ELSEVIER

Information Processing Letters 66 (1998) 35–40

Information Processing Letters

Optimum extensions of prefix codes ☆

Ion I. Măndoiu [1]

College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280, USA

Received 20 February 1997; revised 5 October 1997
Communicated by L. Hemaspaandra

Abstract

An algorithm is given for finding the minimum weight extension of a prefix code. The algorithm runs in $O(n^3)$, where n is the number of codewords to be added, and works for arbitrary alphabets. For binary alphabets the running time is reduced to $O(n^2)$, by using the fact that a certain cost matrix satisfies the quadrangle inequality. The quadrangle inequality is shown not to hold for alphabets of size larger than two. Similar algorithms are presented for finding alphabetic and length-limited code extensions. © 1998 Elsevier Science B.V.

Keywords: Algorithms; Prefix codes; Dynamic programming; Quadrangle inequality

1. Introduction

Huffman's classical algorithm [6] constructs a prefix code with minimum weighted length over a given alphabet. A related problem, introduced in [2], is that of optimally *extending* a prefix code: given a prefix code, $\mathcal{C}$, and n positive weights, $w_1, \ldots, w_n$, find codewords $c_1, \ldots, c_n$ such that $\mathcal{C} \cup \{c_1, \ldots, c_n\}$ remains a prefix code, and, subject to this condition, $\sum_{i=1}^{n} w_i |c_i|$ is minimum. (Here, and throughout the paper, we use $|\omega|$ to denote the length of ω.) It is well known that the extension problem has solution whenever $\mathcal{C}$ satisfies Kraft's strict inequality, $\sum_{\omega \in \mathcal{C}} m^{-|\omega|} < 1$, where m denotes the size of the alphabet.

The extension problem has the same objective function as Huffman's problem, but the two differ in the range of choices available for codewords c_i. We say that ω *extends* $\mathcal{C}$ if $\mathcal{C} \cup \{\omega\}$ is a prefix-free set. An *extension root* of $\mathcal{C}$ is a length-minimal word that extends $\mathcal{C}$; clearly, each codeword c_i must have an extension root as prefix. Calude and Tomescu [2] observe that Huffman's algorithm can be used to find an optimum extension when all extension roots of $\mathcal{C}$ have the same length, in particular when $\mathcal{C}$ has only one extension root. In this paper we present an algorithm that finds an optimum extension for an arbitrary extendable prefix code.

Starting with Gilbert and Moore [4], dynamic programming has been successfully applied to several prefix coding problems (see, for example, [3,7,8,12]). Particularly efficient algorithms are obtained with a speed-up technique devised by Knuth [8]; the speed-up is based on the fact that a certain cost matrix satisfies the quadrangle inequality (an upper-triangular matrix W satisfies quadrangle inequality if $w(i,j) + w(i',j') \leqslant w(i,j') + w(i',j)$ for all $i < i' \leqslant j < j'$). We use dynamic programming to solve the code-

☆ Work supported in part by NSF grant CCR-9627308.

[1] On leave from Bucharest University, Faculty of Mathematics, Str. Academiei 14, R-70109 Bucharest, Romania. Email: mandoiu @cc.gatech.edu.

PII S0020-0190(98)00026-X

extension problem in $O(n^3)$ time, and, in the binary case, we use the quadrangle inequality to speed up the computation by a factor of n. However, the speed-up is achieved not by using Knuth's technique, but the matrix searching algorithm of Aggarwal et al. [1]. We show that this speed-up idea cannot be extended to alphabets of size larger than two, in particular the quadrangle inequality is shown not to hold for nonbinary alphabets. A similar discrepancy between the binary and nonbinary case has been reported in [5] with respect to the construction of minimum multiway search trees.

In analogy with restrictions studied for prefix coding, we consider two other versions of the code-extension problem. In the *alphabetical* extension problem the new codewords have to be lexicographically ordered, while for the *length-limited* version a fixed upper bound is imposed on the length of the new codewords. We use dynamic programming to solve these restricted versions, and apply the same speed-up technique to the binary case.

2. Definitions

Although we introduced the extension problem as a coding problem, by a straightforward correspondence it can be formulated in terms of positional m-ary forests. Following a common practice, from now on we will use this graph theoretical terminology.

Let $m \geqslant 2$ be an integer. The notion of *m-ary tree* is the natural extension of positional binary trees defined in [9]: an m-ary tree is a set of nodes that is either empty, or consists of a root node and an ordered list of m disjoint m-ary trees, called the first, second, ..., respectively the mth subtree of the root. The "parent" and "child" relations between nodes are defined in the expected way, and the ordering that exists between subtrees of a node induces an ordering between its children. An *m-ary forest* is an ordered collection m-ary trees. A leaf is defined as a node with no children. Notice that the ordering of the trees in the forest together with the ordering of the nodes within each tree defines a total left-to-right ordering on the leaves of a forest.

An *m-ary tree with root of depth r* is a pair (T, r), where T is an m-ary tree and r is a nonnegative integer. The depth of a node v of (T, r) is defined by

$$d_v = \begin{cases} r, & \text{if } v \text{ is the root of } T, \\ 1 + d_{\text{parent}(v)}, & \text{otherwise.} \end{cases}$$

An *m-ary forest with root depths $r_1, \ldots, r_k$* is an ordered collection of m-ary trees with roots of the specified depths.

In the following we will only consider trees and forests with weights assigned to the leaves. An *alphabetic* m-ary tree (forest) with leaf weights $w_1, \ldots, w_n$ has the n weights assigned to leaves in left-to-right order. For a non-alphabetic m-ary tree (forest), the 1-to-1 assignment of weights to leaves can be arbitrary. An m-ary tree (forest) with weights $w_1, \ldots, w_n$ has an associated *cost* of $\sum_{j=1}^{n} w_j d_j$, where d_j denotes the depth of the leaf to which w_j is assigned.

The *minimum forest problem* is defined as follows: given k nonnegative integers, $r_1, \ldots, r_k$, and n positive weights, $w_1, \ldots, w_n$, find a minimum cost forest with root depths $r_1, \ldots, r_k$ and leaves labeled by $w_1, \ldots, w_n$. Notice that, if $k > n$, by disposing of the largest $k - n$ root depths we do not increase the cost of a minimum forest. Accordingly, we will assume that $k \leqslant n$ for all instances of the minimum forest problem.

An instance of the code extension problem can be easily translated into an instance of the minimum forest problem, with root depths being determined by the extension roots of the code to be extended. Suppose, for example, that we must extend $\mathcal{C} = \{aa, bbb\}$ over the alphabet $\{a, b\}$. The extension roots of $\mathcal{C}$ are ab, ba, and bba, so the optimum extension of $\mathcal{C}$ with n codewords of weight $w_1, \ldots, w_n$ corresponds to a minimum binary forest with $r_1 = |ab|$, $r_2 = |ba|$, $r_3 = |bba|$, and the same set of weights.

3. Minimum alphabetic forests

In the *alphabetic* version of the minimum forest problem we want to find a minimum cost forest that assigns the weights $w_1, \ldots, w_n$ to its leaves in left-to-right order. This ordering constraint imposes a simple structure on the optimal solutions, making the problem easily solvable by dynamic programming.

For every $1 \leqslant i \leqslant k$ and $0 \leqslant j \leqslant n$, let $C_i(j)$ denote the cost of a minimum alphabetic m-ary forest with root depths $r_1, \ldots, r_i$ and leaf weights $w_1, \ldots, w_j$. Also, for every $1 \leqslant j_1 \leqslant j_2 \leqslant n$, let $T[j_1, j_2]$ be the cost of a minimum alphabetic m-ary tree with root of depth 0 and leaf weights $w_{j_1}, \ldots, w_{j_2}$. Clearly,

$$C_1(j) = T[1, j] + r_1 \sum_{t=1}^{j} w_t. \tag{1}$$

Moreover, for every $2 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant n$,

$$C_i(j) = \min\Bigl(C_{i-1}(j), \min_{0 \leqslant b < j} M_i[b, j]\Bigr), \tag{2}$$

where $M_i[b, j]$ represents the minimum cost of an m-ary forest with root depths $r_1, \ldots, r_i$ and leaf weights $w_1, \ldots, w_j$ that assigns weights $w_{b+1}, \ldots, w_j$ to the leaves of the ith subtree. Because the sub-forests of a minimum alphabetic forest are themselves minimum,

$$M_i[b, j] = C_{i-1}(b) + T[b+1, j] + r_i \sum_{t=b+1}^{j} w_t.$$

From (1) and (2) we get a straightforward algorithm for computing all values $C_i(j)$. First, we compute $T[j_1, j_2]$ for every $1 \leqslant j_1 \leqslant j_2 \leqslant n$ using Itai's algorithm [7]. This takes $O(n^3 \log m)$ time; Itai claims that this can be improved to $O(n^2 \log m)$, but, as remarked in [5], his claim is correct only for $m = 2$. After computing all entries of T and the cumulative weights $\sum_{t=b+1}^{j} w_t$, it takes constant time to evaluate each $M_i[b, j]$. Since the minimum in (2) can be determined in $O(n)$ time, it follows that we can compute all entries $C_i(j)$, as well as an optimum forest, in $O(n^3 \log m + n^2 k)$ time.

We will next show that the running time can be reduced by a factor of n when constructing binary forests, i.e., when $m = 2$. First, we compute all entries of T in $O(n^2)$ time using Knuth's algorithm [8]. Clearly, the cumulative weights $\sum_{t=b+1}^{j} w_t$ can also be computed within the same time bound. Let us extend each M_i, $i \in \{2, \ldots, n\}$, to a full $n \times n$ matrix (with rows indexed from 0 to $n-1$ and columns indexed from 1 to n) by setting $M_i[b, j] = \infty$ for $b \geqslant j$. Computing $C_i(1), \ldots, C_i(n)$ for a fixed $i \in \{2, \ldots, k\}$ amounts by (2) to computing the minimum element in each column of M_i. Aggarwal et al. [1] describe an algorithm that, for an $n \times n$ matrix M satisfying a certain property called total monotony, computes all columnwise minimas in $O(n)$ time. As we shall prove, matrices M_i are totally monotone when $m = 2$. Since after pre-processing an entry of M_i is computed in constant time, by applying the algorithm of Aggarwal et al. to each M_i we obtain all $C_i(j)$'s in $O(nk)$ time. So, a minimum alphabetic binary forest can be constructed in $O(n^2 + nk)$ time.

The fact that matrices M_i are totally monotone when $m = 2$ will follow from the fact that they satisfy the quadrangle inequality. We first prove that matrix T satisfies the quadrangle inequality. The next result has been first noticed by Garey [3, Corrollary 1] for non-alphabetic binary trees, and extended by Wessner [12] to alphabetic binary trees with weights on all nodes (not only on leaves) and limited depth. For the particular case when the weights of internal nodes are zero and the depth limit is n, Wessner's result reads as follows:

Lemma 1 (Cf. [12, Lemma 1]). *Assume that $m = 2$. If $\Delta(i, j) = T(i, j) - T(i, j-1)$, then, for every $j \geqslant i + 2$, $\Delta(i, j) \geqslant \Delta(i+1, j)$.*

Corollary 2. *Assume that $m = 2$. Matrix T satisfies the quadrangle inequality, i.e., for every $1 \leqslant i_0 < i_1 \leqslant j_0 < j_1 \leqslant n$,*

$$T[i_0, j_0] + T[i_1, j_1] \leqslant T[i_0, j_1] + T[i_1, j_0].$$

Proof. Let $1 \leqslant i_0 < i_1 \leqslant j_0 < j_1 \leqslant n$. Lemma 1 implies that $\Delta(i_0, j) \geqslant \Delta(i_1, j)$ for every $j \geqslant i_0 + 2$. Hence,

$$\begin{aligned} &T[i_0, j_1] - T[i_0, j_0] \\ &\quad = \sum_{j=j_0+1}^{j_1} \Delta(i_0, j) \geqslant \sum_{j=j_0+1}^{j_1} \Delta(i_1, j) \\ &\quad = T[i_1, j_1] - T[i_1, j_0]. \qquad \square \end{aligned}$$

Corollary 2 implies that each matrix M_i, $i \in \{2, \ldots, k\}$, satisfies the quadrangle inequality. Indeed, let $0 \leqslant b_0 < b_1 < j_0 < j_1 \leqslant n$ (recall that rows of M_i are indexed from 0 to $n-1$). Since $M_i[b, j] = C_{i-1}(b) + T[b+1, j] + r_i \sum_{t=b+1}^{j} w_t$, we get that

$$\begin{aligned} &M_i[b_0, j_0] + M_i[b_1, j_1] - M_i[b_0, j_1] - M_i[b_1, j_0] \\ &\quad = T[b_0+1, j_0] + T[b_1+1, j_1] \\ &\qquad - T[b_0+1, j_1] - T[b_1+1, j_0] \\ &\quad \leqslant 0. \end{aligned} \tag{3}$$

For an $n \times n$ matrix M, let $b_M(j)$ be the smallest index b such that $M[b, j]$ is the minimum value in the jth column of M. Matrix M is called *monotone* if $b_M(j_0) \leqslant b_M(j_1)$ whenever $j_0 \leqslant j_1$, and *totally monotone* if every 2×2 submatrix is monotone.

Lemma 3. *Assume that $m = 2$. For every $i \in \{2, \ldots, k\}$, M_i is totally monotone.*

Proof. Suppose that some M_i is not totally monotone, and let $0 \leqslant b_0 < b_1 \leqslant n-1$ and $1 \leqslant j_0 < j_1 \leqslant n$ be the row, respectively the column indices determining a nonmonotone 2×2 submatrix of M_i. Monotonicity is trivially satisfied by a 2×2 submatrix of M_i that contains infinite values, so it must be the case that $b_1 < j_0$. Since $M_i[b_1, j_0] < M_i[b_0, j_0]$ and $M_i[b_0, j_1] \leqslant M_i[b_1, j_1]$, we get that $M_i[b_1, j_0] + M_i[b_0, j_1] < M_i[b_0, j_0] + M_i[b_1, j_1]$, in contradiction with (3). □

Unfortunately, this speed-up idea does not extend to nonbinary forests: as shown by the following example, matrices M_i are not necessarily totally monotone when $m \geqslant 3$.

Example 4. Consider $m = 3$, $k = 2$, $d_1 = d_2 = 0$, $n = 6$, and $w_1 = \cdots = w_6 = 1$. As shown in Fig. 1, the 2×2 submatrix of M_2 determined by rows 1 and 2 and columns 5 and 6 is not monotone. Hence, M_2 is not totally monotone.

Example 4 also shows that matrix T need not satisfy the quadrangle inequality when $m \geqslant 3$. Indeed, for

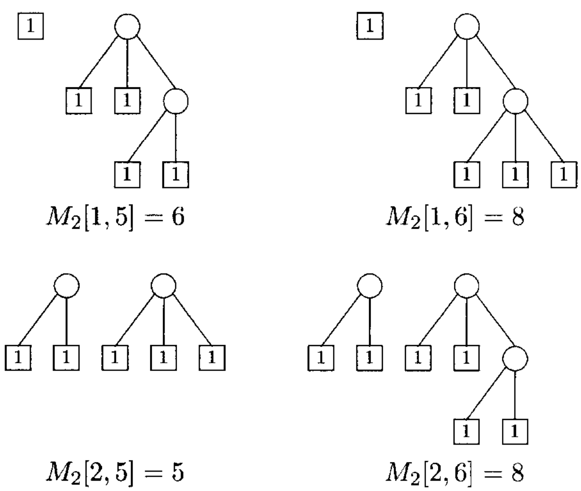

Fig. 1. Matrix M_2 is not totally monotone.

the given weights we have $T[2,5] = 6$, $T[3,6] = 6$, $T[3,5] = 3$, and $T[2,6] = 8$, so $T[2,5] + T[3,6] > T[3,5] + T[2,6]$.

4. Non-alphabetic forests

As observed by Schwartz and Kallick [11], when $w_1 \geqslant w_2 \geqslant \cdots \geqslant w_n$ there exists a minimum binary tree in which the weights are assigned to leaves in left-to-right order, i.e., a minimum binary tree that is alphabetic. For minimum m-ary forests, a similar result follows from:

Lemma 5. *Let F be an m-ary forest with root depths $r_1 \leqslant r_2 \leqslant \cdots \leqslant r_k$ and leaves labeled by $w_1 \geqslant w_2 \geqslant \cdots \geqslant w_n$. Suppose that the weights are assigned to the leaves of F such that the node labeled by w_i has depth smaller than or equal to the depth of the node labeled by w_j whenever $w_i > w_j$. Then, there exists an* alphabetic *m-ary forest F^* having the same root depths as F and leaves labeled by $w_1, \ldots, w_n$ such that* $\mathrm{cost}(F^*) = \mathrm{cost}(F)$.

Proof. We will use a simple re-arrangement argument. Let u_i denote the leaf of F labeled by w_i, and let d_i be the depth of u_i in F. If the weights $w_1, \ldots, w_n$ are not already assigned in left-to-right order to the leaves of F, the set $X = \{(i, j) \mid i < j,\ u_i$ is to the right of $u_j\}$ must be nonempty. Let $i_0 = \min\{i \mid \exists j$ s.t. $(i, j) \in X\}$ and $j_0 = \max\{j \mid (i_0, j) \in X\}$.

If $w_{i_0} = w_{j_0}$, let F' be the forest obtained from F by swapping w_{i_0} with w_{j_0}. If $w_{i_0} > w_{j_0}$, F' is defined as follows (see Fig. 2). First, note that the hypothesis implies that $d_{i_0} \leqslant d_{j_0}$. Moreover, because u_{i_0} is to the right of u_{j_0}, the depth of the root of the tree containing u_{j_0} is no larger that the depth of the root of the tree containing u_{i_0}, and so, no larger than d_{i_0}. Thus, on the path from u_{j_0} to the root of the tree containing it there is a node, u, of depth d_{i_0}. Let F' be the forest obtained from F by swapping w_{i_0} with the subtree rooted at u.

It is easy to see that in both cases $\mathrm{cost}(F') = \mathrm{cost}(F)$. Moreover, the transformation of F into F' either leads to an increase in the value of i_0, or leaves i_0 unchanged and decreases j_0. So, by repeating the above transformation at most n^2 times we obtain an alphabetic forest with the same cost as F. □

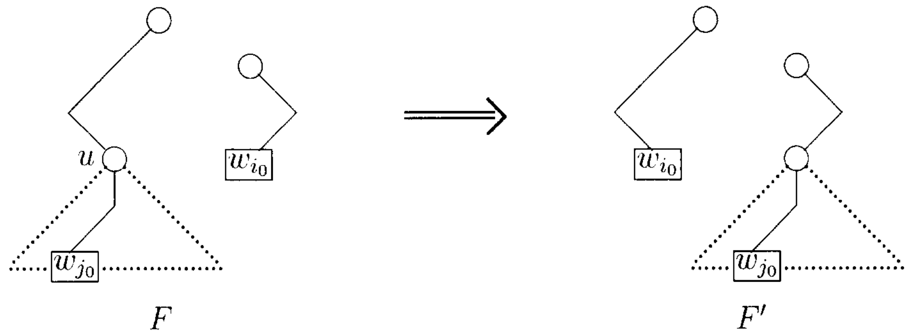

Fig. 2. The construction of F' when $w_{i_0} > w_{j_0}$.

Since any minimum m-ary forest satisfies the property that $d_i \leqslant d_j$ whenever $w_i > w_j$, we obtain:

Corollary 6. *If $r_1 \leqslant r_2 \leqslant \cdots \leqslant r_k$ and $w_1 \geqslant w_2 \geqslant \cdots \geqslant w_n$, then there exists a minimum m-ary forest in which the weights are assigned to leaves in left-to-right order.*

Corollary 6 shows that we can obtain a minimum m-ary forest by sorting the weights and root depths and then applying the algorithm for alphabetic forests. A minimum binary forest can be found in $O(n^2)$ time via this reduction, since in this case we assume that $k \leqslant n$. For arbitrary m the algorithm can be implemented to run in $O(n^3)$ time. For this we need a small modification in the pre-processing step: instead of computing T by applying Itai's algorithm, we compute each $T[j_1, j_2]$ with a call to Huffman's algorithm. Since Huffman's algorithm can be implemented to run in $O(n)$ time when the weights are already sorted (see, for example, [10]), the pre-processing step is now completed in $O(n^3)$ time.

5. Depth-limited forests

In practical applications of prefix coding it is desirable to impose an upper-bound on the length of the codewords. In the *minimum depth-limited forest problem* we optimize $\sum_{j=1}^{n} w_j d_j$ as before, but require that $d_j \leqslant D$ for every $1 \leqslant j \leqslant n$, where D is a given integer. Clearly, we may assume that each r_i is at most D. Since an m-ary tree whose root has depth $r \leqslant D$ can have at most m^{D-r} leaves of depth at most D, it follows that a solution to the problem exists if and only if

$$\sum_{i=1}^{k} m^{D-r_i} \geqslant n.$$

Again, the minimum depth-limited forest problem has an alphabetic and a non-alphabetic version. Since, by Lemma 5, the non-alphabetic version reduces to solving an alphabetic problem after sorting the weights and root depths, we discuss only the alphabetic version here.

Let $T^{(d)}[j_1, j_2]$ be the cost of a minimum alphabetic m-ary tree with root of depth 0, weights $w_{j_1}, \ldots, w_{j_2}$, and leaves of depth at most d. If we denote by $C_i^{(D)}(j)$ the cost of a minimum alphabetic m-ary forest with root depths $r_1, \ldots, r_i$, weights $w_1, \ldots, w_j$, and leaves of depth at most D, it follows that

$$C_1^{(D)}(j) = T^{(D-r_1)}[1, j] + r_1 \sum_{t=1}^{j} w_t. \tag{4}$$

Moreover, for every $2 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant n$,

$$C_i^{(D)}(j) = \min\Big(C_{i-1}^{(D)}(j), \min_{0 \leqslant b < j} N_i[b, j]\Big), \tag{5}$$

where

$$N_i[b, j] = C_{i-1}^{(D)}(b) + T^{(D-r_i)}[b+1, j] + r_i \sum_{t=b+1}^{j} w_t.$$

Let $L = \max_i (D - r_i)$. For arbitrary m, the values $T^{(d)}[j_1, j_2]$, $0 \leqslant d \leqslant L$, $1 \leqslant j_1 \leqslant j_2 \leqslant n$, can be evaluated in $O(n^3 L \log m)$ time with the algorithm suggested by Itai [7]. Thus, using (4) and (5), we obtain a minimum depth-limited alphabetic m-ary forest in $O(n^3 L \log m + n^2 k)$ time.

The running time can be reduced by a factor of n when $m = 2$. First, all values $T^{(d)}[j_1, j_2]$ can be evaluated in $O(n^2 L)$ time [7,12]. Moreover, Lemma 1 holds for matrix $T^{(d)}$ if $m = 2$ (cf. [12, Lemma 1]). Exactly as in Section 3, this implies that $T^{(d)}$ satisfies the quadrangle inequality and matrices N_i, $2 \leqslant i \leqslant k$, are totally monotone. So, by running the matrix searching algorithm of [1] on each N_i we obtain a minimum depth-limited alphabetic binary forest in $O(n^2 L + nk)$ time.

Acknowledgements

I wish to thank Ioan Tomescu and Vijay Vazirani for encouraging my work on the problem and Lawrence Larmore for stimulating discussions. Thanks also go to the anonymous referees for helpful comments and for suggesting reference [5].

References

[1] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, R. Wilber, Geometric applications of a matrix-searching algorithm, Algorithmica 2 (1987) 195–208.

[2] C. Calude, I. Tomescu, Optimum extendable prefix codes, Tech. Rept. No. 114, Dept. of Computer Science, The University of Auckland, New Zealand, 1995.

[3] M.R. Garey, Optimal binary search trees with restricted maximal depth, SIAM J. Comput. 3 (1974) 101–110.

[4] E.N. Gilbert, E.F. Moore, Variable-length binary encodings, Bell Systems Tech. J. 38 (1959) 933–968.

[5] L. Gotlieb, D. Wood, The construction of optimal multiway search trees and the monotonicity principle, Internat. J. Comput. Math. Sec. A 9 (1981) 17–24.

[6] D.A. Huffman, A method for the construction of minimum redundancy codes, Proc. Inst. Radio Engineers 40 (1952) 1098–1101.

[7] A. Itai, Optimal alphabetic trees, SIAM J. Comput. 5 (1976) 9–18.

[8] D.E. Knuth, Optimum binary search trees, Acta Inform. 1 (1971) 14–25.

[9] D.E. Knuth, The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading, MA, 1973.

[10] L.L. Larmore, Height restricted optimal binary trees, SIAM J. Comput. 16 (1987) 1115–1123.

[11] E.S. Schwartz, B. Kallick, Generating a canonical prefix encoding, Comm. ACM 7 (1964) 166–169.

[12] R.L. Wessner, Optimal alphabetic search trees with restricted maximal height, Inform. Process. Lett. 4 (4) (1976) 90–94.