Recursive Greedy Methods*

Guy Even

School of Electrical Engineering

Tel-Aviv University

Tel-Aviv 69978, Israel

E-mail: guy@eng.tau.ac.il

February 26, 2008

1 Introduction

Greedy algorithms are often the first algorithm that one considers for various optimization problems, and ,in particular, covering problems. The idea is very simple: try to build a solution incrementally by augmenting a partial solution. In each iteration, select the "best" augmentation according to a simple criterion. The term greedy is used because the most common criterion is to select an augmentation that minimizes the ratio of "cost" to "advantage". We refer to the cost-to-advantage ratio of an augmentation as the *density* of the augmentation.

In the Set-Cover (SC) problem, every set S has a weight (or cost) $w(S)$. The "advantage" of a set S with respect to a partial cover $\{S_1, \ldots, S_k\}$ is the number of new elements covered by S, i.e., $|S \setminus (S_1 \cup \ldots \cup S_k)|$. In each iteration, a set with a minimum density is selected and added to the partial solution until all the elements are covered. In the SC problem, it is easy to find an augmentation with minimum density simply by re-computing the density of every set in every iteration.

In this chapter we consider problems for which it is NP-hard to find an augmentation with minimum

*Chapter 3 from: *Handbook of Approximation Algorithms and Metaheuristics* edited by Teofilo Gonzalez.

density. From a covering point of view, this means that there are exponentially many sets. However, these sets are succinctly represented using a structure with polynomial complexity. For example, the sets can be paths or trees in a graph. In such problems, applying the greedy algorithm is a nontrivial task. One way to deal with such a difficulty, is to try to approximate a minimum density augmentation. Interestingly, the augmentation itself is computed using a greedy algorithm, and this is why the algorithm is called the recursive greedy algorithm.

The recursive greedy algorithm was presented by Zelikovsky [8] and Kortsarz and Peleg [6]. In [8], the Directed Steiner Tree (DST) problem in acyclic graphs was considered. In the DST problem the input consists of a directed graph $G = (V, E)$ with edge weights $w(e)$, a subset $X \subseteq V$ of terminals, and a root $r \in V$. The goal is to find a minimum weight subgraph that contains directed paths from r to every terminal in X. In [6], the Bounded Diameter Steiner Tree (BDST) problem was considered. In the BDST problem, the input consists of an undirected graph $G = (V, E)$ with edge costs $w(e)$, a subset of terminals $X \subseteq V$, and a diameter parameter d. The goal is to find a minimum weight tree that spans X with diameter bounded by d. In both papers, it is proved that, for every $\varepsilon > 0$, the recursive greedy algorithm achieves an $O(|X|^{\varepsilon})$ approximation ratio in polynomial time. The recursive greedy algorithm is still the only nontrivial approximation algorithm known for these problems.

The presentation of the recursive greedy algorithm was simplified and its analysis was perfected by Charikar et. al. [3]. In [3], the recursive greedy algorithm was used for the DST problem. The improved analysis gave a poly-logarithmic approximation ratio in quasi-polynomial time (i.e., running time is $O(n^{c \log n})$, for a constant c).

The recursive greedy algorithm is a combinatorial algorithm (i.e., no linear programming or high precision arithmetic is used). The algorithm's description is simple and short. The analysis captures the intuition regarding the segments during which the greedy approach performs well. The running time of the algorithm is exponential in the depth of the recursion, and hence, reducing its running time is an important issue.

We present modifications of the recursive greedy algorithm that enable reducing the running time. Unfortunately, these modifications apply only to the restricted case in which the graph is a tree. We demonstrate these methods on the Group Steiner (GS) problem [7] and its restriction to trees [4]. Following [2], we show that for the GS problem over trees, the recursive greedy algorithm can be modified to give a poly-logarithmic approximation ratio in polynomial time. Better poly-logarithmic approximation algorithms were developed for the GS problem, however, these algorithms rely on linear programming [4, 9].

Organization. In Section 2, we review the greedy algorithm for the SC problem. In Section 3, we present three versions of directed Steiner tree problems. We present simple reductions that allow us to focus on only one version. Section 4 constitutes the heart of this chapter; in it the recursive greedy algorithm and its analysis are presented. In Section 5, we consider the Group Steiner problem over trees. We outline modifications of the recursive greedy algorithm that enable a poly-logarithmic approximation ratio in polynomial time. We conclude in Section 6 with open problems.

2 A review of the greedy algorithm

In this section we review the greedy algorithm for the Set-Cover (SC) problem and its analysis.

In the Set-Cover (SC) problem we are given a a set of elements, denoted by $U = \{1, \ldots, n\}$ and a collection $\mathcal{R}$ of subsets of U. Each subset $S \in \mathcal{R}$ is also given a non-negative weight $w(S)$. A subset $\mathcal{C} \subseteq \mathcal{R}$ is a *set cover* if $\bigcup_{S' \in \mathcal{C}} S' = \{1, \ldots, n\}$. The *weight* of a subset of $\mathcal{R}$ is simply the sum of the weights of the sets in $\mathcal{R}$. The goal in the SC problem to find a cover of minimum weight. We often refer to a subset of $\mathcal{R}$ that is not a cover as a *partial cover.*

The greedy algorithm starts with an empty partial cover. A cover is constructed by iteratively asking an *oracle* for a set to be added to the partial cover. This means that no backtracking takes place; every set that is added to the partial cover is kept until a cover is obtained. The oracle looks for a set with the lowest *residual density*, defined as follows.

Definition 1.1 *Given a partial cover $\mathcal{C}$, the residual density of a set S is the ratio*

$$\rho_{\mathcal{C}}(S) \triangleq \frac{w(S)}{|S \setminus \bigcup_{S' \in \mathcal{C}} S'|}.$$

Note that the residual density is non-decreasing (and may even increase) as the greedy algorithm accumulates sets. The performance guarantee of the greedy algorithm is summarized in the following theorem (see Chapter R-2 on Greedy Methods).

Theorem 1.1 *The greedy algorithm computes a cover whose cost is at most $(1 + \ln n) \cdot w(C^*)$, where C^* is a minimum weight cover.*

There are two main questions that we wish to ask about the greedy algorithm:

Question 1: What happens if the oracle is approximate? Namely, what if the oracle does not return a set with minimum residual density, but a set whose residual density is at most α times the minimum residual density? How does such an approximate oracle affect the approximation ratio of the greedy algorithm? In particular we are interested in the case that α is not constant (e.g., α depends on the number of uncovered elements). We note that in the SC problem, an exact oracle is easy to implement. But we will see a generalization of the SC problem in which the task of an exact oracle is NP-hard, and hence we will need to consider an approximate oracle.

Question 2: What happens if we stop the execution of the greedy algorithm before a complete cover is obtained? Suppose that we stop the greedy algorithm when the partial cover covers $\beta \cdot n$ elements in U. Can we bound the weight of the partial cover? We note that one reason for stopping the greedy algorithm before it ends is that we simply run out of "budget" and cannot "pay" for additional sets.

The following lemma helps answer both questions raised above. Let x denote the number of elements that are not covered by the partial cover. We say that the oracle is $\alpha(x)$-approximate if the residual density of the set it finds is at most $\alpha(x)$ times the minimum residual density.

Lemma 1.1 ([3]) *Suppose that the oracle of the greedy algorithm is $\alpha(x)$-approximate and that $\alpha(x)/x$ is a non-increasing function. Let $\mathcal{C}_i$ denote partial cover accumulated by the greedy algorithm after adding i sets. Then,*

$$\frac{w(\mathcal{C}_i)}{w(\mathcal{C}^*)} \leq \int_{n-|\bigcup_{S' \in \mathcal{C}_i} S'|}^{n} \frac{\alpha(x)}{x} dx.$$

Proof

The proof is by induction on n. When $n = 1$, the algorithm simply returns a set S such that $w(S) \leq$

$\alpha(1) \cdot w(\mathcal{C}^*)$. Since $\alpha(x)/x$ is non-increasing, we conclude that $\alpha(1) \leq \int_0^1 \frac{\alpha(x)}{x}dx$, and the induction basis follows.

The induction step for $n > 1$ is proved as follows. Let $\mathcal{C}_i = \{S_1, \ldots, S_i\}$. When the oracle computes S_1, its density satisfies: $w(S_1)/|S_1| \leq \alpha(n) \cdot w(\mathcal{C}^*)/n$. Hence, $w(S_1) \leq |S_1| \cdot \frac{\alpha(n)}{n} \cdot w(\mathcal{C}^*)$. Since $\alpha(x)/x$ is non-increasing, $|S_1| \cdot \frac{\alpha(n)}{n} \leq \int_{n-|S_1|}^n \frac{\alpha(x)}{x}dx$. We conclude that

$$w(S_1) \leq \int_{n-|S_1|}^n \frac{\alpha(x)}{x}dx \cdot w(\mathcal{C}^*). \tag{1.1}$$

Now consider the residual set system over the set of elements $\{1, \ldots, n\} \setminus S_1$ with the sets $S' = S \setminus S_1$. We keep the set weights unchanged, i.e., $w(S') = w(S)$. The collection $\{S'_2, \ldots, S'_i\}$ is the output of the greedy algorithm when given this residual set system. Let $n' = |S'_2 \cup \cdots \cup S'_i|$. Since $\mathcal{C}^*$ induces a cover of the residual set with the same weight as $w(\mathcal{C}^*)$, the induction hypothesis implies that

$$w(S'_2) + \cdots + w(S'_i) \leq \int_{n-(n'+|S_1|)}^{n-|S_1|} \frac{\alpha(x)}{x}dx \cdot w(\mathcal{C}^*). \tag{1.2}$$

The lemma follows now by adding Equations 1.1 and 1.2. □

We remark that for a full cover, since $\int_0^1 dx/x$ is not bounded, one could bound the ratio by $\alpha(1) + \int_1^n \frac{\alpha(x)}{x}dx$. Note that for an exact oracle $\alpha(x) = 1$, this modification of Lemma 1.1 implies Theorem 1.1.

Lemma 1.1 shows that the greedy algorithm works also with approximate oracles. If $\alpha(x) = O(\log x)$, then the approximation ratio of the greedy algorithm is simply $O(\alpha(n) \cdot \log n)$. But, for example, if $\alpha(x) = x^\varepsilon$, then the lemma "saves" a factor of $\log n$ and shows that the approximation ratio is $\frac{1}{\varepsilon} \cdot n^\varepsilon$. So this settles the first question.

Lemma 1.1 also helps settle the second question. In fact, it proves that the greedy algorithm (with an exact oracle) is a bi-criteria algorithm in the following sense.

claim 1.1 *If the greedy algorithm is stopped when $\beta \cdot n$ elements are covered, then the cost of the partial cover is bounded by* $\ln\left(\frac{1}{1-\beta}\right) \cdot w(C^*)$.

The greedy algorithm surly does well with the first set it selects, but what can we say about the remaining selections? Claim 1.1 quantifies how well the greedy algorithm does as a function of the portion of the covered elements. For example, if $\beta = 1 - 1/e$, then the partial cover computed by the greedy algorithm weighs no more than $w(C^*)$. (We ignore here the knapsack-like issue of how to cover "exactly" $\beta \cdot n$ elements, and assume that, when we stopped the greedy algorithm, the partial cover covers $\beta \cdot n$ elements.) The lesson to be remembered here is that the greedy algorithm performs "reasonably well" as long as "few" elements have been covered.

The DST problem is a generalization of the SC problem. In fact, every SC instance can be represented as a DST instance over a layered directed graph with three vertex layers (see Fig. 1.1). The top layer contains only a root, the middle layer contains a vertex for every set, and the bottom layer contains a vertex for every element. The weight of an edge from the root to a set is simply the weight of the set. The weight of all edges from sets to elements are zero. The best approximation algorithm for SC is the greedy algorithm. What form could a greedy algorithm have for the DST problem?

3 Directed Steiner problems

In this section we present three versions of directed Steiner tree problems. We present simple reductions that allow us to focus on only the last version.

Notation and terminology. We denote the vertex set and edge set of a graph G by $V(G)$ and $E(G)$, respectively. An *arborescence* T rooted at r is a directed graph such that: (i) the underlying graph of T is a tree (i.e., if edge directions are ignored in T, then T is a tree), and (ii) there is a directed path in T from the root r to every node in T. If an arborescence T is a subgraph of G, then we say that T *covers* (or *spans*) a subset of vertices X if $X \subseteq V(T)$. If edges have weights $w(e)$, then the weight of a subgraph G' is simply $\sum_{e \in E(G')} w(e)$. We denote by T_v the subgraph of T that is induced all the vertices reachable from v (including v).

3.1 The problems

The directed Steiner tree (DST) problem. In the DST problem the input consists of a directed graph G, a set of terminals $X \subseteq V(G)$, positive edge weights $w(e)$, and a root $r \in V(G)$. An arborescence T rooted at r is a *directed Steiner tree (*DST*)* if it spans the set of terminals X. The goal in the DST problem is to find a minimum weight directed Steiner tree.

The k-DST problem. Following [3], we consider a version of the DST problem, called k-DST, in which only part of the terminals must be covered. In the k-DST problem, there is an additional parameter k, often called the *demand*. An arborescence T rooted at r is a *k-partial* directed Steiner tree (k-DST) if $|V(T) \cap X| \geq k$. The goal in the k-DST problem is to find a minimum weight k-partial directed Steiner tree. We denote the weight of an optimal k-partial directed Steiner tree by $DS^*(G, X, k)$. (Formally, the root r should be a parameter, but we omit it to shorten notation.) We encode DST instances as k-DST instances simply by setting $k = |X|$.

The ℓ-shallow k-DST problem. Following [6], we consider a version of the k-DST problem in which the length of the paths from the root to the terminals is bounded by a parameter ℓ. A rooted arborescence in which every node is at most ℓ edges away from the root is called an *ℓ-layered tree.* (Note that we count the number of layers of edges; the number of layers of nodes is $\ell + 1$.) In the ℓ-shallow k-DST problem the goal is to compute a minimum k-DST among all ℓ-layered trees.

3.2 Reductions

Obviously, the k-DST problem is a generalization of the DST problem. Similarly, the ℓ-shallow k-DST problem is a generalization of the k-DST problem (i.e., simply set $\ell = |V| - 1$). The only nontrivial approximation algorithm we know is for the ℓ-shallow k-DST problem; this approximation algorithm is the recursive greedy algorithm. Since its running time is exponential in ℓ, we need to consider reductions that result with as small as possible values of ℓ.

For this purpose we consider two well known transformations: transitive closure and layering. We now define each of these transformations.

Transitive closure. The *transitive closure* of G is a directed graph $TC(G)$ over the same vertex set. For every $u, v \in V$, the pair (u, v) is an edge in $E(TC(G))$ if there is a directed path from u to v in G. The weight $w'(u, v)$ of an edge in $E(TC(G))$ is the minimum weight of a path in G from u to v.

The weight of an optimal k-DST is not affected by applying transitive closure namely,

$$DS^*(G, X, k) = DS^*(TC(G), X, k). \tag{1.3}$$

This means that replacing G by its transitive closure does not change the weight of an optimal k-DST. Hence we may assume that G is transitively closed, i.e., $G = TC(G)$.

Layering. Let ℓ denote a positive integer. We reduce the directed graph G into an ℓ-layered directed acyclic graph LG_ℓ as follows (see Fig. 1.2). The vertex set $V(LG_\ell)$ is simply $V(G) \times \{0, \ldots, \ell\}$. The jth layer in $V(LG_\ell)$ is the subset of vertices $V(G) \times \{j\}$. We refer to $V(G) \times \{0\}$ as the *bottom layer* and to $V(G) \times \{\ell\}$ as the *top layer*. The graph LG_ℓ is layered in the sense that $E(LG_\ell)$ contains only edges from the $V(G) \times \{j+1\}$ to $V(G) \times \{j\}$, for $j < \ell$. The edge set $E(LG_\ell)$ contains two types of edges: *regular* edges and *parallel* edges. For every $(u, v) \in E(G)$ and every $j < \ell$, there is a regular edge $(u, j+1) \to (v, j) \in E(LG_\ell)$. For every $u \in V$ and every $j < \ell$, there is a parallel edge $(u, j+1) \to (u, j) \in E(LG_\ell)$. All parallel edges have zero weight. The weight of a regular edge is inherited from the original edge, namely, $w((u, j+1) \to (v, j)) = w(u, v)$. The set of terminals X' in $V(LG_\ell)$ is simply $X \times \{0\}$, namely, images of terminals in the bottom layer. The root in LG_ℓ is the node (r, ℓ). The following observation shows that we can restrict our attention to layered graphs.

Observation 1.1 *There is a weight preserving and terminal preserving correspondence between ℓ-layered r-rooted trees in G and (r, ℓ)-rooted trees in LG_ℓ. In particular, $w(LT^*_\ell) = DS^*(LG_\ell, X', k)$, where LT^*_ℓ denotes a minimum weight k-DST among all ℓ-layered trees.*

Observation 1.1 implies that if we wish to approximate LT^*_ℓ, then we may apply layering and assume that the input graph is an ℓ-layered acyclic graph in which the root is in the top layer and all the terminals are in the bottom layer.

Limiting the number of layers. As we pointed out, the running time of the recursive greedy algorithm is exponential in the number of layers. It is therefore crucial to be able to bound the number of layers. The following lemma bounds the penalty incurred by limiting the number of layers in the Steiner tree. The proof of the lemma appears in Appendix A. (A slightly stronger version appears in [5], with the ratio $2^{1-1/\ell} \cdot \ell \cdot k^{1/\ell}$).

Lemma 1.2 ([8] corrected in [5]) *If G is transitively closed, then $w(LT^*_\ell) \leq \frac{\ell}{2} \cdot k^{2/\ell} \cdot DS^*(G, X, k)$.*

It follows that an α-approximate algorithm for an ℓ-shallow k-DST is also an $\alpha\beta$-approximation algorithm for k-DST, where $\beta = \frac{\ell}{2} \cdot k^{2/\ell}$. We now focus on the development of an approximation algorithm for the ℓ-shallow k-DST problem.

4 A recursive greedy algorithm for ℓ-shallow k-DST

In this section present a recursive greedy algorithm for the ℓ-shallow k-DST problem. Based on the layering transformation, we assume that the input graph is an ℓ-layered acyclic directed graph G. The set of terminals, denoted by X, is contained in the bottom layer. The root, denoted by r, belongs top layer.

4.1 Motivation

We now try to extend the greedy algorithm to the ℓ-shallow k-DST problem. Suppose we have a directed tree $T \subseteq G$ that is rooted at r. This tree only covers part of the terminals. Now we wish to augment T so that it covers more terminals. In other words, we are looking for an r-rooted augmenting tree T_{aug} to be added to the T. We follow the minimum density heuristic, and define the residual density of T_{aug} by

$$\rho_T(T_{aug}) \triangleq \frac{w(T_{aug})}{|(T_{aug} \cap X) \setminus (T \cap X)|}.$$

All we need now is an algorithm that finds an augmenting tree with the minimum residual density. Unfortunately, this problem is by itself NP-hard! Consider the following reduction: Let G denote the 2-layered DST instance mentioned above to represent a Set-Cover instance. Add a layer with a single node r' that is connected to the root r of G. The weight of the edge (r', r) should be large (say, n times the sum of the weights of the sets). It is easy to see that every minimum density subtree must span all the terminals. Hence, every minimum density subtree induces a minimum weight set cover, and finding a minimum density subtree in a 3-layered graph is already NP-hard. We show in Section 4.3 that for two or less layers, one can find a minimum density augmenting tree in polynomial time.

We already showed that the greedy algorithm works well also with an approximate oracle. So we try to approximate a subtree with minimum residual density. The problem is how to do it? The answer is by applying a greedy algorithm recursively!

Consider an ℓ-layered directed graph and a root r. The algorithm finds an low density ℓ-layered augmenting tree by accumulating low density $(\ell - 1)$-layered augmenting trees that hang from the children of r. These trees are found by augmenting low density trees that hang from grandchildren of r, and so on. We now formally describe the algorithm.

4.2 The recursive greedy algorithm

Notation. We denote the number of terminals in a subgraph G' by $k(G')$ (i.e., $k(G') = |X \cap V(G')|$). Similarly, for a set of vertices U, $k(U) = |X \cap U|$. We denote the set of vertices reachable in G from u by $desc(u)$. We denote the layer of a vertex u by $layer(u)$ (e.g., if u is a terminal, then $layer(u) = 0$).

Description A listing of the algorithm DS(u, k) appears as Algorithm 1. The stopping condition is when u belongs to the bottom layer or when the number of uncovered terminals reachable from u is less than the demand k (i.e., the instance is infeasible). In either case, the algorithm simply returns the root $\{r\}$.

The algorithm maintains a partial cover T that is initialized to the single vertex u. The augmenting tree T_{aug} is selected as the best tree found by the recursive calls to the children of u (together with the edge from u to its child). Note that the recursive calls are applied to all the children of u and all the possible demands k'. After T_{aug} is added to the partial solution, the terminals covered by T_{aug} are erased from the set of terminals so that the recursive calls will not attempt to cover terminals again. Once the demand is met, namely, k terminals are covered, the accumulated cover T is returned.

The algorithm is invoked with the root r, the demand k, and the set of terminals X. Note that if the instance is feasible (namely, at least k terminals are reachable from the root), then the algorithm never encounters infeasible sub-instances during its execution.

Algorithm 1 DS(u, k, X) - A recursive greedy algorithm for the Directed Steiner Tree Problem. The graph is layered and all the vertices in the bottom layer are terminals. The set of terminals is denoted by X. We are searching for a tree rooted at u that covers k terminals.

```
1: stopping condition: if $layer(u) = 0$ or $k(desc(u)) < k$ then return $(\{u\})$.
2: initialize: $T \leftarrow \{u\}$; $X^{res} \leftarrow X$;.
3: while $k(T) < k$ do
4:    recurse: for every $v \in children(u)$ and every $k' \leq \min\{k - k(T), |desc(v) \cap X^{res})|\}$
          $$T_{v,k'} \leftarrow \mathrm{DS}(v, k', X^{res}).$$
5:    select: Let $T_{aug}$ be a lowest residual density tree among the trees $T_{v,k'} \cup \{(u,v)\}$, where $v \in children(u)$
      and $k' \leq k - k(T)$.
6:    augment & update: $T \leftarrow T \cup T_{aug}$; $X^{res} \leftarrow X^{res} \setminus V(T_{aug})$.
7: end while
8: return $(T)$.
```

4.3 Analysis

Minimum residual density subtree. Consider a partial solution T rooted at u accumulated by the algorithm. A tree T' rooted at u is a *candidate tree* for augmentation if: (i) every vertex $v \in V(T')$ in the bottom layer of G is in X^{res} (i.e., T' covers only new terminals) and (ii) $0 < k(T') \leq k - k(T)$ (i.e., T' does not cover more terminals than the residual demand). We denote by T'_u a tree with minimum residual density among all the candidate trees.

We leave the proof of the following lemma as an exercise.

Lemma 1.3 *Assume that $w_i, k_i > 0$, for every $0 \leq i \leq n$. Then,* $\min_i \frac{w_i}{k_i} \leq \frac{\sum_i w_i}{\sum_i k_i} \leq \max_i \frac{w_i}{k_i}$.

Corollary 1.1 *If u is not a terminal, then we may assume that u has a single child in T'_u.*

Proof

We show that we could pick a candidate tree with minimum residual density in which u has a single child. Suppose that u has more than one child in T'_u. To every edge $e_j = (u, v_j) \in E(T'_u)$ we match a subtree A_{e_j} of T'_u. The subtree A_{e_j} contains u, the edge (u, v_j), and the subtree of T'_u hanging from v_j. The subtrees $\{A_{e_j}\}_{e_j}$ form an edge-disjoint decomposition of T'_u. Let $w_j = w(A_{e_j})$ and $k_j = k(A_{e_j} \setminus T)$. Since u is not a terminal, the subtrees $\{A_{e_j}\}_{e_j}$ partition the terminals in $V(T'_u)$, and $k(T'_u) = \sum_j k_j$. Similarly, $w(T'_u) = \sum_j w_j$. By Lemma 1.3, it follows that one of the trees A_{e_j} has a residual density that is not greater than the residual density of T'_u. Use this minimum residual density subtree instead of T'_u, and the corollary follows. □

Density. Note that edge weights are nonnegative and already covered terminals do not help in reducing the residual density. Therefore, every augmenting tree T_{aug} covers only new terminals and does not contain terminals already covered by T. It follows that every terminal in T_{aug} belongs to X^{res} and, therefore, $k(T_{aug}) = |T_{aug} \cap X^{res}|$. We may assume that the same holds for T'_u; namely, T'_u does not contain already covered terminals. Therefore, where possible, we ignore the "context" T in the definition of the residual density and simply refer to *density*, i.e., the density of a tree T' is $\rho(T') = w(T')/|V(T') \cap X|$.

Notation and Terminology. A *directed star* is a 1-layered rooted directed graph (i.e., there is a center out of which directed edges emanate to the leaves). We abbreviate and refer to a directed star simply as a star. A *flower* is a 2-layered rooted graph in which the root has a single child.

Bounding the density of augmenting trees. When $layer(u) = 1$, if u has least k terminal neighbors, then the algorithm returns a star centered at u. The number of edges emanating from r in the star equals k, and these k edges are the k lightest edges emanating from r to terminals. It is easy to see that in this case the algorithm returns an optimal k-DST.

The analysis of the algorithm is based on the following claim that bounds the ratio between the densities of the augmenting tree and T'_u.

claim 1.2 ([3]) *If $layer(u) \geq 2$, then, in every iteration of the while loop in an execution of $DS(u, k)$, the subtree T_{aug} satisfies:*

$$\rho(T_{aug}) \leq (layer(u) - 1) \cdot \rho(T'_u).$$

Proof

The proof is by induction on $layer(u)$. Suppose that $layer(u) = 2$. By Corr. 1.1, T'_u is a flower that consists of a star S_v centered at a neighbor v of u, the node u, and the edge (u, v). Moreover, S_v contains the $k(T'_u)$ closest terminals to v. When the algorithm computes T_{aug} it considers all stars centered at children v' of u consisting of the $k' \leq k - k(T)$ closest terminals to v'. In particular, it considers the star S_v together with the edge (u, v). Hence, $\rho(T_{aug}) \leq \rho(T'_u)$, as required.

We now prove the induction step for $layer(u) > 2$. Let $i = layer(u)$. The setting is as follows: during an execution of $DS(u, X)$, a partial cover T has been accumulated, and now an augmenting tree T_{aug} is computed. Our goal is to bound the density of T_{aug}.

By Coro. 1.1, u has a single child in T'_u. Denote this child by u'. Let $B_{u'}$ denote the subtree of T'_u that hangs from u' (i.e., $B_{u'} = T'_u \setminus \{u, (u, u')\}$). Let $k' = k(T'_u)$.

We now analyze the selection of T_{aug} while bearing in mind the existence of the "hidden candidate" T'_u that covers k' terminals. Consider the tree $T_{u',k'}$ computed by the recursive call $\mathrm{DS}(u', k', X^{res})$. We would like to argue that $T_{u',k'}$ should be a good candidate. Unfortunately, that might not be true! However, recall that the greedy algorithm does "well" as long as "few" terminals are covered. So we wish to show that a "small prefix" of $T_{u',k'}$ is indeed a good candidate. We now formalize this intuition.

The tree $T_{u',k'}$ is also constructed by a sequence of augmenting trees, denoted by $\{A_j\}_j$. Namely, $T_{u',k'} = \bigcup_j A_j$. We identify the smallest index ℓ for which the union of augmentations $A_1 \cup \cdots \cup A_\ell$ covers at least $k'/(i-1)$ terminals (recall that $i = layer(u)$). Formally,

$$k\left(\bigcup_{j=1}^{\ell-1} A_j\right) < \frac{k'}{(i-1)} \leq k\left(\bigcup_{j=1}^{\ell} A_j\right).$$

Our goal is to prove the following two facts. Fact (1): Let $k'' = k(\bigcup_{j=1}^{\ell} A_j)$, then the candidate tree $T_{u',k''} = \mathrm{DS}(u', k'', X^{res})$ equals the prefix $\bigcup_{j=1}^{\ell} A_j$. Fact (2): The density of $T_{u',k''}$ is small, i.e., $\rho(T_{u',k''}) \leq (i-1) \cdot \rho(B_{u'})$.

The first fact is a "simulation argument" since it claims that the union of the first ℓ augmentations computed in the course of the construction of $T_{u',k'}$ is actually one of the candidate trees computed by the algorithm. This simulation argument holds because, as long as the augmentations do not meet the demand, the same prefix of augmentations is computed. Note that k'' is the formalization of "few" terminals (compared to k'). Using $k'/(i-1)$ as an exact measure for a few terminals does not work because the simulation argument would fail.

The second fact states that the density of the candidate $T_{u',k''}$ is smaller than $(i-1) \cdot \rho(B_{u'})$. Note that $B_{u'}$ and $A_1 \cup \cdots \cup A_{\ell-1}$ may share terminals (if fact, we would "like" the algorithm to "imitate" $B_{u'}$ as much as possible). Hence, the residual density of $B_{u'}$ may increase as a result of adding the trees $A_1, \ldots, A_{\ell-1}$.

However, since $k(A_1 \cup \cdots \cup A_{\ell-1}) < k'/(i-1)$, it follows that even after accumulating $A_1 \cup \cdots \cup A_{\ell-1}$, the residual density of $B_{u'}$ does not grow by much. Formally, the residual density of $B_{u'}$ after accumulating $A_1 \cup \cdots A_{\ell-1}$ is bounded as follows:

$$\begin{aligned}
\rho_{(T\cup A_1 \cup \cdots \cup A_{\ell-1})}(B_{u'}) &= \frac{w(B_{u'})}{k' - k(A_1 \cup \cdots A_{\ell-1})} \\
&\leq \frac{w(B_{u'})}{k' \cdot (1 - \frac{1}{i-1})} \\
&= \left(\frac{i-1}{i-2}\right) \cdot \rho(B_{u'}). \qquad (1.4)
\end{aligned}$$

We now apply the induction hypothesis to the augmenting trees A_j (for $j \leq \ell$), and bound their residual densities by $(layer(u') - 1)$ times the "deteriorated" density of $B_{u'}$. Formally, the induction hypothesis implies that when A_j is selected as an augmentation tree its density satisfies:

$$\begin{aligned}
\rho(A_j) &\leq (i-2) \cdot \rho_{(T\cup A_1 \cdots \cup A_{j-1})}(B_{u'}) \\
&\leq (i-1) \cdot \rho(B_{u'}) && \text{(by Eq. 1.4).}
\end{aligned}$$

By Lemma 1.3, $\rho(\bigcup_{j=1}^{\ell} A_j) \leq \max_{j=1..\ell} \rho(A_j)$. Hence $\rho(T_{u',k''}) \leq (i-1) \cdot \rho(B_{u'})$, and the second fact follows.

To complete the proof, we need to deal with the addition of the edge (u, u').

$$\begin{aligned}
\rho(\{(u,u')\} \cup T_{u',k''}) &= \frac{w(u,u') + w(T_{u',k''})}{k''} \\
&\leq \frac{w(u,u')}{k'} \cdot (i-1) + \rho(T_{u',k''}) && \text{(since } k'' \geq \frac{k'}{i-1}\text{)} \\
&\leq (i-1) \cdot \rho(\{(u,u')\} \cup B_{u'}) && \text{(by fact (2))} \\
&= (i-1) \cdot \rho(T'_u).
\end{aligned}$$

The claim follows since $\{(u,u')\} \cup T_{u',k''}$ is only one of the candidates considered for the augmenting tree T_{aug} and hence $\rho(T_{aug}) \leq \rho(\{(u,u')\} \cup T_{u',k''})$. □

Approximation ratio. The approximation ratio follows immediately from Lemma 1.1.

claim 1.3 *Suppose that G is ℓ-layered. Then, the approximation ratio of Algorithm $DS(r,k,X)$ is $O(\ell \cdot \log k)$.*

Running time. For each augmenting tree, Algorithm $DS(u,k,X)$ invokes at most $n \cdot k$ recursive calls from children of u. Each augmentation tree covers at least one new terminal, so there are at most k augmenting trees. Hence, there are at most $n \cdot k^2$ recursive calls from the children of u. Let $time(\ell)$ denote the running time of $DS(u,k,X)$, where $\ell = layer(u)$. Then the following recurrence holds: $time(\ell) \le (n \cdot k^2) \cdot time(\ell - 1)$. We conclude that the running time is $O(n^\ell \cdot k^{2\ell})$.

4.4 Discussion

Approximation of k-DST The approximation algorithm is presented for ℓ-layered acyclic graphs. In Section 3.2, we presented a reduction from the k-DST problem to the ℓ-shallow k-DST problem. The reduction is based on layering and its outcome is an ℓ-layered acyclic graph. We obtain the following approximation result from this reduction.

Theorem 1.2 ([3]) *For every ℓ, there an $O(\ell^3 \cdot k^{2/\ell})$-approximation algorithm for the k-DST problem with running time $O(k^{2\ell} \cdot n^\ell)$.*

Proof

The preprocessing time is dominated by the running time of $DS(r,k,X)$ on the graph after it is transitively closed and layered into ℓ layers.

Let R^* denote an minimum residual density augmenting tree in the transitive closure of the graph (without the layering). Let T'_{k^*} denote a minimum residual subtree rooted at u in the layered graph among the candidate trees that cover $k(R^*)$ terminals. By Lemma 1.2, $w(T'_{k^*}) \le \ell/2 \cdot k(R^*)^{\ell/2} \cdot w(R^*)$, and hence, $\rho(T'_{k^*}) \le \ell/2 \cdot k(R^*)^{\ell/2} \cdot \rho(R^*)$. Since $\rho(T'_u) \le \rho(T'_{k^*})$, by Claim 1.2 it follows that, $\rho(T_{aug}) \le (\ell - 1) \cdot \ell/2 \cdot k^{2/\ell} \cdot \rho(R^*)$.

We now apply Lemma 1.1. Note that $\int \frac{x^{2/\ell}}{x} dx = \frac{\ell}{2} \cdot x^{2/\ell}$. Hence, $w(T) = O(\ell^3 \cdot k^{2/\ell})$, where T is the tree returned by the algorithm, and the theorem follows. □

We conclude with the following result.

Corollary 1.2 *For every constant $\varepsilon > 0$, there exists a polynomial time $O(k^{1/\varepsilon})$-approximation algorithm for the k-DST problem. There exists a quasi-polynomial time $O(\log^3 k)$-approximation algorithm for the k-DST problem.*

Proof

Substitute $\ell = 2/\varepsilon$ and $\ell = \log k$ in Theorem 1.2. □

Preprocessing Computing the transitive closure of the input graph is necessary for the correctness of the approximation ratio. Recall that Lemma 1.2 holds only if G is transitively closed.

Layering, on the other hand, is used to simplify the presentation. Namely, the algorithm can be described without layering (see [6, 3]). The advantage of using layering is that it enables a unified presentation of the algorithm (i.e., there is no need to deal differently with 1-layered trees). In addition, the layered graph is acyclic, so we need not consider multiple "visits" of the same node. Finally, for a given node u, we know from its layer what the recursion level is (i.e., the recursion level is $\ell - layer(u)$) and what the height of the tree we are looking for is (i.e., current height is $layer(u)$).

Suggestions for improvements. One might try to reduce the running time by not repeating computations associated with the computations of candidate trees. For example, when computing the candidate $T_{v,k-k(T))}$ the algorithm computes a sequence of augmenting trees that is used to build also other candidates rooted at v that cover fewer terminals (we relied on this phenomenon in the simulation argument used in the proof of Claim 1.2) . However, such improvements do not seem to reduce the asymptotic running time; namely, the running time would still be exponential in the number of layers and the basis would still be polynomial. We discuss other ways to reduce the running time in the next section.

Another suggestion to improve the algorithm is to zero the weight of edges when they are added to the partial cover T (see [8]). Unfortunately, we do not know how to take advantage of such a modification in the analysis and, therefore, keep the edge weights unchanged even after we pay for them.

5 Improving the running time

In this section we consider a setting in which the recursive greedy algorithm can be modified to obtain a poly-logarithmic approximation ratio in polynomial time. The setting is with a problem called the Group Steiner (GS) problem, and only part of the modifications are applicable also to the k-DST problem. (Recall that the problem of finding a polynomial time poly-logarithmic approximation algorithm for k-DST is still open.)

Motivation. We saw that the running time of the recursive greedy algorithm is $O((nk^2)^\ell)$, where k is the demand (i.e., number of terminals that need to be covered), the degree of a vertex is $n-1$ (since transitive closure was applied), and ℓ is the bound on the number of layers we allow in the k-DST.

To obtain polynomial running times, we first modify the algorithm and preprocess the input so that its running time is $\log(n)^{O(\ell)}$. We then set $\ell = \log n / \log\log n$. Note that

$$(\log n)^{\frac{\log n}{\log\log n}} = n.$$

Hence, a polynomial running time is obtained!

Four modifications are required to make this idea work:

1. Bound the number of layers - we already saw that the penalty incurred by limiting the number of layers can be bounded. In fact, according to Lemma 1.2, the penalty incurred by $\ell = \log n/\log\log n$ is poly-logarithmic (since $\ell \cdot k^{2/\ell} = (\log n)^{O(1)}$).

2. Degree reduction - we must reduce the maximum degree so that it is poly-logarithmic, otherwise too many recursive calls are invoked. Preprocessing of GS instances over trees achieves such a reduction in the degree.

3. Avoiding small augmenting trees - we must reduce the number of iterations of the while-loop. The number of iterations can be bounded by $(logn)^c$ if we require that every augmenting tree must cover at least a poly-logarithmic fraction of the residual demand.

4. Geometric search - we must reduce the number of recursive calls. Hence, instead of considering all demands below the residual demand, we consider only demands that are powers of $(1+\varepsilon)$.

The Group Steiner (GS) problem over trees. We now present a setting where all four modifications can be implemented. In the GS problem over trees the input consists of: (i) an undirected tree T rooted at r with nonnegative edge edges w(e) and (ii) groups $g_i \subseteq V(T)$ of terminals. A subtree $T' \subseteq T$ rooted at r covers k groups if $V(T')$ intersects at least k groups. We refer to a subtree that covers k groups as a k-GS tree. The goal is to find a minimum weight k-GS tree.

We denote the number of vertices by n and the number of groups by m. For simplicity, assume that every terminal is leaf of T and that every leaf of T is a terminal. In addition, we assume that the groups g_i are disjoint. Note that the assumption that the groups are disjoint implies that $\sum_{i=1}^{m} |g_i| \leq n$.

Bounding the number of layers. Lemma 1.2 applies also to GS instances over trees, provided that the transitive closure is used. Before transitive closure is used, we direct the edges from the node closer to the root to the node farther away from the root. As mentioned above, limiting the number of layers to $\ell = \log n / \log\log n$ incurs a poly-logarithmic penalty.

However, there is a problem with bounding the number of layers according to Lemma 1.2. The problem is that we need to transitively close the tree. This implies that we lose the tree topology and end up with an directed acyclic graph instead. Unfortunately, we only know how to reduce the maximum degree of trees, not of directed acyclic graphs. Hence, we need to develop a different reduction that keeps the tree topology.

In [2], a height reduction for trees is presented. This reduction replaces T by an ℓ-layered tree T'. The penalty incurred by this reduction is $O(n^{c/\ell})$, where c is a constant. The details of this reduction appear in [2].

Reducing the maximum degree. We now sketch how to preprocess the tree T to obtain a tree $\nu(T)$ such that: (i) There is a weight preserving correspondence between k-GS trees in T and in $\nu(T)$. (ii) The maximum number of children of a vertex in $\nu(T)$ is bounded by an integer $\beta \geq 3$. (iii) The number of layers in $\nu(T)$ is bounded by the number of layers in T plus $\lfloor \log_{\beta/2} n \rfloor$. We set $\beta = \lceil \log n \rceil$, and obtain the required reduction.

We define a node $v \in V(T)$ to be *β-heavy* if the number of terminals that are descendants of v is at least n/β; otherwise v is *β-light*.

Given a tree T rooted at u and a parameter β, the tree $\nu(T)$ is constructed recursively as follows. If u is a leaf, then the algorithm returns u. Otherwise, the star induced by u and its children is locally transformed as follows. Let $v_1, v_2, \dots, v_k$ denote the children of u.

1. Edges between u and β-heavy children v_i of u are not changed.

2. The β-light children of u are grouped arbitrarily into minimal bunches such that each bunch (except perhaps for the last) is β-heavy. Note that the number of leaves in each bunch (except perhaps for the last bunch) is in the half closed interval $[n_u/\beta, 2n_u/\beta)$. For every bunch B, a new node b is created. An edge (u, b) is added as well as edges between b and the children of u in the bunch B. The edge weights are set as follows: (a) $w(u, b) \leftarrow 0$, and (b) $w(b, v_i) \leftarrow w(u, v_i)$.

After the local transformation, let $v'_1, v'_2, \dots, v'_j$ be the new children of u. Some of these children are the original children and some are the new vertices introduced in the bunching. The tree $\nu(T)$ is obtained by recursively processing the subtrees $T_{v'_i}$, for $1 \le i \le j$, in essence replacing $T_{v'_i}$ by $\rho(T_{v'_i})$.

The maximum number of children after processing is at most β because the subtrees $\{T_{v'_i}\}_i$ partition the nodes of $V(T_u) - \{u\}$ and each tree except, perhaps one, is β-heavy. The recursion is applied to each subtree $T_{v'_i}$, and hence $\nu(T)$ will satisfies the degree requirement, as claimed. The weight preserving correspondence between k-GS trees in T and in $\nu(T)$ follows from the fact that the "shared" edges (u, b) that were created for bunching together β-light children of u have zero weight.

We now bound the height of $\nu(T)$. Consider a path p in $\nu(T)$ from the root r to a leaf v. All we need to show is that p contains at most $\log_{\beta/2} n$ new nodes (i.e., nodes corresponding to bunches of β-light vertices). However, the number of terminals hanging from a node along p decreases by a factor of $\beta/2$ every time we traverse such a new node, and the bound on the height of $\nu(T)$ follows.

The modified algorithm. We now present the modified recursive greedy algorithm for GS over trees. A listing of the modified recursive greedy algorithm appears as Algorithm 2.

The following notation is used in the algorithm. The input is a rooted undirected tree T which does not appear as a parameter of the input. Instead, a node u is given, and we consider the subtree of T that hangs from u. We denote this subtree by T_u. The partial cover accumulated by the algorithm is denoted by *cover*. The set of groups of terminals is denoted by $\mathcal{G}$. The set of groups of terminals not covered by *cover* is denoted by $\mathcal{G}^{res}$. The number of groups covered by *cover* is denoted by $k(\mathit{cover})$. The height of a tree T_u is the maximum number of edges along a path from u to a leaf in T_u. We denote the height of T_u by $h(T_u)$.

Two parameters λ and γ_v appear in the algorithm. The parameter λ is set to equal $1/h(T)$. The parameter γ_v satisfies $1/\gamma_v = |\mathit{children}(v)| \cdot (1 + 1/\lambda) \cdot (1 + \lambda)$.

Lines that are significantly modified (compared to Algorithm 1) are underlined. In line 4, two modifications take place. First, the smallest demand is not one, but a poly-logarithmic fraction of the residual demand (under the assumption that the maximum degree and the height is poly-logarithmic). Second, only demands that are powers of $(1 + \lambda)$ are considered. In line 7, the algorithm also stores the partial cover that first covers at least $1/h(T_u)$ of the initial demand k. This change is important for the simulation argument in the proof. Since the algorithm does not consider all the demands, we need to consider also the partial cover that the simulation argument points to. Finally, in line 9, we return the partial cover with the best density among *cover* and *cover*$_h$. Again, this selection is required for the simulation argument.

Note that Modified-GS$(u, k, \mathcal{G})$ may return now a cover that covers less than k groups. If this happens in the topmost call, then one needs to iterate until a k-GS cover is accumulated.

Algorithm 2 Modified-GS$(u, k, \mathcal{G})$ - Modified recursive greedy algorithm for k-GS over trees.

```
1: stopping condition: if $u$ is a leaf then return $(\{u\})$.
2: Initialize: $\mathit{cover} \leftarrow \{u\}$ and $\mathcal{G}^{res} \leftarrow \mathcal{G}$.
3: while $k(\mathit{cover}) < k$ do
4:     recurse: for every $v \in$ children$(u)$ and
       for every $k'$ $\underline{\text{power of } (1+\lambda) \text{ in } [\gamma_r \cdot (k - k(\mathit{cover})), k - k(\mathit{cover})]}$
           $$T_{v,k'} \leftarrow \text{Modified-GS}(v, k', \mathcal{G}^{res}).$$
5:     select: (pick the lowest density tree)
           $$T_{aug} \leftarrow \textsc{min-density}\left\{T_{v,k'} \cup \{(u,v)\}\right\}.$$
6:     augment & update: $\mathit{cover} \leftarrow \mathit{cover} \cup T_{aug}$; $\mathcal{G}^{res} \leftarrow \mathcal{G}^{res} \setminus \{g_i : T_{aug} \text{ intersects } g_i\}$.
7:     keep $k/h(T_u)$-cover: $\underline{\textbf{if} \text{ first time } k(\mathit{cover}) \geq k/h(T_u) \textbf{ then } \mathit{cover}_h \leftarrow \mathit{cover}.}$
8: end while
9: return $\underline{(\text{lowest density tree} \in \{\mathit{cover}, \mathit{cover}_h\})}$.
```

The following claim is proved in [2]. It is analogous to Claim 1.2 and is proved by rewriting the proof while taking into account error terms that are caused by the modifications. Due to lack of space, we omit the proof.

claim 1.4 ([2]) *The density of every augmenting tree T_{aug} satisfies:*

$$\rho(T_{aug}) \leq (1+\lambda)^{2h(T_u)} \cdot h(T_u) \cdot \rho(T'_u).$$

The following theorem is proved in [2]. The assumptions on the height and maximum degree are justified by the reduction discussed above.

Theorem 1.3 *Algorithm* Modified-GS$(r, k, \mathcal{G})$ *is a poly-logarithmic approximation algorithm with polynomial running time for GS instances over trees with logarithmic maximum degree and* $O(\log n/\log\log n)$ *height.*

6 Discussion

In this chapter we presented the recursive greedy algorithm and its analysis. The algorithm is designed for problems in which finding a minimum density augmentation of a partial solution is an NP-hard problem. The main advantages of the algorithm are its simplicity and the fact that it is a combinatorial algorithm. The analysis of the approximation ratio of the recursive greedy algorithm is nontrivial and succeeds in bounding the density of the augmentations.

The recursive greedy algorithm has not been highlighted as a general method, but rather as an algorithm for Steiner tree problems. We believe that it can be used to approximate other problems as well.

Open Problems. The quasi-polynomial time $O(\log^3 k)$-approximation algorithm for DST raises the question of finding a polynomial time algorithm with a poly-logarithmic approximation ratio for DST. In particular, the question is whether the running time of the recursive greedy algorithm for DST can be reduced by modifications or preprocessing.

Acknowledgments

I would like to thank Guy Kortsarz for introducing me to the recursive greedy algorithm and sharing his understanding of this algorithm with me. Guy also volunteered to read a draft. I thank Chandra Chekuri for many discussions related to this chapter. Lotem Kaplan listened and read drafts and helped me in the search for simpler explanations. Thanks to the Max-Planck-Institut für Informatik where I had the opportunity to finish writing the chapter. Special thanks to Kurt Mehlhorn and his group for carefully listening to a talk about this chapter.

References

[1] Y. Bartal, M. Charikar, D. Raz. Approximating min-sum k-clustering in metric spaces. *Proc. of STOC*, 11–20, 2001.

[2] Chandra Chekuri, Guy Even, and Guy Kortsarz. A greedy approximation algorithm for the group Steiner problem. To appear in *Discrete Applied Mathematics.*

[3] M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha and M. Li. Approximation Algorithms for directed Steiner Problems. *Journal of Algorithms*, 33, 73–91, 1999.

[4] N. Garg and G. Konjevod and R. Ravi, A polylogarithmic approximation algorithm for the Group Steiner tree problem. *Journal of Algorithms*, 37, 66-84, 2000. Preliminary version in *Proc. of SODA*, 253–259, 1998.

[5] C. H. Helvig, G. Robins, and A. Zelikovsky. Improved approximation scheme for the group Steiner problem. *Networks*, 37(1):8–20, 2001.

[6] G. Kortsarz and D. Peleg. Approximating the Weight of Shallow Steiner Trees. *Discrete Applied Math*, 93, 265-285, 1999.

[7] G. Reich and P. Widmayer. Beyond Steiner's problem: A VLSI oriented generalization. *Proc. of Graph-Theoretic Concepts in Computer Science (WG-89)*, LNCS volume 411, 196–210, 1990.

[8] A. Zelikovsky. A series of approximation algorithms for the acyclic directed Steiner tree problem. *Algorithmica*, 18: 99-110, 1997.

[9] L. Zosin and S. Khuller. On directed Steiner trees. *Proc. of SODA*, 59–63, 2002.

A Proof of Lemma 1.2

We prove that given a k-DST T in a transitive closed directed graph G, there exists a k-DST T' such that: (i) T' is ℓ-layered and (ii) $w(T') \leq \frac{\ell}{2} \cdot k^{2/\ell} \cdot w(T)$.

Notation. Consider a rooted tree T. The subtree of T that consists of the vertices hanging from v is denoted by T_v. Let $\alpha = k^{2/\ell}$. We say that a node $v \in V(T)$ is *α-heavy* if $k(T_v) \geq k(T)/\alpha$. A node v is *α-light* if $k(T_v) < k(T)/\alpha$. A node v is *minimally α-heavy* if v is α-heavy and all its children are α-light. A node v is *maximally α-light* if v is α-light and its parent is α-heavy. Note that if u is minimally α-heavy, then all its children are maximally α-light.

Promotion. We now describe an operation called promotion of a node (and hence the subtree hanging from the node). Let G denote a directed graph that is transitively closed. Let T denote a tree rooted at r that is a subgraph of G. *Promotion* of $v \in V(T)$ is the construction of the rooted tree T' over the same vertex set with the edge set: $E(T') \triangleq E(T) \cup \{(r, v)\} \setminus \{(p(v), v)\}$. The promotion of v simply makes v a child of the root.

Height reduction. The height reduction procedure is listed as Algorithm 3. The algorithm iteratively promotes minimally α-heavy nodes that are not children of the root, until every α-heavy node is a child of the root. The algorithm then proceeds with recursive calls for every maximally α-light node. There are two types of maximally α-light nodes: (1) children of promoted nodes, and (2) α-light children of the root (that have not been promoted).

The analysis of the algorithm is as follows. Let $h_\alpha(k(T))$ denote an upper bound on the height of the returned tree as a function of the number of terminals in T. The recursion is applied only to α-light trees that are one or two edges away from the current root. It follows that $h_\alpha(k(T))$ satisfies the recurrence

$$h_\alpha(k') \leq h_\alpha(k'/\alpha) + 2.$$

Therefore, $h_\alpha(k') \leq 2 \cdot \log_\alpha k'$.

Bounding the weight. We now bound the weight of the tree T' returned by the height reduction algorithm. Note that every edge $e' \in E(T')$ corresponds to a path $\mathit{path}(e') \in T$. We say that an edge $e \in E(T)$ is *charged* by an edge $e' \in E(T')$ if $e \in \mathit{path}(e')$. If we can prove that every edge $e \in E(T)$ is charged at most β times, then $w(T') \leq \beta \cdot w(T)$.

We now prove that every edge $e \in E(T)$ is charged at most $\alpha \cdot \log_\alpha k(T)$ times. It suffices to show that every edge is charged at most α times in each level of the recursion. Since the number of terminals reduces by a factor of at least α in each level of the recursion, the recursion depth is bounded by $\log_\alpha k(T)$. Hence, the bound on the number of times that an edge is charged follows.

Consider an edge $e \in E(T)$ and one level of the recursion. During this level of the recursion, α-heavy nodes are promoted. The subtrees hanging from the promoted nodes are disjoint. Since every such subtree contains at least $k(T)/\alpha$ terminals, it follows that the number of promoted subtrees is at most α. Hence, the number of new edges $(r, v) \in E(T')$ from the root r to a promoted node v is at most α. Each such new edge charges every edge in $E(T)$ at most once, and hence every edge in $E(T)$ is charged at most α times in each recursive call. Note also that the recursive calls in the same level of the recursion are applied to disjoint subtrees. Hence, for every edge $e \in E(T)$, the recursive calls that charge e belong to a single path in the recursion tree.

We conclude that the recursion depth is bounded by $\log_\alpha k(T)$ and an edge is charged at most α times in each recursive call. Set $\ell = 2 \cdot \log_\alpha k(T)$, and then $\alpha \cdot \log_\alpha k(T) = \frac{\ell}{2} \cdot k^{2/\ell}$. The lemma follows. □

Algorithm 3 HR(T, r, α) - A recursive height reduction algorithm. T is a tree rooted at r, and $\alpha > 1$ is a parameter.

```
1: stopping condition: if $V(T) = \{r\}$ then return $(\{r\})$.
2: $T' \leftarrow T$.
3: while $\exists v \in V(T') : v$ is minimally $\alpha$-heavy & $dist(r, v) > 1$ do
4:     $T' \leftarrow promote(T', v)$
5: end while
6: for all maximally $\alpha$-light nodes $v \in V(T')$ do
7:     $T' \leftarrow$ tree obtained from $T'$ after replacing $T'_v$ by HR$(T'_v, v, \alpha)$.
8: end for
9: return $(T')$.
```

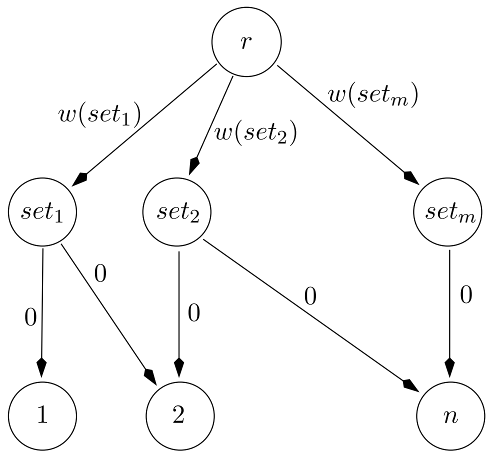

Figure 1.1: Reduction of SC instance to DST instance.

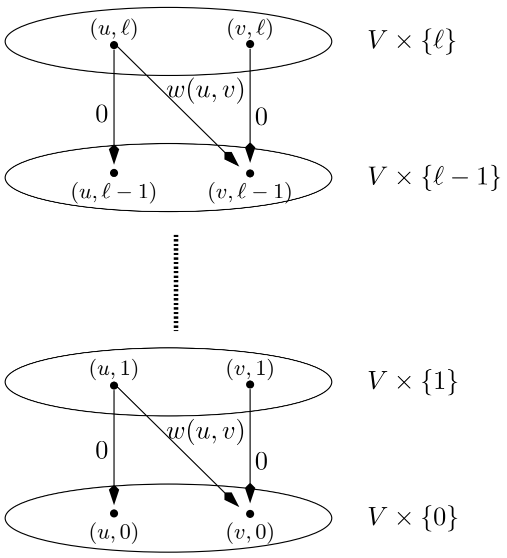

Figure 1.2: Layering of a directed graph G. Only parallel edges incident to images of $u, v \in V(G)$ and regular edges corresponding to $(u, v) \in E(G)$ are depicted.

Index