

Similarly,

$$v(Y \cup \{\ell\}) - v(Y) = \sum_{j \in P} \max \left\{ 0, \left(v_{\ell j} - \max_{i \in Y} v_{ij} \right) \right\}. \quad (2.7)$$

Now since $X \subseteq Y$ for a given $j \in P$, $\max_{i \in Y} v_{ij} \geq \max_{i \in X} v_{ij}$, so that

$$\max \left\{ 0, \left(v_{\ell j} - \max_{i \in Y} v_{ij} \right) \right\} \leq \max \left\{ 0, \left(v_{\ell j} - \max_{i \in X} v_{ij} \right) \right\}.$$

By summing this inequality over all $j \in P$ and using the equalities (2.6) and (2.7), we obtain the desired result. \square

The property of the value function v that we have just proved is one that plays a central role in a number of algorithmic settings, and is often called *submodularity*, though the usual definition of this property is somewhat different (see Exercise 2.10). This definition captures the intuitive property of decreasing marginal benefits: as the set includes more elements, the marginal value of adding a new element decreases.

Finally, we prove Lemma 2.15.

Proof of Lemma 2.15. Let $O - S = \{i_1, \dots, i_p\}$. Note that since $|O - S| \leq |O| \leq k$, then $p \leq k$. Since adding more bank accounts can only increase the overall value of the solution, we have that

$$v(O) \leq v(O \cup S),$$

and a simple rewriting gives

$$v(O \cup S) = v(S) + \sum_{j=1}^p [v(S \cup \{i_1, \dots, i_j\}) - v(S \cup \{i_1, \dots, i_{j-1}\})].$$

By applying Lemma 2.17, we can upper bound the right-hand side by

$$v(S) + \sum_{j=1}^p [v(S \cup \{i_j\}) - v(S)].$$

Since the algorithm chooses $i \in B$ to maximize $v(S \cup \{i\}) - v(S)$, we have that for any j , $v(S \cup \{i\}) - v(S) \geq v(S \cup \{i_j\}) - v(S)$. We can use this bound to see that

$$v(O) \leq v(O \cup S) \leq v(S) + p[v(S \cup \{i\}) - v(S)] \leq v(S) + k[v(S \cup \{i\}) - v(S)].$$

This inequality can be rewritten to yield the inequality of the lemma, and this completes the proof. \square

This greedy approximation algorithm and its analysis can be extended to similar problems in which the objective function $v(S)$ is given by a set of items S , and is monotone and submodular. We leave the definition of these terms and the proofs of the extensions to Exercise 2.10.

2.6 Finding minimum-degree spanning trees

We now turn to a local search algorithm for the problem of minimizing the maximum degree of a spanning tree. The problem we consider is the following: given a graph $G = (V, E)$ we wish to find a spanning tree T of G so as to minimize the maximum degree of nodes in T . We

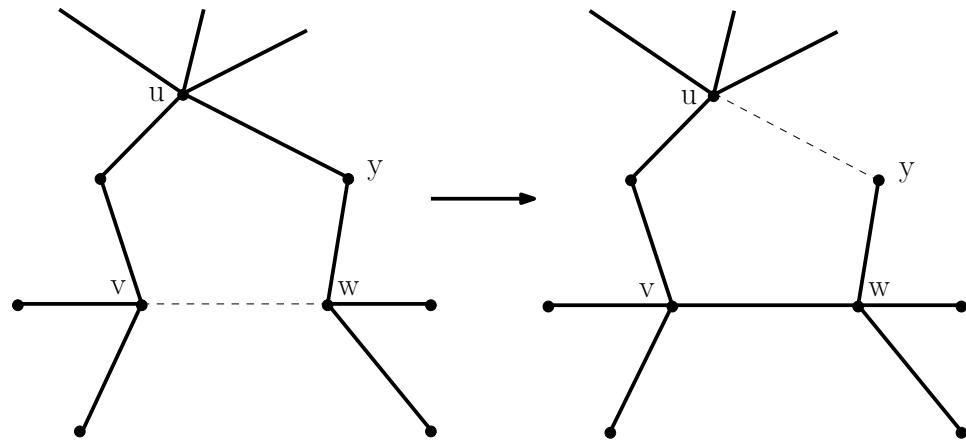


Figure 2.5: Illustration of a local move for minimizing the maximum degree of a spanning tree. The bold solid lines are in the tree, and the dashed lines are graph edges not in the tree.

will call this the *minimum-degree spanning tree problem*. This problem is NP-hard. A special type of spanning tree of a graph is a path that visits all nodes of the graph; this is called a *Hamiltonian path*. A spanning tree has maximum degree two if and only if it is a Hamiltonian path. Furthermore, deciding if a graph G has a Hamiltonian path is NP-complete. Thus, we have the following theorem.

Theorem 2.18: *It is NP-complete to decide whether or not a given graph has a minimum-degree spanning tree of maximum degree two.*

For a given graph G , let T^* be the spanning tree that minimizes the maximum degree, and let OPT be the maximum degree of T^* . We will give a polynomial-time local search algorithm that finds a tree T with maximum degree at most $2\text{OPT} + \lceil \log_2 n \rceil$, where $n = |V|$ is the number of vertices in the graph. To simplify notation, throughout this section we will let $\ell = \lceil \log_2 n \rceil$.

The local search algorithm starts with an arbitrary spanning tree T . We will give a local move to change T into another spanning tree in which the degree of some vertex has been reduced. Let $d_T(u)$ be the degree of u in T . The local move picks a vertex u and tries to reduce its degree by looking at all edges (v, w) that are not in T but if added to T create a cycle C containing u . Suppose $\max(d_T(v), d_T(w)) \leq d_T(u) - 2$. For example, consider the graph in Figure 2.5, in which the edges of the tree T are shown in bold. In this case, the degree of node u is 5, but those of v and w are 3. Let T' be the result of adding (v, w) and removing an edge from C incident to u . In the example, if we delete edge (u, y) , then the degrees of u , v , and w will all be 4 after the move. The conditions ensure that this move provides improvement in general; the degree of u is reduced by one in T' (that is, $d_{T'}(u) = d_T(u) - 1$) and the degrees of v and w in T' are not greater than the reduced degree of u ; that is, $\max(d_{T'}(v), d_{T'}(w)) \leq d_{T'}(u)$.

The local search algorithm carries out local moves on nodes that have high degree. It makes sense for us to carry out a local move to reduce the degree of any node, since it is possible that if we reduce the degree of a low-degree node, it may make possible another local move that reduces the degree of a high-degree node. However, we do not know how to show that such an algorithm terminates in polynomial time. To get a polynomial-time algorithm, we apply the local moves only to nodes whose degree is relatively high. Let $\Delta(T)$ be the maximum degree of T ; that is, $\Delta(T) = \max_{u \in V} d_T(u)$. The algorithm picks a node in T that has degree at least

$\Delta(T) - \ell$ and attempts to reduce its degree using the local move. If there is no move that can reduce the degree of any node having degree between $\Delta(T) - \ell$ and $\Delta(T)$, then the algorithm stops. We say that the algorithm has found a *locally optimal* tree. By applying local moves only to nodes whose degree is between $\Delta(T) - \ell$ and $\Delta(T)$, we will be able to show that the algorithm runs in polynomial time.

We now need to prove two things. First, we need to show that any locally optimal tree has maximum degree at most $2\text{OPT} + \ell$. Second, we need to show that we can find a locally optimal tree in polynomial time. For most approximation algorithms, the proof that the algorithm runs in polynomial time is relatively straightforward, but this is often not the case for local search algorithms. In fact, we usually need to restrict the set of local moves in order to prove that the algorithm converges to a locally optimal solution in polynomial time. Here we do this by restricting the local moves to apply only to nodes with high degree.

Theorem 2.19: *Let T be a locally optimal tree. Then $\Delta(T) \leq 2\text{OPT} + \ell$, where $\ell = \lceil \log_2 n \rceil$.*

Proof. We first explain how we will obtain a lower bound on OPT . Suppose that we remove k edges of the spanning tree. This breaks the tree into $k + 1$ different connected components. Suppose we also find a set of nodes S such that each edge in G connecting two of the $k + 1$ connected components is incident on a node in S . For example, consider the graph in Figure 2.6 that shows the connected components remaining after the bold edges are deleted, along with an appropriate choice of the set S . Observe that any spanning tree of the graph must have at least k edges with endpoints in different components. Thus, the average degree of nodes in S is at least $k/|S|$ for any spanning tree, and $\text{OPT} \geq k/|S|$.

Now we show how to find the set of edges to remove and the set of nodes S so that we can apply this lower bound. Let S_i be the nodes of degree at least i in the locally optimal tree T . We claim that for each S_i , where $i \geq \Delta(T) - \ell + 1$, there are at least $(i - 1)|S_i| + 1$ distinct edges of T incident on the nodes of S_i , and after removing these edges, each edge that connects distinct connected components is incident on a node of S_{i-1} . Furthermore, we claim there exists an i such that $|S_{i-1}| \leq 2|S_i|$, so that the value of OPT implied by removing these edges with $S = S_{i-1}$ is

$$\text{OPT} \geq \frac{(i - 1)|S_i| + 1}{|S_{i-1}|} \geq \frac{(i - 1)|S_i| + 1}{2|S_i|} > (i - 1)/2 \geq (\Delta(T) - \ell)/2.$$

Rearranging terms proves the desired inequality.

We turn to the proofs of the claims. We first show that there must exist some $i \geq \Delta(T) - \ell + 1$ such that $|S_{i-1}| \leq 2|S_i|$. Suppose not. Then clearly $|S_{\Delta(T)-\ell}| > 2^\ell |S_{\Delta(T)}|$ or $|S_{\Delta(T)-\ell}| > n |S_{\Delta(T)}| \geq n$ since $|S_{\Delta(T)}| \geq 1$. This is a contradiction, since any S_i can have at most n nodes.

Now we show that there are at least $(i - 1)|S_i| + 1$ distinct edges of T incident on the nodes of S_i , and after removing these edges, any edge connecting different connected components is incident on a node of S_{i-1} . Figure 2.6 gives an example of this construction for $i = 4$. Each edge that connects distinct connected components after removing the edges of T incident on nodes of S_i either must be one of the edges of T incident on S_i , or must close a cycle C in T containing some node in S_i . Because the tree is locally optimal, it must be the case that at least one of the endpoints has degree at least $i - 1$, and so is in S_{i-1} . In removing the edges in T incident on nodes in S_i , there are at least $i|S_i|$ edges incident on nodes in S_i , since each node has degree at least i . At most $|S_i| - 1$ such edges can join two nodes in S_i since T is a spanning tree. Thus, there are at least $i|S_i| - (|S_i| - 1)$ distinct edges of T incident on the nodes S_i , and this proves the claim. \square

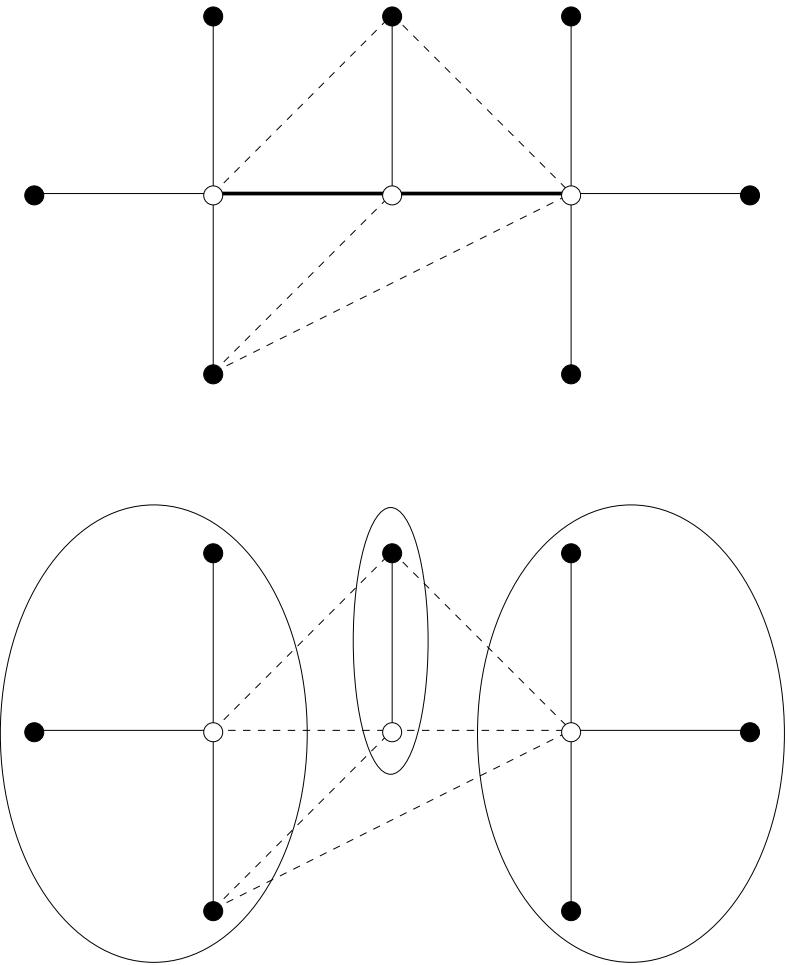


Figure 2.6: Illustration of lower bound on OPT. The vertices in S have white centers. If the bold edges are deleted, every potential edge for joining the resulting connected components has an endpoint in S .

Theorem 2.20: *The algorithm finds a locally optimal tree in polynomial time.*

Proof. To prove that the algorithm runs in polynomial time, we will use a *potential function* argument. The idea of such an argument is that the function captures the current state of the algorithm, and that we can determine upper and lower bounds on this function for any feasible solution, as well as a lower bound on the amount that the function must decrease after each move. In this way, we can bound the number of moves possible before the algorithm must terminate, and of course, the resulting tree must therefore be locally optimal.

For a tree T , we let the potential of T , $\Phi(T)$, be $\Phi(T) = \sum_{v \in V} 3^{d_T(v)}$. Note that $\Phi(T) \leq n3^{\Delta(T)}$, and so the initial potential is at most $n3^n$. On the other hand, the lowest possible potential is for a Hamiltonian path, which has potential $2 \cdot 3 + (n - 2)3^2 > n$. We will show that for each move, the potential function of the resulting tree is at most $1 - \frac{2}{27n^3}$ times the potential function previously.

After $\frac{27}{2}n^4 \ln 3$ local moves, the conditions above imply that the potential of the resulting tree is at most

$$\left(1 - \frac{2}{27n^3}\right)^{\frac{27}{2}n^4 \ln 3} \cdot (n3^n) \leq e^{-n \ln 3} \cdot (n3^n) = n,$$

using the fact that $1 - x \leq e^{-x}$. Since the potential of a tree is greater than n , after $O(n^4)$ local moves there must be no further local moves possible, and the tree must be locally optimal.

We must still prove the claimed potential reduction in each iteration. Suppose the algorithm reduces the degree of a vertex u from i to $i - 1$, where $i \geq \Delta(T) - \ell$, and adds an edge (v, w) . Then the increase in the potential function due to increasing the degree of v and w is at most $2 \cdot (3^{i-1} - 3^{i-2}) = 4 \cdot 3^{i-2}$, since the degree of v and w can be increased to at most $i - 1$. The decrease in the potential function due to decreasing the degree of u is $3^i - 3^{i-1} = 2 \cdot 3^{i-1}$. Observe that

$$3^\ell \leq 3 \cdot 3^{\log_2 n} \leq 3 \cdot 2^{2\log_2 n} = 3n^2.$$

Therefore, the overall decrease in the potential function is at least

$$2 \cdot 3^{i-1} - 4 \cdot 3^{i-2} = \frac{2}{9}3^i \geq \frac{2}{9}3^{\Delta(T)-\ell} \geq \frac{2}{27n^2}3^{\Delta(T)} \geq \frac{2}{27n^3}\Phi(T).$$

Thus, for the resulting tree T' we have that $\Phi(T') \leq (1 - \frac{2}{27n^3})\Phi(T)$. This completes the proof. \square

By slightly adjusting the parameters within the same proof outline, we can actually prove a stronger result. Given some constant $b > 1$, suppose we perform local changes on nodes of degree at least $\Delta(T) - \lceil \log_b n \rceil$. Then it is possible to show the following.

Corollary 2.21: *The local search algorithm runs in polynomial time and results in a spanning tree T such that $\Delta(T) \leq b \text{OPT} + \lceil \log_b n \rceil$.*

In Section 9.3, we will prove a still stronger result: we can give a polynomial-time algorithm that finds a spanning tree T with $\Delta(T) \leq \text{OPT} + 1$. Given that it is NP-hard to determine whether a spanning tree has degree exactly OPT , this is clearly the best possible result that can be obtained. In the next section, we give another result of this type for the edge coloring problem. In Section 11.2, we will show that there are interesting extensions of these results to the case of spanning trees with costs on the edges.

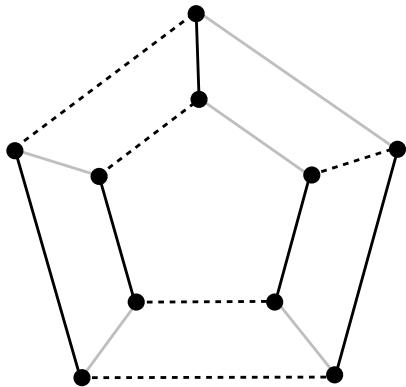


Figure 2.7: A graph with a 3-edge-coloring.

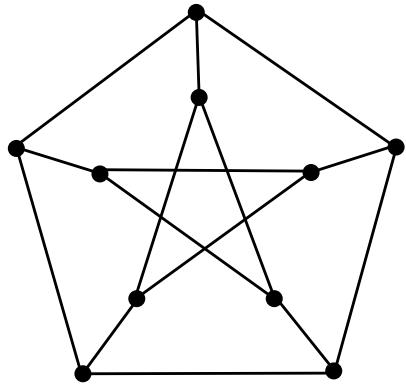


Figure 2.8: The Petersen graph. This graph is not 3-edge-colorable.

2.7 Edge coloring

To conclude this chapter, we give an algorithm that has the elements of both a greedy algorithm and a local search algorithm: it attempts to make progress in a greedy way, but when blocked it makes local changes until progress can be made again.

The algorithm is for the problem of finding an *edge coloring* of a graph. An undirected graph is *k-edge-colorable* if each edge can be assigned exactly one of k colors in such a way that no two edges with the same color share an endpoint. We call the assignment of colors to edges a *k-edge-coloring*. For example, Figure 2.7 shows a graph with a 3-edge-coloring. An analogous notion of *vertex coloring* will be discussed in Sections 5.12, 6.5, and 13.2.

For a given graph, we would like to obtain a k -edge-coloring with k as small as possible. Let Δ be the maximum degree of a vertex in the given graph. Clearly, we cannot hope to find a k -edge-coloring with $k < \Delta$, since at least Δ different colors must be incident to any vertex of maximum degree. Note that this shows that the coloring given in Figure 2.7 is optimal. On the other hand, consider the example in Figure 2.8, which is called the *Petersen graph*; it is not too hard to show that this graph is not 3-edge-colorable, and yet it is easy to color it with four colors. Furthermore, the following has been shown.

Theorem 2.22: *For graphs with $\Delta = 3$, it is NP-complete to decide whether the graph is*

sum $\sum_{j \in N} c_{\sigma^*(j)j} = C^*$. However, the same is true for the second term, the double summation is merely the sum over all possible locations j , and so the second terms contribute a total of $-\sum_{j \in N} c_{\sigma(j)j} = -C$. Now consider the second summation in (9.11). We can upper bound this expression by

$$\sum_{j:\sigma(j)=i} 2c_{\sigma^*(j)j}.$$

What is the effect of adding this summation over all crucial swaps? Each facility $i \in S$ occurs in 0, 1, or 2 crucial swaps; let n_i be the number of swaps in which each $i \in S$ occurs. Thus, we can upper bound the double summation as follows:

$$\sum_{i \in S} \sum_{j:\sigma(j)=i} 2n_i c_{\sigma^*(j)j} \leq 4 \sum_{i \in S} \sum_{j:\sigma(j)=i} c_{\sigma^*(j)j}.$$

But now we can apply the same reasoning as above; each location j is served in the current solution by a unique facility location $\sigma(j) \in S$, and hence the effect of the double summation is merely to sum over each j in N . That is, we have now deduced that this term is at most $4 \sum_{j \in N} c_{\sigma^*(j)j} = 4C^*$. Furthermore, we have concluded that $0 \leq 5C^* - C$, and hence $C \leq 5C^*$. \square

Finally, we observe that the same idea used in the previous section to obtain a polynomial-time algorithm can be applied here. The central ingredients to that proof are that if we restrict attention to moves (in this case swap moves) in which we improve the total cost by a factor of $1 - \delta$, then provided the analysis is based on a polynomial number of moves (each of which generates an inequality that the change in cost from this move is non-negative), we can set δ so that we can obtain a polynomial-time bound, while degrading the performance guarantee by an arbitrarily small constant.

Theorem 9.7: *For any constant $\rho > 5$, the local search algorithm for the k -median problem that uses bigger improving swaps yields a ρ -approximation algorithm.*

9.3 Minimum-degree spanning trees

In this section we return to the minimum-degree spanning tree problem introduced in Section 2.6. Recall that the problem is to find a spanning tree T in a graph $G = (V, E)$ that minimizes the maximum degree. If T^* is an optimal tree that minimizes the maximum degree, let OPT be the maximum degree of T^* . In Section 2.6, we showed that a particular local search algorithm finds a locally optimal tree of maximum degree at most $2\text{OPT} + \lceil \log_2 n \rceil$ in polynomial time. In this section, we will show that another variation on the local search algorithm finds a locally optimal tree of maximum degree at most $\text{OPT} + 1$ in polynomial time. As we discussed in Section 2.6, since it is NP-hard to minimize the maximum degree of a spanning tree, this is the best result possible unless $P = NP$.

As in the algorithm of Section 2.6, we start with an arbitrary spanning tree T and we will make local changes to it in order to decrease the degree of nodes in the tree. Let $d_T(u)$ be the degree of u in T . We pick a node u and attempt to reduce its degree by adding an edge (v, w) to T that creates a cycle C containing u , then removing an edge of the cycle C incident on u . Let $\Delta(T) = \max_{v \in V} d_T(v)$ be the maximum degree of the current tree T . We will make local changes in a way that is driven by the following lemma, which gives us a condition under which the current tree T has $\Delta(T) \leq \text{OPT} + 1$.

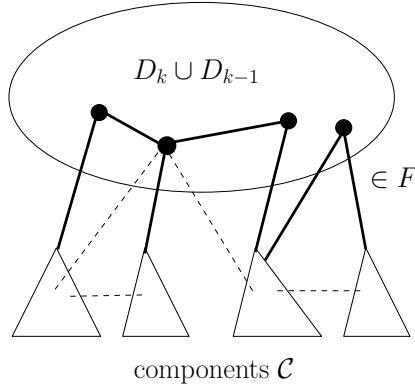


Figure 9.4: Illustration of the terms used in the statement of Lemma 9.8. The edges in F are shown in bold. Some edges in G that are not in the tree T are shown as dotted lines; note that they are not all incident on nodes in $D_k \cup D_{k-1}$. The components \mathcal{C} are the components of the tree T remaining after the edges in F are removed.

Lemma 9.8: Let $k = \Delta(T)$, let D_k be any nonempty subset of nodes of tree T with degree k and let D_{k-1} be any subset of nodes of tree T with degree $k-1$. Let F be the edges of T incident on nodes in $D_k \cup D_{k-1}$, and let \mathcal{C} be the collection of $|F| + 1$ connected components formed by removing the edges of F from T . If each edge of graph G that connects two different components in \mathcal{C} has at least one endpoint in $D_k \cup D_{k-1}$, then $\Delta(T) \leq \text{OPT} + 1$.

Proof. See Figure 9.4 for an illustration of the terms. We use the same idea as in the proof of Theorem 2.19 to obtain a lower bound on OPT. Since any spanning tree in G will need $|F|$ edges of G to connect the components in \mathcal{C} , the average degree of the nodes in $D_k \cup D_{k-1}$ in any spanning tree is at least $|F|/|D_k \cup D_{k-1}|$. Thus $\text{OPT} \geq |F|/|D_k \cup D_{k-1}|$.

We now bound $|F|$ in order to prove the lemma. The sum of the degrees of the nodes in D_k and D_{k-1} must be $|D_k|k + |D_{k-1}|(k-1)$. However, this sum of degrees may double count some edges of F which have both endpoints in $D_k \cup D_{k-1}$. Because T is acyclic, there can be at most $|D_k| + |D_{k-1}| - 1$ such edges. Hence, $|F| \geq |D_k|k + |D_{k-1}|(k-1) - (|D_k| + |D_{k-1}| - 1)$. Thus

$$\begin{aligned} \text{OPT} &\geq \left\lceil \frac{|D_k|k + |D_{k-1}|(k-1) - (|D_k| + |D_{k-1}| - 1)}{|D_k| + |D_{k-1}|} \right\rceil \\ &\geq \left\lceil k - 1 - \frac{|D_{k-1}| - 1}{|D_k| + |D_{k-1}|} \right\rceil \\ &\geq k - 1, \end{aligned}$$

implying that $k = \Delta(T) \leq \text{OPT} + 1$. □

The goal of the local search algorithm is to continue to reduce the degree of the nodes of degree $\Delta(T)$ while trying to attain the conditions of Lemma 9.8. The algorithm works in a sequence of phases, with each phase divided into subphases. At the beginning of the phase, for the current tree T , let $k = \Delta(T)$. At the beginning of a subphase, we let D_k be all the degree k vertices in T , let D_{k-1} be all the degree $k-1$ vertices in T , let F be the edges of T incident to $D_k \cup D_{k-1}$, and let \mathcal{C} be the components of T formed if F is removed from T . The goal of each phase is to make local moves to remove all nodes of degree k from the tree;

the goal of each subphase is to make local moves to remove a single vertex of degree k . In the process of executing a subphase, we discover nodes of degree $k - 1$ in D_{k-1} for which we can make a local move to reduce their degree. We do not yet execute these moves, but we mark the nodes as *reducible* via the particular local move, remove them from D_{k-1} , and update F and \mathcal{C} accordingly by removing edges from F and merging components in \mathcal{C} . The algorithm is summarized in Algorithm 9.1, and we now discuss its details.

In a subphase, we consider all edges of G that connect any two components of \mathcal{C} . If all such edges have an endpoint in $D_k \cup D_{k-1}$, we meet the condition of Lemma 9.8, and the algorithm terminates with a tree T such that $\Delta(T) \leq \text{OPT} + 1$. If there is some such edge (v, w) without an endpoint in $D_k \cup D_{k-1}$, then consider the cycle formed by adding (v, w) to T . Because (v, w) connects two different components of \mathcal{C} , it must be the case that the cycle includes some node $u \in D_k \cup D_{k-1}$. Note that if we desire, we can make a local move to reduce the degree of u by adding (v, w) to the tree T and removing a tree edge incident to u . If the cycle contains nodes of D_{k-1} only, then we do not yet make the local move, but we make a note that for any of the nodes in D_{k-1} on the cycle, we could do so if necessary. We label these nodes in D_{k-1} on the cycle as reducible via the edge (v, w) , then remove them from D_{k-1} , then update F to be the tree edges incident on the current set of $D_k \cup D_{k-1}$, and update \mathcal{C} accordingly. Note that since we removed all nodes on the cycle from D_{k-1} , this only removes edges from F and merges components in \mathcal{C} ; in particular, the two components connected by (v, w) will be merged in the updated \mathcal{C} . If, on the other hand, the cycle includes a node of $u \in D_k$, we go ahead and reduce its degree by adding (v, w) to the tree and removing an edge incident on u . Decreasing the degree of u decreases the number of nodes of degree k in the tree, but we want to ensure that we do not increase the degree of nodes v and w to k . Note that this could only happen if the degree of v or w in the tree is $k - 1$ and at some previous point in the subphase the node was removed from D_{k-1} and labelled reducible. In this case, we carry out the local move that allows us to reduce the degree of reducible node to $k - 2$, then add (v, w) to the tree and remove an edge incident to u from the tree. It is possible that carrying out the local move to reduce the degree of the reducible node to $k - 2$ might cause a cascade of local moves; for instance, if v has degree $k - 1$, and we can reduce its degree by adding edge (x, y) , potentially x also has degree $k - 1$ and is reducible, and so on; we will show that it is possible to carry out all these moves, and reduce the degree of u to $k - 1$ without creating any new nodes of degree k . We say that we are able to *propagate* the local move for u . Once we reduce the degree of u from k to $k - 1$, we start a new subphase. If there are no further nodes of degree k , we start a new phase.

We can now prove that the algorithm is correct and runs in polynomial time.

Theorem 9.9: *Algorithm 9.1 returns a spanning tree T with $\Delta(T) \leq \text{OPT} + 1$ in polynomial time.*

Proof. Because the algorithm terminates only when it meets the conditions of Lemma 9.8, it returns a tree T with $\Delta(T) \leq \text{OPT} + 1$ if it does indeed terminate. We claim that in each subphase, we can propagate local moves to reduce the degree of a reducible node in a component in \mathcal{C} without creating any new nodes of degree k . Then either the algorithm terminates or in each subphase, we find some node u of degree k whose degree can be reduced to $k - 1$ by making a local move with an edge (v, w) . Since v and w must be in separate components of \mathcal{C} , either their degree is less than $k - 1$ or they have degree $k - 1$ and are reducible, and by the claim we can propagate local moves to reduce their degree. Thus we can reduce the degree of u from k to $k - 1$ without creating any new nodes of degree k , and so each phase eliminates all nodes of degree k . Since we cannot have a feasible spanning tree with $\Delta(T) = 1$, the algorithm must

```

Let  $T$  be an arbitrary spanning tree of  $G = (V, E)$ 
while true do
     $k \leftarrow \Delta(T)$  // Start a new phase
    while there are nodes of degree  $k$  in  $T$  do // Start a new subphase
         $D_k \leftarrow$  all nodes of degree  $k$  in  $T$ 
         $D_{k-1} \leftarrow$  all nodes of degree  $k - 1$  in  $T$ 
         $F \leftarrow$  all edges of  $T$  incident on nodes in  $D_k \cup D_{k-1}$ 
         $\mathcal{C} \leftarrow$  all components formed by removing  $F$  from  $T$ 
        All nodes  $u \in D_{k-1}$  are unlabelled
        if for all  $(v, w) \in E$  connecting two components in  $\mathcal{C}$ : either  $v$  or  $w$  in  $D_k \cup D_{k-1}$  then
            return  $T$ 
        for all  $(v, w) \in E$  connecting two components in  $\mathcal{C}$ :  $v, w \notin D_k \cup D_{k-1}$  do
            Let  $C$  be cycle created by adding  $(v, w)$  to  $T$ 
            if  $C \cap D_k = \emptyset$  then
                Mark all  $u \in C \cap D_{k-1}$  reducible via  $(v, w)$ 
                Remove  $C \cap D_{k-1}$  from  $D_{k-1}$ 
                Update  $F$  and  $\mathcal{C}$ 
            else
                if  $u \in C \cap D_k$  then
                    if  $v$  or  $w$  marked reducible then
                        Reduce degree of  $v$  and/or  $w$  via local move and propagate local
                        moves if necessary
                    Reduce degree of  $u$  via local move with  $(v, w)$ 
                    Break for loop // Start new subphase
    
```

Algorithm 9.1: Local search algorithm for the minimum-degree spanning tree problem.

eventually terminate. Clearly the algorithm runs in polynomial time.

We now prove the claim by showing that at any iteration of the subphase, we can propagate local moves to reduce the degree of a reducible node in a component in \mathcal{C} . We prove this by induction on the number of iterations in the subphase. In the first iteration, no nodes are marked reducible and the claim is trivially true. Now suppose we are at some iteration $i > 1$ of the subphase, and let u be labelled reducible in this iteration. The node u is currently reducible because we have a local move with a non-tree edge (v, w) that will reduce the degree of u from $k - 1$ to $k - 2$; furthermore, the components in \mathcal{C} containing v and w are disjoint in the current iteration. If v is reducible, it was labelled such in an iteration $j < i$, and by induction we can carry out local moves to ensure that its degree is at most $k - 2$. The same is true for w , and by induction we can carry out the local moves for both v and w because they are in separate components in \mathcal{C} . In the next iteration the components containing v and w are merged into a single component that also contains u . Since the only changes that can happen to components in \mathcal{C} during a subphase is that components are merged, u , v , and w remain in the same component of \mathcal{C} through the rest of the subphase, and the local moves of adding (v, w) and reducing the degree of u remain available. \square

In Section 11.2, we will consider a version of the problem in which there are costs on the edges and specified bounds on the degrees of the nodes. If a tree with the given degree bounds

exists, we will show how to find a minimum-cost tree such that the degree bounds are only exceeded by one.

9.4 A greedy algorithm for the uncapacitated facility location problem

In this section, we give yet another approximation algorithm for the uncapacitated facility location problem. We will give a greedy algorithm for the problem, then use dual fitting to analyze it; this is similar to what we did in Theorem 1.12 for the set cover problem.

A very natural greedy algorithm is to repeatedly choose a facility and some clients to assign to that facility. We open the facility, assign the clients to the facility, remove the facility and clients from further consideration, and repeat. For a greedy algorithm, we would somehow like to find a facility and set of clients that minimizes total cost for the amount of progress made. To do this, we use the same criterion we used for the greedy set cover algorithm in Section 1.6: we maximize the “bang for the buck” by minimizing the ratio of the total cost per client assigned. To be more precise, let X be the set of facilities opened so far, and let S be the set of clients that are not connected to facilities in X so far. We pick some $i \in F - X$ and $Y \subseteq S$ that minimizes the ratio

$$\frac{f_i + \sum_{j \in Y} c_{ij}}{|Y|}.$$

We then add i to X , and remove Y from S , and repeat. Note that to find the appropriate set $Y \subseteq S$, for any given facility i , we can sort the clients in S by their distance from i , from nearest to farthest, and the set Y minimizing the ratio for i will be some prefix of this ordering.

We now add two simple improvements to this proposed algorithm. The first is that once we select facility i , rather than removing it from the set of facilities that can be chosen in the future, we instead allow it to be chosen again and set its facility cost to zero. The intuition here is that in future iterations it may be more cost-effective to assign some clients to i rather than opening another facility to serve them, and since i has already been opened, we should treat its facility cost as zero. The second idea is that rather than assigning clients to a facility and fixing that assignment from then on, we consider switching assignments to other facilities we open later on. We include the savings gained by switching assignments when trying to choose the facility to open. Let $c(j, X) = \min_{i \in X} c_{ij}$, and let $(a)_+ = \max(a, 0)$. Then if we have already assigned the clients in $D - S$ to some facilities in X , and we are considering opening facility i , we can decrease assignment costs for all clients $j \notin S$ such that $c(j, X) > c_{ij}$ by reassigning them from X to i . The savings achieved is $\sum_{j \notin S} (c(j, X) - c_{ij})_+$. Thus in every step we pick some $i \in F$ and $Y \subseteq S$ that minimizes the ratio

$$\frac{f_i - \sum_{j \notin S} (c(j, X) - c_{ij})_+ + \sum_{j \in Y} c_{ij}}{|Y|}.$$

Our revised greedy algorithm is now given in Algorithm 9.2.

To analyze this algorithm, we will use a dual fitting analysis: we will construct an infeasible solution to the dual of the linear programming relaxation such that the cost of the primal solution is equal to the value of the dual objective. Then we show that scaling the dual solution by a factor of 2 makes it feasible. This implies that the cost of the primal solution is at most twice the value of a solution to the dual of the linear programming relaxation, which implies that the algorithm is an 2-approximation algorithm.

11.2 Minimum-cost bounded-degree spanning trees

In this section, we consider the weighted case of a problem we considered in Sections 2.6 and 9.3. In those sections, we considered the problem of finding a spanning tree of a graph such that we minimized the maximum degree of the tree. Here, we consider the problem of finding a spanning tree of minimum cost such that the degree of node v is no more than some specified bound. We refer to this problem as the *minimum-cost bounded-degree spanning tree problem*. More formally, we are given as input an undirected graph $G = (V, E)$, costs $c_e \geq 0$ for all $e \in E$, a set $W \subseteq V$, and integer bounds $b_v \geq 1$ for all $v \in W$. Let OPT be the cost of the minimum spanning tree such that the degree of each node $v \in W$ is no more than b_v (if such a tree exists). In the first part of this section, we give an algorithm that finds a spanning tree of cost at most OPT such that each node $v \in W$ has degree at most $b_v + 2$. In the latter part of this section, we show that we can find a spanning tree of cost at most OPT such that each node $v \in W$ has degree at most $b_v + 1$. As we argued at the end of Section 2.6 for unweighted spanning trees, no better result is possible unless $P = NP$.

We begin by giving an integer programming formulation of the problem for a graph $G = (V, E)$. Given a vertex set $S \subseteq V$, let $E(S)$ be the subset of edges in E that have both endpoints in S , and let $\delta(S)$ be the subset of edges that have exactly one endpoint in S . We will denote $\delta(\{v\})$ by $\delta(v)$. We let $x_e \in \{0, 1\}$ indicate whether an edge e is in the spanning tree or not. Every spanning tree has exactly $|V| - 1$ edges, so we have

$$\sum_{e \in E} x_e = |V| - 1.$$

Furthermore, since for any set $S \subseteq V$ with $|S| \geq 2$, a spanning tree does not have a cycle in $E(S)$, we have

$$\sum_{e \in E(S)} x_e \leq |S| - 1.$$

Finally, we want the spanning tree to respect the degree bounds for all $v \in W$, so that we have

$$\sum_{e \in \delta(v)} x_e \leq b_v.$$

Relaxing the integrality constraints to $x_e \geq 0$ gives us the following linear programming relaxation of the problem:

$$\text{minimize } \sum_{e \in E} c_e x_e \tag{11.12}$$

$$\text{subject to } \sum_{e \in E} x_e = |V| - 1, \tag{11.13}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \quad \forall S \subseteq V, |S| \geq 2, \tag{11.14}$$

$$\sum_{e \in \delta(v)} x_e \leq b_v, \quad \forall v \in W, \tag{11.15}$$

$$x_e \geq 0, \quad \forall e \in E. \tag{11.16}$$

We can use the ellipsoid method introduced in Section 4.3 to solve the linear program by giving a polynomial-time separation oracle. It is easy to check that the constraints (11.13),

(11.15), and (11.16) are satisfied. To check that the constraints (11.14) are satisfied, we need to set up a sequence of maximum flow problems. We first explain the general max-flow construction, and then we modify it to check the constraints (11.14). We create a new graph G' where we add source and sink vertices s and t , edges (s, v) from the source to every vertex $v \in V$, and edges (v, t) from every vertex $v \in V$. The capacity of every edge e from G is $\frac{1}{2}x_e$, the capacity of every edge (s, v) is $\frac{1}{2}\sum_{e \in \delta(v)} x_e$, and the capacity of every edge (v, t) is 1. Then consider the capacity of any s - t cut $S \cup \{s\}$ for $S \subseteq V$. It contains edges (v, t) for all $v \in S$, all $e \in \delta(S) \cap E$ and (s, v) for all $v \notin S$. So its total capacity is

$$|S| + \frac{1}{2} \sum_{e \in \delta(S)} x_e + \frac{1}{2} \sum_{v \notin S} \sum_{e \in \delta(v)} x_e = |S| + \sum_{e \in \delta(S)} x_e + \sum_{e \in E(V-S)} x_e,$$

since adding up $\frac{1}{2}x_e$ for each $e \in \delta(v)$ where $v \notin S$ gives x_e for all edges e with both endpoints not in S plus $\frac{1}{2}x_e$ for all edges e with exactly one endpoint not in S . Using the fact that $\sum_{e \in E} x_e = |V| - 1$, this is equal to

$$|S| + |V| - 1 - \sum_{e \in E(S)} x_e = |V| + (|S| - 1) - \sum_{e \in E(S)} x_e.$$

Thus the capacity of this cut is at least $|V|$ if and only if $\sum_{e \in E(S)} x_e \leq |S| - 1$. We want to make sure $|S| \geq 2$, so for each pair $x, y \in V$, we construct the max-flow instance as above, but alter the capacity of the edges (s, x) and (s, y) to be infinite. This ensures that any minimum s - t cut $S \cup \{s\}$ has $x, y \in S$. Then by the reasoning above, the value of the maximum flow for this instance will be at least $|V|$ if and only if constraints (11.14) are satisfied for all $S \supseteq \{x, y\}$. If the flow is at least $|V|$ for all pairs $x, y \in V$, then all constraints (11.14) are satisfied. If the flow value is less than $|V|$, then the corresponding minimum s - t cut $S \cup \{s\}$ will give a violated constraint.

We assume from here on that there is a feasible solution to this linear program. If there is no feasible solution, then obviously there is no tree in the graph G that has the given degree bounds.

As a warmup to the algorithm and its approach, we first show that if we have no degree bounds (that is, $W = \emptyset$), then we can find a spanning tree of cost no more than the value of the linear program (11.12); this gives us a minimum spanning tree. Given an LP solution x , define $E(x)$ to be the support of x ; that is, $E(x) = \{e \in E : x_e > 0\}$. The algorithm for finding a minimum spanning tree depends on the following lemma. We defer the proof of the lemma for the moment.

Lemma 11.3: *For any basic feasible solution x to the linear program (11.12) with $W = \emptyset$, there is some $v \in V$ such that there is at most one edge of $E(x)$ incident on v .*

Our algorithm then works as follows. We maintain a set of edges F for our solution, which is initially empty. While there is more than one vertex in the current graph, we solve the linear program (11.12) for the current graph $G = (V, E)$, and obtain a basic optimal solution x . We remove from the edge set E all edges e for which $x_e = 0$. By Lemma 11.3, there exists some vertex $v \in V$ such that there is at most one edge of $E(x)$ incident on it; suppose the edge is (u, v) . Then we add (u, v) to our solution set F , remove v and (u, v) from the graph, and repeat. Intuitively, each iteration finds a leaf v of the spanning tree, then recursively finds the rest of it. The algorithm is summarized in Algorithm 11.1.

Theorem 11.4: *Algorithm 11.1 yields a spanning tree of cost no more than the value of the linear program (11.12).*

```

 $F \leftarrow \emptyset$ 
while  $|V| > 1$  do
    Solve LP (11.12) on  $(V, E)$ , get basic optimal solution  $x$ 
     $E \leftarrow E(x)$ 
    Find  $v \in V$  such that there is one edge  $(u, v)$  in  $E(x)$  incident on  $v$ 
     $F \leftarrow F \cup \{(u, v)\}$ 
     $V \leftarrow V - \{v\}$ 
     $E \leftarrow E - \{(u, v)\}$ 
return  $F$ 

```

Algorithm 11.1: Deterministic rounding algorithm for finding a minimum-cost spanning tree.

Proof. We begin by showing that if the edge e^* is chosen to add to F in some iteration of the algorithm, then $x_{e^*} = 1$ in that iteration. To prove this, we first claim that $\sum_{e \in \delta(w)} x_e \geq 1$ for any vertex $w \in V$. Then for the edge $e^* = (u, v)$ chosen by the algorithm, we know that there is one edge in $E(x)$ incident on v , and therefore $x_{e^*} \geq 1$. To see that $\sum_{e \in \delta(v)} x_e \geq 1$, note that for $S = V - \{v\}$, the LP constraint (11.14) enforces that $\sum_{e \in E(S)} x_e \leq |S| - 1 = (|V| - 1) - 1 = |V| - 2$. But $\sum_{e \in E} x_e = |V| - 1$ by constraint (11.13) and also $\sum_{e \in \delta(v)} x_e = \sum_{e \in E} x_e - \sum_{e \in E(S)} x_e$. Therefore $\sum_{e \in \delta(v)} x_e \geq (|V| - 1) - (|V| - 2) = 1$. Now consider the constraint (11.14) for $S = \{u, v\}$. This enforces $x_{e^*} \leq 1$, so it must be the case that $x_{e^*} = 1$.

We now prove by induction on the size of the graph that the algorithm produces a spanning tree of cost no more than the value of the linear program. In the base case, we have a graph with two vertices, and the algorithm returns a single edge e . By the above, we have $x_e = 1$, so that the value of the LP is at least $c_e x_e = c_e$, and the statement holds. Now suppose that the statement holds for every graph with at most $k > 2$ vertices, and our graph has $k + 1$ vertices. We solve the LP, get a solution x , and find a vertex v such that there is only one edge $e^* = (u, v)$ in $E(x)$ incident on v . Let $V' = V - \{v\}$, and $E' = E(x) - \{e^*\}$. By the inductive hypothesis, we will find a spanning tree F' on (V', E') of cost at most the value of the LP on (V', E') ; let x' be an optimal solution to this LP. Obviously $F' \cup \{e^*\}$ is a spanning tree of (V, E) , so that the algorithm returns a spanning tree. To show that its cost is at most $\sum_{e \in E} c_e x_e$, we shall show that x_e for $e \in E'$ is a feasible solution to the LP on (V', E') . It then follows that the cost of the spanning tree returned is

$$\sum_{e \in F'} c_e + c_{e^*} \leq \sum_{e \in E'} c_e x'_e + c_{e^*} x_{e^*} \leq \sum_{e \in E'} c_e x_e + c_{e^*} x_{e^*} = \sum_{e \in E} c_e x_e,$$

as desired.

We now must show that x_e for $e \in E'$ is a feasible solution to the LP on (V', E') . Since the LP constraints (11.14) for (V', E') are a subset of the constraints for (V, E) , x is feasible for them. Thus we only need to show that the first constraint holds, namely $\sum_{e \in E'} x_e = |V'| - 1 = |V| - 2$. This follows since the only edges in $E - E'$ are e^* , which has $x_{e^*} = 1$, and edges such that $x_e = 0$. Thus $\sum_{e \in E'} x_e = \sum_{e \in E} x_e - x_{e^*} = (|V| - 1) - 1 = |V'| - 1$, and x_e for $e \in E'$ is feasible for the LP on (V', E') . \square

Now we return to the minimum-cost bounded-degree spanning tree problem. Recall that we have a subset of vertices W such that we wish to find a tree with the degree of v at most b_v for all $v \in W$. We cannot hope for a result as strong as that of Lemma 11.3 for this problem, since we would be able to translate this into an algorithm to find an optimal tree. Instead, we will

```

 $F \leftarrow \emptyset$ 
while  $|V| > 1$  do
    Solve LP (11.12) on  $(V, E)$  and  $W$ , get basic optimal solution  $x$ 
     $E \leftarrow E(x)$ 
    if  $\exists v \in V$  such that there is one edge  $(u, v)$  in  $E(x)$  incident on  $v$  then
         $F \leftarrow F \cup \{(u, v)\}$ 
         $V \leftarrow V - \{v\}$ 
         $E \leftarrow E - \{(u, v)\}$ 
        if  $u \in W$  then
             $b_u \leftarrow b_u - 1$ 
        else
             $\exists v \in W$  such that there are at most three edges in  $E(x)$  incident on  $v$ 
             $W \leftarrow W - \{v\}$ 
    return  $F$ 

```

Algorithm 11.2: Deterministic rounding algorithm for finding a minimum-cost bounded-degree spanning tree.

be able to show the following. This will lead to a result in which degree bounds are exceeded by at most two.

Lemma 11.5: *For any basic feasible solution x to the linear program (11.12), either there is some $v \in V$ such that there is at most one edge of $E(x)$ incident on v , or there is some $v \in W$ such that there are at most three edges of $E(x)$ incident on v .*

Note that Lemma 11.3 is a special case of this lemma; the second possibility cannot occur when $W = \emptyset$. We again defer the proof of this lemma, and instead show how it leads to the desired algorithm. As before, we maintain a solution set F , which is initially empty. We solve the linear program for the current graph (V, E) and current bound set W , and obtain a basic optimal solution x . We remove all edges e with $x_e = 0$ from the edge set. If, as before, there is some $v \in V$ such that there is at most one edge (u, v) in $E(x)$ incident on v , we add (u, v) to F , remove v from V , and remove (u, v) from E . If $u \in W$, we also decrease b_u by one. We then iterate. If instead, there is some $v \in W$ such that there are at most three edges of $E(x)$ incident on v , we remove v from W and repeat. The algorithm is summarized in Algorithm 11.2.

We can now prove the following.

Theorem 11.6: *Algorithm 11.2 produces a spanning tree F such that the degree of v in F is at most $b_v + 2$ for $v \in W$, and such that the cost of F is at most the value of the linear program (11.12).*

Proof. In each step of the algorithm, we either add a spanning tree edge to F , or we remove a vertex from W . Thus the algorithm will terminate in at most $(n - 1) + n = 2n - 1$ iterations.

Observe that the proof that the algorithm returns a spanning tree whose cost is at most the value of the linear program is almost identical to that of Theorem 11.4. In that proof we considered the graph (V', E') resulting from adding an edge (u, v) to our solution F , then removing v and (u, v) from the graph. We showed that the LP solution x for the graph (V, E) is feasible for the new graph (V', E') when restricted to the edges in E' . Here we also need to consider the set of degree bounds W , and show that x is feasible for the new graph (V', E') and the new degree bounds after b_u has been decreased by one. If x was feasible for the constraints (11.15) before, then $\sum_{e \in \delta(u)} x_e \leq b_u$. Thus after edge $e = (u, v)$ with $x_e = 1$ is removed from

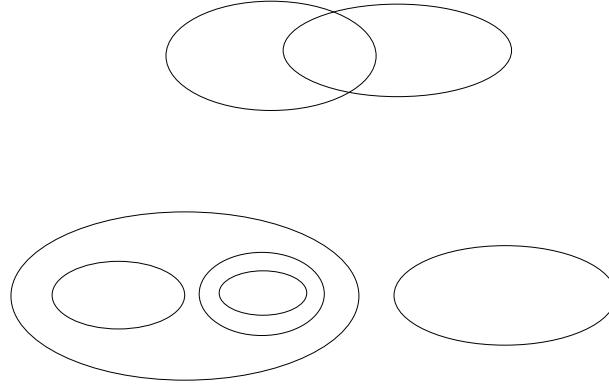


Figure 11.2: Top: two intersecting sets. Bottom: A laminar collection of sets.

the graph, the solution x restricted to the remaining edges in E' will be feasible for the same constraint on the new graph (V', E') with the degree bound $b_u - 1$.

Now consider any vertex v initially in W . We show by induction on the algorithm that the degree of v in the solution F is at most $b_v + 2$. In each iteration, one of three things can happen. First, we can choose an edge incident on v and decrease b_v by 1, in which case the statement follows by induction. Second, we can choose an edge incident on v and remove v from the graph; in this case, we must have had $b_v \geq 1$ in order to have a feasible solution to the LP, so the statement holds. Third, we can remove v from W . In this case, we must have $b_v \geq 1$ in order for there to be any edges in $E(x)$ incident on v ; yet there are at most 3 edges in $E(x)$ incident on v . Thus, in all future iterations we can add at most three edges incident on v , since all edges not in $E(x)$ are removed from the graph for all future iterations. Thus, v will have degree at most $b_v + 2$. \square

We can now turn to the proof of Lemma 11.5. In order to do this, we will need to introduce some definitions and notation. From here on, for the given solution x , we assume that all edges e such that $x_e = 0$ have been removed from the edge set; that is, $E = E(x)$.

Definition 11.7: For $x \in \mathbb{R}^{|E|}$ and a subset of edges F , we define $x(F) = \sum_{e \in F} x_e$.

Definition 11.8: For a solution x to LP (11.12), we say that a constraint (11.14) corresponding to a set $S \subseteq V$, $|S| \geq 2$, is tight if $x(E(S)) = |S| - 1$. A constraint (11.15) corresponding to a vertex $v \in W$ is tight if $x(\delta(v)) = b_v$.

We may also say that the set $S \subseteq V$ is tight if $x(E(S)) = |S| - 1$ or that the vertex v is tight if $x(\delta(v)) = b_v$.

Definition 11.9: We say two sets A and B are intersecting if $A \cap B$, $A - B$, and $B - A$ are all nonempty.

Definition 11.10: We say a collection of sets \mathcal{S} is laminar if no pair of sets $A, B \in \mathcal{S}$ are intersecting.

See Figure 11.2 for an example of intersecting sets and laminar set collections.

Definition 11.11: For a subset of edges $F \subseteq E$, the characteristic vector of F is $\chi_F \in \{0, 1\}^{|E|}$, where $\chi_F(e) = 1$ if $e \in F$ and 0 otherwise.

We are now able to state the following theorem, which we will need to prove Lemma 11.5.

Theorem 11.12: *For any basic feasible solution x to the linear program (11.12), there is a set $Z \subseteq W$ and a collection \mathcal{L} of subsets of vertices with the following properties:*

1. *For all $S \in \mathcal{L}$, $|S| \geq 2$ and S is tight, and for all $v \in Z$, v is tight.*
2. *The vectors $\chi_{E(S)}$ for $S \in \mathcal{L}$ and $\chi_{\delta(v)}$ for $v \in Z$ are linearly independent.*
3. $|\mathcal{L}| + |Z| = |E|$.
4. *The collection \mathcal{L} is laminar.*

We will defer the proof of this theorem for a moment, and will next show how Lemma 11.5 can be derived from the theorem. First, however, we observe that for any basic feasible solution x to the LP, there is a collection of sets \mathcal{S} and set $Y \subseteq W$ such that the first three properties hold for \mathcal{S} and Y ; the key statement of the theorem is the last property which states that for any basic feasible solution, there is a laminar collection of such sets. A basic solution is formed by taking $|E|$ linearly independent constraints from the linear program, setting them at equality, and solving the resulting linear system. This is precisely what the first two properties state. The third property states that the number of constraints set to equality is equal to the number of nonzero variables (recall that we have assumed that $E = E(x)$).

We first need the following short lemma.

Lemma 11.13: *Let \mathcal{L} be a laminar collection of subsets of V , where each $S \in \mathcal{L}$ has $|S| \geq 2$. Then $|\mathcal{L}| \leq |V| - 1$.*

Proof. We can prove this by induction on the size of $|V|$. For the base case $|V| = 2$, obviously \mathcal{L} can contain only one set of cardinality 2. For $|V| > 2$, pick a minimum cardinality set R in \mathcal{L} . Let V' be V with all but one vertex of R removed, and let \mathcal{L}' be the sets in \mathcal{L} restricted to the elements of V' , with the set R removed. Note that \mathcal{L}' fulfills the conditions of the lemma; it is still laminar, and any set must contain at least two elements. So $|\mathcal{L}'| \leq |V'| - 1$ by induction, and since $|\mathcal{L}'| = |\mathcal{L}| - 1$ and $|V'| \leq |V| - 1$, the lemma statement follows. \square

Now we recall the statement of Lemma 11.5, and give its proof.

Lemma 11.5: *For any basic feasible solution x to the linear program (11.12), either there is some $v \in V$ such that there is at most one edge of $E(x)$ incident on v , or there is some $v \in W$ such that there are at most three edges of $E(x)$ incident on v .*

Proof. We prove the statement by contradiction. If the statement is not true, then for every vertex $v \in V$, there are at least two edges of $E(x)$ incident on it, and for every $v \in W$, there are at least four edges of $E(x)$ incident on it. Then it must be the case that

$$|E(x)| \geq \frac{1}{2} (2(|V| - |W|) + 4|W|) = |V| + |W|.$$

However, by Theorem 11.12, we know that $|E(x)| = |\mathcal{L}| + |Z| \leq |\mathcal{L}| + |W|$ for laminar \mathcal{L} . Since each set $S \in \mathcal{L}$ has cardinality at least two, by Lemma 11.13, we get that $|E(x)| \leq |V| - 1 + |W|$, which is a contradiction. \square

Finally, we can turn to the proof of Theorem 11.12. The basic idea of the proof is simple, and has proven enormously useful in a number of different contexts. We start out with a collection of sets \mathcal{S} that may not be laminar, and as long as we have two intersecting sets $S, T \in \mathcal{S}$, we show that we can “uncross” them and replace them with two non-intersecting sets. A first step in this proof is to show the following.

Lemma 11.14: *If S and T are tight sets such that S and T are intersecting, then $S \cup T$ and $S \cap T$ are also tight sets. Furthermore,*

$$\chi_{E(S)} + \chi_{E(T)} = \chi_{E(S \cup T)} + \chi_{E(S \cap T)}.$$

Proof. We begin by showing that $x(E(S))$ is supermodular; namely,

$$x(E(S)) + x(E(T)) \leq x(E(S \cap T)) + x(E(S \cup T)).$$

This follows by a simple counting argument: any edge in $E(S)$ or in $E(T)$ is in $E(S \cup T)$, while any edge in both $E(S)$ and $E(T)$ appear both in $E(S \cap T)$ and $E(S \cup T)$. The right-hand side may be greater than the left-hand side since edges with one endpoint in $S - T$ and the other in $T - S$ appear in $E(S \cup T)$ but not in $E(S)$ or $E(T)$.

Since S and T are intersecting, $S \cap T \neq \emptyset$. Thus by the feasibility of x

$$(|S| - 1) + (|T| - 1) = (|S \cap T| - 1) + (|S \cup T| - 1) \geq x(E(S \cap T)) + x(E(S \cup T)).$$

By supermodularity,

$$x(E(S \cap T)) + x(E(S \cup T)) \geq x(E(S)) + x(E(T)).$$

Finally, since S and T are tight sets,

$$x(E(S)) + x(E(T)) = (|S| - 1) + (|T| - 1).$$

Thus all of these inequalities must be met with equality, and $S \cap T$ and $S \cup T$ are tight sets. Furthermore, as $x(E(S \cap T)) + x(E(S \cup T)) = x(E(S)) + x(E(T))$, it must be the case that $\chi_{E(S)} + \chi_{E(T)} = \chi_{E(S \cup T)} + \chi_{E(S \cap T)}$, since we have assumed that the only edges in E have $x_e > 0$. \square

Let \mathcal{T} be the collection of all tight sets for the LP solution x ; that is, $\mathcal{T} = \{S \subseteq V : x(E(S)) = |S| - 1, |S| \geq 2\}$. Let $\text{span}(\mathcal{T})$ be the span of the set of vectors $\{\chi_{E(S)} : S \in \mathcal{T}\}$. We now show that we can find a laminar collection of sets with span at least that of \mathcal{T} .

Lemma 11.15: *There exists a laminar collection of sets \mathcal{L} such that $\text{span}(\mathcal{L}) \supseteq \text{span}(\mathcal{T})$, where each $S \in \mathcal{L}$ is tight and has $|S| \geq 2$, and the vectors $\chi_{E(S)}$ for $S \in \mathcal{L}$ are linearly independent.*

Proof. Let \mathcal{L} be a laminar collection of sets such that each $S \in \mathcal{L}$ is tight and has $|S| \geq 2$, and the vectors $\chi_{E(S)}$ for $S \in \mathcal{L}$ are linearly independent. We assume that \mathcal{L} is the maximal such collection; that is, we cannot add any additional sets to \mathcal{L} such that all of these properties continue to hold. We will give a proof by contradiction that $\text{span}(\mathcal{L}) \supseteq \text{span}(\mathcal{T})$; assume otherwise. Then there must be a tight set S with $|S| \geq 2$ such that $\chi_{E(S)} \in \text{span}(\mathcal{T})$ and $\chi_{E(S)} \notin \text{span}(\mathcal{L})$; we choose an S such that there is no other such set intersecting fewer sets in \mathcal{L} . Note that such an S must be intersecting with at least one set in \mathcal{L} , otherwise \mathcal{L} is not maximal.

Now pick a set $T \in \mathcal{L}$ such that S and T intersect. By Lemma 11.14, $\chi_{E(S)} + \chi_{E(T)} = \chi_{E(S \cap T)} + \chi_{E(S \cup T)}$ and both $S \cap T$ and $S \cup T$ are tight. We will argue that it cannot be the case that both $\chi_{E(S \cap T)} \in \text{span}(\mathcal{L})$ and $\chi_{E(S \cup T)} \in \text{span}(\mathcal{L})$. We know that $T \in \mathcal{L}$ so that $\chi_{E(T)} \in \text{span}(\mathcal{L})$, and we know that $\chi_{E(S)} = \chi_{E(S \cap T)} + \chi_{E(S \cup T)} - \chi_{E(T)}$. Thus if both $\chi_{E(S \cup T)}$ and $\chi_{E(S \cap T)}$ are in $\text{span}(\mathcal{L})$, this implies $\chi_{E(S)} \in \text{span}(\mathcal{L})$, which contradicts our assumption that $\chi_{E(S)} \notin \text{span}(\mathcal{L})$. Hence at least one of $\chi_{E(S \cap T)}$ and $\chi_{E(S \cup T)}$ is not in $\text{span}(\mathcal{L})$.

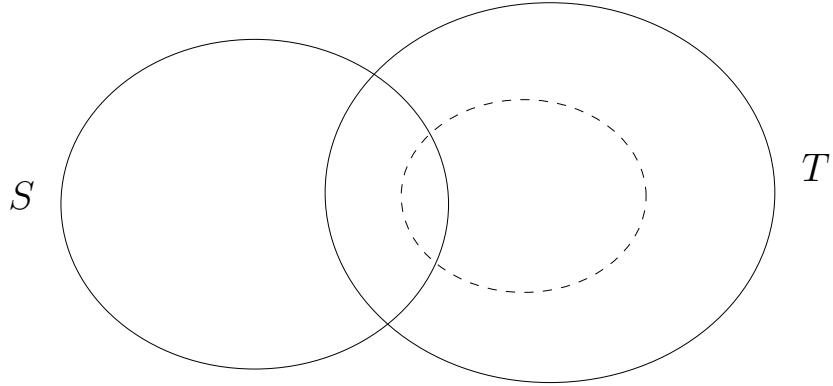


Figure 11.3: Proof of Lemma 11.15. The only sets from the laminar collection \mathcal{L} , with $T \in \mathcal{L}$, that can intersect $S \cap T$ must also intersect S .

Without loss of generality, suppose that $\chi_{E(S \cap T)} \notin \text{span}(\mathcal{L})$; note that this implies that $\chi_{E(S \cap T)} \neq 0$, so that $|S \cap T| \geq 2$. We claim that $S \cap T$ must intersect fewer sets in \mathcal{L} than S , and with this claim, we contradict the choice of S , since $S \cap T$ is tight, $|S \cap T| \geq 2$, and $\chi_{E(S \cap T)} \notin \text{span}(\mathcal{L})$. To prove the claim, we observe that any set in the laminar collection \mathcal{L} intersecting $S \cap T$ must also intersect S ; see Figure 11.3. However, S intersects T , while $S \cap T$ does not intersect T , so $S \cap T$ must intersect fewer sets in \mathcal{L} than S . \square

We can now give the proof of Theorem 11.12, which we restate here.

Theorem 11.12: *For any basic feasible solution x to the linear program (11.12), there is a set $Z \subseteq W$ and a collection \mathcal{L} of subsets of vertices with the following properties:*

1. *For all $S \in \mathcal{L}$, $|S| \geq 2$ and S is tight, and for all $v \in Z$, v is tight.*
2. *The vectors $\chi_{E(S)}$ for $S \in \mathcal{L}$ and $\chi_{\delta(v)}$ for $v \in Z$ are linearly independent.*
3. *$|\mathcal{L}| + |Z| = |E|$.*
4. *The collection \mathcal{L} is laminar.*

Proof. As we said previously, we know that there exists a collection \mathcal{S} and set $Y \subseteq W$ that have the first three properties of the theorem. Let $\text{span}(\mathcal{S}, Y)$ be the span of the set of vectors $\{\chi_{E(S)} : S \in \mathcal{S}\} \cup \{\chi_{\delta(v)} : v \in Y\}$. Since there are $|E|$ linearly independent vectors in this set and the vectors have $|E|$ coordinates, clearly $\text{span}(\mathcal{S}, Y) = \mathbb{R}^{|E|}$. By Lemma 11.15, if \mathcal{T} is the set of all tight sets, then there exists a laminar collection of tight sets \mathcal{L} such that $\text{span}(\mathcal{L}) \supseteq \text{span}(\mathcal{T})$. Since $\mathcal{S} \subseteq \mathcal{T}$, then $\mathbb{R}^{|E|} = \text{span}(\mathcal{S}, Y) \subseteq \text{span}(\mathcal{T}, Y) \subseteq \text{span}(\mathcal{L}, Y) \subseteq \mathbb{R}^{|E|}$, so that $\text{span}(\mathcal{S}, Y) = \text{span}(\mathcal{L}, Y) = \mathbb{R}^{|E|}$. From the proof of Lemma 11.15 the vectors $\chi_{E(S)}$ for all $S \in \mathcal{L}$ are linearly independent. We now let $Z = Y$, and as long as there exists some $v \in Z$ such that $\chi_{\delta(v)} \in \text{span}(\mathcal{L}, Z - v)$, we remove v from Z . Note that $\text{span}(\mathcal{L}, Z)$ never decreases, and so the span is always $\mathbb{R}^{|E|}$. When this process terminates, the vectors $\chi_{E(S)}$ for $S \in \mathcal{L}$ and $\chi_{\delta(v)}$ for $v \in Z$ must be linearly independent, and furthermore since their span is $\mathbb{R}^{|E|}$, we must have $|E| = |\mathcal{L}| + |Z|$. \square

To summarize, we have been able to show that we can find a spanning tree of cost at most OPT such that each node $v \in W$ has degree at most $b_v + 2$. To do this, we used the properties

```

while  $W \neq \emptyset$  do
  Solve LP (11.12) on  $(V, E)$ ,  $W$ , get basic optimal solution  $x$ 
   $E \leftarrow E(x)$ 
  Select  $v \in W$  such that there are at most  $b_v + 1$  edges in  $E$  incident on  $v$ 
   $W \leftarrow W - \{v\}$ 
  Compute minimum-cost spanning tree  $F$  on  $(V, E)$ 
return  $F$ 

```

Algorithm 11.3: Deterministic rounding algorithm for finding a minimum-cost bounded-degree spanning tree.

of a basic feasible solution; we showed that the basic optimal solution has structure in terms of a laminar collection of tight sets. This structure allows us to prove Lemma 11.5, which, informally speaking, states that either we can find a leaf of the tree or some node $v \in W$ that will have degree at most $b_v + 2$. The lemma leads naturally to an algorithm for finding the desired tree.

We now give an algorithm that finds a spanning tree of cost at most OPT such that each node $v \in W$ has degree at most $b_v + 1$. As with the previous algorithm, we find a basic optimal solution of a linear programming relaxation of the problem. The key to the algorithm is proving a stronger version of Lemma 11.5: we show that if $W \neq \emptyset$, we can find some $v \in W$ such that there are at most $b_v + 1$ edges of $E(x)$ incident on v . Thus in our new algorithm, we will show that we will be able to remove degree bounds from the linear programming relaxation until none are left, in such a way that the cost of the solution to the corresponding relaxation does not increase. When all the degree bounds are removed, we will have a solution to the linear programming relaxation of the minimum spanning tree problem, and we previously saw in Theorem 11.4 that we can find a minimum spanning tree of cost at most the value of the linear programming relaxation.

As before, we define $E(x) = \{e \in E : x_e > 0\}$. Our algorithm will depend on the following lemma, whose proof we defer for a moment.

Lemma 11.16: *For any basic feasible solution x to the linear program (11.12) in which $W \neq \emptyset$, there is some $v \in W$ such that there are at most $b_v + 1$ edges incident on v in $E(x)$.*

Given the lemma, the algorithm is relatively straightforward. We solve the LP relaxation to obtain a basic optimal solution. All edges $e \in E$ such that $x_e = 0$ are removed from E . If $W \neq \emptyset$, the lemma states that there must exist some $v \in W$ such that there are at most $b_v + 1$ edges in $E(x)$. We remove v from W , since we know that when we obtain the minimum-cost spanning tree F , there can be at most $b_v + 1$ edges of F incident on v . If $W = \emptyset$, then by Theorem 11.4, we can compute a minimum spanning tree of cost at most the value of the linear programming relaxation. The algorithm is summarized in Algorithm 11.3. The following proof of the correctness of the algorithm is then not difficult.

Theorem 11.17: *Algorithm 11.3 produces a spanning tree F such that the degree of v in F is at most $b_v + 1$ for $v \in W$, and such that the cost of F is at most the value of the linear program (11.12).*

Proof. As in the proofs of Theorems 11.4 and 11.6, we show by induction that in each iteration, the LP solution x_e on (V, E) and W is feasible for the input (V, E') and W' for the next iteration. It is easy to see this since any time we remove an edge e from E , $x_e = 0$, which does not affect the feasibility of the solution. Also, we only remove vertices from W , imposing fewer

constraints. Thus the cost of the optimal LP solution on the input (V, E') and W' is no greater than that on the input (V, E) and W .

Since we remove a vertex v from W in each iteration, eventually $W = \emptyset$, and the algorithm computes a minimum spanning tree F on the final set of edges. By Theorem 11.4, this cost is at most the cost of the final linear programming solution. Since the value of the linear program never increases as we modify E and W , the cost of the minimum spanning tree is at most the value of initial linear program (11.12).

Furthermore, we removed v from W only if there were at most $b_v + 1$ edges remaining in E incident on v . Thus the computed spanning tree can have at most degree $b_v + 1$ for each vertex v in the initial set W . \square

We now turn to the proof of Lemma 11.16. As before, for the basic feasible solution x , we use the existence of a laminar collection \mathcal{L} and a set of vertices Z as given in Theorem 11.12. We also need the following lemma.

Lemma 11.18: *For any $e \in E$ such that $x_e = 1$, $\chi_e \in \text{span}(\mathcal{L})$.*

Proof. By the proof of Theorem 11.12, the laminar collection \mathcal{L} is such that $\text{span}(\mathcal{L}) \supseteq \text{span}(\mathcal{T})$, where \mathcal{T} is the collection of all tight sets. Notice that if $x_e = 1$ for $e = (u, v)$, then the set $S = \{u, v\}$ is tight, since $x_e = x(E(S)) = |S| - 1 = 1$. Thus $\chi_e = \chi_{E(S)} \in \text{span}(\mathcal{T}) \subseteq \text{span}(\mathcal{L})$. \square

We now restate Lemma 11.16 and give its proof.

Lemma 11.16: *For any basic feasible solution x to the linear program (11.12) in which $W \neq \emptyset$, there is some $v \in W$ such that there are at most $b_v + 1$ edges incident on v in $E(x)$.*

Proof. Assume the statement of the lemma is false, so that there are at least $b_v + 2$ edges in E incident on every $v \in Z$, with $W \neq \emptyset$. We derive the contradiction via a charging scheme, in which we assign each $v \in Z$ and each $S \in \mathcal{L}$ a certain nonnegative charge, possibly fractional. We will then show by the falsity of the lemma statement, each $v \in Z$ and each $S \in \mathcal{L}$ receives a charge of at least one. This will imply that the total charge is at least $|Z| + |\mathcal{L}| = |E|$. However, we will also show that the total charge assigned must have been strictly less than $|E|$, giving the contradiction. Thus there must exist some $v \in W \neq \emptyset$ such that there are at most $b_v + 1$ edges of E incident on v .

To carry out our charging scheme, for each $e \in E$ we assign a charge of $\frac{1}{2}(1 - x_e)$ to each endpoint of e that is in Z , and we assign a charge of x_e to the smallest $S \in \mathcal{L}$ that contains both endpoints of e , if such an S exists. Note then that have assigned a total charge of at most $(1 - x_e) + x_e = 1$ per edge $e \in E$.

Now we show that each $v \in Z$ and each $S \in \mathcal{L}$ receives a charge of at least one. Each $v \in Z$ receives a charge of $\frac{1}{2}(1 - x_e)$ from each edge of E incident on it. By hypothesis, there are at least $b_v + 2$ edges incident on v . Since $v \in Z$ implies v is tight, we know that $\sum_{e \in \delta(v)} x_e = b_v$. Thus the total charge received by v is at least one, since

$$\begin{aligned} \sum_{e \in \delta(v)} \frac{1}{2}(1 - x_e) &= \frac{1}{2} \left(|\delta(v)| - \sum_{e \in \delta(v)} x_e \right) \\ &\geq \frac{1}{2} [(b_v + 2) - b_v] \\ &= 1. \end{aligned}$$

Now consider a set $S \in \mathcal{L}$. If S contains no other set $C \in \mathcal{L}$, then since S is a tight set, $\sum_{e \in E(S)} x_e = |S| - 1$, and the total charge assigned to S is $|S| - 1$. Since $|S| \geq 2$, the total charge assigned to S is at least one. Now suppose that S contains some $C \in \mathcal{L}$. We call a set $C \in \mathcal{L}$ a *child* of S if it is strictly contained in S , and no other set $C' \in \mathcal{L}$ that contains C is also strictly contained in S . Let C_1, \dots, C_k be the children of S . Recalling that the sets S and C_1, \dots, C_k are tight, we have that $x(E(S)) = |S| - 1$ and $x(E(C_i)) = |C_i| - 1$ and thus are integral. Furthermore, the $E(C_i)$ are disjoint and are contained in $E(S)$ by the laminarity of \mathcal{L} . Hence the total charge assigned to S is

$$x(E(S)) - \sum_{i=1}^k x(E(C_i)) \geq 0.$$

However, it cannot be the case that $E(S) = \bigcup_{i=1}^k E(C_i)$ since then $\chi_{E(S)} = \sum_{i=1}^k \chi_{E(C_i)}$, which violates the second property of Theorem 11.12, which says that these vectors are linearly independent. Thus it must be the case that the total charge assigned to S is

$$x(E(S)) - \sum_{i=1}^k x(E(C_i)) > 0,$$

and since

$$x(E(S)) - \sum_{i=1}^k x(E(C_i)) = (|S| - 1) - \sum_{i=1}^k (|C_i| - 1)$$

is integral, the difference must be at least one. Therefore at least one unit of charge is assigned to S .

Finally we show that the total charge is strictly less than $|E|$ via a three-case argument. First, if $V \notin \mathcal{L}$, then there must exist some edge e such that $e \notin E(S)$ for all $S \in \mathcal{L}$. The charge $x_e > 0$ is not made to any set, so that the total charge is strictly less than $|E|$. Second, if there exists some $e = (u, v) \in E$ with $x_e < 1$ such that one of its two endpoints is not in Z (say $u \notin Z$), then the charge of $\frac{1}{2}(1 - x_e) > 0$ is not made to u , and again the total charge is strictly less than $|E|$. Finally, suppose that $V \in \mathcal{L}$ and for any $e \in E$ with $x_e < 1$, both endpoints of e are in Z . We will show that this case gives rise to a contradiction, so one of the first two cases must occur. We observe that $\sum_{v \in Z} \chi_{\delta(v)} = 2\chi_{E(Z)} + \chi_{\delta(Z)}$ and by hypothesis, each edge $e \in \delta(Z) \cup E(V - Z)$ has $x_e = 1$. Now we show that we can express $2\chi_{E(Z)} + \chi_{\delta(Z)}$ by the sum of vectors $\chi_{E(S)}$ for $S \in \mathcal{L}$, which will contradict the linear independence of the vectors $\chi_{E(S)}$ for $S \in \mathcal{L}$ and $\chi_{\delta(v)}$ for $v \in Z$. By Lemma 11.18, for any $e \in \delta(Z) \cup E(V - Z)$, $\chi_e \in \text{span}(\mathcal{L})$ since $x_e = 1$. Thus $\sum_{v \in Z} \chi_{\delta(v)} = 2\chi_{E(Z)} + \chi_{\delta(Z)} = 2\chi_{E(V)} - 2\sum_{e \in E(V - Z)} \chi_e - \sum_{e \in \delta(Z)} \chi_e$, and every term on the right-hand side is in $\text{span}(\mathcal{L})$; this proves that $\chi_{\delta(v)}$ for $v \in Z$ and $\chi_{E(S)}$ for $S \in \mathcal{L}$ are linearly dependent, which is a contradiction. \square

In summary, we have been able to show that we can find a spanning tree of cost at most OPT with degree at most $b_v + 1$ for $v \in W$. We did this by proving a lemma showing that a basic feasible solution x to the linear programming relaxation of the problem must have some $v \in W$ such that at most $b_v + 1$ edges of $E(x)$ are incident on v . The proof of this lemma, given above, uses the structure of the basic feasible solution as a laminar collection of tight sets, as well as a charging scheme that uses fractional charges. In the next section we will use many of these same ideas in giving an approximation algorithm for another network design problem.