The Structure of Optimal Prefix-Free Codes in Restricted Languages: the Uniform Probability Case (Extended Abstract)*

Mordecai J. Golin and Zhenming Liu

Abstract. In this paper we discuss the problem of constructing minimum-cost, prefix-free codes for equiprobable words under the assumption that all codewords are restricted to belonging to an arbitrary language $\mathcal{L}$ and extend the classes of languages to which $\mathcal{L}$ can belong.

Note: This extended abstract is essentially the version which appears in The proceedings of WADS'05*, but with extra diagrams added.*

Varn Codes are minimum-cost prefix-free codes for equiprobable words when the encoding alphabet has *unequal-cost letters*. They can be modelled by the leaf-set of minimum external-path length *lopsided trees*, which are trees in which different edges have different lengths, corresponding to the costs of the different letters of the encoding alphabet. There is a very large literature in the information theory and algorithmic literature devoted to analyzing the cost [26] [20] [12] [13] [3] [18] [23] [1] [7] [24] [7] [24] of such codes/trees and designing efficient algorithms for building them [18] [8] [28] [10] [22] [17] [7].

It was recently shown [15] that the Varn coding problem can be rewritten as the problem of constructing a minimum-cost prefix-free code for equiprobable words, under the assumption that all codewords are restricted to belonging to an arbitrary language $\mathcal{L}$ where $\mathcal{L}$ is a special type of language, specifically a regular language accepted by a DFA with only one accepting state. Furthermore, [15] showed that the techniques developed for constructing Varn Codes could then be used to construct optimal codes restricted to *any* regular $\mathcal{L}$ that is accepted by a DFA with only *one* accepting state. Examples of such languages are where "$\mathcal{L}$ is all words in Σ^* ending with a particular given string $P \in \Sigma^*$," i.e., $\mathcal{L} = \Sigma^* P$ (the simplest case of such a language are the 1-ended codes, $\mathcal{L} = (0+1)^*1$ [4, 5]). A major question left open was how to construct minimum-cost prefix-free codes for equiprobable words restricted to $\mathcal{L}$ when $\mathcal{L}$ does not fit this criterion.

In this paper we solve this open problem for *all* regular $\mathcal{L}$, i.e., languages accepted by Deterministic Finite Automaton, as long as the language satisfies a very general non-degeneracy criterion. Examples of such languages are $\mathcal{L}$ of the type, $\mathcal{L}$ is all words in Σ^* ending with *one* of the given strings $P_1, P_2, \ldots, P_n \in \Sigma^*$. More generally our technique will work when $\mathcal{L}$ is a language accepted by any Deterministic Automaton, even automaton with a countably *infinite* number of states, as long as the number of accepting states in the automaton is finite.

Our major result is a combinatorial theorem that, given language $\mathcal{L}$ accepted by a Deterministic Automaton, exactly describes the general *structure* of all optimal prefix-free codes restricted to $\mathcal{L}$. This theorem immediately leads to a simple algorithm for *constructing* such codes given the restriction language $\mathcal{L}$ and the number of leaves n.

0.1 Formal Statement of the Problem

We start with a quick review of basic definitions. Let Σ be a finite alphabet, e.g., $\Sigma = \{0,1\}$, or $\Sigma = \{a,b,c\}$. A *code* is a set of words $C = \{w_1, w_2, \ldots, w_n\} \subset \Sigma^*$. A word $w = \sigma_1\sigma_2\ldots\sigma_l$ is a *prefix* of another word $w' = \sigma'_1\sigma'_2\ldots\sigma'_{l'}$ if w is the start of w'. For example 01 is a prefix of 010011. Finally, a code is said to be *prefix-free* if for all pairs $w, w' \in C$, w is not a prefix of w'.

Let $P = \{p_1, p_2, p_3, \ldots, p_n\}$ be a discrete probability distribution, that is, $\forall i,\ 0 \le p_i \le 1$ and $\sum_i p_i = 1$. The *cost* of code C with distribution P is $\mathrm{cost}(C,P) = \sum_i |w_i| \cdot p_i$ where $|w|$ is the length of word w; $\mathrm{cost}(C,P)$

* Department of Computer Science, HKUST, Clear Water Bay, Kowloon, Hong Kong. email:{golin,cs_lzm}@cs.ust.hk. The work of both authors was partially supported by HK RGC CERG grants HKUST6312/04E and DAG03/04.EG06

$$A_1 = \{aabca, babca, cabca, dabca\}, \qquad A_2 = \{01, 0011, 1100, 1010, 1001\}$$
$$B_1 = \{aabca, babca, cabca, aaabca\}, \qquad B_2 = \{01, 10, 0011, 101010, 000111\}$$

Fig. 1. Examples of optimal (A_1, A_2) and non-optimal (B_1, B_2) codes in $\mathcal{L}_1$ and $\mathcal{L}_2$. $\text{cost}(A_1) = 20$; $\text{cost}(B_1) = 21$; $\text{cost}(A_2) = 18$; $\text{cost}(B_2) = 20$.

is therefore the average length of a word under probability distribution P. The *prefix-coding problem* is, given P, to find a prefix-free code C that minimizes $\text{cost}(C, P)$. This problem is well-studied and can easily and efficiently be solved by the well-known Huffman-coding algorithm, see e.g., [11]. When the codewords are *equiprobable,* i.e., $\forall i$, $p_i = 1/n$, then $\text{cost}(C, P) = \frac{1}{n}\sum_i |w_i| = \frac{1}{n}\text{cost}(C)$ where $\text{cost}(C) = \sum_i |w_i|$. $\text{cost}(C, P)$ is then minimized when $\text{cost}(C)$ is minimized. We will call such a code an *optimal uniform-cost code.*

In this paper we are interested in what happens to the uniform-cost code problem when it is restricted so that all of the words in C must be contained in some language $\mathcal{L} \subseteq \Sigma^*$,. As examples consider $\mathcal{L} = \mathcal{L}_1$, the set of all words in $\{a, b, c\}^*$ that end with the pattern *abca* and $\mathcal{L} = \mathcal{L}_2$, the set of all words in $\{0, 1\}^*$ in which the number of '0's is equal to the number of '1's.

In Figure 1, the codes A_i are optimal prefix-free codes (for 4/5) words in $\mathcal{L}_i$ (i=1,2). That is, no codes with the same number of words in $\mathcal{L}_i$ have smaller cost than the A_i. The B_i are non-optimal codes in the same languages.

Let language $\mathcal{L}$ be fixed. We would like to answer the questions:

- What is the optimal (min-cost) prefix-free code C_n containing n words in $\mathcal{L}$?
- How does C_n change with n?

We call this the *$\mathcal{L}$-restricted prefix-coding problem.* Our major tools for attacking this problem are *generalized lopsided trees.*

Note: In this extended abstract we only state our main results and provide intuition as to why they are correct. The full proofs are omitted.

1 Generalized Lopsided Trees

Definition 1. *See Figures 2 and 3.*
We are given a finite set $T = \{t_1, t_2, ..., t_k\}$ and two functions

$$cost(\cdot,\cdot) : T \times N^+ \to N^+ \quad and \quad type(\cdot,\cdot) : T \times N^+ \to T$$

where N^+ is the set of nonnegative integers; T, cost() *and* type() *are the* tree parameters.

- ***A generalized lopsided tree*** *for T, $cost(\cdot,\cdot)$ and $type(\cdot,\cdot)$ is a tree (of possibly unbounded node-degree) in which every node is labelled with one element T.*
- *The label of a node is its* ***type****; equivalently, a node of type t_i is a t_i-node.*
- *By convention, unless otherwise explicitly stated, the root of a generalized lopsided tree must be a t_1-node.*
- *The jth child of a t_i node, if it exists, will have type $type(t_i, j)$. The* ***length*** *(weight) of the edge from a t_i-node to its jth child, will be $cost(t_i, j)$. By convention, we will assume that if $j \le j'$, then $cost(t_i, j) \le cost(t_i, j')$.*

Note that it is possible that a type $t_i \in T$ node could be restricted to have at most a finite number k of possible defined children. In this case, $cost(t_i, j)$ and $type(t_i, j)$ are undefined for $j > k$.

Note too that it is possible for a node to be "missing" its middle children, e.g, the 1st and 3rd child of a node might be in the tree, but the 2nd child might not.

When designing an algorithm for constructing optimal trees we will assume that the values $\text{cost}(t_i, j)$, $\text{type}(t_i, j)$ *and* $Num(i, m, h) = |\{j : \text{cost}(t_i, j) = h \text{ and } \text{type}(t_i, j) = t_m\}|$, *can all be returned in $O(1)$ time by some oracle.*

Finally, we point out that our definition restricts cost$(\cdot,\cdot)$ to be nonnegative integers. If cost$(\cdot,\cdot)$ were arbitrary nonnegative rationals they could be scaled to be integers and the problem would not change. Allowing cost$(\cdot,\cdot)$ to be nonnegative irrationals *would change the problem and require modifying many of the lemmas and theorems in this paper. In this extended abstract we restrict ourselves to the simpler integer case since, as we will soon see, restricted languages can be modelled using integer costs.*

Definition 2. *See Figures 2 and 3. Let u be a node and Tr be a generalized lopsided tree.*

- $\mathrm{depth}(u)$ *is the sum of the lengths of the edges on the path connecting the root to u.*
- *The **height** of Tr is $H(Tr) = \max_{u\in Tr} \mathrm{depth}(u)$.*
- *The **leaf set** of Tr is $\mathrm{leaf}(Tr)$, the set of leaves of Tr.*
- *The **cost** of Tr is its **external path length** or $C(Tr) = \sum_{v\in \mathrm{leaf}(Tr)} \mathrm{depth}(v)$*

Tree Tr is optimal *if it has minimum external path length over all trees with $|\mathrm{leaf}(Tr)|$ leaves, i.e.,*

$$\mathrm{cost}(Tr) = \min\{\mathrm{cost}(Tr') \,:\, Tr' \text{ a tree with } |\mathrm{leaf}(Tr')| = |leaf(Tr)|\}$$

Definition 3. $\mathrm{opt}(n)$ *denotes an arbitrary generalized lopsided tree that has minimum cost among all generalized lopsided trees with n leaves.*

As an example, the trees in the second row in Figure 4 illustrate opt(3), opt(4), and opt(5) given the cost$(\cdot,\cdot)$ and *type*$(\cdot,\cdot)$ function in the first row.

For given T, *cost*, and *type* the problem in which we are interested is: **Given n, characterize the combinatorial structure of** $\mathrm{opt}(n)$ **and propose an algorithm for the construction of** $\mathrm{opt}(n)$**.**

Figure 2 illustrates a case in which $|T| = 1$ and Figure 3 a case in which $|T| = 2$.

The $|T| = 1$ case has been extensively studied in the literature under the name *lopsided trees* (hence, *generalized* lopsided trees for the extension studied here). The name *lopsided trees* was introduced in 1989 by Kapoor and Reingold [18] but the trees themselves have been implicitly present in the literature at least since 1961 when Karp [19] used them to model minimum-cost prefix-free (Huffman) codes in which the length of the edge of the letters in the encoding alphabet were unequal; c_i represented the length of the i^{th} letter in the encoding alphabet (the *idea* of such codes was already present in Shannon [26]).

A major motivation for analyzing lopsided trees was the study of Varn-codes [28] [23]. Suppose that we wish to construct a prefix-free encoding of n symbols using an encoding alphabet of r letters, $\Sigma = \{\alpha_1, \ldots, \alpha_r\}$ in which the length of character α_i is c_i, where the c_is may all be different.

If a symbol is encoded using string $\omega = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_l}$, then $cost(\omega) = \sum_{j\le l} c_{i_j}$ is the length of the string. For example if $r = 2$, $\Sigma = \{0, 1\}$ and $c_1 = c_2 = 1$ then the cost of the string is just the number of bits it contains. This last case is the basic one encountered in regular Huffman encoding e.g., [11].

Now suppose that the n symbols to be encoded are known to occur with equal frequency. The *cost* of the code is then defined to be $\sum_{i\le n} cost(\omega_i)$ (which divided by n is the average cost of transmitting or *length* of a symbol). Given $c_1 \le c_2 \le \cdots \le c_r$, a *Varn-code* for n symbols is a minimum-cost code. Varn codes have been extensively studied in the compression and coding literature([23] [2] both contain large bibliographies).

Such codes can be naturally modelled by lopsided trees in which the length of the edge from a node to its i^{th} child is c_i. See Figure 2. Suppose that v is a leaf in a lopsided tree and the unique path from the tree's root to v first traverses an i_1^{st} edge then an i_2^{nd} edge and so on up to an i_l^{th} edge. We can then associate with this leaf the codeword $\omega = \alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_l}$. The cost of this codeword is exactly the same as the depth of v in the tree, i.e., $\sum_{j\le l} c_{i_j}$. Using this correspondence, every tree with n leaves corresponds to a prefix-free set of n codewords and vice-versa; the cost of the code is exactly equal to the external path length of the tree which we will henceforth call the *cost* of the tree. This correspondence is extensively used, for example, in the analysis of Huffman codes.

A lopsided tree with minimal cost for n leaves will be called an *optimal (lopsided) tree*.

With this correspondence and notation we see that the problems of constructing a Varn code and calculating its cost are equivalent to those of constructing an optimal (lopsided) tree and calculating its cost.

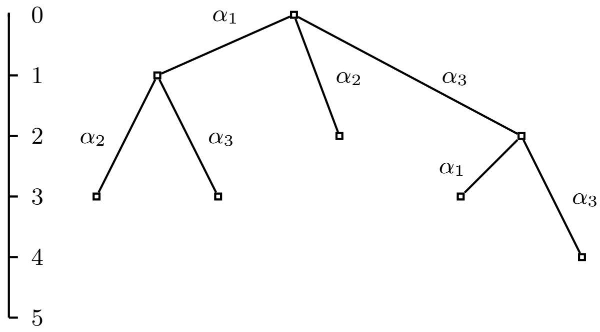
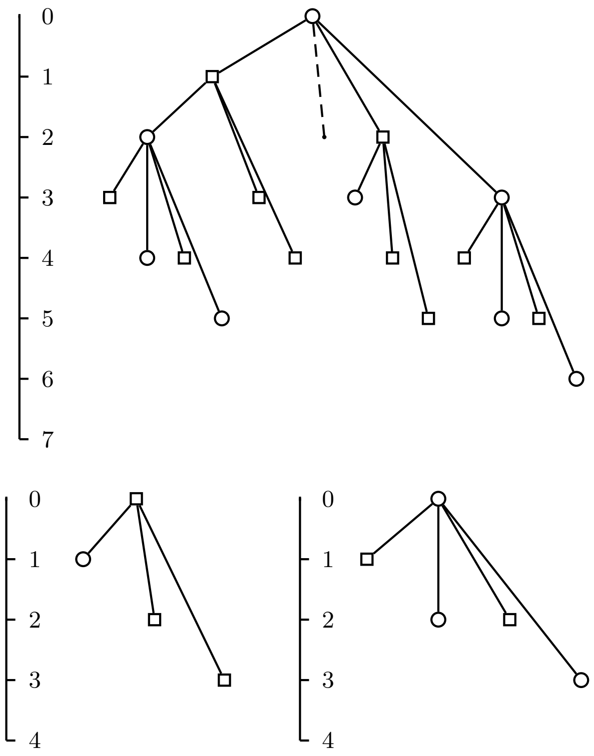


Fig. 2. Example: Using a lopsided tree (with only one type of node) to model a Varn code with letter costs $c_1 = 1$, $c_2 = c_3 = 2$. Edge costs are represented by vertical distances in the diagram. Let $T = \{t\}$. Then $\text{cost}(t, 1) = 1$, and $\text{cost}(t, 2) = \text{cost}(t, 3) = 2$. The code represented by the tree is the set of all external paths, which is $\alpha_1\alpha_2, \alpha_1\alpha_3, \alpha_2, \alpha_3\alpha_1, \alpha_3\alpha_3$. The cost of the tree is $2 + 3 + 3 + 3 + 4 = 15$; its height is 4.



$\text{type}(sq, 1) = circ$ $\quad$ $\text{cost}(sq, 1) = 1$
$\text{type}(sq, 2) = sq$ $\quad$ $\text{cost}(sq, 1) = 2$
$\text{type}(sq, 3) = sq$ $\quad$ $\text{cost}(sq, 1) = 3$

$\text{type}(circ, 1) = sq$ $\quad$ $\text{cost}(circ, 1) = 1$
$\text{type}(circ, 2) = circ$ $\quad$ $\text{cost}(circ, 2) = 2$
$\text{type}(circ, 3) = sq$ $\quad$ $\text{cost}(circ, 2) = 2$
$\text{type}(circ, 4) = circ$ $\quad$ $\text{cost}(circ, 3) = 4$

Fig. 3. A Generalized Lopsided tree (on the top) with $T = \{circle(circ), square(sq)\}$. Cost of the tree is $3 \cdot 3 + 5 \cdot 4 + 4 \cdot 5 + 6 = 51$; height is 6. The two trees on the bottom describe the functions cost and type on the two types of nodes, *(sq)* and *(circ)*. For comparison's sake, the functions are also explicitly written out. **Note that the second child of the root is missing.**

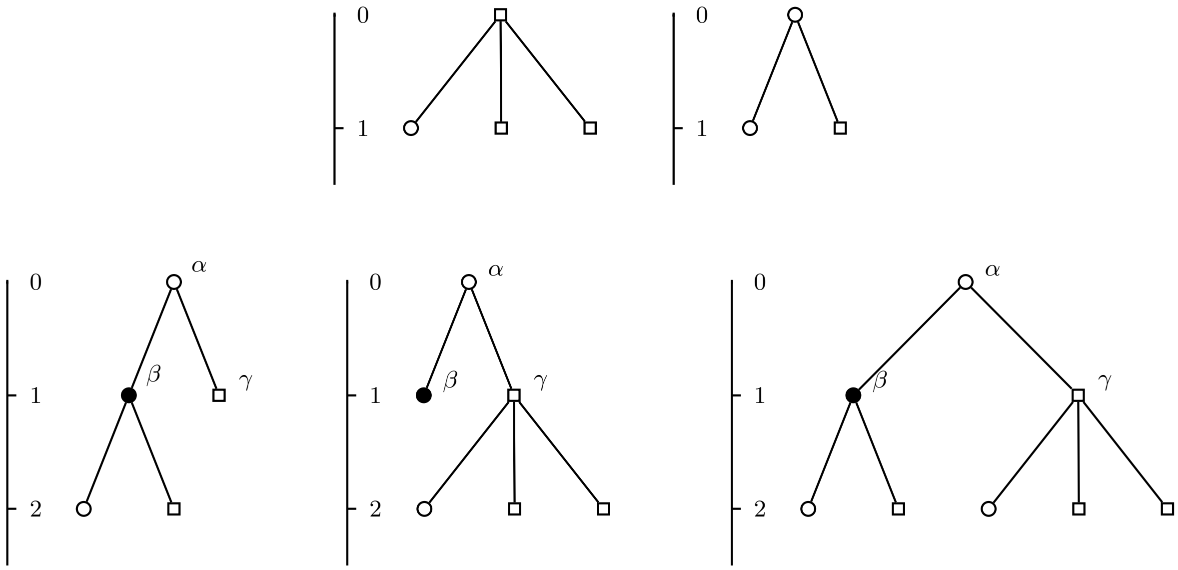


Fig. 4. $T = \{circ, sq\}$ with cost and type being defined by the two trees in the top row. The optimal generalized lopsided trees for $n = 3, 4, 5$ leaves are given in the bottom row. Note that as n increases, β alternates between being an internal node and a leaf. This phenomenon **did not** occur in regular lopsided trees where, once anode became internal, it remained internal.

This is what was studied by most of the papers listed in the first paragraph of this note and this problem is now essentially fully understood.

[15] noted that if $\Sigma = \{\alpha_1, \ldots, \alpha_r\}$ and the c_i are all integral then the Varn coding problem can be modelled by introducing new alphabet $\Sigma' = \{x_1, x_2, \ldots, x_r\}$ and *Varn language* $\mathcal{L} = (x_1^{c_1} + x_2^{c_2} + \ldots + x_r^{c_r})^* \subseteq \Sigma'^*$. A 1-1 correspondence between character α_i and string $x_i^{c_i}$ shows that there is a 1-1 correspondence between Varn codes and prefix-codes restricted to $\mathcal{L}$ and, similarly, between lopsided trees and prefix-codes restricted to $\mathcal{L}$. Thus, the problem of finding the smallest cost prefix-code restricted to $\mathcal{L}$ is equivalent to finding a min-cost lopsided tree. [15] then noted that this was true not just for codes restricted to Varn-languages but that codes restricted to *any* regular $\mathcal{L}$ accepted by a DFA with one accepting state (like Varn Languages) could also (almost) be modelled by lopsided trees, thus permitting using the same techniques to find the cost of such codes. For example let $\mathcal{L} = \Sigma^* P$ for some fixed $P \in \Sigma^*$, i.e., $\mathcal{L}$ is all words that end in P. Such a $\mathcal{L}$ is always accepted by some DFA with one accepting state, so the results in [15] permit finding optimal codes of n words restricted to such $\mathcal{L}$.

A problem that was left open in [15] was how to solve this problem if $\mathcal{L}$ is not in this restricted form. For example, let $\Sigma = \{0, 1\}$. The simple regular language $\mathcal{L} = \Sigma^*(000 + 111)\Sigma^*$ of all words containing at least one occurance of 000 or 111 is not accepted by any DFA with only one accepting state. Another example of a regular language not accepted by any DFA with only one accepting state is $\mathcal{L} = 0^*1(0000^*1)^*0^*$ the language containing all words in which every two consecutive ones are separated by at least 3 zeros.

We now see that the *$\mathcal{L}$-restricted prefix-coding problem* can be modelled using *generalized lopsided trees* for regular languages $\mathcal{L}$. Let $\mathcal{L}$ be accepted by some Deterministic Automaton $\mathcal{M}$ with accepting states $A = \{a_1, a_2, ..., a_n\}$. Without loss of generality we may assume that the empty string $\epsilon \in \mathcal{L}$ so the *start state* of $\mathcal{M}$ is in A. Now define the parameters of the lopsided tree as follows: $T = \{t_1, ..., t_k\}$ where t_i corresponds to state a_i. For any fixed i enumerate, by increasing length (breaking ties arbitrarily) all paths in $\mathcal{M}$ that start at a_i and end at some node $a_j \in A$ *without passing through any other node in A in the interior of the path.* Let these paths be $p_i^{(1)}, p_i^{(2)}, ...$ Set $\mathbf{end}(p) = j$, where a_j terminates path p. We complete the remaining parameters of the generalized trees by defining the functions

$$\text{type}(t_i, j) = t_{\mathbf{end}\left(p_i^{(j)}\right)}, \qquad \text{cost}(t_i, j) = \mathbf{length}\left(p_i^{(j)}\right). \tag{1}$$

There is then a simple one-one correspondence between prefix-free codes restricted to $\mathcal{L}$ and the leaves of the defined generalized lopsided tree with the cost of the code being equal to the cost (external path length) of the tree. Thus, finding the min-cost prefix free code with n words restricted to $\mathcal{L}$ is exactly equivalent to finding the min-cost generalized lopsided tree with n leaves. The remainder of this paper will therefore be devoted to analyzing generalized lopsided trees and how they change as n grows.

As mentioned, the case of regular lopsided trees, i.e., when $|T| = 1$, is well-understood. The difficulty in extending the results on the growth of lopsided trees to that of generalized lopsided trees is that there is a fundamental difference between $|T| = 1$ and $|T| > 1$. Let $\text{opt}(n)$ be the optimal lopsided tree with n nodes and I_n the set of *internal (non-leaf)* nodes in $\text{opt}(n)$. In [7] it was shown that, even though it is not necessarily true that $\text{opt}(n) \subset \text{opt}(n+1)$, i.e., the trees can not be grown greedily, it is always true that $I_n \subseteq I_{n+1}$. So, with a little more analysis, one can "incrementally" construct the trees by greedily growing the set of internal nodes. As Figure 4 illustrates, though, because of the interactions between the various types of nodes, this last property is *not* true for *generalized* lopsided trees. We therefore have to develop a new set of tools to analyze these trees, which is the purpose of this paper.

Note: our correspondence only required that $\mathcal{L}$ be accepted by a Deterministic Automaton with a finite set of accepting states. Since all regular languages are accepted by Deterministic Finite *Automatons our technique will suffice to analyze all restrictions to regular languages.*

We point out that there are many non-regular languages accepted by Non-finite Deterministic Automata (automaton that can have countable infinite states) with a finite set of accepting *states. For example, the language $\mathcal{L}_2$, the set of all words in $\{0,1\}^*$ in which the number of "0"s is equal to the number of "1"s, has this property. Since these can also be modelled by generalized lopsided trees, our technique will work for restrictions to those languages as well.*

2 Definitions

In this section, we introduce definitions that will be used in the sequel. In what follows T, cost and type will be assumed fixed and given.

Definition 4. *Let Tr be a generalized lopsided tree and v a node in Tr.*

- *$internal(Tr)$ is the set of internal nodes of Tr.*
- $\text{type}(v)$ *is the type of v*
- $\text{parent}(v)$ *is the parent of v;note that the parent of the root is undefined*

In the rightmost tree in Figure 4, $\text{type}(\beta) = \textbf{circle}$. $\text{parent}(\beta) = \alpha$, and $internal(TreeC) = \{\alpha, \beta, \gamma\}$.

Our main technique will involve building a larger tree Tr' out of smaller tree Tr by replacing some leaf $v \in \text{leaf}(Tr)$ with some new tree T_2 rooted at a $\text{type}(v)$-node. The increase in the number of leaves from Tr to Tr' is $|\text{leaf}(T_2)| - 1$. The average cost $\text{cost}(T_2)/(|\text{leaf}(T_2)| - 1)$ of the new leaves will be crucial to our analysis and we therefore define

Definition 5. *The **average replacement cost** of tree Tr is*

$$\text{ravg}(Tr) = \text{cost}(Tr)/(|\text{leaf}(Tr) - 1).$$

Intuitively, we prefer to use the subtree with smallest ravg to expand the existing lopsided tree. This motivates us to study the trees with minimum ravg. Recall that the set T represents the collection of types.

Definition 6. *Let $t_k \in T$. Set*

$$MinS(t_k) = \min\{\text{ravg}(Tr), : \text{type}(root(Tr)) = t_k\}. \tag{2}$$

The corresponding tree attaining $MinS(t_k)$ is denoted by $\mathbf{MinS}(t_k)$.

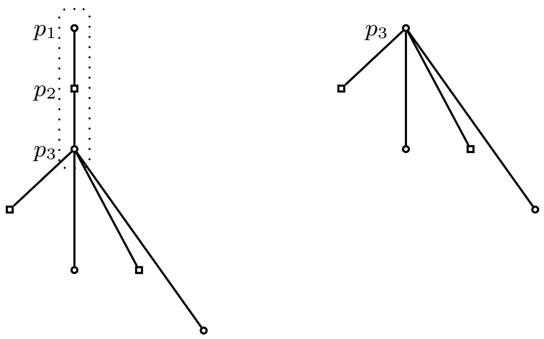


Fig. 5. An example of an improper lopsided tree (left hand side) and a proper tree (right hand side) with better cost that results from collapsing the repeated path.

Note that this definition *does not depend upon* n, but only upon T, cost(), and type(). There might be more than one tree that attains[1] the minimum cost. In such a case, we select an arbitrary tree attaining the minimum that contains the least number of nodes.

We can now define certain essential quantities regulating the growth of lopsided trees.

Definition 7. *Let l be an integer. Set bottom$(l) = l + \min_i\{\lfloor MinS(t_i)\rfloor\}$ and*

$$\mathrm{lev}_{t_k}(l) = \begin{cases} \mathrm{bottom}(l) - MinS(t_k) & \text{if } MinS(t_k) \text{ is an integer;} \\ \lfloor \mathrm{bottom}(l) + 1 - MinS(t_k)\rfloor & \text{if } MinS(t_k) \text{ is not an integer.} \end{cases} \tag{3}$$

In our analysis we often manipulate unused or *free* nodes. In order to do so, we must first introduce a reference tree containing all nodes.

Definition 8. *The **Infinite Generalized Lopsided Tree (ILT)** is the rooted infinite tree such that for each node v in the tree, the ith child's type is type(v, i); the length of the edge connecting v and its ith child is cost(v, i), i.e., every node contains all of its legally defined children.*

We can now define

Definition 9. *A leaf v in the infinite lopsided tree is* free with respect to *tree Tr if $v \notin Tr$ and* parent$(v) \in$ *Tr; the free set of Tr is*

$$free(Tr) = \{v \,:\, v \text{ is free with respect to } Tr\}.$$

In our study of lopsided trees we need to somehow avoid repeated paths that do not contribute any benefit to the tree. We therefore define:

Definition 10. *Figure 5. An improper lopsided tree is a tree containing a path $p_1p_2....p_k$, where $k > |T|$ in which each p_i has only one child (that is, p_1 has only child p_2, p_2 has only child p_3, ...). A* proper lopsided tree *is a tree which is not improper.*

It is not difficult to see that improper trees can not be optimal. We may therefore restrict ourselves to studying proper trees. Note that a proper tree with n leaves can only contain $O(n)$ nodes in total (where the constant in the $O()$ depends upon the tree parameters); we will need this fact in the sequel.

We need one more definition:

Definition 11. *Lopsided Tree parameters T,* cost(), *and* type() *are* non-degenerate *if they satisfy the following condition:*
There exists $N > 0$ such that, $\forall l \geq N$; if the number of nodes on level l in **ILT** *is $\neq 0$ then the number of nodes on level* (bottom$(l) + 1$) *in* **ILT** *is* $\geq \max_i\{|\mathrm{leaf}(\mathbf{MinS}(t_i))|\}$.

[1] It is easy to prove that the minimum *is* attained but we do not do so in this extended abstract

Essentially, the parameters are non-degenerate if deep enough into the infinite tree, the number of nodes per level can't get too small. A technicality occurs because it is quite easy to construct languages in which many levels of the infinite tree have *no* nodes, e.g., the language of all words in $w \in \{0,1\}^*$ in which # of 0's in w equals # of 1's in w. In this language all words have even length, so all odd levels are empty. The condition is stated to handle such cases as well. While the non-degeneracy definition is quite technical, it is usually quite easy to show that most interesting classes of languages satisfy it. For example, $\mathcal{L}$ of the type, $\mathcal{L}$ is "all words in Σ^* containing at least one of the specified patterns $P_1, P_2, \ldots, P_k \in \Sigma^*$" always satisfy this condition.

3 The Structure of Optimal Generalized Trees

Theorem 1. *Let Tr be any optimal tree, v_1, v_2 two nodes in in Tr with* $\mathrm{type}(v_1) = \mathrm{type}(v_2)$. *Then if v_1 is internal in Tr and v_2 is a leaf then* $\mathrm{depth}(v_1) \le \mathrm{depth}(v_2)$. *Furthermore, there exists a constant N, dependent only upon the tree parameters, such that if Tr has $n \ge N$ leaves then*

1. *if v is a leaf in Tr, then $H(Tr) - depth(v) \le \lceil MinS(\mathrm{type}(v))\rceil$ and*
2. *if v is internal in Tr, then $H(Tr) - depth(v) \ge \lceil MinS(\mathrm{type}(v))\rceil - 1$*

This lemma can be read as saying that $\mathrm{opt}(n)$ always has a layered structure, i.e., there exists integers $l_1, \ldots l_{|T|}$, such that (i) all t_i nodes *on or above* level l_i are internal (ii) all t_i nodes *below* level l_i+1 are leaves and (ii) t_i nodes *on* level l_i+1 could be either internal or leaves. Furthermore, $H(Tr)-(l_i + \lceil MinS(t_i)\rceil) \in \{0,1\}$ so (up to an additive factor of 1), it is *independent* of n. See Figure 8.

The proof of this theorem is a quite technical case-by-case one and is omitted from this extended abstract. The basic intuition behind it is quite simple, though. First, it is easy to see that, for fixed type t_i, there must be *some* level l_i above which all t_i-nodes are internal and below which all t_i-nodes are leaves; otherwise, we can swap a higher leaf with a lower internal to get a cheaper tree with the same number of leaves. The actual *location* of l_i is derived by (i) calculations noting that if a leaf v is higher than the given level, then the tree can be improved by turning v into an internal node by rooting a $\mathbf{MinST}(t_i)$ tree at it and removing $|\mathrm{leaf}(\mathbf{MinST}(t_i))|$ leaves from the bottom level of the tree; and (ii) calculations noting that if an internal node v is lower than the specified level then it and all of its descendents can be removed and replaced by new free leaves located at the bottom level or one level below the bottom. The existence of the nodes in (i) to remove and nodes in (ii) to add follows from the non-degeneracy condition.

Definition 12. *Set*

$$V(l) = \{v \in \mathbf{ILT} \mid \mathrm{depth}(v) \le \mathrm{lev}_{\mathrm{type}(v)}(l)\},$$

that is, for each i, $V(l)$ contains exactly all of the t_i nodes with depth $\le \mathrm{lev}_{t_i}(l)$. Now set

$$\mathbf{TreeA}(l) = V(l) \cup \{v | v \in \mathbf{ILT} \ \textit{and}\ \mathrm{parent}(v) \in V(l) \ \textit{and}\ \mathrm{depth}(v) \le \mathrm{bottom}(l)\}$$

and

$$\begin{aligned}\mathbf{TreeB}(l) &= V(l) \cup \{v | v \in \mathbf{ILT} \ \textit{and}\ \mathrm{parent}(v) \in V(l) \ \textit{and}\ \mathrm{depth}(v) \le \mathrm{bottom}(l)+1\} \\ &= \mathbf{TreeA}(l) \cup \{v | v \in \mathbf{ILT} \ \textit{and}\ \mathrm{parent}(v) \in V(l) \ \textit{and}\ \mathrm{depth}(v) = \mathrm{bottom}(l)+1\}\end{aligned}$$

Note that $V(l)$ is the set of internal nodes of $\mathbf{TreeA}(l)$ and also the set of internal nodes of $\mathbf{TreeB}(l)$.

Lemma 1. *Let l be an integer, then*

$$|\mathrm{leaf}(\mathbf{TreeA}(l))| \le |\mathrm{leaf}(\mathbf{TreeB}(l))| \le |\mathrm{leaf}(\mathbf{TreeA}(l+1))|.$$

Even though it is possible that, for some l, $|\mathrm{leaf}(\mathbf{TreeA}(l))| = |\mathrm{leaf}(\mathbf{TreeA}(l+1))|$ it is not difficult to see that, if the non-degeneracy condition is satisfied, $\lim_{l\to\infty} |\mathrm{leaf}(\mathbf{TreeA}(l))| = \infty$ so, for every n we can find an l such that $|\mathrm{leaf}(\mathbf{TreeA}(l))| \le n < |\mathrm{leaf}(\mathbf{TreeA}(l+1))|$.

We can now state our main theorem:

Theorem 2. *Suppose parameters T, $\mathrm{cost}()$, and $\mathrm{type}()$ are non-degenerate.*
For a given integer l, set $A(l) = \mathrm{leaf}(\mathbf{TreeA}(l))$ and $B(l) = \mathrm{leaf}(\mathbf{TreeB}(l))$ Then

1. *If $n = |A(l)|$, then the tree $\mathbf{TreeA}(l)$ is optimal.*
2. *If $|A(l)| < n \leq |B(l)|$, then the tree obtained by appending the $n - |A(l)|$ highest free (with respect to $\mathbf{TreeA}(l)$) leaves to $\mathbf{TreeA}(l)$ is optimal.*
3. *If $|B(l)| < n < |A(l+1)|$,*
   - *All nodes in $V(l)$ are internal in $\mathrm{opt}(n)$.*
   - *No t_i-node whose depth is greater than $\mathrm{lev}_{t_i}(l) + 1$ is internal in $\mathrm{opt}(n)$.*

Figure 6 illustrates an example of this theorem. Figure 6A is $TreeA(3)$; since it contains 81 leaves, it is optimal for $n = 81$. $\mathrm{bottom}(3) = 7$ and there are $m = 13$ children of $V(3)$ on level $(\mathrm{bottom}(3) + 1) = 8$. Optimal trees for $81 < n \leq 81 + m$ are constructed by adding any $n - 81$ level-8 children of nodes in $V(3)$ to $TreeA(3)$. If all m leaves are added then we get $TreeB(3)$ in Figure 6B, which is optimal for $n = 81 + m = 94$ leaves.

Figure 7 is $TreeA(4)$ which has 168 leaves (and is therefore optimal for $n = 168$). Part 3 of the theorem then states that *optimal*(n) for $94 < n \leq 168$ is constructed by starting with $TreeB(3)$ and making internal some t_1-nodes on level six and t_2-nodes on level four. Note that some of the t_1-nodes to be made internal on level six are actually first created by first making some t_2-nodes on level four internal.

This suggests how to find $\mathrm{opt}(n)$ given n. First, find l such that $A(l) \leq n < A(l+1)$. Then, calculate $B(l)$. If $A(l) \leq n \leq B(l)$ then $\mathrm{opt}(n)$ is just $TreeA(l)$ with the highest $n - A(l)$ free leaves in $TreeA(l)$ added to it. The complicated part is when $B(l) < n < A(l+1)$. In this case Theorem 2 tells us that the set of t_i internal nodes in $\mathrm{opt}(n)$ is all of the t_i nodes on or above depth $\mathrm{lev}_{t_i}(l) + 1$ plus some t_i nodes at depth $\mathrm{lev}_{t_i}(l) + 1$. If we exactly knew the set of all internal nodes we can easily construct the tree by appending the n highest leaves. So, our problem reduces down to finding exactly how many t_i internal nodes there are on $\mathrm{lev}_{t_i}(l) + 1$. We therefore define a vector that represents these numbers:

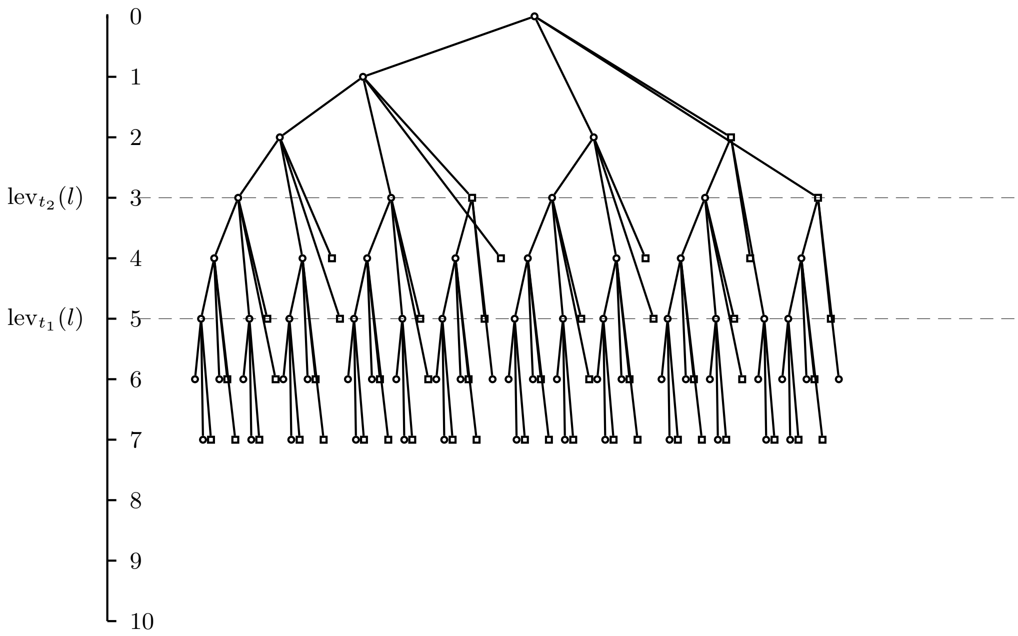
Definition 13. *Let n and l be such that $B(l) < n < A(l+1)$ and Let $\mathrm{opt}(n)$ be an optimal tree for n leaves and v_i be the number of t_i-internal nodes exactly at depth $\mathrm{lev}_{t_i}(l) + 1$. The* feature vector *for $\mathrm{opt}(n)$ is $\mathbf{v} = (v_1, v_2, ..., v_{|T|})$.*

Theorem 2, our combinatorial structure theorem, now immediately yields a straightforward algorithm for constructing $\mathrm{opt}(n)$. The first stage of the algorithm is to find l such that $A(l) \leq n < A(l+1)$. Note that this can be done in $O(|T|^2 l^2)$ time by iteratively building $A(1), A(2), \ldots, A(l+1)$ (l is the first integer such that $n < A(l+1)$). This is done not by building the actual tree but by constructing an *encoding* of the tree that, on each level, keeps track of how many t_i-leaves and t_i internals there are on each level. So, an encoding of a height i tree uses $O(|T|i)$ space. From the definition of $\mathbf{TreeA}(i)$ it is easy to see that its encoding can be built from the encoding of $\mathbf{TreeA}(i-1)$ in $O(|T|^2 i)$ time so l can be found in $\sum_{i \leq l+1} O(|T|^2 i) = O(|T|^2 l^2)$.

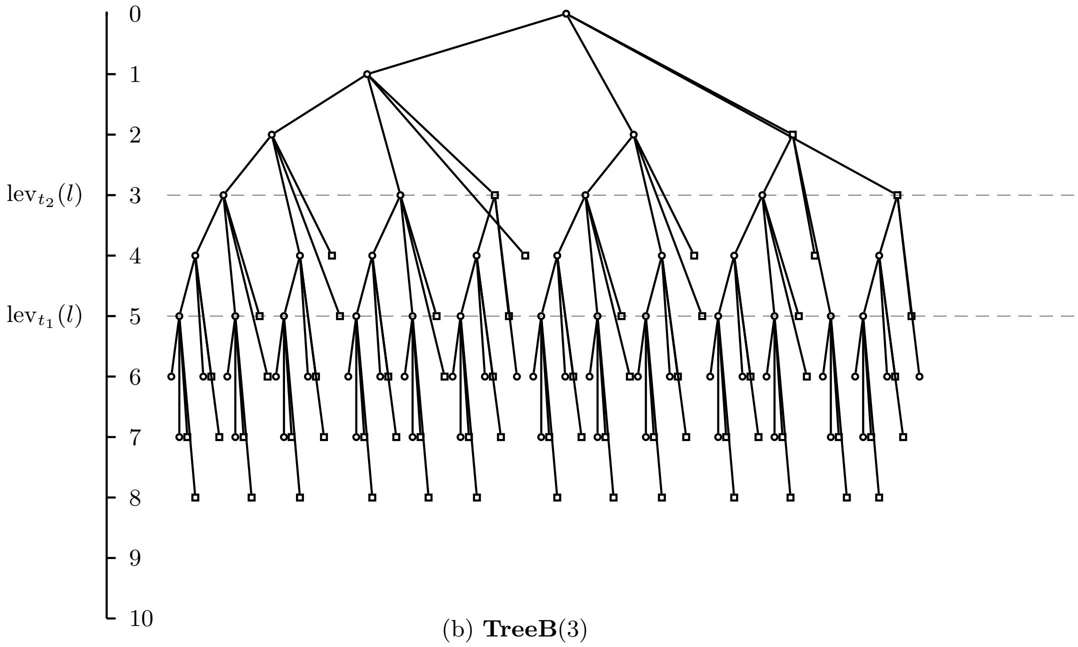
Now note that, because the tree is *proper*, the total number of nodes in $\mathrm{opt}(n)$ is $O(n)$ (where the constants in the $O()$ depend upon the parameters of the lopsided tree) so all of the $v_i = O(n)$. In particular, this means that there are at most $O(n^{|T|})$ possible feature vectors.

Given n, l and some vector $\mathbf{v}$ it is easy to check, in $O(|T|^2 l)$ time, whether a tree with feature vector $\mathbf{v}$ actually exists. This can be done by starting with the encoding of $\mathbf{TreeA}(l)$ and then, working from level $l_{|T|}$ down, using the given $\mathbf{v}$ to decide whether there are enough type-t_i leaves available on level l_i to transform into internals and, if there are, then transforming them. While doing this, we always remember how many leaves L exist *above* the current level. After finishing processing level l_1, we then add the highest available $n - L$ leaves below l_1 if they exist, or find that no such tree exists. If such a tree can be built, then, in $O(|T|l)$ time, its cost can be calculated from the encoding.

Combining the above then gives an $O\left(|T|^2 l^2 n^{|T|}\right)$ algorithm for constructing $\mathrm{opt}(n)$. Simply try every possible feature vector and return the one that gives the minimal cost. The fact that the tree is proper implies that $l = O(n)$ so, in the worst case, this is an $O\left(|T|^2\, n^{|T|+2}\right)$ algorithm. In many interesting cases, e.g., when all nodes have a bounded number of defined children, $l = O(\log n)$ so this beomes an $O\left(\log^2 n\, |T|^2\, n^{|T|}\right)$ algorithm.



(a) $\mathbf{TreeA}(3)$

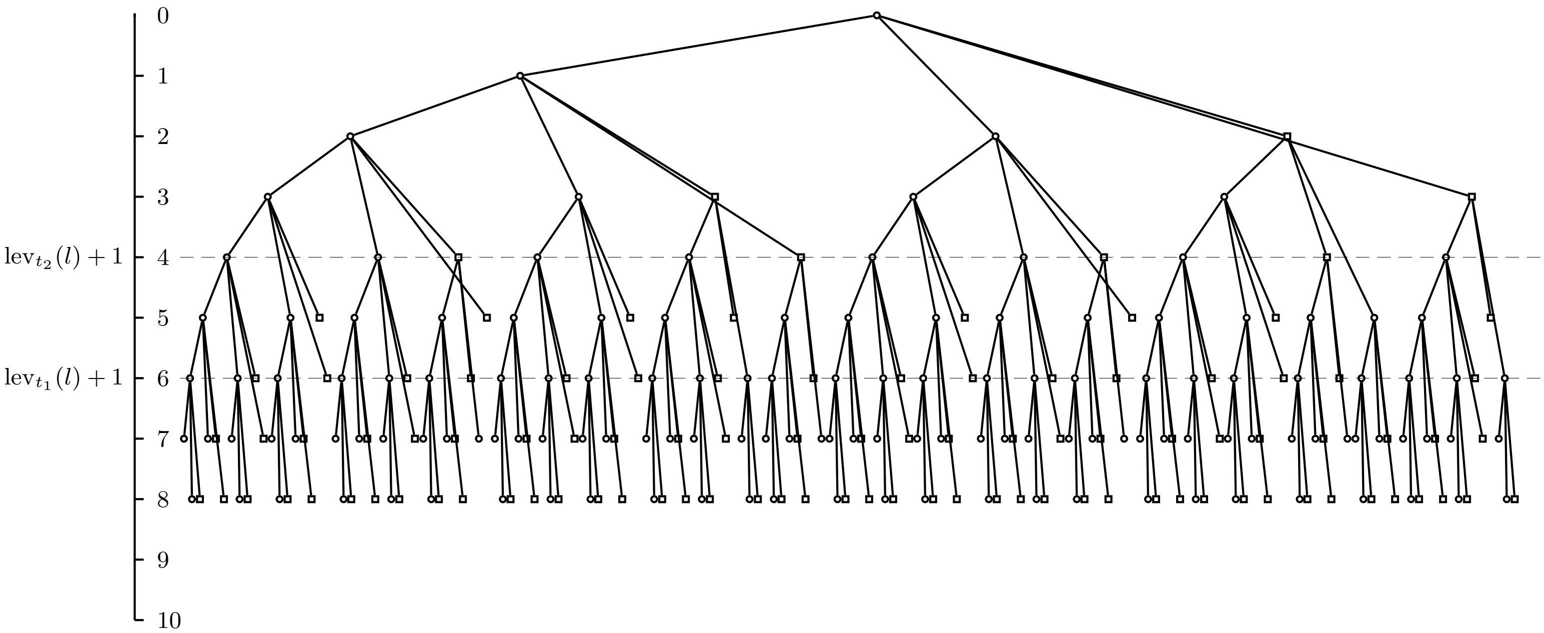


(b) $\mathbf{TreeB}(3)$

$\mathrm{type}(circ, 1) = circ \quad \mathrm{cost}(circ, 1) = 1$
$\mathrm{type}(circ, 2) = circ \quad \mathrm{cost}(circ, 2) = 2$
$\mathrm{type}(circ, 3) = sq \quad \mathrm{cost}(circ, 3) = 2$
$\mathrm{type}(circ, 4) = sq \quad \mathrm{cost}(circ, 4) = 3$

$\mathrm{type}(sq, 1) = circ \quad \mathrm{cost}(sq, 1) = 1$
$\mathrm{type}(sq, 2) = sq \quad \mathrm{cost}(sq, 2) = 2$
$\mathrm{type}(sq, 3) = circ \quad \mathrm{cost}(sq, 3) = 3$

Fig. 6. The trees $\mathbf{TreeA}(l)$ and $\mathbf{TreeB}(l)$ for $l = 3$. Note that the internal sets of these two trees are identical. The leaves with depth 8 in $\mathbf{TreeB}(l)$ are the new leaves.



$\mathrm{type}(circ,1) = circ$	$\mathrm{cost}(circ,1) = 1$	$\mathrm{type}(sq,1) = circ$	$\mathrm{cost}(sq,1) = 1$
$\mathrm{type}(circ,2) = circ$	$\mathrm{cost}(circ,2) = 2$	$\mathrm{type}(sq,2) = sq$	$\mathrm{cost}(sq,2) = 2$
$\mathrm{type}(circ,3) = sq$	$\mathrm{cost}(circ,3) = 2$	$\mathrm{type}(sq,3) = circ$	$\mathrm{cost}(sq,3) = 3$
$\mathrm{type}(circ,4) = sq$	$\mathrm{cost}(circ,4) = 3$		

Fig. 7. TreeA(4). Compare to **TreeA**(3) and **TreeB**(3) in the previous figure.

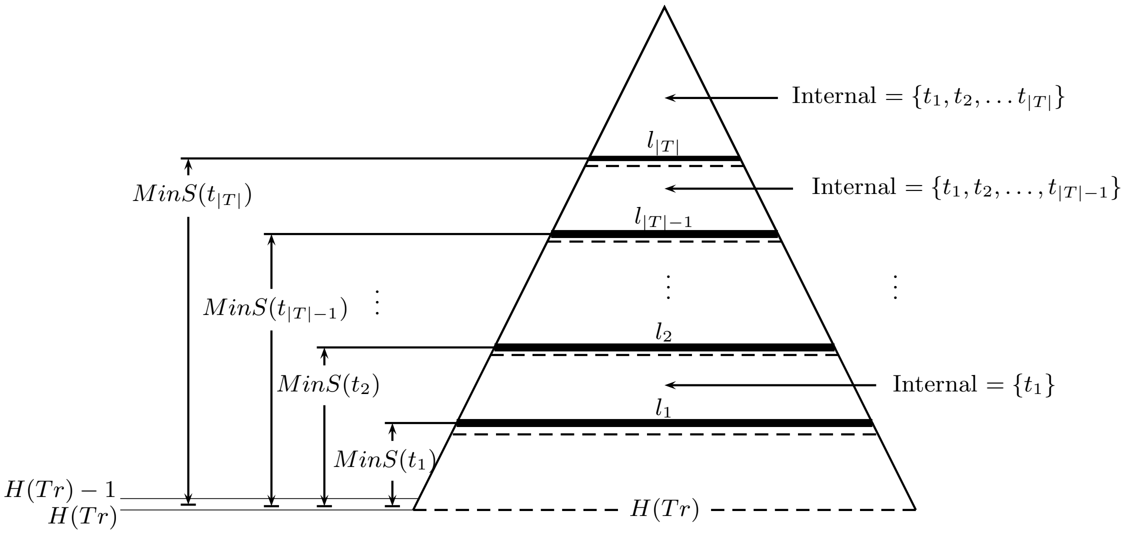


Fig. 8. The structure of an optimal tree. The solid lines are the l_i; all t_i nodes on or above level l_i are internal. The dotted lines are l_i+1; all t_i nodes on or below l_i+1 are leaves. The t_i nodes *on* l_i can be *either* leaves *or* internal nodes. Note that, for all types t_i, $H(Tr)-1 \le l_i + MinST(t_i) \le H(Tr)$ (note that $l_i + MinST(t_i)$ is not necessarily integral).

References

1. Julia Abrahams, "Varn codes and generalized Fibonacci trees," *The Fibonacci Quarterly,* **33**(1) (1995) 21-25.
2. Julia Abrahams, "Code and Parse Trees for Lossless Source Encoding," *Communications in Information and Systems,* **1** (2) (April 2001) 113-146.
3. Doris Altenkamp and Kurt Mehlhorn, "Codes: Unequal Probabilities, Unequal Letter Costs," *Journal of the Association for Computing Machinery*, **27**(3) (July 1980) 412-427.
4. T. Berger and R. W. Yeung, "Optimum "1"-ended Binary Prefix codes," *IEEE Transactions on Information Theory*, **36** (6) (Nov. 1990), 1435-1441.
5. R. M. Capocelli, A. De Santis, and G. Persiano, "Binary Prefix Codes Ending in a "1"," *IEEE Transactions on Information Theory*, **40** (1) (Jul. 1994), 1296-1302.
6. Chan Sze-Lok and M. Golin, "A Dynamic Programming Algorithm for Constructing Optimal "1"-ended Binary Prefix-Free Codes," *IEEE Transactions on Information Theory,* **46** (4) (July 2000) 1637-44.
7. V. S.-N. Choi and Mordecai J. Golin, "Lopsided Trees I: Analyses," *Algorithmica,* **31**, 240-290, 2001.
8. D.M. Choy and C.K. Wong, "Construction of Optimal Alpha-Beta Leaf Trees with Applications to Prefix Codes and Information Retrieval," *SIAM Journal on Computing,* **12**(3) (August 1983) pp. 426-446.
9. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms,* The MIT Press, (1990).
10. N. Cot, "Complexity of the Variable-length Encoding Problem," *Proceedings of the 6th Southeast Conference on Combinatorics, Graph Theory and Computing,* (1975) 211-224.
11. Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory,* John Wiley and Sons. (1991).
12. I. Csiszár, "Simple Proofs of Some Theorems on Noiseless Channels," *Inform. Contr.,* bf 14 (1969) pp. 285-298
13. I. Csiszár, G. Katona and G. Tsunády, "Information Sources with Different Cost Scales and the Principle of Conservation of Energy," *Z. Wahrscheinlichkeitstheorie verw,* **12**, (1969) pp. 185-222
14. M. Golin and G. Rote, "A Dynamic Programming Algorithm for Constructing Optimal Prefix-Free Codes for Unequal Letter Costs," *IEEE Transactions on Information Theory,* **44**(5) (September 1998) 1770-1781.
15. M. Golin and HyeonSuk Na, "Optimal prefix-free codes that end in a specified pattern and similar problems: the uniform probability case.," *Data Compression Conference, DCC'2001,* (March 2001) 143-152.
16. M. Golin and Assaf Schuster "Optimal Point-to-Point Broadcast Algorithms via Lopsided Trees," *Discrete Applied Mathematics,* **93** (1999) 233-263.
17. M. Golin and N. Young, "Prefix Codes: Equiprobable Words, Unequal Letter Costs," *SIAM Journal on Computing,* **25**(6) (December 1996) 1281-1292.
18. Sanjiv Kapoor and Edward Reingold, "Optimum Lopsided Binary Trees," *Journal of the Association for Computing Machinery,* **36** (3) (July 1989) 573-590.
19. R. M. Karp, "Minimum-redundancy coding for the discrete noiseless channel," *IRE Transactions on Information Theory,* **7**, pp. 27-39, 1961
20. R. M. Krause, "Channels Which Transmit Letters of Unequal Duration," *Inform. Contr.,* **5** (1962) pp. 13-24,
21. Harry R. Lewis and Christos H. Papadimitriou, *Elements of the Theory of Computation (2nd ed.),* Prentice Hall. (1998).
22. Y. Perl, M. R. Garey, and S. Even. "Efficient Generation of Optimal Prefix Code: Equiprobable Words Using Unequal Cost Letters," *Journal of the Association for Computing Machinery*, 22(2):202–214, April 1975.
23. Serap A. Savari, "Some Notes on Varn Coding," *IEEE Transactions on Information Theory,* **40**(1) (Jan. 1994) 181-186.
24. Serap A. Savari, "A Probabilistic Approach to Some Asymptotics in Noiseless Communications," *IEEE Transactions on Information Theory,* **46**(4) (July 2000) 1246-1262.
25. Raymond Yeung, *A First Course in Information Theory,* Kluwer Academic/Plenum Publishers, New York. (2002).
26. C.E. Shannon "A Mathematical Theory of Communication," *Bell System Technical Journal* **27** (1948) 379-423, 623-656.
27. L. E. Stanfel, "Tree Structures for Optimal Searching," *Journal of the Association for Computing Machinery,* **17**(3) (July 1970) 508-517.
28. B.F. Varn, "Optimal Variable Length Codes (Arbitrary Symbol Costs and Equal Code Word Probabilities)," *Informat. Contr.,* **19** (1971) 289-301