Optimal prefix-free codes that end in a specified pattern and similar problems: the uniform probability case (Extended Abstract) *

Mordecai J. Golin
Dept. of Computer Science
HKUST, Clear Water Bay
Kowloon, Hong Kong
email: golin@cs.ust.hk

HyeonSuk Na
Dept. of Mathematics
Postech, Korea
email: hsnaa@postech.ac.kr

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 $\mathcal{L}$. We examine how, given certain types of $\mathcal{L}$, the structure of the minimum-cost code changes as n, the number of codewords, grows.

1 Introduction

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 $\text{Cost}(C,P) = \sum_i |w_i| \cdot p_i$ where $|w|$ is the length of word w; $\text{Cost}(C,P)$ 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 $\text{Cost}(C,P)$. When the codewords are equiprobable, i.e., $\forall i$, $p_i = 1/n$, then $\text{Cost}(C,P) = \frac{1}{n}\sum_i |w_i|$ and $\text{Cost}(C,P)$ is minimized when $\sum_i |w_i|$ is minimized. We will call such a code an *optimal uniform-cost code.*

*The work of both authors was partially supported by HK RGC CERG grants HKUST6137/98E and HKUST6162/00E and HKUST DAG98/99.EG23.

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 $\mathcal{L} \subseteq \Sigma^*$. Given a fixed $\mathcal{L}$, what is the optimal (min-cost) code C_n containing n words in $\mathcal{L}$? How does C_n change with n? How does $\text{Cost}(C_n, P)$ grow as a function of n?

The original problem that motivated us was a very simple one; the 1-*ended* binary-code problem in which every word in C_n must end in a 1, i.e., $\mathcal{L} = (1+0)^*1$. This type of prefix-free code has recently garnered interest [1] [2] [3] because, among other reasons, it can be used in the design of self-synchronizing codes. This problem can be generalized to the examination of optimal prefix-free codes for all $\mathcal{L} \subset \Sigma^*$ that end with a specified pattern $\mathcal{P}$, i.e., $\mathcal{L} = \Sigma^* \mathcal{P}$. This generalization seems to be quite difficult to solve (no one seems to know how difficult, though. For example, it is unkown if it is NP-Hard). We therefore restricted ourselves to the problem of finding optimal *uniform cost* prefix-free codes for these cases. We found that the techniques developed for this problem will actually work for a much wider range of languages, more specifically for all regular languages that are accepted by a deterministic finite automaton (DFA) of a special class that will be described in the next section. The answer to all of the questions above will be shown to depend upon simple parameters of the DFA. That is, given a language $\mathcal{L}$, if we know a DFA $\mathcal{M}_{\mathcal{L}}$ in our particular class that accepts $\mathcal{L}$, we can calculate parameters $h_{\mathcal{M}_{\mathcal{L}}}$ and $d_{\mathcal{M}_{\mathcal{L}}}$ of $\mathcal{M}_{\mathcal{L}}$ and, from these parameters, exactly derive the structure of C_n, the optimal uniform-cost prefix-free code containing n words such that $C_n \subseteq \mathcal{L}$.

In the next section we quickly review the definition of deterministic finite automatons and describe the special type of automatons that accept the languages that we will be studying. In section 3 we describe a standard transformation from prefix-free codes into trees, transform our prefix-free code problem into a min-cost tree problem and then introduce some useful definitions. Section 4 then states our main lemmas and theorems on tree structures.

Note: In this extended abstract we omit the proofs of most of the lemmas and theorems. The proofs can be found in the full version of this paper.

2 A Quick Introduction to DFAs

In this section we give a quick description of Deterministic Finite Automatons (DFAs) and then describe the particular DFAs we will be interested in. Our notation follows that of [5]. A DFA $\mathcal{M}$ is a 5-tuple $(Q, q_0, F, \Sigma, \delta)$ in which

- Q is a finite set of *states*, $q_0 \in Q$ is the *start state*,
- $F \subseteq Q$ is a distinguished set of *final* or *accepting states*,
- Σ is a finite *input alphabet*,

- δ is a function from $Q \times \Sigma \to Q$, called the *transition function.*

Informally $\mathcal{M}$ can be visualized as a directed graph whose nodes are the states Q; there is one *start node* q_0 and a set of *accepting nodes, F.* Every node q has $|\Sigma|$ directed edges leaving it, one labelled with each of the distinct characters of Σ.

Let $w = \sigma_1\sigma_2 \dots \sigma_k \in \Sigma^*$ be a word. The DFA starts in state q_0 and reads the characters of w one at a time. Each time it reads a character, it follows the edge labelled by that character from its current node to a new node. Let q be the state that $\mathcal{M}$ ends in after scanning all of w. w is *accepted* by $\mathcal{M}$ if and only if $q \in F$. The language $\mathcal{L}_{\mathcal{M}}$ is the the set of all words accepted by $\mathcal{M}$. It is known that a language is *regular* if and only if it is accepted by some DFA.

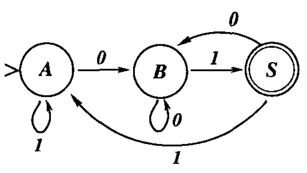
It is also well known that given $\mathcal{P} \in \Sigma^*$, there is a *pattern matching* DFA with $|\mathcal{P}| + 1$ states and only one final state that accepts the language $\mathcal{L} = \Sigma^* \mathcal{P}$. See [5] for details. (We note that this DFA can be easily modified to accept the language $\mathcal{L} = \Sigma^* \mathcal{P} \Sigma^*$, i.e., the language consisting of all words that *contain* $\mathcal{P}$ as opposed to just those that *end* with $\mathcal{P}$, by redirecting all edges leaving the final state so that they point to itself). Figures 1 and 3 illustrate the DFAs accepting the strings $(0+1)^*01$ and $(a+b+c)^*abca$, i.e., the languages of all words ending, respectively, in 01 and $abca$.

Given a language $\mathcal{L}$ and integer n, our goal will be to find the optimal uniform-cost prefix free code with all n words in $\mathcal{L}$. We will be able to do this for all regular languages that are accepted by a DFA that has only one final state and also satisfies one other technical condition given in Section 3 (this non-degeneracy condition is needed to guarantee that the number of words in the language grow with the word size.) The regular languages satisfying this condition will include the two types of languages just described; "all words ending with a given pattern $\mathcal{P}$," and "all words containing a given pattern $\mathcal{P}$".

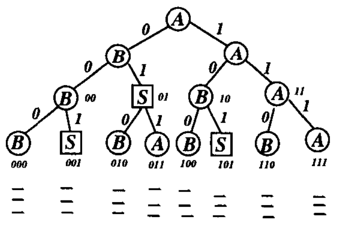
It will also contain another interesting large class. Suppose that $\Sigma = \{a_1, \dots, a_k\}$. Let $r_1, \dots, r_k$ be a sequence of nonnegative integers. Now set $\mathcal{L} = (a_1^{r_1} + \cdots + a_k^{r_k})^*$. There is a DFA with only one final state that accepts this language as well. Figure 2 gives the example of $(00 + 111)^*$ which has $(r_1, r_2) = (2, 3)$.

These languages, which we will call *Varn languages*, can all be accepted by DFAs that satisfy the non-degeneracy condition in Section 3 so our result will also characterize the optimal prefix-free codes restricted by this type of $\mathcal{L}$.

We note that there is another interpretation of optimal codes over Varn Languages. Suppose that we are given k characters, $\alpha_1, \alpha_2, \dots, \alpha_k$ with unequal lengths/costs such that the respective costs of the characters are $r_1, r_2 \dots, r_k$. The cost of a word over these characters is defined to be the sum of the lengths of the characters in the word. The cost of a code is the sum of the costs of its codewords. An equiprobable minimum-cost code with n words over these characters is known as a *Varn Code*, hence Varn languages, and there is a large body of work describing how to find Varn codes and their costs [4] [6] [7] [8] [9] [10]. By fixing

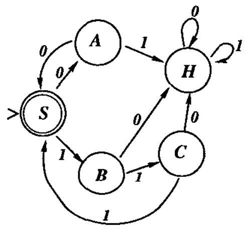


(a) DFA $\mathcal{M}_1$ with accepting state S and $\Sigma = \{0,1\}$.

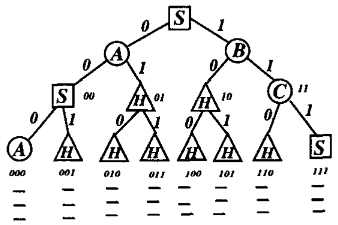


(b) Infinite tree $T_{\mathcal{M}_1}$. The nodes in states A or B *bad* and those in state S *good*.

Figure 1: $\mathcal{M}_1$ accepts binary strings ending in '01'.

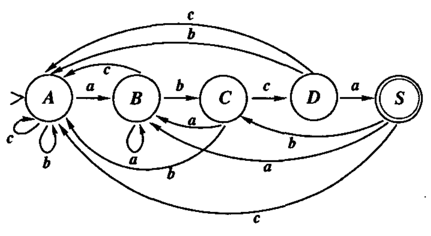


(a) DFA $\mathcal{M}_2$ with accepting state S, $\Sigma = \{0,1\}$ and sink state H.

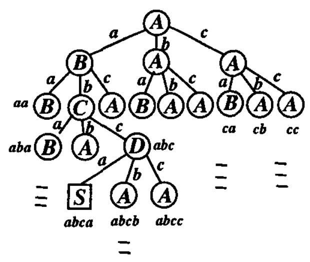


(b) Infinite tree $T_{\mathcal{M}_2}$.

Figure 2: $\mathcal{M}_2$ accepts the *Varn language* $L = (00 + 111)^*$.



(a) DFA $\mathcal{M}_3$ with accepting state S and $\Sigma = \{a,b,c\}$.



(b) Infinite tree $T_{\mathcal{M}_3}$.

Figure 3: $\mathcal{M}_3$ accepts strings ending in '*abca*'.

$\mathcal{L}$ as above we see that our results provide yet another complete characterization for Varn codes with integral letter costs.

3 Definitions

It will be helpful to move our problem from the domain of codes to that of trees. Let $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_k\}$, $\mathcal{L} \subseteq \Sigma^*$ be a language and $\mathcal{M}$ a DFA that accepts $\mathcal{L}$. In what follows we will strongly use the assumption that $\mathcal{M}$ has only one final state. Set $T_{\mathcal{M}}$ to be the rooted infinite r-ary tree such that: (i) the root corresponds to the empty string; (ii) every node has k edges coming out of it with the i-th edge being labelled by the letter $\sigma_i \in \Sigma$; (iii) each node is labelled by the string generated by traversing the path from the root to that node. (Figures 1(b), 2(b) and 3(b) provide examples.)

There is then a one-to-one correspondence between the nodes and words in Σ^*. Let $w \in \Sigma^*$ and $u \in T_{\mathcal{M}}$ its associated node. Define *depth*(u) to be the number of edges in the path from the root to u. Then, by definition, *depth*$(u) = |w|$, the number of letters in w.

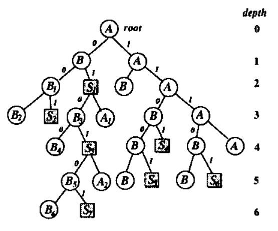
In what follows we will identify node u with word w, e.g., when writing "u is in $\mathcal{L}$", the intent is that "the word w corresponding to u is in $\mathcal{L}$". The *state* of node u will be the state that $\mathcal{M}$ is in after reading w. Node u will be *good* if and only if its associated string is accepted by $\mathcal{M}$.

Definition 1 *See Figure 4.*

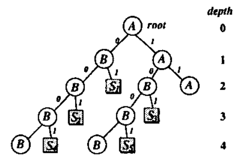
- *Let $T' \subset T_{\mathcal{M}}$ be a subtree that contains the root of $T_{\mathcal{M}}$. The* external path length*(EPL) or* cost, *$C(T')$, of T' is the sum of the depths of all leaves in T' that are* good *nodes.*
- *A rooted subtree $T' \subset T_{\mathcal{M}}$ with n good leaves is called* optimal *if T' has the minimal cost among all the subtrees of $T_{\mathcal{M}}$ having n good leaves. We use T_n to denote such a tree (in the case of there being many optimal trees with n leaves, T_n will denote some arbitrary one).*

A word w' is a prefix of word w if and only if the corresponding node u' is on the path from the root of $T_{\mathcal{M}}$ to u. The good leaves of any subtree T' therefore correspond to some prefix-free set of codewords in $\mathcal{L}$ with the EPL of the tree equaling the sum of the codeword costs. Similarly, given a prefix-free code with n words there is some (often many) tree(s) whose good leaves are exactly the nodes corresponding to the codewords. Thus, the problem of finding the min cost prefix-free code with n words is the same as that of finding an optimal tree with n good leaves. This is the problem that we will actually be solving.

Figure 5 illustrates a sequence of optimal binary trees for the language containing all words that end in 01. That is, if left edges are labelled with 0s and



(a) T'



(b) T_5

Figure 4: Two rooted subtrees of $T_{\mathcal{M}_1}$ with 5 good leaves. $C(T') = 3+4+5+5+6 = 23$; $C(T_5) = 2+3+3+4+4 = 16$. It can be shown that T_5 is optimal for all trees with 5 good leaves.

right edges with 1s then the path from the root to every good leaf must end with a left-edge, right-edge sequence. From this figure we note that sometimes T_{n+1} can be constructed by simply greedily adding one unused good leaf to T_n but sometimes the construction actually requires *deleting* old nodes. Examining the sequence of trees it seems as if there is some repetitive rule involved that determines when we add nodes and when we delete them. Such a rule does exists and depends upon $\mathcal{M}$. Its derivation is the main result of this paper.

Definition 2 *Let u, v be good nodes of $T_{\mathcal{M}}$. v is the* good parent *of u if v is an ancestor of u and there is no good node between u and v. If u has no good ancestor then set the root to be goodparent(u).*

Let v be an arbitrary good node of $T_{\mathcal{M}}$. Consider the good children of v in $T_{\mathcal{M}}$; these are the good descendants of v whose closest good ancestor (travelling up the tree) is v. Label the good children of v as $u_1, u_2, u_3, \cdots$ in order of increasing depth, breaking ties arbitrarily. Let $l_i = depth(u_i) - depth(v)$. Define

$$\forall i \geq 2, \quad x_i = \frac{l_1 + l_2 + \cdots + l_i}{i-1}.$$

Note that the values x_i depend only upon $\mathcal{M}$ and not upon the particular starting vertex v.

We can then prove (this is a generalization of a lemma in [4])

Lemma 1 *There exists $d_{\mathcal{M}}$ (simply d), such that*
$x_2 > x_3 > \cdots > x_d \leq x_{d+1} \leq \cdots$. *(If $x_2 \leq x_3$, then $d = 2$.)*

We will also need the following definitions:

Definition 3 *For d and x_d defined in Lemma 1 set $h_{\mathcal{M}} = \lfloor x_d \rfloor$ and $\lambda_{\mathcal{M}} = (d-1)(h_{\mathcal{M}}+1) - (l_1 + \cdots + l_d)$. For $\mathcal{M}$ with $d = 2$, note that $x_d = l_1 + l_2 = h_{\mathcal{M}}$.*

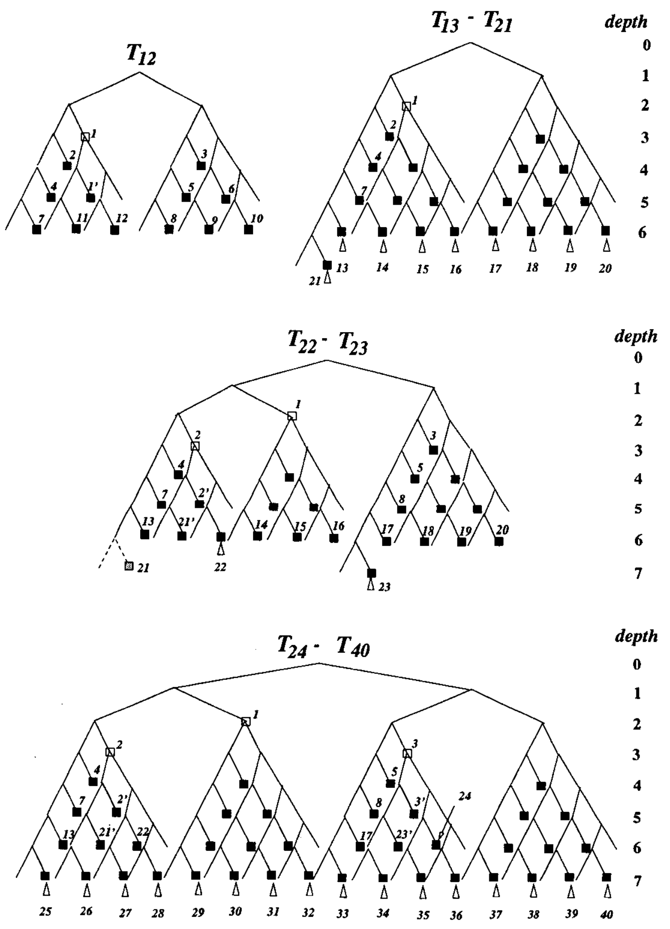


Figure 5: For $\mathcal{M}_1$, this figure illustrates a sequence $\{T_n\}$ of optimal trees for $12 \leq n \leq 40$. Left edges should be thought of as labelled with 0s; right edges with 1s. $\mathcal{M}_1$ accepts words that end with "01". Starting with optimal tree T_{12}, optimal trees T_{13}-T_{21} are constructed greedily by simply always adding the highest available good leaf. To construct T_{22}, though, we must *erase* leaf 21, make leaf 2 internal and add its good children 2′, 21′ and 22. To construct T_{23} we again simply add the highest available unused good leaf. To construct T_{24}, though, we need again to erase a good leaf, this time 23, make 3 internal and add the new good leaves 3′, 23′ and 24. Optimal trees T_{25}-T_{40} are then constructed greedily. We see from this example that there are times when the greedy strategy works and there are times when it is better to make a highest good leaf internal by adding some good children of that node. These moments are exactly when we move from $T_{R_{|gV_{t-1}|}}$ to $T_{L_{|gV_{t-1}|+1}}$ and from $T_{R_{|gV_t|-1}}$ to $T_{L_{|gV_t|}}$. (See Definition 5.)

Definition 4 *Let V be a set of nodes in $T_{\mathcal{M}}$ and T' a rooted subtree of $T_{\mathcal{M}}$.*

- $\mathcal{GLEAF}(V) := \{u \text{ is a good node in } T_{\mathcal{M}} \mid goodparent(u) \in V, u \notin V\}$,
- $\mathcal{GLEAF}_r(V)$ *is a set of r nodes of smallest depth in* $\mathcal{GLEAF}(V)$,

For a set of nodes V, let $|V|$ be the number of nodes in V.. Let $gV_i := \{u \text{ is a good node in } T_{\mathcal{M}} \mid depth(u) \leq i\}$. Given $\mathcal{M}$ and its corresponding tree $T_{\mathcal{M}}$, we say that $\mathcal{M}$ *satisfies the* **Non-degeneracy Condition** if there exists $N_{\mathcal{M}} \in \mathbb{N}$ such that for $\forall N \geq N_{\mathcal{M}}$,

- if $N \leq t \leq N + h_{\mathcal{M}} - 1$,
  then $|\{u \in \mathcal{GLEAF}(gV_{N-1}) \mid depth(u) = t\}| \geq 2$,
- if $t \geq N + h_{\mathcal{M}}$
  then $|\{u \in \mathcal{GLEAF}(gV_{N-1}) \mid depth(u) = t\}| \geq (d-1)$.

This technical condition is needed in order to prove our results. If it does not hold then an analogous, but somewhat messier result, can be proven. The important fact for us is that

Lemma 2 *The following three types of languages are all accepted by a DFA that satisfies the non-degeneracy condition: all words that end with a given pattern $\mathcal{P}$; all words that contain a given pattern $\mathcal{P}$; all Varn languages.*

We point out that the value $N_{\mathcal{M}}$, in the Non-degeneracy Condition is very DFA specific but can usually be calculated efficiently. We also point out that the three types of languages listed in the lemma are not the only ones that satisfy the condition; many others do as well. For example, the language $\mathcal{L} \subset \{0,1\}^*$, containing all words whose number of 1s is divisible by 2 and whose number of 0s is divisible by 3 is also accepted by a DFA with one final state that satisfies the non-degeneracy condition and would therefore be covered by our result.

4 The Main result

In this section we state, without proof, our main result. Before starting we note that, if $\mathcal{M}$ satisfies the non-degeneracy condition then, $\forall t \geq N_{\mathcal{M}} + 1$, there exist at least two good nodes on level t so $|gV_{t-1}| < |gV_t|$ and $|gV_{t-1}| + 1 \leq |gV_t| - 1$.

Now recall that $h_{\mathcal{M}} = \lfloor x_d \rfloor$. The next definition will depend upon whether $x_d = h_{\mathcal{M}}$ or $x_d > h_{\mathcal{M}}$.

Definition 5 *Let $\mathcal{M}$ be a DFA with one final state that satisfies the non-degeneracy condition. We define a sequence of pairs L_m, R_m as follows:*

Case1: If $x_d = h_{\mathcal{M}}$ then

1. *If, for some* t, $m = |gV_{t-1}|$:

$$\begin{aligned} L_m &= |\{u \in \mathcal{GLEAF}(gV_{t-1}) \mid depth(u) \leq t + h_{\mathcal{M}} - 2\}|, \\ R_m &= |\{u \in \mathcal{GLEAF}(gV_{t-1}) \mid depth(u) \leq t + h_{\mathcal{M}} - 1\}| + d - 2. \end{aligned}$$

2. *Otherwise* $|gV_{t-1}| + 1 \leq m \leq |gV_t| - 1$: *Let* $\sigma = m - |gV_{t-1}|$ *and set*

$$\begin{aligned} L_m &= |\{u \in \mathcal{GLEAF}(gV_{t-1} \cup \mathcal{GLEAF}_\sigma(gV_{t-1})) \mid depth(u) \leq t + h_{\mathcal{M}} - 1\}|, \\ R_m &= |\{u \in \mathcal{GLEAF}(gV_{t-1} \cup \mathcal{GLEAF}_\sigma(gV_{t-1})) \mid depth(u) \leq t + h_{\mathcal{M}} - 1\}| + d - 2. \end{aligned}$$

Case2: If $x_d > h_{\mathcal{M}}$ *then*

1. *If, for some* t, $m = |gV_{t-1}|$ *set:*

$$\begin{aligned} L_m &= |\{u \in \mathcal{GLEAF}(gV_{t-1}) \mid depth(u) \leq t + h_{\mathcal{M}} - 1\}| - \lambda_{\mathcal{M}}, \\ R_m &= |\{u \in \mathcal{GLEAF}(gV_{t-1}) \mid depth(u) \leq t + h_{\mathcal{M}}\}| + d - 2 - \lambda_{\mathcal{M}}. \end{aligned}$$

2. *Otherwise* $|gV_{t-1}| + 1 \leq m \leq |gV_t| - 1$: *Let* $\sigma = m - |gV_{t-1}|$ *and set*

$$\begin{aligned} L_m &= |\{u \in \mathcal{GLEAF}(gV_{t-1} \cup \mathcal{GLEAF}_\sigma(gV_{t-1})) \mid depth(u) \leq t + h_{\mathcal{M}}\}| - \lambda_{\mathcal{M}} \\ R_m &= |\{u \in \mathcal{GLEAF}(gV_{t-1} \cup \mathcal{GLEAF}_\sigma(gV_{t-1})) \mid depth(u) \leq t + h_{\mathcal{M}}\}| + d - 2 - \lambda_{\mathcal{M}}. \end{aligned}$$

Lemma 3 $\forall m \geq |gV_{N_{\mathcal{M}}}|$, $\quad L_m < R_m + 1 = L_{m+1}$.

This Lemma implies that, for each $n \geq L_{|gV_{N_{\mathcal{M}}}|}$, there exists a unique m such that $n \in [L_m, R_m]$. We then define:

Definition 6 *Let* m, L_m *and* R_m *be as in Definition 5. For every* m, *define* W_m *and* T_n^m *as follows:*

- W_m *is the set consisting of the root and* m *highest, i.e., smallest depth non-root good nodes.*
- *For* $L_m \leq n \leq R_m$, $\quad T_n^m = W_m \cup \mathcal{GLEAF}_n(W_m)$

This is important because of the following:

Theorem 1 *Let* $n \in [L_m, R_m]$. *Then* T_n^m *is optimal for* n *leaves.*

Lemma 3 and Theorem 1 are the heart of our result (and require the majority of the full paper to prove). They state that an optimal tree for n good leaves, and therefore a minimum cost prefix-free code, can be found first by (i) calculating m such that $n \in [L_m, R_m]$, setting W_m to be the set of the root and m highest good nodes and then (ii) choosing the n highest good children of W_m that are not in W_m. These n good children will be the leaves of the optimal tree (and thus the tree can be reconstructed from them).

It is beyond the scope of this extended abstract to show but, with this Lemma and Theorem in hand, we can, given $\mathcal{L}$ and n, (a) derive a $O(n)$ time algorithm for constructing T_n^m and also (b) derive an exact closed formula $T(n) = C(T_n^m)$ for the cost of the optimal trees/prefix-free codes as a function of n.

To recap, these results derive what we started out searching for, a combinatorial description of the optimal uniform cost prefix-free codes/trees under the assumption that the codewords must all end in an arbitrary pattern, $\mathcal{P}$. By recasting our problem we showed that our analysis actually worked for a much larger set of restrictions, e.g., when the codewords belong to any language $\mathcal{L}$ that is accepted by a certain subclass of DFAs. One obvious continuation of our investigations, and one that we are currently attempting, is to extend our analysis to work for *any* $\mathcal{L}$ that is accepted by a DFA, i.e., to all regular languages $\mathcal{L}$. This will require additional techniques since our results until now have been strongly dependent upon the fact that the accepting machine $\mathcal{M}$ has only one final state.

References

[1] T. Berger and R. W. Yeung, "Optimum "1"-ended Binary Prefix codes," *IEEE Transactions on Information Theory*, **36** (6) (Nov. 1990), 1435-1441.

[2] 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.

[3] 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.

[4] V. S.-N. Choi and Mordecai J. Golin, "Lopsided Trees I: Analyses," To appear in *Algorithmica*.

[5] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, The MIT Press, (1990).

[6] Sanjiv Kapoor and Edward Reingold, "Optimum Lopsided Binary Trees," *Journal of the Association for Computing Machinery*, **36** (3) (July 1989) 573-590.

[7] 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.

[8] Serap A. Savari, "Some Notes on Varn Coding," *IEEE Transactions on Information Theory*, **40**(1) (Jan. 1994) 181-186.

[9] Serap A. Savari, "A Probabilistic Approach to Some Asymptotics in Noiseless Communications," *IEEE Transactions on Information Theory*, **46**(4) (July 2000) 1246-1262.

[10] B.F. Varn, "Optimal Variable Length Codes (Arbitrary Symbol Costs and Equal Code Word Probabilities)," *Informat. Contr.*, **19** (1971) 289-301