Generalizations of Length Limited Huffman Coding for Hierarchical Memory Settings

Shashwat Banchhor
Department of Computer Science, Indian Institute of Technology, New Delhi, India

Rishikesh Gajjala
Indian Institute of Science, Bangalore, India
Department of Computer Science, Indian Institute of Technology, New Delhi, India

Yogish Sabharwal
IBM Research, New Delhi, India

Sandeep Sen[1]
Department of Computer Science, Shiv Nadar University, Uttar Pradesh, India

Abstract

In this paper, we study the problem of designing prefix-free encoding schemes having minimum average code length that can be decoded efficiently under a decode cost model that captures memory hierarchy induced cost functions. We also study a special case of this problem that is closely related to the length limited Huffman coding (LLHC) problem; we call this the *soft-length limited Huffman coding* problem. In this version, there is a penalty associated with each of the n characters of the alphabet whose encodings exceed a specified bound $D(\leq n)$ where the penalty increases linearly with the length of the encoding beyond D. The goal of the problem is to find a prefix-free encoding having minimum average code length and total penalty within a pre-specified bound $\mathcal{P}$. This generalizes the LLHC problem. We present an algorithm to solve this problem that runs in time $O(nD)$. We study a further generalization in which the penalty function and the objective function can both be arbitrary monotonically non-decreasing functions of the codeword length. We provide dynamic programming based exact and PTAS algorithms for this setting.

2012 ACM Subject Classification Theory of computation → Data compression

Keywords and phrases Approximation algorithms, Hierarchical memory, Prefix free codes

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2021.8

Related Version Preliminary versions of this work appeared as posters in Data compression conference 2020 [5] and Data compression conference 2021 [6].
Previous Version: `https://ieeexplore.ieee.org/document/9105878`
Previous Version: `https://ieeexplore.ieee.org/document/9418722`

Acknowledgements We are grateful to an anonymous reviewer for suggesting the use of Monge property and simplifying the proofs of Theorem 1 and Theorem 2 in a previous version of this manuscript.

1 Introduction

Data compression algorithms aim to reduce the number of bits required to represent data in order to save storage capacity, speed up file transfer, and decrease costs for storage hardware and network bandwidth. Compression techniques are primarily divided into two categories: lossless and lossy. Lossless compression enables data to be restored to its original state, without the loss of a single bit of data, when it is uncompressed (decoded). Huffman encoding is a basic and popular approach for lossless data compression based on variable length prefix-free encoding [20], where the characters of the alphabet are encoded with variable

[1] Currently on leave from Dept. of Comp. Science, Indian Institute of Technology, New Delhi, India

Character	Frequency	Huffman Encoding
A	25	0
B	9	11
C	6	101
D	4	1001
E	1	10000
F	1	10001

(a)

Lookup Table I (2 bits)	
2 Bits lookup	Code
00	A,1
01	A,1
10	Table II
11	B,2

(b)

Lookup Table II (3 bits)	
3 Bits lookup	Code
000	E,3
001	F,3
010	D,2
011	D,2
100	C,1
101	C,1
110	C,1
111	C,1

(c)

Figure 1 Consider an alphabet with frequencies and Huffman encoding as shown in Table (a). Tables (b) and (c) illustrate the 1^{st} and 2^{nd} level lookup tables of width 2 and 3 bits respectively.

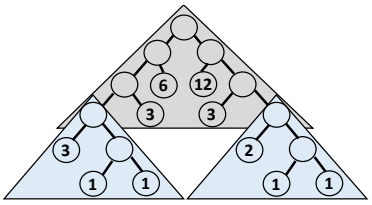

Figure 2 Illustration of blocking scheme: $<(3,1),(2,1)>$.

length codewords and no character encoding is a prefix of another. Huffman encoding is widely used in many applications including file compression (e.g. GZIP [12], PKZIP [12], BZIP2 [10], etc.) and image and video storage formats (JPEG [28], PNG [7], MP3 [8], etc.).

Traversal of a Huffman tree to decode compressed data has an inherent cost proportional to the path length that can be prohibitively slow for many real time applications. One such application is inference task in deep learning. As the sizes of deep learning models are quite large, smaller models are obtained by using Huffman coding in conjunction with other techniques to reduce the memory consumption [18]. The model is decoded in real-time when inference has to be performed. In such settings, it is acceptable to trade-off the compression ratio for improved decode time as this is a critical aspect for a good user experience. Since the data is encoded only once, it may be beneficial to spend the extra time in suitably encoding data to expedite decoding.

To avoid repeated sequential path traversals of the Huffman tree, we can exploit the indirect addressing capabilities of the RAM model by using lookup tables; code trees are employed where small tables are used to represent subtrees [26]. If w bits are used (called the width of the table), then the size of the table is 2^w. So we partition the code into prefixes of smaller lengths when the tree is not balanced, to economize space. If a prefix of the lookup bits forms a valid code word, then the table entry points to the corresponding code word and the input slides ahead by the number of bits used in the encoding of the code word. Otherwise, the table entry points to another table where a lookup is performed with the next fixed number of bits (possibly different than w) of the input; this is repeated until a valid word is decoded. This is illustrated in Figure 1.

This scheme is further complicated by the memory hierarchy that limits the storage at the faster levels of memory and has increasing latencies as we access deeper tables. The prefix tree can be viewed as multiple levels of blocks where each block corresponds to a lookup table used during decoding. Figure 2 demonstrates the concept of blocking where we assume that the blocks that require the same number of indirections have similar latencies. This problem can be formulated as follows:

Consider an alphabet C such that the size of alphabet, $|C| = n$. For each character c in C, let the attribute $freq(c)$ denote the frequency of c in the input data to be encoded. Given a prefix tree T corresponding to a prefix-free code for C, let $d_T(c)$ denote the depth of the leaf corresponding to the encoding of c in the tree. Note that $d_T(c)$ is also the length of the codeword for character c. The average code length of the encoding represented by the tree T is given by

$$len(T) = \sum_{c \in C} freq(c) \cdot d_T(c) \tag{1}$$

Define a *blocking scheme* of m *block levels* as a sequence of m block parameters, $<(w_1, q_1)$, (w_2, q_2), $\ldots$,$(w_m, q_m)>$, where w_j and q_j specify the width and the access cost of a block, respectively, at *block level* j in the tree. For a blocking scheme, the number of memory hierarchies is the number of times the access cost changes when traversing the blocks in order. For a character c having depth $d_T(c)$ in a prefix tree T, the cost of looking up (decoding) the character under the scheme BS, $\delta_T(c)$, is given by the total sum of the cost of accessing the blocks starting from the first *block level* up to the *block level* to which the character belongs, i.e., $\delta_T(c) = \sum_{i \leq W(c)} q_i$ where $W(c) = \arg\min_h \left\{ \sum_{j=1}^{h} w_j \geq d_T(c) \right\}$. The total decode time of the encoding for a prefix tree T is given by: $\delta(T) = \sum_{c \in C} freq(c) \cdot \delta_T(c)$.

Problem Definition (COPT)**:** Given a blocking scheme BS and parameter Δ, called the *permitted cost*, the goal of our problem is to determine a prefix tree, T, that minimizes the average code length, $len(T)$, subject to $\delta(T) \leq \Delta$. We call this the *code optimal prefix tree problem* and denote it by COPT(Δ). With slight abuse of notation, we shall also refer to the decode time of the associated solution as COPT(Δ).

We present an exact and a PTAS algorithm for the $COPT(\Delta)$ problem:

▶ **Theorem 1.**

(a) *There exists a dynamic programming based algorithm to solve the* $COPT(\Delta)$ *problem that runs in time* $O(n^{2+m})$ *for* m *block levels.*

(b) *For the case where the number of block levels,* m*, is a constant, there exists an algorithm that returns a prefix tree having code-length* $\leq (1+\epsilon)COPT(\Delta)$*. The running time of the algorithm is* $O\left(\frac{n^2}{\epsilon} \max\left(\frac{1}{\epsilon^2}, \log^2(n)\right)\right)$.

Another technique for optimizing the decode time that is popular in practice was proposed by Moffat and Turpin [26]. Their algorithm looks up one entry of an *offset* array (sequentially) for every bit of the compressed data read from the input. To speed up their algorithm, they use a lookup table using a fixed number of bits from the input. This is then followed by looking up an entry of the offset array for every subsequent bit of the input. The lookup table is often kept in fast memory as compared to the offset array. The overall decode time can be optimized by accommodating more words in the lookup table. This can be modeled as a special case of the COPT problem where the memory hierarchy comprises of only two levels. The first level corresponds to a cache or scratchpad having constant memory access cost. The second level corresponds to the main memory for which every access incurs a cost of q. This corresponds to a blocking scheme of $<(w_1, z),(w_2, q),(w_2, q),\ldots>$. Any entry of the prefix tree residing in the cache can be accessed with constant cost z and thereafter every entry in the main memory is accessed with cost q. Intuitively, if the codes cannot fit into the topmost block, we need a design that will minimize the number of higher level (deep) blocks. Having a hard-bound on the code word length has been previously dealt under Length Limited Huffman Code (LLHC) problem[21]; we define a variation to deal with the current problem using a notion of penalties.

LLHC is a well studied variant of Huffman coding motivated by the construction of optimal prefix-free codes under certain practical conditions [14] such as computer file searching and text retrieval systems [30]. The LLHC(C, D) problem outputs a prefix-free encoding over alphabet C, whose lengths are bounded by D such that the average code length is minimized. The encoding length bound, D, is a hard bound in the LLHC problem and is naturally

bounded by the size of the alphabet n. Consider a soft version of the LLHC problem, where there is a penalty associated with the character encodings exceeding bound D that increases linearly with the length of the encoding. Given a bound on the admissible penalty, the goal of the problem is to find a prefix-free encoding having minimum average code length and penalty within the specified admissible bound. We note that this problem also allows us to consider settings where the desired encoding length D is smaller than $\log n$; this is impossible in the LLHC setting because of the information theoretic bottleneck.

We next define this generalized version of the LLHC problem more formally. For a character having depth λ in a prefix tree T, we associate a penalty, $p(.)$ as follows:

$$p(\lambda) = \begin{cases} z & \text{if } \lambda \leq D \\ z + q \cdot (\lambda - D) & \text{if } \lambda > D \end{cases}$$

for some constants z and q. Here, z is a constant cost for encodings having length no more than D and q is the penalty for every extra encoding bit used beyond D. The reader may note that this is a simplification from the natural blocking model where the number of bits may be more than 1. However, this assumption allows us to exploit certain properties leading to very fast solutions that are likely to work well in practice. The penalty of the prefix tree is the sum of the penalties of all the characters weighted by their frequencies, i.e.,

$$P(T) = \sum_{c \in C} freq(c) \cdot p(d_T(c)). \tag{2}$$

Problem Definition (Soft-LLHC**):** Given parameters z, q & D, which define the penalty function $p(.)$ and a penalty bound $\mathcal{P}$, the goal of the *Soft length limited Huffman coding problem*, denoted Soft-LLHC$(\mathcal{P}, z, q, D)$, is to determine a prefix tree, T, that minimizes the average code length $len(T)$ subject to $P(T) \leq \mathcal{P}$.

Figure 3 illustrates the Huffman coding for an alphabet C, the corresponding LLHC and Soft-LLHC when $D = 3$. We note that LLHC is a special case of this problem wherein $z = 0$, $q = 1$ and $\mathcal{P} = 0$. This setting does not allow for any penalty, and constrains codewords to have length $\leq D$. Thus Soft-LLHC is a generalization of the LLHC problem. We present a fast algorithm for the Soft-LLHC problem:

▶ **Theorem 2.** *There exists an algorithm to solve the* Soft-LLHC$(\mathcal{P}, z, q, D)$ *problem with running time* $O(nD)$ *when the characters of* C *are given in sorted order of frequencies. For the case when* $D = o(\log n)$*, the running time of the algorithm can be bounded by* $O(n + D2^D)$.

Note that a special case of our COPT problem with two levels of memory hierarchy for $BS =< (w_1, z), (1, q), (1, q) \cdots >$ can be mapped to the Soft-LLHC problem by taking $D = w_1$ and $\mathcal{P} = \Delta$.

Lastly, we study a more generalized version of the Soft-LLHC problem that also generalizes the COPT problem. In this problem, the penalty and cost functions can be any monotonically non-decreasing function of the code length.

We next define this problem formally.

Problem Definition (Gen-LLHC**):** Given parameters $\mathcal{P}$, called the *penalty bound*, a penalty function $p(\cdot)$ and an objective function $f(\cdot)$ that are both monotonically non-decreasing functions, the goal of the *Generalized length limited Huffman coding problem*, denoted Gen-LLHC$(\mathcal{P}, p(\cdot), f(\cdot))$, is to determine a prefix tree, T, that minimizes

$$F(T) = \sum_{c \in C} freq(c) \cdot f(d_T(c))$$

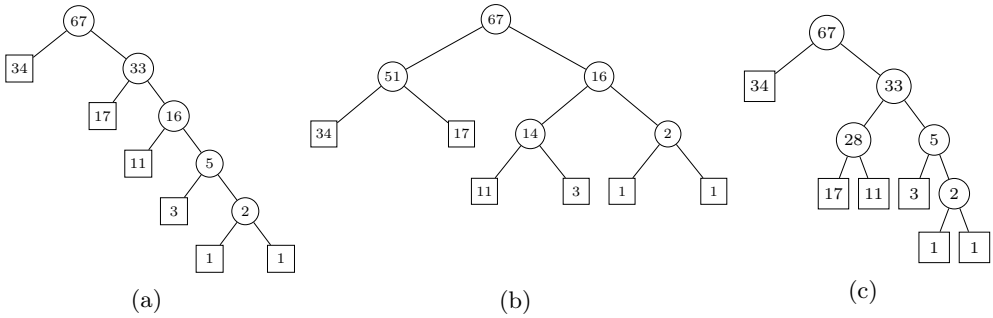

Figure 3 Consider an alphabet with 6 characters with frequencies $1, 1, 3, 11, 17, 34$ and $D = 3$. Any character with depth $w \leq 3$ bits has a penalty of z unit whereas characters with depth $w > 3$ bits have penalty $z + q \cdot (w - D)$. (a) illustrates the corresponding Huffman tree that has code length of $5 \cdot 1 + 5 \cdot 1 + 4 \cdot 3 + 3 \cdot 11 + 2 \cdot 17 + 1 \cdot 34 = 123$ and penalty of $(z + 2q) \cdot 1 + (z + 2q) \cdot 1 + (z + q) \cdot 3 + z \cdot 11 + z \cdot 17 + z \cdot 34 = 67z + 7q$. (b) Illustrates the LLHC prefix tree with higher code length of $3 \cdot 1 + 3 \cdot 1 + 3 \cdot 3 + 3 \cdot 11 + 2 \cdot 34 + 2 \cdot 17 = 150$ but a penalty of $z \cdot 1 + z \cdot 1 + z \cdot 3 + z \cdot 11 + z \cdot 17 + z \cdot 34 = 67z$. (c) Illustrates the soft-LLHC prefix tree with code length of $4 \cdot 1 + 4 \cdot 1 + 3 \cdot 3 + 3 \cdot 11 + 2 \cdot 17 + 1 \cdot 34 = 128$ and penalty of $(z + q) \cdot 1 + (z + q) \cdot 1 + z \cdot 3 + z \cdot 11 + z \cdot 17 + z \cdot 34 = 67z + 2q$.

subject to the penalty being bounded by the specified penalty bound, i.e.,

$$P(T) = \sum_{c \in C} freq(c) \cdot p(d_T(c)) \leq \mathcal{P}.$$

Note that in Gen-LLHC the penalty function is not necessarily linear, as it was in Soft-LLHC.

Also note that the COPT problem can be modeled as the Gen-LLHC problem by taking the penalty function as $p(d_T(c)) = \delta_T(c)$, $\mathcal{P}$ as Δ and the function f mapping to the code length, i.e., $f(d_T(c)) = d_T(c)$. Note that the effect of BS is handled in the way $p(d_T(c))$ is defined. We present the following results for the Gen-LLHC problem:

▶ **Theorem 3.**

(a) *There exists a dynamic programming algorithm to solve the* Gen-LLHC$(\mathcal{P}, p(\cdot), f(\cdot))$ *problem that runs in* $O(n^3 \cdot \mathcal{P})$ *time.*

(b) *There exists a dynamic programming algorithm that returns a prefix-tree having objective value at most* $(1 + \epsilon)$ *times that of the optimal solution to* Gen-LLHC$(\mathcal{P}, p(\cdot), f(\cdot))$ *and penalty no more than* $\mathcal{P}$*. The running time of this algorithm is* $O(n^4/\epsilon)$*.*

▶ Remark 4. Note that while the running time in Theorem 2 has no dependence on $\mathcal{P}$, Theorem 3(a) is not a strictly polynomial time algorithm for super polynomial values of $\mathcal{P}$.

▶ Remark 5. Theorem 3(a),(b) assume the functions $p(\cdot), f(\cdot)$ can be computed in $O(1)$ time.

Hardness. Note that it follows from Theorem 2 that the Soft-LLHC problem is in P as D can be at most n. We do not have a hardness result for the Gen-LLHC problem though we present a PTAS for the problem in Theorem 3. The COPT problem is a special case of the Gen-LLHC problem for which we give an algorithm which runs in polynomial time when the number of block levels is constant.

1.1 Related Work

The first algorithm for LLHC was due to Karp[21] and was based on an integer linear programming formulation. Gilbert[15] then gave an enumeration based algorithm for LLHC. Both these algorithms had exponential running time. Later Hu and Tan[19] gave an $O(nD2^D)$ time Dynamic Programming algorithm. Note that D is bounded by n in the worst case. In 1974, Garey[14] presented the first polynomial time algorithm, running in time $O(n^2D)$ for the case of binary encoded alphabets. Larmore[23] combined techniques of [19] and [14] to give an algorithm with running time $O(n^{3/2}D\log^{1/2} n)$ for the binary case. Larmore and Hirschberg [24] then designed a completely new algorithm with running time $O(nD)$; this algorithm was based on a reduction to the coin collector's problem, which was then solved using a technique they called the Package-Merge algorithm. There have been several subsequent works that have improved the running time further for the special case when $D = \omega(\log n)$ to $O(n\sqrt{D\log n} + n\log n)$ by Aggarwal, Schieber and Tokuyama [2] and to $n2^{O(\sqrt{\log D \log\log n})}$ by Schieber [27]. Baer [3] studied a variant of the Huffman coding problem wherein there is a continuous (strictly) monotonic increasing cost (penalty) function, called Campbell penalties [11], associated with the length of a character encoding; the goal of the problem is to minimize the "mean" length of the cost function over all the characters of the alphabet. We note that this problem seeks to minimize an objective different from the average code length, thereby addressing a different setting compared to Huffman coding, LLHC and our Soft-LLHC problems. In particular, the Soft-LLHC problem seeks to minimize the average code length constrained by a budget on the admissible penalty.

Generalized cost functions for building Huffman trees have been studied before. Fujiwara and Jacobs [13] studied the Generalized Huffman Tree (GHT) problem in which the cost of each encoded character depends on its depth in the tree by an arbitrary function. Here the goal is to determine a prefix tree, T, that minimizes $\sum_{i=1}^{|C|} f_i(d_T(c_i))$ for the GHT problem and minimizes $\max_{i=1}^{|C|} f_i(d_T(c_i))$ for the Max-GHT problem. This is a further generalization of our cost function, where a separate function is associated with each character.

On the other hand, the Soft-LLHC problem corresponds to optimizing the objective function allowing deviations from the individual code lengths for which we provide bi-criterion results that are novel to the best of our knowledge. We do note however that the LLHC problem is a special case of both the Soft-LLHC problem (as specified earlier) as well as the GHT problem (by taking the cost function to be ∞ when depth exceeds D and equal to frequency times depth otherwise).

Fujiwara and Jacobs [13] further prove that the Max-GHT problem is NP-hard when the cost functions are allowed to be arbitrary and provide a polynomial time algorithm when the cost functions are non-decreasing. We observe that the hardness result crucially depends on the the prefix tree being a complete binary tree. However, we show that for certain functions, the optimal prefix tree for Max-GHT need not necessarily be a complete binary tree (see Appendix A). As a matter of fact, we present a simple polynomial time construction for the relaxed version of Max-GHT by reducing the Max-GHT problem with arbitrary functions into Max-GHT problem with non-decreasing functions in $O(n^2)$ time. Using the polynomial time algorithm of Fujiwara and Jacobs [13] for the case when the cost functions are non-decreasing, this actually yields a polynomial time algorithm for the case of arbitrary functions as well. This result is captured in the following theorem and it's proof is presented in Appendix A.

▶ **Theorem 6.** *There is an $O(n^2\log n)$ algorithm for Max-GHT with arbitrary functions.*

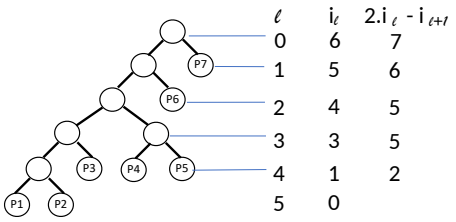

Figure 4 Illustration of the calculation of the number of characters below level ℓ $(2i_\ell - i_{\ell+1})$. This figure is taken from [16].

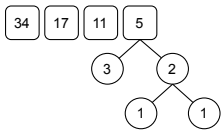

Figure 5 The **3-level forest** to the tree T, shown in Figure 3(a).

Organization of the paper. Our algorithms build on the dynamic program for Huffman codes proposed by Larmore and Przytycka[22] and extended in Golin[16]. This algorithm is discussed in Section 2. In Section 3, we first present our algorithm for the simplest of the problems, Soft-LLHC; this provides the proof for Theorem 2. In Section 4, we discuss the algorithmic approach for the generalized version of the Gen-LLHC problem; this corresponds to Theorem 3. In Section 5, we present the algorithms for the COPT problem. This is presented last as the proofs reuse results from the algorithm for Gen-LLHC. We present the proof for the PTAS corresponding to Theorem 1(b) and defer the proof of Theorem 1(a) to the Appendix. We end with concluding remarks in Section 6.

2 Preliminaries: a DP for Huffman codes

Consider a prefix tree, T. The nodes of T can be classified as either leaf nodes (i.e., nodes with no child nodes), or internal nodes (i.e., nodes with exactly 2 child nodes). Leaf nodes represent characters of the alphabet. Let $d_T(u)$ denote the depth of any node in the tree, T (with the root being at depth 0). The depth (or height) of the tree, denoted $h(T)$ is the maximum depth of any node in the tree, i.e., $h(T) = \max_{u \in T}\{d_T(u)\}$. We use the variable ℓ to refer to the level starting from the top of the tree ($\ell = 0$ for the root). Further, let i_ℓ denote the number of internal nodes at or deeper than level ℓ.

This is illustrated in Figure 4. The following proposition relates the number of characters below some level with the number of internal nodes at different levels.

▶ **Proposition 7.** *The number of characters below (deeper than) level ℓ is $2i_\ell - i_{\ell+1}$.*

The formal proof of the proposition can be found in [16]. The intuitive idea is as follows: to form each internal node we need two child nodes (can be either internal or leaf). Hence, for i_ℓ internal nodes we would require $2i_\ell$ nodes at or deeper than level $(\ell + 1)$. Since of these $2i_\ell$ nodes $i_{\ell+1}$ are internal nodes at or deeper than level $(\ell + 1)$, the number of characters or leaf nodes below level ℓ must be $2i_\ell - i_{\ell+1}$.

The following Theorem is an adaptation of a result from Golin and Zhang[16] that specifies a condition for us to be able to construct a valid prefix tree. Note that Golin and Zhang [16] did not require the condition that $\forall \ell \leq h-2, n \geq (2i_\ell - i_{\ell+1}) \geq (2i_{\ell+1} - i_{\ell+2})$. They instead proved that any sequence that is an optimal solution to the LLHC problem corresponds to a valid prefix tree (Lemma 2 and 8 in [16]). We instead show that this extra condition is necessary and sufficient, for any $\mathcal{I}$ to correspond to a valid full binary prefix tree.

▶ **Theorem 8.** *Given a decreasing sequence of integers, $\mathcal{I} = \langle i_k, i_{k+1}, \ldots, i_h = 0\rangle$, such that $\forall \ell \leq h-2$, $n \geq (2i_\ell - i_{\ell+1}) \geq (2i_{\ell+1} - i_{\ell+2})$ and $i_k \leq n-1$ we can construct a forest, rooted at level k, such that the number of internal nodes at or below level ℓ is i_ℓ.*

We defer the proof of Theorem 8 to Appendix B. Corollary 9 follows from Theorem 8 when $k = 0$ and $i_0 = n - 1$.

▶ **Corollary 9.** *Given a decreasing sequence of integers, $\mathcal{I} = \langle i_0 = n-1, i_1, \ldots, i_h = 0\rangle$, such that $\forall \ell \leq h-2 :\ n \geq (2i_\ell - i_{\ell+1}) \geq (2i_{\ell+1} - i_{\ell+2})$, we can construct a prefix tree of height h such that the number of internal nodes at or below level ℓ is i_ℓ.*

Observe that in any optimal prefix-tree, a character with higher frequency cannot appear lower than a character having lower frequency (otherwise we could swap them leading to an improved codelength). Using this fact, the following result from Golin and Zhang[16] helps us to rewrite the code length of the code represented by a prefix tree as the sum of contributions of prefix sums at each level.

▶ **Theorem 10.** *Let $S = [S_1, S_2 \cdots S_n]$ be prefix sum array of frequencies, where $S_i = \sum_{j=1}^{i} freq(j)$ and frequencies are sorted in increasing order of the depths of the characters in the tree T. Then the code length of the tree, T, can be written as a sum of h prefix sums, where each sum represents the code length contribution by each level of the tree, i.e., $len(T) = \sum_{\ell=0}^{h-1} S_{2i_\ell - i_{\ell+1}}$.*

The formal proof can be found in [16]. The intuitive idea is as follows:

by the definitions of $len(T)$ and $d_T(c)$ (Eqn 1 and depth of character c in tree T), we have

$$len(T) = \sum_{c \in C} freq(c) \cdot d_T(c) = \sum_{c \in C} \sum_{\ell=1}^{d_T(c)} freq(c)$$

By rearranging the summation over each level and using Proposition 7, we get $len(T) = \sum_{\ell=0}^{h-1} \sum_{j=1}^{2i_\ell - i_{\ell+1}} freq(j)$ and viewing the inner sum as prefix sum, we get $len(T) = \sum_{\ell=0}^{h-1} S_{2i_\ell - i_{\ell+1}}$.

The goal of Golin and Zhang[16] is to determine a prefix tree, T, for which the code length, i.e., $len(T)$ is minimum. The idea of their dynamic program is as follows. Let $H(i)$ denote the minimum code length amongst all forests having exactly i internal nodes. Then $H(n-1)$ yields the optimal code length. As mentioned in Theorem 9, it suffices to obtain a sequence of i_ℓ's to determine the prefix tree. Suppose that $i_\ell = i$ and $i_{\ell+1} = j$, then due to Larmore and Przytycka [22] we have, $H(i) = H(j) + S_{2i-j}$. This allows us to determine the optimal values of i_ℓ's as follows. We initialize H to ∞ for all entries and then use the following recurrence:

$$H(i) = \min_{\substack{j \in [max(0, 2i-n), i-1] \\ \&\ 2i-j\ \geq\ 2j-k}} H(j) + S_{2i-j}$$

where k is the recursive index used in populating $H(j)$, i.e., $H(j)$ was minimized for $H(k) + S_{2j-k}$ (this can be recorded in a separate table); the condition $(2i - j\ \geq\ 2j - k)$ ensures that the number of leaves below the root level in the structure with i internal nodes is greater than or equal to that in the structure with j internal nodes. Here, $H(0)$ and S_0 are initialized to 0.

Time complexity. As there are n entries of H and each entry requires $O(n)$ computations to compare the recurrences, the algorithm takes $O(n^2)$ time. Using the concavity of S_i, this was improved to $O(n)$ time in [25] by filling the cells using Concave Least weight Subsequence (CLWS), which can be solved using SMAWK algorithm as a subroutine in $O(n)$ time [29].

▶ Remark. We use a slightly different notion of level than [16]. While [16] considers levels starting with the bottom most level as 0 and increasing up to the root, we consider levels to start with 0 from the root and increasing down the tree. The above theorems and lemmas have been rephrased accordingly.

3 Algorithm for the Soft-LLHC (Soft-LLHC) Problem

Note that Soft-LLHC$(\mathcal{P}, z, q, D)$ can be reformulated as Soft-LLHC$(\mathcal{P}', 0, 1, D)$ by taking $\mathcal{P}' = \frac{1}{q} \cdot (\mathcal{P} - z \sum_{c \in C} freq(c))$. Here on we work with this reformulation of the problem.

Consider the structure of the prefix tree, T, in a solution to the Soft-LLHC problem. Recall that a character with higher frequency cannot appear below a character with lower frequency. We can view the tree as comprising of levels starting with level 0 at the root. We define the **d-level forest of T, denoted** $F_d(T)$, to be the forest induced on T, obtained by removing all the internal nodes having depth less than d (along with their incident edges). Note that the leaf nodes having depth less than or equal to d become singleton trees in $F_d(T)$. See Figure 5 for an illustration. Let $d_{F_d(T)}(c)$ denote depth of character c in this forest. Note that in the reformulated version of our problem, the penalty of the entire tree T is equal to the codelength of the forest rooted at level D for any tree. Hence for $d \leq D$, the penalty of the tree T can be written as

$$\sum_{c \in C : d_{F_d(T)}(c) > D-d} \big(d_{F_d(T)}(c) - (D-d)\big) \cdot freq(c)).$$

We maintain a table $\overline{H}$ of size $\mathcal{C} \times D$. Intuitively, for $0 \leq i < |\mathcal{C}|$ and $1 \leq d \leq D$, an entry $\overline{H}(i, d)$ of this table tries to capture the structure of the d-level forest, $F_d(T')$, corresponding to the best prefix tree, T', for which $F_d(T')$ comprises i internal nodes. More precisely, an entry $\overline{H}(i, d)$ of this table represents the minimum amongst the code lengths of all forests (over the alphabet $\mathcal{C}$) comprising of exactly i internal nodes and additionally satisfying the condition that the penalty condition is not violated, i.e.,

$$\sum_{c \in C : d_{F_d(T)}(c) > D-d} \big(d_{F_d(T)}(c) - (D-d)\big) \cdot freq(c)) \leq \mathcal{P}'.$$

Note that for $d < D$, the d level forest $F_d(T^*)$ in the optimal tree T^* is formed by introducing new internal nodes that combine some of the trees of the forest $F_{d+1}(T^*)$ (by merging their roots pairwise to form new internal nodes). From the previous section, the code length of a prefix-tree of height h can be represented as sum of h prefix-sums. This is applicable for the Soft-LLHC problem as well. Thus, the values of the table $\overline{H}$ can be computed as follows. Initialize all entries of $\overline{H}$ to ∞ and then use the following recurrence for $d < D$:

$$\overline{H}(i, d) = \min_{\substack{j \in [max(0, 2i-n), i-1] \\ \& \ 2i-j \geq 2j-k}} \overline{H}(j, d+1) + S_{2i-j}.$$

Here k corresponds to the recursive index used in populating $\overline{H}(j, d+1)$, i.e., $\overline{H}(j, d+1)$ was minimized for $\overline{H}(k, d+2) + S_{2j-k}$ (this can be recorded in a separate table).

Note that at level D, an entry $\overline{H}(i, D)$ corresponds to the minimum code length amongst all forests having exactly i internal nodes and penalty no more than $\mathcal{P}'$. Since the penalty of this tree corresponds exactly to its codelength (as $z = 0$ and $q = 1$ for the reformulated Soft-LLHC), this actually corresponds exactly to the entry $H(i)$, provided $H(i) \leq \mathcal{P}'$ and we can thus initialize $\overline{H}(i, D) = H(i)$. Note that if $H(i) > \mathcal{P}$, then there does not exist a

tree with penalty less than $\mathcal{P}$ and having i internal nodes with depth at least D; thus we can set $\overline{H}(i, D) = \infty$ in this case. The final solution is then obtained from the entry $\overline{H}(n-1, 0)$. The prefix tree can be constructed by alluding to Theorem 9.

Time complexity. The entries of H can be computed in time $O(n)$ as discussed previously. For computing $\overline{H}$, there are nd cells and each cell takes $O(n)$ time to fill using the recurrence above. Hence the running time is $O(n^2 D)$. This time can be improved by employing properties of Monge matrices. This is discussed next.

Improving the running time using Monge property. Monge property is a discrete extension of concavity which allows for the speeding up of several algorithms[9]. SMAWK is one such classical algorithm, using which row-minima can be found. It was used by Golin and Zhang[16] to solve the LLHC problem. We follow a similar approach. Consider the recurrence

$$\widehat{H}(i, d) = \min_{j \in [max(0, 2i-n), i-1]} \widehat{H}(j, d+1) + S_{2i-j}.$$

Note that we drop the condition $2i - j \geq 2j - k$ from the recurrence for $\overline{H}$. This is because we can argue that the optimal sequence will correspond to a valid prefix-tree. As all the solutions of $\widehat{H}$ satisfy the penalty constraint, we only minimize the code length. If the optimal sequence doesn't correspond to a valid prefix-tree, it is possible to construct a sequence using Lemma 8 in [16], with a smaller value of $\sum_{\ell=0}^{D-1} S_{2i_\ell - i_{\ell+1}}$, leading to a contradiction.

Now consider a new implicit matrix $M^{(d)}$ such that for all $0 \leq i, j \leq n$: $M^{(d)}_{i,j} = \widehat{H}(j, d+1) + S_{2i-j}$ when $0 \leq 2i - j \leq n$ and ∞ when $2i - j > n$ or $2i - j < 0$. We show that $M^{(d)}$ is a Monge matrix. This follows from the following Lemma.

▶ **Lemma 11.** $M^{(d)}_{i,j} + M^{(d)}_{i+1,j+1} \leq M^{(d)}_{i+1,j} + M^{(d)}_{i,j+1}$.
(Note that SMAWK allows for this condition to be satisfied when both sides evaluate to ∞).

Proof. We consider the following three (exhaustive) cases:
(I) *When* $2i - j < 1$*:* $M^{(d)}_{i,j+1}$ is ∞ and thus the result holds by definition (as $2i - (j+1) < 0$).
(II) *When* $2i - j > n - 2$*:* $M^{(d)}_{i+1,j}$ is ∞ and thus the result holds by definition (as $2(i+1) - j > n$).
(III) *When* $1 \leq 2i - j \leq n - 2$*:* entries $M^{(d)}_{i,j}, M^{(d)}_{i+1,j+1}, M^{(d)}_{i+1,j}, M^{(d)}_{i,j+1}$ are defined and we have:

$$\begin{aligned}
(M^{(d)}_{i,j} + M^{(d)}_{i+1,j+1}) \;&-\; (M^{(d)}_{i+1,j} + M^{(d)}_{i,j+1}) \\
&\leq\; (\widehat{H}(j, d+1) + S_{2i-j} + \widehat{H}(j+1, d+1) + S_{2i-j+1}) \\
&\qquad -\; (\widehat{H}(j, d+1) + S_{2i-j+2} + \widehat{H}(j+1, d+1) + S_{2i-j-1}) \\
&=\; S_{2i-j} + S_{2i-j+1} - S_{2i-j+2} - S_{2i-j-1} \\
&=\; freq(c_{2i-j}) - freq(c_{2i-j+2}) \quad \leq \quad 0
\end{aligned}$$

where c_i is the ith least frequent character and thus the result holds. ◀

Observe that, by definition, $\widehat{H}(i, d) = \min_{0 \leq j \leq i} M^{(d)}_{i,j} = \min_{0 \leq j \leq n} M^{(d)}_{i,j}$. The Monge property on $M^{(d)}$ implies that the SMAWK algorithm[1] can solve for the row minima of each $M^{(d)}$ matrix in $O(n)$ time. Thus our algorithm repeats the process of finding row minima of each $M^{(d)}$ matrix for $d = D - 1$ to 0 to obtain the minima corresponding to $\widehat{H}(., d)$. Thus solving for D such matrices takes $O(nD)$ time(See Algorithm 1). The number of internal nodes with depth at most D is bounded by 2^{D+1} and hence the computation of $H(i)$ takes $O(n)$ time. Thus, the run time can be bounded by $O(n + D2^D)$ when $D = o(\log n)$.

Algorithm 1 (for **Theorem 2**).

Input: Weighted Alphabet $C = \{c_1, c_2, \ldots, c_n\}$; Penalty bound $\mathcal{P}$;
Output: Minimum code length prefix tree having penalty less than $\mathcal{P}$

```
S ← Prefix sum array of sorted frequencies
H(i) ← From CLWS for all i ∈ [0, n − 1]
for i ← 0 to n − 1 do
    if H(i) ≤ P then
        Ĥ(i, D) = H(i)
    if H(i) > P then
        Ĥ(i, D) = ∞
for d ← D − 1 to 0 do
    SMAWK(M^(d)) uses Ĥ(i, d + 1) and computes Ĥ(i, d)
C* ← Ĥ(n − 1, 0)
T* ← Obtain the prefix tree by following the parent pointers of C*
return T*;
```

4 Algorithms for the Generalized LLHC (Gen-LLHC) Problem

We build on the ideas of Golin and Zhang[16] (see Section 2 for details). By extending their construction, we show that for Gen-LLHC$(\mathcal{P}, p(\cdot), f(\cdot))$, the objective value $F(T)$, for any tree T, can also be written as a sum of h terms. The i^{th} term representing the product of the sum of the frequencies of all the leaf nodes with depth less than or equal to i and the difference in the objective values at depth i and $i-1$, that is $f(i) - f(i-1)$ $(f(0) = 0)$. However our goal is to minimize the objective value, $F(T)$, of the tree. We handle the penalty bound involved by maintaining an extra parameter in our proposed dynamic program. We store structures with the minimum objective value having penalty less than the new parameter (corresponding to the admissible values of penalty bound) introduced. Using this formulation we obtain an exact algorithm referred to in Theorem 3(a).

The running time of the exact algorithm is $O(n^3 \cdot \mathcal{P})$ which may be super polynomial in n for large values of $\mathcal{P}$. Subsequently, we are able to bound the number of feasible penalty values using standard rounding techniques and get an approximate algorithm which runs in $O(n^4/\epsilon)$ and has code length no more than $(1+\epsilon)$ time the optimal value. We prove a slightly generalized variant of the problem, denoted Gen-LLHC$^*(\mathcal{P}, p(\cdot), f(\cdot), h)$ that takes an additional parameter h representing a height bound and determines a prefix tree T of height at most h that minimizes $F(T)$ subject to the penalty bound as before. We show that

▶ **Theorem 12.** *There exists a dynamic programming algorithm that returns a prefix-tree having height at most h and objective value at most $(1+\epsilon)$ times that of the optimal solution to* Gen-LLHC$^*(\mathcal{P}, p(\cdot), f(\cdot), h)$ *and penalty $\leq \mathcal{P}$ with running time of $O(n^2h^2/\epsilon)$.*

Theorem 3(b) follows by taking the parameter h as n as that is the maximum height possible.

The details of the algorithms and proofs are presented in following subsections.

Note that unlike the Gen-LLHC problem, the Soft-LLHC has strictly polynomial running time as we use $\mathcal{P}$ only to filter and remove the infeasible solutions.

As mentioned before, we do not have a hardness result for the Gen-LLHC problem. We note that proving hardness is challenging for several problems related to Huffman coding. For instance, hardness results are not known for Huffman coding with unequal letter costs[17]

that admit a PTAS. As another instance, we have shown that the hardness result for a closely related problem, MAX-GHT, due to Fujiwara and Jacobs[13] in prior literature is not correct (See Theorem 6 and the associated discussion in Section 1.1).

4.1 Exact DP for GEN-LLHC: Proof of Theorem 3(a)

We start with a simple proposition.

▶ **Proposition 13.** *A character having higher frequency will appear at the same or lower level (that is closer to the root) than a character having lower frequency.*

The proposition is easy to verify - if this was not true, one could simply swap the two characters thereby improving the objective value as well as the penalty.

Note that for the GEN-LLHC problem also, the code length can be represented as sum of h prefix-sums. We will now show that for GEN-LLHC$(\mathcal{P}, p(\cdot), f(\cdot))$, the objective value $F(T)$, for any tree T, can also be written as a sum of h terms, where each term corresponds to the contribution by the corresponding level, to the objective value, of the tree. Recall that $f(\cdot)$ was a monotonically non-decreasing function. This result is captured in Lemma 14. We define two new function $\hat{f}(.), \hat{p}(.)$:

$$\hat{f}(i) = \begin{cases} f(1) & \text{if } i = 1 \\ f(i) - f(i-1) & \text{if } i > 1 \end{cases}$$

$$\hat{p}(i) = \begin{cases} p(1) & \text{if } i = 1 \\ p(i) - p(i-1) & \text{if } i > 1 \end{cases}$$

Now if h represents the total height of a tree, T, then we have the following lemma.

▶ **Lemma 14.**

$$F(T) = \sum_{\ell=0}^{h-1} \hat{f}(\ell+1) \cdot (S_{2i_\ell - i_{\ell+1}})$$

The proof of Lemma 14 is deferred to Appendix C. Using Lemma 14, it remains to find a sequence of i_l's as before except that now the goal is to minimize the objective value, $F(T)$, of the tree. The prefix tree can be constructed by alluding to Theorem 9. We describe a recurrence to obtain such a sequence. Let $D(i, \ell, P)$ denote the minimum objective value amongst all forests rooted at level ℓ, having i internal nodes with penalty at most P (here, the objective value of the forest is the sum of the objective values of the trees in the forest).

A dynamic program using the above recurrence can be designed as follows. Let h be some upper bound on the height of the optimal prefix tree.

Base Case. For all forests with no internal nodes, we initialize the objective value to 0, i.e.,

$$\forall\ \ell \in [0, h] \text{ and } P \in [0, \mathcal{P}]: \quad D(0, \ell, P) = 0$$

Inductive Case. To compute $D(i, \ell, P)$, we iterate over the number of internal nodes at depths greater than ℓ. If j internal nodes are at depths strictly greater than ℓ, then there are $(2i - j)$ characters at depths strictly greater than ℓ and $\hat{f}(\ell+1) \cdot S_{2i-j}$ is the contribution, to the objective value $F(T)$, of all the characters having level(depth) $> \ell$, due to the access at level $(\ell + 1)$. Furthermore, $D(j, \ell+1, P')$ denotes the objective value contributed by all accesses made at levels(depths) greater than $\ell + 1$. This yields the following recurrence:

$$D(i, \ell, P) = \min_{\substack{j \in [max(0, 2i-n), i-1] \\ \& \ 2i-j \ \geq \ 2j-k}} \left\{ D(j, \ell+1, P') + \hat{f}(\ell+1) \cdot S_{2i-j} \right\} \tag{3}$$

where $P' = P - \hat{p}(\ell+1) \cdot S_{2i-j}$ and k is the recursive index using which $D(j, \ell+1, P')$ was populated. We only need to recurse if $P' > 0$. The tree with the optimal objective value can be obtained by maintaining the parent pointers of each update and backtracking (similar to as shown in pseudo-code of Theorem 3(b) in Appendix D).

Time complexity. There are $O(n)$ characters, h levels and $O(\mathcal{P})$ values for penalty; hence there are $O(nh\mathcal{P})$ cells in the table. As each cell can be filled in $O(n)$ time, the time complexity is $O(n^2 h\mathcal{P})$. As height, h is at most n, we get the time complexity to be $O(n^3\mathcal{P})$. As $\mathcal{P}$ may not be polynomial in n, this is a pseudo-polynomial time algorithm.

As the above algorithm is symmetric in terms of penalty and objective value, we can find the tree having the minimum penalty and objective value at most $\mathcal{C}$ in $O(n^3\mathcal{C})$ time using the recurrence

$$D(i, \ell, C) = \min_{\substack{j \in [max(0,2i-n),i-1] \\ \& \ 2i-j \geq 2j-k}} \{D(j, \ell+1, C - S_{2i-j}) + \hat{p}(\ell+1) \cdot S_{2i-j}\} \tag{4}$$

Let the penalty of the solution to the above DP be $\mathcal{P}_{dual}$, we will use it to give a $PTAS$ algorithm for $Gen - LLHC$ in the next section.

4.2 PTAS for Gen-LLHC: Proof of Theorem 3(b) and Theorem 12

We first prove Theorem 12. Theorem 3(b) follows as h takes value at most n.

The algorithm presented in the previous section has linear running time dependency on the parameter $\mathcal{P}$. In this section, we propose a polynomial time approximation algorithm that runs in time $O(n^4/\epsilon)$ and returns a prefix tree having penalty at most $\mathcal{P}$ and objective value with in $(1+\epsilon)$ times the optimal value. We first give an algorithm which returns a tree with penalty at most the value of the minimum penalty possible for tree with objective value at most $\mathcal{C}$, and objective value at most $(1+\epsilon)\mathcal{C}$.

For this we restrict the parameter C to only take on values that are multiples of $\lambda = \lfloor(\epsilon \cdot \mathcal{C})/2h\rfloor$ ranging from $0 \cdot \lambda$ upto $((2h/\epsilon) + h) \cdot \lambda$ where h is some upper bound on the height of the optimal prefix tree. We denote the dynamic program table maintained by this algorithm with $\overline{D}$. Let $\overline{D}(i, \ell, C)$ denote the minimum penalty amongst all forests rooted at level ℓ, having i internal nodes with objective value at most C (here, the penalty of the forest is the sum of the penalties of the trees in the forest). Note, here each DP cell stores a structure having minimum penalty as compared to the exact algorithm of Gen-LLHC, where each DP cell stores a structure having minimum objective value.

We define a rounding function $\mathbf{r}$ as follows:

$$\mathbf{r}(x) = \left\lceil \frac{x}{\lambda} \right\rceil \cdot \lambda.$$

We change the recurrence from the previous section as follows: The base case becomes: for all forests with no internal nodes, we initialize the objective value to 0, i.e., $\forall\ \ell \in [0, h]$ and C a multiple of λ and $C \in [0, \mathbf{r}(\mathcal{C}) + h\lambda]$: $D(0, \ell, C) = 0$.. The inductive step is modified to:

$$\overline{D}(i, \ell, C) = \min_{\substack{j \in [max(0,2i-n),i-1] \\ \& \ 2i-j \geq 2j-k}} \left\{\overline{D}(j, \ell+1, C') + \hat{p}_{\ell+1} \cdot S_{2i-j}\right\} \tag{5}$$

where $C' = C - \mathbf{r}(\hat{f}_{\ell+1} \cdot S_{2i-j})$ and k is the recursive index using which $\overline{D}(j, \ell+1, C')$ was populated. Note that we only update entries of $\overline{D}$ for which the C parameter is itself a multiple of λ. It is easy to see that the C parameter will take on only $O(h/\epsilon)$ values. The table can be compressed accordingly and maintained only for these entries, however we omit these implementation details in the interest of better readability.

The following Lemma shows we can get a prefix-tree with penalty $\leq \mathcal{P}_{dual}$ by sacrificing an additive λ factor for every level in the objective value.

▶ **Lemma 15.** *For any valid values of i, ℓ and P:* $D(i,\ell,C) \geq \overline{D}(i,\ell,\mathbf{r}(C)+(h-\ell)\cdot\lambda)$.

From the previous section, we know that the optimal solution is captured by $D(h-1,0,\mathcal{C})$. Hence the above Lemma (proof in Appendix D) implies that the optimal solution is also captured by $\overline{D}(h-1,0,\mathbf{r}(\mathcal{C})+h\cdot\lambda)$. We now look at all the entries $\overline{D}(h-1,0,\mathbf{r}(C)+h\cdot\lambda) \leq P_{dual}$ and pick the entry with the minimum value of $\mathbf{r}(C)+h\cdot\lambda$.

Hence using $\overline{D}$, given a objective function threshold $\mathcal{C}$, we can find a prefix tree with penalty $\leq P_{dual} \leq \mathcal{P}$ and objective value $\leq (1+\epsilon)\cdot\mathcal{C}$. Now, if substitute $\mathcal{C} = C^*$, where C^* is the objective value of the solution to the Gen-LLHC problem, we will have the solution to the $PTAS$ of Gen-LLHC problem. Now, instead of calculating C^* directly we use binary search in the range $[0, \mathcal{F}\cdot f(n)]$, where $\mathcal{F}$ is the cumulative frequency of all the characters in the prefix tree and $f(n)$ is the value of objective function at level n. As the depth is at most n for all characters and $f(.)$ is an increasing function, the objective value is at most $\mathcal{F}\cdot f(n)$. Thus, we have a $PTAS$ algorithm for Gen-LLHC.

Time complexity. There are at most $O(h)$ levels and $O(n)$ characters. C can have at most $O(h/\epsilon)$ possible values. Hence, there are $O(nh^2/\epsilon)$ cells in the table. Each cell can be filled in at most $O(n)$ time. So the time complexity is $O(n^2h^2/\epsilon)$. Since there are a total of h recursive calls, the error in objective function value is bounded by $h\lambda \leq \epsilon\cdot\mathcal{C}$. Thus, we can find a prefix tree having objective value less than $(1+\epsilon)\cdot C^*$ and penalty at most $\mathcal{P}$ in $O(n^2h^2/\epsilon)$ time. This proves Theorem 12. Taking the upper bound for the height, h, as n, we get the time complexity to be $O(n^4/\epsilon)$, which proves Theorem 3(b).

5 Algorithms for the Code Optimal Prefix Tree (COPT) problem

For a fixed number of block levels, m, the possible number of values corresponding to the decode time for the forests in the dynamic program is n^{m-1} . We use this to give an $O(n^{m+2})$ algorithm for Theorem 1(a) in Appendix F. We now present the proof of Theorem 1(b).

5.1 Proof of Theorem 1(b)

Consider the blocking scheme in the definition of the $COPT$ problem. As mentioned in the introduction, the number of block levels, m, is typically a small constant in practice. We now present a more efficient dynamic program based pseudo-approximation algorithm for the case when the number of block levels is constant.

We first prove some results (c.f. Propositions 16, 17 and Lemma 18) required in the formulation of our new dynamic program. The following proposition shows that given a set of characters, we can construct a (nearly complete) prefix tree of depth $\lceil \log n \rceil$.

▶ **Proposition 16.** *There exists a prefix tree for a set of characters, C, having depth $\lceil \log |C| \rceil$.*

Proof. It is easy to verify that we can place $2^{\lceil \log |C| \rceil} - |C|$ characters at depth $\lceil \log |C| \rceil - 1$ in the subtree and the remaining characters at depth $\lceil \log |C| \rceil$ to form a valid prefix tree (additional nodes are added to serve as internal nodes). ◀

Consider a set of characters, C. The following proposition shows that given an arbitrary tree, T, with characters of C appearing as leaf nodes in T, we can always construct a valid prefix tree over C that has height no more than that of T and in which each character appears at a depth no more than its depth in T.

▶ **Proposition 17.** *Let C denote a set of characters. Given a tree, T, in which the characters of C appear as leaf nodes, there exists a valid prefix tree, T', over C that has no greater height than T and in which $d_{T'}(c) \leq d_T(c)$ $\forall c \in C$.*

Proof. We start with the tree T and iteratively modify it until we obtain a valid prefix tree.

We find a node(say u) that violates any of these conditions and modify the tree as follows:

- Is a leaf node but does not correspond to a character: We simply delete u.
- Has only one child(say node v): We remove u and directly attach v to the parent of u.

It is easy to see that when no more violating nodes are left, we get a valid prefix tree. It is also straightforward to observe that we never increase the depth of any node in this process. ◀

The following Lemma shows that there cannot be too many levels in the optimal prefix tree between two consecutive characters of the alphabet when sorted in order of frequencies.

▶ **Lemma 18.** *In a complete binary tree, if c_i is a character at level ℓ and c_{i+1} is at level ℓ' then $\ell' - \ell < \lceil \log(n) \rceil$.*

Proof. We prove this by contradiction. Let us assume that $\ell' - \ell \geq \lceil \log(n) \rceil$. Since there is a leaf c_{i+1} at depth greater than ℓ, there must be at least one internal node at the level ℓ. By our assumption there are no leaves in the tree rooted at this internal node, till the next $\lceil \log(n) \rceil$ levels. Hence there are at least n internal nodes above level ℓ'. But the tree we started with has exactly $n-1$ internal nodes as it has n leaves. Contradiction. ◀

The following lemma shows that there exists a tree having bounded height that has almost the same code length and decode time as the optimal prefix tree of $COPT(\mathcal{P})$.

▶ **Lemma 19.** *Given $\delta > 0$, there exists a prefix tree, T', for which the code length is at most $(1+\delta)$ times the code length of $COPT(\mathcal{P})$ and the height of T' is no more than $2m(\lceil 1/\delta \rceil + \lceil \log n \rceil)$, where m is the number of block levels.*

Proof. Let h^* denote the height (total number of tree levels) of the optimal prefix tree (solution to $COPT(\mathcal{P})$). If $h^* \leq 2m(\lceil 1/\delta \rceil + \lceil \log n \rceil)$, then the claim is trivially satisfied. We therefore focus on the case when $h^* > 2m(\lceil 1/\delta \rceil + \lceil \log n \rceil)$. As there are at most m block levels, at least one of these has more than $2(\lceil 1/\delta \rceil + \lceil \log n \rceil)$ tree levels.

Let us focus on one such block level, and let the starting tree level for the block level be ℓ'. There must be at least one node c_i between the tree levels $\ell' + \lceil 1/\delta \rceil + \lceil \log n \rceil)$ and $\ell' + \lceil 1/\delta \rceil + 2\lceil \log n \rceil)$ due to Lemma 18.

▶ **Proposition 20.** *In the optimal prefix tree, the kth highest frequency is at a level at most $k + \lceil \log(n) \rceil$.*

The proof of Proposition 20 is deferred to Appendix G. From the proposition, there are at least $\lceil 1/\delta \rceil$ nodes with frequency higher than c_i.

Let T^* be the prefix tree corresponding to the optimal solution $COPT(\mathcal{P})$ and $\ell = \ell' + \lceil 1/\delta \rceil + \lceil \log n \rceil$. We modify T^* to construct another prefix tree, T' as follows:

- all characters up to $\ell + \lceil \log n \rceil$ retain the same level as in T^*, except for c_i
- c_i is replaced with a new internal node, say u, and made a child of u (c_i is at level $\ell + 1$).
- We call a character of T^* *deep* if it has depth more than $2m(\lceil 1/\delta \rceil + \lceil \log(n) \rceil)$. Let γ be the number of deep characters in T^*. Using Proposition 16, there exists a subtree comprising of all the deep characters of T^*, having depth at most $\lceil \log \gamma \rceil \leq \lceil \log n \rceil$. We attach this subtree as the second child of u. The level of any of the characters in this subtree is no more than $\ell + \lceil \log n \rceil + 1$.
- We finally invoke Proposition 17, to get a valid prefix tree.

We now show that the codelength and decode time of T' are no more than $(1+\delta)$ times the corresponding parameters of T^*. Note that both the code length and decode time of the deep characters of T^* only reduces as their depth reduces in T'. Therefore the code length and decode time can only increase due to the character c_i moving one level (tree level) down.

We first analyze the increase in code length due to c_i moving one (tree) level down. Recall that the block level to which c_i belonged was divided into $2\lceil 1/\delta \rceil$ partitions and c_i belongs to the $\lceil 1/\delta \rceil^{th}$ partition. Moreover each partition contains a character. Also, there are at least $\lceil 1/\delta \rceil$ nodes with frequency higher than c_i and hence have tree level same or above that of c_i. Thus $len(T^*) \geq \lceil 1/\delta \rceil \cdot f_i$. The increase in code length incurred by moving c_i down one tree level is f_i. Thus $f_i \leq \delta \cdot (\lceil 1/\delta \rceil \cdot f_i) \leq \delta \cdot len(T^*)$. Therefore $len(T') \leq (1+\delta) \cdot len(T^*)$.

As c_i lies between the tree levels $\ell' + (\lceil 1/\delta \rceil + \lceil \log n \rceil)$ and $\ell' + (\lceil 1/\delta \rceil + 2\lceil \log n \rceil)$, the next tree level to c_i must also be in the same block level. Therefore $\Delta(T') \leq \Delta(T^*) \leq \mathcal{P}$. ◀

Given the above Lemma, the algorithm is quite straightforward - we simply invoke Theorem 12 by bounding the h parameter by $2m(\lceil \log(n) \rceil + \lceil 1/\delta \rceil)$.

Time Complexity. The analysis is same as that of the algorithm for Theorem 12; we know that the time taken is $O(h^2 n^2/\epsilon)$.Taking the bound on the height h to be $2m(\lceil \log(n) \rceil + \lceil 1/\delta \rceil)$ and setting δ to be ϵ, the running time becomes $O\left(\frac{n^2 \cdot m^2}{\epsilon}\left(\log^2(n) + \frac{1}{\epsilon^2}\right)\right)$. For constant m, this yields a complexity of $O\left(\frac{n^2}{\epsilon} max\left(\frac{1}{\epsilon^2}, \log^2(n)\right)\right)$.

6 Conclusion and open problems

Motivated by many practical challenges in implementing compression, we introduce and study a novel variation of finding optimal prefix trees where one is allowed to deviate from the optimal code length within a specified bound. This allows us to capture more generalized decoding costs for which we develop a bi-criterion framework and present efficient algorithms. An important application of this framework is to a natural class of memory access cost functions that use blocking and to the best of our knowledge, this is the first work that lays the theoretical foundations and present a family of algorithms with provable guarantees. An open problem is to proving NP-hardness for the Gen-LLHC problem that could be quite challenging as exemplified by Theorem 6. Another interesting future direction is to study the empirical performance of our algorithms with real world data sets on practical systems with hierarchical memory; we anticipate promising results, similar to those obtained for a closely related variant in the hierarchical memory setting where the goal is to minimize the decode time and the average code length is bound by a threshold parameter[4].

References

1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

2 Alok Aggarwal, Baruch Schieber, and Takeshi Tokuyama. Finding a minimum-weightk-link path in graphs with the concave monge property and applications. *Discrete & Computational Geometry*, 12(3):263–280, 1994.

3 M.B. Baer. Source coding for quasiarithmetic penalties. *IEEE Transactions on Information Theory*, 52(10):4380–4393, 2006. doi:10.1109/TIT.2006.881728.

4 Shashwat Banchhor, Rishikesh R. Gajjala, Yogish Sabharwal, and Sandeep Sen. Decode efficient prefix codes. *CoRR*, abs/2010.05005v2, 2020. arXiv:2010.05005v2.

5 Shashwat Banchhor, Rishikesh R. Gajjala, Yogish Sabharwal, and Sandeep Sen. Decode-efficient prefix codes for hierarchical memory models. In *Data Compression Conference, DCC 2020*, page 360. IEEE, 2020. doi:10.1109/DCC47342.2020.00077.

6 Shashwat Banchhor, Rishikesh R. Gajjala, Yogish Sabharwal, and Sandeep Sen. Efficient algorithms for decode efficient prefix codes. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *31st Data Compression Conference, DCC 2021, Snowbird, UT, USA, March 23-26, 2021*, page 338. IEEE, 2021. doi:10.1109/DCC50243.2021.00080.

7 Thomas Boutell. PNG (portable network graphics) specification ver 1.0. *RFC*, 2083:1–102, 1997. doi:10.17487/RFC2083.

8 Vladimir Britanak. A survey of efficient MDCT implementations in MP3 audio coding standard: Retrospective and state-of-the-art. *Signal Process.*, 91(4):624–672, 2011. doi:10.1016/j.sigpro.2010.09.009.

9 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996. doi:10.1016/0166-218X(95)00103-X.

10 M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Digital Systems Research Centre, 1994.

11 LL Campbell. Definition of entropy by means of a coding problem. *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, 6(2):113–118, 1966.

12 Peter Deutsch. DEFLATE compressed data format specification ver 1.3. *RFC*, 1951:1–17, 1996. doi:10.17487/RFC1951.

13 Hiroshi Fujiwara and Tobias Jacobs. On the huffman and alphabetic tree problem with general cost functions. *Algorithmica*, 69(3):582–604, 2014. doi:10.1007/s00453-013-9755-6.

14 M. R. Garey. Optimal binary search trees with restricted maximal depth. *SIAM J. Comput.*, 3(2):101–110, 1974. doi:10.1137/0203008.

15 Edgar N. Gilbert. Codes based on inaccurate source probabilities. *IEEE Trans. Inf. Theory*, 17(3):304–314, 1971. doi:10.1109/TIT.1971.1054638.

16 M. Golin and Y. Zhang. A dynamic programming approach to length-limited huffman coding: Space reduction with the monge property. *IEEE Trans. on Information Theory*, 56(8):3918–3929, 2010. doi:10.1109/TIT.2010.2050947.

17 Mordecai J. Golin, Claire Kenyon, and Neal E. Young. Huffman coding with unequal letter costs. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 785–791. ACM, 2002. doi:10.1145/509907.510020.

18 Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR 2016*, 2015. arXiv:1510.00149.

19 TC Hu and KC Tan. Path length of binary search trees. *SIAM Journal on Applied Mathematics*, 22(2):225–234, 1972.

20 D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. doi:10.1109/JRPROC.1952.273898.

21 Richard M. Karp. Minimum-redundancy coding for the discrete noiseless channel. *IRE Trans. Inf. Theory*, 7(1):27–38, 1961. doi:10.1109/TIT.1961.1057615.

22 Lawrence Larmore and Teresa Przytycka. Constructing huffman trees in parallel. *SIAM Journal on Computing*, 24, July 1998. doi:10.1137/S0097539792233245.

23 Lawrence L. Larmore. Height restricted optimal binary trees. *SIAM J. Comput.*, 16(6):1115–1123, 1987. doi:10.1137/0216070.

24 Lawrence L. Larmore and Daniel S. Hirschberg. A fast algorithm for optimal length-limited huffman codes. *J. ACM*, 37(3):464–473, 1990. doi:10.1145/79147.79150.

25 Lawrence L. Larmore and Teresa M. Przytycka. A parallel algorithm for optimum height-limited alphabetic binary trees. *J. Parallel Distributed Comput.*, 35(1):49–56, 1996. doi:10.1006/jpdc.1996.0067.

26 A. Moffat and A. Turpin. On the implementation of minimum redundancy prefix codes. *IEEE Transactions on Communications*, 45(10):1200–1207, 1997.

27 Baruch Schieber. Computing a minimum weightk-link path in graphs with the concave monge property. *J. Algorithms*, 29(2):204–222, 1998. doi:10.1006/jagm.1998.0955.

28 G. K. Wallace. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.

29 Robert Wilber. The concave least-weight subsequence problem revisited. *J. Algorithms*, 9(3):418–425, September 1988. doi:10.1016/0196-6774(88)90032-6.

30 Justin Zobel and Alistair Moffat. Adding compression to a full-text retrieval system. *Softw. Pract. Exp.*, 25(8):891–903, 1995. doi:10.1002/spe.4380250804.

A Algorithm for Max-GHT: Proof of Theorem 6

A.1 Introduction

The problems GHT(Generalized Huffman Tree) and Max-GHT(Max Generalized Huffman tree) were formulated by Fujiwara and Jacobs[13]. We first state their problem definitions.

▶ **Definition 21** (GHT). *Given n arbitrary functions $f_1, f_2 \cdots f_n$ corresponding to n leaves, the objective of GHT is to determine a binary tree T with these n leaves, such that $\sum_{i=1}^{i=n} f_i(d_i)$ is minimized, where the ith leaf is at depth d_i in T.*

▶ **Definition 22** (Max-GHT). *Given n arbitrary functions $f_1, f_2 \cdots f_n$ corresponding to n leaves, the objective of Max-GHT is to determine a binary tree T with these n leaves, such that $\max_{i=1}^{i=n} f_i(d_i)$ is minimized, where the ith leaf is at depth d_i in T.*

Fujiwara et al. proved that Max-GHT and GHT are NP-hard for general functions $f_1, f_2 \cdots f_n$. However, they also proved that if each f_i is non-decreasing, then Max-GHT can be solved in $O(n^2 \log n)$ time. The complexity of GHT was unresolved, if f_i is non-decreasing.

However, there is an implicit assumption in their hardness proof. They assume that there exists a solution which is a full binary tree(all internal nodes have exactly two children) for both GHT and Max-GHT. While this has to be true when the functions are non-decreasing, it need not be true when the functions are arbitrary. Consider the following simple counter example - when there are two leaves and for $i = 1, 2$ we have function values $f_i(1) = 1$ and $f_i(2) = 0$. The optimal solution(with zero cost for both GHT and Max-GHT) will have both leaves at level 2 and hence such tree cannot be full binary. This re-opens the problems they posed and we present a simple $O(n^2)$ algorithm to convert Max-GHT and GHT with general functions to problems where Max-GHT and GHT have non-decreasing functions. As a direct consequence of this, we have an $O(n^2 \log n)$ algorithm to solve Max-GHT with general functions. Due to this reduction, we conclude that if GHT with non-decreasing functions can be solved in polynomial time, then GHT with general functions can also be solved in polynomial time. We also note that there is a solution with full binary tree for both GHT and Max-GHT with non-decreasing functions.

A.2 Reduction

▶ **Lemma 23.** *The GHT and Max-GHT problem with n arbitrary functions $f_1, f_2 \cdots f_n$, can be reduced to a problem with n non-decreasing functions $g_1, g_2 \cdots g_n$ in $O(n^2)$ time, where $g_i(j) = \min_{l=j}^{n} f_i(l)$*

Proof. We update $g_i's$ in a bottom to top manner with l^{th} entry as $\min(f_i(l), g_i(l+1))$ for $l < n$ and $g_i(n) = f_i(n)$. Hence the total time taken is $O(n)$ per function and $O(n^2)$ in total. It's easy to see that these functions evaluate to $g_i(j) = \min_{l=j}^{n} f_i(l)$ using induction.

For correctness, the key property we use is that there is an optimal tree which is a solution to GHT/Max-GHT, such that for any pair of depths (d_1, d_2), if $d_1 < d_2$ and $f_i(d_1) \geq f_i(d_2)$, then the ith leaf can not be at d_1 for any i. This is due to a simple exchange argument as we switch a node from d_1 to d_2, the Kraft sum decreases (hence the tree is feasible) and cost will not increase.(Note that if the Kraft sum for a given set of depths is less than 1, we can always construct a binary tree with those function values) Therefore the ith leaf can not be at level l if $g_i(l) \neq f_i(l)$. Hence, the structure of the optimal solution remains unchanged by changing the values for such l. ◀

We note that as the tree need not be full binary, the maximum height need not be n like in [13]. The above algorithm's correctness remains valid even when maximum height exceeds n and the run time would be $O(m)$, where m is the input size(previously n^2).

B Proof of Theorem 8

We prove this constructively by induction. For the sequence $\mathcal{I}' = \langle i_{h-1}, i_h = 0 \rangle$, we can construct a forest with i_{h-1} trees, each containing one internal node and two leaves. Since this forest has no internal nodes at or below level h, we have $i_h = 0$. Also, since the only internal nodes are the roots of the trees at level $h-1$, we have i_{h-1} internal nodes at or below level $h-1$. Further, as $2i_{h-1} \leq n$ we have sufficient characters to construct this forest).
Now, let us assume there is a valid forest corresponding to the sequence $\mathcal{I}' = \langle i_{k+1}, \cdots i_h = 0 \rangle$. Note that this forest has $i_{k+1} - i_{k+2} > 0$ trees. We now add another $(2i_k - i_{k+1}) - (2i_{k+1} - i_{k+2})$ leaves (characters) at level $k+1$ and construct a forest with $i_k - i_{k-1}$ trees, having a total of i_k internal nodes. Note that $(2i_k - i_{k+1}) - (2i_{k+1} - i_{k+2}) \geq 0$ and $(2i_k - i_{k+1}) \leq n$, hence, we have sufficient characters to create such a forest. This proves the theorem.

C Proof of Lemma 14 pertaining to Exact DP for GEN-LLHC

Proof.

$$F(T) = \sum_{\ell=0}^{h-1} \hat{f}(\ell+1) \cdot (S_{2i_\ell - i_{\ell+1}})$$

By definition of $F(T)$ we have

$$F(T) \quad = \quad \sum_{c \in C} (freq(c) \cdot f(d_T(c)))$$

By using the definition of $\hat{f}(i)$ we get

$$F(T) = \sum_{c \in C} \left(freq(c) \cdot \left(\sum_{i \leq d_T(c)} \hat{f}(i) \right) \right)$$

By rearranging the summation over each level and using proposition 7 we get

$$F(T) = \sum_{\ell=0}^{h-1} \left(\hat{f}(\ell+1) \cdot \sum_{j=1}^{2i_\ell - i_{\ell+1}} freq(j) \right)$$

and using Theorem 10 we get

$$F(T) = \sum_{\ell=0}^{h-1} \hat{f}(\ell+1) \cdot S_{2i_\ell - i_{\ell+1}}$$

This completes the proof of the Lemma. ◀

D Pseudo-code for PTAS for GEN-LLHC: Theorem 3(b)

We present the Pseudo-code for PTAS for GEN-LLHC in Algorithm 2.

Algorithm 2 (for **Theorem 3(b)**).

Input: Weighted Alphabet $C = \{c_1, c_2, \ldots, c_n\}$; Penalty bound $\mathcal{P}$; penalty function p(.); objective function $f(.)$; function $\hat{f}(.)$ defined over $f(.)$; Approximation constant ϵ

Output: Prefix tree having penalty $\leq \mathcal{P}$ and objective value less than $\leq (1+\epsilon) \cdot C^*$

$C_{PTAS} \leftarrow \infty$
for $val \leftarrow 0$ ***to*** $\log_2(\mathcal{F} \cdot f(n))$ **do**
  $\mathcal{C} \leftarrow 2^{val}$
  $\lambda = \lfloor (\epsilon \cdot \mathcal{C})/2h \rfloor$
  for $\ell \leftarrow 0$ ***to*** n **do**
    for $b \leftarrow 0$ ***to*** $((2h/\epsilon) + h)$ **do**
      $C = b \cdot \lambda$
      $\overline{D}(0, \ell, C) := 0$
  for $i \leftarrow 1$ ***to*** $(n-1)$ **do**
    for $\ell \leftarrow (h-1)$ ***downto*** 0 **do**
      for $b \leftarrow 0$ ***to*** $((2h/\epsilon) + h)$ **do**
        $C = b \cdot \lambda$
        $bestPenalty := \infty$
        for $j \leftarrow \max(0, 2i-n)$ ***to*** $i-1$ **do**
          $Penalty := \infty$
          $C' = C - \mathbf{r}(p(\ell+1) \cdot S_{2i-j})$
          $k :=$ recursive index where $\overline{D}(j, \ell+1, C')$ was minimized
          if $2i - j < 2j - k$ **then**
            continue;
          if $0 \leq C'$ **then**
            $Penalty := \overline{D}(j, \ell+1, C') + \hat{f}_{(l+1)} \cdot S_{2i-j}$
          if $Penalty < bestPenalty$ *and* $Penalty < \mathcal{P}$ **then**
            $bestPenalty := Penalty$
        $\overline{D}(i, \ell, C) := bestPenalty$
  $C_{PTAS} := \min_C(\overline{D}(n-1, 0, C)$ *corresponds to a valid prefix tree*$)$
  if $C_{PTAS} \leq \mathcal{C}$ **then**
    break
return C_{PTAS};

E Proof of Lemma 15 pertaining to PTAS for Gen-LLHC

Proof. For any valid values of i, ℓ and P:

$$D(i,\ell,C) \;\geq\; \overline{D}(i,\ell,\mathbf{r}(C)+(h-\ell)\cdot\lambda)$$

In our proof we will be using the following fact:

▶ **Proposition 24.** *Let C' and C'' be multiples of λ. Then, $\overline{D}(z,\ell+1,C') \geq \overline{D}(z,\ell+1,C'')$ whenever $C' \leq C''$.*

This proposition holds because the best solution having objective value at most C' is also a candidate solution having objective value at most C'' (other parameters remaining same).

We now prove the lemma by induction on the value of ℓ decreasing from h to 0.
For $\ell = h$: From our initialization, the entries of D and $\overline{D}$ are all initialized to 0 for $\ell = h$ and hence the claim trivially holds.
For $\ell < h$: Consider $D(i,\ell,C)$. From recurrence (4), there must be some choice of j for which $D(i,\ell,C)$ is minimized. Let z be that choice of j, i.e.,

$$D(i,\ell,C) = D(z,\ell+1,C-\hat{f}_{\ell+1}\cdot S_{2i-z}) + \hat{p}_{\ell+1}\cdot S_{2i-z}$$

Now we obtain the following relations:

$$\begin{aligned}
&D(i,\ell,C)\\
&= D(z,\ell+1,C-\hat{f}_{\ell+1}\cdot S_{2i-z}) + \hat{p}_{\ell+1}\cdot S_{2i-z}\\
&\geq \overline{D}(z,\ell+1,\mathbf{r}(C-\hat{f}_{\ell+1}\cdot S_{2i-z}) + (h-(\ell+1))\lambda) + \hat{p}_{\ell+1}\cdot S_{2i-z}\\
&\geq \overline{D}(z,\ell+1,\mathbf{r}(C) - (\mathbf{r}(\hat{f}_{\ell+1}\cdot S_{2i-z}) - \lambda) + (h-(\ell+1))\lambda) + \hat{p}_{\ell+1}\cdot S_{2i-z}\\
&= \overline{D}(z,\ell+1,\mathbf{r}(C) - \mathbf{r}(\hat{f}_{\ell+1}\cdot S_{2i-z}) + (h-\ell)\lambda) + \hat{p}_{\ell+1}\cdot S_{2i-z}\\
&\geq \overline{D}(i,\ell,\mathbf{r}(C)+(h-\ell)\lambda)
\end{aligned}$$

where the first inequality follows by induction, the second inequality follows from Proposition 24 and the last inequality follows from the fact that $\overline{D}(z,\ell+1,\mathbf{r}(C) - \mathbf{r}(\hat{f}_{\ell+1}\cdot S_{2i-z}) + (h-\ell)\lambda) + \hat{p}_{\ell+1}\cdot S_{2i-z}$ is also a candidate for consideration in recurrence (5) for $\overline{D}(i,\ell,\mathbf{r}(C)+(h-\ell)\lambda)$.

This completes the proof of the Lemma. ◀

F Exact DP for COPT: Proof of Theorem 1(a)

There exists a dynamic program algorithm to solve the COPT($\mathcal{P}$) problem that runs in time $O(n^{2+m})$ for m block levels.

The dynamic programming algorithm is similar to that for the exact algorithm with the main difference being that instead of iterating over lengths we iterate over the decode times of the tree.

Let $\tilde{D}(i,\ell,T)$ denote the minimum codelength amongst all forests rooted at level ℓ, having i internal nodes with decode time at most T. Also, define $\mathcal{T} = \sum_{c=1}^{n}\cdot\sum_{i=1}^{m} q_i$ (here, the decode time of the forest is the sum of the decode times of the trees in the forest)

For a fixed number of block levels, m, the following lemma holds:

▶ **Lemma 25.** *The number of possible values of decode time for the forests rooted at some level is n^{m-1}.*

Proof. Let there be x_i characters in the ith block level $\forall i \in [1, m]$ and x_0 be the number of characters which are not present in the forest corresponding to $\tilde{D}(i, \ell, T)$. These characters corresponding to x_0 will not have any decode time contribution for T. We have $\sum_{i=0}^{m} x_i = n$. For $l > w_1$, the width of first block, we know that x_1 is zero. When $l \leq w_1$, we know that x_0 is zero. That is not both of x_0, x_1 can be non-zero. Hence, there are $O(n^{m-1})$ possible sequences of x_i's satisfying this. For each sequence of x_i's, we can uniquely determine the decode time value. Hence there are $O(n^{m-1})$ possible decode time values. ◀

A dynamic program using the above recurrence can be designed as follows:

Base Case. For all forests with no merges, i.e. no internal nodes, we initialize the decode time to 0, i.e.,

$$\forall\ \ell \in [0, n]\ and\ T \in [0, \mathcal{T}]: \quad \tilde{D}(0, \ell, T) = 0$$

Inductive Case. To compute $\tilde{D}(i, \ell, T)$, we iterate over the number of internal nodes that are at depth strictly greater than ℓ. If j internal nodes are at depth strictly greater than ℓ, then there are $(2i - j)$ characters at depth strictly greater than ℓ, then $q_{\ell+1} \cdot P_{2i-j}$ is the decode-time contribution of all the characters having level $> \ell$, due to the access at level $(\ell + 1)$. Furthermore, $\tilde{D}(j, \ell + 1, T')$ denotes the decode time contributed by all accesses made at depths greater than $\ell + 1$. This yields the following recurrence:

$$\tilde{D}(i, \ell, T) = \min_{\substack{j \in [max(0, 2i-n), i-1] \\ \&\ 2i-j\ \geq\ 2j-k}} \left\{ \tilde{D}(j, \ell + 1, T') + P_{2 \cdot i-j} \right\} \tag{6}$$

where $T' = T - \hat{q}_{lvl+1} \cdot P_{2 \cdot i-j}$ and k is the recursive index using which $\tilde{D}(j, \ell + 1, T')$ was populated. We only need to recursively check if $T' > 0$. The tree with the optimal decode time can be obtained by maintaining the parent pointers of each update and then backtracking.

Note that we only update entries of $\tilde{D}$ for which the T parameter corresponds to a sequence of $\langle\ x_0, x_1, x_2, \ldots, x_m\ \rangle$, from lemma 25. The decode time for a sequence $\langle x_0, x_1, x_2, \ldots, x_m \rangle$ is $Dec(\langle x_i \rangle) = \sum_{i=1}^{m} x_i \cdot \hat{q}_i$

From Lemma 25, we know that the T parameter will take on only $O(n^{m-1})$ values. The table can accordingly be compressed and maintained only for these entries, however we omit these implementation details in the interest of better exposition.

After the DP is filled, we check all the entries of the form $\tilde{D}(n - 1, 0, t)$ which have code length parameter $t \leq \mathcal{P}$ and find the the optimal code length corresponding to it.

Time complexity. We note that i and ℓ can take n possible values each and T takes n^{m-1} possible values. So, there are n^{m+1} cells in the DP. Each cell can be filled in at most $O(n)$ time. So, the time complexity of the DP is $O(n^{m+2})$. Checking the DP table to find the optimal decode time will take $O(n^{m+1})$ time. Hence, we can solve the $COPT$ problem in $O(n^{m+2})$ time when the number of block levels is a constant m.

G Proof of Proposition 20 pertaining to Theorem 1(b)

Proof. We prove this using induction. The base case holds from Lemma 18. Consider the two nodes at level one.

We first consider the case where not all k highest frequencies are in the same sub-tree rooted at one of the nodes. By induction assumption, in the sub-tree in which kth highest frequency is present, the kth highest frequency is at level at most $k - 1 + \lceil \log(n) \rceil$. Therefore given holds.

We now consider the case where all k highest frequencies are in the same sub-tree rooted at one of the nodes. Let the highest frequency in the other sub tree be $k + r'$th frequency for some $r' > 0$. If $k + r'$th frequency is at level at most $k + \lceil \log(n) \rceil$, since higher frequencies are at a lower level, kth highest frequency is at level at most $k + \lceil \log(n) \rceil$. If not, the subtree has more than $2^{k+\lceil \log(n) \rceil - 1} > n - 1$ nodes. Contradiction. ◀