Optimal Website Design with the Constrained Subtree Selection Problem

Brent Heeringa and Micah Adler

Department of Computer Science
University of Massachusetts, Amherst
140 Governors Drive
Amherst, MA 01003
{heeringa,micah}@cs.umass.edu

Abstract. We introduce the Constrained Subtree Selection (CSS) problem as a model for the optimal design of websites. Given a hierarchy of topics represented as a DAG G and a probability distribution over the topics, we select a subtree of the transitive closure of G which minimizes the expected path cost. We define path cost as the sum of the page costs along a path from the root to a leaf. Page cost, γ, is a function of the number of links on a page. We give a sufficient condition for γ which makes CSS NP-Complete. This result holds even for the uniform probability distribution. We give a polynomial time algorithm for instances of CSS where G does not constrain the choice of subtrees and γ favors pages with at most k links. We show that CSS remains NP-Hard for constant degree DAGs, but also provide an $O(\log(k)\gamma(d+1))$ approximation for any G with maximum degree d, provided that γ favors pages with at most k links. We also give a complete characterization of the optimal trees for two special cases: (1) linear degree cost in unconstrained graphs and uniform probability distributions, and (2) logarithmic degree cost in arbitrary DAGs and uniform probability distributions.

1 The Constrained Subtree Selection Problem

In this paper, we study the optimal design of websites given a set of page topics, weights for the topics, and a hierarchical arrangement of the topics. Automatic website design provides a principled choice for information organization, facilitates individualized and user-centric site layout, and decreases the average time spent searching for relevant information.

As an example, imagine that A Different Drummer's Kitchen is creating a new website for their catalog of kitchenware. They want a website where their customers can quickly find information on specific products by descending a hierarchy of general to specific categories, much like the *Yahoo!* portal. They want to minimize the number of intermediate pages it takes to find pepper mills but not at the expense of filling a page with links to marginally related products like tea kettles, cookie cutters and aprons.

Constrained Subtree Selection (CSS) models these website design problems. We suppose that prior to site development, topics are hierarchically arranged by a designer to represent their natural organization. We represent this initial hierarchy as a rooted, directed acyclic graph, called the *constraint graph* where the nodes are categories, the leaves are topics and the edges are topical constraints. A path through the constraint graph follows a general to specific trajectory through the categories. For example, in the kitchenware hierarchy cutlery leads to knives leads to paring knives. Note that a particular paring knife may belong to other categories (like the knife manufacturer), and thus the constraint graph may be a DAG that is not a directed tree.

A website should preserve this logical relationship in its own topology. We represent websites as directed trees, where pages are represented by nodes and links are represented by directed edges. We require that the directed tree satisfy two conditions. First, there must be a one-to-one mapping $\mathcal{M}$ of nodes in the website to nodes in the constraint graph. This is a constraint since adding new nodes would infer structure that is not represented in the constraint graph. Second, if categories in the constraint graph are not included in the website, a user should still be able to descend naturally toward the desired topic. This means that if page A descends directly from page B in the website then $\mathcal{M}(A)$ must be reachable from $\mathcal{M}(B)$ in the constraint graph. A necessary and sufficient condition for both of these conditions to be satisfied is that the website be a directed subtree of the transitive closure of the constraint graph. In this way, the initial hierarchy offers a set of constraints on topic layout but frees the web site developer to move specific pages to more general categories. Finally, we stipulate that the subtree include the root and leaves of the constraint graph since they represent the entry and endpoints of any natural descent in the website.

Our objective is to find the website which minimizes the expected time searching for a topic. We say the cost of a search is the sum of the cost of the pages along the search path. We represent page cost as a function of the number of links on a page, so we call it the *degree cost*. Adding more links decreases the height of the tree, but increases the time spent searching a page; minimizing the number of links on a page makes finding the right link easy, but adds height to the website. For this reason, we can also think of the degree cost as capturing the inherent tension between breadth and depth. Different scenarios demand different tradeoffs between these competing factors. For example, if network latency is a problem when loading web pages then favoring flatter trees with many links per page decreases idle waiting. In contrast, web browsers on handheld devices have little screen area, so to reduce unnecessary scrolling it's better to decrease the number of links in favor of a deeper tree. In the spirit of generality, we attempt to keep our results degree-cost independent. At times however, we examine particular degree costs such as logarithmic and linear.

Naturally, some pages are more popular than others. We capture this aspect with a probability distribution over the topics, or equivalently by topic weights. Given a path, we say the weighted path cost is the sum of the page costs along the path (i.e. the unweighted path cost) multiplied by the topic weight. Since we want a website that minimizes the average search time for a topic, we take the cost of a tree as the expected path cost for a topic chosen from the probability distribution over the topics. An optimal tree is any minimal cost subtree of the transitive closure of the constraint graph that includes the leaves and root.

We're now in a position to define our model more formally. Let T be a directed tree (a branching) with n leaves where leaf u_i has weight w_i. Let $u_i = (u_{i_1}, \ldots, u_{i_m})$ be a path from the root of T to the i^{th} leaf of T. If $\delta(v)$ is the out-degree of node v and γ is a function from the positive integers to the reals, then the cost of u_i is:

$$c(u_i) = \sum_{j=1}^{m-1} \gamma(\delta(u_{i_j}))$$

and the weighted cost is $w_i \cdot c(u_i)$. The cost of T is the sum of the n weighted paths:

$$c(T) = \sum_{i=1}^{n} w_i \cdot c(u_i)$$

An instance of the *Constrained Subtree Selection* problem is a triple $I = (G, \gamma, (w_i))$ where G is a rooted, directed, acyclic *constraint* graph with n leaves, γ is a function from the positive integers to the non-negative reals, and $(w_i) = (w_1 \ldots w_n)$ are non-negative, real-valued leaf weights summing to one. A solution to I is a directed subtree T (hereafter a tree) of the transitive closure of G that includes the leaves and root of G. An optimal solution is one that minimizes the cost function under γ. Sometimes we consider instances of CSS with fixed components. For example, we might study the problem when the degree cost is always linear, or leaf weights form a uniform probability distribution. We refer to these cases as *CSS with* γ or *CSS with equal leaf weights* so that it is clear that γ and (w_i) are not part of the input.

Websites are not the only realization of this model. For example, consider creating and maintaining user-specific directory structures on a file system. One can imagine that the location of `/etc/httpd` may be promoted to the root directory for a system administrator whereas a developer might find `~/projects/source` directly linked in their home directory. Similarly, users may have individualized views of network filesystems targeted to their own computing habits. In this scenario a canonical version of the network structure is maintained, but the CSS problem is tailored to the individual. In general, any hierarchical environment where individuals actively use the hierarchy to find information invites modeling with CSS.

1.1 Results

In this paper, we give results on the complexity of CSS, polynomial time algorithms and characterizations of the optimal solution for certain restricted instances of CSS, and a polynomial time constant approximation algorithm for fixed-degree constraint graphs in a broad class of degree costs.

First, we show a sufficient condition on the degree cost which makes Constrained Subtree Selection NP-Complete in the strong sense for arbitrary input

DAGs. Many natural degree costs (e.g., linear, exponential, ceiling of the logarithm) meet this condition. Furthermore, this result holds even for the case of uniform leaf weights.

Because of this negative result, we turn our attention to restricted scenarios and approximation algorithms. We first consider the case of inputs where the topological constraints of the graph are removed (i.e., where the constraint graph allows any website tree to be constructed). Within this scenario, we consider a general class of degree functions, called k*-favorable* degree costs, where the optimal solution favors trees such that all the nodes have out-degree k or less. We give an $O(n^{k+\gamma(k)})$ time algorithm for finding an optimal tree when the topological constraints of the graph are removed and when γ is non-decreasing, restricted to functions with integer co-domains, and k-favorable. This result holds for arbitrary leaf weights, and demonstrates that the computational hardness of the CSS problem is a result of the conditions imposed by the constraint graph. We also provide an exact characterization of the optimal solution for the linear cost function (which is 3-favorable) in the case of a uniform probability distribution and no topological constraints.

We next consider the case of bounded out-degree constraint graphs. We demonstrate that when γ favors complete k-ary trees, CSS remains NP-Hard for graphs with degree at most $k+5$ and uniform leaf weights. However, we also give a polynomial time constant factor approximation algorithm for constraint graphs with degree no greater than d and arbitrary leaf weights, provided that γ is k-favorable for some k. The approximation ratio depends on both d and γ. Additionally, we show the linear degree cost favors complete k-ary trees.

Finally, for arbitrary constraint graphs, $\gamma(x) = \lceil \log_2(x) \rceil$, and uniform leaf weights, we demonstrate that even though this case is NP-Complete, the depth-one tree approximates the optimal solution within an additive constant of 1. Due to space constraints, most of the proofs of our results appear in [1].

1.2 Related Work

Constrained Subtree Selection is related to three distinct bodies of work. The first is work in the AI community by Perkowitz and Etzioni [2]. While the authors are concerned with many issues related to building intelligent websites, they concentrate on the *index page synthesis problem* which seeks to "automatically generate index pages to facilitate efficient navigation of a site or to offer a novel view of the site" using new clustering and concept learning algorithms which harness the access logs of the website. Here efficient means making sure visitors find their topic of interest (recall) and minimizing the amount of time spent finding that topic (effort). The time spent finding a topic is measured by the time it takes to scan successive pages for the right link and the overall number of links taken. Notice their definition of effort strongly resembles our notion of cost. In this light, our work may be viewed as supplying a model for the index page synthesis problem as it relates to minimizing the average effort in finding the topic of interest.

The Hotlink Assignment (HA) problem introduced by Bose et. al ([3], [4]) also relates to our problem. Here, a website is represented by a DAG with a probability distribution over the leaves. A constant number of arcs, called hotlinks, are added to the DAG to minimize the expected distance from the root to leaves. Since multiple paths from the root to a leaf may exist, the expected distance is computed using the shortest path. The problem is NP-Hard for arbitrary graphs, but tractable for binary trees with arbitrary probability distributions over the leaves. Recently, the problem was revised so that nodes have a fixed page cost proportional to the size of the web page they represent [5]. In this formulation, the cost of a path is not its length, but instead the sum of the page costs on the path. The problem seeks to assign at most k hotlinks per node to minimize the expected page cost.

Hotlink Assignment (HA) is different from CSS for a number of reasons. The first is how we model page cost. In HA, page cost does not change with the addition of hotlinks. In CSS, the cost of a page is a function of the number of links it contains. This means we can think of CSS as minimizing the expected amount of choice a user faces when traversing a website as opposed to HA which essentially minimizes the expected amount of time waiting for pages to load. Note that the generality of our degree function means we can also include a network latency term in to our degree cost. Another difference is how we view the initial topologies. With HA, the DAG represents a website that needs improving. In CSS, we take the DAG as a set of constraints for building a website. This difference is both conceptual and technical. While the *shortest path* tree can be extracted from the Hotlink DAG after the links are assigned, a tree with longer paths cannot be considered. We consider all paths in our subtree selection since longer paths are viewed in terms of constraints and not cost. Finally, HA assigns a constant number of hotlinks where CSS has no restriction. The constant number is important to HA because without this restriction, the optimal website would always have hotlinks from the root to all the leaves. In CSS this corresponds to a constant degree function where the optimal tree is always the depth-one tree.

Certain relaxed versions of the Constrained Subtree Selection problem bear resemblance to the Optimal Prefix-free Coding (OPC) problem: The general problem asks for a minimal prefix code for n weighted words using at most r symbols where symbol i has cost c_i ([6], [7]). This problem is equivalent to finding a tree with n leaves where all internal nodes having degree at most r, the length of the i^{th} edge of a node is c_i, and the external weighted path length is minimized. There is no known polynomial time solution for the general problem, but it is not known to be NP-Hard. When the costs are restricted to fixed integers, there is an $O(n^{C+2})$ time dynamic programming algorithm where C is the maximum integer cost [8].

On the surface, our problems appear similar because they both ask to minimize external weighted path cost—the sum of weighted path costs from the root to each of the leaves. However the cost in OPC is edge-based, where the cost of CSS is node-based. More appropriately, the node cost in CSS is dynamic; adding an additional edge means the cost of the node changes. If we view the

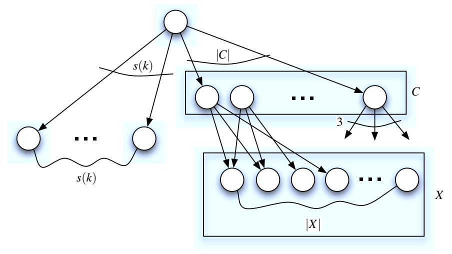

Fig. 1. Setting up the constraint graph given an instance of Exact Cover by 3-sets.

node costs as edge costs, than adding an edge potentially changes the edge costs of all its siblings. This difference, along with the lack of prior constraints on the tree structure in prefix-free codes, distinguish the problems enough that it seems difficult to transform one to the other. Still, by relaxing the graph constraints, and restricting the degree cost, we can show that some instances of CSS are exactly instances of OPC for a binary alphabet with equal character costs, and that in more general cases, we can adapt portions of the dynamic programming algorithm for finding optimal prefix-free codes to our find optimal trees in the CSS problem.

2 Complexity

In this section we show that even when the leaf weights are equal, the CSS problem is NP-Complete in the strong sense for a large class of degree functions. The reduction is from Exact Cover by 3-Sets (XC3) [9] which, when given a set X of $3k = n$ items and a set C of three item subsets of X, asks whether a subset of C exists that exactly covers X. The related decision problem for CSS asks whether a subtree of G exists with cost at most D.

Definition 1. *Let γ be a non-decreasing function. If for all $k \geq 1$, $k \in \mathbb{Z}^+$ there exists some $c > 0$ and some function $s(k) \in O(k^c)$ such that*

$$\gamma(s(k) + k + 1) > \gamma(s(k) + k) + \gamma(3)\frac{3k}{s(k) + 3k}$$

then γ is degree-3-increasing

Many degree costs are degree-3-increasing. For example, the linear degree cost, $\gamma(x) = x$, (choose $s(k) = 7k$), exponential degree cost $\gamma(x) = \exp(x)$ (again, $s(k) = 7k$ will work) and ceiling of the logarithm degree cost $\gamma(x) = \lceil \log_2(x) \rceil$ (choose $s(k) = 3k$) all meet the definition. The following theorem tells us that when γ is degree-3-increasing and in NP, that CSS with γ is NP-complete for any DAG and any probability distribution.

Theorem 1. *For any degree-3-increasing degree cost γ where γ is in NP, CSS with γ is NP-Complete.*

Proof. The theorem follows immediately from the following lemmas.

Lemma 1. *For any degree-3-increasing degree cost γ, CSS with γ is NP-Hard*

Proof. The reduction is from Exact Cover by 3-sets (XC3). Let (X, C) be an instance of XC3 where $|X| = 3k = n$ with k a positive integer, and C is a collection of 3-element subsets of X. Furthermore, let γ be degree-3-increasing. We will build a constraint graph such that (X, C) has an exact cover if and only if we can find a subtree with cost

$$D = (s(k) + n)\gamma(s(k) + k) + n\gamma(3)$$

The idea is to create a constraint graph that affords a low cost solution when an exact cover exists, but increases in cost when no exact cover is available. Keeping this in mind, construct the constraint graph $G = (V, E)$ from (X, C) in the following way:

- Make a leaf node for each $x \in X$
- Make an interior node for each $c \in C$
- Make $s(k)$ additional leaf nodes.
- Make a root node r and connect it to the $s(k)$ leaf nodes and the c interior nodes.
- For each $c = \{x_i, x_j, x_k\} \in C$, create edges from c to x_i, x_j, and x_k.

The topology of the constraint graph is given pictorially in Fig. 1. It has $s(k)+n$ leaf nodes and $|C|$ internal nodes, each having degree 3. Let $(G, \gamma, (w_i))$ be an instance of CSS where $w_i = 1$ for all i, G is the constraint graph formed from (X, C), and γ is degree-3-increasing.

Suppose (X, C) has an exact cover, then choose a subtree T from G by selecting the subtree with paths to the k interior nodes of G that partition X, along with their corresponding paths to the n leaf nodes. T must also have $s(k)$ paths to the remaining $s(k)$ leaf nodes. T has a root with degree $s(k) + k$ and k interior nodes each having degree 3. The length 1 paths each cost $\gamma(s(k) + k)$ and the n remaining paths, each have cost $\gamma(s(k) + k) + \gamma(3)$ making the total cost

$$(s(k) + n)\gamma(s(k) + k) + n\gamma(3)$$

Now suppose (X, C) does not have an exact cover. Any subtree of the transitive closure of G must have $s(k)$ edges to the $s(k)$ leaf nodes and at least $k + 1$ additional edges to reach the remaining n leaves. Let e be the number of edges leading directly from the root to some subset of the remaining n leaves; let f be the number of edges from the root an interior node with degree 2 and; let g be the number of edges from the root to an interior node with degree 3. Note that $e + f + g \geq k + 1$ and $e + 2f + 3g = n$. The cost of T is

$$(s(k) + n)\gamma(s(k) + e + f + g) + 2f\gamma(2) + 3g\gamma(3)$$

which, because γ is non-decreasing, is greater than or equal to

$$\begin{aligned}(s(k)+n)\gamma\Big(s(k)+k+1\Big) &> (s(k)+n)\Big(\gamma(s(k)+k)+\gamma(3)\frac{n}{s(k)+n}\Big)\\ &= (s(k)+n)\gamma(s(k)+k)+n\gamma(3)\end{aligned}$$

which is the cost of the subtree when an exact cover exists.

Lemma 2. *For any degree cost γ in NP, CSS with γ is in NP.*

Proof. Let $I = (G, \gamma, (w_i))$ be an instance of CSS with γ in NP. Let T be a solution to I and D the cost of T. We can use the polynomial time verifiers for the out-degree of each node in T (since γ is in NP) to confirm the cost of each node. WIth the costs of the nodes in hand, it's straightforward to compute the cost of T and verify that it is indeed D.

□

Because CSS is not a number problem when the leaf weights are equal (i.e. we can ignore them when computing cost), we can show that it is NP-Complete in the strong sense for a broad class of degree costs.

Theorem 2. *For any degree-3-increasing degree cost γ, γ in NP, if there exists $c > 0$ such that $\gamma(s(n/3) + n/3)$ in $O(n^c)$ then CSS with γ is NP-Complete in the strong sense.*

Proof. From Lemma 1 it's clear CSS remains NP-Hard when the n leaf weights are equal and since we γ is fixed, we need not worry about its encoding. The only number we need to encode is the bound on cost D. Since D is bounded by a polynomial in n, we know the magnitude of the maximum element can be bound above above by a polynomial factor of the encoding of the instance. Since γ is degree-3-increasing and in NP we have that CSS remains in NP and the problem is NP-Complete in the strong sense. Note that many degree costs make D polynomial in n. For example, $\gamma(x) = x$ keeps D quadratic in n. □

Finally, we note (without proof) that CSS is NP-Hard for many costs which are not degree-3-increasing (e.g., the logarithmic degree cost) when considering problem instances with arbitrary probability distribution.

3 Subtree Selection without Constraints

Imagine we are building a website without any prior knowledge of the organization of the topics. The most natural solution is to build a website that minimizes the expected search time for the topics, but has no constraints on the topology. This design problem is an instance of CSS where any website is a subtree of the transitive closure of the constraint graph. In this section we'll show that these instances are solvable in polynomial time for a broad class of degree functions. This is interesting because it means the NP-Hardness of our problem comes from the graphical constraints rather than the degree cost and leaf weights.

The unconstrained CSS problem also bears a resemblance to the Optimal Prefix-free Code (OPC) problem. Both CSS and OPC attempt to minimize external weighted path cost, but with CSS, path cost changes with a change in node degree. This distinction, along with the topological constraints, prevents us from casting CSS as an Optimal Prefix-free Code problem. Still, in many circumstances we can show instances of CSS that equate to finding optimal Huffman codes and that in more general cases, we can give a polynomial time solution by adapting Golin and Rote's dynamic programming algorithm for optimal prefix-free codes with integer weights.

We begin with some definitions.

Definition 2 (full tree). *A tree is* full *when every interior node has at least two children.*

Definition 3 (constraint-free graphs). *A constraint graph G with n leaves is called* constraint-free *when every full tree with n leaves is a subtree of the transitive closure of G*

Definition 4 (monotone tree). *A tree is* monotone *when the leaf weights cannot be permuted (among the leaves) to yield a tree of less cost.*

Definition 2 is self-explanatory. Definition 3 means that G does not constrain the optimal subtree. Definition 4 says if we listed the leaves in increasing order by path cost, the weights of the leaves would be in decreasing order. From these definitions it's easy to see that every instance of CSS has at least one optimal solution that is full and that all solutions to CSS are monotone when the the graph is constraint-free.

The following definition is also useful because it gives us a bound on the out-degree of any node in an optimal solution to the CSS problem where the graph is constraint-free.

Definition 5 (k-favorability). *A degree cost γ is k-*favorable *if and only if there exists $k > 0$ such that any instance of CSS where G is constraint-free has an optimal solution under γ where the out-degree of every node is at most k.*

Many degree costs are k-favorable. For example, the linear degree function, $\gamma(x) = x$ is 3-favorable.

Lemma 3. *The linear degree cost, $\gamma(x) = x$, is 3-favorable.*

Proof. Let $I = (G, \gamma, (w_i))$ be an instance of CSS where G is constraint-free and $\gamma(x) = x$ and T an optimal tree for I. When $\delta(v) \geq 4$ for some node v in T we can split the node's children proportionally among two additional nodes, connect them to the original root and not increase the cost.

Let T be an optimal tree induced from a constraint-free graph with n leaves under $\gamma(x) = x$. When the degree of a node v is greater than or equal to 4 we can always split the node's children proportionally among two additional nodes, connect them to the original root and never increase the cost since:

$$\gamma(x) \geq p(\gamma(\lceil x/2 \rceil) + \gamma(2)) + (1-p)(\gamma(\lfloor x/2 \rfloor) + \gamma(2))$$

where p is the proportion of weight in the left child of the root and $(1-p)$ is proportion of weight in the right child of the root.

Interestingly, the linear degree cost is not 2-favorable. When a node has out-degree 3 and none of the three subtrees has probability greater than the sum of the remaining two subtrees (i.e. the weight of the subtrees is somewhat uniform), we cannot transform the degree 3 node to a series of degree 2 nodes without increasing the overall cost of the tree. To see this, imagine the binary tree with three leaf nodes. Two paths from the root have length 2 and one path has length 1. The length 2 paths each have total cost 4, while the length 1 path has total cost 2. If one of the leaf nodes has weight over 1/2 we can choose it as the leaf on the length 1 path and the average path cost falls below 3, however when no such node exists it is advantageous to create a single node with three paths, each having cost 3 making the average cost 3 as well. It is worth noting that any instance of CSS where G is constraint-free and γ is 2-favorable reduces to the optimal prefix code problem for a binary alphabet with equal letter costs. In other words, Huffman's greedy algorithm (see [10]) solves these problems. Examples of degree costs that favor binary trees are $\gamma(x) = \lceil \log(x) \rceil$ and $\gamma(x) = e^x$.

But what happens when γ is k-favorable but not $k-1$-favorable and $k > 2$? Asked differently, is there a polynomial time algorithm that solves $(G, \gamma, (w_i))$ when G is constraint-free and γ is k-favorable? If we restrict γ to be non-decreasing from the positive integers to the positive integers, then we can give an $O(n^{\gamma(k)+k})$ dynamic programming algorithm for finding the optimal tree.

We adapt the dynamic programming algorithm for finding optimal prefix-free codes given by Golin and Rote ([8]) to the CSS problem when G is constraint-free and γ is k-favorable, non-decreasing and maps the positive integers to the positive integers. This adaptation is possible for many reasons. First, we have an upper bound on cost when γ is non-decreasing and k-favorable. In prefix-free codes, this equates to the character with largest cost. Second, the dynamic programming algorithm works for integer character costs. Since γ maps the positive integers to the positive integers, all the path costs take on integer values. Third, prefix-free codes are monotonic and correspond to full, monotone trees. Since G is constraint-free, we know that optimal subtrees exist which are full and monotone. Fourth, in the unconstrained problem any two trees with leaves having identical non-weighted path costs have equal total cost since the optimal tree is monotone. This equivalence on trees helps reduce the search space in both OPC and CSS, however the nature of search is different in the problems, creating a divergence between the OPC and CSS algorithms.

When considering the cost of a tree, we can push a node's degree cost to its edges and view the tree as lopsided [11]. With lopsided trees, a node's depth is equal to its path cost from the root. For example, a lopsided tree for the linear degree cost is given in Fig. 2 (a). We can view our trees as lopsided too, but

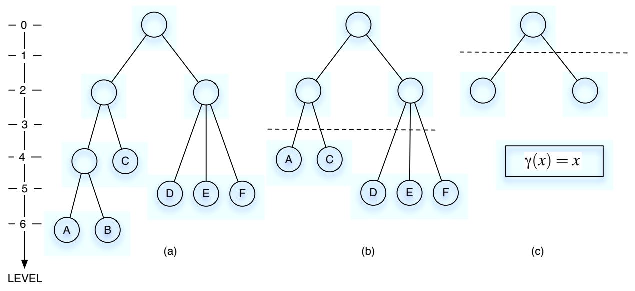

Fig. 2. (a) A tree with 6 leaves depicted as a lopsided tree under the linear degree cost. Truncations at level-3 (b) and level-1 (c) of the tree in (a).

prefer to retain the traditional meaning of depth (the number of edges between a node and the root) and use *level* to indicate a node's path cost. In other words, if $\mathcal{C}$ is the path cost from the root of T to some node v in T, then v is at *level* $\mathcal{C}$ and $\text{level}(v) = \mathcal{C}$.

Golin and Rote associate a signature with each tree level so that if a tree has l levels, then it has l signatures. A signature corresponds to an entry in their dynamic programming table. Each entry records the minimum cost of all trees bearing that signature. Each cost can be written in terms of the cost of the tree associated with the signature that precedes it. Likewise, given a tree's signature at level i, it's possible to enumerate what other signatures follow it, in essence, finding all the trees whose signature at level $(i + 1)$ comes from a tree matching the signature at level i. This provides a natural method for filling in the dynamic programming table.

Our dynamic programming table is similar to Golin and Rote's but we are forced to fill in the entries differently. Notice that in our problem decreasing the out-degree of a node may significantly change the structure of the lopsided tree—adding or removing an edge from a node changes the cost of the entire subtree rooted at that node. With prefix-free trees, adding or removing an edge from a node affects only the edge and its descendants. Golin and Rote exploit these properties when filling in table entries. We must take a different approach. For clarity and completeness, we give adapted definitions for tree *signature*, *truncation*, and *cost* first presented in [8].

Definition 6. *Let T be a tree. The* level-i-truncation *of T, denoted $Trunc_i(T)$, prunes away all nodes of T with parents at levels deeper than i.*

For example, the level-3- and level-1-truncations of the tree in Fig. 2 (a) are given in Figs. 2 (b) and (c) respectively. Truncation helps us define tree cost and tree signature:

Definition 7. *Let T be a tree with n nodes, $v_1, \ldots, v_n$, given by level in increasing order. Let $w_1 \ldots, w_n$ be the leaf weights given in decreasing order. The* level-i-cost *of T is*

$$c_i(T) = \sum_{j=1}^{m} \mathit{level}(v_j) w_j + \sum_{s=m+1}^{n} i \cdot w_s$$

where m is the number of leaf nodes in $\mathit{Trunc}_i(T)$.

It's easy to see that the level-i-cost of a tree is a lower bound on the overall cost of a tree. It records the exact cost of the $m \leq n$ leaves but only the cost to level i of the remaining $n - m$ leaves. When $m = n$, the cost of the tree is the same as the level-i-cost. We associate the cost of a tree with its signature:

Definition 8. *Let T be a tree. The* level-i-signature *of T is the $(\gamma(k)+1)$ vector:*

$$\mathit{sig}_i(T) = (m, l_1, \ldots, l_{\gamma(k)})$$

where m is the number of leaf nodes at levels 0 through i and l_j is the number of nodes at level $i + j$ in $\mathit{Trunc}_i(T)$.

For example, the level-4-signature of the tree in Fig. 2 (a) is $(1, 3, 2, 0)$ and the level-3-signature of the same tree is $(0, 2, 3, 0)$. We equate the signature of a tree $(m, l_1, \ldots, l_{\gamma(k)})$ with an entry in the dynamic programming table MIN$[m, l_1, \ldots, l_{\gamma(k)}]$. This entry gives the minimum cost of all trees with signature $(m, l_1, \ldots, l_{\gamma(k)})$. Filling in the table is tantamount to finding the ways two trees with the same signature at level i can differ in their level-$(i + 1)$-signature.

Golin and Rote show that if T is a tree with level-i-signature $\mathrm{sig}_i(T) = (m, l_1, \ldots, l_{\gamma(k)})$, and T' is a tree such that $\mathrm{Trunc}_i(T') = \mathrm{Trunc}_i(T)$ then $\mathrm{Trunc}_{i+1}(T')$ differs from T in at most l_1 ways. This is because nodes at levels-$(i + 2)$ and deeper cannot have children under a level-$(i + 1)$-truncation—only the nodes at level $(i + 1)$ are candidates. Some number q of these nodes are internal, and the choice of q uniquely determines the signature at level $(i + 1)$. This is not true of our problem. Because our node cost is dynamic, we must develop a process quite different than the ones used for the OPC problem. We are left then, with not only choosing how many of the level $(i + 1)$ nodes will be internal, but explicitly choosing among those, which will have degree 2, degree 3, and so on. We denote these choices with a $(k + 1)$-vector called a *child vector*:

Definition 9. *Let T be a tree with $\mathit{sig}_i(T) = (m, l_1, \ldots, l_{\gamma(k)})$. Let $a = (a_0, \ldots, a_k)$ be a* level-i-child vector *of T where a_0 is the number of nodes at level-$(i+1)$ that are internal to T and each a_j is the number among those a_0 having degree j.*

Definition 9 gives us a way to talk about how signatures at level i relate to signatures at level $(i + 1)$. First, note that $a_0 \leq l_1$ and that $a_1 = 0$ since there is always an optimal tree without single out-degree nodes. Also, since $\sum_{j=2}^{k} a_j = a_0$ we know there are $O(n^{k-1})$ choices for a. In other words, given a level-i-signature, it is the possible parent of $O(n^{k-1})$ level-$(i+1)$-signatures. The following Lemma tells us exactly which signatures are children of the level-i-signature.

Lemma 4. *Let T be a tree with $sig_i(T) = (m, l_1, \dots, l_{\gamma(k)})$ and $a = (a_0, a_1, \dots, a_k)$ be the level-i-child vector of T yielding T', then $sig_{i+1}(T') = (m', l'_1, \dots, l'_{\gamma(k)})$ where*

$$(m', l'_1, \dots, l'_{\gamma(k)}) = (m + l_1, l_2, \dots, l_{\gamma(k)}, 0) + b$$

and $b = (b_0, \dots, b_{\gamma(k)})$ where $b_0 = -a_0$ and $b_{\gamma(i)} = i \cdot a_i$ for $2 \le i \le k$

Proof. Let T and be a tree with $\text{sig}_i(T) = (m, l_1, \dots, l_{\gamma(k)})$ and $a = (a_0, \dots, a_k)$ be the level-i-child vector of T yielding T'. Given a it is straightforward to compute $\text{sig}_{i+1}(T')$: We know exactly how many of the l_1 nodes are internal nodes at level $(i+1)$ and the distribution of degrees among them. To calculate $\text{sig}_{i+1}(T') = (m', l'_1, \dots, l'_{\gamma(k)})$ we shift the levels from $\text{sig}_i(T)$ to the left one index, subtract away the number of internal nodes a_0 from $m + l_1$, and then use a to count the new nodes at each level. Note that if there are a_j degree-j nodes then we must add an additional $j \cdot a_j$ nodes to level $\gamma(j)$. This gives us the $(\gamma(k)+1)$-vector:

$$b = (-a_0, b_1, \dots, b_{\gamma(k)}) \text{ where } b_{\gamma(j)} = j \cdot a_j \text{ for } 2 \le j \le k$$

Note that $b_0 = -a_0$ since a_0 of the nodes are internal. Adding b component wise to the shifted signature of T at level i gives us:

$$(m', l'_1, \dots, l'_{\gamma(k)}) = (m + l_1, l_2, \dots, l_{\gamma(k)}, 0) + b$$

□

As an example, consider the tree in Fig. 2 (a) and its level-3-signature $(0, 2, 3, 0)$. The level-i-child vector for this tree is $(1, 0, 1, 0)$ since one of l_1 the nodes is internal, and it is a degree 2 node, we have $b = (-1, 0, 2, 0)$. Shifting the level-3-signature and adding b gives us $(2, 3, 0, 0) + (-1, 0, 2, 0) = (1, 3, 2, 0)$ which is exactly the level-4-signature.

While Lemma 4 tells us how level-i-signatures relate to level-$(i{+}1)$-signatures, it does not tell us how the costs relate. The second part of Lemma 5 from [8] tells us that if T is a tree with $\text{sig}_i(T) = (m, l_1, \dots, l_{\gamma(k)})$ then

$$c_{i+1}(T) = c_i(T) + \sum_{j=m+1}^{n} w_j \tag{1}$$

Fortunately, this result holds for all monotone, lopsided trees with level-i-costs defined as we did in Def. 7. To see why, recall that the level-i-cost of T gives the exact weighted cost of all m leaves at or shallower than level-i in addition to the cost up to level i of the remaining $n - m$ leaves. When moving from level-i to level-$(i+1)$, we know $l_1 - a_0$ of the nodes become leaves and we need to record their exact cost, but the remaining $n - m - (l_1 - a_0)$ leaves must also incremented by a level in the cost structure. Taken together, we need to update the $n - m$ deepest paths which gives us the summation term in 1.

We've now established a method for filling in the dynamic programming table. What's left to give is an ordering of the table entries that is consistent with their dependency structure. Golin and Rote give a linear ordering of the table entries and demonstrate that it respects the dependencies. This ordering works for our problem too, but their proof of this fact no longer applies because our table entries have a different dependency structure. We repeat the ordering here and prove that under it, we never expand a node until all its parents have been processed.

Definition 10. *Let $S = (m, l_1, \ldots, l_{\gamma(k)})$ and $S' = (m', l'_1, \ldots, l'_{\gamma(k)})$ We say that $S \ll S'$ if and only if*

$$(m + l_1 + \cdots + l_{\gamma(k)}, m + l_1 + \cdots + l_{\gamma(k)-1}, \ldots, m + l_1, m)$$

is lexicographically smaller than

$$(m' + l'_1 + \cdots + l'_{\gamma(k)}, m' + l'_1 + \cdots + l'_{\gamma(k)-1}, \ldots, m' + l'_1, m')$$

As an example, compare the vector for the level-3-signature of Fig. 2 (a) (5,5,2,0) with the level-4-signature of Fig. 2 (a) (6,6,4,1). Since $5 < 6$ we would expand the level-3-signature before the level-4-signature. If the first positions of the vectors had the same magnitude we would compare the the second positions and so on. This ordering guarantees that we will never expand a signature until all its parents are expanded. As stated in the main text, we cannot apply Golin and Rote's result because of the differences in dependency structure between OPC and CSS.

Lemma 5. *Let $S = (m, l_1, \ldots, l_{\gamma(k)})$ and $S' = (m', l'_1, \ldots, l'_{\gamma(k)})$. If $a = (a_0, \ldots, a_k)$ takes S to S' then $S \ll S'$.*

Proof. Let $S = (m, l_1, \ldots, l_{\gamma(k)})$ and $S' = (m', l'_1, \ldots, l'_{\gamma(k)})$ such that S yields S' with level-i-child vector $a = (a_0, \ldots, a_k)$. If $a_0 > 0$ then $m' + l'_1 + \cdots + l'_{\gamma(k)} > m + l_1 + \cdots + l_{\gamma(k)}$ since a_0 of the l_1 nodes are replaced by at least $2 \cdot a_0$ nodes so $S \ll S'$. If $a_0 = 0$ then each term s'_i in S' is greater than or equal to its corresponding term s_i in S with at least one of the final terms in S' exclusively greater since at least one of the l_i terms is non-zero. Again, in this case we have $S \ll S'$. $\square$

Since we have a consistent ordering for filling in the table entries, as well as a method for knowing the dependencies among the entries, we can build trees, level by level, using the level-i-signatures. A description of the algorithm is given in the Appendix under Fig. 3. Note that all the table entries are initially set to ∞ save the entries of all the depth-one trees, which we set to 0. As previously shown, checking the dependencies for a table entry takes time $O(n^{k-1})$ and there are $O(n^{\gamma(k)+1})$ entries to check, so the total time of the algorithm is $O(n^{\gamma(k)+k})$. Finally, note that we store the child vectors which lead to the minimum cost for each level in the EXP table entries. We can recreate the tree corresponding to the optimal cost by using the child vectors to backtrack through the table entries.

```
Initialization
$\mathrm{MIN}[m, l_1, \ldots, l_{\gamma(k)}] \leftarrow \infty$ for all $m + l_1 + \cdots + l_{\gamma(k)} \leq n$
for $j \leftarrow 2$ to $\gamma(k)$
    do $\mathrm{MIN}[m, l, 0, \ldots, 0] \leftarrow 0$ for $l = 2$ to $\gamma(k)$
Body
Foreach $(m, l_1, \ldots, l_{\gamma(k)})$ in lexicographic order from $(0, 2, 0, \ldots, 0)$ to $(n, 0, \ldots, 0)$
    do $cost \leftarrow \mathrm{MIN}[m, l_1, \ldots, l_{\gamma(k)}] + \sum_{s=m+1}^{n} w_s$
        Foreach expansion vector $(a_0, \ldots, a_k)$ where
            $a_0 \leq l_1$, $a_1 = 0$, and $\sum_{t=2}^{k} a_t = a_0$
            do $b = (b_0, \ldots, b_{\gamma(k)})$ where
                $b_0 = -a_0$ and $b_{\gamma(i)} = i \cdot a_i$ for $2 \leq i \leq k$
                $(m', l'_1, \ldots, l'_{\gamma(k)}) \leftarrow (m + l_1, l_2, \ldots, l_{\gamma(k)}, 0) + b$
                if $m' + l_1^l + \cdots + l'_{\gamma(k)} \leq n$
                    then $\mathrm{MIN}[m', l'_1, \ldots, l'_{\gamma(k)}] \leftarrow \min(\mathrm{MIN}[m', l'_1, \ldots, l'_{\gamma(k)}], cost)$
                         $\mathrm{EXP}[m', l'_1, \ldots, l'_{\gamma(k)}] \leftarrow (a_0, \ldots, a_k)$
$\mathrm{MIN}[n, 0, \ldots, 0]$ is the cost of the optimal subtree
$\mathrm{EXP}[n, 0, \ldots, 0]$ tells us how to create the optimal subtree
```

Fig. 3. The algorithm for finding minimal cost subtrees with k-favorable degree costs from constraint-free graphs with arbitrary leaf weights

4 Approximations

Many hierarchies have the property that no category has more than a constant number of subcategories. This means the out-degree of every node in the constraint graph is bounded above by a constant. In this section we give two theorems dealing with such cases. The first theorem says that even if we restrict the problem to DAGs of constant maximum degree, CSS remains NP-Hard for certain degree costs. The second theorem gives an $O(\log(k)\gamma(d+1))$ approximation algorithm for all instances of CSS where the maximum degree of the constraint graph is bounded above by some constant d, and γ is k-favorable and has a lower bound of 1.

Let a cost function be *k-tree optimal* if, for all instances of CSS with constraint-free graphs and equal leaf weights, the unique optimal website tree with k^c leaves, for any positive integer c, is a complete k-ary tree of depth c. For example, in subsection 5.1 we show that the linear degree function is 3-tree optimal.

Theorem 3. *For any cost function that is k-tree optimal, for any $k \geq 3$, the CSS problem is NP-Hard even when restricted to the uniform probability distribution and DAGs with degree at most $k+5$.*

Proof. Consider the *Partitioned Exact Cover by 3 Sets* problem, which we define here, and abbreviate by PX3S. The input is a set S of $3q$ elements, where q is an integer, a collection C of subsets of S of size 3, and a partition P of the collection C into exactly q cells. We ask whether there is an exact cover of S that uses exactly one subset from each cell of P.

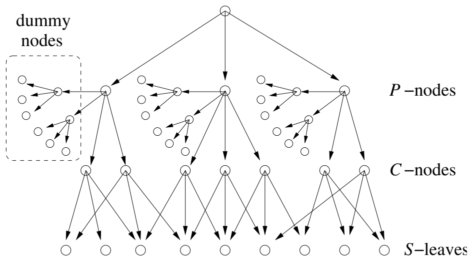

Fig. 4. An example DAG construction for the case where $k = 3$ and q is a power of 3.

The proof is in two parts. We first show that the PX3S problem is reducible to the CSS problem with a k-tree optimal cost function, restricted to DAGs of degree at most $k + r - 1$, where r is the maximum number of subsets in any cell of the partition P. We then show that the PX3S problem is NP-Complete even when we restrict r to six.

We construct a CSS problem with S leaves augmented by a number of dummy leaves to be made precise below. We refer to the former as S-leaves. We use the uniform probability distribution. The DAG we use has a single node for each cell of the partition P, called a P-node, as well as a single node for each of the subsets in C, called a C-node. Each P-node x points to the (at most r) C-nodes that belong to the cell for x. In addition, x points to $k - 1$ dummy nodes only used by x, and each of these in turn point to k distinct dummy leaves. There are also an additional $k - 3$ dummy nodes for x. Let y be a C node pointed to by x. y points to these $k - 3$ dummy nodes, as well as to the 3 S-leaves for the subset represented by y.

If q is a power of k, then we are done merely by connecting to the P-nodes using a complete k-ary tree of depth $\log_3 q$. See Figure 4 for an example of this. If not, let t be the smallest power of k such that $t > q$. We add $t - q$ complete k-ary trees of height two to the DAG, and then connect to these nodes as well as the P-nodes using a complete k-ary tree of depth $\lceil \log_3 q \rceil$.

Claim. This DAG has a complete k-ary tree as a subgraph iff the original PX3S input was a YES instance.

To see that this claim is true, note that a YES instance of PX3S can be converted into a complete k-ary tree simply by having each P-node point to the single C-node that corresponds to the subset of S included in the solution to the PX3C input. Each P-node also points to all of its $k - 1$ child dummy nodes, which in turn point to their dummy leaves.

For the other direction of the if and only if, note that for any possible complete k-ary tree, the distance from the root to any S-leaf must be exactly $\lceil \log_k q \rceil$. This implies that each S leaf must have a parent that is a C-node. Thus, to have a complete k-ary tree, each P-node must have exactly one C-node as a child. The set of such C-nodes defines a solution to the PX3S problem.

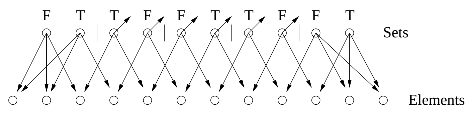

Fig. 5. An example variable structure. The sets (in the top row) point to the elements (the bottom row) that they contain. This does not include the elements for each clause. Note also that the partition of the sets is depicted.

From this claim, we see that PX3S reduces to CSS, since the complete k-ary tree is the unique optimal solution to the unconstrained version of the problem, and thus cannot be matched by any other solution.

We next show that PX3S is NP-Complete by a reduction from Not-All-Equal-3-SAT (denoted here by NE3SAT). In order to do so, we start by taking the NE3SAT input ϕ, and converting it to the formula ϕ', where each clause in ϕ appears in ϕ' twice: once as it is in ϕ, and once with each literal negated from its value in ϕ. It is easy to see that ϕ' is valid iff ϕ is valid. Furthermore, we shall take advantage of the facts that (a) each variable in ϕ' appears as many times negated as unnegated, and (b) each such pair of clauses in ϕ' must have exactly 3 true literals in any valid assignment to the variables.

We next describe a variable structure used for each variable x. Let $n(x)$ be the number of appearances of x in the formula (exactly half of which will be negated). There will be $n(x)+4$ 3-sets for x, as well as $n(x)+6$ elements of S for x. We denote the 3-sets $c_1,\ldots,c_\ell$, $\ell=n(x)+4$, and the elements $s_1,\ldots,s_m$, $m=n(x)+6$. The set c_1 contains s_1, s_2 and s_3. The set c_2 contains s_1, s_2 and s_4. The set $c_{\ell-1}$ contains s_{m-3}, s_{m-1} and s_m. The set c_ℓ contains s_{m-2}, s_{m-1} and s_m. For $3 \le i \le \ell-2$, c_i contains s_i and s_{i+2}, as well as a third element to be described later. An example of this is depicted in Figure 5.

It is now not difficult to see that in any possible exact three cover, it must be the case that the cover includes either exactly those sets c_j such that $j = 1 \bmod 4$ or $j = 0 \bmod 4$, or those sets c_j such that $j = 2 \bmod 4$ or $j = 3 \bmod 4$. The first of these will represent that variable being set to FALSE, and the second will represent that variable being set to TRUE. We partition the sets $c_1,\ldots,c_\ell$ into consecutive pairs; each of these pairs forms one cell of the final partition P.

For each pair of clauses in ϕ' (that corresponded to a single clause in ϕ), we have a set of six elements: one for each literal. Each of these elements is used as the remaining element of a set c_i, for $3 \le i \le \ell-2$. In particular, if the literal is $\overline{x}$, then we use a set c_i for the variable x such that $i = 1 \bmod 4$ or $i = 0 \bmod 4$. If the literal is x, then we use a set c_i for the variable x such that $i = 2 \bmod 4$ or $i = 3 \bmod 4$.

Finally, for each pair of clauses we have a set of six sets that form a single cell of the partition C. Since the literals in the first clause in the pair are negated versions of the literals in the second clause, there are exactly six ways to pick three literals set to false in a valid assignment of variables in this pair of clauses. Each of these last six sets contains the three elements corresponding to the false literals for one of these six settings.

It is now not difficult to see that there is an exact three cover for our constructed PC3S problem if and only if there is a valid (not all equal) assignment to the variables of ϕ. Furthermore, each cell of the partition P has at most 6 subsets.

Theorem 4. *For any constraint graph G with m nodes where every node has out-degree at most d and for every k-favorable degree cost γ where γ is bounded below by 1, CSS with G and γ has a $O(m^2)$ time $O(\log(k)\gamma(d+1))$-approximation to the optimal solution.*

Proof. We begin by giving a lower bound on any instance of CSS where the degree cost is k-favorable and bounded below by 1. Take W as the probability distribution over leaf weights, $W(x)$ as the total weight of the leaves in the subtree rooted at x and H as the entropy function.

Lemma 6. *For any k-favorable degree cost γ with γ bounded below by 1, $\frac{H(W)}{\log(k)}$ is a lower bound on the cost of an optimal solution to CSS with γ.*

Proof. Let $I = (G, \gamma, (w_i))$ be an instance of CSS where γ is k-favorable, and $\gamma(x) \geq 1$ for all $x \geq 1$. Let T be an optimal tree for I. Now consider the tree T' corresponding to the optimal prefix-free code for n words weighted by (w_i) using a k-character alphabet where the cost of each character is 1. By Shannon's theorem, the expected path length of T' is bounded below by $\frac{H(W)}{\log(k)}$. If c is the cost function under γ and c' is the cost function for $\gamma(x) = 1$ then $c'(T)$ is a lower bound on $c(T)$. But $c'(T')$ must be a lower bound on $c'(T)$, so $\frac{H(W)}{\log(k)}$ is a lower bound on $c(T)$.

Our approximation algorithm also requires the following result.

Lemma 7. *For any tree with with weights on its m nodes, there exists one node, which, when removed, divides the tree into subtrees where every subtree has at most half the weight of original tree. Furthermore we can find this node in $O(m)$ time.*

Proof. Let T be a tree with weights on its m nodes. Let $\mathcal{W}$ be the sum of all the weights. If we do a pre-order traversal of the tree, assigning interior nodes an additional new weight which is the sum of the additional weights of its children plus its own regular weight, then the first node we encounter with new weight exceeding half of $\mathcal{W}$ should be removed.

Claim. The node we remove divides the tree into subtrees where each subtree has at most half the weight of the original tree.

It's clear the children of the removed node are all roots of new subtrees, each with weight less than half of $\mathcal{W}$ since their parent is the first node in the pre-order traversal having exceeding half the total weight. Likewise, the root of the original tree forms a new subtree with weight less than half of $\mathcal{W}$ because the total weight of the subtree rooted at the removed node exceeds half of $\mathcal{W}$. Calculating $\mathcal{W}$ and performing the pre-order traversal takes time $O(m)$.

Let $I = (G, \gamma, (w_i))$ be an instance of CSS where where every node in G has out-degree at most d and γ is k-favorable. Extract any spanning tree T from G. Using Lemma 7 we can identity a node in T called the *splitter* which, when removed, divides T into subtrees where each subtree has at most half the probability mass of T. In our algorithm, we don't remove the splitter from the tree but rather, remove the edge(s) connecting it to its parent(s). We reconnect the splitter to the root of T. Recursively apply this procedure on the subtrees rooted by the children of the root of T and call the final tree T'. Note that T' is still a subtree of the transitive closure of G since the splitter node is always descendent of the root of the tree under consideration. If G has m nodes than extracting a spanning tree from G takes $O(m)$ time. The complete procedure takes $O(m^2)$ time since we apply Lemma 7 to all m nodes m times.

Claim. If r and s are nodes in T' where r is the grandparent of s, then $W(r) \geq 2 \cdot W(s)$

This claim follows immediately from the construction of T' with respect to Lemma 7. Since any two hops in T' divides the probability mass of the subtree in half, we know the depth of leaf i is bounded above by $-2\log_2(w_i)$. Since each node in T' has degree at most $d+1$, the cost of T' is at most

$$2 \cdot \gamma(d+1) \sum_{i=1}^{n} w_i(-\log_2(w_i)) = 2 \cdot \gamma(d+1) H(W)$$

Since $O(\gamma(d+1)H(W))$ approximates the lower bound of $\frac{H(W)}{\log(k)}$ by a multiplicative factor of $O(\log(k)\gamma(d+1))$ we have an $O(m^2)$ time, $O(\log(k)\gamma(d+1))$ approximation algorithm to all instances of CSS where G has no nodes with degree greater than d and γ is k-favorable. $\square$

5 Leaves of Equal Weights

It is easy to imagine nascent companies building websites without any prior popularity statistics on their products. To gather such statistics, they may want a website which puts all their products on an equal footing. Finding the optimal website for equally-weighted topics corresponds to instances of CSS with a uniform probability distribution over the leaves. We characterize optimal trees for these instances of CSS for the linear degree cost when the graph is constraint-free, and for the logarithmic degree cost for any DAG.

5.1 Linear Degree Cost

Characterizing the optimal tree for the linear degree cost in constraint-free graphs involves three parts. First, we push all the out-degree-two nodes in an optimal tree down toward the leaves so that all but a few interior nodes have out-degree-three. Next, we show that balancing the tree decreases its cost, and finally, we show an optimal arrangement of the leaves for the balanced tree. All the results in this section assume a constraint-free graph, equal leaf weights, and the linear degree function, $\gamma(x) = x$. Additionally, we use the term *degree* in lieu of *out-degree* since we only concern ourselves with the out-degree of any node. Since γ is 3-favorable, we always assume the optimal trees have only out-degree-two and out-degree-three nodes.

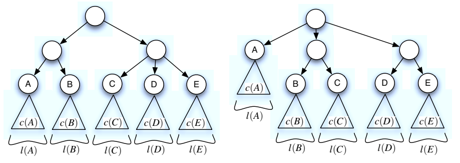

Fig. 6. (left) A degree 2 node with one degree-two child and one degree-three child. (right) The node after the transformation.

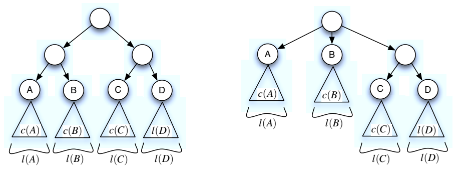

Fig. 7. (left) A degree-two node with two degree-two children. (right) The node after the transformation.

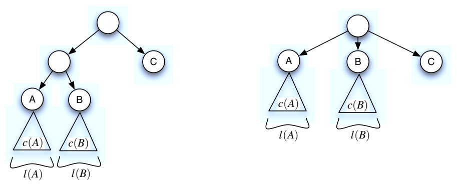

Fig. 8. (left) A degree-two node with one degree-two child and one leaf child. (right) The node after the transformation.

Theorem 5. *If $(G, \gamma, (w_i))$ is an instance of CSS where G is constraint-free, $\gamma(x) = x$, and the n leaf weights are equal, then if $n \leq 2 \cdot 3^k$, where $k = \lfloor \log_3(n) \rfloor$,*

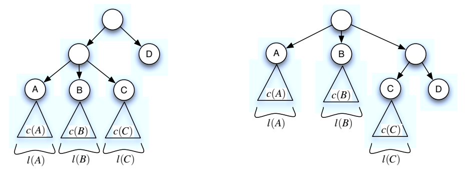

Fig. 9. (left) A degree-two node with one degree-three child and one leaf child. (right) The node after the transformation.

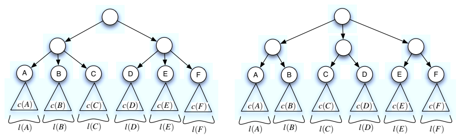

Fig. 10. (left) A degree-two node with two degree-three children. (right) The node after the transformation.

an optimal tree has cost

$$3nk + 4(n - 3^k)$$

otherwise it has cost

$$3(k+1)(3^{k+1}) - ((3^{k+1} - n)(3(k+1)+2))$$

Proof. We begin by showing the existence of an optimal tree with the following property:

Property 1. Every nodes has degree-two or degree-three. Additionally, all nodes with degree-two are *only* parents of leaves.

Lemma 8. *If $(G, \gamma, (w_i))$ is an instance of CSS where G is constraint-free, the leaf weights are distributed uniformly, and $\gamma(x) = x$ then there is an optimal tree with Property 1.*

Proof. The proof is by cases. We know an optimal tree exists with only degree-two and degree-three nodes since $\gamma(x)$ is three-favorable. Suppose this optimal tree has an interior degree-two node which is the grandparent or some earlier ancestor of a leaf node. There are five possible combinations of node degrees for the children of this interior node. We'll show that in each case, we can push the degree two nodes down in the tree without increasing the overall cost. In our analysis, we'll let $l(a)$ be the number of leaves in subtree rooted at node a (which corresponds to the number of paths through node a) and we'll let $c(a)$ be the total non-weighted cost of the subtree rooted at a.

– Consider a degree-two node with one degree-two child and one degree-three child like the one given on the left in Figure 6. Let $l(A) \geq l(B)$ and $T = c(A) + c(B) + c(C) + c(D) + c(E)$. The cost of the tree is $4(l(A) + l(B)) +$

$5(l(C)+l(D)+l(E))+T$, but the tree on the right has total cost $3l(A)+5(l(B)+l(C)+l(D)+l(E))+T$ which has cost no greater than the tree on the left since $l(A) \geq l(B)$.

- Consider a degree-two node with two degree-two children like the one given on the left in Figure 7. Let $l(A) \geq l(B) \geq l(C) \geq l(D)$ and $T = c(A)+c(B)+c(C)+c(D)$. The cost of the tree is $4(l(A)+l(B)+l(C)+l(D))+T$, but the tree on the right has cost $3(l(A)+l(B))+5(l(C)l(D))+T$ and since $l(A)+l(B) \geq l(C)+l(D)$ we know the cost of the tree never increases when transforming the structure from the left figure to the right figure.
- Consider a degree-two node with one degree-three child and one leaf child like the one given on the left in Figure 9 where $l(A) \geq l(B) \geq l(C) \geq l(D) = 1$ and $T = c(A)+c(B)+c(C)+c(D)$. The cost of the left tree is

$$\begin{aligned} 5(l(A)+l(B)+l(C))+2l(D)+T &\geq 4l(A)+5(l(B))+l(C))+3l(D)+T \\ &\geq 4(l(A)+l(B)+l(D))+5(l(C)+T \\ &\geq 3l(A)+4l(B)+5(l(C)+l(D))+T \\ &> 3(l(A)+l(B))+5(l(C)+l(D))+T \end{aligned}$$

  which is the cost of the right tree. So transforming the left tree to the right tree always decreases the total cost.
- Consider a degree-two node with one degree-two child and one leaf child like the one given on the left in Figure 8 where $l(A) \geq l(B) \geq l(C) = 1$ and $T = c(A)+c(B)+c(C)$. The cost of the tree on the left is

$$\begin{aligned} 4(l(A)+l(B))+2l(C)+T &\geq 3(l(A)+l(C))+4l(B)+T \\ &> 3(l(A)+l(B)+l(C))+T \end{aligned}$$

  which is the cost of the tree on the right. So transforming the left tree to the right tree always decreases the cost.
- Consider a degree-two node with two degree-three children like the one on the left in Figure 10. Transforming the left tree into the right tree keeps the cost of the tree invariant because of the symmetry between two degree-three nodes and three degree-two nodes. In each arrangement, the path cost is always five from the root to the grandchildren.

Since these cases are exhaustive and since we never increase the cost of the tree by pushing degree-two nodes down in the topology, we can always transform an optimal tree into another optimal tree where every ancestor above the parent of a leaf has degree three.

Next we show that making a tree more balanced only improves the overall cost of the tree. In addition we characterize the fringe of the tree.

Lemma 9. *If $(G, \gamma, (w_i))$ is an instance of CSS where G is constraint-free, $\gamma(x) = x$, and the leaf weights are distributed uniformly, then any optimal tree having Property 1 has the following properties:*

(a) no two leaves differ in depth by more than one.
(b) degree-two internal nodes have the same depth.
(c) no internal node has degree greater than any other internal node of less depth.
(d) no degree-three internal nodes have the same depth of a leaf.

Proof. We begin by proving (a). Let T be an optimal tree meeting Property 1. Such a tree exists by by Lemma 8. Let a and b be leaf nodes and a' and b' be their respective parents. Suppose that $depth(a) > depth(b) + 1$, then if the path cost to b' is $3k$ for some non-negative integer k, the path cost to a' must be at least $3(k + 2)$. We can always move a to b' and decrease the cost of the tree: If b' is a degree-two node, we can add a leaf and increase the overall cost by $3k + 5$. If b' is a degree-three node, we can make b an internal degree-two node and increase the overall cost by $3k + 7$. If a' is a degree-two node, removing a saves us $3k + 10$ and if a' is a degree-three-node, removing a saves us $3k + 11$. In all four combinations of leaf removal from a' and leaf addition to b' we decrease the cost of the tree, a contradiction of the optimality of T, so all leaves of T differ in depth by at most one.

Since all degree-two nodes are only parents of leaves, and all leaves differ in depth by at most one, to prove (b) and (c), we need to show that if b is a leaf at depth k with a degree-two parent b', that a' cannot be a degree-three or degree-two parent of a $k + 1$-depth leaf. Suppose this were true, then we can add a leaf to b', increasing the cost by $3k + 2$ and remove a leaf at a', decreasing the cost by at least $3k + 4$, giving us at least a net savings of two, a contradiction.

Since T has properties (a), (b), and (c), to prove (d), we need only show that if a is a leaf node at depth $k + 1$ with a degree-three parent node a' at depth k, that no leaf b exists at depth k. Suppose this were true, then we could make b an internal node (rename it b'') with two children a and b at a cost of $3k + 4$, but removing a from a' saves us $3k + 5$, again, contradiction of optimality.

At this point we know the exact structure of an optimal tree T. Lemma 8 and property (a) of Lemma 9 tell us that the tree is balanced, and that any degree-two node must be the parent of only leaf nodes. Property (b) restricts degree-two nodes to parents of leaves at the lowest depth, so that if T has n leaves, it always begins with a complete tertiary tree of height $\lfloor \log_3(n) \rfloor$. Finally, property (c) implies that a leaf has a degree-three parent, only when all the leaves of the tree have the same depth. In total, given n leaves, we build the largest complete tertiary tree of size $k = \lfloor \log_3(n) \rfloor$, then add additional leaves by making leaves of the height-k tertiary tree into degree-two parents of leaves, and finally adding an additional leaf to the binary nodes when no leaves at depth k remain. This means, the bottom row of the optimal tree gets filled with degree-two nodes first and then, when additional leaves are required, the degree-two nodes are expanded to degree-three nodes. Given this fixed structure, we can give an exact measure of its cost. When $n \leq 2 \cdot 3^k$ we know the bottom row is exclusively degree-two nodes, so the total cost of the optimal tree is $3nk + 4(n - 3^k)$ where the first component of the sum is the cost of the complete tertiary tree, and the second component of the sum is the cost of the additional leafs. When the bottom row contains leaves coming from degree-three parents, we can think of the cost of the optimal tree in terms of subtracting away from the complete tertiary tree of depth $k+1$. Making a degree three node a degree two node means removing one leaf which reduces the overall cost by $3(k+1)$, but since the degree on the parent changes, each sibling path is reduced by one, making the overall savings $3(k+1)+2$. This means the total cost of the tree is $3(k+1)(3^{k+1}) - ((3^{k+1} - n)(3^{k+1} + 2)$ where the first part of the sum is the cost of the complete tertiary tree and the second part is the reduction in cost by removing the appropriate number of leaves. $\square$

In Section 4 we defined a degree cost γ as k-tree optimal when, for constraint-free graphs and equal leaf weights, the the unique optimal tree with k^c leaves, for any positive integer c, is a complete k-ary tree of depth c. Here we show that the linear degree cost is 3-tree-optimal.

Theorem 6. *The degree cost $\gamma(x) = x$ is 3-tree-optimal.*

Proof. Let T be an optimal tree for the CSS problem where the graph is constraint-free, the degree cost is $\gamma(x) = x$, and there are $n = 3^k$ leaves, for some positive integer k, of equal weight. T may have nodes with degree two, three, or four. Nodes with degree five or larger don't appear in an optimal tree since replacing them with degree-two and degree-three nodes always decreases the cost of the tree. We replace all degree-four nodes from T with complete binary trees of size three at no additional cost. By Lemma 8 we can push the degree-two nodes down to the fringe. Furthermore, since T is optimal, we won't match Figures 9 and 8 because they are strictly cost-decreasing transformations. Since the other transformations always preserve at least one of the degree-two nodes, any optimal tree with degree-two nodes has a corresponding tree of equal cost with at least one degree-two node and all degree-two nodes are parents of only leaves (i.e., it exhibits Property 1). By Lemma 9, we know T has at least $3^{k-1}+1$ leaves, but no more than $3^k - 1$ leaves because there is at least one degree-two parent of only leaves. But T has $n = 3^k$ leaves, a contradiction, so T cannot have degree-two or degree-four nodes. This means the optimal tree has only degree-three nodes, and by Theorem 5, is a complete tertiary tree with $n = 3^k$ leaves. $\square$

5.2 Logarithmic Degree Costs

Another natural choice of degree cost is $\gamma(x) = \lg(x)$ (where $\lg = \log_2$) because it gives the number of bits needed to encode the out-degree of the node. In this section we'll show the depth-one tree (where the root has n edges of its n leaves) is an optimal solution to any instance of CSS where the n leaf weights are equal and $\gamma(x) = \lg(x)$. This result holds for arbitrary graphs because the depth-one tree is always a subtree of the transitive closure.

Theorem 7. *Let $I = (G, \gamma, (w_i))$ be an instance of CSS where $\gamma(x) = \log(x)$ and the n leaf weights are equal. An optimal tree for I is the depth-one tree.*

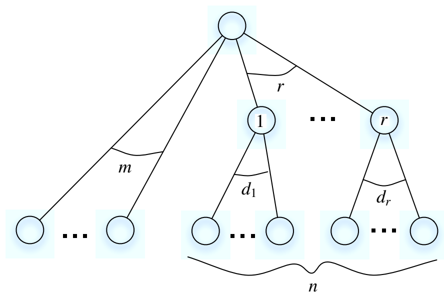

Fig. 11. The topology of an *mrn*-tree.

Finally, we noted in Sec. 2 that CSS with degree cost $\gamma(x) = \lceil \log_2(x) \rceil$ is NP-Hard even with equal leaf weights. This is somewhat surprising given the depth-one tree is optimal for $\gamma(x) = \log(x)$ with equal leaf weights. The result holds because the ceiling provides a place where the cost jumps enough so that any non-optimal tree suffers the impact of this slight increase. We show here though as a corollary to Theorem 7, that the depth-one tree approximates the optimal solution to $\gamma(x) = \lceil \log_2(x) \rceil$ within an additive constant of 1.

Proof. For clarity, we distinguish between the leaves at depth one and the leaves at depth two. Let $T = (m, (d_1, \ldots, d_r)), m, d_i \in \mathbb{Q}^+$ be a depth-two tree with $m+n$ leaves: m depth-one leaves and $n = \sum_{i=1}^{r} d_i$ depth-two leaves. Each depth-two leaf has a single path from the root to it through one of the r intermediate nodes. We denote the degree of each intermediate node i as d_i. We call these trees *mrn*-trees and illustrate the general topology in Figure 11. We begin by proving an intermediate result on the nature of convex functions.

Lemma 10. *Let I be any real interval and $(x-\alpha, x+c+\alpha)$, a subinterval of I where c and α are real values with $c \geq 0$ and $\alpha > 0$. If $g : I \to \mathbb{R}$ is convex on I (and has a second derivative in I) we have*

$$g(x+c+\alpha) + g(x-\alpha) > g(x+c) + g(x)$$

Proof. Suppose $g(x)$ is defined over some real interval I and $(x-\alpha, x+c+\alpha)$ is a subinterval of I with $c \geq 0$ and $\alpha > 0$. Suppose further that $g(x)$ is convex on I (and has a second derivative on I), then $g'(x)$ is increasing so

$$\begin{aligned} g'(x+c+y) - g'(x-\alpha+y) > 0 \qquad & c > 0, \alpha \geq 0 \text{ and } 0 \leq y \leq \alpha \\ \Leftrightarrow \int_0^\alpha g'(x+c+y) - g'(x-\alpha+y)dy > 0 & \\ \Leftrightarrow \int_0^\alpha g'(x+c+y)dy > \int_0^\alpha g'(x-\alpha+y)dy & \\ \Leftrightarrow g(x+c+\alpha) - g(x+c) > g(x) - g(x-\alpha) & \\ \Leftrightarrow g(x+c+\alpha) + g(x-\alpha) > g(x+c) + g(x) & \end{aligned}$$

Lemma 10 means that making the degrees of the intermediate nodes of an mrn-tree proportional always improves its cost.

Lemma 11. *If $\gamma(x)$ is an increasing, differentiable function over the positive reals, then the r intermediate nodes of the minimum cost mrn-tree have out-degree n/r when the weights of the leaves are equal.*

Proof. Lemma 11 Let $\gamma(x)$ be increasing and differentiable over the positive reals. Then $g(x) = x\gamma(x)$ has a positive second derivative on the positive reals, so it is convex. Let $T = (m, (d_1, \ldots, d_r))$ be an mrn-tree with $m + n$ equally weighted leaves. For the sake of contradiction, suppose T has at least two intermediate nodes, i and j, with disproportionate out-degree, so that $d_i > d_j > 0$. Let $d_i = d_j + \beta$ for some positive $\beta \in \mathbb{Q}$. Shifting some rational part of the path, $0 < \alpha \leq \beta/2$ from node i to node j to make i and j more proportional, decreases the cost of the subtree rooted at i by

$$d_i\gamma(d_i) - (d_i - \alpha)\gamma(d_i - \alpha) \tag{2}$$

but the cost of the subtree rooted at j increases by

$$(d_j + \alpha)\gamma(g_j + \alpha) - d_j\gamma(d_j) \tag{3}$$

Letting $g(x) = x\gamma(x)$, we need to show that the decrease in cost is greater than the increase in cost:

$$g(d_i) - g(d_i - \alpha) > g(d_j + \alpha) - g(d_j) \tag{4}$$

Since $\alpha \leq \beta/2$ we have $\beta = 2\alpha + c$ for some $c \geq 0$ so $d_i = d_j + 2\alpha + c$. Letting $x = d_j + \alpha$ we have $d_i = x + \alpha + c$ and by rearranging the terms from the inequality in 4 we have

$$g(x + c + \alpha) + g(x - \alpha) > g(x + c) + g(x)$$

which holds from Lemma 10 and gives us a contradiction, so the minimum cost mrn-tree with equal leaf weights must have proportional degrees among its r interior nodes. In other words, each interior node with out-degree n/r.

From the proportional mrn-tree, we can move to the depth-one tree with $m + n$ leaves without increasing the overall cost:

Lemma 12. *If $\gamma(x) = \log(x)$ then any mrn-tree with equal leaf weights has an equivalent cost depth-one tree with equal leaf weights.*

Proof. Let $\gamma(x) = \log(x)$ and $T = (m, (d_1, \ldots, d_r))$ be an mrn-tree with equal leaf weights. By Lemma 11 we can transform T, at no additional cost, so that each intermediate node has out-degree n/r. This makes the cost of T:

$$(n + m)\log(r + m) + n\log(n/r)$$

whereas the cost of the equivalent depth-one tree with $(n+m)$ leaves is

$$(n+m)\log(n+m)$$

We'd like to show that the cost of the depth-one tree is no greater than the cost of the mrn-tree. Subtracting the cost of the depth-one tree from the cost of the mrn-tree gives us:

$$f(n,m,r) = (n+m)\log(r+m) + n\log(n/r) - (n+m)\log(n+m) \tag{5}$$

where n, m, r are non-negative rational values and $r \leq n$. Showing f never takes on negative values gives us the desired result, so we show it is non-decreasing in n, m and r. Taking the partial derivative of f with respect to m give us:

$$\frac{\partial f}{\partial m} = \log(r+m) + \frac{n-r}{r+m} - \log(n+m) \tag{6}$$

and taking the partial derivative of 7 gives us:

$$\frac{\partial^2 f}{\partial m \partial n} = \frac{1}{r+m} - \frac{1}{n+m} \tag{7}$$

which is greater than or equal to 0 since $r \leq n$. It immediately follows that 7 is non-decreasing, so we know 6 is non-decreasing in n and since n is bounded below by r, we let $n = r$ whence 6 equals 0. This means 5 is non-decreasing in m, so we let $m = 0$ in f:

$$f(n,0,r) = n\log(r) + n\log(n/r) - n\log(n) = 0$$

Since f is non-decreasing in all variables, the cost of the depth-one tree is no greater than the cost of the mrn-tree.

We're now in a position to prove our result. Let $I = (G, \gamma, (w_i))$ be an instance of CSS where $\gamma(x) = \log(x)$ and G has n leaves of equal weight. Let T be an optimal tree for I containing the minimum number of nodes. Suppose T is not a depth-one tree. Then there exists a node having both (and only) children and grandchildren. This is an mrn-tree, and by Lemma 12, we can covert it into a depth-one tree at no additional cost, transforming T into a new optimal tree with less nodes, a contradiction, so T must be a depth-one tree. $\square$

Corollary 1. *If $(G, \gamma, (w_i))$ is an instance of CSS with $\gamma(x) = \lceil \log_2(x) \rceil$ and n leaf weights are equal, then the depth-one tree approximates the optimal cost tree within an additive constant of 1.*

Proof. Let $I = (G, \gamma, (w_i))$ be an instance of CSS with degree cost $\gamma(x) = \lceil \log_2(x) \rceil$ and n uniformly distributed leaf weights. Furthermore, let c be the cost of a tree under $\gamma(x) = \lceil \log_2(x) \rceil$ and c' be the cost of a tree under $\gamma(x) = \log_2(x)$.

If T is an optimal solution for I and T' is a depth-one tree with n leaves, we have the following inequality:

$$\begin{aligned}\log_2(n) &= c'(T') \\ &\leq c'(T) \\ &\leq c(T) \\ &\leq c(T') \\ &= \lceil \log_2(n) \rceil \\ &\leq \log_2(n) + 1\end{aligned}$$

which gives the desired lower and upper bounds. □

6 Final Thoughts

While we have positive results for CSS when the initial hierarchy is constraint-free, and negative results when it is a DAG, we have yet to characterize the problem for directed trees. We have looked at specific tree topologies, like binary trees and complete r-ary trees, but even in these cases, have not characterized the optimal solutions for the linear degree cost. Additionally, we have not explored probability distributions other than arbitrary and uniform. For example, what happens with a geometric or Zipfian distribution? Finally, we are interested in CSS in dynamic environments. For example, on a website, page statistics are constantly changing. Is there a way to dynamically update the optimal tree without unnecessary computation?

References

1. Heeringa, B., Adler, M.: Optimal website design with the constrained subtree selection problem. Technical Report 04-09, University of Massachusetts Amherst (2004)
2. Perkowitz, M., Etzioni, O.: Towards adaptive web sites: Conceptual framework and case study. Artificial Intelligence **118** (2000) 245–275
3. Bose, P., Czyzowicz, J., Gasienicz, L., Kranakis, E., Krizanc, D., Pelc, A., Martin, M.V.: Strategies for hotlink assignments. In Lee, D.T., Teng, S.H., eds.: Algorithms and Computation, 11th International Conference. Volume 1969 of Lecture Notes in Computer Science., Springer (2000) 23–34
4. Czyzowicz, J., Kranakis, E., Krizanc, D., Pelc, A., Martin, M.V.: Evaluation of hotlink assignment heuristics for improving web access. In: Second International Conference on Internet Computing, CSREA Press (2001) 793–799
5. Czyzowicz, J., Kranakis, E., Krizanc, D., Pelc, A., Martin, M.V.: Enhancing hyperlink structure for improving web performance. Journal of Web Engineering **1** (2003) 93–127
6. Karp, R.: Minimum-redundancy coding for the discrete noiseless channel. IRE Transactions on Information Theory **IT** (1961) 27–29

7. Golin, M.J., Kenyon, C., Young, N.E.: Huffman coding with unequal letter costs. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM Press (2002) 785–791
8. Golin, M.J., Rote, G.: A dynamic programming algorithm for constructing optimal prefix-free codes with unequal letter costs. IEEE Transactions on Information Theory **44** (1998) 1770–1781
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York, New York (1979)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. 2 edn. The MIT Press/McGraw-Hill Book Company (2001)
11. Siu-Ngan Choi and Golin, M.: Lopsided Trees I: A Combinatorial Analysis. Algorithmica **31** (2001) 240–290