Optimal Website Design with the Constrained Subtree Selection Problem

Brent Heeringa[1,2] and Micah Adler[1]

[1] Department of Computer Science, University of Massachusetts, Amherst
140 Governors Drive Amherst, MA 01003
[2] Department of Computer Science, Williams College, Williamstown, MA, 01267
{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.

[⋆] This research partially funded under NSF Research Infrastructure Award EIA-0080119, NSF Faculty Early Career Development Award CCR-0133664, and NSF ITR Grant ITR-0325726

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 [9].

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 [1]. 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 ([2], [3]) 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 [4]. 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 ([5], [6]). 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 [7].

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 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) [8] 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 integers $k \geq 1$, 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.*

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 $\gamma(s(n/3) + n/3) = O(n^c)$ then CSS with γ is NP-Complete in the strong sense.*

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.

We begin with some definitions. A tree is *full* when every interior node has at least two children. 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. This means that G does not constrain the optimal subtree. A tree is *monotone* when the leaf weights cannot be permuted (among the leaves) to yield a tree of lower cost. Hence, 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.

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. This definition is 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. Proving that a particular γ exhibits k-favorability for some k typically means showing that any node with out-degree at least $(k+1)$ can be split into nodes of smaller degree with no increase to the overall cost of the tree. Many degree costs are k-favorable. For example the linear degree cost $\gamma(x) = x$ is 3-favorable, but not 2-favorable [9]. In section 5 we characterize the optimal tree for the linear degree cost when the graph is constraint-free and the weights are equal. 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 ([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 $k > 2$? More generally, is there a polynomial time algorithm that solves $(G, \gamma, (w_i))$ when G is constraint-free and γ is k-favorable? In this section we give a dynamic programming algorithm which leads to the following result.

Theorem 3. *There is a $O(n^{\gamma(k)+k})$ time algorithm which finds an optimal solution to any instance of CSS where G is constraint-free, γ is k-favorable for some integer k, non-decreasing and maps the positive integers to the positive integers.*

We adapt the dynamic programming algorithm for finding optimal prefix-free codes (OPC) given by Golin and Rote ([7]) to the CSS problem. We highlight some of the similarities and differences between the two algorithms here but give a complete description of our algorithm and a proof of Theorem 3 in [9].

The solution to the optimal prefix-free coding problem with integer costs relies on a lopsided representation of a tree. A lopsided tree equates a node's level to its path cost from the root. In other words, if u is a node in T, and the path cost to u is C, then we say u is at level C. Restricting the cost to integers means the levels are also integers. Golin and Rote associate a signature with each tree level so that if a tree has h levels, then it has h signatures. Signatures are always taken with respect to the truncation of a tree at a certain level. If T is a tree with n leaves, then the *level-i-truncation* of T, denoted $\text{Trunc}_i(T)$, prunes away all nodes of T with *parents* at levels deeper than i. The *level-i-signature* of T is the $(C+1)$ vector: $sig_i(T) = (m, l_1, \ldots, l_C)$ where m is the number of leaf nodes at levels 0 through i, l_j is the number of nodes at level $i+j$ in $\text{Trunc}_i(T)$, and C is the largest symbol (edge) cost. If $v_1, \ldots, v_n$ are the leaves of T given in increasing order by level and $w_1 \ldots, w_n$ are the leaf weights given in decreasing order then the *level-i-cost* of T is $c_i(T) = \sum_{j=1}^{m} \text{level}(v_j) w_j + \sum_{s=m+1}^{n} i \cdot w_s$ where m is the number of leaf nodes in $\text{Trunc}_i(T)$.

The level-i-signature of a tree $(m, l_1, \ldots, l_C)$ equates to an entry in the dynamic programming table $\mathrm{MIN}[m, l_1, \ldots, l_C]$. This entry gives the minimum level-i-cost of all trees with signature $(m, l_1, \ldots, l_C)$. There are $O(n^{C+1})$ table entries since the number of nodes at the fringe of the tree never exceeds n. Note that the signature does not indicate level, so the value of an entry may correspond to the level-i-cost of trees at a variety of levels. Given a tree's signature at level $i+1$, it's possible to enumerate what level-i-signatures lead to it. Similarly, the level-$(i+1)$-cost of a tree can be written in terms of the level-i-cost of the tree associated with the signature that precedes it which gives a natural method for filling in the dynamic programming table.

When considering how level-$(i+1)$-signatures relate to level-i-signatures, we must consider structural changes to the tree. In the OPC domain, adding an edge does not change the lopsided structure of the rest of the tree. In our domain when an edge is added, the lopsided structure of the tree does change because the node degree changes. As a result, we cannot apply Golin and Rote's algorithm verbatim; we can use the subproblem representation (i.e. the signatures) by letting $C = \gamma(k)$ but filling in the table requires a different approach.

We must examine the way two trees with the same signature at level i can differ in their level-$(i+1)$-signature. Given a level-i-signature we must first choose how many level $(i+1)$ nodes will be internal, and them among those, which will have degree 2, degree 3, and so on. We denote these choices with a $(k+1)$-vector $a = (a_0, \ldots, a_k)$ called a child vector 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. Note that $a_0 \le l_1$ and that $a_1 = 0$ since there is always an optimal tree with no nodes having out-degree 1. 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 a level-i-signature parent.

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

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

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

While Lemma 1 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 [7] tells us that if T is a tree with $\mathrm{sig}_i(T) = (m, l_1, \ldots, l_{\gamma(k)})$ then $c_{i+1}(T) = c_i(T) + \sum_{j=m+1}^{n} w_j$. Fortunately, this result holds for all monotone, lopsided trees with level-i-costs defined as above so even though our problem has a different dependency structure in the table, it does not require a new way of computing cost in terms of cost to subproblems. Golin and Rote give a linear ordering of the table entries that respects their dependency structure. This ordering works for us too, although their proof of this fact no longer applies because our table entries have a different dependency structure. We describe the ordering in [9] and show that it works for our problem too. What's most important is that viewing table entries as nodes and dependencies as edges still leaves us with a DAG, so any topological sort yields an appropriate order for filling in the table.

Here is a description of our algorithm. We repeatedly process table entries in an order that respects the dependency structure, beginning with the entry corresponding to the level-1-truncation of a single node with two children ($\mathrm{MIN}[0, 2, 0, \ldots, 0]$) and ending with the entry corresponding to a tree with n leaves ($\mathrm{MIN}[n, 0, \ldots, 0]$). Given an entry we consider all its children (via Lemma 1) and then update the cost of the children (by Lemma 5 in [7]) if there is an improvement. After completing the table, the entry $\mathrm{MIN}[n, 0, \ldots, 0]$ contains the cost of the minimum tree. We can keep an additional table relaying points to the entries which yield the optimal cost to easily reconstruct the optimal tree. The $O(n^{\gamma(k)+k})$ running time of the algorithm follows because the table has $O(n^{\gamma(k)+1})$ entries of which each has at most $O(n^{k-1})$ dependencies to check.

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 [9] we show that the linear degree cost is 3-tree optimal.

Theorem 4. *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 $k + 5$.*

Consider the *Partitioned Exact Cover by 3 Sets* (PX3S) problem, which we define here. 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. The proof of Theorem 4 appears in [9], but we provide a high level overview here. 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.

Theorem 5. *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 an $O(m)$ 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 2. *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.*

The proof of the lemma appears in [9] but the main idea is that the cost of any optimal tree to the CSS problem is bounded below by the cost of the optimal prefix-free code over a k-ary alphabet with character costs 1 which is bounded below by $\frac{H(W)}{\log(k)}$ by Shannon's theorem.

Our approximation algorithm also requires the following result which is easy to prove (although we provide a proof in [9]).

Claim. For any tree with with weights on its m nodes, there exists at least 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 such a node in $O(m)$ time.

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 Claim 4 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 then extracting a spanning tree from G takes $O(m)$ time since each node has constant degree. The complete procedure takes $O(m)$ time since applying Claim 4 to all m nodes can be accomplished in $O(m)$ time with some bookeeping.

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

This claim follows immediately from the construction of T' with respect to Claim 4. 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 $-2\log_2(w_i)$. Since each node in T' has degree at most $d+1$, the cost of T' is at most $2 \cdot \gamma(d+1) \sum_{i=1}^{n} w_i(-\log_2(w_i)) = 2 \cdot \gamma(d+1) \cdot H(W)$

Since $O(\gamma(d+1)H(W))$ approximates the lower bound of $H(W)/\log(k)$ by a multiplicative factor of $O(\log(k)\gamma(d+1))$ we have the desired result.

5 Leaves of Equal Weight

It is easy to imagine fledgling 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

Theorem 6 gives the cost of an optimal tree for the linear degree function when the graph is constraint-free and $\gamma(x) = x$. We arrive at this cost by showing how to construct an optimal tree. Proof of the the construction's optimality is involved, but the tree is simple to describe: An optimal tree with n leaves begins with a complete tertiary tree with $\lfloor \log_3(n) \rfloor$ leaves. Additional leaves are added in pairs by splitting the leaves of the complete tertiary tree into binary nodes. Finally, if we still require more leaves, we add an additional edge to each binary node. In some sense, an optimal tree for n leaves is one that is always trying to be the most complete tertiary tree with n leaves.

Theorem 6. *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 $k = \lfloor \log_3(n) \rfloor$, an optimal tree has cost $3nk + 4(n - 3^k)$ otherwise it has cost $3(k+1)(3^{k+1}) - ((3^{k+1} - n)(3(k+1) + 2))$*

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 show the depth-one tree (where the root has n edges directly to 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 constraint graphs because the depth-one tree is always a subtree of the transitive closure. Proof of Theorem 7 is given in [9].

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.*

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. A corollary to Theorem 7 is that the depth-one tree approximates the optimal solution when $\gamma(x) = \lceil \log_2(x) \rceil$ within an additive constant of 1.

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.*

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 in time proportional to the height of the tree?

References

1. Perkowitz, M., Etzioni, O.: Towards adaptive web sites: Conceptual framework and case study. Artificial Intelligence **118** (2000) 245–275
2. 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
3. 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
4. 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
5. Karp, R.: Minimum-redundancy coding for the discrete noiseless channel. IRE Transactions on Information Theory **IT** (1961) 27–29
6. 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
7. 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
8. 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)
9. Heeringa, B., Adler, M.: Optimal website design with the constrained subtree selection problem. Technical Report 04-09, University of Massachusetts Amherst (2004)
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)