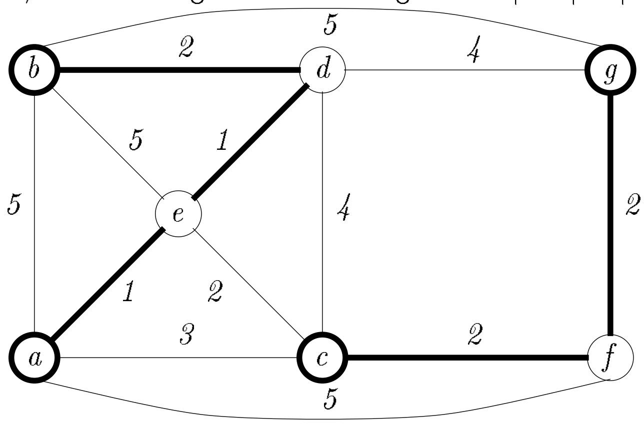
Fact: If S is a subset of V such that $f(S) = 0$, and B is a subset of S such that $f(B) = 0$, then $f(S \backslash B) = 0$.

Figure 8.1: Steiner tree problem on the graph $G = (V, E)$ with terminal node set $N = \{a, b, f\}$ is approximately solved using the GW algorithm. In the list Γ, the active components are indicated by boxes. The final solution F' is indicated in thick lines.



edge $e^* = ij$ added to F	$\epsilon = \frac{c_{ij}-d(i)-d(j)}{f(C(i))+f(C(j))}$	$d(v)$ $a\ b\ c\ d\ e\ f\ g$	LB	Γ
START $F = \emptyset$	—	0 0 0 0 0 0 0	0	$\boxed{\{a\}}, \boxed{\{b\}}, \{c\}, \{d\}, \{e\}, \boxed{\{f\}}, \{g\}$
ae	$\frac{1-0-0}{1+0} = 1$	1 1 0 0 0 1 0	3	$\boxed{\{a,e\}}, \boxed{\{b\}}, \{c\}, \{d\}, \boxed{\{f\}}, \{g\}$
de	$\frac{1-0-0}{0+1} = 1$	2 2 0 0 1 2 0	6	$\boxed{\{a,e,d\}}, \boxed{\{b\}}, \{c\}, \boxed{\{f\}}, \{g\}$
fg	$\frac{2-2-0}{1+0} = 0$	2 2 0 0 1 2 0	6	$\boxed{\{a,e,d\}}, \boxed{\{b\}}, \{c\}, \boxed{\{f, g\}}$
cf	$\frac{2-0-2}{0+1} = 0$	2 2 0 0 1 2 0	6	$\boxed{\{a,e,d\}}, \boxed{\{b\}}, \boxed{\{c, f, g\}}$
bd	$\frac{2-2-0}{1+1} = 0$	2 2 0 0 1 2 0	6	$\boxed{\{a, b, d, e\}}, \boxed{\{c, f, g\}}$
ce	$\frac{2-0-1}{1+1} = \frac{1}{2}$	$\frac{5}{2}\ \frac{5}{2}\ \frac{1}{2}\ \frac{1}{2}\ \frac{3}{2}\ \frac{5}{2}\ \frac{1}{2}$	7	$\{a, b, c, d, e, f, g\}$
$F^* = \{ae, bd, ce, de, cf, fg\}$				
the CLEAN-UP step removes edge fg final forest: $F' = \{ae, bd, ce, de, cf\}$, $c(F') = 8$.				

Figure 8.2: Generalized Steiner forest problem on the graph $G = (V, E)$ with terminal node sets $N_1 = \{a, b\}$ and $N_2 = \{c, g\}$ is approximately solved using the GW algorithm. The proper function $f : 2^V \to \{0, 1\}$ is given by $f(S) = 1$ if either $\emptyset \neq S \cap N_1 \neq N_1$ or $\emptyset \neq S \cap N_2 \neq N_2$, and $f(S) = 0$ otherwise. In the list Γ, the active components are indicated by boxes. The final solution F' is indicated in thick lines.
F' is an optimal solution, since the edge costs are integral and $\lceil \text{LB} \rceil = \lceil 7.5 \rceil \geq c(F')$.



edge $e^* = ij$ added to F	$\epsilon = \frac{c_{ij} - d(i) - d(j)}{f(C(i)) + f(C(j))}$	$d(v)$ $a\ b\ c\ d\ e\ f\ g$	LB	Γ
START $F = \emptyset$	—	0 0 0 0 0 0 0	0	$\boxed{\{a\}}, \boxed{\{b\}}, \boxed{\{c\}}, \{d\}, \{e\}, \{f\}, \boxed{\{g\}}$
ae	$\frac{1-0-0}{1+0} = 1$	1 1 1 0 0 0 1	4	$\boxed{\{a,e\}}, \boxed{\{b\}}, \boxed{\{c\}}, \{d\}, \{f\}, \boxed{\{g\}}$
ce	$\frac{2-1-0}{1+1} = \frac{1}{2}$	$\frac{3}{2}\ \frac{3}{2}\ \frac{3}{2}\ 0\ \frac{1}{2}\ 0\ \frac{3}{2}$	6	$\boxed{\{a,c,e\}}, \boxed{\{b\}}, \{d\}, \{f\}, \boxed{\{g\}}$
cf	$\frac{2-(3/2)-0}{1+0} = \frac{1}{2}$	2 2 2 0 1 0 2	$\frac{15}{2}$	$\boxed{\{a,c,e,f\}}, \boxed{\{b\}}, \{d\}, \boxed{\{g\}}$
fg	$\frac{2-0-2}{1+1} = 0$	2 2 2 0 1 0 2	$\frac{15}{2}$	$\boxed{\{a,c,e,f,g\}}, \boxed{\{b\}}, \{d\}$
de	$\frac{1-0-1}{0+1} = 0$	2 2 2 0 1 0 2	$\frac{15}{2}$	$\boxed{\{a,c,d,e,f,g\}}, \boxed{\{b\}}$
bd	$\frac{2-2-0}{1+1} = 0$	2 2 2 0 1 0 2	$\frac{15}{2}$	$\{a, b, c, d, e, f, g\}$

$F^* = \{ae, bd, ce, de, cf, fg\}$

the CLEAN-UP step removes edge ce
final forest: $F' = \{ae, bd, de, cf, fg\}$, $c(F') = 8$.

Proof: (of the fact) Consider the set $S \backslash B$, and note that its complement, $V \backslash (S \backslash B)$ is the union of two disjoint sets $V \backslash S$ and B. Now,

$$\begin{aligned} f(S\backslash B) &= f(V\backslash(S\backslash B)), \quad \text{by symmetry,} \\ &= f((V\backslash S) \cup (B)) \\ &\leq f(V\backslash S) + f(B), \quad \text{by maximality,} \\ &= f(S) + f(B), \quad \text{since } f(V\backslash S) = f(S) \text{ by symmetry,} \\ &= 0 + 0. \end{aligned}$$

▮

The lemma is proved by contradiction. Suppose that there is a node set C such that $f(C) = 0$ and $|\delta(C) \cap F'| = 1$. Let e^* be the unique edge of F' with exactly one end in C. Let F^* denote the forest F just before the CLEAN-UP step. Let C^* be the node set of the connected component of the forest F^* that contains C, and let the node sets of the two connected components formed from C^* by removing e^* be N^* and $(C^* \backslash N^*)$. Fix the notation such that C is a subset of N^*. There are two cases.

case (1): $N^* = C$.
Then $f(N^*) = f(C) = 0$. Further, $f(C^*) = 0$, since by the definition of F^*, the node set D of each connected component of (V, F^*) has $f(D) = 0$. From the above fact it follows that $f(C^* \backslash N^*) = 0$. This gives the desired contradiction, since the CLEAN-UP step should have removed the edge e^* from F'.

case (2): $N^* \supset C$.
Then $\delta(C) \cap F^*$ has one or more edges, besides the edge e^*. Let $e^*, e_1, \ldots, e_k$ be the edges in $\delta(C) \cap F^*$. Removing each of the edges e_i, $1 \leq i \leq k$, from F^* disconnects C^* into two connected components, one containing C and one contained in N^*. Let $C_1, \ldots, C_k$, $C_i \subset N^*(1 \leq i \leq k)$, denote the node sets of the latter kind of connected components of $(V, F^* \backslash e_i)$, $1 \leq i \leq k$. Clearly, the node sets $C, C_1, \ldots, C_k$ are (pairwise) disjoint, and N^* is the union of these node sets. Since the CLEAN-UP step removes the edges $e_1, \ldots, e_k$ from F^* to obtain F', it follows that $f(C_i) = 0$ for each i, $1 \leq i \leq k$. Hence, by maximality,

$$f(N^*) \leq f(C) + f(C_1) + \ldots + f(C_k) = 0.$$

Now, the proof is completed as in case (1).

▮

Proof: (**Theorem 8.2**) For every edge e added by the algorithm to F

$$\sum_{S: e \in \delta(S)} y_S = c_e.$$

That is, each edge e added to F satisfies the primal complementary slackness condition. Hence,

$$\begin{aligned} \sum_{e \in F'} c_e &= \sum_{e \in F'} \sum_{S: e \in \delta(S)} y_s \\ &= \sum_{S \subset V} y_S \cdot |\delta(S) \cap F'|. \end{aligned}$$

The proof is completed by using induction on the number of iterations to show that

$$\sum_{S\subset V} y_S \cdot |\delta(S)\cap F'| \quad \leq \quad 2\sum_{S\subset V} y_S. \tag{1}$$

The induction basis holds, since at the start of the algorithm $y_S = 0,\ \ \forall S \subset V$, hence inequality (1) holds. For the induction step, note that in each iteration, for each active component C in the list Γ, y_C increases by ϵ, and for all remaining sets $S \subset V$, y_S stays the same. That is, the left-hand side of inequality (1) increases by

$$\epsilon \sum_{C:C\in\Gamma} f(C)\cdot|\delta(C)\cap F'|,$$

and the right-hand side of inequality (1) increases by

$$2\epsilon \sum_{C:C\in\Gamma} f(C).$$

Inequality (1) would continue to hold if

$$\sum_{C:C\in\Gamma} f(C)\cdot|\delta(C)\cap F'| \quad \leq \quad 2\sum_{C:C\in\Gamma} f(C), \tag{2}$$

that is, if the average number of edges of F' in the "boundary" $\delta(C)$ of an active component C is at most two.

The proof is completed by showing that inequality (2) holds. Construct a graph H from the list Γ: the nodes of H correspond to the node sets in Γ, and there is an edge CD in H iff in the original graph $G = (V, E)$ there is an edge $e \in F'$ such that $e \in \delta(C)$ and $e \in \delta(D)$. That is, the graph H is obtained from the subgraph (V, F') of G by contracting each node set C in the list Γ.

Fact: The graph H is a forest.

For the rest of the proof, assume that no tree of H has a single node, i.e., assume that in the original graph G each node set C in the list Γ has $|\delta(C)\cap F'| \geq 1$. The proof is easily modified for the case when the assumption does not hold.

Since H is a forest, it has at most $|V(H)|-1$ edges, and so the sum of the degrees of the nodes in H is at most $2(|V(H)|-1)$. In terms of the original graph G this gives

$$\sum_{C\in\Gamma} |\delta(C)\cap F'| \quad \leq \quad 2(|\Gamma|-1) \tag{3}$$

From Lemma 8.3 and the assumption above, it follows that in the original graph G for each node set C in the list Γ with $f(C) = 0$, $|\delta(C)\cap F'|$ is at least two. Summing over all node sets C in Γ with $f(C) = 0$ gives

$$\sum_{C\in\Gamma} (1-f(C))\cdot|\delta(C)\cap F'| \quad \geq \quad 2\sum_{C\in\Gamma}(1-f(C)),$$

or, multiplying through by -1,

$$\sum_{C\in\Gamma} (f(C)-1)\cdot|\delta(C)\cap F'| \quad \leq \quad 2\sum_{C\in\Gamma}(f(C)-1). \tag{4}$$

Adding inequalities (3) and (4) gives the desired inequality (2).

The proof of the induction step is now complete, and so the theorem has been proved. ∎

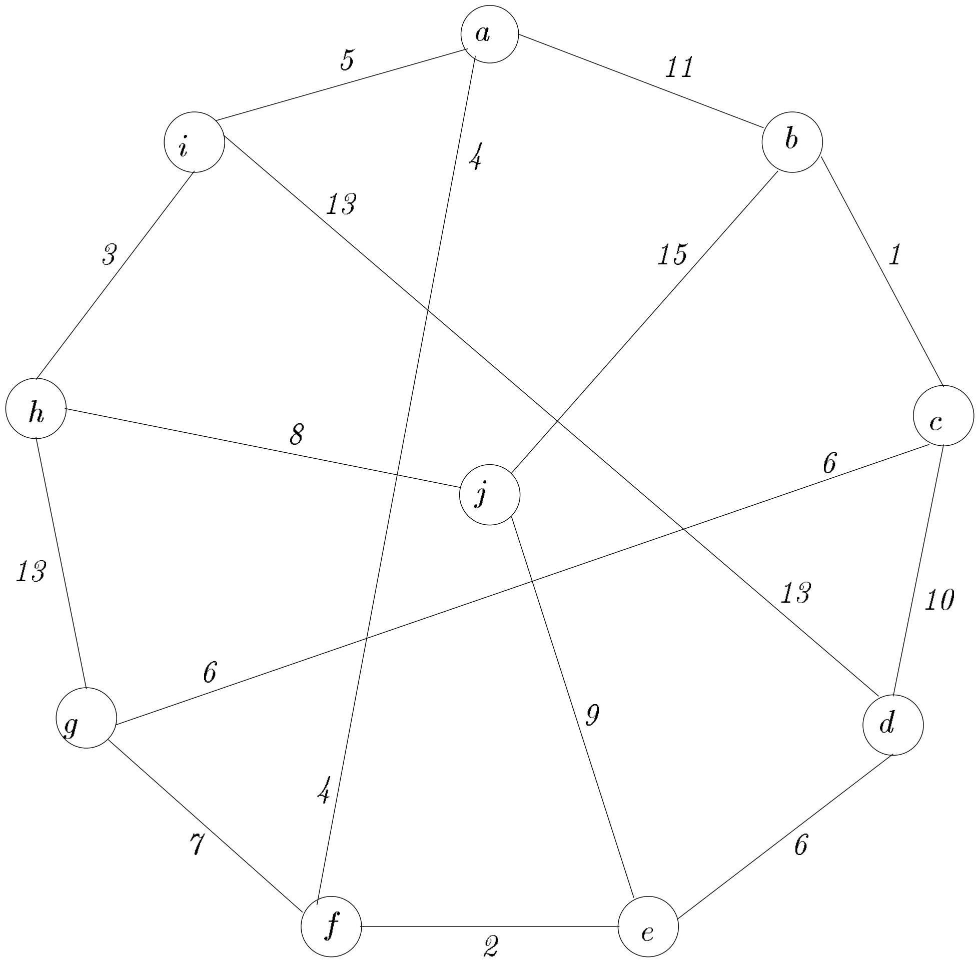


Figure 8.3: A network $G,\ c$ for constrained forest problems.

8.7 Exercises

1. Let f be a proper function, and let $S^* \subset V$ be a set maximizing $f(S^*)$. Prove that there is a node $v \in S^*$ such that $f(\{v\}) = f(S^*)$.

2. A function $f : 2^V \to \{0,1\}$ is said to satisfy the *complementarity* property if for all $A \subseteq S \subseteq V$, $f(S) = f(A) = 0 \implies f(S \backslash A) = 0$.

   (a) Prove that for symmetric functions f, the maximality property is equivalent to the complementarity property.

   (b) Let $f : 2^V \to \{0,1\}$ be a (not necessarily symmetric) function satisfying maximality and complementarity. Show that the symmetrization f_{sym} is a proper function, where f_{sym} is given by
$$f_{sym}(S) = \max(f(S),\ f(V \backslash S)), \qquad \forall S \subseteq V.$$

3. A function $h : 2^V \to \{0,1\}$ is said to be *uncrossable* if $h(V) = 0$, and
$$\forall A \subseteq V, B \subseteq V \qquad h(A) = h(B) = 1 \qquad \Longrightarrow \qquad \begin{array}{l}\text{either } h(A \cup B) = h(A \cap B) = 1, \text{ or} \\ h(A \backslash B) = h(B \backslash A) = 1.\end{array}$$
   Prove that any function $g : 2^V \to \{0,1\}$ satisfying maximality is uncrossable.

4. In each of parts (ii), (iii) and (iv) of Section 8.3, prove that the function f is proper.

5. Let $G = (V, E)$ be a graph, and let $c : E \to \Re_+$ be a nonnegative cost function on the edges. The goal is to use the GW (Goemans-Williamson) algorithm to find a minimum spanning tree (MST) of G, c.

   (a) State the function $f : 2^V \to \{0,1\}$ to use for finding an MST, and prove that the function f is proper.

   (b) Explain why the following is true:
   If the GW algorithm is applied to G, c with an appropriate proper function f, then it finds a minimum spanning tree.

6. Use the constrained forest algorithm of Goemans and Williamson to solve the following problems on the network G, c in Figure 8.3.

   (a) Find (an approximate) Steiner minimal tree with terminal node set $N = \{b, e, h\}$.

   (b) Solve (approximately) the generalized Steiner forest problem with $N_1 = \{a, b, e\}$ and $N_2 = \{d, j\}$.

   (c) Solve (approximately) the point-to-point connection problem with the source set $C = \{b, e, f\}$ and the destination set $D = \{a, h, j\}$.

   (d) Solve (approximately) the T-join problem with $T = \{a, b, e, f, h, j\}$.

Bibliography

[1] A. Agrawal, P. N. Klein and R. Ravi, *When trees collide: An approximation algorithm for the generalized Steiner problem on networks*, SIAM J. Computing **24** (1995). Preliminary version in Proc. 23rd Ann. ACM S. T. O. C. (1991), pp. 134–144.

[2] F. A. Chudak, M. X. Goemans, D. S. Hochbaum, and D. P. Williamson, *A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs.* To appear in Operations Research Letters. Available at `http://www.research.ibm.com/people/w/williamson/`

[3] M. X. Goemans and D. P. Williamson, *A general approximation technique for constrained forest problems*, SIAM J. Computing **24** (1995).

[4] M. X. Goemans and D. P. Williamson, *The primal-dual method for approximation algorithms and its application to network design problems*, in *Approximation algorithms for NP-hard problems*, ed. D. S. Hochbaum, PWS publishing co., Boston, 1996.

[5] M. Grötschel, C. L. Monma and M. Stoer, *Design of survivable networks*, chapter from the Handbook in Operations Research and Management Science, March 1993.

[6] F. K. Hwang, D. S. Richards and P. Winter, *The Steiner Tree Problem.* Annals of Discrete Mathematics, 53, North-Holland, Amsterdam, 1992.

[7] K. Jain, *A factor 2 approximation algorithm for the generalized Steiner network problem.* Manuscript, January, 1998, `http://www.cc.gatech.edu/people/home/kjain/`
To appear in the Proc. 39th IEEE FOCS, Palo Alto, CA, November 1998.

[8] T. L. Magnanti and L. A. Wolsey, *Optimal Trees.* CORE discussion paper 9426, C. O. R. E., Universite Catholique de Louvain, Louvain-la-neuve, Belgium, May 1994.

[9] R. Ravi and D. P. Williamson, *An approximation algorithm for minimum-cost vertex-connectivity problems. Algorithmica* **18** (1997), 21-43.

[10] M. Stoer and G. Dahl, *A polyhedral approach for multicommodity survivable network design*, Numerische Mathematik **68** (1994), pp. 149–167.

[11] D. P. Williamson, M. X. Goemans, M. Mihail and V. V. Vazirani, *A primal-dual approximation algorithm for generalized Steiner network problems*, Proceedings 25th ACM Symposium on Theory of Computing, 1993.

Chapter 9

Approximating minimum k-connected spanning subgraphs

9.1 Introduction

This chapter focuses on (approximately) minimum k-connected spanning subgraphs of a given graph $G = (V, E)$. We study both k-edge connected spanning subgraphs (abbreviated k-ECSS), and k-node connected spanning subgraphs (k-NCSS). When stating facts that apply to both a k-ECSS and a k-NCSS, we use the term k-connected spanning subgraph (k-CSS). We take G to be an undirected graph. Mostly, we take G to be a simple graph (i.e., G has no loops nor multiple edges), but while discussing the general k-ECSS problem, we study both simple graphs and multi graphs (i.e., graphs with multiple copies of one or more edges). Let n and m denote the number of nodes and the number of edges, respectively.

Several different types of the linear objective function (i.e., vector of edge costs c_{vw}) have been studied. The most general case is when the objective function is nonnegative but is otherwise unrestricted. Two special types of objective functions turn out to be of interest in theory and practice: (1) the case of unit costs, i.e., the optimal solution is a k-ECSS or a k-NCSS with the minimum number of edges, and (2) the case of metric costs, i.e., the edge costs c_{vw} satisfy the triangle inequality.

Table 9.1 summarizes the best approximation guarantees currently known for the several types of k-CSS problems discussed above. At present, for minimum k-CSS problems, approximation guarantees better than 2 are known only for the case of unit costs and for some cases of metric costs. For nonnegative costs, it is not known whether or not the following problem is NP-complete: for a constant $\epsilon > 0$, find, say, a 2-ECSS whose cost is at most $(2 - \epsilon)$ times the minimum 2-ECSS cost.

Note that every node in a k-CSS has degree $\geq k$, hence, the number of edges in a k-ECSS or a k-NCSS is $\geq kn/2$.

The problem of finding a minimum k-ECSS or minimum k-NCSS is already NP-hard for the case $k = 2$ and unit costs. There is a direct reduction from the Hamiltonian cycle problem because G has a Hamiltonian cycle iff it has 2-ECSS (or 2-NCSS) with n edges. Recently, Fernandes [10, Theorem 5.1] showed that the minimum-size 2-ECSS problem on graphs is MAX SNP-hard.

Table 9.1: A summary of current approximation guarantees for minimum k-edge connected spanning subgraphs (k-ECSS), and minimum k-node connected spanning subgraphs (k-NCSS); k is an integer ≥ 2. The references are to:
- Cheriyan & Thurimella, IEEE F.O.C.S. (1996),
- Frederickson & Ja'Ja', Theor. Comp. Sci. 19 (1982) pp. 189–201,
- Khuller & Vishkin, JACM 41 (1994) pp. 214–235,
- Khuller & Raghavachari, J. Algorithms 21 (1996) pp. 434–450, and
- Ravi & Williamson, 6th ACM-SIAM S.O.D.A. (1995) pp. 332–341.

	Type of objective function		
	Unit costs	Metric costs	Nonnegative costs
k-ECSS simple-edge model	$1 + (2/(k+1))$ [CT96] 1.5 for $k = 2$ [KV94]	see last entry 1.5 for $k = 2$ [FJ82]	2 [KV94]
k-NCSS	$1 + (1/k)$ [CT96]	$2 + (2(k-1)/n)$ [KR96]	$2H(k) = O(\log k)$ [RW95]

The last section of this chapter has some bibliographic remarks, and discusses the sequence of papers that led up to the results in this chapter, see Section 9.12. The discussion may not be complete. (We hope to rectify any errors and omissions in future revisions of the chapter.)

9.2 Definitions and notation

For a subset S' of a set S, $S \backslash S'$ denotes the set $\{x \in S \mid x \notin S'\}$.

Let $G = (V, E)$ be a graph. By the *size* of G we mean $|E(G)|$. For a subset M of E and a node v, we use $\deg_M(v)$ to denote the number of edges of M incident to v; $\deg(v)$ denotes $\deg_E(v)$. An x-y path refers to a path whose end nodes are x and y. We call two paths *openly disjoint* if every node common to both paths is an end node of both paths. Hence, two (distinct) openly disjoint paths have no edges in common, and possibly, have no nodes in common. A set of $k \geq 2$ paths is called openly disjoint if the paths are pairwise openly disjoint. By a *component* (or connected component) of a graph, we mean a maximal connected subgraph, as well as the node set of such a subgraph. Hopefully, this will not cause confusion.

For node set $S \subseteq V(G)$, $\delta_G(S)$ denotes the set of all edges in $E(G)$ that have one end node in S and the other end node in $V(G) \backslash S$ (when there is no danger of confusion, the notation is abbreviated to $\delta(S)$); $\delta(S)$ is called a *cut*, and by a *k-cut* we mean a cut that has exactly k edges. A graph $G = (V, E)$ is said to be k-edge connected if $|V| \geq k + 1$ and the deletion of any set of $< k$ edges leaves a connected graph. For testing k-edge connectivity, currently Gabow [17] has a deterministic algorithm that runs in time $O(m + k^2 n \log(n/k))$, while Karger [27] has a randomized algorithm that runs in time $O(m + kn(\log n)^3)$.

For a subset $Q \subseteq V$, $N(Q)$ denotes the set of neighbors of Q in $V \backslash Q$, $\{w \in V \backslash Q \mid wv \in E,\ v \in$

$Q\}$. A *separator* S of G is a subset $S \subset V$ such that $G\backslash S$ has at least two components. A k-separator means a separator that has exactly k nodes. A graph $G = (V, E)$ is said to be *k-node connected* if $|V| \geq k+1$, and the deletion of any set of $< k$ nodes leaves a connected graph. For testing k-node connectivity, currently Rauch Henzinger, Rao and Gabow [37] have (1) a deterministic algorithm that runs in time $O(\min(k^2n^2,\ k^4n + kn^2))$ and (2) a randomized algorithm that runs in time $O(kn^2)$ with high probability provided $k = O(n^{1-\epsilon})$, where $\epsilon > 0$ is a constant.

An edge vw of a k-node connected graph G is called *critical* w.r.t. k-node connectivity if $G\backslash vw$ is *not* k-node connected. Similarly, we have the notion of critical edges w.r.t. k-edge connectivity.

9.2.1 Matching

A *matching* of a graph $G = (V, E)$ is an edge set $M \subseteq E$ such that $\deg_M(v) \leq 1,\ \forall v \in V$; furthermore, if every node $v \in V$ has $\deg_M(v) = 1$, then M is called a *perfect matching.* A graph G is called *factor critical* if for every node $v \in V$, there is a perfect matching in $G\backslash v$, see [32]. An algorithm due to Micali and Vazirani (1984) finds a matching of maximum cardinality in time $O(m\sqrt{n})$. If the graph is bipartite, there is a much simpler algorithm for finding a matching of maximum cardinality due to Hopcroft and Karp (1972), but the running time remains the same.

9.3 A 2-approximation algorithm for minimum weight k-ECSS

Let $G = (V, E)$ be a graph of edge connectivity $\geq k$, and let $c : E \to \Re_+$ assign a nonnegative cost to each edge $vw \in E$. This section gives an algorithm that finds a k-ECSS $G' = (V, E')$ such that the cost $c(E') = \sum_{vw \in E'} c(vw)$ is at most $2c(E_{opt})$, where E_{opt} denotes the edge set of a minimum-cost k-ECSS (i.e., for every k-ECSS (V, E''), $c(E'') \geq c(E_{opt})$). This result is due to Khuller & Vishkin [30]. The algorithm is a straightforward application of the weighted matroid intersection algorithm, which is due to Lawler and Edmonds. For our application there is an efficient implementation due to Gabow [17]. This section and the next one use directed graphs, and so we include definitions and notation pertaining to directed graphs in the box below.

For a directed graph $D = (V, A)$, where V is the set of nodes and A is the set of arcs, we use (v, w) to denote an arc (or directed edge) from v to w. The node v is called the *tail* of (v, w), and the node w is called the *head*. The arc (v, w) is said to *leave* v and to *enter* w. For a node set $S \subseteq V$, an arc (v, w) is said to leave S if $v \in S$ and $w \in V\backslash S$, and (v, w) is said to enter S if $w \in S$ and $v \in V\backslash S$. For a node set $S \subseteq V$, the *directed cut* $\delta_D(S)$ or $\delta(S)$ consists of all arcs leaving S (note that $\delta(S)$ has no arcs entering S). The *bidirected graph* $D = (V, A)$ of an undirected graph $G = (V, E)$ has the same node set, and for each edge $vw \in E$, the arc set A has both the arcs (v, w) and (w, v). The *undirected graph* $G = (V, E)$ of a directed graph $D = (V, A)$ has the same node set, and for each arc $(v, w) \in A$ or each arc pair $(v, w), (w, v) \in A$, the edge set E has *one* edge vw (i.e., G has one edge corresponding to a pair of oppositely oriented arcs). A directed graph is called *acyclic* if its undirected graph has no cycles. A directed graph is called a *directed spanning tree* if its undirected graph is a spanning tree. A *branching* (V, B) with *root node* v_0 is a directed spanning tree such that for each node $w \in V$, there is a directed path from v_0 to w; in other words, $|B| = |V| - 1$, each node $w \in V\backslash\{v_0\}$ has precisely one entering arc, v_0 has no entering arc, and (V, B) is acyclic.

The weighted matroid intersection algorithm efficiently solves the following problem (P) (and many others). Let $D = (V, A)$ be a directed graph, let $c : A \rightarrow \Re$ assign a real-valued cost to each arc, let v_0 be a node of D, and let $k > 0$ be an integer. The goal is to find a minimum-cost arc set $F \subseteq A$ such that F is the union of (the arc sets of) k arc-disjoint branchings with root v_0. In other words, the goal is to find $F \subseteq A$ such that $c(F)$ is minimum and $F = B_1 \cup \ldots \cup B_k$, where $B_1, \ldots, B_k$ are pairwise arc disjoint, and for $i = 1, \ldots, k$, (V, B_i) is a branching with root v_0. Gabow's implementation [17] either finds an optimal F or reports that no feasible F exists, and the running time is $O(k|V| \log |V|(|A| + |V| \log |V|))$.

To find a minimum-weight k-ECSS of G, c, we first construct the bidirected graph $D = (V, A)$ of G, and assign arc costs to D by taking $c(v, w) = c(w, v) = c(vw)$ for each edge $vw \in E$. Note that $c(A) = 2c(E)$. (It may be helpful to keep an example in mind: take G to be a cycle on $n \geq 3$ nodes, and take $k = 2$.) Choose an arbitrary node $v_0 \in V$. Observe that for every node set S with $v_0 \in S$ and $S \neq V$, the directed cut $\delta_D(S)$ has $\geq k$ arcs because the corresponding cut in G, $\delta_G(S)$, has $\geq k$ arcs. The next result shows that this directed graph D has a feasible arc set $F \subseteq A$ for problem (P) above.

Theorem 9.1 (Edmonds) *If a directed graph $D = (V, A)$ has $|\delta_D(S)| \geq k$ for every $S \subseteq V$ with $v_0 \in S$ and $S \neq V$, where v_0 is a node of D, then D has k arc-disjoint branchings with root v_0.*

We apply the weighted matroid intersection algorithm to D, c, v_0, where v_0 is an arbitrary node, to find an optimal arc set F for problem (P). Let $\delta_F(\cdot)$ denote a directed cut of (V, F). Clearly, $|\delta_F(S)| \geq k$, for every $S \subseteq V$ with $v_0 \in S$ and $S \neq V$, because F contains k arc-disjoint directed paths from v_0 to w, for an arbitrary node $w \in V \backslash S$. Let $G' = (V, E')$ be the undirected graph of (V, F). First, note that G' is k-edge connected (i.e., every nontrivial cut of G' has $\geq k$ edges), because for every $S \subseteq V$ with $\emptyset \neq S \neq V$, either $v_0 \in S$ or $v_0 \in V \backslash S$ and so either $|\delta_F(S)| \geq k$ or $|\delta_F(V \backslash S)| \geq k$.

We claim that $c(E') \leq 2c(E_{opt})$. To see this, focus on the minimum-cost k-ECSS $G_{opt} = (V, E_{opt})$. The directed graph D_{opt} of G_{opt} has total arc cost $= 2c(E_{opt})$, and (reasoning as above) the arc set of D_{opt} contains a feasible arc set F for our instance of problem (P). Hence, the arc set F found by the weighted matroid intersection algorithm has cost $\leq 2c(E_{opt})$. Moreover, $c(E') \leq c(F)$, so $c(E') \leq 2c(E_{opt})$.

Theorem 9.2 *There is a 2-approximation algorithm for the minimum cost k-ECSS problem. The running time is $O(kn \log n(m + n \log n))$.*

9.4 An $O(1)$-approximation algorithm for minimum metric cost k-NCSS

Let $G = (V, E)$ be a graph of node connectivity $\geq k$, and let the edge costs $c : E \rightarrow \Re_+$ form a metric, i.e., the edge costs satisfy the triangle inequality, $c(vw) \leq c(vx) + c(xw)$, for every ordered triple of nodes v, w, x. This section gives an algorithm that finds a k-NCSS $G' = (V, E')$ such that the cost $c(E') = \sum_{vw \in E'} c(vw)$ is at most $(2 + (2k/n))c(E_{opt})$, where E_{opt} denotes the edge set of a minimum-cost k-NCSS. This result is due to Khuller & Raghavachari [29], and it is based on an algorithm of Frank & Tardos [14] for finding an optimal solution to the following problem. Given a directed graph $D = (V, A)$ with arc costs $c : A \to \Re_+$, and a node $v_0 \in V$, find a minimum-cost arc set $F \subseteq A$ such that (V, F) has k openly-disjoint directed paths from v_0 to w, for each node $w \in V \backslash \{v_0\}$. Gabow [16] has given an implementation of the Frank-Tardos algorithm that runs in time $O(k^2|V|^2|A|)$.

The k-NCSS algorithm first modifies the given undirected graph G by adding a "root" node v_0. For this, we examine all nodes $v \in V$ to find a node v_1 such that the total cost of the cheapest $k-1$ edges incident to v_1 is minimum possible. Let $v_2, \ldots, v_k$ be $k-1$ neighbors of v_1 such that $\sum_{i=2}^{k} c(v_1v_i)$ gives this minimum. We add a new node v_0 to G, together with the edges $v_0v_1, v_0v_2, \ldots, v_0v_k$, and we assign each new edge a cost of zero (the edge costs may no longer form a metric, but this does not matter). Let $D = (V \cup \{v_0\}, A)$ be the directed graph of the resulting undirected graph $(V \cup \{v_0\}, E \cup \{v_0v_1, \ldots, v_0v_k\})$. The arc costs of D are assigned by taking $c(v, w) = c(w, v) = c(vw)$ for every edge vw in the graph. We apply the Frank-Tardos algorithm to D, c, v_0, to find a minimum-cost arc set $F \cup \{(v_0, v_1), \ldots, (v_0, v_k)\}$ such that $(V \cup \{v_0\}, F \cup \{(v_0, v_1), \ldots, (v_0, v_k)\})$ has k openly-disjoint directed paths from v_0 to w, for each $w \in V$. We obtain a k-NCSS $G' = (V, E')$ by taking the undirected graph of (V, F) and for $1 \leq i < j \leq k$, adding the edge v_iv_j if it is not already present, i.e., G' is the "union" of the undirected graph of (V, F) and a clique on the nodes $v_1, \ldots, v_k$. (Note that G' is a simple graph.)

Suppose that G' is not k-node connected. Then G' has a $(k-1)$-separator S, i.e., there is a node set S with $|S| \leq k-1$ such that $G' \backslash S$ has ≥ 2 components. All the nodes in $\{v_1, \ldots, v_k\} \backslash S$ must be in the same component since G' has a clique on $v_1, \ldots, v_k$. Moreover, each node $w \in V$ has k paths to $v_1, \ldots, v_k$ such that these paths have only the node w in common; to see this, focus on the k openly-disjoint directed paths from v_0 to w in the directed graph $(V \cup \{v_0\}, F \cup \{(v_0, v_1), \ldots, (v_0, v_k)\})$. For every node $w \in V \backslash S$, at least one of these k paths is (completely) disjoint from S. Therefore, in $G' \backslash S$, every node $w \in V \backslash S$ has a path to some node in $\{v_1, \ldots, v_k\}$. This shows that $G' \backslash S$ is connected, and contradicts our assumption that S is a separator of G'. Consequently, G' is k-node connected.

Consider the total edge cost of G', $c(E')$. Reasoning as in Section 9.3, note that $c(F) \leq 2c(E_{opt})$. (In detail, the directed graph of $(V \cup \{v_0\}, E_{opt} \cup \{(v_0, v_1), \ldots, (v_0, v_k)\})$ has cost $2c(E_{opt})$, and the arc set of this directed graph gives a feasible solution for the problem solved by the Frank-Tardos algorithm; hence, the optimal arc set F found by the Frank-Tardos algorithm has cost $\leq 2c(E_{opt})$.) Let c^* denote the total cost of the $k-1$ cheapest edges incident to v_1, i.e., $c^* = \sum_{i=2}^{k} c(v_1v_i)$. Now consider the total edge cost of the clique on $v_1, \ldots, v_k$. Since each edge v_iv_j (for $1 \leq i < j \leq k$) has $c(v_iv_j) \leq c(v_1v_i) + c(v_1v_j)$, it can be seen that $\sum_{1 \leq i < j \leq k} c(v_iv_j) \leq (k-1)c^*$. For each node $v \in V$, let $\delta_{opt}(v)$ denote the set of edges of E_{opt} incident to v; clearly, $|\delta_{opt}(v)| \geq k$, $\forall v \in V$. By our choice of v_1 and $v_2, \ldots, v_k$, each node $v \in V$ has $c(\delta_{opt}(v)) = \sum_{vw \in \delta_{opt}(v)} c(vw) \geq kc^*/(k-1)$. Since $\sum_{v \in V} c(\delta_{opt}(v)) = 2c(E_{opt})$, we have $c^* \leq 2(k-1)c(E_{opt})/(kn)$. Hence, $\sum_{1 \leq i < j \leq k} c(v_iv_j) \leq 2(k-1)^2 c(E_{opt})/(kn)$. Summarizing, we have $c(E') \leq c(F) + \sum_{1 \leq i < j \leq k} c(v_iv_j) \leq (2 + 2(k-1)^2/(kn))c(E_{opt})$.

Theorem 9.3 *Given a graph G and metric edge costs c, there is a $(2 + (2k/n))$-approximation* *algorithm for finding a minimum-cost k-NCSS. The running time is $O(k^2n^2m)$.*

9.5 2-Approximation algorithms for minimum-size k-CSS

In this section, we focus on the minimum-size k-CSS problem (note that every edge has unit cost) and sketch simple 2-approximation algorithms. Then, in preparation for algorithms with better approximation guarantees, we give an example that illustrates the difficulty in improving on the 2-approximation guarantee for minimum-size k-CSS problems.

A graph H is called *edge minimal* with respect to a property $\mathcal{P}$ if H possesses $\mathcal{P}$, but for every edge e in H, $H \backslash e$ does *not* possess $\mathcal{P}$. Thus, if a k-edge connected graph G is edge minimal, then for every edge $e \in E(G)$, $G \backslash e$ has a $(k-1)$-cut. Similarly, if a k-node connected graph G is edge minimal, then for every edge $e \in E(G)$, $G \backslash e$ has a $(k-1)$-separator.

The proof of the next proposition is sketched in the exercises, see Exercise 1

Proposition 9.4 (Mader [33, 34]) *(1) If a k-edge connected graph is edge minimal, then the number of edges is $\leq kn$.*

(2) If a k-node connected graph is edge minimal, then the number of edges is $\leq kn$.

Parts (1) and (2) of this proposition immediately give 2-approximation algorithms for the minimum-size k-ECSS problem and the minimum-size k-NCSS problem, respectively. Here is the k-NCSS approximation algorithm; we skip the k-ECSS approximation algorithm since it is similar. Assume that the given graph $G = (V, E)$ is k-node connected, otherwise, the approximation algorithm will detect this and report failure. We start by taking $E' = E$. At termination, E' will be the edge set of the approximately minimum-size k-NCSS. We examine the edges in an arbitrary order $e_1, e_2, \ldots, e_m$ (where $E = \{e_1, e_2, \ldots, e_m\}$). For each edge e_i (for $1 \leq i \leq m$) we test whether or not the subgraph $(V, E' \backslash e_i)$ is k-node connected. If yes, then the edge e_i is not essential for k-node connectivity, so we update E' by removing e_i from E', otherwise (i.e., if $(V, E' \backslash e_i)$ is not k-node connected), we retain e_i in E'. At termination, (V, E') will be an edge-minimal k-NCSS because whenever we retain an edge in E' then that edge is critical w.r.t. k-node connectivity. The approximation guarantee of 2 follows because every k-NCSS has $\geq kn/2$ edges, whereas $|E'| \leq kn$ by the proposition. The approximation algorithm runs in polynomial time, but is not particularly efficient, since it executes $|E|$ tests for k-node connectivity. Simple and fast 2-approximation algorithms for the minimum-size k-CSS problem are now available, yet the simplicity of the proofs for the above approximation algorithm is an advantage.

Another easy and efficient method for finding a k-CSS with $\leq kn$ edges follows from results of Nagamochi & Ibaraki [36] and follow-up papers. A k-ECSS (V, E') with $|E'| \leq kn$ can be found as follows (assume that G is k-edge connected): we take E' to be the union of (the edge sets of) k edge-disjoint forests $F_1, \ldots, F_k$, where each F_i (for $1 \leq i \leq k$) is the edge set of a maximal but otherwise arbitrary spanning forest of $G \backslash (F_1 \cup \ldots \cup F_{i-1})$. In more detail, we take F_1 to be the edge set of an arbitrary spanning tree of G. Then, we delete all edges in F_1 from G. The resulting graph $G \backslash (F_1)$ may have several connected components. In general, we take F_i (for $2 \leq i \leq k$) to be the union of the edge sets of spanning trees of each of the components of $G \backslash (F_1 \cup \ldots \cup F_{i-1})$. The next result is due to [36] and Thurimella [39], independently.

Proposition 9.5 *If $G = (V, E)$ is k-edge connected, then the subgraph (V, E') is also k-edge connected, where $E' = F_1 \cup \ldots \cup F_k$ and F_i $(1 \le i \le k)$ is the edge set of a maximal spanning forest of $G \backslash (F_1 \cup \ldots \cup F_{i-1})$.*

Proof: Suppose that (V, E') is not k-edge connected. Then it has a cut $\delta'(S)$ of cardinality $\le k-1$. Since G is k-edge connected, there must be an edge vw in G such that $vw \notin E'$ and $v \in S$, $w \notin S$ (i.e., $vw \in \delta_G(S)$). For $i = 1, \ldots, k$, note that $vw \notin F_i$ implies that F_i has a v-w path (otherwise, adding vw to F_i gives a forest of larger size). Clearly, the v-w paths in $F_1, \ldots, F_k$ are edge disjoint. This is a contradiction since G' has both k edge disjoint v-w paths and a $k-1$ cut separating v and w. ∎

Obviously, $|E'| \le k(n-1)$. Consequently, the k-ECSS found by this algorithm has size within a factor of 2 of minimum. The obvious implementation of this algorithm runs in time $O(km)$. Nagamochi & Ibaraki [36] give a linear-time implementation for this algorithm.

In fact, Nagamochi & Ibaraki [36] show that the maximal forests $F_1, \ldots, F_k$ computed by their algorithm are such that the subgraph (V, E') is k-node connected if G is k-node connected, where $E' = F_1 \cup \ldots \cup F_k$. A *scan-first-search* spanning forest with edge set F is constructed as follows: Initially, $F = \emptyset$. An arbitrary node v_1 is chosen and *scanned.* This may add some edges to F. Then repeatedly an unscanned node is chosen and scanned, until all nodes are scanned. If the current F is incident to one or more unscanned nodes, then any such node may be chosen for scanning, otherwise, an arbitrary unscanned node is chosen. When a node v is scanned, all edges in $E \backslash F$ incident to v are examined; if the addition of an edge vw to F will create a cycle in F (i.e., if F already has a v-w path), then the edge is rejected, otherwise vw is added to F. The next result is due to Nagamochi & Ibaraki [36]. Other proofs are given in [13, 3]. We skip the proof.

Proposition 9.6 *If $G = (V, E)$ is k-node connected, then the subgraph (V, E') is also k-node connected, where $E' = F_1 \cup \ldots \cup F_k$ and F_i $(1 \le i \le k)$ is the edge set of a maximal scan-first-search spanning forest of $G \backslash (F_1 \cup \ldots \cup F_{i-1})$.*

It follows that the algorithm in [36] is a linear-time 2-approximation algorithm for the minimum-size k-NCSS problem.

9.5.1 An illustrative example

Here is an example illustrating the difficulty in improving on the 2-approximation guarantee for minimum-size k-CSS problems. Let the given graph G have n nodes, where n is even. Suppose that the edge set of G, $E(G)$, is the union of the edge set of the complete bipartite graph $K_{k,(n-k)}$ and the edge set E_{opt} of an n-node, k-regular, k-edge connected (or k-node connected) graph. For example, for $k = 2$, $E(G)$ is the union of $E(K_{2,(n-2)})$ and the edge set of a Hamiltonian cycle. A naive heuristic may return $E(K_{k,(n-k)})$ which has size $k(n-k)$, roughly two times $|E_{opt}|$. A heuristic that significantly improves on the 2-approximation guarantee must somehow return many edges of E_{opt}.

9.6 Khuller and Vishkin's 1.5-approximation algorithm for minimum size 2-ECSS

This section describes a simple and elegant algorithm of Khuller & Vishkin [30] for finding a 2-ECSS (V, E') of a graph $G = (V, E)$ such that $|E'| \leq 1.5|E_{opt}|$, where E_{opt} is the edge set of a minimum size 2-ECSS. Assume that the given graph $G = (V, E)$ is 2-edge connected. Khuller & Vishkin's algorithm is based on dfs (depth-first search). (The relevant facts about dfs are summarized below.) We use T to denote the dfs tree as well as its edge set. The subtree of T rooted at a node v is denoted by $T(v)$. For notational convenience, we identify the nodes with their dfs numbers, i.e., $v < w$ means that v precedes w in the dfs traversal (or preorder traversal) of T. For a node v, the *deepest backedge* emanating from $T(v)$ is denoted $db(v)$, i.e., $db(v) = wx$, where wx is a backedge, w is a node of $T(v)$, and for every backedge uy with u in $T(v)$, $x \leq y$.

We initialize E' to be the edge set of the dfs tree, T. Then we make a dfs traversal of T, and when backing up over an edge uv in T (at this point the algorithm has already completed a dfs traversal of $T(v)$) we check whether uv is a cutedge of the current subgraph (V, E'). If yes, then we add $db(v)$ to E', otherwise, we keep the same E'.

At termination, (V, E') is a 2-ECSS of G because there are no cutedges in (V, E'). To see this, note that G has no cut edges, and so every edge $uv \in T$ has a well-defined backedge $db(v)$ such that $x \leq u$, where x is the end node of $db(v)$ that is not in $T(v)$. In other words, if $uv \in T$ is a cutedge of the current subgraph (V, E'), then we will "cover" uv with a backedge wx such that w is in $T(v)$ and $x \leq u$.

The key result for proving the 1.5 approximation guarantee is this:

Proposition 9.7 *For every pair of nodes v_i and v_j such that the algorithm adds backedges $db(v_i)$ and $db(v_j)$ to E', the cuts $\delta(T(v_i))$ and $\delta(T(v_j))$ have no edges in common.*

Proof: Let v_i precede v_j in the dfs traversal. Let $db(v_i) = wx$ and let $db(v_j) = yz$. Either v_i is an ancestor of v_j, or there is a node v with children v_1 and v_2 such that v_i is a descendant of v_1 and v_j is a descendant of v_2. In the first case, $v_i \leq z$ (i.e., $u_i v_i \in T$ is not "covered" by the backedge $db(v_j)$, where u_i is the parent of v_i in T), and so every edge in the cut $\delta(T(v_j))$ has both end nodes in $T(v_i)$; hence, the two cuts $\delta(T(v_i))$ and $\delta(T(v_j))$ are edge disjoint. In the second case, the proposition follows immediately. ∎

Theorem 9.8 *Let $G = (V, E)$ be a 2-edge connected graph, and let E_{opt} be the edge set of a minimum-size 2-ECSS. There is a linear-time algorithm to find a 2-ECSS (V, E') such that $|E'| \leq 1.5|E_{opt}|$.*

Proof: It is easily checked that the algorithm runs in linear time. Consider the approximation guarantee. Clearly, $|E_{opt}| \geq n$, since every node is incident to ≥ 2 edges of E_{opt}. We need another lower bound on $|E_{opt}|$. Let $v_1, v_2, \ldots, v_p$ denote all the nodes such that the algorithm adds the backedge $db(v_i)$ (for $i = 1, \ldots, p$) to E', i.e., $E' = T \cup \{db(v_1), \ldots, db(v_p)\}$. Since the cuts $\delta(T(v_1)), \ldots, \delta(T(v_p))$ are mutually edge disjoint, and E_{opt} has at least two edges in each of these cuts, we have $|E_{opt}| \geq 2p$. Hence, $|E_{opt}| \geq \max(n, 2p)$. Since $|E'| = (n-1) + p$, we have

$$\frac{|E'|}{|E_{opt}|} \leq \frac{n-1}{n} + \frac{p}{2p} \leq 1.5.$$

9.7 Mader's theorem and a 1.5-approximation algorithm for minimum size 2-NCSS

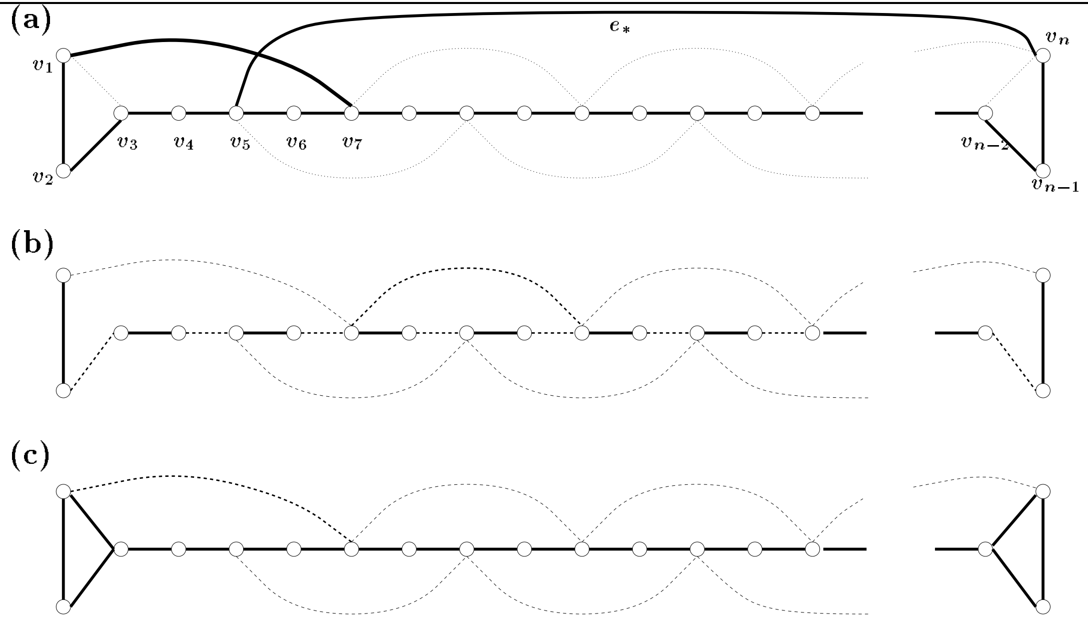


Figure 9.1: Illustrating the 2-NCSS heuristic on a 2-node connected graph $G = (V, E)$; $n = |V|$ is even, and $k = 2$. Adapted from Garg, Santosh & Singla [20, Figure 7].
(a) A minimum-size 2-node connected spanning subgraph has $n + 1$ edges, and is indicated by thick lines (the path $v_1, v_2, \ldots, v_n$ and edges v_1v_7 and $e_* = v_5v_n$).
(b) The first step of the heuristic in Section 9.7 finds a minimum-size $M \subseteq E$ such that every node is incident to $\geq (k-1) = 1$ edges of M. The thick lines indicate M; it is a perfect matching. The second step of the heuristic finds an (inclusionwise) minimal edge set $F \subseteq E$ such that $(V, M \cup F)$ is 2-node connected. F is indicated by dashed lines – the "key edge" e_* is not chosen in F. $|M \cup F| = 1.5n - 5$.
(c) Another variant of the heuristic first finds a minimum-size $M \subseteq E$ such that every node is incident to $\geq k = 2$ edges of M. The thick lines indicate M (M is the path $v_1, v_2, \ldots, v_n$ and edges v_1v_3, $v_{n-2}v_n$). The second step of the heuristic finds the edge set $F \subseteq E$ indicated by dashed lines – the "key edge" e_* is not chosen in F. $(V, M \cup F)$ is 2-node connected, and for every edge vw in F, $(V, M \cup F)\backslash vw$ is *not* 2-node connected. $|M \cup F| = 1.5n - 3$.

This section focuses on the design of a 1.5-approximation algorithm for finding a minimum-size 2-NCSS. The analysis of the 1.5-approximation guarantee hinges on a deep theorem due to Mader. Section 9.8 has a straightforward generalization (from $k = 2$ to an arbitrary integer $k \geq 2$) of the algorithm and its analysis for finding a k-NCSS with an approximation guarantee of $1 + (2/k)$. A more careful analysis improves the approximation guarantee of the generalized algorithm to $1+(1/k)$; we sketch this but skip the proof of a key theorem. Although the analysis of approximation guarantee relies on Mader's theorem only and not its proof, a proof of Mader's theorem is given in Section 9.9.

The running time of the approximation algorithm for 2-NCSS is $O(m\sqrt{n})$, because it uses a subroutine for maximum cardinality matching, and the fastest maximum matching algorithm known has this running time. Given a constant $\epsilon > 0$, the approximation algorithm for 2-NCSS can be modified to run in linear time but the approximation guarantee becomes $(1.5 + \epsilon)$. Also, the linear-time variant uses a linear-time algorithm of Han et al [23] for finding an edge minimal 2-NCSS. The first algorithm to achieve an approximation guarantee of 1.5 for finding a minimum-size 2-NCSS is due to Garg et al [20]; moreover, this algorithm runs in linear time. The Garg et al algorithm may be easier to implement and it may run faster in practice, but the analysis of the approximation guarantee is more sophisticated and specialized than the analysis in this section. We do not describe the algorithm of Garg et al, but instead refer the interested reader either to [20] or to the survey paper by Khuller [31].

Assume that the given graph $G = (V, E)$ is 2-node connected. The algorithm for approximating a minimum-size 2-NCSS consists of two steps.

The first step finds a minimum edge cover $M \subseteq E$ of G. An *edge cover* of G is a set of edges $X \subseteq E$ such that every node of G is incident with some edge in X. An edge cover of minimum cardinality is called a *minimum edge cover*. One way of finding a minimum edge cover M is to start with a *maximum matching* $\widetilde{M}$ of G, and then to add one edge incident to each node that is not matched by $\widetilde{M}$. Clearly, M is an edge cover. Let $\operatorname{def}(G)$ denotes the number of nodes not matched by a maximum matching of G, i.e., $\operatorname{def}(G) = |V| - 2|\widetilde{M}|$. Then we have $|M| = |\widetilde{M}| + \operatorname{def}(G)$. We leave it as an exercise for the reader that every edge cover of G has cardinality $\geq |\widetilde{M}| + \operatorname{def}(G)$, hence, M is in fact a minimum edge cover. (Hint: for an edge cover X, let q be the minimum number of edges to remove from X to obtain a matching; now focus on $|X|$ and q.)

The second step is equally simple. We find an (inclusionwise) minimal edge set $F \subseteq E \backslash M$ such that $M \cup F$ gives a 2-NCSS. In other words, $(V, M \cup F)$ is 2-node connected, but for each edge $vw \in F$, $(V, M \cup F) \backslash vw$ is *not* 2-node connected. An edge vw of a 2-node connected graph H is *critical* (w.r.t. 2-node connectivity) if $H \backslash vw$ is *not* 2-node connected. The next result characterizes critical edges; for a generalization see Proposition 9.15.

Proposition 9.9 *An edge vw of a 2-node connected graph H is not critical iff there are at least 3 openly disjoint v-w paths in H (including the path vw).*

Proof: If H has exactly two openly disjoint v-w paths, then vw is obviously a critical edge since $H \backslash vw$ has a cut node (since $H \backslash vw$ does not have two openly disjoint v-w paths). For the other part, suppose that H has ≥ 3 openly disjoint v-w paths. By way of contradiction, let c be a cut node of $H \backslash vw$, i.e., let $S = \{c\}$ be a 1-separator of $H \backslash vw$. Nodes v and w must be in the same component of the graph H' obtained by deleting S from $H \backslash vw$ (since $H \backslash vw$ has $\geq 2 > |S|$ openly disjoint v-w paths). This gives a contradiction, because adding the edge vw to H' gives a disconnected graph $H' + vw$ (since the new edge joins two nodes in the same component), but $H' + vw = H \backslash S$, and $H \backslash S$ must be a connected graph, since H is 2-node connected and $|S| = 1$. ∎

To find F efficiently, we start with $F = \emptyset$ and take the current subgraph to be $G = (V, E)$ (which is 2-node connected). We examine the edges of $E \backslash M$ in an arbitrary order, say, $e_1, e_2, \ldots, e_\ell$ ($\ell = |E \backslash M|$). For each edge $e_i = v_i w_i$, we attempt to find 3 openly disjoint v_i-w_i paths in the current subgraph. If we succeed, then we remove the edge e_i from the current subgraph (since e_i is not critical), otherwise, we retain e_i in the current subgraph and add e_i to F (since e_i is critical). At termination, the current subgraph with edge set $M \cup F$ is 2-node connected, and every edge $vw \in F$ is critical. The running time for the second step is $O(m^2)$.

Let E' denote $M \cup F$, and let $E_{opt} \subseteq E$ denote a minimum-cardinality edge set such that (V, E_{opt}) is 2-*edge* connected.

Our proof of the 1.5-approximation guarantee hinges on a theorem of Mader [34, Theorem 1]. A proof of Mader's theorem appears in Section 9.9. For another proof of Mader's theorem see Lemma I.4.4 and Theorem I.4.5 in [1]. Recall that an edge vw of a k-node connected graph H is called critical (w.r.t. k-node connectivity) if $H \backslash vw$ is *not* k-node connected.

Theorem 9.10 (Mader [34, Theorem 1]) *In a k-node connected graph, a cycle consisting of critical edges must be incident to at least one node of degree k.*

Lemma 9.11 $|F| \leq n - 1$.

Proof: Consider the 2-node connected subgraph returned by the heuristic, $G' = (V, E')$, where $E' = M \cup F$. Suppose that F contains a cycle C. Note that every edge in the cycle is critical, since every edge in F is critical. Moreover, every node v incident to the cycle C has degree ≥ 3 in G', because v is incident to two edges of C, as well as to at least 1 edge of $M = E' \backslash F$. But this contradicts Mader's theorem. We conclude that F is acyclic, and so has $\leq n - 1$ edges. The proof is done. ∎

Lemma 9.12 $|E'| = |M| + |F| \leq 1.5n + \text{def}(G) - 1$.

Proof: By the previous lemma, $|F| \leq n - 1$. A minimum edge cover M of G has size $|M| = |\widetilde{M}| + \text{def}(G)$, where $\widetilde{M}$ is a maximum matching of G. Obviously, $|\widetilde{M}| \leq n/2$. The result follows. ∎

The next result, due to Chong and Lam, gives a lower bound on the size of a 2-ECSS.

Proposition 9.13 (Chong & Lam [5, Lemma 3]) *Let $G = (V, E)$ be a graph of edge connectivity ≥ 2, and let $|E_{opt}|$ denote the minimum size of a 2-ECSS.*
Then $|E_{opt}| \geq \max(n + \text{def}(G) - 1, n)$.

Proof: Consider a closed ear decomposition of (V, E_{opt}), i.e., a partition of E_{opt} into paths and cycles $P_1, P_2, \ldots, P_q$ such that P_1 is a cycle, and each P_i (for $2 \leq i \leq q$) has its end nodes but no internal nodes in common with $P_1 \cup \ldots \cup P_{i-1}$ (the end nodes of P_i may coincide). By the minimality of E_{opt}, each P_i contains at least two edges, i.e., there are no single-edge ears. Clearly, $|E_{opt}| = q + n - 1$, where q is the number of ears in the decomposition. By deleting one edge of P_1, and the first and the last edge of each P_i ($i \geq 2$), we obtain a partition of V into completely disjoint paths. Each of these disjoint paths has a matching such that at most one node is not matched. Taking the union of these matchings, we obtain a matching of (V, E_{opt}) such that at most q nodes are not matched. Clearly, $q \geq \text{def}(G)$, since $\text{def}(G)$ is the number of nodes not matched by a *maximum* matching of $G = (V, E)$. Hence, $|E_{opt}| \geq \text{def}(G) + n - 1$. ∎

Theorem 9.14 *Let $G = (V, E)$ be a graph of node connectivity ≥ 2. The heuristic described above finds a 2-NCSS (V, E') such that $|E'| \leq 1.5|E_{opt}|$, where $|E_{opt}|$ denotes the minimum size of a 2-ECSS. The running time is $O(m\sqrt{n})$.*

Let $\epsilon > 0$ be a constant. A sequential linear-time version of the heuristic achieves an approximation guarantee of $(1.5 + \epsilon)$.

Proof: The approximation guarantee follows from Lemma 9.12 and Proposition 9.13, since

$$\frac{|E'|}{|E_{opt}|} \leq \frac{1.5n + \text{def}(G) - 1}{\max(n + \text{def}(G) - 1, n)} \leq 1 + \frac{0.5n}{n} \leq 1.5.$$

Step 1 can be implemented to run in $O(m\sqrt{n})$ time, since a maximum matching can be computed within this time bound. The obvious implementation of Step 2 takes $O(m^2)$ time, but this can be improved to $O(n+m)$ time by using the algorithm of Han et al [23]. Thus the overall running time is $O(m\sqrt{n})$.

Consider the variant of the algorithm that runs in linear time. Let $\widetilde{M}$ denote a maximum matching of G. For Step 1, we find an approximately maximum matching. For a constant ϵ, $0 < \epsilon < 0.5$, the algorithm finds a matching M' with $|M'| \geq (1 - 2\epsilon)|\widetilde{M}|$ in $O((n+m)/\epsilon)$ time. We obtain an (inclusionwise) minimal edge cover M of size $\leq (1 + 2\epsilon)|\widetilde{M}| + \text{def}(G)$ by adding to M' one edge incident to every node that is not matched by M'. Moreover, in linear time, we can find an edge minimal 2-NCSS whose edge set contains the minimal edge cover M, see [23]. Now, the approximation guarantee is $(1.5 + \epsilon)$. ∎

9.8 A $(1 + \frac{1}{k})$-approximation algorithm for minimum-size k-NCSS

This section presents the heuristic for finding an approximately minimum-size k-NCSS, and proves an approximation guarantee of $1+(1/k)$. The analysis of the heuristic hinges on a theorem of Mader [34, Theorem 1], see Theorem 9.10. Given a graph $G = (V, E)$, a straightforward application of Mader's theorem shows that the number of edges in the k-NCSS returned by the heuristic is at most

$$(n - 1) + \min\{|M| \,:\, M \subseteq E \text{ and } \deg_M(v) \geq (k - 1), \forall v \in V\},$$

see Lemma 9.16 below. An approximation guarantee of $1+(2/k)$ on the heuristic follows, since the number of edges in a k-node connected graph is at least $kn/2$, by the "degree lower bound", see Proposition 9.17. Often, the key to proving improved approximation guarantees for (minimizing) heuristics is a nontrivial lower bound on the value of every solution. We improve the approximation guarantee from $1 + (2/k)$ to $1 + (1/k)$ by exploiting a new lower bound on the size of a k-edge connected spanning subgraph, see Theorem 9.18:

> The number of edges in a k-edge connected spanning subgraph of a graph $G = (V, E)$ is at least $\lfloor n/2 \rfloor + \min\{|M| \,:\, M \subseteq E \text{ and } \deg_M(v) \geq (k - 1), \forall v \in V\}$.

Assume that the given graph $G = (V, E)$ is k-node connected, otherwise, the heuristic will detect this and report failure.

Let $E^* \subseteq E$ denote a minimum-cardinality edge-set such that the spanning subgraph (V, E^*) is k-*edge* connected. Note that every k-node connected spanning subgraph (V, E') (such as the optimal solution) is necessarily k-edge connected, and so has $|E'| \geq |E^*|$.

We need a few facts on b-matchings, because the k-NCSS approximation algorithm uses a subroutine for maximum b-matchings. Let $G = (V, E)$ be a graph, and let $b : V \to \mathbf{Z}_+$ assign a nonnegative integer b_v to each node $v \in V$. The perfect b-matching (or perfect degree-constrained subgraph) problem is to find an edge set $M \subseteq E$ such that each node v has $\deg_M(v) = b_v$. The maximum b-matching (or maximum degree-constrained subgraph) problem is to find a maximum-cardinality $M \subseteq E$ such that each node v has $\deg_M(v) \leq b_v$. The b-matching problem can be solved in time $O(m^{1.5}(\log n)^{1.5}\sqrt{\alpha(m,m)})$, see [18, Section 11] (for our version of the problem, note that each edge has unit cost and unit capacity, and each node v may be assumed to have $0 \leq b_v \leq \deg(v)$). Also, see [21, Section 7.3].

The heuristic has two steps. The first finds a *minimum-size* spanning subgraph (V, M), $M \subseteq E$, whose minimum degree is $(k-1)$, i.e., each node is incident to $\geq (k-1)$ edges of M. Clearly, $|M| \leq |E^*|$, because (V, E^*) has minimum degree k, i.e., every node is incident to $\geq k$ edges of E^*. To find M efficiently, we use the algorithm for the maximum b-matching problem. Our problem is:

$$\min\{|M| : \deg_M(v) \geq (k-1),\ \forall v \in V, \text{ and } M \subseteq E\}.$$

To see that this is a b-matching problem, consider the equivalent problem of finding the complement $\overline{M}$ of M w.r.t. E, where $\overline{M} = E \backslash M$:

$$\max\{|\overline{M}| : \deg_{\overline{M}}(v) \leq \deg(v) + 1 - k,\ \forall v \in V, \text{ and } \overline{M} \subseteq E\}.$$

The second step is equally simple. We find an (inclusionwise) minimal edge set $F \subseteq E \backslash M$ such that $M \cup F$ gives a k-node connected spanning subgraph, i.e., $(V, M \cup F)$ is k-node connected and for each edge $vw \in F$, $(V, M \cup F) \backslash vw$ is *not* k-node connected. Recall that an edge vw of a k-node connected graph H is *critical* (w.r.t. k-node connectivity) if $H \backslash vw$ is *not* k-node connected. The next result characterizes critical edges.

Proposition 9.15 . *An edge vw of a k-node connected graph H is not critical iff there are at least $k+1$ openly disjoint v-w paths in H (including the path vw).*

To find F efficiently, we start with $F = \emptyset$ and take the current subgraph to be $G = (V, E)$ (which is k-node connected). We examine the edges of $E \backslash M$ in an arbitrary order, say, $e_1, e_2, \ldots, e_\ell$ ($\ell = |E \backslash M|$). For each edge $e_i = v_i w_i$, we attempt to find $(k+1)$ openly disjoint v_i-w_i paths in the current subgraph. If we succeed, then we remove the edge e_i from the current subgraph (since e_i is not critical), otherwise, we retain e_i in the current subgraph and add e_i to F (since e_i is critical). At termination, the current subgraph with edge set $M \cup F$ is k-node connected, and every edge $vw \in F$ is critical. The running time for the second step is $O(km^2)$.

The proof of the next lemma hinges on a theorem of Mader [34, Theorem 1], see Theorem 9.10. The proof is similar to the proof of Lemma 9.11 and so is omitted.

Lemma 9.16 *$|F| \leq n - 1$.*

Proposition 9.17 *Let $G = (V, E)$ be a graph of node connectivity $\geq k$. The heuristic above finds a k-NCSS (V, E') such that $|E'| \leq (1 + (2/k))|E_{opt}|$, where $|E_{opt}|$ denotes the cardinality of an optimal solution. The running time is $O(k^3n^2 + m^{1.5}(\log n)^2)$.*

Proof: The approximation guarantee follows because $|E_{opt}| \geq (kn/2)$, so

$$\frac{|M| + |F|}{|E_{opt}|} = \frac{|M|}{|E_{opt}|} + \frac{|F|}{|E_{opt}|} \leq 1 + \frac{n}{(kn/2)} = 1 + (2/k).$$

We have already seen that M can be found in time $O(m^{1.5}(\log n)^2)$ via the maximum b-matching algorithm, and F can be found in time $O(km^2)$. The running time of the second step can be improved to $O(k^3n^2)$; this is left as an exercise. ▮

To improve the approximation guarantee to $1 + (1/k)$, we present an improved lower bound on $|E^*|$, where E^* denotes a minimum-cardinality edge set such that $G^* = (V, E^*)$ is k-edge connected. Suppose that E^* contains a perfect matching P_0 (so $|P_0| = n/2$). Then $|E^*| \geq (n/2) + \min\{|M^*| \; : \; M^* \subseteq E, \; \deg_{M^*}(v) \geq (k-1), \forall v \in V\}$. To see this, focus on the edge set $M' = E^* \backslash P_0$. Clearly, every node $v \in V$ is incident to at least $(k-1)$ edges of M', because $\deg_{E^*}(v) \geq k$ and $\deg_{P_0}(v) = 1$. Since M^* is a minimum-size edge set with $\deg_{M^*}(v) \geq (k-1)$, $\forall v \in V$, we have $|M^*| \leq |M'| = |E^*| - (n/2)$. The next theorem generalizes this lower bound to the case when E^* has no perfect matching. We skip the proof.

Theorem 9.18 *Let $G^* = (V, E^*)$ be a graph of edge connectivity $\geq k \geq 1$, and let n denote $|V|$. Let $M^* \subseteq E^*$ be a minimum-size edge set such that every node $v \in V$ is incident to $\geq (k-1)$ edges of M^*. Then $|E^*| \geq |M^*| + \lfloor n/2 \rfloor$.*

Theorem 9.19 *Let $G = (V, E)$ be a graph of node connectivity $\geq k$. The heuristic described above finds a k-NCSS (V, E') such that $|E'| \leq (1 + (1/k))|E_{opt}|$, where $|E_{opt}|$ denotes the cardinality of an optimal solution. The running time is $O(k^3n^2 + m^{1.5}(\log n)^2)$.*

Proof: The approximation guarantee of $1 + (1/k)$ follows easily from Theorem 9.18, using an argument similar to Proposition 9.17. We have $E' = M \cup F$, where $|F| \leq (n-1)$. Moreover, since M is a minimum-size edge set with $\deg_M(v) \geq (k-1)$, $\forall v \in V$, Theorem 9.18 implies that $|M| \leq |E_{opt}| - \lfloor n/2 \rfloor \leq |E_{opt}| - (n-1)/2$. Hence,

$$\frac{|M| + |F|}{|E_{opt}|} \leq \frac{|E_{opt}| - (n-1)/2 + (n-1)}{|E_{opt}|} \leq 1 + \frac{n/2}{|E_{opt}|} \leq 1 + (1/k),$$

where the last inequality uses the "degree lower bound", $|E_{opt}| \geq kn/2$.

The running time analysis is the same as that in Proposition 9.17. ▮

9.9 Mader's theorem

This section has Mader's original proof of Theorem 9.10; no other proof of this theorem is known. Recall that an edge vw of a k-node connected graph G is called critical if $G \backslash vw$ is not k-node connected. In other words, vw is critical if $G\backslash vw$ has a separator of cardinality $< k$, i.e., if there exists a set S with $|S| \leq k-1$ such that $(G\backslash vw)\backslash S$ is disconnected. Note that this graph has precisely two components, one containing v and the other containing w, because by adding the edge vw to this graph we obtain the *connected* graph $G\backslash S$ (since G is k-node connected and $|S| < k$). This observation is used several times in the proof.

We repeat the statement of Mader's theorem, see Theorem 9.10.

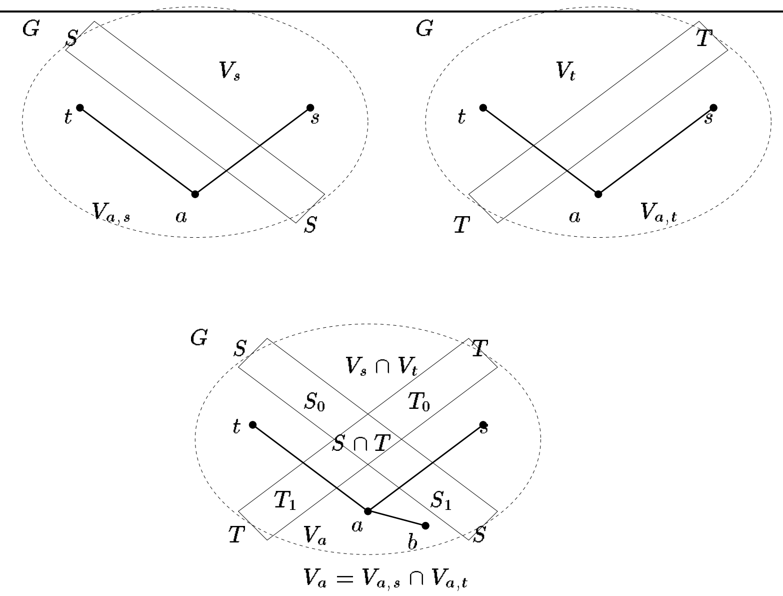


Figure 9.2: An illustration of the proof of Mader's theorem.

Theorem (Mader) *In a k-node connected graph, a cycle consisting of critical edges must be incident to at least one node of degree k.*

Proof: Let $G = (V, E)$ be a k-node connected graph. By way of contradiction, let $C = a_0, a_1, \ldots, a_{\ell-1}, a_0$ be a cycle such that each edge is critical. Suppose that $\deg(a_0)$ is $\geq k+1$. For notational convenience, let $a = a_0$, $s = a_1$ and $t = a_{\ell-1}$. In the graph $G\backslash as$, let S be an arbitrary $(k-1)$-separator whose deletion results in two components (S exists because edge as is critical for G), and let $V_{a,s}$ and V_s denote (the node sets of) the two components, where $a \in V_{a,s}$ and $s \in V_s$. Similarly, let $V_{a,t}$ and V_t denote (the node sets of) the two components of $(G\backslash at)\backslash T$, where T is an arbitrary $(k-1)$-separator of $G\backslash at$, and $a \in V_{a,t}$ and $t \in V_t$. See Figure 9.2. The key point is that

$$|V_t| < |V_{a,s}| \quad \text{and symmetrically} \quad |V_s| < |V_{a,t}|;$$

this is proved as Claim 1 below.

The theorem follows easily from this inequality. Suppose that each node a_i incident to the cycle C has degree $\geq k+1$. For $0 \leq i \leq \ell-1$, let n_i denote the number of nodes in the component of $(G\backslash a_i a_{i+1})\backslash S_i$ that contains node a_i, where S_i is an arbitrary but fixed $(k-1)$-separator of $(G\backslash a_i a_{i+1})$ (the indexing is modulo ℓ, so $a_\ell = a_0$). For example, using our previous notation,

$n_0 = |V_{a,s}|$ and $n_{\ell-1} = |V_t|$. By repeatedly applying the above inequality we have,

$$n_{\ell-1} < n_0 < n_1 < \ldots < n_{\ell-1}.$$

This contradiction shows that some node a_i incident to the cycle C has $\deg(a_i) = k$.

Claim 1 *Let G be a k-node connected graph. Let a be a node with $\deg(a) \geq k+1$, and let as and at be critical edges. Let S and T be arbitrary $(k-1)$-separators of $G\backslash as$ and $G\backslash at$, respectively. Let the node sets of the two components of $(G\backslash as)\backslash S$ be $V_{a,s}$ and V_s, where $a \in V_{a,s}$ and $s \in V_s$. Similarly, let the node sets of the two components of $(G\backslash at)\backslash T$ be $V_{a,t}$ and V_t, where $a \in V_{a,t}$ and $t \in V_t$. Then*

$$|V_t| < |V_{a,s}| \quad \textit{and symmetrically} \quad |V_s| < |V_{a,t}|.$$

The claim follows from three subclaims. See Figure 9.2. Observe that the node set V is partitioned into three sets w.r.t. S, namely, $V_{a,s}, V_s, S$. This partition induces a partition of T into three sets that we denote by $T_0 = V_s \cap T$, $T_1 = V_{a,s} \cap T$ and $T_2 = S \cap T$, respectively (possibly some of these subsets of T may be empty). Similarly, V is partitioned into three sets w.r.t. T, namely, $V_{a,t}, V_t, T$, and this gives a partition of S into three sets $S_0 = V_t \cap S$, $S_1 = V_{a,t} \cap S$ and $S_2 = S \cap T$. Let V_a denote $V_{a,s} \cap V_{a,t}$, and note that $a \in V_a$.

One way to see the proof is to focus on the four "arms" of the "crossing" separators S and T. By taking two consecutive "arms" together with the "hub" $S \cap T$, we get a candidate separator, say, X; note that X may not be a separator of G. The proof focuses on the "bottom" candidate separator $X = T_1 \cup (S \cap T) \cup S_1$ and the "top" one $Y = T_0 \cup (S \cap T) \cup S_0$. A closer examination shows that $X \cup \{a\}$ is a genuine separator of G but Y is not.

Subclaim 1 $|S_0| \leq |T_1|$ and symmetrically $|T_0| \leq |S_1|$.
By way of contradiction, suppose that $|S_0|$ is $> |T_1|$. Focus on the set $X = T_1 \cup (S \cap T) \cup S_1$. Since $|X| = |S| - |S_0| + |T_1|$ and $|S| = k-1$, we have $|X| \leq k-2$. Since $\deg(a) \geq k+1$, a has at least three neighbors in $V\backslash X$; two of these are s and t; let b be a third one, i.e., $ab \in E$ and $b \notin X \cup \{s,t\}$. By the definition of S and T, $b \notin V_s$ and $b \notin V_t$, hence, $b \in V_a = V_{a,s} \cap V_{a,t}$. Therefore, $V_a\backslash\{a\}$ is a nonempty set. It is easily checked that $N(V_a\backslash\{a\}) \subseteq \{a\} \cup X$. (This is left as an exercise for the reader.) Clearly, $|\{a\} \cup X| \leq k-1$, and $|V_a\backslash\{a\}| \leq |V| - (k+3)$, since the complementary node set contains $S \cup T \cup \{a,s,t\}$. We have a contradiction, because the k-node connectivity of G implies that every node set V' with $0 < |V'| \leq |V| - k$ has at least k neighbors. This shows that $|S_0| \leq |T_1|$. Similarly, it follows that $|T_0| \leq |S_1|$.

Subclaim 2 $V_s \cap V_t = \emptyset$.
Let $Y = S_0 \cup (S \cap T) \cup T_0$. Note that $|Y| = |S| - |S_1| + |T_0| \leq |S| = k-1$, by the previous subclaim. By focusing on $V_s \cap V_t$, and carefully observing that neither a nor one of a's neighbors is in $V_s \cap V_t$, it is easily checked that $|V_s \cap V_t| \leq |V| - (k+2)$ and $N(V_s \cap V_t) \subseteq Y$. As in the proof of the previous subclaim, the k-node connectivity of G implies that the set $V_s \cap V_t$ is empty.

Subclaim 3 $|V_t| < |V_{a,s}|$ and symmetrically $|V_s| < |V_{a,t}|$.
We have

$$|V_t| \quad = \quad |V_{a,s} \cap V_t| + |S \cap V_t| + |V_s \cap V_t|$$

$$\begin{aligned} &\leq |V_{a,s} \cap V_t| + |V_{a,s} \cap T| \\ &= |V_{a,s}| - |V_a| \leq |V_{a,s}| - 1, \end{aligned}$$

where the first inequality follows because $|V_{a,s} \cap T| = |T_1| \geq |S_0| = |S \cap V_t|$ by Subclaim 1, and $|V_s \cap V_t| = 0$ by Subclaim 2, and the second inequality follows because $|V_a| = |V_{a,s} \cap V_{a,t}| \geq 1$. Similarly, it can be proved that $|V_s| < |V_{a,t}|$.

∎

9.10 Approximating minimum-size k-ECSS

The heuristic can be modified to find an approximately minimum-size k-ECSS. We prove a $(1 + (2/(k+1)))$-approximation guarantee. The analysis hinges on Theorem 9.22 which may be regarded as an analogue of Mader's theorem [34, Theorem 1] for k-edge connected graphs.

In this section, an edge e of a k-edge connected graph H is called *critical* if $H \backslash e$ is *not* k-edge connected. Assume that the given graph $G = (V, E)$ is k-edge connected, otherwise, the heuristic will detect this and report failure.

The first step of the heuristic finds an edge set $M \subseteq E$ of minimum cardinality such that every node in V is incident to $\geq k$ edges of M. Clearly, $|M| \leq |E_{opt}|$, where $E_{opt} \subseteq E$ denotes a minimum-cardinality edge set such that (V, E_{opt}) is k-edge connected. The second step of the heuristic finds an (inclusionwise) minimal edge set $F \subseteq E \backslash M$ such that $M \cup F$ is the edge set of a k-ECSS. In detail, the second step starts with $F = \emptyset$ and $E' = E$. Note that $G' = (V, E')$ is k-edge connected at the start. We examine the edges of $E \backslash M$ in an arbitrary order $e_1, e_2, \ldots$. For each edge $e_i = v_i w_i$ (where $1 \leq i \leq |E \backslash M|$), we determine whether or not $v_i w_i$ is critical for the current graph by finding the maximum number of edge disjoint v_i-w_i paths in G'.

Proposition 9.20 *An edge $v_i w_i$ of a k-edge connected graph is not critical iff there exist at least $k+1$ edge disjoint v_i-w_i paths (including the path $v_i w_i$).*

If $v_i w_i$ is noncritical, then we delete it from E' and G', otherwise, we retain it in E' and G', and also, we add it to F. At termination of the heuristic $G' = (V, E')$, $E' = M \cup F$, is k-edge connected and every edge $vw \in F$ is critical w.r.t. k-edge connectivity. Theorem 9.22 below shows that $|F| \leq kn/(k+1)$ for $k \geq 1$. Since $|E_{opt}| \geq kn/2$, the minimum-size k-ECSS heuristic achieves an approximation guarantee of $1 + (2/(k+1))$ for $k \geq 1$.

The next lemma turns out to be quite useful. A straightforward counting argument gives the proof, see Mader [33, Lemma 1].

Lemma 9.21 *Let $G = (V, M)$ be a simple graph of minimum degree $k \geq 1$.*
(i) Then for every node set $S \subseteq V$ with $1 \leq |S| \leq k$, the number of edges with exactly one end node in S, $|\delta(S)|$, is at least k.
(ii) If a node set $S \subseteq V$ with $1 \leq |S| \leq k$ contains at least one node of degree $\geq (k+1)$, then $|\delta(S)|$ is at least $k+1$.

The goal of Theorem 9.22 is to estimate the maximum number of critical edges in the "complement" of a spanning subgraph of minimum degree k in an arbitrary k-edge connected graph H.

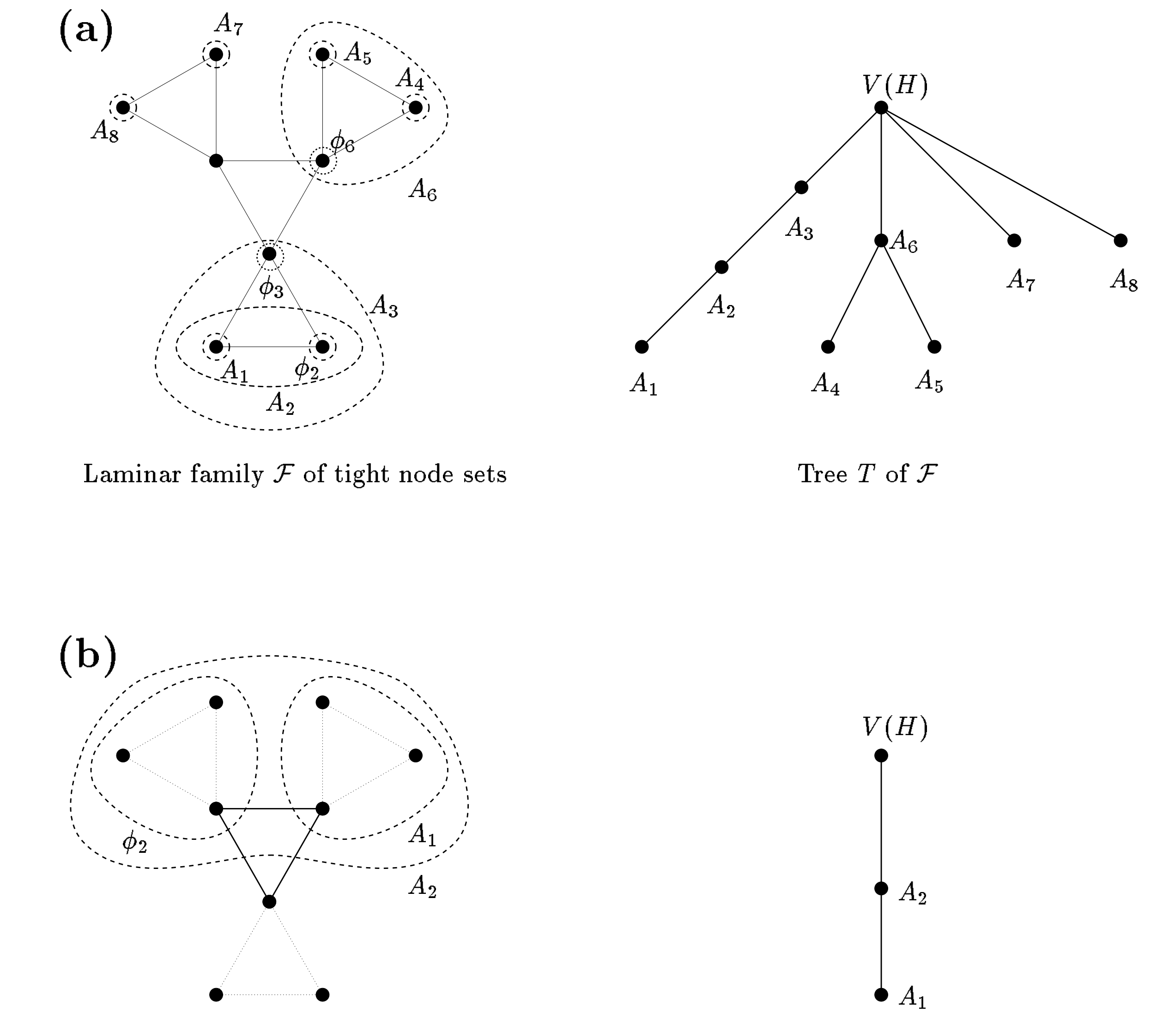


Figure 9.3: Two laminar families of tight node sets for a 2-edge connected graph H ($k = 2$).
(a) The laminar family $\mathcal{F}$ covers *all* critical edges of H. $\mathcal{F}$ consists of the node sets $A_1, \ldots, A_8$, where each A_i is tight since $|\delta(A_i)| = 2 = k$. For a node set A_i, ϕ_i is the node set $A_i \backslash \bigcup\{A_j \in \mathcal{F} \mid A_j \subset A_i,\ A_j \neq A_i\}$. Note that $\phi_i = A_i$ for the inclusionwise minimal A_i, i.e., for $i = 1, 4, 5, 7, 8$. Also, the tree T corresponding to $\mathcal{F} \cup \{V(H)\}$ is illustrated.
(b) The laminar family $\mathcal{F}'$ covers all critical edges of $E(H)\backslash M$, where $M \subset E(H)$ is such that every node is incident to at least $k = 2$ edges of M. M is indicated by dotted lines. All edges of $E(H)\backslash M$ are critical. $\mathcal{F}'$ consists of the tight node sets A_1, A_2. Also, the node sets ϕ_1, ϕ_2 are indicated ($\phi_1 = A_1$), and the tree T' representing $\mathcal{F}' \cup \{V(H)\}$ is illustrated.

Clearly, every critical edge $e \in E(H)$ is in some k-cut $\delta(A_e)$, $A_e \subseteq V(H)$. By a *tight* node set S of a k-edge connected graph H we mean a set $S \subset V(H)$ with $|\delta_H(S)| = k$, i.e., a node set S such that $\delta_H(S)$ is a k-cut. As usual, a family of sets $\{S_i\}$ is called *laminar* if for any two sets in the family, either the two sets are disjoint, or one set is contained in the other. For an arbitrary subset F' of the critical edges of H, it is well known that there exists a laminar family $\mathcal{F}$ of tight node sets *covering* F', i.e., there exists $\mathcal{F} = \{A_1, A_2, \ldots, A_\ell\}$, where $A_i \subseteq V(H)$ and $\delta(A_i)$ is a k-cut, for $1 \leq i \leq \ell$, such that each edge $e \in F'$ is in some $\delta(A_i)$, $1 \leq i \leq \ell$. (For details, see [11, Section 5].) It is convenient to define a tree T corresponding to $\mathcal{F} \cup \{V(H)\}$: there is a T-node corresponding to each set $A_i \in \mathcal{F}$ and to $V(H)$, and there is a T-edge A_iA_j (or $V(H)A_j$) iff $A_j \subset A_i$ and no other node set in $\mathcal{F}$ contains A_j and is contained in A_i. Note that the T-node corresponding to the node set A_i of the laminar family $\mathcal{F}$ is denoted by A_i, and the T-node corresponding to the node set $V(H)$ is denoted by $V(H)$. Each T-edge corresponds to a k-cut of H. Suppose that the tree T is rooted at the T-node $V(H)$. We associate another node set $\phi_i \subseteq V(H)$ with each node set A_i of $\mathcal{F}$:

$$\phi_i = A_i \backslash \bigcup \{A \in \mathcal{F} \mid A \subset A_i, A \neq A_i\}.$$

In other words, a T-node $A_i \in \mathcal{F}$ that is a leaf node of T has $\phi_i = A_i$, otherwise, ϕ_i consists of those H-nodes of A_i that are not in the node sets $A', A'', \ldots$, where $A', A'', \ldots \in \mathcal{F}$ correspond to the children of A_i in the tree T. For distinct T-nodes A_i and A_j, note that ϕ_i and ϕ_j are disjoint. Another useful fact is that $\bigcup_{i=1}^{\ell} \delta(A_i) = \bigcup_{i=1}^{\ell} \delta(\phi_i)$, because every edge in $\delta(\phi_i)$ is either in $\delta(A_i)$ or in $\delta(A'), \delta(A''), \ldots$, where $A', A'', \ldots \in \mathcal{F}$ correspond to the children of A_i in the tree T. See Figure 9.3 for an illustration of $\mathcal{F} = \{A_i\}$, the family of node sets $\{\phi_i\}$, and the tree T for a particular graph.

We skip the proof of the next theorem.

Theorem 9.22 *Let H be a k-edge connected, n-node graph ($k \geq 1$), and let $M \subseteq E(H)$ be an edge set such that every node in $V(H)$ is incident to at least k edges of M. Let F be the set consisting of edges of $E(H) \backslash M$ that are critical w.r.t. k-edge connectivity, i.e., $F \subseteq E(H) \backslash M$ and every edge $e \in F$ is in a k-cut of H. Then, $|F| \leq \frac{k}{k+1}(n-1)$.*

Theorem 9.22 is asymptotically tight. Consider the k-edge connected graph G obtained as follows: take $\ell + 1$ copies of the $(k+1)$-clique, $C_0, C_1, \ldots, C_\ell$, and for each $i = 1, \ldots, \ell$, choose an arbitrary node v_i in C_i and add k (nonparallel) edges between v_i and C_0. Take $M = \bigcup_{i=0}^{\ell} E(C_i)$, and $F = E(G) \backslash M$. Observe that $|F| = k(n - (k+1))/(k+1)$.

Theorem 9.23 *Let $G = (V, E)$ be a graph of edge connectivity $\geq k \geq 1$. The heuristic described above finds a k-edge connected spanning subgraph (V, E') such that $|E'| \leq (1 + (2/(k+1)))|E_{opt}|$, where $|E_{opt}|$ denotes the cardinality of an optimal solution. The running time is $O(k^3n^2 + m^{1.5}(\log n)^2)$.*

9.11 The multi edge model for minimum k-ECSS problems

For minimum k-ECSS problems, two different models have been studied, depending on the number of copies of an edge $e \in E(G)$ that can be used in the desired subgraph: (1) in the simple-edge

Table 9.2: A summary of current approximation guarantees for minimum k-edge connected spanning subgraphs (k-ECSS) in the multi edge model; k is an integer ≥ 2. The references are to: • Goemans & Bertsimas, Math. Programming 60 (1993) pp. 145–166, and • Goemans, Williamson & Tardos, personal communication (1994) cited in Karger's Ph.D. thesis.

	Type of objective function		
	Unit costs	Metric costs	Nonnegative costs
k-ECSS multi-edge model	see last entry $1+O(1)/k$ [GTW94]	see last entry	1.5 for k even [GB93] $1.5+(1/2k)$ for k odd [GB93]

model, at most *one* copy of an edge can be used, and (2) in the multiedge model, an arbitrary number of copies of an edge may be used. Some but not all of the approximation algorithms and guarantees for the simple-edge model extend to the multiedge model; this happens when the input graph may be taken to be a multigraph, because then we can take the given (simple) graph G and modify it into a multigraph by taking k copies of every edge $e \in E(G)$. In the other direction, some of the current approximation guarantees in the multiedge model are strictly better than the corresponding guarantees in the simple-edge model.

For minimum k-ECSS problems and the multiedge model, there is no difference between metric costs and nonnegative costs, because we can replace the given graph G and edge costs c by the "metric completion" G', c', where G' is the complete graph on the node set of G, and c'_{vw} is the minimum c-cost of a v-w path in G, see Goemans & Bertsimas [22, Theorem 3].

9.12 Bibliographic remarks

Given a graph, consider the problem of finding a minimum-size 2-edge connected spanning subgraph (2-ECSS), or a minimum-size 2-node connected spanning subgraph (2-NCSS). Khuller & Vishkin [30] achieved the first significant advance by obtaining approximation guarantees of 1.5 for the minimum-size 2-ECSS problem. Garg et al [20], building on the results in [30], obtained an approximation guarantee of 1.5 for the minimum-size 2-NCSS problem. These algorithms are based on depth-first search (DFS), and they do not imply efficient parallel algorithms for the PRAM model. Subsequently, Chong & Lam [5] gave a (deterministic) **NC** algorithm on the PRAM model with an approximation guarantee of $(1.5+\epsilon)$ for the minimum-size 2-ECSS problem, and later they [7] and independently [4] gave a similar algorithm for the minimum-size 2-NCSS problem. In the context of approximation algorithms for minimum-size k-connected spanning subgraph problems, Chong & Lam [5] appear to be the first to use matching. For the minimum-size k-ECSS problem on simple graphs, Cheriyan & Thurimella [4], building on earlier work by Khuller & Raghavachari [29] and Karger [26], gave a $1+(2/(k+1))$-approximation algorithm. The k-ECSS approximation algorithm in [4] does not apply to multigraphs. For the minimum-size k-ECSS problem on multigraphs, a 1.85-approximation algorithm is given in [29], and a randomized (Las Vegas) algorithm with an approximation guarantee of $1+\sqrt{[O(\log n)/k]}$ is given in [26].

In the context of augmenting the node connectivity of graphs, the first application of Mader's theorem is due to Jordán [25, 24].

One of the first algorithmic applications of Mader's theorem appears to be due to Jordán [25, 24]; Jordán applied the theorem in his approximation algorithm for augmenting the node connectivity of graphs. The key lemma in the analyses in Sections 9.7, 9.8 above, namely, Lemma 9.11 (also, Lemma 9.16) is inspired by these earlier results of Jordán. The analysis of the k-NCSS heuristic for digraphs is similar, and hinges on another theorem of Mader [35, Theorem 1], which may be regarded as the generalization of [34, Theorem 1] to digraphs. An approximation guarantee of $1 + (1/k)$ is proved on the digraph heuristic by employing a simpler version of Theorem 9.18, to give a lower bound on the number of edges in a solution.

9.13 Exercises

1. Prove both parts of Proposition 9.4 using the following sketch.

   For part 2, note that every edge $e \in E(G)$ is critical w.r.t. k-node connectivity, since G is edge-minimal k-node connected. Apply Mader's theorem (Theorem 9.10) and focus on edges that have degree $\geq k+1$ at both end nodes.

2. Prove the following generalization of Chong and Lam's lower bound on the number of edges in a 2-ECSS.

   Proposition 9.24 *Let $G = (V, E)$ be a graph of edge connectivity $\geq k \geq 1$, and let $|E_{opt}|$ denote the minimum size of a k-edge connected spanning subgraph. If G is not factor critical, then $|E_{opt}| \geq \frac{k}{2}(n + \mathrm{def}(G))$. In general, $|E_{opt}| \geq \frac{k}{2}\max(n + \mathrm{def}(G) - 1,\, n)$.*

   (Hint: One way is via the Gallai-Edmonds decomposition theorem of matching theory.)

3. Adapt the 1.5-approximation algorithm for a 2-NCSS in Section 9.7 to find a 2-ECSS whose size is within a factor of 1.5 of minimum. Assume that the given graph G is 2-edge connected.

   (Hint: Focus on a block (i.e., a maximal 2-node connected subgraph) G' of G. Is it true that the size of an optimal 2-NCSS of G' equals the size of an optimal 2-ECSS of G'?)

4. Show that the running time of the second step of the approximation algorithm for a minimum-size k-NCSS can be improved to $O(k^3 n^2)$.

   (Hint: Use Nagamochi & Ibaraki's [36] sparse certificate $\widetilde{E}$ for k-node connectivity. Here, $\widetilde{E} \subseteq E$, $|\widetilde{E}| \leq kn$, and for all nodes v, w, $(V, \widetilde{E})$ has k openly disjoint v-w paths iff G has k openly disjoint v-w paths.)

5. (Research problem) Given a graph, is there a $1 + (1/k)$-approximation algorithm for finding a minimum-size k-ECSS? What about the special case $k = 3$?

Bibliography

[1] B. Bollobás, *Extremal Graph Theory*, Academic Press, London, 1978.

[2] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, American Elsevier Publishing Co., New York, 1976.

[3] J. Cheriyan, M. Y. Kao and R. Thurimella, "Scan-first search and sparse certificates: An improved parallel algorithm for k-vertex connectivity," *SIAM J. Computing* **22** (1993), 157–174.

[4] J. Cheriyan and R. Thurimella, "Approximating minimum-size k-connected spanning subgraphs via matching," Proc. 37th IEEE F.O.C.S. 1996.

[5] K. W. Chong and T. W. Lam, "Approximating biconnectivity in parallel," Proc. 7th Annual ACM SPAA 1995, 224–233.

[6] K. W. Chong and T. W. Lam, "Improving biconnectivity approximation via local optimization," Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms 1996, 26–35.

[7] K. W. Chong and T. W. Lam, "Towards more precise parallel biconnectivity approximation," to appear in the Proc. 7th International Symposium on Algorithms and Computation, December 1996.

[8] S. Even, *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.

[9] J. Edmonds, "Edge-disjoint branchings," in *Combinatorial Algorithms*, R. Rustin, Ed., Algorithmics Press, New York, 1972, 91–96.

[10] C. G. Fernandes, "A better approximation ratio for the minimum k-edge-connected spanning subgraph problem," to appear in the Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms 1997.

[11] A. Frank, "Submodular functions in graph theory," *Discrete Mathematics* **111** (1993), 231–243.

[12] A. Frank, "Connectivity augmentation problems in network design," in *Mathematical Programming: State of the Art*, (Eds. J. R. Birge and K. G. Murty), The University of Michigan, Ann Arbor, MA, 1994, 34–63.

[13] A. Frank, T. Ibaraki and H. Nagamochi, "On sparse subgraphs preserving connectivity properties," *J. Graph Theory* **17** (1993), 275–281.

[14] A. Frank and E. Tardos, "An application of submodular flows," *Linear Algebra and its Applications*, **114/115** (1989), 320–348.

[15] H. N. Gabow, "A scaling algorithm for weighted matching on general graphs," Proc. 26th Annual IEEE FOCS 1985, 90–100.

[16] H. N. Gabow, "A representation for crossing set families with applications to submodular flow problems," Proc. 4th ACM-SIAM S.O.D.A. (1993), 202–211.

[17] H. N. Gabow, "A matroid approach to finding edge connectivity and packing arborescences," *Journal of Computer and System Sciences* **50** (1995), 259–273.

[18] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for general graph matching problems," *Journal of the ACM* **38** (1991), 815–853.

[19] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[20] N. Garg, V. S. Santosh, and A. Singla, "Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques," Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms 1993, 103–111.

[21] A. M. H. Gerards, "Matching," in Handbook of Operations Research and Management Science (Eds. M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser), North-Holland, Amsterdam, 1995.

[22] M. X. Goemans and D. J. Bertsimas, "Survivable networks, linear programming relaxations and the parsimonious property," *Mathematical Programming* **60** (1993), 143–166.

[23] X. Han, P. Kelsen, V. Ramachandran and R. Tarjan, "Computing minimal spanning subgraphs in linear time," *SIAM J. Computing* **24** (1995), 1332-1358.

[24] T. Jordán, "Increasing the vertex-connectivity in directed graphs," *Proc. Algorithms — ESA'93, 1st Annual European Symposium*, **LNCS 726**, Springer, New York, (1993), 236–247.

[25] T. Jordán, "On the optimal vertex-connectivity augmentation," *J. Combinatorial Theory Series B* **63** (1995), 8–20. Preliminary version in *Proc. 3rd I.P.C.O.* (1993), 75-88.

[26] D. R. Karger, "Random sampling in cut, flow, and network design problems," Proc. 26th Annual ACM STOC 1994, 648–657.

[27] D. R. Karger, "Minimum cuts in near-linear time," Proc. 28th Annual ACM STOC 1996, 56–63.

[28] P. Kelsen and V. Ramachandran, "On finding minimal two-connected subgraphs," *Journal of Algorithms* **18** (1995), 1–49.

[29] S. Khuller and B. Raghavachari, "Improved approximation algorithms for uniform connectivity problems," *Journal of Algorithms* **21** (1996), 434–450. Preliminary version in Proc. 27th Annual ACM STOC 1995, 1–10.

[30] S. Khuller and U. Vishkin, "Biconnectivity approximations and graph carvings," *Journal of the ACM* **41** (1994), 214–235. Preliminary version in: Proc. 24th Annual ACM STOC 1992, 759–770.

[31] S. Khuller, "Approximation algorithms for finding highly connected subgraphs," in *Approximation algorithms for NP-hard problems*, ed. D. S. Hochbaum, PWS publishing co., Boston, 1996.

[32] L. Lovász and M. D. Plummer, *Matching Theory*, North-Holland, Amsterdam, 1986.

[33] W. Mader, "Minimale n-fach kantenzusammenhängende Graphen," *Math. Ann.* **191** (1971), 21–28.

[34] W. Mader, "Ecken vom Grad n in minimalen n-fach zusammenhängenden Graphen," *Archive der Mathematik* **23** (1972), 219–224.

[35] W. Mader, "Minimal n-fach zusammenhängenden Digraphen," *J. Combinatorial Theory Series B* **38** (1985), 102–117.

[36] H. Nagamochi and T. Ibaraki, "A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph," *Algorithmica* **7** (1992), 583–596.

[37] M. Rauch Henzinger, S. Rao and H. N. Gabow, "Computing vertex connectivity: New bounds from old techniques," Proc. 37th IEEE F.O.C.S. (1996), 462–471.

[38] R. Ravi and D. P. Williamson, "An approximation algorithm for minimum-cost vertex-connectivity problems," Preliminary version in *Proc. 6th ACM-SIAM S.O.D.A.* (1995), 332-341. To appear in *Algorithmica.*

[39] R. Thurimella, *Techniques for the Design of Parallel Graph Algorithms*, Ph.D. Thesis, The University of Texas at Austin, August 1989.

Chapter 10

Minimum cuts

In this note, we first look at an algorithm due to Stoer and Wagner for finding minimum cuts in graphs without computing flows. We also look at a randomized approach to finding min cuts due to Karger and co-authors that allows us to bound the number of minimum cuts in a graph. Then, we describe two kinds of spanning trees of a capacitated undirected graph that capture the structure of minimum cuts in the graph - flow-equivalent and cut-equivalent trees.

We then present a simple 2-approximation algorithm for the problem of breaking up an undirected graph into k connected components for a pre-specified k using edges of minimum total capacity: this is the k-cut problem. The multiway cut problem is to find a minimum capacity set of edges or nodes whose removal puts a given set of terminals $s_1, s_2, \ldots, s_k$ in an undirected graph in different connected components. We finally describe a 2-approximation algorithm for this problem based on a half-integral relaxation of an integer programming formulation of the problem.

10.1 A simple minimum cut algorithm

10.1.1 Introduction

Let $G = (V, E)$ be a connected undirected graph. Given a node set $Q \subseteq V$, $\delta(Q)$ denotes the set of all edges with one end in Q and the other end in $V \backslash Q$. (Informally, $\delta(Q)$ is the "boundary" of the node set Q in G.) A cut consists of all edges that have one end in Q and the other end in $V \backslash Q$, where Q is a node set such that $Q \neq \emptyset$ and $Q \neq V$; this cut is denoted $(Q, V \backslash Q)$.

Let every edge $ij \in E$ be assigned a nonnegative capacity $c(ij)$. The capacity of a cut is defined as the sum of the capacities of the edges in it, i.e. $c(Q, V \backslash Q) = \sum_{ij \in \delta(Q)} c(ij)$. The minimum cut problem is to find a cut in G with the smallest capacity.

Background

The minimum cut problem arose first in studies relating to how much "flow" could be sent between a source and destination in a network. A flow can be intuitively thought of as a set of paths between the source and destination that share the capacities of the edges without over-using them. An obvious upper limit on the number of such flow paths between a pair of nodes is the capacity of any cut that puts the pair in opposite sides of the cut. The well-known maximum-flow minimum-cut theorem [6, 7, 21] asserts that the maximum number of such flow paths between a pair is equal to the capacity of a minimum cut separating them. Notice that we are not talking about the minimum cut in the graph here but a minimum cut separating the pair in question.

Connectivity of a graph

The capacity of a minimum cut in an undirected graph is also called the *edge-connectivity* of the graph. Consider the case of an undirected graph with all edge capacities being one. The edge-connectivity between a pair of nodes is simply the maximum number of paths between the pair such that the paths are pairwise edge-disjoint. In the sense of the previous paragraph, this is just the maximum flow between this pair, whose value equals the minimum cut separating them. The edge-connectivity of the whole graph is defined as the minimum edge-connectivity between any pair of nodes in it. Thus, trees have edge-connectivity one, a cycle has edge-connectivity two and a k-dimensional hypercube has edge-connectivity k (check this yourselves!).

The notion of edge-connectivity can be easily generalized to node-connectivity by thinking of the nodes as being capacitated. Thus we now consider simple graphs where each node is assumed to have a capacity of one. The node-connectivity between a pair of nodes is defined as the maximum number of paths between the pair such that these paths are pairwise node-disjoint except at the two extremes (source and destination itself). Such paths are sometimes also termed openly-disjoint. There is a counterpart of the maximum-flow minimum-cut theorem for node-capacitated graphs, which ascertains that the maximum number of openly disjoint paths between a pair of *nonadjacent* nodes equals the minimum capacity of any *node-cut* that separates them. This node-cut is defined as a set of nodes that puts the pair in two different connected components upon its removal.

An undirected graph is said to be k-edge connected if its edge-connectivity is k and k(-node) connected if its node connectivity is k.

Algorithms for finding cuts

The first algorithms for computing the minimum cut in a graph were based on finding maximum flows. Combined with a structure called Gomory-Hu trees, these methods involved n different calls to an algorithm to find a maximum-flow between a pair of vertices on an n-node graph[1]. The fastest flow algorithm currently available is based on the Push-Relabel technique of Goldberg and Tarjan [10, 19] and run in time $\tilde{O}(nm)$ on an n-node m-edge graph [2]. Hao and Orlin [15] extended this method to piggyback all the n flow computations in asymptotically the same time for one, giving an algorithm for finding the minimum cut in time $\tilde{O}(nm)$.

An alternate set of algorithms for finding the minimum cut do not use flow. They grew out of early work of Mader [20] that showed that every k-edge connected graph has a subgraph that is k-edge connected and has only $O(kn)$ edges. Subsequent work of Nishizeki and Poljak [25] showed how this subgraph can be constructed as a union of forests. Nagamochi and Ibaraki [22, 23] gave fast algorithms for constructing such subgraphs. A short proof of a generalization of the results of Nagamochi and Ibaraki to mixed cuts containing both edges and nodes was presented by Frank,

[1] Actually, you don't really need to know about the Gomory-Hu tree to show that you can find the min cut in at most n flow computations. Find a simple way to do this yourself by a suitable choice of source-destination pairs.

[2] The $\tilde{O}$ hides poly-logarithmic factors.

Ibaraki and Nagamochi [8]. Since we only examining the edge-version of the minimum cut problem we follow an even simpler treatment due to Stoer and Wagner [27]. The paper by Frank, Ibaraki and Nagamochi remains a valuable reference for the most general result provable using this approach.

10.1.2 The Stoer-Wagner Algorithm

First we will introduce a simple operation on a graph G involving a pair of distinct nodes u and v called *node identification*. This operation simply identifies these two nodes into one new node, deleting self loops if any but retaining parallel edges; the resulting graph is denoted by G_{uv}. The following proposition sheds light on the effect of a node identification on a minimum cut in a graph.

Proposition 10.1 *The cuts of the graph G_{uv} are exactly the cuts of G that do not separate u and v.*

Therefore, we infer that a minimum cut in G is the minimum of two quantities: the minimum cut in G_{uv} and a minimum cut separating u and v (i.e., with these two nodes on the two sides of the cut). The proposition guarantees that these two cases are mutually exhaustive. Furthermore we have the following simple algorithm for finding a minimum cut.

Algorithm Node Identification Algorithm for finding a Minimum Cut
input: Graph $G = (V, E)$, nonnegative edge capacities c
output: A subset of nodes $S \subset V$ such that $\delta(S)$ is a minimum capacity cut of G.

STEP 1: initialize cut $S \leftarrow$ undefined and capacity $C \leftarrow \infty$;
    while G has more than one node do
STEP 2: Pick two distinct nodes s and t and
        compute a minimum capacity cut $\delta(S')$ separating s and t;
STEP 3: **if** $c(S', V \backslash S') < C$
STEP 4: $C \leftarrow c(S', V \backslash S')$ and $S \leftarrow S'$;
        end (if);
STEP 5: replace G by G_{st};
    end (while);
    output the cut $\delta(S)$;

At the outset this algorithm does not appear particularly better than the earlier algorithms - in fact, it still requires the computation of $n-1$ minimum cut and hence maximum flow computations, on an n-node graph. The only minor advantage is that the size of the graph on which we are computing the flow is deceasing by one as we progress in the algorithm.

A critical advantage of the above algorithm reveals itself from a close inspection of STEP 2. In this step we have the choice of picking *any* two surviving nodes s and t in the graph to find a minimum cut separating s and t. The idea is to choose a pair s, t such that this minimum cut is easy to find without using flows.

Legal Orderings

Define an ordering of the nodes of G, say, $v_1, v_2, \ldots, v_n$ to be *legal* if v_1 is arbitrary and for every $i \geq 2$, v_i is the node with the maximum total capacity of edges joining it to the nodes $\{v_1, v_2, \ldots, v_{i-1}$. If we use $c_j(v)$ to denote $\sum_{x=1}^{j} c(v_x v)$, then v_i is a node maximizing $c_{i-1}(v)$ over all the remaining nodes v. The usefulness of legal orderings stems from the following result.

Theorem 10.2 *If $v_1, \ldots v_n$ is a legal ordering of the vertices of G, then $\delta(\{v_n\})$ is a minimum cut separating v_{n-1} and v_n in G.*

For any node subset V' of G, let $G(V')$ denote the subgraph of G induced by the nodes in V', namely, only the set of edges with both endpoints in V'. The theorem follows from the following lemma about legal orderings.

Lemma 10.3 *Let $v_1, \ldots v_n$ be a legal ordering of the vertices of G. Let v be any vertex in $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$. Then in the graph $G(\{v_1, v_2, \ldots, v_i, v\})$, the minimum cut separating v and v_i has capacity at least $c_i(v)$.*

Proof: The proof is by induction on i. The basis when $i = 1$ is trivial. Assume the lemma is true for $i-1$; we'll prove for i. Consider a vertex $v \in \{v_{i+1}, \ldots, v_n\}$. By the induction hypothesis, in the graph $G(\{v_1, v_2, \ldots, v_{i-1}, v\})$, the minimum cut separating v and v_i has capacity at least $c_{i-1}(v)$. Similarly, in the graph $G(\{v_1, v_2, \ldots, v_i\})$, the minimum cut separating v and v_i has capacity at least $c_{i-1}(v_i)$. These imply that in the graph $G(\{v_1, v_2, \ldots, v_i, v\})$, the minimum cut separating v and v_i has capacity at least $c_i(v)$.

Suppose for a contradiction that in the graph $G(\{v_1, v_2, \ldots, v_i, v\})$, the minimum cut separating v and v_i has capacity less than $c_i(v)$. Since $c_i(v) = c(v_i v) + c_{i-1}(v)$, this implies that after deleting the edge $v_i v$ (if it exists) from this graph, the minimum cut separating v and v_i has capacity less than $c_{i-1}(v)$. Call this cut C. Consider which side of this cut lies the vertex v_{i-1}.

Case 1. If v_{i-1} lies on the same side of C as v, then C is also a cut separating v_{i-1} and v_i in the graph $G(\{v_1, v_2, \ldots, v_i\})$. Thus the capacity of C is at least $c_{i-1}(v_i)$. By the choice of v_i in the legal ordering, $c_{i-1}(v_i) \geq c_{i-1}(v)$. This contradicts the assumption that C has capacity less than $c_{i-1}(v)$.

Case 2. If v_{i-1} lies on the opposite side of C to v, then C is also a cut separating v_{i-1} and v in the graph $G(\{v_1, v_2, \ldots, v_{i-1}, v\})$. Thus its capacity must be at least $c_{i-1}(v)$, a contradiction.

This completes the proof of the lemma. ∎

How efficiently can a legal ordering be found? The key step is to find the next vertex in the ordering by looking for the vertex with the maximum capacity into the set of already assigned nodes. This step is similar to a step of Dijkstra's shortest path algorithm or a step of Prim's minimum spanning tree algorithm. By using similar data structures a legal ordering of an n-node m-edge graph can be found in time $O(m + n \log n)$.

Min cut algorithm using legal orderings

A legal ordering identifies a pair of vertices, v_{n-1} and v_n such that the cut around the singleton set $\{v_n\}$ is a minimum cut separating v_{n-1} and v_n. Notice that we cannot force a given pair of vertices to be the last two vertices of a legal ordering. Nevertheless, after an efficient computation, we find a pair of vertices and a minimum cut separating them. By using this pair of vertices in STEP 2 of the node identification algorithm, we can implement each iteration of the algorithm efficiently giving an overall running time of $O(mn + n^2 \log n)$ for our minimum cut algorithm.

10.1.3 A bound on the number of min cuts

A randomized version of the node identification algorithm due to Karger and Stein [18] can be used to show a simple upper bound on the number of minimum cuts in an undirected graph. Let $c(e)$ denote the capacity of an edge $e \in E$; we extend the notation and use $c(E)$ to denote the total capacity of all the edges in the graph.

Algorithm Random Contraction Algorithm
input: Graph $G = (V, E)$, nonnegative edge capacities c
output: A set of edges forming a minimum cut in G with probability at least $\frac{2}{n(n-1)}$.

while G has more than two nodes do
STEP 1: choose an edge randomly where edge e is chosen with probability $\frac{c(e)}{c(E)}$.
STEP 2: **if** the chosen edge is uv, replace G by G_{uv}.
output the set of edges forming the unique cut of the resulting graph G;

Theorem 10.4 *Fix C to be a minimum cut of a graph G on n nodes. Then the probability that the random contraction algorithm running on G outputs C is at least $\frac{2}{n(n-1)}$.*

Proof: We would like to lower bound the probability of the good event that no edge of C is touched in the course of the contraction algorithm.

First let us estimate the probability that an edge of C is contracted in the first iteration of the algorithm when the first edge to be contracted is chosen.

$$Pr(\text{an edge of } C \text{ is contracted}) = \frac{c(C)}{c(E)}$$

where $c(C)$ denotes the total capacity of the cut C. Note that we can write $c(E) = \frac{1}{2}\sum_{v \in G} c(\{v\}, V\backslash\{v\})$. Since C is a minimum cut, we have for any $v \in G$ that $c(C) \leq c(\{v\}, V\backslash\{v\})$. Substituting back we get that $c(E) \geq \frac{1}{2} \cdot n \cdot c(C)$ and that

$$Pr(\text{an edge of } C \text{ is contracted}) \leq \frac{2}{n}$$

It is straightforward to extend this argument to show that after i contractions by the algorithm,

$$Pr(\text{an edge of } C \text{ is contracted}) \leq \frac{2}{n-i}$$

Therefore after i contractions, assuming that C survives,

$$Pr(\text{an edge of } C \text{ is not contracted}) \geq 1 - \frac{2}{n-i} = \frac{n-i-2}{n-i}$$

Finally we have that

$$\begin{aligned} Pr(\text{the algorithm outputs } C) &= Pr(\text{no edge of } C \text{ is ever contracted}) \\ &\geq (\frac{n-2}{n})(\frac{n-2-1}{n-1})\cdots(\frac{n-2-(n-3)}{n-(n-3)}) \\ &\geq \frac{(n-2)(n-3)\ldots 2\cdot 1}{n(n-1)\ldots 4\cdot 3} \\ &\geq \frac{2}{n(n-1)} \end{aligned}$$

▮

We have the following corollary.

Corollary 10.5 *The number of distinct minimum cuts in an undirected graph is at most* $\frac{n(n-1)}{2}$.

The corollary follows by using the Theorem to observe that for every distinct minimum cut, the probability of the algorithm returning that particular cut is at least $\frac{2}{n(n-1)}$. Returning a pair of distinct minimum cuts is a pair of mutually exclusive events and the bound follows.

This bound on the number of different minimum cuts is the best possible in the sense that there is an n-node graph with exactly $\frac{n(n-1)}{2}$ cuts. Namely, each pair of edges in an n-cycle is a minimum cut.

Karger and co-workers have extended this paradigm to provide a series of algorithms with better and better running times and better success probabilities of returning a minimum cut. For details of these methods and an empirical evaluation of the different approaches to computing minimum cuts, see [3] and the references therein.

10.2 Gomory-Hu Trees: Existence

In this section, we look at two kinds of spanning trees of a capacitated undirected graph that capture the structure of minimum cuts in the graph.

10.2.1 Introduction

Let $G = (V, E)$ be a connected undirected graph, and as before, let every edge $ij \in E$ be assigned a nonnegative capacity $c(ij)$. We define two types of trees on the same vertices as G. Note that we do *not* require that the edges of T be a subset of the edges of G – in both cases, the tree T is only intended to capture the structure of minimum cuts in G.

A **cut-equivalent tree** (relative to G) is defined as a tree T with $V(T) = V(G)$ with the following property: for every edge $xy \in E(T)$, the removal of xy from T partitions $V(T) = V(G)$

into two subsets $X \ni x$ and $Y \ni y$; then we require the cut $\delta(X)$ in G to be a minimum capacity cut separating x and y.

A **flow-equivalent tree** is defined as a tree with $V(T) = V(G)$ along with nonnegative capacities c' assigned to the edges of T with the following property: for any pair of nodes x and y, the maximum flow between x and y has the same value in both G and T. By the max-flow min-cut theorem, this translates to requiring that the minimum cut separating x and y has the same value in both G and T.

Note that a cut-equivalent tree is uncapacitated while the flow-equivalent tree is capacitated. However, there is a natural assignment of capacities c' to the edges of a cut-equivalent tree: for every edge xy in the tree, let X be the component of $T - xy$ containing x; then we assign $c'(xy) = c(X, V\backslash X)$, namely, the capacity of the cut $\delta(X)$ separating x and y in G. A cut-equivalent tree with these capacities is called a **Gomory-Hu tree**, named after its discoverers Gomory and Hu [12].

First we begin with a simple observation.

Proposition 10.6 *Let $v_1, v_2, \ldots, v_k$ be a sequence of distinct nodes of G. Let c_{ij} denote the value of a minimum cut separating v_i and v_j. Then*

$$c_{ik} \geq \min\{c_{12}, c_{23}, \ldots, c_{k-1,k}\}$$

Proof: Consider tracing the path from v_1 to v_k along $v_2, \ldots, v_{k-1}$ (this path need not exist in G). This path must cross any cut separating v_1 and v_k at least once. Hence, a minimum cut of value c_{1k} separating v_1 and v_k must also separate a consecutive pair of nodes v_i and v_{i+1} for some $i \in \{1, 2, \ldots, k-1\}$. This implies the claim. ▮

Proposition 10.7 *A cut-equivalent tree is also flow-equivalent, when each edge is assigned its capacity in the natural way described above.*

Proof: Let T be the cut-equivalent tree with capacities c', and let s and t be a pair of distinct nodes. We'll show that the minimum cut separating s and t has value equal to the minimum c'-value of an edge in the $s-t$ path in T. Let this path be $P = \{s = v_1, v_2, \ldots v_k = t\}$. Applying Proposition 10.6 to P yields that $c_{st} \geq c_{i,i+1} = c_{xy}$ for some pair $v_i = x$ and $v_{i+1} = y$ of consecutive vertices in the path.

Since T is cut-equivalent, the cut induced by the two components of $T - xy$ is a minimum cut separating x and y of value c_{xy} in G. Note that this cut also separates s and t, and has value c_{xy}, giving $c_{st} \leq c_{xy}$.

Combining the two inequalities above, we see that $c_{st} = c_{xy}$, which shows that T is flow-equivalent. ▮

10.2.2 Warm-up: Maximum spanning trees

The existence of flow-equivalent trees implies that there is a lot of redundancy in the minimum cut values between different pairs of vertices. Among the possible $\frac{n(n-1)}{2}$ values, there are at most $n-1$ distinct ones represented in the flow-equivalent tree. Note that the minimum cut between a pair of vertices in the flow-equivalent tree is the smallest capacity of an edge in the unique tree path between them.

To build intuition, we will first show a simple proof of existence of flow-equivalent trees. This proof will imply an algorithm for constructing such trees which involves carrying out $\frac{n(n-1)}{2}$ minimum cut computations. In the next section, we will strengthen this result in two directions: by reducing the number of cut computations to $n-1$ and by building a cut-equivalent tree.

Given an undirected graph $G = (V, E)$ with nonnegative weights w_e on its edges $e \in E$, a maximum spanning tree of G is a spanning tree of maximum total weight in G. Methods for finding minimum spanning trees extend to find maximum spanning trees as well. For example, a Kruskal-like algorithm adds edges in decreasing order of weight disregarding edges that form a cycle, until a spanning tree is formed. A Prim-like algorithm grows a core of nodes and extends the core by always using the maximum weight edge coming out of the core to identify the next vertex to pull into the core. A simple property of maximum weight spanning trees is summarized below.

Proposition 10.8 *Let T be a maximum spanning tree and let $f = xy$ be a non tree edge. Then $w_f \leq w_e$ for an edge tree e on the path between x and y in T.*

Proof: The argument is similar to that for minimum spanning trees. Assume for a contradiction that $w_f > w_e$ for appropriate edges e and f. Replacing e with f in T results in a different tree $T - e + f$ that has more weight than T, contradicting that T is maximum weight. ∎

To show that a capacitated undirected graph G has a flow-equivalent tree, we construct a complete weighted graph G^* with the same nodes as G. We define the weight of an edge xy in G^* to be the value c_{xy} of a minimum cut separating x and y in G.

Lemma 10.9 *A maximum spanning tree of G^* is a flow-equivalent tree for G.*

Proof: Let T be a maximum spanning tree of G^*, and let s and t be a pair of distinct nodes in T. Let $P = \{s = v_1, v_2, \ldots v_k = t\}$ be that path in T between s and t.

Suppose $k = 2$ and st is an edge of T, then trivially, by the definition of the weight function of G^*, the minimum cut separating s and t has value equal to the weight of the edge st in T.

Now suppose $k \geq 3$ and so the edge st is not in T. By Proposition 10.6, $c_{st} \geq c_{i,i+1} = c_{xy}$ for some pair $v_i = x$ and $v_{i+1} = y$ of consecutive vertices in the path P. By Proposition 10.8, $c_{st} \leq c_{xy}$. Combining these gives that $c_{st} = c_{xy}$ and thus T is flow-equivalent for G. ∎

The above proof gives a simple algorithm for constructing a flow-equivalent tree. First run $\frac{n(n-1)}{2}$ minimum cut computations to determine the weights of the edges in G^*. Then we can use a maximum spanning tree algorithm to compute T from G^*.

10.3 Gomory-Hu Trees: Construction

In this section, we present an algorithm for constructing a cut-equivalent tree of an n-node graph using only $n-1$ cut computations. Before that we present a few preliminaries on uncrossing cuts.

10.3.1 Uncrossing cuts

A pair of sets S and Q (where $\emptyset \neq S, Q \neq V$) are said to be *crossing* if each of the four sets $S \cap Q, S - Q, Q - S$ and $V - Q - S$ is non-empty. A pair of cuts $\delta(S)$ and $\delta(Q)$ are said to be crossing if the sets S and Q are crossing sets.

In the sequel, we use the term $s-t$ minimum cut to denote a cut of minimum capacity separating a pair of distinct vertices s and t in G. Also we will always use $Y = V \backslash X, B = V \backslash A$ and similarly for Y' and B'.

A real-valued set function $f : 2^V \to \Re$ defined on node-subsets is *submodular* if it satisfies

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$$

for all $X, Y \subseteq V$. We may extend the definition and call f *weakly submodular* if it satisfies

$$f(X) + f(Y) \geq \max(f(X \cup Y) + f(X \cap Y), f(X - Y) + f(Y - X))$$

for all $X, Y \subseteq V$.

Consider the function defined on $S \subseteq V$ as $c(S) = c(S, V \backslash S)$, the capacity of the cut $\delta(S)$.

Proposition 10.10 *The capacity function on node subsets $c : 2^V \to \Re_+$ is weakly submodular.*

Proof: The proof proceeds by showing the following two inequalities.

(i) for all $X, A \subseteq V$,
$$c(X) + c(A) \geq c(X \cup A) + c(X \cap A)$$

(ii) for all $X, A \subseteq V$,
$$c(X) + c(A) \geq c(X - A) + c(A - X)$$

The proof for both parts follows from a simple counting argument: consider two cuts $\delta(X)$ and $\delta(A)$ and look at the different kinds of edges in the right-hand side to verify that they occur with at least the same multiplicity in the left hand side as well. See Figure 10.3.1. ∎

The main application of the weakly submodular property of cuts is in *uncrossing* a pair of crossing cuts. By uncrossing this pair of cuts, we mean replacing them by another pair, say $\delta(X')$ and $\delta(A')$ such that X' and A' do not cross, and the two new cuts have the same properties as the old cuts. For example, if $\delta(X)$ and $\delta(A)$ were both minimum cuts in G, than $\delta(X')$ and $\delta(A')$ will also both be minimum cuts.

As another example, suppose $\delta(X)$ and $\delta(A)$ were $x - y$ and $a - b$ minimum cuts respectively; Then, the pair of uncrossed cuts $\delta(X')$ and $\delta(A')$ will also be $x - y$ and $a - b$ minimum cuts respectively. We show these two applications of weak submodularity to uncrossing cuts in the next two lemmas.

Lemma 10.11 *Suppose $\delta(X)$ and $\delta(A)$ are crossing minimum cuts of G. Then $\delta(X \cap A)$ and $\delta(X \cup A)$ are also minimum cuts and they are non-crossing. Similarly, $\delta(X - A)$ and $\delta(A - X)$ are also non-crossing minimum cuts.*

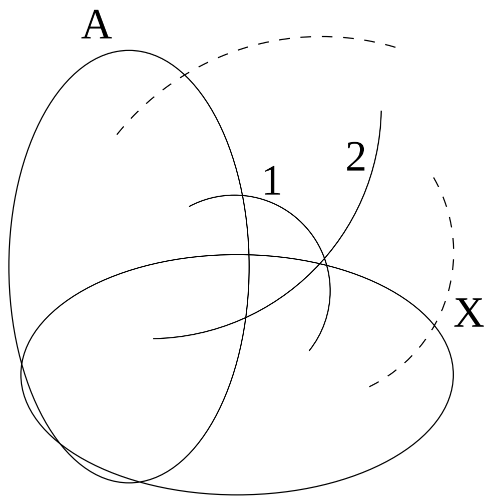


Figure 10.1: Proof of Proposition 10.10. The dotted edges are counted with the same multiplicity in both inequalities. Edges of the type labeled 1 are not counted in the right-hand side of (i), while edges of type 2 are not counted in the right-hand side of (ii). This results in the inequality rather than equality.

Proof: The proof is a direct application of proposition 10.10. Part (i) gives that

$$c(X) + c(A) \geq c(X \cup A) + c(X \cap A)$$

Since X and A cross, both $\delta(X \cup A)$ and $\delta(X \cap A)$ are nonempty sets of edges and have capacity at least that of a minimum cut. But both terms on the left hand side are values of the minimum cut by definition and so both $\delta(X \cup A)$ and $\delta(X \cap A)$ must be minimum cuts as well. The proof for the other pair is similar using the second part of proposition 10.10. ∎

Lemma 10.12 *Let $\delta(S)$ be an $s - t$ minimum cut where $s \in S$ and $t \in T = V \backslash S$, and let $\delta(X)$ be an $x - y$ minimum cut, where $x \in X$ and $y \in Y = V \backslash X$. Assume that S and X cross. Then there exists a pair $\delta(S')$ and $\delta(X')$ of noncrossing of $s - t$ and $x - y$ minimum cuts in G.*

Proof: The proof is essentially a lengthy case analysis investigating the various relative locations of the vertices x, y, s and t, and using Proposition 10.10 extensively. We will consider two possible cases for the location of x and y relative to s and t.

Case A: Both x and y are in the same side of the cut $\delta(S)$, say in S. Without loss of generality, assume that $s \in S \cap X$. There are two subcases to consider. (see Figure 10.3.1).

1. $t \in T \cap Y$: Apply proposition 10.10 (i) to the pair of crossing cuts $\delta(S)$ and $\delta(X)$ to get

   $$c(S) + c(X) \geq c(S \cap X) + c(S \cup X)$$

   Note that $\delta(S)$ and $\delta(X)$ are minimum cuts for the pairs $s - t$ and $x - y$ respectively. Furthermore, $\delta(S \cap X)$ separates x and y while $\delta(S \cup X)$ separates s and t in this case. Thus the inequality must be tight with $\delta(S \cap X)$ being a minimum cut separating x and y and $\delta(S \cup X)$ being a minimum cut separating s and t.

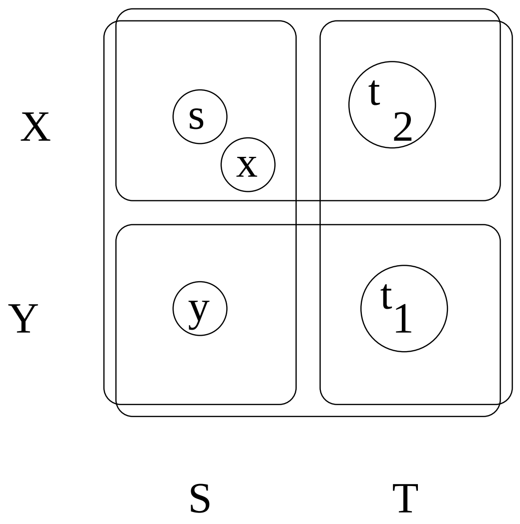


Figure 10.2: Case A in the proof of Lemma 10.12. The location of the node t in the two subcases 1 and 2 are shown as t_1 and t_2 respectively.

2. $t \in T \cap X$: The proof is similar. We apply proposition 10.10 (i) to the pair $\delta(S)$ and $\delta(Y)$ in this case to obtain
$$c(S) + c(Y) \geq c(S \cap Y) + c(S \cup Y)$$
Again $\delta(S)$ and $\delta(X)$ are minimum cuts for the pairs $s-t$ and $x-y$ respectively. Furthermore, $\delta(S \cap Y)$ separates x and y while $\delta(S \cup Y)$ separates s and t in this case. Thus the inequality must be tight with $\delta(S \cap Y)$ being a minimum cut separating x and y and $\delta(S \cup Y)$ being a minimum cut separating s and t.

Case B: x and y are in opposite sides of the cut $\delta(S)$, say $x \in S$ and $y \in T$. There are two subcases to consider depending on the location of s. (see Figure 10.3.1).

1. $s \in S \cap X$. In this case, the cut $\delta(S \cap X)$ is an $s - t$ cut while $\delta(S \cup X)$ is an $x - y$ cut. By submodularity, both are also minimum.

2. $s \in S \cap Y$. There are two further subcases depending on the location of t. In the first and more routine case, $t \in T \cap Y$ (denoted t_{21} in figure 10.3.1). In this case, $\delta(Y \cap T)$ is an $s - t$ cut while $\delta(Y \cup T)$ is an $x - y$ cut. Again, by submodularity, they are both minimum.

   The more interesting subcase is when $t \in T \cap X$ (denoted t_{22} in the figure). Note that we cannot directly apply submodularity to any pair of crossing cuts here. However, in this case, $\delta(S)$ is an $x - y$ cut while $\delta(X)$ is an $s - t$ cut. Since both cuts are minimum, they both have the same value. Applying weak submodularity for a single pair of cuts, say $\delta(S)$ and $\delta(X)$ now shows that all four cuts $\delta(S \cap X), \delta(S \cup X), \delta(S - X)$ and $\delta(X - S)$ have the same value. We can now use $\delta(S \cap X)$ and $\delta(S \cap Y)$ as the pair of minimum $x - y$ and $s - t$ cuts respectively.

∎

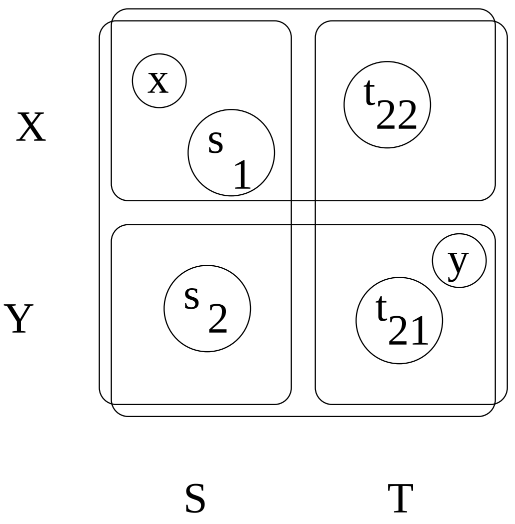


Figure 10.3: Case B in the proof of Lemma 10.12. The location of the node s in the two subcases 1 and 2 are shown as s_1 and s_2 respectively. In the subcase 2 when s is located at s_2, the two possible locations of the node t are labeled t_{21} and t_{22} respectively.

Corollary 10.13 *Consider an undirected graph G and let s and t be distinct nodes. Also let x and y be distinct nodes such that x may be a and y may be b. Let $\delta(X)$ be an $x-y$ minimum cut, where $x \in X$ and $y \in Y = V \backslash X$. Let $G(x^*)$ denote the graph obtained from G by identifying all the nodes in X to a single node, and let $G(y^*)$ be similarly defined. Then, either $\delta(X)$ is a minimum $s-t$ cut, or there is a minimum $s-t$ cut of G that is also a cut in either in $G(x^*)$ or $G(y^*)$.*

Proof: Suppose $\delta(X)$ is not a minimum $s-t$ cut. Let $\delta(S)$ be a minimum $s-t$ cut. If S and X do not cross, then the side of $\delta(X)$ containing S continues to contain a minimum $s-t$ cut of G.

Otherwise, S and X cross and we can use the proof of Lemma 10.12 to conclude that there is a minimum $s-t$ cut in G that is also a cut in either $G(x^*)$ or $G(y^*)$. ▌

10.3.2 The construction algorithm

The algorithm for constructing a cut-equivalent tree starts with the tree being a single "supernode" containing all the nodes of G. At each step, a supernode containing at least a pair of real nodes is split by computing a minimum cut between the pair, and a tree edge is added between the two resulting supernodes. The procedure stops when every supernode contains exactly one original node of G at which point we have a tree spanning the nodes of G.

Algorithm Cut-Equivalent Tree
input: Graph $G = (V, E)$, nonnegative edge capacities c.
output: A cut-equivalent tree T for G.
STEP 1: initialize T to be a single supernode containing all nodes of G;

while there is a supernode containing more than a single node do
STEP 2: pick a supernode U containing nodes x and y;
STEP 3: form a new contracted graph G_c^U as follows:
for each subtree of T attached to U,
STEP 4: contract to a single node all the nodes of G contained in this subtree;
(delete self loops but retain parallel edges)
STEP 5: find a min cut $\delta(X)$ separating x and y in G_C^U;
STEP 6: update T as follows:
STEP 7: split supernode U into two supernodes X and Y
partitioning the nodes in U according to the cut defined by X;
for every supernode W adjacent to U originally
STEP 8: reconnect W to either X or Y
depending on which side of the cut $\delta(X)$ that W appeared in G_c^U;
output the tree T;

The time complexity of the algorithm is dominated by the minimum cut computations in step 5, and there are $n-1$ calls to this procedure for an n-node graph.

Correctness

Theorem 10.14 *The algorithm above outputs a cut-equivalent tree.*

The theorem is proved by showing the following lemma[3]. Let T_i represent the tree after i iterations of the algorithm through steps 2-8, i.e., after i minimum cut computations. Note that any edge T_i represents a unique cut in G defined by the bipartition of nodes according to this edge in T_i. Also the cut represented by any edge in T_i continues to be represented in all the subsequent trees $T_j,\ j > i$.

Lemma 10.15 *Consider the tree T_i. Let a and b be any two nodes in G where $a \in S(a)$ and $b \in S(b)$ for supernodes $S(a)$ and $S(b)$ in T_i. Suppose that the supernodes $S(a)$ and $S(b)$ are either identical or separated by at most one edge in T_i. Then there is a minimum $a-b$ cut such that*

(i) all of the edges of the cut are either in $G_c^{S(a)}$ or in $G_c^{S(b)}$ or

(ii) this cut is defined by the edge separating $S(a)$ and $S(b)$ in T_i.

Note that case (*ii*) of the claim applies only when $S(a)$ and $S(b)$ are distinct supernodes that are adjacent in T_i.

Proof: The proof is by induction on i. The basis when $i = 0$ corresponds to a single supernode and the minimum $a-b$ cut obviously appears in the graph induced by all the nodes. Suppose the claim holds for i, we'll prove it for $i+1$. Suppose in T_i that $a \in S(a)$ and $b \in S(b)$. By the inductive hypothesis, the minimum $a-b$ cut is either in $G_c^{S(a)}$ or $G_c^{S(b)}$ (assume without loss of generality it

[3]Thanks to Rachel Rue for proposing the lemma that unifies two lemmas suggested earlier for the proof.

is in $G_c^{S(a)}$ in this case), or is the cut represented by the edge between the two distinct supernodes $S(a)$ and $S(b)$ in T_i. We consider these two cases separately.

1. The minimum $a-b$ cut is in $G_c^{S(a)}$. Let $\delta(A)$ be the minimum $a-b$ cut and $B = V(G_c^{S(a)})\backslash A$. If the $(i+1)$st iteration puts a and b in supernodes that are more than one edge away in T_{i+1} we cease to worry about this pair of nodes in enforcing the inductive claim.

   Suppose that a and b are in supernodes at most one edge away in T_{i+1}. Then the cut in $G_c^{S(a)}$ between them will be affected only if the $(i+1)$st cut was found on the supernode $S(a)$, i.e., in this iteration, in step 2, $U = S(a)$. In the following, if $S(a) \neq S(b)$, replace b by b^*, the supernode representing $S(b)$ in $G_c^{S(a)}$. Suppose the minimum $x-y$ cut found by the algorithm in the graph $G_c^{S(a)}$ in step $i+1$ is $\delta(X)$, where $a \in X$ without loss of generality. Note that x may be identical to a and/or y may be identical to b. Let $Y = V(G_c^{S(a)})\backslash X$. The supernode containing a after this iteration is $S'(a) = S(a)\backslash Y$, and that containing b is $S'(b) = S(b)\backslash X$. By Corollary 10.13 with $G = G_c^{S(a)}$ and $A = S$ and $B = T$, the minimum $a-b$ cut $\delta(A)$ of $G_c^{S(a)}$ is either $\delta(X)$ or it survives in one of $G_c^{S'(a)}$ or $G_c^{S'(b)}$. In the former case, we have a cut in T_{i+1} of type (ii), while in the latter, it is of type (i).

2. The minimum $a-b$ cut is defined by the edge separating $S(a)$ and $S(b)$ in T_i, where $S(a) \neq S(b)$. If the $(i+1)$st iteration puts a and b in supernodes more than one edge away, we do not worry about enforcing the inductive claim for the pair.

   Otherwise, irrespective of whether the cut computation in step $i+1$ involves node $S(a)$ or $S(b)$, the edge separating the supernodes $S'(a)$ and $S'(b)$ in T_{i+1} containing a and b represents the same cut as the edge between $S(a)$ and $S(b)$ in T_i. This is a cut of type (ii).

Notice that all pairs a, b obeying the condition of the lemma in T_{i+1} must have obeyed this condition even in T_i and hence the proof is complete. ∎

For an alternate version of the algorithm that allows for easy integration with existing code for finding minimum cuts, see [14]. This version does not require contractions and the necessary operations that go with it in finding the cut-equivalent tree. Other interesting properties of cut and flow-equivalent trees appear in [1, 16].

10.4 Multicuts

In this section, we examine an approximation algorithm for the problem of breaking up an undirected graph into k connected components for a pre-specified k using edges of minimum total capacity.

10.4.1 Introduction

Let $G = (V, E)$ be a connected undirected graph, and let every edge $ij \in E$ be assigned a nonnegative capacity $c(ij)$. For a given positive integer k, the minimum k-cut problem is to find a set of edges of minimum total capacity whose removal from G leaves at least k connected components. Note that since the capacities are nonnegative, a minimal solution will result in exactly k components upon its removal (if there are more, we can "put back" some edges from the solution into the graph until we have exactly k components). Notice that a k-cut has more than two shores (connected component) for $k > 2$ and hence cannot be specified by a single subset of edges; Rather, we now need a partition of the vertices into multiple blocks to specify such a cut. Hence, such cuts also also termed *multicuts*.

The minimum k-cut problem is solvable in time $O(n^{k^2})$ on an n-node graph [11], while on a planar graph, the running time can be reduced to $O(n^{ck})$ [5]. However for arbitrary k (when k is part of the input specification), this problem is NP-hard [11]. We will look at an approximation algorithm for this problem due to Saran and Vazirani [26] that achieves performance ratio $2(1 - \frac{1}{k})$. The algorithm they propose in fact gives an approximate solution for a minimum i-cut for every value of i from 2 to k due to a *extendible* property of the approximate solution – a near-optimal minimum i-cut can be extended to a near optimal $i + 1$-cut.

10.4.2 Two simple algorithms

First we examine two simple heuristics – GREEDY and SPLIT. We then show how they are both dominated in performance and running time by a better heuristic EFFICIENT. Then, in the next section, we prove the performance of EFFICIENT. We shall use lower-case letters with subscripts (such as b_i, c_i, g_i, and s_i to denote subsets of edges in the sequel.

A greedy method

A simple greedy way to create more components is to pick the lightest (minimum capacity) cut that creates more components in the current graph. A naive method to implement this idea will look at a list of *all* cuts in G sorted by nondecreasing capacity $c_1, c_2, c_3, \ldots$, and iteratively picking the first one in the sequence that creates more components. Notice the resemblance to Kruskal's algorithm for finding minimum spanning trees.

It is however impractical to create such a list, but we can still implement this method by looking at a much more restricted list of cuts: for every edge e, we find a minimum cut s_e separating the endpoints of e, and sort this list of cuts by capacity. We run our algorithm on this smaller list. This gives us the first algorithm.

Algorithm GREEDY
input: Graph $G = (V, E)$, nonnegative edge capacities c.
output: A k-cut of capacity at most $2(1 - \frac{1}{k})$ of the minimum.
STEP 1: Initialize the multicut $s \leftarrow \emptyset$, and $i \leftarrow 1$;
        for each edge e
STEP 2:   pick a minimum cut s_e separating the endpoints of e;
STEP 3: Sort these cuts in the order of nondecreasing capacity to get the list
        $s_1, s_2, \ldots, s_m$, where $m = |E|$;
        while $(V, E - s)$ has fewer than k connected components **do**
STEP 3:   **if** $s_i \not\subseteq s$, **then** $s \leftarrow s \cup s_i$;
STEP 4: $i \leftarrow i + 1$;
        output s;

Note that $E \subseteq s_1 \cup s_2 \cup \ldots \cup s_m$ and hence the algorithm will be able to find a k-cut for any value of $k \leq n$. A simplistic implementation of this algorithm may require up to m pairwise minimum cut computations. However, we can use a cut-equivalent tree to determine all the relevant cuts, leading to the algorithm EFFICIENT which is the subject of the next section.

A simple splitting method

Another simple idea to generate more components is to work in each of the current components in the graph and find the lightest cut that splits one of them. If this is repeated $k-1$ times, we get a k-cut, giving the following algorithm.

Algorithm SPLIT
input: Graph $G = (V, E)$, nonnegative edge capacities c.
output: A k-cut of capacity at most $2(1 - \frac{1}{k})$ of the minimum.
STEP 1: Initialize the multicut $s \leftarrow \emptyset$, and $i \leftarrow 1$;
  for i from 1 to $k-1$ **do**
    for each connected component G_j of the graph $(V, E - s)$
STEP 2:   pick a minimum cut s_j of G_j;
STEP 3:   Let s' be a cut with minimum capacity among the cuts s_j; Update $s \leftarrow s \cup s'$;
  output s;

Again, a simple implementation of the algorithm uses $i-1$ minimum cut computations at iteration i giving a total of k^2 minimum cut computations. The cut-equivalent tree may be used to find all the cuts s_j at each iteration leading to the algorithm EFFICIENT that we present next.

10.4.3 An efficient algorithm and analysis

Both the algorithms GREEDY and SPLIT when implemented using a cut-equivalent tree essentially boil down to picking the lightest $k-1$ cuts represented in this tree. This is due to the fact that every cut represented by the cut-equivalent tree leaves exactly two components on its removal from the graph. Thus we have the following algorithm.

Algorithm EFFICIENT
input: Graph $G = (V, E)$, nonnegative edge capacities c.
output: A k-cut of capacity at most $2(1 - \frac{1}{k})$ of the minimum.
STEP 1: Construct a cut-equivalent tree T for G where every edge $e \in T$
  is labeled with the capacity of the corresponding cut in G;
STEP 2: Sort the cuts represented by the tree in nondecreasing order of capacity to get the list
  $g_1, g_2, \ldots, g_{n-1}$;
  output $g_1 \cup g_2 \cup \ldots \cup g_{k-1}$;

Performance guarantee

The performance ratio of the above algorithm is proved by comparing it with the optimal solution, say $a \subseteq E$. Let $V_1, V_2, \ldots V_k$ denote that connected components of $(V, E - a)$. Let $a_i = \delta(V_i)$. Recall that $c(a)$ denotes the total capacity of the set of edges a. Then, we have $c(a) = \frac{1}{2}\sum_{i=1}^{k} c(a_i)$. Assume without loss of generality that $c(a_1) \leq c(a_2) \leq \ldots c(a_k)$. The performance ratio is shown by arguing that

$$\sum_{i=1}^{k-1} c(g_i) \leq \sum_{i=1}^{k-1} c(a_i) \leq 2(1 - \frac{1}{k})c(a)$$

To show this, consider the partition of the node set V into blocks $V_1, V_2, \ldots V_k$ by the optimal k-cut a. The number of distinct edges of the tree T that crosses this partition is at least $k-1$ since T spans V. Let these edges of T be $e_1 = u_1v_1, e_2 = u_2v_2, \ldots e_p = u_pv_p$ where $p \geq k-1$. Consider an auxiliary graph $G(a)$ whose vertices are $v_1, v_2, \ldots v_k$ representing the contracted versions of the node sets $V_1, V_2, \ldots V_k$ respectively and whose edges are $e_1, \ldots, e_p$. Note that $G(a)$ may have parallel edges and is connected. Let $T(a)$ be a directed spanning tree of $G(a)$ rooted at the node v_k, with all edges directed towards the root.

Suppose the edge $e = uv$ is directed from the node v_i to v_j in $T(a)$. Then, $u \in V_i$ and $v \in V_j$ in G. Moreover $c(e)$ represents the minimum capacity of any cut separating u and v in G. But $a_i = \delta(V_i)$ is one such cut and so $c(e) \leq c(a_i)$. This charging argument charges the $k-1$ different edges e in the tree $T(a)$ to the cuts $a_1, \ldots a_{k-1}$. Note that v_k is the root and therefore a_k is not charged. Thus the capacity of the cuts $a_1, \ldots a_{k-1}$ is at least as much as the $k-1$ edges from the cut-equivalent tree that appear in $T(a)$. This in turn is at least as much as the $k-1$ lightest edges in the cut-equivalent tree $g_1, \ldots g_{k-1}$ that was picked by the algorithm EFFICIENT completing the proof.

10.5 Multiway Cuts

The multiway cut problem is to find a minimum capacity set of edges or nodes whose removal puts a given set of terminals $s_1, s_2, \ldots, s_k$ in an undirected graph in different connected components. We will examine an approximation algorithm for this problem based on a relaxation of an integer programming formulation of the problem.

10.5.1 Introduction

Let $G = (V, E)$ be a connected undirected graph, and let every edge $ij \in E$ be assigned a nonnegative capacity $c(ij)$. Assume also that every node v is assigned a capacity c_v, and a set of k *terminal* nodes $t_1, t_2, \ldots t_k$ are specified. The multiway edge/node cut problem is to find a minimum capacity set of edges/nodes such that after the removal of these edges/nodes, no pair of terminals are in the same connected component.

A minimal solution to a multiway edge cut problem with k terminals is a k-cut, namely, a set of edges that leaves at least k components, one for each terminal. While the minimum k-cut problem is polynomial-time solvable for fixed k, the multiway edge cut problem is NP-hard even for $k = 3$ [5].

Note that the multiway node cut problem is more general than the edge version, since the edge version can be transformed to an instance of the node version as follows: subdivide every edge to introduce a new node whose capacity equals the capacity of the edge it represents. All the resulting half-edges may be assumed to have capacity zero. In fact, this shows that if we solve the node version of the problem we can even model the version where both nodes and edges have capacities and we are allowed to find a mixed cut containing both nodes and edges to disconnect the terminals. In what follows, we will therefore focus only on the node-capacitated version and refer to this as the multiway cut problem.

We will examine an approximation algorithm with performance ratio $2(1-\frac{1}{k})$, same as for the k-cut problem, for the multiway node cut problem with k terminals. This algorithm, due to Garg, Vazirani and Yannakakis [8], uses an integer programming (IP) formulation of the problem and shows that the linear programming (LP) relaxation of the formulation has an optimum solution all of whose components are half-integers. Their proof also gives a method to find an approximate multiway cut from any optimal linear programming solution. In the next section, we present the IP formulation and derive it's LP dual. In the following section we show that the LP has a half-integral solution and derive the approximation algorithm.

10.5.2 IP formulation of multiway cuts

A direct formulation uses a 0-1 choice variable x_v to indicate if node v is chosen in the cut. How can we enforce that the set of nodes chosen separate a pair of terminals? A simple (but costly) method is to simply insist that *every* path between this pair of terminals contains at least one chosen node. This gives the following formulation. We use T to denote the set of terminal nodes.

$$
\begin{array}{llrcll}
\text{(IP)} & \text{minimize} & \sum_{v\in V\setminus T} c_v x_v & & & \\
& \text{subject to} & \sum_{v\in p_{ij}^k} x_v & \geq & 1 & \forall\ t_i - t_j \text{ paths } p_{ij}^k \\
& & x_v & \in & \{0,1\}, & \forall v \in V\setminus T
\end{array}
$$

In the formulation, p_{ij}^k represents the k^{th} distinct path between the pair of terminals t_i and t_j. The number of constraints in this formulation is enormous on account of this. Furthermore, since IP is NP-hard as well, we shall only be interested in solving the LP relaxation of this IP.

$$
\begin{array}{llrcll}
\text{(LP)} & \text{minimize} & \sum_{v\in V\setminus T} c_v x_v & & & \\
& \text{subject to} & \sum_{v\in p_{ij}^k} x_v & \geq & 1 & \forall\ t_i - t_j \text{ paths } p_{ij}^k \\
& & x_v & \geq & 0, & \forall v \in V\setminus T
\end{array}
$$

Compact reformulation

Due to the enormous number of constraints in this LP, it is not clear at first sight if this LP can be solved in polynomial time. However, it is not hard to show that the ellipsoid method can be used to accomplish this. To do this, the key requirement is a polynomial-time *separation oracle* (for more details on why the separation oracle is sufficient for optimization, see [13]). Given a candidate solution to the LP, namely, a set of values x_v for $v \in V \backslash T$, the oracle must determine if all the constraints of (LP) are satisfied by this solution, and if not, find a constraint that is violated (not satisfied) by the solution. This is not hard for the given set of constraints – let us design a separation oracle for enforcing the constraints involving a specific pair of terminals t_i and t_j. Suppose the candidate solution violates some constraint for this pair. Then there is a path between them with total x-value less than one. This implies also that the smallest node-weighted path between this pair using the x-values as node-weights is less than one. Since the x-values are nonnegative, we can determine the value of such a smallest node-weighted path by a straightforward modification of a shortest path algorithm such as Dijkstra's method. This gives us the separation oracle.

The idea for the separation oracle also allows us to come up with a more compact reformulation of (LP). We will sketch how to reformulate the set of inequalities enforcing the requirement or a pair of terminals t_i and t_j using shortest paths. To model the smallest node-weight path between t_i and t_j, we use a distance label $d_{ij}(v)$ at each node v. We set $d_{ij}(t_i)$ to be zero and $d_{ij}(t_k)$ for $k \neq i, j$ to be ∞. The intent is to enforce that $d_{ij}(t_j)$ represents the weight of the smallest node weight of path from t_i to t_j. So we enforce that for every node $v \in V \backslash T$, the label $d_{ij}(v)$ represents the smallest weight of a path from t_i to the node v. This we model as for every node $v \in V \backslash T$, $d_{ij}(v) \leq d_{ij}(u) + x_v$ for every neighbor u of v. This alone is not sufficient to enforce that d_{ij} denote node-distances, since for instance, setting all of these variables to zero is still legitimate. To complete the implement of our separation idea, we also add the inequality that $d_{ij}(t_j) \geq 1$. This "tests" if the d_{ij} values can be adjusted to denote weights of the smallest paths from t_i such that this value is at least 1 for t_j. Note that we have only added about as many inequalities as the number of edges of G and as many new variables as the number of nodes in G. We can repeat this for every pair of terminals to get a full compact formulation. It is easy to see that this compact formulation is equivalent to (LP) and therefore has the same solutions. We will continue to work with (LP) due to its simplicity, but remember that we can either use the ellipsoid algorithm or this compact formulation to find a solution in polynomial time.

A dual linear program

Next we consider the dual linear program of (LP). Recall that we have one variable in the dual for every constraint of (LP), thus a variable f_{ij}^k corresponding to constraint for the path p_{ij}^k. The coefficients of these variables in the objective is the right-hand side of (LP), namely, all ones. Finally, we have one constraint in the dual program for every variable in the primal, thus we have a constrain corresponding to every node $v \in V \backslash T$. The dual program is given below.

$$
\begin{array}{|llrcll|}
\hline
\text{(DP)} & \text{maximize} & \sum_{i,j,k} f_{ij}^k & & & \\
& \text{subject to} & \sum_{i,j,k:\ v \in p_{ij}^k} f_{ij}^k & \leq & c_v & \forall \text{ nodes } v \in V \backslash T \\
& & f_{ij}^k & \geq & 0, & \forall\ t_i - t_j \text{ paths } p_{ij}^k \\
\hline
\end{array}
$$

We can interpret the nonnegative value f_{ij}^k associated with the path p_{ij}^k as a flow along this path from t_i to t_j (the direction is immaterial). The constraint corresponding too a node $v \in V \backslash T$ then specifies that the total of all such flows going through v must not exceed its capacity c_v. The objective of (DP) is to maximize the total such flow that can be routed. In short, the LP-dual of the minimum multiway cut problem is to maximize the total flow that can sent between all the pairs of terminals simultaneously (sometimes also called concurrent flow) under node capacities c_v.

As a quick check, let us consider the case when $|T| = 2$. This corresponds to finding a minimum capacity set of nodes that separate a pair of given nodes, namely, a minimum node cut between the pair. The dual of this problem, as in the max-flow min-cut theorem, is to maximize the total flow that can be sent between the pair under the node capacities in the graph. This is exactly what our dual interpretation boils down to in this case.

Complementary slackness

We can use our linear formulations to get a solution to the relaxation of the multiway cut problem. In fact, we can use linear programming to derive have a pair of optimal solutions for the pair of programs (LP) and (DP). The theory of linear programming (see, e.g., [4]) specifies that the pair of solutions obey two complementary slackness conditions; In general, for a pair of optimal solutions to two dual linear programs, this condition states that if a variable is non-zero in one program, then the constrain corresponding to this variable in the dual program must be satisfied with equality (must be tight). Applying this condition to the variables in (LP) and (DP) gives the following.

1. Any node $v \in V \backslash T$ with a non-zero value of x_v in (LP) is saturated, i.e., the total flow through this vertex in (DP) is exactly its capacity c_v.
2. Any flow path f_{ij}^k that carries non-zero flow must have node-distance (sum of x-values from (LP)) exactly one.

These conditions will be useful later.

Conditions for optimality

Before we investigate further structure of the solution to (LP), we recall another useful fact from linear programming - a pair of solutions to two dual linear programs are both optimal if both are feasible for their respective programs and both have the same objective value. This is a simple consequence of weak duality. In our case, weak duality implies that any primal solution to (LP) has objective value at least as much as that of any solution (DP) as shown below. Here let x and f denote feasible solutions to (LP) and (DP) respectively.

$$\begin{aligned}
\sum_{v \in V \backslash T} c_v x_v &\geq \sum_{v \in V \backslash T} x_v \sum_{i,j,k : v \in p_{ij}^k} f_{ij}^k \\
&= \sum_{i,j,k} f_{ij}^k \sum_{v \in p_{ij}^k} x_v \\
&\geq \sum_{i,j,k} f_{ij}^k \cdot 1
\end{aligned}$$

Thus if the pair x and f achieve the same objective value, then each has achieved its extreme value.

10.5.3 A half-integrality property

In this section, we will show that for any specification of terminals and for any capacity values on the nodes, there is always an optimal solution to (LP) that is half-integral. Assume that x and f represent a pair of dual optimal solutions in the following.

Other half-integral LPs

An immediate application of half-integrality will be to round up all the variables of (LP) that are non-zero. Since the constraint matrix has no negative co-efficient, it is easy to check that this maintains feasibility of the solution. Furthermore, since any variable whose value increases to one must have been set to a half to start with, we scale the linear solution by a factor of at most two to convert it to a rounded integer solution.

A similar simple rounding method of rounding is also well known for the vertex cover problem, and the half-integrality of the corresponding linear program is also known [24]. The most general case currently know for which the linear program has half-integral solutions is one where the constraint matrix has at most two non-zero entries in each column that are +1 or -1 [17].

A decomposition

Given an optimal solution x to (LP), we think of these as node-lengths as in the formulation. For each terminal t_i, we can then identify a subset of nodes T_i, all of distance zero from t_i under this length. In other words, T_i defines a region of the graph containing all nodes that can be reached via paths involving only zero-length nodes from t_i. We must address the issue of the node-length of any terminal for this definition. We can assume that this is zero for every terminal, and check that this is consistent with our definition of the length of a path in the formulation, even if we include the either or both endpoints of the path in computing the length of the path. Note that by the feasibility of (LP), the regions $T - i$ corresponding to different terminals t_i are disjoint.

Next we define the boundary of T_i denoted C_i as the set of nodes in $V \backslash T_i$ that are neighbors of some node in T_i. If a node v belongs to two different boundaries C_i and C_j, then $x_v = 1$ due to the distance constraint between t_i and t_j – there is a path between these nodes in which only v has non-zero node-length. Define $C = \cup_{i=1}^{k} C_i$ and $C^1 \subseteq C$ as the set of nodes in at least two different boundaries, and therefore set to node-length one. Let $C' = C \backslash C^1$.

Lemma 10.16 *Let f_{ij}^k be set to a non-zero value in (DP). Then the path p_{ij}^k contains exactly one node from C^1 or exactly two nodes from C'.*

Proof: Let the first and last nodes along p_{ij}^k in C_i and C_j be v_i and v_j respectively.

(i) $v_i = v_j$.
In this case, this node is in C^1 and we have exactly this node from C^1 and no other node of C in the path.

(ii) $v_i \neq v_j$.
Suppose that either v_i or v_j is in C^1. Then the total length of p_{ij}^k is greater than one, contradicting the second complementary slackness condition. Thus, both these nodes are in C'.

Next suppose that there is another node $v \in C'$ distinct from v_i and v_j in the path, and let $v \in C_k$. In this case, we can find a path between t_k and either of t_i or t_j with node length less than one contradicting the feasibility of x. To see this, consider the prefix of the path p_{ij}^k from t_i to node v, and extend this to a path ending in t_k via nodes in C_k. then length of this path is strictly less than that of p_{ij}^k since we do not have node v_j that has non-zero length and the extension has zero length. Finally, the length of path p_{ij}^k is exactly one from the second complementary slackness condition, giving that the new path has length less than one, a contradiction.

▮

Half-integrality

Theorem 10.17 *There is an optimal solution to (LP) that is half-integral.*

Proof: The half-integral solution we propose can be constructed from *any* given optimal solution x. Using these as node-distances, we compute C^1 and C'. Next we consider a different solution x^* to (LP) as follows.

$$x_v^* = \begin{cases} 1 & \forall v \in C^1 \\ \frac{1}{2} & \forall v \in C' \\ 0 & \text{otherwise} \end{cases}$$

We show next that x^* is feasible and optimal for (LP).

Any path from t_i and t_j must use nodes from both C_i and C_j, not necessarily distinct. Let v_i and v_j the corresponding nodes. If these are the same, then this node must be in ^{1} and the length of this path is one. Otherwise, the path uses at least two distinct nodes from C' and hence the length is at least one again. This shows that x^* is feasible.

To show optimality, we show that its value is equal to that of the dual objective associated with x. The value of the dual is the total concurrent flow between the terminals. From lemma 10.16, every flow path carrying non-zero flow crosses either one node in C^1 exactly once or exactly two distinct nodes in C'. Thus the total value of all the flow can be accounted as $\sum_{v \in C^1} c_v + \frac{1}{2}\sum_{v \in C'} c_v$. This is exactly the value of the solution x^*. hence x^* is optimal. ▮

An approximation algorithm

We present an algorithm that exploits the half-integrality property to find a rounded solution.

Algorithm MULTIWAY CUT
input: Graph $G = (V, E)$, nonnegative node capacities c,
and a node subset T of terminals with $|T| = k$.
output: A multiway cut for T of capacity at most $2(1 - \frac{1}{k})$ of the minimum.
STEP 1: compute an optimal solution x to (LP);
STEP 2: **for** every terminal $t_i \in T$, compute the set of nodes T_i
reachable from t_i via paths with zero x-value, and C_i, the node neighbors of T_i;

Set $C \leftarrow \cup_i C_i$;
STEP 3: **for** every i, compute C_i', the set of nodes
in C_i that are not in any other C_j for $j \neq i$;
STEP 3: find C'_{max}, the set of maximum node capacity among the C_i''s;
output $C - C'_{max}$;

It is clear that the algorithm can be implemented in polynomial time by using a node-weighted version of Dijkstra's algorithm. It is not hard to show feasibility of the output solution as well. Any path between two terminals has to either contain one node in C^1 or at least two nodes in C_i's. Since we only exclude one such set C_i' in forming the final solution, every path must cross at least one node from the output set showing feasibility.

Performance guarantee

The performance ratio of the algorithm follows almost directly from Theorem 10.17. Define $C' = C - C^1$ where C and C^1 are computed by the algorithm. By a simple averaging argument, it follows that $c(C'_{max}) \geq \frac{1}{k} c(C')$ where c denotes the capacity of the set.

$$
\begin{aligned}
c(C - C'_{max}) &= c(C) - c(C'_{max}) \\
&= c(C^1) + \sum_{i=1}^{k} c(C_i') - c(C'_{max}) \\
&\leq c(C^1) + (1 - \frac{1}{k}) \sum_{i=1}^{k} c(C_i') \\
&\leq 2(1 - \frac{1}{k})(c(C^1) + \frac{1}{2} \sum_{i=1}^{k} c(C_i'))
\end{aligned}
$$

By the theorem, the right hand side of the last inequality is at most $2(1 - \frac{1}{k})$ times the value of an optimal solution to (LP). This is a value of a relaxation of the multiway cut problem and hence is a lower bound on the capacity of the minimum multiway cut, completing the proof.

Approximate min-max relation

Note that our proof generalizes the classical max-flow min-cut theorem for the node-capacitated case in a nice way. When $k = 2$, the performance ratio is one giving us that the cut we find by the rounding algorithm is a minimum cut and our results show that the value of a maximum node-capacitated flow is equal to the value of the minimum cut; this is the max-flow min-cut theorem.

For larger k, we continue to have an exact relation between the flow and cut values, if we allow both of them to be fractional. This is what linear programming duality implies for the pair of programs (LP) and (DP). However, if we strengthen our notion of a multiway cut to mean an integral solution to (LP), the theorem guarantees that we can achieve a total (fractional) flow of value at least a fraction half of the minimum such integral cut.

Recent Improvement

For the edge-capacitated case of the multiway cut problem described above, Călinescu et al. [2] have recently devised a $\frac{3}{2}$-approximation algorithm via a novel randomized rounding of a strengthened linear programming relaxation for the problem.

10.6 Exercises

1. A legal ordering of the nodes of G identifies a pair of vertices, v_{n-1} and v_n such that the cut around the singleton set $\{v_n\}$ is a minimum cut separating v_{n-1} and v_n. Show that if G has at least three vertices, there is a second pair of vertices with this property.
2. Does the framework of the node identification algorithm work for finding minimum node-cuts in an undirected graph? Why or why not? Answer the same question for finding a node-cut separating some pair of vertices using legal orderings.
3. Does the upper bound of $\frac{n(n-1)}{2}$ on the number of min cuts of an n-node graph that we proved for undirected graphs also hold for directed graphs? Why or why not? Also, is this bound tight for undirected graphs? (read as, are there capacitated undirected graphs which have as many as $\frac{n(n-1)}{2}$ min cuts?)
4. For any constant positive half-integer $\alpha \geq 1$, show that the number of α-minimum cuts in an undirected graph on n nodes is at most $O(n^{2\alpha})$. (An α-min cut is one whose capacity is at most α times that of a minimum cut).
5. Is there a compact representation of all the $n(n-1)$ minimum cuts in a directed graph? (The capacity of a cut separating nodes x and y is defined the sum of the capacities of all the edges coming out of the component containing x.) For instance, could you hope to represent all of the by a tree with only $n-1$ values. What is the maximum number of distinct values of minimum cuts you can find in a directed graph?
6. We saw in class that a cut-equivalent tree is also a flow-equivalent tree. Show that the reverse implication does not always hold.
7. Cheng and Hu showed that flow-equivalent trees exist even when the capacity of a cut is defined by an arbitrary general function, i.e., a function $f : S \Rightarrow R^+$ that assigns a nonnegative value to every subset $S \subset V$ of vertices of the given undirected graph. Prove the Cheng-Hu result that a flow-equivalent tree *exists* in this case.

   Extra credit: Assume that you have an oracle for determining a minimum cut separating a pair of nodes. Devise a method to construct such a flow-equivalent tree using at most $n-1$ calls to the oracle.
8. Show that every graph has a flow-equivalent tree that is a (Hamiltonian) path.
9. Give an example to show that the integrality gap of 2 is tight for the node-multiway cut problem. Do the same for the edge-multiway cut case.
10. Is the analysis of the performance guarantee for the k-cut problem tight? Show an example to illustrate this.

11. Does the hereditary property of approximate solutions for the k-cut problem carry over to optimal solutions? In other words, note that the first $k' - 1$ cuts among the $k - 1$ chosen as a 2-approximate solution to the k0cut problem, form a 2-approximate solution to the k' cut problem (for $k' < k$). Is it true that any optimal solution for such a k'-cut problem can be extended to an optimum solution for the k-cut problem?

12. The Optimal Tree Arrangement problem (OTA) is defined on an unweighted undirected graph G as follows. The goal is to find a spanning tree T on the vertices of the graph minimizing the sum over all the edges of G, of the distance in T between the endpoints of the edge. Give a polynomial time algorithm for the OTA problem.

13. The Multicommodity Flow Tree problem (MFT) is defined on an undirected graph G with nonnegative capacities c on the edges as follows. The goal is to find a spanning tree T with nonnegative capacities assigned to its edges such that for every pair of vertices s, t in the graph, the flow between s and t in the tree T is at least the maximum $s - t$ flow in the graph G. In this way, the tree T supports all the flows that G does. The objective is to minimize the the maximum capacity of any edge in T. Give a polynomial time solution to the MFT problem.

14. If all cuts in a cut-equivalent tree of a connected undirected graph are even, is the graph Eulerian (i.e., have even degree at all the nodes)?

15. (Extra credit) In showing a 2-approximation algorithm for node-multiway cuts, we showed half-integrality of optimal solutions for a certain linear relaxation of the problem. The dual to this relaxation is a linear program whose objective is to maximize the total concurrent flow between terminals specified in the problem. Prove or disprove: for any specification of integral capacities, there is a maximum flow solution that is also half-integral.

Bibliography

[1] A. Benczúr, *Counterexamples for directed and node capacitated cut-trees*, SIAM J. Comput. **24:3** (1995), pp. 505-510.

[2] G. Călinescu, H. Karloff, and Y. Rabani, *An improved approximation algorithm for Multiway Cut*, Proc. STOC '98 (1998), 48-52.

[3] C. Chekuri, A. Goldberg, D. Karger, M. Levine, and C. Stein, *Experimental study of minimum cut algorithms*, NECI TR 96-132 (October 1996). Available from `http://www.neci.nj.nec.com/homepages/avg/webpub/webpub.html`.

[4] V. Chvátal, *Linear Programming*, (1983) W. H. Freeman and Company.

[5] E. Dalhaus, D. S. Johnson, C. H. Papadimitriou, P. Seymour, and M. Yannakakis, *The complexity of multiway cuts*, Proc. 24th Annual ACM Symp. on Theory of Comput. (1992), pp. 241-251.

[6] P. Elias, A. Feinstein and C. E. Shannon, *Note on maximum flow through a network*, IRE Trans. Inform. Theory **IT-2** (1956), pp. 117-119.

[7] L. R. Ford and D. R. Fulkerson, *Maximal flow through a network*, canad. J. Math. **8** (1956), pp. 399-404.

[8] A. Frank, T. Ibaraki, and H. Nagamochi, *On Sparse Subgraphs Preserving Connectivity Properties*, J. Graph Theory **17:3** (1993), pp. 275-281.

[9] N. Garg, V. Vazirani and M. Yannakakis, *Multiway cuts in directed and node-weighted graphs*, Proc. ICALP '94 (1994), pp. 487-498.

[10] A. V. Goldberg and R. E. Tarjan, *A New Approach to the Maximum Flow Problem*, J. Assoc. Comput. Mach. **35** (1988), pp. 921-940.

[11] O. Goldschmidt and D. S. Hochbaum, *Polynomial algorithm for the k-cut problem*, Proc. 29th Annual Symp. on Foundations of Comp. Sci. (1988), pp. 444-451.

[12] R. Gomory and T. C. Hu, *Multi-terminal network flows*, J. SIAM **9** (1961), pp. 551-570.

[13] M. Grötschel, L. Lovász and A. Schrijver, *Geometric algorithms and combinatorial optimization*, (1988) Springer-Verlag.

[14] D. Gusfield, *Very simple methods for all pairs network flow analysis*, SIAM J. Comput. **19:1** (1990), pp. 143-155.

[15] J. Hao and J. B. Orlin, *A Faster Algorithm for Finding the Minimum Cut in a Directed Graph*, J. Algorithms **17** (1994), pp. 424-446.

[16] D. Hartvigsen, and F. Margot, *Multiterminal flows and cuts*, Oper. Res. Lett. **17** (1995), pp.201-204.

[17] D. S. Hochbaum, N. Meggido, J. Naor and A. Tamir, *Tight bounds and 2-approximations for integer programs with two variables per inequality*, Math. Programming **62** (1993), pp. 69-83.

[18] D. Karger and C. Stein, *An* $\tilde{O}(n^2)$ *algorithm for finding minimum cuts*, Proc. 25th Annual ACM Symp. on Theory of Computing (1993), pp. 757-765.

[19] V. King, S. Rao, and R. E. Tarjan, *A Faster Deterministic Maximum Flow Algorithm*, J. Algorithms **17** (1994), pp. 447-474.

[20] W. Mader, *Über minimal n-fach zusammenhängende unendliche Graphen und ein Extremal problem*, Arch. Mat. **23** (1972), pp. 553-560.

[21] K. Menger, *Zur allgemeinen Kurventheorie*, Fund. Math. **10** (1927), pp. 96-115.

[22] H. Nagamochi and T. Ibaraki, *Computing Edge-Connectivity in Multigraphs and Capacitated Graphs*, SIAM J. Disc. Meth., **5** (1992), pp. 54-66.

[23] H. Nagamochi and T. Ibaraki, *A Linear Time Algorithm for Finding a Sparse k-Connected Spanning Subgraph of a k-Connected Graph*, Algorithmica **7** (1992), pp. 583-596.

[24] G. Nemhauser and L. Trotter, *Vertex packings: Structural properties and algorithms*, Math. Programming **8** (1975), pp. 232-248.

[25] T. Nishizeki and S. Poljak, *k-Connectivity and Decompositions of Graphs into Forests*, Disc. Appl. Math. **55:3** (1994), pp. 295.

[26] H. Saran and V. Vazirani, *Finding k-cuts within twice the optimal*, SIAM J. on Computing **24:1** (1995), pp. 101-108. Extended abstract appeared in Proc. 32nd Annual Symp. on Foundations of Comp. Sci. (1991), pp. 743-751.

[27] M. Stoer and F. Wagner, *A Simple Min Cut Algorithm*, Algorithms - ESA '94 **LNCS 855** (1994), pp. 141-147.

Chapter 11

Graph Separators

The focus of this note is on the sparsest cut in a capacitated, undirected graph. It turns out that the problem of finding a sparsest cut is NP-hard, yet the problem is interesting and important. The problem has applications to divide-and-conquer algorithms for approximately solving many hard combinatorial problems, such as balanced cuts, feedback arc set, minimum fill-in ordering, VLSI layout, minimum crossing-number layout, etc.

First, we will motivate and prove a key theorem due to Leighton & Rao showing that the objective value of a sparsest cut is within a logarithmic factor of the optimal value of an LP (linear programming) relaxation of the sparsest cut problem, and moreover, there is an efficient algorithm for finding such an approximately sparsest cut, i.e., a cut whose objective value is within a logarithmic factor of the objective value of the sparsest cut.

After that, we introduce the notions of metric spaces and ℓ_1-embeddings, discuss some of the connections to sparsest cuts and multicommodity flows, and prove a generalization of the Leighton-Rao theorem and algorithm, based on results of Bourgain and others. For completeness, we include a proof of Bourgain's theorem: by allowing a logarithmic distortion, any metric space has an ℓ_1-embedding.

Finally, we discuss a paradigm for approximately solving hard combinatorial problems by using a subroutine for finding approximately sparsest cuts. This paradigm easily gives a log-squared approximation guarantee.

11.1 The sparsest cut problem and an LP relaxation

Let $G = (V, E)$ be an undirected graph, and let n denote $|V|$. Let $u : E \to \Re_+$ assign a nonnegative capacity to each edge. Throughout this chapter, for a cut $\delta(S)$ of G, the *ratio* means the quantity $\dfrac{u(\delta(S))}{|S| \cdot |V \backslash S|}$, and a *sparsest cut* is one whose ratio is minimum. (Note that $\delta(S)$ is a set of edges, but we also associate the node set S with it.)

Let us write down an LP formulation of the sparsest cut problem. For motivation, let us quickly recapitulate the LP for the minimum s-t cut problem. The goal is to assign a nonnegative weight d_e to each edge $e \in E$ such that $\sum_{e \in E} u_e d_e$ is minimized, while the weight of a shortest path between s and t (w.r.t. edge weights d) is at least one. It is easily checked that the 0-1 incidence vector of every s-t cut is a feasible solution to this LP, i.e., taking $d_e = 1$ iff e is in the s-t cut satisfies all constraints of the LP. Moreover, by the max-flow min-cut theorem, one optimal solution to the LP is given by the 0-1 incidence vector of a minimum s-t cut.

Here is our first attempt at formulating the sparsest cut problem as a "fractional LP", i.e., as an LP where the objective function is rational rather than linear. For two nodes v and w, we use $\mathcal{P}_{v,w}$ to denote the set of all v-w paths in the graph; the cardinality of $\mathcal{P}_{v,w}$ may be exponential in n.

$$\begin{array}{llllll}
\text{(F1) } z = \text{minimize} & \dfrac{\sum_{e\in E} u_e d_e}{\sum_{v,w\in V} \lambda_{v,w}} & & & \\
& & & & \\
\text{subject to} & \sum(d_e \mid e \in P) & \geq & \lambda_{v,w} & \forall \text{ paths } P \in \mathcal{P}_{v,w} \\
& d_e & \geq & 0 & \forall e \in E \\
& \lambda_{v,w} & \geq & 0 & \forall v, w \in V.
\end{array}$$

Above, the edge weights d_e are supposed to represent incidence vectors of cuts, and the variables $\lambda_{v,w}$ are supposed to represent the weight of a shortest path (w.r.t. d) between nodes v and w. To obtain an LP formulation, we remove the denominator from the objective function and add an extra constraint that the denominator be at least one.

$$\begin{array}{llllll}
\text{(F2) } z = \text{minimize} & \sum_{e\in E} u_e d_e & & & \\
\text{subject to} & \sum_{v,w\in V} \lambda_{v,w} & \geq & 1 & \\
& \sum(d_e \mid e \in P) & \geq & \lambda_{v,w} & \forall \text{ paths } P \in \mathcal{P}_{v,w} \\
& d_e & \geq & 0 & \forall e \in E \\
& \lambda_{v,w} & \geq & 0 & \forall v, w \in V.
\end{array}$$

At this point, the reader may like to prove that the formulations (F1) and (F2) are equivalent in the following sense: for every optimal solution d, λ of (F1) there exists a feasible solution of (F2) with the same objective value, and vice versa. Here is another LP formulation (F3) that we claim is equivalent to (F2). The reason for studying (F3) is that it has only $O(n^3)$ constraints and $O(n^2)$ variables, i.e., it is a compact LP, in contrast with (F2) (and (F1)) whose number of constraints may be exponential in n. Again, we will leave it to the reader to prove that (F2) and (F3) are equivalent. In the LP (F3), the goal is to find a metric d on V such that the "total weighted capacity" is minimum and the sum over all entries of the metric is at least one.

$$\begin{array}{llllll}
\text{(F3) } z = \text{minimize} & \sum_{e\in E} u_e d_e & & & \\
\text{subject to} & \sum_{v,w\in V} d_{v,w} & \geq & 1 & \\
& d_{v,w} & \leq & d_{v,r} + d_{r,w} & \forall r, v, w \in V \\
& d_{v,r} & \leq & d_{v,w} + d_{w,r} & \forall r, v, w \in V \\
& d_{w,r} & \leq & d_{w,v} + d_{v,r} & \forall r, v, w \in V \\
& d_{v,w} & \geq & 0 & \forall v, w \in V.
\end{array}$$

First, focus on the formulation (F1). Consider the 0-1 incidence vector χ of an arbitrary cut, say, $\delta(S)$ such that both shores ($G[S]$ and $G[V\backslash S]$) are connected. If we take $d = \chi$ and $\lambda_{v,w} = 1$ or 0 depending on whether or not v and w are separated by $\delta(S)$ (i.e., $\lambda_{v,w} = 0$ if either $v, w \in S$ or $v, w \notin S$, otherwise $\lambda_{v,w} = 1$), then it is easily seen that d, λ forms a feasible solution whose

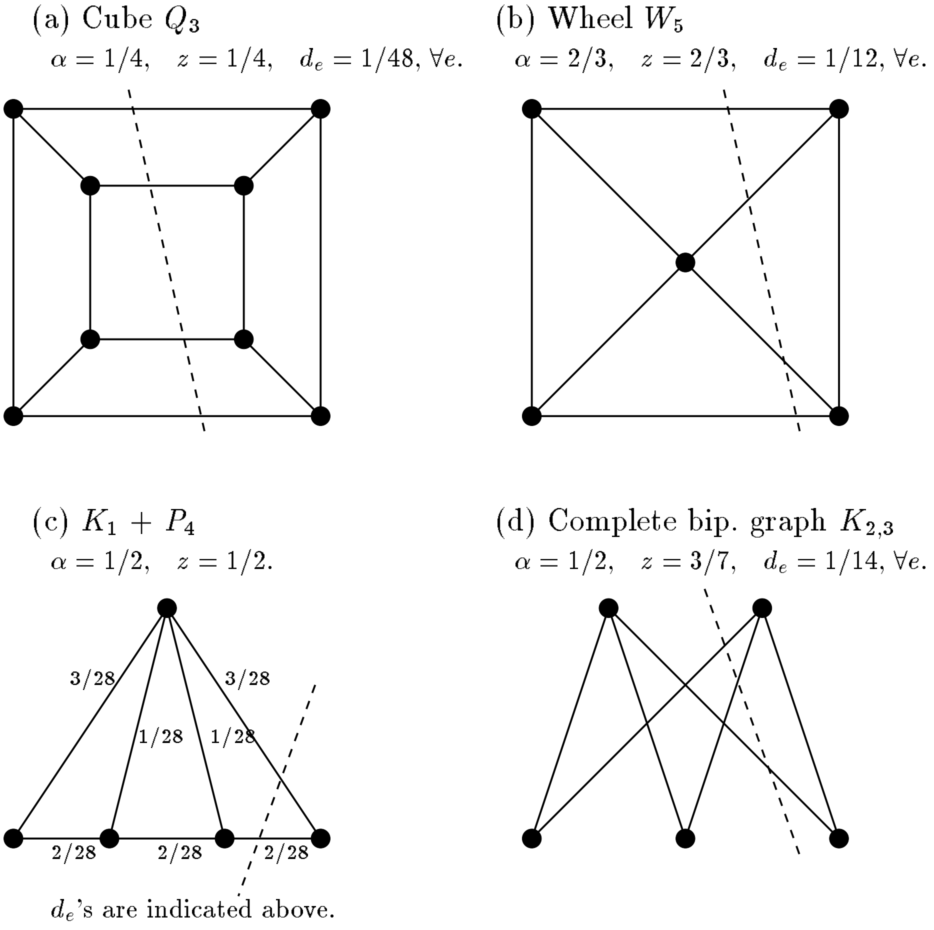


Figure 11.1: Sparsest cuts and optimal solutions to the LP (F2) for four graphs (a) the cube Q_3 (b) the wheel W_5, (c) $K_1 + P_4$, and (d) $K_{2,3}$. For each graph a sparsest cut is indicated by a dashed line. The ratio of the sparsest cut is given as α, the optimal value of LP (F2) is given as z, and the optimal solution of (F2) is given as $d_e, \forall e \in E$.

objective value equals the ratio of the cut $\delta(S)$. Hence, the optimal value of (F1) is a lower bound on the ratio of a sparsest cut.

The big question is: does the optimal value of (F1) equal the ratio of a sparsest cut, and if not, then how much can these values differ? Let us look at a few examples. It is convenient to work with the formulation (F2) or (F3) rather than (F1). Figure 11.1 shows four graphs, the cube Q_3 on 8 nodes, the wheel W_5 on 5 nodes, the graph $K_1 + P_4$ (5-cycle with two adjacent diagonals), and the complete bipartite graph $K_{2,3}$. We take each edge to have unit capacity. The cube Q_3 has sparsest-cut ratio 1/4 and (F2) optimal value $z = 1/4$. The wheel W_5 has sparsest-cut ratio 2/3 and (F2) optimal value $z = 2/3$. The graph $K_1 + P_4$ has sparsest-cut ratio 1/2 and (F2) optimal value $z = 1/2$. The graph $K_{2,3}$ has sparsest-cut ratio 1/2 and (F2) optimal value $z = 3/7$. (For the three graphs Q_3, W_5 and $K_{2,3}$, the optimal solution has $d_e = z/|E|$ for each edge e, and has $\lambda_{v,w}$ equal to the weight of a shortest v-w path w.r.t. edge weights d.) As a side remark, we mention that $K_{2,3}$ is the smallest graph such that the sparsest-cut ratio differs from the (F2) optimal value. (Q: IS THIS ASSUMING UNIT CAPACITIES ONLY?)

Let us look at another example (due to Leighton & Rao) where the sparsest-cut ratio is $\Omega(\log n)$ times the optimal value of (F2). A bounded-degree expander G is a k-regular graph with $k = O(1)$ (i.e., every node has degree $k = O(1)$) such that for every set of nodes S with $|S| \leq n/2$,

$$u(\delta(S)) \geq c \cdot |S|,$$

where c is an absolute constant (note that every edge has unit capacity). Clearly, the sparsest-cut ratio is $\geq c/n$. Now, consider z, the optimal value of (F2). We claim that there are $n^2/4$ pairs of nodes v, w such that $\lambda'_{v,w} \geq \log_k(n/2) = (\log n/2)/(\log k) \geq c' \log n$, where $\lambda'_{v,w}$ denotes the number of edges in a shortest v-w path of G, and $c' < 1$ is an absolute constant. To see this, just note that for a fixed node w the number of nodes within a radius $1, 2, \ldots, i$ of w is at most $(1+k), (1+k^2), \ldots, k^i$, i.e., $|\{v \mid \lambda'_{v,w} \leq i\}| \leq k^i$, for $i \geq 3$. Now consider the feasible solution to (F2) given by $d_e = (c'(\log n)n^2/4)^{-1} = 4/(c'n^2 \log n)$ for each edge e, and $\lambda_{v,w}$ equal to the weight of a shortest v-w path of G w.r.t. edge weights d. The constraint $\sum\limits_{v,w \in V} \lambda_{v,w} \geq 1$ holds by our claim, and so d, λ is indeed a feasible solution. Since each edge has unit capacity and $|E| = kn/2$, the objective value of this feasible solution is $\sum\limits_{e \in E} u_e d_e = 4|E|/(c'n^2 \log n) = 2k/(c'n \log n) = \Omega(1)/(n \log n)$. Since z, the optimal value of (F2), is at most this value, we are done: the sparsest-cut ratio is at least $\Omega(\log n)$ times z.

The main result in the next section is that the expander example is the worst possible (up to constant factors), because there always exists a cut whose ratio is at most $O(\log n)$ times the optimal value of (F2).

11.2 The sparsest cut problem: A region-growing algorithm

The strategy is to start with an optimal solution $d : E \to \Re_+$ to the LP, and then to run a "region-growing procedure" on the graph with edge weights d to find a cut whose ratio is at most $O(\log n)z$, where z is the optimal value of the LP. The intuition behind the region-growing procedure is to imitate Dijkstra's shortest paths algorithm to construct a moat packing $\{y_S \mid S \subset V\}$. Throughout this section a region means a set of nodes. The procedure keeps growing the current region as long as there is sufficient expansion, and as a result, when the procedure stops, the region is shallow, i.e., it has small radius.

The inputs to the region-growing procedure are the edge weights d, the optimal LP value $z = \sum_{e\in E} u_e d_e$, a set of candidate root nodes $V' \subset V$, and a parameter ϵ (whose role will be clear later). For our application to sparsest cuts, $V' = V$, but other choices of V' are needed in other applications. We start by choosing a root node $r \in V'$. The region-growing procedure maintains a set of nodes A, called the *active set*, and for each such set there is a variable y_A. Initially, the active set is the singleton given by the root, $\{r\}$. One important constraint holds throughout. The y-variables must form a *packing* w.r.t. the edge weights d, i.e., $\sum\limits_{S|e\in\delta(S)} y_S \le d_e, \forall e \in E$. The weight of r is defined to be $wt(r) = \dfrac{z}{|V'|} = \dfrac{z}{n}$, and the *weight* of an active set A is defined to be $wt(A) = wt(r) + \sum\limits_{S\subset A} y_S u(\delta(S))$. The algorithm increases the variable y_A till the packing constraint becomes tight, i.e., till some edge $e = pq$ with $p \in A$, $q \notin A$, has $d_e = \sum_{S\subset A,\ e\in\delta(S)} y_S$. Then we make $A' = A \cup \{q\}$ the active set and check whether

$$u(\delta(A')) > \epsilon \cdot wt(A'). \tag{11.1}$$

If yes, then we start increasing $y_{A'}$, and continue as above. For a set A, define the radius to be $rad(A) = \sum_{S\subset A} y_S$.

Lemma 11.1 *Let A^* denote the active set at the termination of the region-growing procedure, and w.l.o.g. assume that y_{A^*} is positive. Then*

$$rad(A) < \frac{\ln(1+|V'|)}{\epsilon} \le \frac{\ln(1+n)}{\epsilon}.$$

Proof: Let the active sets with positive y-variables be $S_1, S_2, \ldots, S_\ell$ where the sets became active in this sequence. For notational convenience, let $y_1, y_2, \ldots, y_\ell$ denote the y-variables of $S_1, S_2, \ldots, S_\ell$, and let $U_1, U_2, \ldots, U_\ell$ denote the capacities of the cuts $\delta(S_1), \delta(S_2), \ldots, \delta(S_\ell)$. Focus on $wt(S_\ell) = wt(r) + \sum_{i=1}^{\ell} y_i U_i$. Since the y-variables form a packing, we have

$$\sum_{i=1}^{\ell} y_i U_i = \sum_{i=1}^{\ell} y_{S_i} \left(\sum_{e\in\delta(S_i)} u_e \right) = \sum_{e\in E} u_e \left(\sum_{S_i|e\in\delta(S_i)} y_{S_i} \right) \le \sum_{e\in E} u_e d_e = z.$$

Hence, $wt(S_\ell) \le (z/n) + z$.

We can derive a lower bound on $wt(S_\ell)$ too. The weight of S_i can be written as

$$wt(S_i) = wt(S_{i-1}) + y_i U_i \ge wt(S_{i-1}) + \epsilon \cdot y_i wt(S_i), \tag{11.2}$$

where we used the fact that $U_i > \epsilon \cdot wt(S_i)$, for $i = 1, 2, \ldots, \ell - 1$, because the algorithm did not terminate with S_i, whereas the last active set $S_\ell = A^*$ has $U_\ell = \epsilon \cdot wt(S_\ell)$. This gives a useful inequality,

$$wt(S_i) \ge \frac{wt(S_{i-1})}{(1 - \epsilon y_i)},$$

where $\epsilon y_i < 1$, as can be seen from inequality (11.2), since $wt(S_{i-1})$ is positive. Hence,

$$wt(S_\ell) \geq \frac{wt(S_0)}{(1-\epsilon y_1)\ldots(1-\epsilon y_\ell)}.$$

Combining this inequality and the upper-bound on $wt(S_\ell)$ obtained in the previous paragraph, we have

$$\frac{z}{n}(n+1) \geq \frac{z}{n}\left(\frac{1}{(1-\epsilon y_1)\ldots(1-\epsilon y_\ell)}\right).$$

Taking natural logarithms on both sides,

$$\ln(n+1) \geq \sum_{i=1}^{\ell} \ln(1-\epsilon y_i)^{-1} \geq \epsilon \sum_{i=1}^{\ell} y_i,$$

since $\ln(1/(1-y)) \geq y$, for all y with $0 < y < 1$.

The proof is done since $rad(A^*) = \sum_{i=1}^{\ell} y_i \leq \epsilon^{-1}\ln(n+1)$. ∎

Corollary 11.2 *For all pairs of nodes $p, q \in A^*$, $d(r,p) \leq rad(A^*)$ and $d(p,q) \leq 2rad(A^*)$. Here, $d(p,q)$ is the weight of a shortest path between nodes p and q in the graph G w.r.t. the edge weights d.*

Lemma 11.3 *Let $A_1^*, A_2^*, \ldots, A_h^*$ ($h \leq |V'| \leq n$) be the regions obtained by iterating the region-growing procedure on the graphs $G_1 = G, G_2 = G_1 - A_1^*, \ldots, G_j = G_{j-1} - A_{j-1}^*, \ldots, G_h = G_{h-1} - A_{h-1}^*$. That is, A_i^* ($1 \leq i \leq h$) is the active set at the termination of the ith iteration, where the graph is taken to be $G_i = G - A_1^* - \ldots - A_{i-1}^*$. Then (i) the total capacity of the union of the coboundaries in G, $\mu = u(\delta(A_1^*) \cup \ldots \cup \delta(A_h^*))$ is $\leq 2\epsilon z$. Also, (ii) $\sum_{i=1}^{h} u(\delta(A_i^*)) \leq 4\epsilon z$.*

Proof: Let $\delta_i(S)$ denote the set of edges of G_i that have exactly one end in $S \subseteq V$. Note that $\delta(A_1^*), \delta(A_2^*), \ldots, \delta(A_h^*)$ may have edges in common since these are coboundaries in G, but $\delta_1(A_1^*), \delta_2(A_2^*), \ldots, \delta_h(A_h^*)$ have no edges in common by definition of $G_1, G_2, \ldots, G_h$. Also, every edge of $\delta(A_1^*) \cup \ldots \cup \delta(A_h^*)$ is present in $\delta_1(A_1^*) \cup \ldots \cup \delta_h(A_h^*)$. By termination of the region-growing procedure, $u(\delta_i(A_i^*)) \leq \epsilon \cdot wt(A_i^*)$, for each $i = 1, 2, \ldots, h$, where $wt(A_i^*) = wt(r_i) + \sum_{S \subseteq A_i^*} y_S u(\delta(S))$, and r_i is the root at the ith iteration. Hence,

$$\begin{aligned}
\mu = \sum_{i=1}^{h} u(\delta_i(A_i^*)) \quad &\leq \quad \epsilon \sum_{i=1}^{h} wt(A_i^*) = \\
&\epsilon \left(\sum_{i=1}^{h} \sum_{S \subseteq A_i^*} y_S u(\delta_i(S)) + \sum_{i=1}^{h} wt(r_i) \right) \leq \\
&\epsilon \left(\sum_{e \in E(G)} u_e d_e + \sum_{i=1}^{h} \frac{z}{n} \right) \leq 2\epsilon z.
\end{aligned}$$

In the second last inequality, $\sum_{S\subseteq A_i^*} y_S u(\delta_i(S)) \leq \sum_{e\in E(G)} u_e d_e$ holds by the packing constraints.

The last part of the lemma follows from the first part because an edge in two coboundaries $u(\delta(A_i^*))$ and $u(\delta(A_j^*))$, $1 \leq i < j \leq h$, contributes once to the first part but two times to the second part. ∎

The next theorem is essentially due to Leighton & Rao, but we give a tighter version due to Garg, Vazirani & Yannakakis. The proof due to Garg, Vazirani & Yannakakis is based on the original proof of Leighton & Rao, but is cleaner and simpler than the original proof.

For a cut $\delta(S)$ of G, recall that the ratio means the quantity $\dfrac{u(\delta(S))}{|S|\cdot|V\backslash S|}$.

Theorem 11.4 (Leighton & Rao (1988)) *Let $d : E \to \Re_+$ be the optimal solution to the LP, and let $z = \sum_{e\in E} u_e d_e$ be the optimal value of the LP. Also, let α be the ratio of the sparsest cut of G, $\alpha = \min_{S\subset V} \dfrac{u(\delta(S))}{|S|\cdot|V\backslash S|}$. Then*

$$\frac{\alpha}{8\ln(n+1)+2} \leq z \leq \alpha.$$

Moreover, there is an efficient algorithm that given the optimal edge weights d finds a cut whose ratio is $\leq (8\ln(n+1)+2)z = O(z\log n)$.

Proof: The goal of the proof is to construct a cut with small ratio, by using d and z (which are assumed to be known). We use the region-growing procedure (as in the above lemmas) and choose the parameter ϵ such that one of regions $A_1^*,\ldots,A_h^*$ either has cardinality $\geq n/2$, or gives a cut with small ratio. If one of the regions A_i^* has cardinality $\geq n/2$, then we "erase" the other regions, and keep growing A_i^* node by node till we find a cut with small ratio.

Let $\hat{\alpha} = (2+8\ln(n+1))z$. We apply the region-growing procedure iteratively, with the parameter ϵ fixed at $\hat{\alpha}n^2/(8z)$.

Let the regions be $A_1^*,\ldots,A_h^*$. If one of the regions $A_i^* = A$ gives a cut with ratio $u(\delta(A))/|A|\cdot|V\backslash A| \leq \hat{\alpha}$, then we stop and output this cut. Clearly, such a cut has ratio at most $(2+8\ln(n+1)) = O(\log n)$ times the ratio of a sparsest cut. If we do not find such a cut, then we claim that one of the regions $A_1^*,\ldots,A_h^*$ has cardinality $\geq n/2$. The claim is proved by contradiction. By assumption, each of the regions A_i^* has $u(\delta(A_i^*)) > \hat{\alpha}|A_i^*|\cdot|V\backslash A_i^*| \geq (\hat{\alpha}n/2)|A_i^*|$. Hence,

$$\sum_{i=1}^{h} u(\delta(A_i^*)) > (\hat{\alpha}n/2)\sum_{i=1}^{h}|A_i^*| = \hat{\alpha}n^2/2.$$

On the other hand, by Lemma 11.3 part (ii), and by our choice of ϵ,

$$\sum_{i=1}^{h} u(\delta(A_i^*)) \leq 4\epsilon z = \hat{\alpha}n^2/2.$$

This contradiction proves our claim: one of the regions has cardinality $\geq n/2$.

Let us denote this region by A^*, so $|A^*| \geq n/2$. We continue to grow A^* further, attempting to include all nodes in this set. All y-variables corresponding to sets S, $S \not\subseteq A^*$, are reset to zero.

In this last stage, we use a modified version of the region-growing procedure that does not check for the expansion condition (i.e., the condition in inequality (11.1)), but continues until either the currently active set A gives a cut with small ratio (i.e., $u(\delta(A))/|A| \cdot |V \backslash A| \leq \hat{\alpha}$) or all nodes are included in A. We claim that the algorithm will find a cut with small ratio. Again, the proof is by contradiction.

Let r be the root of the region A^*. Let the sets with positive y-variables be $\{r\} = S_1 \subset \ldots \subset S_\ell = A^* \subset \ldots \subset V$. For every node v, recall that $d(r,v) = \sum_{S \subseteq S_i} y_S$, where $v \in S_{i+1} \backslash S_i$. Focus on

$$\sum_{v \in V} d(r,v) = \sum_S y_S(n - |S|) = \sum_{S \subseteq A^*} y_S(n-|S|) + \sum_{S \supset A^*} y_S(n-|S|).$$

The first term satisfies

$$\sum_{S \subseteq A^*} y_S(n-|S|) \leq n \sum_{S \subseteq A^*} y_S = n \cdot rad(A^*) \leq n \ln(n+1)/\epsilon \leq 8 \ln(n+1) z/(\hat{\alpha} n).$$

Consider the second term. By our assumption, each superset S of A^* satisfies $u(\delta(S))/|S| \cdot |V \backslash S| > \hat{\alpha}$, and since $|S| > n/2$, we have $|V \backslash S| < 2u(\delta(S))/(\hat{\alpha} n)$. Hence,

$$\sum_{S \supset A^*} y_S(n-|S|) < (2/\hat{\alpha} n) \sum_{S \supset A^*} y_S u(\delta(S)) \leq (2/\hat{\alpha} n) z,$$

where the last inequality follows from the packing constraints on the y-variables. Consequently, $\sum_{v \in V} d(r,v) < (2z/\hat{\alpha} n)(4 \ln(n+1) + 1)$.

Finally, we apply the inequality

$$\sum_{p,q \in V} d(p,q) \leq (n-1) \sum_{v \in V} d(r,v),$$

where the summation on the left is over *unordered pairs* p, q of nodes; note that there are $\binom{n}{2}$ such pairs. This inequality easily follows from the triangle inequality for triples p, q, r, namely, $d(p,q) \leq d(p,r) + d(r,q)$.

Going back to our proof, we see that

$$\sum_{p,q \in V} d(p,q) \leq n \sum_{v \in V} d(r,v) < (2z/\hat{\alpha})(4 \ln(n+1) + 1) \leq 1,$$

where the last inequality follows from our choice of $\hat{\alpha}$. This is a contradiction, since the edge weights d form a feasible solution to the LP and so satisfy the LP constraint, $\sum_{p,q \in V} d(p,q) \geq 1$. Our claim follows: one of the sets found by the last stage of the region-growing procedure gives a cut of ratio $\leq \hat{\alpha}$. ∎

11.3 Metrics, ℓ_1 embeddings and a logarithmic guarantee for the generalized sparsest cut problem

Our goal in this section is to give another proof of the Leighton-Rao theorem. In fact, the result here is more general since it applies to the version of the sparsest cut problem where an arbitrary nonnegative demand $dem_{v,w}$ is given for each (unordered) node pair v, w; note that the sparsest cut problem is the special case where $dem_{v,w} = 1, \forall v, w \in V$. Let k be the number of node pairs that have positive demands. Following Aumann & Rabani, and Linial, London & Rabinovich, we will prove that there exists a cut whose ratio is at most $O(\log k)$ times the optimal value of an LP relaxation (F4). The proof is an easy consequence of a theorem of Bourgain on "near isometric embeddings" of metric spaces into ℓ_1.

We start with some definitions and preliminary results. Most readers may prefer to skip the next subsection, and to refer back to the definitions when needed.

11.3.1 Definitions and preliminaries

Let V be a finite set of points, and let n denote $|V|$. A *metric* d on V associates a nonnegative real number $d_{v,w}$ with each pair $v, w \in V$ and satisfies

$$d_{v,w} \leq d_{v,r} + d_{r,w}, \qquad \text{for all ordered triples } v, w, r \in V \times V \times V.$$

We also use $d(v, w)$ to denote $d_{v,w}$. The metric d may be viewed as a vector in $\Re_+^{\binom{n}{2}}$, i.e., d is an $\binom{n}{2}$-dimensional vector with one entry per pair $v, w \in V$. We allow $d_{v,w} = 0$ for distinct points $v, w \in V$, hence, d is a semimetric rather than a metric. A familiar example is the shortest paths metric of a graph, i.e., $d_{v,w}$ is taken to be the minimum number of edges in a v-w path of G.

A *norm* associates a real number $\|v\|$ with every point $v \in \Re^h$ (the h-dimensional real space) and satisfies

- $\|v\| \geq 0$, with equality only if $v = 0$,
- $\|rv\| = |r| \cdot \|v\|$, for every real number r, and
- $\|v + w\| \leq \|v\| + \|w\|$, for every $v, w \in \Re^h$.

A norm $\|\cdot\|$ has an associated metric, namely,

$$d_{v,w} = \|v - w\|, \forall v, w \in \Re^h.$$

For a point (or vector) $v \in \Re^h$ we denote the coordinates (or entries) of v by $v_1, \ldots, v_h$. For our purposes, three familiar norms (and their associated metrics) are of interest:

- the ℓ_1 norm, $\|v\|_1 = \sum_{i=1}^{h} |v_i|$, and the associated *Manhattan* metric,
- the ℓ_2 norm, $\|v\|_2 = \sqrt{\sum_{i=1}^{h} (v_i)^2}$, and the associated *Euclidean* metric, and
- the ℓ_∞ norm, $\|v\|_\infty = \max_{i=1}^{h} |v_i|$, and the associated ℓ-infinity metric.

We use ℓ_1^h, ℓ_2^h, or ℓ_∞^h to denote $\Re^h$ equipped with the ℓ_1 norm, the ℓ_2 norm or the ℓ_∞ norm, respectively.

An *isometric embedding* (or isometry) is a mapping ϕ from a metric space (V, d) to a metric space (X, d') that preserves all "distances", i.e., $\phi : V \to X$ is an isometry if

$$d'(\phi(v), \phi(w)) = d(v, w), \quad \forall v, w \in V.$$

Also, V is said to isometrically embed into X.

Proposition 11.5 *Every metric space (V, d) has an isometric embedding into ℓ_∞^n, i.e., into $\Re^n$ equipped with the ℓ_∞ norm, where n denotes $|V|$.*

Proof: Let $V = \{v^{(1)}, \ldots, v^{(n)}\}$. We map each $v^{(i)}$ to the vector $\phi(v^{(i)}) \in \Re^n$, where

$$\phi(v^{(i)}) = [d(v^{(1)}, v^{(i)}), d(v^{(2)}, v^{(i)}), \ldots, d(v^{(n)}, v^{(i)})].$$

In other words, $v^{(i)}$ is mapped to the ith column of the $n \times n$ "distance" matrix $[d_{v,w} \mid v, w \in V \times V]$. Then observe that

$$\|\phi(v^{(i)}) - \phi(v^{(j)})\|_\infty = \max_{k=1}^{n} |d(v^{(k)}, v^{(i)}) - d(v^{(k)}, v^{(j)})| \geq d(v^{(i)}, v^{(j)}).$$

On the other hand, the triangle inequality gives

$$|d(v^{(k)}, v^{(i)}) - d(v^{(k)}, v^{(j)})| \leq d(v^{(i)}, v^{(j)}).$$

The proof is done since $\|\phi(v^{(i)}) - \phi(v^{(j)})\|_\infty = d(v^{(i)}, v^{(j)})$. ∎

Proposition 11.6 *Let (V, d_1) be the metric space ℓ_1^h for some integer $h > 0$, i.e., d_1 is the metric induced by ℓ_1 for the set of points V in $\Re^h$. Then d_1 is the sum of h metrics $d_{1,1}, \ldots, d_{1,h}$, where $d_{1,i}$ is a metric induced by ℓ_1 in one dimension.*

Proof: For each i, $i = 1, \ldots, h$, define $d_{1,i} = |v_i - w_i|$, i.e., $d_{1,i}$ gives the "distances" in the ith coordinate. The result is obvious, since for each pair $v, w \in V$,

$$d_1(v, w) = \|v - w\|_1 = \sum_{i=1}^{h} |v_i - w_i| = d_{1,1}(v, w) + d_{1,2}(v, w) + \ldots + d_{1,h}(v, w).$$

∎

11.3.2 ℓ_1 embeddings and generalized sparsest cuts

Recall that the sparsest cut problem on a graph $G = (V, E)$ with edge capacities $u : E \to \Re_+$ is to find a cut $\delta(S)$ whose ratio $u(\delta(S))/|S| \cdot |V \backslash S|$ is minimum. We will also use u to denote the $\binom{n}{2}$-dimensional vector indexed by node pairs $v, w \in V$ such that $u_{v,w}$ equals either the capacity of the edge vw (if it exists) or zero (if there is no edge vw). Let $\mathbf{1}$ denote the $\binom{n}{2}$-dimensional vector of all ones.

Based on Proposition 11.5, we may view the LP formulation (F3) of the sparsest cut problem as follows: the optimal solution finds an embedding $\phi : V \to \ell_\infty^n$ (i.e., the nodes are embedded into $\Re^n$ equipped with the ℓ_∞ norm) such that (i) the sum of the $\binom{n}{2}$ "distances" in ℓ_∞^n is at least one (i.e., $\sum_{v,w \in V} \|\phi(v) - \phi(w)\|_\infty \geq 1$), and (ii) the sum of the capacities of the edges weighted by the edge "lengths" is minimum. What (if any) is the relation of such an embedding to the sparsest cut in the graph? There seems to be no direct relation, but there is a relation via an embedding into ℓ_1^n. To see this, suppose that we can somehow find an embedding $\phi' : V \to \ell_1^n$ such that (i) and (ii) hold w.r.t. "distances" and edge "lengths" in ℓ_1^n, i.e., such that $\sum_{v,w \in V} \|\phi'(v) - \phi'(w)\|_1 \geq 1$ and $z' = \sum_{e=vw \in E} u_{vw} \cdot \|\phi'(v) - \phi'(w)\|_1$ is minimum. There is something remarkable about such an ℓ_1 embedding because it turns out that z' equals the ratio of a sparsest cut. The proof is given below, in a few easy steps. (Since computing the ratio of a sparsest cut is NP-hard, there is no hope of computing ϕ' efficiently, modulo P $\neq$ NP. Nevertheless, it will be useful to study ϕ'.)

For a subset S of a point set V, the *cut metric* d_S is defined to be the metric on V given by

$$d_S(v, w) = \begin{cases} 1 & \text{if either } v \in S, w \notin S \text{ or } v \notin S, w \in S \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the cut metric is the 0-1 incidence vector of the point pairs separated by the cut. Observe two things. For a cut $\delta(S)$ of the graph G, the capacity $u(\delta(S))$ may be written as the inner product of two $\binom{n}{2}$-dimensional vectors, $u \cdot d_S$, and secondly, the ratio of the cut may be written as $u \cdot d_S / (\mathbf{1} \cdot d_S)$.

Proposition 11.7 *Let (V, d_1) be a metric space such that d_1 is induced by ℓ_1, i.e., $d_1(v, w) = \|v - w\|_1, \forall v, w \in V$. Then d_1 is in the cone generated by the cut metrics $\{d_S \mid S \subset V\}$, i.e., d_1 can be written as a linear combination of cut metrics*

$$d_1 = \sum_j \{a_j d_{S_j} \mid S_j \subset V\}$$

such that each coefficient a_j is nonnegative. Moreover, the number of cut metrics required is at most $h \cdot |V|$, where h is the dimension of (V, d_1).

Proof: Since d_1 is the sum of h "one-dimensional ℓ_1 metrics" (by Proposition 11.6), it will suffice to prove that a "one-dimensional ℓ_1 metric" d_1 can be written as a conical combination (i.e., a linear combination with nonnegative coefficients) of at most $|V|$ cut metrics. Observe that (V, d_1) is an embedding of V into the real line $\Re^1$, because d_1 is one-dimensional. Let the points in V occur in the order $v^{(1)}, \ldots, v^{(i)}, \ldots, v^{(n)}$ in the real line. Define n cut metrics d_{S_j} by taking $S_1 = \{v^{(1)}\}, \ldots, S_i = \{v^{(1)}, \ldots, v^{(i)}\}, S_n = V$. Clearly, $d_1 = \sum_{j=1}^{n-1} a_j d_{S_j}$, where $a_j = d_1(v^{(j)}, v^{(j+1)})$. ∎

Our claim (z' equals the ratio of a sparsest cut) follows from Proposition 11.7 and a simple fact about ratios.

Fact 11.8 *Let $x_1, \ldots, x_k$ be nonnegative numbers and let $y_1, \ldots, y_k$ be positive numbers. Then*

$$\min_{i=1}^{k} \frac{x_i}{y_i} \leq \frac{x_1 + x_2 + \ldots + x_k}{y_1 + y_2 + \ldots + y_k} \leq \max_{i=1}^{k} \frac{x_i}{y_i}.$$

In the next proposition, rather than using the linear objective function of the LP (F3) we use the rational objective function of (F1).

Proposition 11.9 *Let $\delta(S^*)$ be a sparsest cut of the (capacitated) graph $G = (V, E), u$, and let α^* be the ratio of this cut. Let h be a positive integer. Then*

$$\alpha^* = z' = \min_{\phi: V \to \Re^h} \frac{\sum_{e=vw \in E} u_{vw} \cdot \|\phi(v) - \phi(w)\|_1}{\sum_{v,w \in V} \|\phi(v) - \phi(w)\|_1}.$$

Proof: Let $(\phi(V), d_1)$ be the metric space that minimizes z'. First, note that $z' \le \alpha^*$ since we can take d_1 to be the cut metric d_{S^*} of the sparsest cut (i.e., we take $\phi(v) = 1$ if $v \in S^*$ and $\phi(v) = 0$ otherwise). To see that $z' \ge \alpha^*$, use Proposition 11.7 to write d_1 as $\sum_{j=1}^{p} a_j d_{S_j}$, where each S_j corresponds to a cut $\delta(S_j)$, and $p \le hn$. Then

$$z' = \frac{\sum_{j=1}^{p} u \cdot (a_j d_{S_j})}{\sum_{j=1}^{p} \mathbf{1} \cdot (a_j d_{S_j})} \ge \min_{j=1}^{p} \frac{u \cdot (a_j d_{S_j})}{\mathbf{1} \cdot (a_j d_{S_j})} = \min_{j=1}^{p} \frac{u(\delta(S_j))}{|S_j| \cdot |V \backslash S_j|} \ge \alpha *,$$

where the first inequality follows from the previous fact. The proof is done. ▌

The optimal solution of (F1) (or of the LP (F3)) is an ℓ_∞-induced metric minimizing our objective function, whereas a sparsest cut corresponds to an ℓ_1-induced metric minimizing our objective function. To efficiently find an approximately sparsest cut via the optimal solution to the LP, we can try to embed the optimal ℓ_∞-induced metric into ℓ_1 such that all pairwise "distances" are preserved as much as possible. This motivates the next definition. Note that both z (the optimal value of (F1)) and z' (in Proposition 11.9) remain unchanged if all the pairwise "distances" are scaled by the same quantity.

Let (V, d) be a metric space and let ϕ be an embedding of (V, d) into (V', d') such that $d'(\phi(v), \phi(w)) \le d(v, w), \forall v, w \in V$. The *distortion* of ϕ is defined to be

$$\max_{v,w \in V} \frac{d(v, w)}{d'(\phi(v), \phi(w))}.$$

Another proof of the Leighton-Rao theorem follows from an important theorem of Bourgain and Proposition 11.9. (A proof of Bourgain's theorem is given in the next section.)

Theorem 11.10 (Bourgain (1986)) *Let (V, d) be a metric space, and let n denote $|V|$. There exists an embedding ϕ from (V, d) into ℓ_1^h, where $h = O(\log^2 n)$, such that the distortion is $O(\log n)$. Moreover, ϕ can be computed in polynomial time by a randomized algorithm.*

Theorem 11.11 (Leighton & Rao (1988)) *Let d be the optimal solution to the LP (F3), and let z be the optimal value of the LP (F3). Let α be the ratio of a sparsest cut of G, u. Then*

$$\frac{\alpha}{O(\log n)} \le z \le \alpha.$$

Moreover, there is an efficient (deterministic) algorithm that finds a cut whose ratio is $O(z \log n)$.

Proof: After computing an optimal solution d for the LP (F3), we use Bourgain's theorem to compute an embedding ϕ of (V,d) into ℓ_1^h, where $h = O(\log^2 n)$, such that the distortion Δ is $O(\log n)$. Let d_1 denote the ℓ_1-induced metric. Then,

$$z' = \frac{u \cdot d_1}{\mathbf{1} \cdot d_1} \leq \Delta \cdot \frac{u \cdot d}{\mathbf{1} \cdot d} = \Delta z,$$

where the inequality follows because $d(v,w)/\Delta \leq d_1(v,w) \leq d(v,w), \forall v,w \in V$. By Proposition 11.9, we can find a cut $\delta(S)$ whose ratio is $\leq z'$ by examining the ratios of at most $h \cdot |V|$ cuts. This completes the proof. A randomized algorithm is easily obtained because the construction in the proof of Bourgain's theorem is randomized. Linial, London & Rabinovich (1995) (and independently Garg (1995)) have given a method for derandomizing the Bourgain construction. ∎

In fact, a generalization of the Leighton-Rao theorem can be proved by the method of ℓ_1 embeddings. For each pair of nodes $v,w \in V$ of the (capacitated) graph G,u, let there be a nonnegative real-valued demand $dem_{v,w}$. Also, let dem denote the corresponding $\binom{n}{2}$-dimensional vector indexed by node pairs $v,w \in V$. Define the demand of a cut $\delta(S)$ to be

$$dem(S) = \sum_{v,w \in V} \{dem_{v,w} \mid v \in S, w \notin S \text{ or } v \notin S, w \in S\} = dem \cdot d_S.$$

Now, redefine the *ratio* of a cut $\delta(S)$ to mean

$$u(\delta(S))/dem(S) = u \cdot d_S/(dem \cdot d_S),$$

and define a *generalized sparsest cut* to be a cut whose ratio is minimum. By changing one constraint in the LP (F3), we get the following LP relaxation of the generalized sparsest cut problem.

$$\begin{array}{lllll} \text{(F4)} \quad z & = & \text{minimize} & \sum_{e \in E} u_e d_e & \\ \text{subject to} & & & \sum_{v,w \in V} dem_{v,w} d_{v,w} & \geq 1 \\ & & & d \text{ is a metric.} & \end{array}$$

Theorem 11.12 (Aumann & Rabani (1994), Linial, London & Rabinovich (1995))
Let $z = \sum_{e \in E} u_e d_e$ be the optimal value of the LP (F4). Let α be the ratio of a generalized sparsest cut of G, u, dem, and let k denote the number of pairs $v,w \in V$ with positive demands. Then

$$\frac{\alpha}{O(\log k)} \leq z \leq \alpha.$$

Moreover, there is an efficient algorithm that finds a cut whose ratio is $O(z \log k)$.

We leave the proof to the reader, since it is similar to the proof of Theorem 11.11 given above. To obtain an approximation guarantee of $O(\log k)$ rather than $O(\log n)$ we need a small generalization of Bourgain's theorem.

Proposition 11.13 *Let (V,d) be a metric space, and let T be a subset of V. There exists an embedding ϕ from (V,d) into ℓ_1^h, where $h = O(\log^2 |T|)$, such that for each $v,w \in V$, $\|\phi(v) - \phi(w)\|_1 \leq d(v,w)$, and for each $v,w \in T$, $d(v,w)/\Delta \leq \|\phi(v) - \phi(w)\|_1$, where $\Delta = O(\log |T|)$. Moreover, ϕ can be computed in polynomial time by a randomized algorithm.*

11.4 Bourgain's theorem

We give a proof of Bourgain's theorem, omitting some computational details. The reader may either fill in the details (by solving Exercise 8), or refer to Linial, London & Rabinovich (Combinatorica 1995) or to the survey paper by Shmoys (1996).

Theorem 11.10 *Let (V, d) be a metric space, and let n denote $|V|$. There exists an embedding ϕ from (V, d) into ℓ_1^h, where $h = O(\log^2 n)$, such that the distortion is $O(\log n)$. Moreover, ϕ can be computed in polynomial time by a randomized algorithm.*

Proof: The mapping $\phi : V \to \Re^h$ is constructed using randomization. For a point $v \in V$ and a subset $S \subseteq V$, let $d(v, S) = \min_{w \in S} d(v, w)$. Let $p = \lfloor \log_2 n \rfloor$, so that $2^p \leq n < 2^{p+1}$, and let $q = 10p$; in fact, we will have $h = pq$. For each $j = 0, 1, 2, \ldots, p-1$, we randomly and independently choose q subsets of V, each of cardinality $n/2^j$, $A_{j,1}, A_{j,2}, \ldots, A_{j,q}$, i.e., each of the $\binom{n}{n/2^j}$ subsets of V of cardinality $n/2^j$ is equally likely to be chosen as a set $A_{j,k}$, and the choices are mutually independent. Next, each point $v \in V$ is mapped to a vector

$$\begin{aligned}\phi'(v) &= [d(v, A_{0,1}), d(v, A_{0,2}), \ldots, d(v, A_{0,q}), d(v, A_{1,1}), d(v, A_{1,2}), \ldots, d(v, A_{1,q}),\\ &\quad \ldots, d(v, A_{p-1,1}), d(v, A_{p-1,2}), \ldots, d(v, A_{p-1,q})],\end{aligned}$$

i.e., v is mapped to a vector of dimension pq whose ith coordinate is the "d-distance" between v and the ith random set in the list $A_{0,1}, \ldots, A_{0,q}, \ldots, A_{p-1,1}, \ldots, A_{p-1,q}$. Let the ith coordinate of the vector $\phi'(v)$ be denoted by $\phi_i'(v)$. We obtain the final mapping ϕ by scaling each vector $\phi'(v)$ by $1/(pq)$, i.e., $\phi(v) = \phi'(v)/(pq), \forall v \in V$.

To see that $\|\phi(v) - \phi(w)\|_1 \leq d(v, w)$, focus on an arbitrary coordinate $i = jq+k$ ($0 \leq j \leq p-1$, $1 \leq k \leq q$) and let A be the corresponding random set $A_{j,k}$. We have $|\phi_i'(v) - \phi_i'(w)| = |d(v, A) - d(w, A)| \leq d(v, w)$, because w.l.o.g. there exist $a \in A$ and $b \in A$ such that $|d(v, A) - d(w, A)| = d(v, a) - d(w, b) \leq d(v, b) - d(w, b) \leq d(v, w)$, where the last step follows by the triangle inequality on d. Hence, $\|\phi(v) - \phi(w)\|_1 = \frac{1}{(pq)} \sum_{i=1}^{pq} |\phi_i'(v) - \phi_i'(w)| \leq d(v, w)$.

The ingenious part of the proof is to show that for all pairs $v, w \in V$, the other inequality ($\|\phi(v) - \phi(w)\|_1 \geq d(v, w)/\Delta$) holds with high probability. Consider an arbitrary pair $v, w \in V$. Here is an informal argument that skips a few details; these details are handled either in the next paragraph or in the exercises. We will partition the "line segment" joining v and w into $O(\log n)$ segments such that the sum of the "d-lengths" of the first p segments is at least $d(v, w)/3$. See Figure 11.2. Let ρ_j be the "d-length" of the jth segment. Consider an arbitrary coordinate $i = jq + k$ ($0 \leq j \leq p-1, 1 \leq k \leq q$) of $\phi(v)$ and $\phi(w)$. We need a key claim:

there exists a constant $c < 1$ such that with probability $\geq c$, $|\phi_i'(v) - \phi_i'(w)| \geq \rho_j$.

If we fix j and sum over all coordinates $i = jq + k$, for $k = 1, 2, \ldots, q$, then by applying Chernoff bounds it can be seen that with high probability, $\sum_{k=1}^{q} |\phi_{jq+k}'(v) - \phi_{jq+k}'(w)| \geq qc\rho_j/2$ (see Exercise 8 for details). Finally, summing over all coordinates $i = 1, 2, \ldots, (pq)$, we see that with high

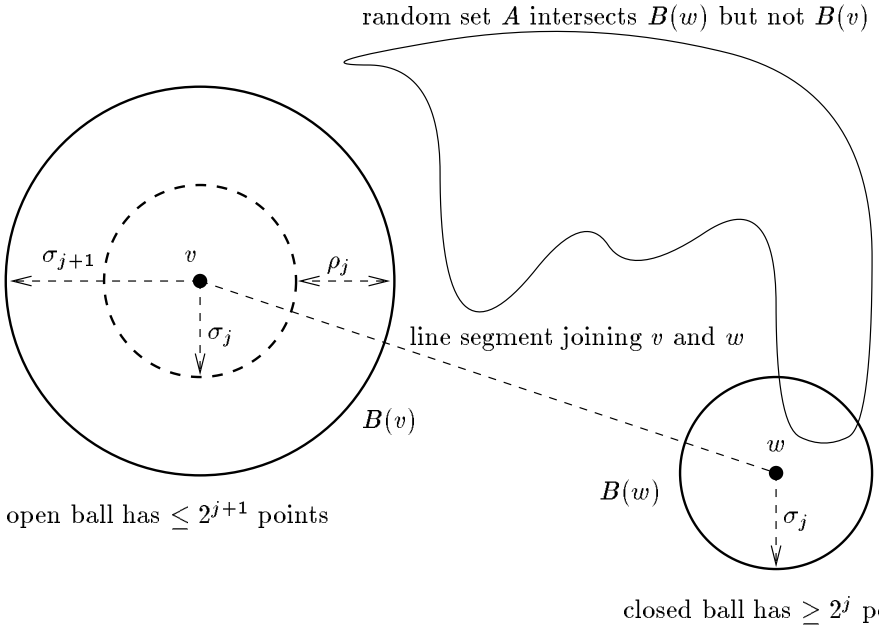


Figure 11.2: An illustration of the proof of Bourgain's theorem.

probability,

$$\sum_{i=1}^{pq} |\phi_i'(v) - \phi_i'(w)| = \sum_{j=0}^{p-1} \left(\sum_{k=1}^{q} |\phi_{jq+k}'(v) - \phi_{jq+k}'(w)| \right) \geq \frac{qc}{2} \sum_{j=0}^{p-1} \rho_j \geq \frac{qc}{2} \frac{d(v,w)}{3}.$$

This proves the theorem, since $\sum_{i=1}^{pq} |\phi_i(v) - \phi_i(w)| \geq \dfrac{c}{6p} d(v,w)$, and so the distortion is $\Delta = 6p/c = O(\log n)$.

The claim above needs to be proved in detail. For $j = 0, 1, \ldots, p-1$, let σ_j be the minimum of $d(v,w)/3$ and the minimum real number σ such that both $|\{a \in V \mid d(v,a) \leq \sigma\}| \geq 2^j$ and $|\{b \in V \mid d(w,b) \leq \sigma\}| \geq 2^j$, i.e., we compute the minimum radius σ such that each of the balls of d-radius σ centered at v and w, respectively, contains at least 2^j points of V, and further, if $\sigma \leq d(v,w)/3$, then we take $\sigma_j = \sigma$, otherwise we take $\sigma_j = d(v,w)/3$. Note that $\sigma_0 = 0$. See Figure 11.2. Consider j $(0 \leq j \leq p-2)$ such that $\sigma_j < d(v,w)/3$. (The other case $\sigma_j = d(v,w)/3$ is trivial.) W.l.o.g. the set of points of V in the open ball of d-radius σ_{j+1} centered at v, $B^o(v, \sigma_{j+1})$, has cardinality $\leq 2^{j+1}$. Moreover, the set of points of V in the closed ball of d-radius σ_j centered at w, $B(w, \sigma_j)$, has cardinality $\geq 2^j$. Now, the random set $A = A_{j,k}$ (for some k, $1 \leq k \leq q$) has $n/2^j$ points of V. It is an exercise in elementary probability to show that there exists a constant $c < 1$ such that with probability $\geq c$, $A \cap B^o(v, \sigma_{j+1})$ is empty and $A \cap B(w, \sigma_j)$ is nonempty. Hence, with probability $\geq c$, $d(v,A) \geq \sigma_{j+1}$ and $d(w,A) \leq \sigma_j$. Consequently, with probability $\geq c$,

$$|\phi_{jq+k}'(v) - \phi_{jq+k}'(w)| = |d(v, A_{j,k}) - d(w, A_{j,k})| \geq \sigma_{j+1} - \sigma_j.$$

Note that this analysis applies also to the largest j such that $\sigma_j < d(v,w)/3$. Also, note that in terms of the previous paragraph, the jth segment (in the "line segment" joining v and w) has "d-length" $\rho_j = \sigma_{j+1} - \sigma_j$. ∎

11.5 From sparsest cuts to balanced separators

In this section, we sketch how the result on approximating the sparsest cut can be applied to find balanced separators in graphs. An α-balanced separator (for $\alpha \geq \frac{1}{2}$) is defined as a set of edges whose removal breaks up the graph into two components each of size at least $\alpha \cdot n$. A $\frac{1}{2}$-balanced separator is also sometimes called a bisector. By a balanced separator, we mean any α-balanced separator for some fixed constant α. This is because in many applications, what is important is that separator is balanced rather than the exact quality of the balance.

Define the *flux* of a cut $\delta(S)$ of G to mean the quantity $\dfrac{u(\delta(S))}{\min(|S|,|V\backslash S|)}$[1]. A *minimum flux cut* is one whose flux is minimum.

Flux and sparsity are very closely related quantities. Suppose that the sparsest cut has ratio s and the minimum flux cut has flux f, then

$$\frac{n}{2}s \leq f \leq ns$$

since the bigger half of any cut has size between $\frac{n}{2}$ and n. Thus the approximation algorithm from the previous section also gives an $O(\log n)$ approximation for finding a minimum flux cut in a given graph. We can use this result to derive an approximation result of the following type.

Theorem 11.14 *Suppose that we are given an ρ-approximation algorithm for finding a minimum flux cut in a graph, and suppose that B denotes the minimum capacity of any bisector (i.e., $\frac{1}{2}$-balanced separator) in the graph. Then we can find a $\frac{2}{3}$-balanced separator of the graph of capacity at most $8\rho \cdot B$.*

Proof: We prove the theorem by using a simple greedy algorithm to construct the $\frac{2}{3}$-balanced cut. The algorithm is simple. We iterate finding the minimum flux cut in a graph and deleting the vertices in the smaller side of the cut, until the number of vertices that remain is at most $\frac{2n}{3}$. The set of all the deleted vertices form one shore of the cut while the remaining vertices form the other.

Let us denote by u_i and x_i, the capacity and the size of the smaller side of the flux cut found in iteration number i. Say the greedy procedure ran for t iterations; we wish to bound $\sum_{i=1}^{t} u_i$ in terms of B.

The key step is to observe that while we are still running the greedy procedure, the minimum bisector induced in the remaining graph gives a cut of small flux. In particular, since we have deleted at most $\frac{n}{3}$ nodes, in the worst case, all these nodes may be deleted from the same side of the minimum bisector, and this cut in the remaining graph has flux at most $\frac{B}{\frac{n}{6}}$. This is because

[1] The flux has also been called the edge expansion or the quotient cost of a cut.

the smaller side of this cut has at least $\frac{n}{2} - \frac{n}{3}$ nodes in it. Using a ρ-approximation algorithm at every iteration guarantees that

$$\frac{u_i}{x_i} \le \rho \cdot \frac{6B}{n} \tag{11.3}$$

Next we write down a recurrence for the number of nodes we "need" to delete to achieve our goal. Namely, let n_i represent the number of nodes before iteration i that we need to delete so that the remaining graph has at most $\frac{2n}{3}$ nodes. Thus, e.g., $n_1 = \frac{n}{3}$, and $n_t \ge 1$. We get the following simple recurrence.

$$n_{i+1} = n_i - x_i$$

By equation 11.3, we have $x_i \ge \frac{nu_i}{6\rho B} \ge \frac{n_i u_i}{2\rho B}$. Substituting and simplifying, we get

$$n_{i+1} \le n_i(1 - \frac{u_i}{2\rho B}) \le n_i e^{-\frac{u_i}{2\rho B}}$$

Expanding out the recurrence, we get

$$n_t \le n_1 e^{-\sum_{i=1}^{t-1} \frac{u_i}{2\rho B}}$$

Taking natural logarithm and simplifying, we finally get

$$\sum_{i=1}^{t-1} u_i \le 2\rho B \log \frac{n_i}{n_t} \le 2\rho B \log n \tag{11.4}$$

To bound the capacity of the last cut u_t, we use equation 11.3 again to get

$$u_t \le \rho \cdot x_t \frac{6\rho B}{n} \le 3\rho B$$

since $x_t \le \frac{n}{2}$ (we always choose the smaller side of a cut). The two equations above imply that the cost of the $\frac{2}{3}$-balanced separator found is $O(B \log n)$ where B is the capacity of a minimum bisector. ∎

11.6 Applications of separators

Finding balanced separators has several applications in the design of good approximation algorithms for other problems whose objective can be realted to the value of a bisector – these approaches typically employ divide-and-conquer using the separator approximation in the divide part of the procedure. This approach was pioneered in the work of Bhatt and Leighton [4] and applied to a variety of problems by Leighton and Rao [9]. Examples include finding minimum area layout of a graph, finding layouts that minimize the total number of edge crossings, and finding a minimum feedback arc set (arcs whose removal leaves the remaining digraph acyclic). Other more involved applications are in graph completion problems to find the minimum-size supergraph of a given graph that has certain nice properties, e.g., is an interval graph[12] or a chordal graph[2]; The latter problem models finding good orderings for sparse Gaussian elimination.

Two overall features are useful in identifying problems that may be amenable to the divide-and-conquer approximation approach using separators. Firstly, the objective of the problem must be related to the size of the separator, and secondly, an optimal solution for the overall problem should be "decomposable" into solutions to the subproblems that are obtained in the dividing stage. We illustrate these two features by looking at two simple linear arrangement problems.

Linear arrangements

A *linear arrangement* of an undirected graph $G = (V, E)$ (where $|V| = n$) is a linear ordering of its nodes, or more formally, a bijection A of the vertices to the set $\{1, 2, \dots, n\}$. Pictorially, we can think of a linear arrangement as a drawing of the nodes of a graph in a line in the order specified by the bijection A.

A linear arrangement defines $n - 1$ sequential cuts formed by the sets $\{A^{-1}(1), A^{-1}(2), \dots, A^{-1}(i)\} = S_i$ for each i. Furthermore, every edge uv with $A(v) > A(u)$ say, is "stretched" in the arrangement to an extent $stretch(uv) = A(v) - A(u)$. Four problems arise by minimizing the maximum or the sum of each of the two quantities: the stretch of edges and the size of each of the sequential cuts.

Finding a linear arrangement to minimize the maximum size of any of the sequential cuts corresponds to solving the *minimum cut linear arrangement (MCLA) problem.* Minimizing the maximum stretch of any edge in the arrangement corresponds to finding a *minimum bandwidth ordering.* A moment of though reveals that the two objectives involving minimizing the sum of the stretches and the cuts are really the same (by switching the summation between the edges and the sequential cuts). This ordering problem is to find an *optimal linear arrangement (OLA).* We will use our separator-based method for the minimum cut and optimal linear arrangement problems. No nontrivial approximation algorithm is known for the bandwidth minimization problem on general graphs.

11.6.1 Minimum cut linear arrangement

To discover the relation of the size of a balanced separator to the value of a cut in a linear arrangement, consider an optimal MCLA. Let OPT denote the maximum size of a cut in this arrangement. Since the cut corresponding to the set $S_{\frac{n}{2}}$ is also a candidate for computing this maximum value, and this cut is a bisector, we have that $OPT \geq B$ where B is the size of a minimum bisector. On the other hand, we have a separator approximation that finds a balanced separator of size $O(\log n \cdot B)$.

This motivates the following simple divide and conquer algorithm: use the approximation algorithm for finding an $\frac{2}{3}$-balanced separator to break the graph into two parts, say L and R. Recursively compute an arrangement of the nodes of L and the nodes of R and concatenate them to form the final arrangement. The basis is when the graph has only one node in which case the ordering is trivial. Alternatively, the divide steps of this algorithm can be represented by a "separator tree" whose internal nodes represent a divide step that finds a balanced separator, and the leaves are the original nodes of G. The arrangement output is the preorder traversal of these nodes, or simply the left to right ordering of the leaves.

It is not hard to bound the performance ratio of this algorithm. First we notice that the depth of recursion, or alternatively, the depth of the separator tree, is $O(\log n)$ since we use a banaced separator in each divide step. The maximum value of any of the cuts in the output arrangement is the maximum over the sum of the values of $O(\log n)$ different separators found by following some recursive step in the algorithm. We already argued that the top-level separator has size $O(\log n \cdot B) = O(\log n \cdot OPT)$. If we can argue that each of the separators found in any recursive step is also of size $O(\log n \cdot OPT)$, a performance ratio of $O(\log^2 n)$ would follow.

The decomposition property is useful in showing that for any recursive subproblem, the size of the separator found is $O(\log n \cdot OPT)$. Consider a fixed depth of recursion corresponding to a particular level of the separator tree. Let the subgraphs corresponding to the different portions of the original graph on which we are finding separators be $G_1, \ldots G_k$. Note that the nodes of these subgraphs partition those of G. Consider the ordering induced on the nodes of G_i by the MCLA. The crucial observation is that the maximum cut in this induced arrangement has at most OPT edges. Moreover by looking at the central cut in this arrangement that splits G_i into two equal pieces, we see that the minimum size of a bisector of G_i is at most the size of the central cut. This in turn is at most OPT. The separator we find for G_i has size at most a logarithmic factor larger than the size of a minimum bisector for G_i, and hence at most $O(\log n \cdot OPT)$. This completes the proof that the divide-and-conquer approach gives an $O(\log^2 n)$ approximation algorithm.

11.6.2 Optimal linear arrangement

Before we present an algorithm for this problem we first state a simple extension of the previous result on finding balanced separators. Earlier, we argued that we can find a $\frac{2}{3}$-balanced separator by applying a greedy algorithm and using a minimum bisector to bound the quality of the solution. We can instead attempt to find say a $\frac{3}{4}$-balanced separator by using a minimum $\frac{2}{3}$-balanced separator in the analysis. This gives the following result.

Theorem 11.15 *Suppose that we are given an ρ-approximation algorithm for finding a minimum flux cut in a graph, and suppose that $B_{\frac{2}{3}}$ denotes the minimum capacity of any $\frac{2}{3}$-balanced separator in the graph. Then we can find a $\frac{3}{4}$-balanced separator of the graph of capacity at most $O(\rho \cdot B_{\frac{2}{3}})$.*

We can proceed as before by first trying to relate the value OPT of the optimal linear arrangement to the size of a separator. To do this, note that OPT is a sum of $n-1$ terms corresponding to the sizes of the sequential cuts in the linear order. If we consider the middle third of these cuts ($\delta(S_i)$ for i from $\frac{n}{3}$ to $\frac{2n}{3}$), each of these cuts define a $\frac{2}{3}$-balanced cut of the graph and must have size at that of a minimum such separator, say $B_{\frac{2}{3}}$. Thus we have

$$OPT \geq \frac{n \cdot B_{\frac{2}{3}}}{3}.$$

As before, we use a simple divide and conquer algorithm: apply the approximation algorithm for finding an $\frac{3}{4}$-balanced separator in the above theorem to break the graph into two parts, say L and R. Recursively compute an arrangement of the nodes of L and the nodes of R and concatenate them to form the final arrangement. Again, the ordering can be represented by the left to right ordering of the leaves of the corresponding separator tree.

We use a slightly different argument to show a performance ratio of $O(\log^2 n)$ in this case. At the top level of recursion, we split the graph by using edges of a $\frac{3}{4}$-balanced separator of size at most $O(\log n \cdot B_{\frac{2}{3}})$. The contribution of these edges to the value of this arrangement is the sum of their stretches which is trivially at most $n-1$. Thus the overall contribution of this separator to the objective is $O(n \log n \cdot B_{\frac{2}{3}}) = O(\log n \cdot OPT)$.

To employ a similar strategy at every level of recursion, we must show that for all the different disjoint graphs formed at any depth in the recursion, the contributions of all the separator edges found in this level is $O(\log n \cdot OPT)$. Note that these subgraphs represent a partition of the nodes of G and there are at most $(\frac{4}{3})^i$ of them at depth i of recursion.

We use a decomposition property to prove the claim for a general level of the recursion: consider the linear arrangement induced on each of the subgraphs in this level by the optimal global arrangement. Since the stretch of any edge in an induced arrangement is at most that in the global arrangement, the sum of the values of the induced arrangements, say $OPT_1 + OPT_2 + \ldots + OPT_k$ is at most OPT. By our previous argument applied to each of the subgraphs, we see that $OPT_i \geq \frac{n_i \cdot B^i_{\frac{2}{3}}}{3}$ where n_i is the number of nodes in the subgraph and $B^i_{\frac{2}{3}}$ is the value of a $\frac{2}{3}$-balanced separator in that subgraph G_i. The number of edges in the separator found for this graph is $O(\log n_i \cdot B^i_{\frac{2}{3}})$ and each has a stretch of at most $n_i - 1$ trivially. Thus the contribution by these edges is $O(n_i \cdot \log n_i \cdot B^i_{\frac{2}{3}}) = O(\log n_i \cdot OPT_i) = O(\log n \cdot OPT_i)$. Thus the total contribution of all the edges in a level is $O(\log n \cdot (OPT_1 + OPT_2 + \ldots + OPT_k)) = O(\log n \cdot OPT)$ as required. This shows that the overall method has performance ratio $O(\log^2 n)$ since there are $O(\log n)$ levels of recursion.

For several of the problems described in [12], the log-squared approximation guarantees were slightly improved to $O(\log n \log\log n)$ by Even et al. [6]. More recently, Rao and Richa [11] further improve some of these results to $O(\log n)$-approximations, including the optimal linear arrangement problem described above.

11.7 Exercises

1. Given a graph $G = (V, E)$ with edge capacities $u : E \rightarrow \Re_+$, the *densest cut* problem is to find a cut $\delta(S)$ such that the ratio $u(\delta(S))/(|S| \cdot |V \backslash S|)$ is **maximum**.

   (a) Prove that a densest cut can be computed efficiently if and only if a sparsest cut can be computed efficiently.

   (b) Prove that the problem of finding the densest cut is NP-hard. Does this mean that the sparsest cut problem is NP-hard?

   (HINT: Give a reduction from the maxcut problem (find a cut of maximum capacity) on a graph G'; the maxcut problem is known to be NP-hard. Construct the prism G'' of G' by taking two copies of G' and adding an edge between the two copies of each G'-node.)

   (c) Give a 2-approximation algorithm for the densest cut problem, i.e., give an efficient algorithm for finding a cut whose ratio is at least half the ratio of a densest cut. Does this mean that there exists an $O(1)$-approximation algorithm for the sparsest cut problem?

   (HINT: Focus on the star cuts, $\delta(v), \forall v \in V$.)

2. Prove that the relaxation (F1) of the sparsest cut problem is equivalent to the LP relaxation (F2), in the sense that for every optimal solution d, λ of (F1) there exists a feasible solution of (F2) with the same objective value, and vice versa. Prove that the LP relaxations (F2) and (F3) are equivalent.

3. In Theorem 11.11, improve the integrality gap (and approximation guarantee) of $8\ln(n+1)+2$ by a factor of 2, by improving the analysis in the proof.

   (HINT: The *flux* of a cut $\delta(S)$ is defined to be $u(\delta(S))/\min(|S|, |V\backslash S|)$. Let β be the minimum flux of a cut. Note that $\alpha n/2 \leq \beta \leq \alpha n$, where α is the ratio of a sparsest cut. Modify the analysis at appropriate places by using the flux of a cut instead of the ratio of a cut.)

4. The *minimum multicut* problem is defined as follows. The input consists of a graph $G = (V, E)$ with edge capacities $u : E \to \Re_+$, together with a set of demand pairs (or demand edges) $s_1t_1, s_2t_2, \ldots, s_kt_k$ (each demand pair is an unordered pair of nodes; demand pairs have no relation to the edges in E, i.e., an edge s_it_i may or may not be present in E). The problem is to find an edge set $C \subseteq E$ such that the capacity $u(C) = \sum_{e\in C} u_e$ is minimum and such that $G\backslash C$ has no s_i-t_i path for each $i = 1, \ldots, k$; in other words, an optimal solution for the problem is a minimum-capacity edge set whose removal disconnects (or separates) all k demand pairs.

   (a) Write down an LP relaxation (FM) for the minimum multicut problem, based on the LP relaxation (F2) for the sparsest cut problem.

   (HINT: Replace the constraint $\sum_{v,w\in V} \lambda_{v,w} \geq 1$ by the k constraints $\lambda_{s_i,t_i} \geq 1$, $i = 1, \ldots, k$.)

   (b) Prove that the minimum capacity of a multicut is at most $O(\log k)$ times the optimal value of (FC). Give an efficient algorithm for finding a multicut whose capacity is at most $O(\log k)$ times the minimum capacity of a multicut.

   (HINT: Apply the region-growing procedure with the parameter ϵ chosen to guarantee that each region contains at most one node of a demand pair.)

   (c) (N. Kahale 1993) Use the algorithm in part (b) to design an $O((\log D)(\log k))$-approximation algorithm for the generalized sparsest cut problem (see Section 11.3), where D denotes the total demand $\sum_{v,w\in V} dem_{v,w}$. W.l.o.g. assume that each demand $dem_{v,w}$ is a (non-negative) integer.

   (HINT: First, find the optimal solution d of the LP relaxation (F4) of the generalized sparsest cut problem. Next, find a set $Q = \{s_i, t_i\}$ of demand pairs such that

$$d(Q) \geq \frac{1}{dem(Q) \cdot H(dem(Q))},$$

   where $d(Q) = \min\limits_{s_i,t_i\in Q} d_{s_i,t_i}$, $dem(Q) = \sum\limits_{s_i,t_i\in Q} dem_{s_i,t_i}$, and $H(i)$ is the ith harmonic number $1 + 1/2 + \ldots + 1/i$. To find Q, order the pairs $\{s_i, t_i\}$ with positive demands according to decreasing values of d_{s_i,t_i}, and renumber these pairs such that $d_{s_1,t_1} \geq d_{s_2,t_2} \geq \ldots \geq d_{s_i,t_i} \geq \ldots$, and let $D_i = \sum_{j=1}^{i} dem_{s_i,t_i}$. If for each i, $d_{s_i,t_i} < \dfrac{1}{D_i \cdot H(D_i)}$, then obtain a contradiction by showing that $\sum_i d_{s_i,t_i} dem_{s_i,t_i} < 1$. Having found Q, deduce that $d/d(Q)$ is a feasible solution to the LP relaxation (FM) of the minimum multicut problem specified by G, u and Q (i.e., the demand pairs for (FM) are precisely the pairs in Q). Finally, prove that the multicut found by the approximation algorithm in part (b) has ratio at most $O((\log D)(\log k))$ times the ratio of a generalized sparsest cut.)

5. Suppose that the graph $G = (V, E)$ is a tree, and let $u : E \rightarrow \Re_+$ assign capacities to the edges.
   (a) Can a sparsest cut be computed efficiently?

   (HINT: One way is to use the method of ℓ_1 embeddings: for any weight function on the edges $d : E \rightarrow \Re_+$, prove that the resulting shortest-paths metric space of G isometrically embeds into ℓ_1^n.)
   (b) Can a minimum multicut be computed efficiently? Note that the demand pairs may be arbitrary.

   (HINT: Attempt a reduction from the minimum node cover problem.)
6. The r-dimensional cube Q_r is defined recursively in terms of the cartesian product of Q_{r-1} and the complete graph on two nodes, K_2, as follows:

$$\begin{aligned} Q_1 &= K_2 \\ Q_r &= K_2 \times Q_{r-1}. \end{aligned}$$

   Alternatively, Q_r may be defined as a graph whose node set consists of 2^r r-dimensional boolean vectors, where two nodes are adjacent whenever they differ in exactly one coordinate. Note that Q_r has 2^r nodes and $r2^r/2$ edges.
   (a) Find a sparsest cut for Q_r and determine its ratio. True or false: the integrality gap for the LP relaxation (F2) on Q_r is one, i.e., the optimal value of (F2) equals the ratio of a sparsest cut.
   (b) Consider the LP relaxation (FM) of the minimum multicut problem on Q_r. Give a tight estimate (up to constant factors) for the integrality gap, i.e., find a function $g(r)$ such that the minimum capacity of a multicut on Q_r is at most $O(g(r))$ times the optimal value of (FM), and such that the integrality gap is $\Omega(g(r))$ on some example. Note that the demand pairs may be arbitrary.
7. (T. C. Hu's theorem) Consider the generalized sparsest cut problem such that exactly two pairs s_1, t_1 and s_2, t_2 have positive demands. Prove that the ratio of a generalized sparsest cut equals the optimal value of the LP relaxation (F4).

   (HINT: One way is to use the method of ℓ_1 embeddings. Construct an isometric embedding from ℓ_∞^2 to ℓ_1^2.)

8. The goal here is to fill in the details for the proof of Bourgain's theorem (Theorem 11.10).

   (a) Let V be a set of n points, and let B^o and B^c be disjoint subsets of V such that $|B^o| \leq 2^{j+1}$ and $|B^c| \geq 2^j$, where $0 \leq j \leq \lfloor \log_2 n \rfloor$ is an integer. Let A be a random subset of V of cardinality $n/2^j$. Prove that there exists a constant $c < 1$ such that with probability $\geq c$, $A \cap B^o$ is empty and $A \cap B^c$ is nonempty.

   (HINT: Argue that the events "$A \cap B^o$ is empty" and "$A \cap B^c$ is nonempty" are not negatively correlated, i.e., $\Pr(A \cap B^o = \emptyset \text{ and } A \cap B^c \neq \emptyset) \geq \Pr(A \cap B^o = \emptyset) \cdot \Pr(A \cap B^c \neq \emptyset)$. Then show that there exist constants c_1 and c_2 such that $\Pr(A \cap B^o = \emptyset) \geq c_1$ and $\Pr(A \cap B^c = \emptyset) \leq c_2$.)

   (b) Use Chernoff bounds to show that with high probability, $\sum_{k=1}^{q} |\phi'_{jq+k}(v) - \phi'_{jq+k}(w)| \geq qc\rho_j/2$, assuming that with probability $\geq c$ for each $k = 1, \ldots, q$, $|\phi'_{jq+k}(v) - \phi'_{jq+k}(w)| \geq \rho_j$.

9. Generalize the proof of Bourgain's theorem to embed into ℓ_p^h for an arbitary p, $p \geq 1$, instead of embedding into ℓ_1^h. The dimension h stays $O(\log^2 n)$ and the distortion stays $O(\log n)$.

   (HINT: Use the following monotonicity property of pth moment averages: For positive numbers $x_1, \ldots, x_k$, $\dfrac{x_1 + \ldots + x_k}{k} \leq \left(\dfrac{x_1^p + \ldots + x_k^p}{k}\right)^{1/p}$.)

10. Construct an example to show that the estimate of the distortion in Bourgain's theorem is tight up to constant factors, i.e., construct a metric space (V, d) such that every embedding into ℓ_1^h has distortion $\Omega(\log n)$, where the dimension h is arbitrary and $n = |V|$.

    (HINT: Consider the example of the sparsest cut problem on the expander graph G in Section 11.1, and focus on the metric space $(V(G), d)$, where d is the shortest-paths metric of G.)

Bibliography

[1] D. Adolphson and T. C. Hu, *Optimal linear ordering*, SIAM J. Appl. Math. **25** (1973), pp. 403-423.

[2] A. Agrawal, P. Klein, and R. Ravi, *Cutting down on fill using nested dissection: provably good elimination orderings*, in Graph Theory and Sparse Matrix Computation, edited by A. George, J. Gilbert, and J. W. H. Liu, Vol. 56 in the *IMA Volumes in Mathematics and its Applications*, Springer-Verlag (1993).

[3] Y. Aumann and Y. Rabani, *An $O(\log k)$ approximate max-flow min-cut theorem and approximation algorithm*, manuscript (1994) to appear in SIAM J. on Comput.

[4] S. N. Bhatt and F. T. Leighton, *A framework for solving VLSI graph layout problems*, J. Comput. Sys. Sciences, **28** (1994), pp. 300-343.

[5] J. Bourgain, *On Lipschitz embedding of finite metric spaces in Hilbert spaces*, Israeli J. Math. **52** (1985), 46-52.

[6] G. Even, J. Naor, S. Rao and B. Schieber, *Divide-and-conquer approximation algorithms via spreading metrics*, Proc. 36th FOCS '95 (1995), 62-71.

[7] N. Garg, *Approximating sparsest cuts*, manuscript, (1994).

[8] N. Garg, V. Vazirani, and M. Yannakakis, *Approximate max-flow min-(multi)cut theorems and their applications*, SIAM J. on Computing, **25:2** (1996), pp. 235-251.

[9] F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms*, manuscript, 1994. An abstract appears in Proc. of the 29th Annual IEEE Conference on Foundations of Computer Science (1988), pp. 422-431.

[10] N. Linial, E. London and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica **15 (2)** (1995), pp. 215-245.

[11] S. Rao and A. W. Richa, *New approximation techniques for some ordering problems*, Proc. 9th SODA '98 (1998), 211-218.

[12] R. Ravi, A. Agrawal, and P. Klein, *Ordering problems approximated: single-processor scheduling and interval graph completion*, Proc. International Colloquium on Automata, Languages and Processing (ICALP '91) LNCS 510 (1991), pp. 751-762.

[13] D. Shmoys, *Cut problems and their application to divide-and-conquer*, in Approximation Algorithms, ed. Dorit Hochbaum, PWS (1997).

Chapter 12

Bicriteria Network Design problems

12.1 Introduction

A generic bicriteria network design problem, (**A**, **B**, **S**), is defined by identifying two minimization objectives, **A** and **B**, from a set of possible objectives, and specifying a membership requirement in a class of subgraphs, **S**. The problem specifies a budget value on the first objective, **A**, under one cost function, and seeks to find a network having minimum possible value for the second objective, **B**, under another cost function, such that this network is within the budget on the first objective. The solution network must belong to the subgraph-class **S**.

As an example, consider the problem of designing networks capable of accommodating multimedia (both audio and video) traffic in a multicast (simultaneous transmission of data to multiple destinations) environment [Ch91, FW+85, KJ83, KP+92A, KP+92B, KP+93]. As argued in [KP+92A], one of the popular solutions to multicast routing involves tree construction. Two optimization criteria – (1) the minimum worst-case transmission delay and (2) the minimum total cost – are typically sought to be minimized in the construction of these trees. As pointed out in [KP+92A], in the problem of finding good multicast trees, each edge has associated with it two edge costs: the construction cost and the delay cost. The construction cost is typically a measure of the amount of buffer space or channel bandwidth used and the delay cost is a combination of the propagation, transmission and queuing delays.

In our terminology, the problem of finding low-cost and low-transmission-delay multimedia networks [KP+92A, KP+93] can be modeled as the (Diameter, Total cost, Spanning tree)-bicriteria problem: given an undirected graph $G = (V, E)$ with two different weight functions c_e (modeling the construction cost) and d_e (modeling the delay cost) for each edge $e \in E$, and a bound $\mathcal{D}$ (on the total delay), find a minimum c-cost spanning tree such that the diameter of the tree under the d-costs is at most $\mathcal{D}$.

Such multi-criteria network design problems, with separate cost functions for each optimization criterion, also occur naturally in VLSI designs (see [ZP+94] and the references therein). With the advent of deep micron VLSI designs, the feature size has shrunk to sizes of 0.5 microns and less. As a result, the interconnect resistance, being proportional to the square of the scaling factor, has increased significantly. An increase in interconnect resistance has led to an increase in interconnect delays thus making them a dominant factor in the timing analysis of VLSI circuits. Therefore

VLSI circuit designers aim at finding minimum cost (spanning or Steiner) trees given delay bound constraints on source-sink connections.

Network design problems where even one cost measure must be minimized, are often **NP**-hard [GJ79]. But, in several applications, it is often the case that the network to be built is required to minimize multiple cost measures simultaneously, with different cost functions for each measure. In the past, the problem of minimizing two cost measures was often dealt with by attempting to minimize some combination of the two, thus converting it into a uni-criterion problem. This approach often fails, especially when the two criteria are very disparate. We have chosen, instead, to model bicriteria problems as that of minimizing one criterion subject to a budget on the other. We argue that this approach is both general as well as robust. It is more general because it subsumes the case where one wishes to minimize some functional combination of the two criteria. It is more robust because the quality of approximation is independent of which of the two criteria we impose the budget on. We elaborate on this more in Sections 12.3.1 and 12.3.2.

12.1.1 Objective functions

In this class, we study the complexity and approximability of a number of bicriteria network design problems. The three objectives we consider are: (i) total cost, (ii) diameter and (iii) degree of the network. These reflect the price of synthesizing the network, the maximum delay between two points in the network and the reliability of the network, respectively. The *Total cost* objective is the sum of the costs of all the edges in the subgraph. The *Diameter* objective is the maximum distance between any pair of nodes in the subgraph. The *Degree* objective denotes the maximum over all nodes in the subgraph, of the degree of the node. We may generalize the degree objective to the case of *Weighted Degree* given costs on the edges as follows: the weighted degree of a node in a subgraph is the sum of the costs of the edges incident on the node in the subgraph. The weighted degree objective is the maximum over all nodes in the subgraph, of the weighted degree of the node. When all the edges have unit costs, the weighted degree objective reduces to the regular degree objective.

The class of subgraphs we consider in are mainly one-connected networks such as *Spanning trees*. However, most of the results we discuss also extend to more general one-connected networks such as Steiner trees and generalized Steiner forests[AK+95, GW92].

12.1.2 Performance guarantees

As we mentioned earlier, most of the problems considered in this paper, are **NP**-hard for arbitrary instances even when we wish to find optimum solutions with respect to a single criterion. Given the hardness of finding optimal solutions, we concentrate on devising approximation algorithms with worst case performance guarantees. Recall that an approximation algorithm for a minimization problem Π provides a **performance guarantee** of ρ if for every instance I of Π, the solution value returned by the approximation algorithm is within a factor ρ of the optimal value for I. Here, we extend this notion to apply to bicriteria optimization problems. An (α, β)-approximation algorithm for an (**A**, **B**, **S**)-bicriteria problem is defined as a polynomial-time algorithm that produces a solution in which the first objective (**A**) value, is at most α times the budget, and the second objective (**B**) value, is at most β times the minimum for any solution that is within the budget on **A**. The solution produced must belong to the subgraph-class **S**. Analogous definitions can be given when **A** and/or **B** are maximization objectives.

The main results we discuss in this note are as follows.

1. Very often, bicriteria problems are harder than uni-criterion problems even to approximate. We demonstrate this by showing an approximation-preserving reduction from the Set Cover problem to the (Diameter, Total cost, Spanning tree) problem that proves that obtaining a $(< \frac{7}{6}, < \ln n)$ approximation for the latter problem is NP-hard. Such "double" hardness results also justify the definition of bicriteria approximations in the way we did.
2. We justify the claim that the formulation of bicriteria problem by using a budget for one objective and minimizing the other is more robust than alternative formulations, and remains consistent irrespective of which of the two objectives we choose to budget.
3. For bicriteria problems where both the objectives are similar, i.e., represent the same objective function but computed under two distinct cost functions (e.g. (Total cost, Total cost, Spanning tree)), we show a simple method based on parametric search to convert uni-criterion approximation results for this objective to a bicriteria result.
4. We present a general framework for approximating bicriteria one-connected network problems that is applicable to all pairwise combinations of the three objective functions we mentioned earlier (total cost, diameter and degree) to give performance ratio $(\log n, \log n)$ on an n-node graph. Given the framework, it remains to reason and fill in the appropriate polynomial time subroutine that is applicable for the corresponding pair of objectives.

Table 1 contains the performance guarantees of our approximation algorithms for finding spanning trees, **S**, under different pairs of minimization objectives, **A** and **B**. For each problem cataloged in the table, two different costs are specified on the edges of the undirected graph: the first objective is computed using the first cost function and the second objective, using the second cost function. The rows are indexed by the budgeted objective. For example the entry in row **A**, column **B**, denotes the performance guarantee for the problem of minimizing objective **B** with a budget on the objective **A**. All the results in Table 1 extend to finding Steiner trees with at most a constant factor worsening in the performance ratios. All the results in the table extend to finding Steiner trees with at most a constant factor worsening in the performance ratios (Exercise!).

Cost Measures	Degree	Diameter	Total Cost
Degree	$(O(\log n), O(\log n))^*$	$(O(\log^2 n), O(\log n))$[Ra94]	$(O(\log n), O(\log n))$[RM+93]
Diameter	$(O(\log n), O(\log^2 n))$[Ra94]	$(1+\gamma, 1+\frac{1}{\gamma})^*$	$(O(\log n), O(\log n))^*$
Total Cost	$(O(\log n), O(\log n))$[RM+93]	$(O(\log n), O(\log n))^*$	$(1+\gamma, 1+\frac{1}{\gamma})^*$

Table 1. Performance Guarantees for finding spanning trees in an arbitrary graph on n nodes. Asterisks indicate results obtained in this paper. $\gamma > 0$ is a fixed accuracy parameter.

The diagonal entries in the table follow as a corollary of a general result (Theorem 12.8) which is proved using a parametric search algorithm. The entry for (Degree, Degree, Spanning tree) follows by combining Theorem 12.8 with the $O(\log n)$-approximation algorithm for the degree problem in [RM+93]. In [RM+93] they actually provide an $O(\log n)$-approximation algorithm for the weighted degree problem. (The weighted degree of a subgraph is defined as the maximum over all nodes of the sum of the costs of the edges incident on the node in the subgraph). Hence we actually get an $(O(\log n), O(\log n))$-approximation algorithm for the (Weighted degree, Weighted degree, Spanning tree)-bicriteria problem. Similarly, the entry for (Diameter, Diameter, Spanning tree) follows by combining Theorem 12.8 with the known exact algorithms for minimum diameter spanning trees [CG82, RS+94]; while the result for (Total cost, Total cost, Spanning tree) follows by combining Theorem 12.8 with an exact algorithm to compute a minimum spanning tree [Kr56, Pr57].

We also describe a different cluster-based approximation algorithm and a solution based decomposition technique for devising approximation algorithms for problems when the two objectives are different. In this note, we describe this technique techniques yield $(O(\log n), O(\log n))$-approximation algorithms for the (Diameter, Total cost, Steiner tree) and the (Degree, Total cost, Steiner tree) problems.

12.1.3 Previous Work

The area of uni-criterion optimization problems for network design is vast and well-explored (See [Ho95, CK95] and the references therein.). Ravi et al. [RM+93] studied the degree-bounded minimum cost spanning tree problem and provided an approximation algorithm with performance guarantee $(O(\log n), O(\log n))$. Though they were doing bicriteria optimization they did not state it as such in their paper.

The (Degree, Diameter, Spanning tree) problem was studied in [Ra94] in the context of minimizing broadcast time in arbitrary networks. There he provides an approximation algorithm for the (Degree, Diameter, Spanning tree) problem with performance guarantee $(O(\log^2 n), O(\log n))$[1].

The (Diameter, Total cost, Spanning tree) entry in Table 1 corresponds to the diameter-constrained minimum spanning tree problem introduced earlier. It is known that this problem is **NP**-hard even in the special case where the two cost functions are identical [HL+89]. Awerbuch, Baratz and Peleg [AB+90] gave an approximation algorithm with $(O(1), O(1))$ performance guarantee for this problem - i.e. the problem of finding a spanning tree that has simultaneously small diameter (i.e., shallow) and small total cost (i.e., light), both under the same cost function. Khuller, Raghavachari and Young [KR+93] studied an extension called *Light, approximate Shortest-path Trees (LAST)* and gave an approximation algorithm with $(O(1), O(1))$ performance guarantee. Kadaba and Jaffe [KJ83], Kompella et al. [KP+92A], and Zhu et al. [ZP+94] considered the (Diameter, Total cost, Steiner tree) problem with two edge costs and presented heuristics without any guarantees. It is easy to construct examples to show that the solutions produced by these heuristics in [ZP+94, KP+92A], can be arbitrarily bad with respect to an optimal solution. A closely related problem is that of finding a diameter-constrained shortest path between two pre-specified vertices s and t, or (Diameter, Total cost, s-t path). This problem, termed the multi-objective shortest path problem (MOSP) in the literature, is **NP**-complete and Warburton [Wa87] presented the first fully polynomial approximation scheme (**FPAS**) for it. Hassin [Ha92] provided a strongly polynomial **FPAS** for the problem which improved the running time of Warburton [Wa87]. This result was further improved by Philips [Ph+93].

The (Total cost, Total cost, Spanning tree)-bicriteria problem has been recently studied by Ganley et al. [GG+95]. They consider a more general problem with more than two weight functions.

[1] The result in [Ra94] is actually somewhat stronger - given a budget, D, on the degree he finds a tree whose total cost is at most $O(\log n)$ times the optimal and whose degree is at most $O(D \log n + \log^2 n)$.

They also gave approximation algorithms for the restricted case when each weight function obeys triangle inequality. However, their algorithm does not have a bounded performance guarantee with respect to each objective.

12.2 Hardness results

The problem of finding a minimum degree spanning tree is strongly **NP**-hard [GJ79]. This implies that all spanning tree bicriteria problems, where one of the criteria is degree, are also strongly **NP**-hard. In contrast, it is well known that the minimum diameter spanning tree problem and the minimum cost spanning tree problems have polynomial time algorithms [CG82, HL+89, Kr56, Pr57, RS+94].

The (Diameter, Total Cost, Spanning tree)-bicriteria problem is strongly **NP**-hard even in the case where both cost functions are identical [HL+89]. We now show that the (Diameter, Total-cost, Steiner tree) problem is hard to approximate within a logarithmic factor. This is in contrast to the approximation algorithm provided in Section 12.5. There is however a gap between the results of Theorems 12.2 and 12.15.

Our non-approximability result is obtained by an approximation preserving reduction from the **MIN SET COVER**. An instance (T, X) of the **MIN SET COVER** problem consists of a universe $T = \{t_1, t_2, \ldots, t_k\}$ and a collection of subsets $X = \{X_1, X_2, \ldots, X_m\}$, $X_i \subseteq T$, each set X_i having an associated cost c_i. The problem is to find a minimum cost collection of the subsets whose union is T. Recently Feige [Fe95] has shown the following non-approximability result:

Theorem 12.1 *Unless $NP \subseteq DTIME(n^{\log\log n})$, the* **MIN SET COVER** *problem, with a universe of size k, cannot be approximated to better than a $\ln k$ factor.*

Theorem 12.2 *Unless $NP \subseteq DTIME(n^{\log\log n})$, given an instance of the (Diameter, Total Cost, Steiner tree) problem with k sites, there is no polynomial-time approximation algorithm that outputs a Steiner tree of diameter at most the bound D, and cost at most R times that of the minimum cost diameter-D Steiner tree, for $R < \ln k$.*

Proof: Approximation preserving reduction from the **MIN SET COVER** problem to the (Diameter, Total Cost, Steiner tree) problem. Given an instance (T, X) of the **MIN SET COVER** problem where $T = \{t_1, t_2, \ldots, t_k\}$ and $X = \{X_1, X_2, \ldots, X_m\}$, $X_i \subseteq T$, where the cost of the set X_i is c_i, we construct an instance G of the (Diameter, Total Cost, Steiner tree) problem as follows. The graph G has a node t_i for each element t_i of T[2], a node x_i for each set X_i, and an extra "enforcer-node" n. For each set X_i, we attach an edge between nodes n and x_i of c-cost c_i, and d-cost 1. For each element t_i and set X_j such that $t_i \in X_j$ we attach an edge (t_i, x_j) of c-cost, 0, and d-cost, 1. In addition to these edges, we add a path P made of two edges of c-cost, 0, and d-cost, 1, to the enforcer node n All other edges in the graph are assigned infinite c and d-costs. The nodes t_i along with n and the two nodes of P are specified to be the terminals for the Steiner tree problem instance. We claim that G has a c-cost Steiner tree of diameter at most 4 and cost $\mathcal{C}$ if and only if the original instance (T, X) has a solution of cost $\mathcal{C}$.

[2] There is a mild abuse of notation here but it should not lead to any confusion.

It is easy to see that any Steiner tree of diameter at most 4 must contain a path from t_i to n, for all i, that uses an edge (x_j, n) for some X_j such that $t_i \in X_j$. Hence any Steiner tree of diameter at most 4 provides a feasible solution of equivalent c-cost to the original Set cover instance. The proof now follows from Theorem 12.1.

∎

Exercise Show that a construction similar to the one above can be used to strengthen the condition of the output Steiner tree having diameter at most D in the above Theorem to having diameter less than $\frac{6D}{5}$.

12.3 Bicriteria Formulations: Simple Properties

12.3.1 Equivalence of Bicriteria Formulations: Robustness

There are two natural alternative ways of formulating general bicriteria problems: (i) where we impose the budget on the first objective and seek to minimize the second and (ii) where we impose the budget on the second objective and seek to minimize the first. We show that an (α, β)-approximation algorithm for one of these formulations naturally leads to a (β, α)-approximation algorithm for the other. Thus our definition of a bicriteria approximation is independent of the choice of the criterion that is budgeted in the formulation. This makes it a robust definition and allows us to fill in the entries for the problems (**B**, **A**, **S**) by transforming the results for the corresponding problems (**A**, **B**, **S**).

Let G be a graph with two (integral)[3] cost functions, c and d. c and d are typically edge costs or node costs. Let **A** (**B**) be a minimization objective computed using cost function c (d). Let the budget bound on the c-cost[4] (d-cost) of a solution subgraph be denoted by $\mathcal{C}$ ($\mathcal{D}$).

There are two natural ways to formulate a bicriteria problem: (i) (**A**, **B**, **S**) - find a subgraph in **S** whose **A**-objective value (under the c-cost) is at most $\mathcal{C}$ and which has minimum **B**-objective value (under the d-cost), (ii) (**B**, **A**, **S**) - find a subgraph in **S** whose **B**-objective value (under the d-cost) is at most $\mathcal{D}$ and which has minimum **A**-objective value (under the c-cost).

Note that bicriteria problems are meaningful only when the two criteria are *hostile* with respect to each other - the minimization of one criterion conflicts with the minimization of the other. A good example of hostile objectives are the degree and the total edge cost of a spanning tree in an unweighted graph [RM+93]. Two minimization criteria are formally defined to be hostile whenever the minimum value of one objective is monotonically nondecreasing as the budget (bound) on the value of the other objective is decreased.

Let $\mathbf{XYZ}(G, \mathcal{C})$ be any (α, β)-approximation algorithm for (**A**, **B**, **S**) on graph G with budget $\mathcal{C}$ under the c-cost. We now show that there is a transformation which produces a (β, α)-approximation algorithm for (**B**, **A**, **S**). The transformation uses binary search on the range of values of the c-cost with an application of the given approximation algorithm, **XYZ**, at each step of this search. Let the minimum c-cost of a $\mathcal{D}$-bounded subgraph in **S** be OPT_c. Let $\mathcal{C}_{hi}$ be an upper

[3] In case of rational cost functions, our algorithms can be easily extended with a small additive loss in the performance guarantee.

[4] We use the term "cost under c" or "c-cost" in this section to mean the value of the objective function computed using c, and not to mean the total of all the c costs in the network.

bound on the c-cost of any $\mathcal{D}$-bounded subgraph in $\mathbf{S}$. Note that $\mathcal{C}_{hi}$ is at most some polynomial in n times the maximum c-cost (of an edge or a node). Hence $\log(\mathcal{C}_{hi})$ is at most a polynomial in terms of the input specification. Let Heu_c (Heu_d) denote the c-cost (d-cost) of the subgraph output by ALGORITHM BICRITERIA-EQUIVALENCE given below.

ALGORITHM BICRITERIA-EQUIVALENCE:

- *Input:* G - graph, $\mathcal{D}$ - budget on criterion $\mathbf{B}$ under the d-cost, $\mathbf{XYZ}$ - an (α, β)-approximation algorithm for ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{S}$).
- 1. Let $\mathcal{C}_{hi}$ be an upper bound on the c-cost of any $\mathcal{D}$-bounded subgraph in $\mathbf{S}$.
  2. Do binary search and find a $\mathcal{C}'$ in $[0, \mathcal{C}_{hi}]$ such that
     (a) $\mathbf{XYZ}(G, \mathcal{C}')$ returns a subgraph with d-cost greater than $\beta\mathcal{D}$, and
     (b) $\mathbf{XYZ}(G, \mathcal{C}' + 1)$ returns a subgraph with d-cost at most $\beta\mathcal{D}$.
  3. If the binary search in Step 2 fails to find a valid $\mathcal{C}'$ then output "NO SOLUTION" else output $\mathbf{XYZ}(G, \mathcal{C}' + 1)$.
- *Output:* A subgraph from $\mathbf{S}$ such that its c-cost is at most α times that of the minimum c-cost $\mathcal{D}$-bounded subgraph and its d-cost is at most $\beta\mathcal{D}$.

Claim 12.3 *If G contains a $\mathcal{D}$-bounded subgraph in* $\mathbf{S}$ *then* ALGORITHM BICRITERIA-EQUIVALENCE *outputs a subgraph from* $\mathbf{S}$ *whose c-cost is at most α times that of the minimum c-cost $\mathcal{D}$-bounded subgraph and whose d-cost is at most $\beta\mathcal{D}$.*

Proof: Since $\mathbf{A}$ and $\mathbf{B}$ are hostile criteria it follows that the binary search in Step 2 is well defined. Assume that $\mathbf{S}$ contains a $\mathcal{D}$-bounded subgraph. Then, since $\mathbf{XYZ}(G, \mathcal{C}_{hi})$ returns a subgraph with d-cost at most $\beta\mathcal{D}$, it is clear that ALGORITHM BICRITERIA-EQUIVALENCE outputs a subgraph in this case. As a consequence of Step 2a and the performance guarantee of the approximation algorithm $\mathbf{XYZ}$, we get that $\mathcal{C}' + 1 \leq OPT_c$. By Step 2b we have that $Heu_d \leq \beta\mathcal{D}$ and $Heu_c \leq \alpha(\mathcal{C}'+1) \leq \alpha OPT_c$. Thus ALGORITHM BICRITERIA-EQUIVALENCE outputs a subgraph from $\mathbf{S}$ whose c-cost is at most α times that of the minimum c-cost $\mathcal{D}$-bounded subgraph and whose d-cost is at most $\beta\mathcal{D}$.

∎

Note however that in general the resulting (β, α)-approximation algorithm is, not *strongly* polynomial since it depends on the range of the c-costs. But it is a *polynomial-time* algorithm since its running time is linearly dependent on $\log \mathcal{C}_{hi}$ the largest c-cost. The above discussion leads to the following theorem.

Theorem 12.4 *Any (α, β)-approximation algorithm for* ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{S}$) *can be transformed in polynomial time into a (β, α)-approximation algorithm for* ($\mathbf{B}$, $\mathbf{A}$, $\mathbf{S}$).

12.3.2 Comparing with other functional combinations: Generality

Our formulation is more general because it subsumes the case where one wishes to minimize some functional combination of the two criteria. We briefly comment on this next. For the purposes of illustration let **A** and **B** be two objective functions and let us say that we wish to minimize the sum of the two objectives **A** and **B**. Call this an (**A** + **B**, **S**) problem. Let $\mathbf{XYZ}(G, \mathcal{C})$ be any (α, β)-approximation algorithm for (**A**, **B**, **S**) on graph G with budget $\mathcal{C}$ under the c-cost. We show that there is a polynomial time $\max\{\alpha, \beta\}$-approximation algorithm for the (**A** + **B**, **S**) problem. The transformation uses binary search on the range of values of the c-cost with an application of the given approximation algorithm, **XYZ**, at each step of this search. Let the optimum value for the (**A** + **B**, **S**) problem on a graph G be $OPT_{c+d} = (V_c + V_d)$, where V_c and V_d denote respectively the contribution of the two costs c and d for **A** and **B**. Let $Heu_c(\mathcal{C})$ ($Heu_d(\mathcal{C})$) denote the c-cost (d-cost) of the subgraph output by $\mathbf{XYZ}(G, \mathcal{C})$. Finally, let $Heu_{c+d}(\mathcal{C})$ denote the value computed by ALGORITHM CONVERT.

ALGORITHM CONVERT:

- *Input:* G - graph, **XYZ** - an (α, β)-approximation algorithm for (**A**, **B**, **S**).
- 1. Let $\mathcal{C}_{hi}$ be an upper bound on the c-cost of any subgraph in **S**.
  2. Employ binary search to find a $\mathcal{C}'$ in $[0, \mathcal{C}_{hi}]$ such that
     (a) $\mathbf{XYZ}(G, \mathcal{C}')$ returns a subgraph with the minimum value of $Heu_c(\mathcal{C}') + Heu_d(\mathcal{C}')$.
- *Output:* A subgraph from **S** such that the sum of its c-cost and its d-costs is at most $\max\{\alpha, \beta\}(OPT_{c+d})$.

Theorem 12.5 *Let* $\mathbf{XYZ}(G, \mathcal{C})$ *be any* (α, β)*-approximation algorithm for* (**A**, **B**, **S**) *on graph* G *with budget* $\mathcal{C}$ *under the* c*-cost. Then, there is a polynomial time* $\max\{\alpha, \beta\}$*-approximation algorithm for the* (**A** + **B**, **S**) *problem.*

Proof Sketch: Consider the iteration of the binary search in which the bound on the c-cost is V_c. Then as a consequence of the performance guarantee of the approximation algorithm **XYZ**, we get that $Heu_c(V_c) \leq \alpha V_c$. By Step 2a and the performance guarantee of the algorithm **XYZ**, we have that $Heu_d(V_c) \leq \beta V_d$. Thus $Heu_{c+d}(V_c) \leq \alpha V_c + \beta V_d \leq \max\{\alpha, \beta\}(V_c + V_d)$. Since ALGORITHM CONVERT outputs a subgraph from **S** the sum of whose c-cost and d-cost is minimized, we have that

$$\min_{\mathcal{C}' \in [0, \mathcal{C}_{hi}]} (Heu_c(\mathcal{C}') + Heu_d(\mathcal{C}')) \leq \max\{\alpha, \beta\}(OPT_{c+d}).$$

A similar argument shows that an (α, β)-approximation algorithm $\mathbf{XYZ}(G, \mathcal{C})$, for a (**A**, **B**, **S**) problem can be used to find devise a polynomial time $\alpha\beta$ approximation algorithm for the (**A** $\times$ **B**, **S**) problem. A similar argument can also be given for other basic functional combinations. We make two additional remarks.

1. The use of approximation algorithms for (**A**, **B**, **S**)-bicriteria problems, to solve (f(**A**, **B**), **S**) problems (f denotes a function combination of the objectives) does not always yield the best possible solutions. For example problems such as (Total Cost + Total Cost , Spanning Tree) and (Total Cost/Total Cost , Spanning Tree) [Ch77, Me83] can be solved exactly in

polynomial time by direct methods but can only be solved approximately using any algorithm for the (Total Cost, Total Cost , Spanning Tree)-bicriteria problem.[5]

2. Algorithms for solving (f(**A**, **B**), **S**) problems can not in general guarantee any bounded performance ratios for solving the (**A**, **B**, **S**) problem. For example, a solution for the (Total Cost + Total Cost , Spanning Tree) problem or the (Total Cost/Total Cost , Spanning Tree) problem can not be directly used to find a good (α, β)-approximation algorithm for the (Total Cost, Total Cost, Spanning Tree)-bicriteria problem.

The above discussion points out that while a good solution to the (**A**, **B**, **S**)-bicriteria problem yields a "good" solution to any uni-criterion version, the converse is not true. It is in this sense that we say our formulation of bicriteria network design problems is general and subsumes other functional combinations.

12.4 Parametric Search

In this section, we present approximation algorithms for a broad class of bicriteria problems where both the objectives in the problem are of the same type (e.g., both are total edge costs of some network computed using two different costs on edges, or both are diameters of some network calculated using two different costs etc.).

As before, let G be a graph with two (integral) cost functions, c and d. Let $\mathcal{C}$ denote the budget on criteria **A**. We assume that the c and d cost functions are of the same kind, i.e. they are both costs on edges or, costs on nodes. Let $\mathbf{UVW}(G, f)$ be any ρ-approximation algorithm that on input G produces a solution subgraph in **S** minimizing criterion **A**, under the single cost function f. In a mild abuse of notation, we also let $\mathbf{UVW}(G, f)$ denote the (f-)cost of the subgraph output by $\mathbf{UVW}(G, f)$ when running on input G under cost function f. We use the following additional notation in the description of the algorithm and the proof of its performance guarantee. Given constants a and b and two cost functions f and g, defined on edges (nodes) of a graph, $af + bg$ denotes the composite function that assigns a cost $af(e) + bg(e)$ to each edge (node) in the graph. Let $h(\hat{\mathcal{D}})$ denote the cost of the subgraph, returned by $\mathbf{UVW}(G, (\frac{\hat{\mathcal{D}}}{\mathcal{C}})c + d)$ (under the $((\frac{\hat{\mathcal{D}}}{\mathcal{C}})c + d)$-cost function). Let the minimum d-cost of a $\mathcal{C}$-bounded subgraph in **S** be OPT_d. Let Heu_c (Heu_d) denote the c-cost (d-cost) of the subgraph output by ALGORITHM PARAMETRIC-SEARCH given below.

Let $\gamma > 0$ be a fixed accuracy parameter. In what follows, we devise a $((1 + \gamma), (1 + \frac{1}{\gamma}))$-approximation algorithm for (**A**, **A**, **S**), under the two cost functions c and d. The algorithm consists of performing a binary search with an application of the given approximation algorithm, **UVW**, at each step of this search.

[5] This is true since the (Total Cost, Total Cost, Spanning Tree)-bicriteria problem is **NP**-complete and therefore unless **P** = **NP** can not be solved in polynomial time.

ALGORITHM PARAMETRIC-SEARCH:

- *Input:* G - graph, $\mathcal{C}$ - budget on criteria **A** under the c-cost, **UVW** - a ρ-approximation algorithm that produces a solution subgraph in **S** minimizing criterion **A**, under a single cost function, γ - an accuracy parameter.
- 1. Let $\mathcal{D}_{hi}$ be an upper bound on the d-cost of any $\mathcal{C}$-bounded subgraph in **S**.
  2. Do binary search and find a $\mathcal{D}'$ in $[0, \frac{\mathcal{D}_{hi}}{\gamma}]$ such that
     (a) $\mathbf{UVW}(G, (\frac{\mathcal{D}'}{\mathcal{C}})c + d)$ returns a subgraph such that $\frac{h(\mathcal{D}')}{\mathcal{D}'} > (1+\gamma)\rho$, and
     (b) $\mathbf{UVW}(G, (\frac{\mathcal{D}'+1}{\mathcal{C}})c + d)$ returns a subgraph such that $\frac{h(\mathcal{D}'+1)}{(\mathcal{D}'+1)} \leq (1+\gamma)\rho$.
  3. If the binary search in Step 2 fails to find a valid $\mathcal{C}'$ then output "NO SOLUTION" else output $\mathbf{UVW}(G, (\frac{\mathcal{D}'+1}{\mathcal{C}})c + d)$.
- *Output:* A subgraph from **S** such that its d-cost is at most $(1+\frac{1}{\gamma})\rho$ times that of the minimum d-cost $\mathcal{C}$-bounded subgraph and its c-cost is at most $(1+\gamma)\rho\mathcal{C}$.

Claim 12.6 *The binary search, in Step 2 of* ALGORITHM PARAMETRIC-SEARCH *is well-defined.*

Proof: Since $(\frac{1}{R}\mathbf{UVW}(G, f))$ is the same as $\mathbf{UVW}(G, \frac{f}{R})$, we get that $\frac{h(\hat{\mathcal{D}})}{\hat{\mathcal{D}}} = \frac{1}{\hat{\mathcal{D}}}\,\mathbf{UVW}(G, (\frac{\hat{\mathcal{D}}}{\mathcal{C}})c + d) = \mathbf{UVW}(G, (\frac{1}{\mathcal{C}})c + \frac{1}{\hat{\mathcal{D}}}d)$, Hence $\frac{h(\hat{\mathcal{D}})}{\hat{\mathcal{D}}}$ is a monotone non-increasing function of $\hat{D}$. Thus the binary search in Step 2 of ALGORITHM PARAMETRIC-SEARCH is well-defined. ∎

Claim 12.7 *If G contains a $\mathcal{C}$-bounded subgraph in* **S** *then* ALGORITHM PARAMETRIC-SEARCH *outputs a subgraph from* **S** *whose d-cost is at most $(1+\frac{1}{\gamma})\rho$ times that of the minimum d-cost $\mathcal{C}$-bounded subgraph and whose c-cost is at most $(1+\gamma)\rho\mathcal{C}$.*

Proof: By claim 12.6 we have that the binary search in Step 2 of ALGORITHM PARAMETRIC-SEARCH is well-defined.

Assume that **S** contains a $\mathcal{C}$-bounded subgraph. Then, since $\mathbf{UVW}(G, (\frac{\mathcal{D}_{hi}}{\gamma\mathcal{C}})c + d)$ returns a subgraph with cost at most $(1+\gamma)\rho\mathcal{D}_{hi}$, under the $((\frac{\mathcal{D}_{hi}}{\gamma\mathcal{C}})c + d)$-cost function, it is clear that ALGORITHM PARAMETRIC-SEARCH outputs a subgraph in this case.

As a consequence of Step 2a and the performance guarantee of the approximation algorithm **UVW**, we get that

$$\mathcal{D}' + 1 \leq \frac{OPT_d}{\gamma}.$$

By Step 2b we have that the subgraph output by ALGORITHM PARAMETRIC-SEARCH has the following bounds on the c-costs and the d-costs.

$$Heu_d \leq h(\mathcal{D}'+1) \leq \rho(1+\gamma)(\mathcal{D}'+1) \leq (1+\frac{1}{\gamma})\rho OPT_d$$

$$Heu_c \leq (\frac{\mathcal{C}}{\mathcal{D}'+1})h(\mathcal{D}'+1) \leq (\frac{\mathcal{C}}{\mathcal{D}'+1})(1+\gamma)\rho(\mathcal{D}'+1) = (1+\gamma)\rho\mathcal{C}.$$

Thus ALGORITHM PARAMETRIC-SEARCH outputs a subgraph from **S** whose d-cost is at most $(1+\frac{1}{\gamma})\rho$ times that of the minimum d-cost $\mathcal{C}$-bounded subgraph and whose c-cost is at most $(1+\gamma)\rho\mathcal{C}$.

∎

Note however that the resulting $((1+\gamma)\rho, (1+\frac{1}{\gamma})\rho)$-approximation algorithm for (**A**, **A**, **S**) may not be *strongly* polynomial since it depends on the range of the d-costs. But it is a *polynomial-time* algorithm since its running time is linearly dependent on $\log D_{hi}$. Note that $\mathcal{D}_{hi}$ is at most some polynomial in n times the maximum d-cost (of an edge or a node). Hence $\log(\mathcal{D}_{hi})$ is at most a polynomial in terms of the input specification.

The above discussion leads to the following theorem.

Theorem 12.8 *Any ρ-approximation algorithm that produces a solution subgraph in* **S** *minimizing criterion* **A** *can be transformed into a $((1+\gamma)\rho, (1+\frac{1}{\gamma})\rho)$-approximation algorithm for (**A**,**A**,**S**).*

12.5 Diameter-Constrained Trees

In this section, we describe ALGORITHM DCST, our $(O(\log n), O(\log n))$-approximation algorithm for (Diameter, Total cost, Steiner tree) or the diameter-bounded minimum Steiner tree problem. Note that (Diameter, Total cost, Steiner tree) includes (Diameter, Total cost, Spanning tree) as a special case. We first state the problem formally: given an undirected graph $G=(V,E)$, with two cost functions c and d defined on the set of edges, diameter bound D and terminal set $K \subseteq V$, the (Diameter, Total cost, Steiner tree) problem is to find a tree of minimum c-cost connecting the set of terminals in K with diameter at most D under the d-cost.

The technique underlying ALGORITHM DCST is very general and has wide applicability. Hence, we first give a brief synopsis of it. The basic algorithm works in $(\log n)$ phases. Initially the solution consists of the empty set. During each phase of the algorithm we execute a subroutine Ω to choose a subgraph to add to the solution. The subgraph chosen in each iteration is required to possess two desirable properties. First, it must not increase the budget value of the solution by more than D; second, the solution cost with respect to **B** must be no more than OPT_c, where OPT_c denotes the minimum c-cost of a $\mathcal{D}$ bounded subgraph in **S**. Since the number of iterations of the algorithm is $O(\log n)$ we get a $(\log n, \log n)$-approximation algorithm. The basic technique is fairly straightforward. The non-trivial part is to devise the right subroutine Ω to be executed in each phase. Ω must be chosen so as to be able to prove the required performance guarantee of the solution. We use the solution based decomposition technique [KR93, Ra94, RM+93] in the analysis of our algorithm. The basic idea (behind the solution based decomposition technique) is to use the existence of an optimal solution to prove that the subroutine Ω finds the desired subgraph in each phase.

We now present the specifics of ALGORITHM DCST. The algorithm maintains a set of connected subgraphs or *clusters* each with its own distinguished vertex or *center*. Initially each terminal is in a cluster by itself. In each phase, clusters are merged in pairs by adding paths between their centers. Since the number of clusters comes down by a factor of 2 each phase, the algorithm terminates in $\lceil \log_2 |K| \rceil$ phases with one cluster. It outputs a spanning tree of the final cluster as the solution.

ALGORITHM DIAMETER-CONSTRAINED-STEINER-TREE (DCST):

- *Input:* $G = (V, E)$ - graph with two edge cost functions, c and d, D - a bound on the diameter under the d-cost, $K \subseteq V$ - set of terminals, ϵ - an accuracy parameter.
- 1. Initialize the set of clusters $\mathcal{C}_1$ to contain $|K|$ singleton sets, one for each terminal in K. For each cluster in $\mathcal{C}$, define the single node in the cluster to be the center for the cluster. Initialize the phase count $i := 1$.
  2. Repeat until there remains a single cluster in $\mathcal{C}_i$
     (a) Let the set of clusters $\mathcal{C}_i = \{C_1 \ldots, C_{k_i}\}$ at the beginning of the i'th phase (observe that $k_1 = |K|$).
     (b) Construct a complete graph G_i as follows: The node set V_i of G_i is $\{v : v \text{ is the center of a cluster in } \mathcal{C}\}$. Let path P_{xy} be a $(1 + \epsilon)$-approximation to the minimum c-cost diameter D-bounded path between centers v_x and v_y in G. Between every pair of nodes v_x and v_y in V_i, include an edge (v_x, v_y) in G_i of weight equal to the c-cost of P_{xy}.
     (c) Find a minimum-weight matching of largest cardinality in G_i.
     (d) For each edge $e = (v_x, v_y)$ in the matching, merge clusters C_x and C_y, for which v_x and v_y were centers respectively, by adding path P_{xy} to form a new cluster C_{xy}. The node (edge) set of the cluster C_{xy} is defined to be the union of the node (edge) sets of C_x, C_y and the nodes (edges) in P_{xy}. One of v_x and v_y is (arbitrarily) chosen to be the center v_{xy} of cluster C_{xy}. C_{xy} is added to the cluster set $\mathcal{C}_{i+1}$ for the next phase.
     (e) $i := i + 1$.
  3. Let C', with center v' be the single cluster left after Step 2. Output a shortest path tree of C' rooted at v' using the d-cost.
- *Output:* A Steiner tree connecting the set of terminals in K with diameter at most $2\lceil \log_2 n \rceil D$ under the d-cost and of total c-cost at most $(1 + \epsilon)\lceil \log_2 n \rceil$ times that of the minimum c-cost diameter D-bounded Steiner tree.

We make a few points about ALGORITHM DCST:

1. The clusters formed in Step 2d need not be disjoint.
2. All steps, except Step 2b, in algorithm DCST can be easily seen to have running times independent of the weights. We employ Hassin's strongly polynomial **FPAS** for Step 2b [Ha92]. Hassin's approximation algorithm for the D-bounded minimum c-cost path runs in time $O(|E|(\frac{n^2}{\epsilon} \log \frac{n}{\epsilon}))$. Thus ALGORITHM DCST is a strongly polynomial time algorithm.
3. Instead of finding an exact minimum cost matching in Step 2c, we could find an approximate minimum cost matching [GW92]. This would reduce the running time of the algorithm at the cost of adding a factor of 2 to the performance guarantee.

We now state some observations that lead to a proof of the performance guarantee of ALGORITHM DCST. Assume, in what follows, that G contains a diameter D-bounded Steiner tree.

Claim 12.9 *Algorithm DCST terminates in* $\lceil \log_2 |K| \rceil$ *phases.*

Proof: Let k_i denote the number of clusters in phase i. Note that $k_{i+1} = \lceil \frac{k_i}{2} \rceil$ since we pair up the clusters (using a matching in Step 2d). Hence we are left with one cluster after phase $\lceil \log_2 |K| \rceil$ and algorithm DCST terminates.

∎

Claim 12.10 *Let* $C \in \mathcal{C}_i$ *be any cluster in phase* i *of algorithm DCST. Let* v *be the center of* C. *Then any node* u *in* C *is reachable from* v *by a diameter-*iD *path in* C *under the* d*-cost.*

Proof: Note that the existence of a diameter D-bounded Steiner tree implies that all paths added in Step 2d have diameter at most D under d-cost. The proof now follows in straightforward fashion by induction on i.

∎

Lemma 12.11 *Algorithm DCST outputs a Steiner tree with diameter at most* $2\lceil \log_2 |K| \rceil \cdot D$ *under the* d*-cost.*

Proof: The proof follows from Claims 12.9 and 12.10.

∎

This completes the proof of performance guarantee with respect to the d-cost. We now proceed to prove the performance guarantee with respect to the c-costs. We first recall the following pairing lemma.

Claim 12.12 *[KR93, RM+93] Let* T *be an edge-weighted tree with an even number of marked nodes. Then there is a pairing* (v_1, w_1), ..., (v_k, w_k) *of the marked nodes such that the* $v_i - w_i$ *paths in* T *are edge-disjoint.*

Proof: A pairing of the marked nodes that minimizes the sum of the lengths of the tree-paths between the nodes paired up can be seen to obey the property in the claim above.

∎

Claim 12.13 *Let* OPT *be any minimum* c*-cost diameter-*D *bounded Steiner tree and let* OPT_c *denote its* c*-cost. The weight of the largest cardinality minimum-weight matching found in Step 2d in each phase* i *of algorithm DCST is at most* $(1+\epsilon) \cdot OPT_c$.

Proof: Consider the phase i of algorithm DCST. Note that since the centers at stage i are a subset of the nodes in the first iteration, the centers v_i are terminal nodes. Thus they belong to OPT. Mark those vertices in OPT that correspond to the matched vertices, $v_1, v_2, \ldots, v_{2\lfloor \frac{k_i}{2} \rfloor}$, of G_i in Step 2c. Then by Claim 12.12 there exists a pairing of the marked vertices, say $(v_1, v_2), \ldots, (v_{2\lfloor \frac{k_i}{2} \rfloor - 1}, v_{2\lfloor \frac{k_i}{2} \rfloor})$, and a set of edge-disjoint paths in OPT between these pairs. Since these paths are edge-disjoint their total c-cost is at most OPT_c. Further these paths have diameter at most D under the d-cost. Hence the sum of the weights of the edges $(v_1, v_2), \ldots, (v_{2\lfloor \frac{k_i}{2} \rfloor - 1}, v_{2\lfloor \frac{k_i}{2} \rfloor})$ in G_i , which forms a perfect matching on the set of matched vertices, is at most $(1+\epsilon)\cdot OPT_c$. But in Step 2c of ALGORITHM DCST, a minimum weight perfect matching in the graph G_i was found. Hence the weight of the matching found in Step 2d in phase i of ALGORITHM DCST is at most $(1+\epsilon)\cdot OPT_c$. ∎

Lemma 12.14 *Let OPT be any minimum c-cost diameter-D bounded Steiner tree and let OPT_c denote itsc-cost.* ALGORITHM DCST *outputs a Steiner tree with total c-cost at most* $(1+\epsilon)\lceil\log_2|K|\rceil\cdot OPT_c$.

Proof: From Claim 12.13 we have that the c-cost of the set of paths added in Step 2d of any phase is at most $(1+\epsilon)\cdot OPT_c$. By Claim 12.9 there are a total of $\lceil\log_2|K|\rceil$ phases and hence the Steiner tree output by ALGORITHM DCST has total c-cost at most $(1+\epsilon)\lceil\log_2|K|\rceil\cdot C_D$. ∎

From Lemmas 12.11 and 12.14 we have the following theorem.

Theorem 12.15 *There is a strongly polynomial-time algorithm that, given an undirected graph $G=(V,E)$, with two cost functions c and d defined on the set of edges, diameter bound D, terminal set $K\subseteq V$ and a fixed $\epsilon>0$, constructs a Steiner tree of G of diameter at most $2\lceil\log_2|K|\rceil D$ under the d-costs and of total c-cost at most $(1+\epsilon)\lceil\log_2|K|\rceil$ times that of the minimum-c-cost of any Steiner tree with diameter at most D under d.*

12.6 Exercises

1. Given an undirected graph with nonnegative costs on the edges, the routing cost of any of its spanning trees is the sum over all pairs of nodes, of the cost of the path between the pair in the tree. Give a 2-approximation algorithm for finding a spanning tree of minimum routing cost. (Hint: Consider all the different shortest-path trees).
2. Consider the bicriteria problem (Total edge cost, Routing cost, spanning tree) on an undirected graph where both the total cost and routing costs are computed using the same cost function c_e on the edges. Give a $(O(1),O(1))$ approximation for the problem. (Hint: Reduce the routing cost problem to a shortest path problem, and look in the literature for an appropriate bicriteria result).
3. The buy-at-bulk network design problem [SC+97] is one where the capacities on the edges must be bought in pre-specified bundles for a fixed cost per unit length. In the single-sink single-cable-type version of the problem, we are given an undirected complete graph with a length metric on the nodes, and a specification of the (dollar) cost of buying one unit length of cable with a bandwidth (capacity) of one unit. Moreover, we are given demand amounts at the non-sink nodes that must be routed to the sink node. The problem is to lay down appropriate number of copies of the cable on the edges that supports a routing of all the demands to the sink at minimum total cable cost.

   Two natural lower bounds for the problem arise from *connectivity*, i.e., having to connect all the positive demand nodes to the sink in a Steiner tree with at least one copy of the cable on each edge, and from *routing*, i.e., having to route the demands, each at least along a shortest path to the sink fractionally. Identify the appropriate bicriteria problem that is relevant to the design of an approximation algorithm for the above buy-at-bulk problem; Search the literature for a method that results in an $O(1)$-approximate solution. (Hint: The result of the literature search in the previous problem may be useful for this one as well.)

4. Consider the general strategy for obtaining $(O(\log n), O(\log n))$ approximation algorithms for bicriteria spanning trees on an n-node graph. Suppose the two objectives are (weighted degree, total cost), and these are computed under two distinct cost function (The weighted degree of a node in an edge-weighted spanning tree is the sum of the weights of the edges incident on the node in the tree). What is the subroutine that you need at each stage to get the $(O(\log n), O(\log n))$ result? Do a literature survey to find such a result. What if the objectives are (unweighted degree, total cost), i.e., the case when all edges in the graph have unit weight?

Bibliography

[AB+90] B. Awerbuch, A. Baratz, and D. Peleg, "Cost-sensitive analysis of communication protocols," *Proceedings of the 9th Symposium on Principles of Distributed Computing (PODC)*, pp. 177-187 (1990).

[AK+95] A. Agrawal, P. Klein and R. Ravi, "When trees collide: an approximation algorithm for the generalized Steiner problem on networks," *SIAM Journal on Computing*, vol.24, pp. 440-456 (1995).

[CG82] P. M. Camerini, and G. Galbiati, "The bounded path problem," *SIAM Journal on Algebraic and Discrete Methods* vol. 3, no. 4, pp. 474-484 (1982).

[Ch77] R. Chandrasekaran, "Minimum Ratio Spanning Trees," *Networks,* vol. 7, pp. 335-342, (1977).

[Ch91] C.-H. Chow, "On multicast path finding algorithms," *Proceedings of IEEE INFOCOM 1991*, pp. 1274-1283 (1991).

[CK95] P. Crescenzi and V. Kann, "A compendium of **NP** optimization problems," Manuscript, (1995).

[Fe95] U. Feige, "A threshold of $\ln n$ for approximating set cover," To appear in the *Proceedings of the 28th Annual ACM Symposium on the Theory of Computation* (1996).

[FW+85] A. Frank, L. Wittie, and A. Bernstein, "Multicast communication in network computers," *IEEE Software*, vol. 2, no. 3, pp. 49-61 (1985).

[GG+95] J. L. Ganley, M. J. Golin and J. S. Salowe, "The multi-weighted spanning tree problem," *Proceedings of the First Conference on Combinatorics and Computing (COCOON)*, Springer Verlag, LNCS pp. 141-150 (1995).

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco (1979).

[GW92] M. X. Goemans and D. P. Williamson, "A general approximation technique for constrained forest problems," *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 307-316 (1992). To appear in *SIAM Journal on Computing.*

[Ha92] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, vol. 17, no. 1, pp. 36-42 (1992).

[HL+89] J. Ho, D.T. Lee, C.H. Chang and C.K. Wong, "Bounded diameter spanning tree and related problems," *Proceedings of the Annual ACM Symposium on Computational Geometry*, pp. 276-282 (1989).

[Ho95] D. Hochbaum, *Approximation algorithms for NP-hard problems,* D.S. Hochbaum Ed., PWS Publishing Company, Boston, MA (1995).

[Kr56] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the AMS*, Vol. 7, No. 1, pp. 48-50 (1956).

[KJ83] B. Kadaba and J. Jaffe, "Routing to multiple destinations in computer networks," *IEEE Transactions on Communications*, Vol. COM-31, pp. 343-351 (March 1983).

[KR+93] S. Khuller, B. Raghavachari, and N. Young, "Balancing minimum spanning and shortest path trees," *Algorithmica,* vol. 14 (4), pp. 305-321, (1995).

[KP+92A] V.P. Kompella, J.C. Pasquale and G.C. Polyzos, "Multicasting for multimedia applications," *Proceedings of IEEE INFOCOM 1992* (May 1992).

[KP+92B] V.P. Kompella, J.C. Pasquale and G.C. Polyzos, "Two distributed algorithms for the constrained Steiner tree problem," Technical Report CAL-1005-92, Computer Systems Laboratory, University of California, San Diego (Oct. 1992).

[KP+93] V.P. Kompella, J.C. Pasquale and G.C. Polyzos, "Multicast routing for multimedia communication," *IEEE/ACM Transactions on Networking*, pp. 286-292 (1993).

[KR93] P. N. Klein, and R. Ravi, "A nearly best-possible approximation for node-weighted Steiner trees," *Journal of Algorithms,* No. 19, pp. 104-115, (1995).

[Me83] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *Journal of the ACM (JACM)*, vol. 30, pp. 852-865, (1983).

[Ph+93] C. Philips, "The Network Inhibition Problem," *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pp. 776-785, (1993).

[Pr57] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, vol. 36, pp. 1389-1401 (1957).

[Ra94] R. Ravi, "Rapid rumor ramification: approximating the minimum broadcast time," *Proceedings of the 35th Annual IEEE Foundations of Computer Science*, pp. 202-213 (1994).

[RG95] R. Ravi and M. Goemans, "The constrained spanning tree problem," to appear in the *Proceedings of the 5th Scandinavian Workshop on Algorithmic Theory*, 1996.

[RM+93] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H.B. Hunt III, "Many birds with one stone: multi-objective approximation algorithms," *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pp. 438-447 (1993). (Expanded version appears as Brown University Technical Report TR-CS-92-58.)

[RS+94] R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi, "Spanning trees short or small," in *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 546-555 (1994). Journal version to appear in *SIAM Journal on Discrete Mathematics.*

[SC+97] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian, "Buy-at-Bulk Network Design: Approximating the single-sink edge installation problem," *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA '97)*, 619-628 (1997).

[Wa87] A. Warburton, "Approximation of Pareto optima in multiple-objective, shortest path problems," *Operations Research*, vol. 35, pp. 70-79 (1987).

[ZP+94] Q. Zhu, M. Parsa, and W.W.M. Dai, "An iterative approach for delay-bounded minimum Steiner tree construction," *Technical Report UCSC-CRL-94-39*, UC Santa Cruz (1994).

Index