March, 1983

OPTIMUM SPLIT TREES

Yehoshua Perl

DCS-TR-125

Department of Computer Science
Rutgers University
New Brunswick, N.J. 08903

and

Bar-Ilan University
Ramat-Gan, Israel

Partially supported by the NCRD-CNRS exchange program while visiting the Institute de Programmation, Universite de Paris VI.

March, 1983

OPTIMUM SPLIT TREES

Yehoshua Perl

DCS-TR-125

Department of Computer Science
Rutgers University
New Brunswick, N.J. 08903

and

Bar-Ilan University
Ramat-Gan, Israel

Partially supported by the NCRD-CNRS exchange program while visiting the Institute de Programmation, Universite de Paris VI.

Abstract

A split tree is a special kind of a binary search tree introduced by Sheil [S]. He conjectured that constructing an optimum split tree is an intractable problem since there is a difficulty in applying the dynamic programming technique. We realize that the difficulty arise since top down decisions are required while applying the bottom up dynamic programming technique.

We demonstrate that it is possible in this case to overcome such a difficulty, and present a polynomial algorithm for constructing an optimum split tree. This algorithm incorporates top down decisions into a dynamic programming procedure similar to Knuth's [K2] algorithm for constructing an optimum binary search tree. An improved algorithm of complexity $O(n^4)$ is finally presented. A modification for the case that equiprobable keys are permitted is discussed.

1. Introduction

In [S] Sheil introduces the new technique of split trees for searching in a sequence of keys with a highly skewed frequency distribution. A split tree is a binary search tree each node of which contains two key values. The first is a node value which is a maximally frequent key in the subtree rooted at this node. The second is a split value which partitions the remaining keys of this subtree (with respect to their lexical ordering) between the left and right subtrees of this node, where the split value itself belongs (according to an arbitrary decision) to the left subtree.

The advantage of the split tree is the combination of two approaches used in constructing economical frequency ordered binary search (FOBS) trees. The first is the greedy approach storing the most frequent key of each subtree at the root of this subtree (see Knuth [K1] p. 432). The second is the divide and conquer approach balancing between the left and right subtrees of each node (see [K1] p. 439). Both approaches do not guarantee optimum FOBS trees. A compromise between these two approaches as suggested in [WG] yields better results but yet does not guarantee optimum FOBS trees (see also [K1] p. 439).

In the split tree both approaches can be applied simultaneously without interfering with one another. The first approach can be applied for the node values and the second approach can be applied for the split values. Two comparisons are required for each node while searching in a split tree. The first is an identity comparison with the node value and the second is an inequality comparison with the split value. However, two comparisons are usually required for each node in a FOBS tree (unless the language used has a 3-way branching command). Thus for the price of extra storage, the split tree may require a shorter average search time than a FOBS tree. For example, the example median split tree provided by Sheil [S] requires shorter average search time than an optimum FOBS tree.

An optimum split tree (OST) is a split tree which minimizes the average search time. Sheil [S] remarks that there is a difficulty in applying the dynamic programming technique which finds an optimum FOBS tree [K2] (see also K1 pp. 434-439) for finding an OST because: "The removal of the most frequent key into the node value, irrespective of its position in the lexical order, results in the set of keys that has to be petitioned between the left and right subtrees being noncontiguous in the lexical order." It is conjectured in [S] that finding an OST is computationally intractable. In other words the difficulty is to incorporate the top down decisions concerning the node values of the roots of the subtrees into the bottom up dynamic programming procedure.

We consider first the case in which no two keys are equiprobable. In Section 2 we demonstrate that such difficulties can be overcome after applying a careful analysis of the problem. A polynomial dynamic programming procedure for constructing an optimum split tree is presented. This procedure is a modification of Knuth's [K2] algorithm for constructing an optimum FOBS tree incorporating in it the top down decisions concerning

the node values of the roots of the subtrees. Another example of incorporating top down decisions into a bottom up dynamic programming procedure appears in [CFP].

In Section 3 an improved version of the algorithm with complexity $O(n^4)$ is presented, where n keys are given. The improvement is based on a modification of an O (n) speedup which appears in [K2].

In Section 4 the case in which equiprobable key are allowed is considered. A modification of the above algorithm is described. This modified algorithm is less efficient than the above algorithm.

2. Constructing an Optimum Split Tree (OST)

We are given n keys $K(1)<K(2)<...<K(n)$ with the probability $p(i)$, $1\le i\le n$ of searching for the key K(i). We assume first that $P(i)\neq P(j)$, $i\neq j$. The case in which equiprobable keys are allowed is considered in Section 4. For convenience we assume the existence of two artificial keys $K(0)=-\infty$ and $K(n+1)=\infty$ and that $p(0)=p(n+1)=0$. The probability of searching a key K such that $K(i)<K<K(i+1)$ is q(i).

Let T be a split tree for the given sequence of probabilities such that K(i), $1\le i\le n$, is a node value in the level l(i) and l'(i), $0\le i\le n$, is the level of the terminal node of T representing a search of a key K such that $K(i)<K<K(i+1)$. If the level of the root of T is defined as zero then the cost C(T) of an average search in T is

$$C(T) = \sum_{i=1}^{n} p(i)\ (l(i)+1) + \sum_{i=0}^{n} q(i)\ l'(i).$$

Let $[i,j]$, $0\le i<j\le n$ denote the subsequence of probabilities $(q(i), p(i+1), q(i+1),..., q(j-1), p(j))$. Let $[i,j,k]$, $0\le i<j\le n$, $0\le k<j-i$, denote the subsequence of $[i,j]$ which does not contain the k largest p(i)'s in $[i,j]$. Let $OT[i,j,k]$ denote an optimum split subtree for the subsequence $[i,j,k]$.

The following theorem shows that although the subsequence of probabilities $[i,j,k]$ of a subtree of an OST is noncontious in the lexical order of the given sequence of probabilities, we still know exactly which k of the probabilities are missing.

Theorem 1: Let T_1 be the subtree of an OST T, containing the probabilities of the subsequence $[i,j]$ except for k of its p(i)'s. Then T_1 contains the subsequence $[i,j,k]$ of the probabilities.

Proof: Let p(l) be any of the probabilities of $[i,j]$ not contained in T_1. The structure of a split tree implies that the corresponding key K(l) appears as a node value of one of the nodes on the path from the root of T to the root of T_1. Thus K(l) was chosen as a node

value for the root of a subtree T_2, of T, containing the subtree T_1. Hence, by the way node values are chosen, p(l) was a largest search probability of the subsequence of probabilities of T_2.

The subsequence of probabilities of T_1 is a subsequence of the subsequence of the probabilities of T_2. Thus p(l) is not smaller than any of the p(i)'s of T_1. Hence T_1 contains the subsequence [i,j,k] of probabilities.■

By the principle of optimality (see e.g., [DL]) the split subtrees of an OST are optimum split subtrees. Thus the above theorem implies that a dynamic programming procedure which finds the optimum split subtrees OT(i,j,k) for every triple (i,j,k) such that $0 \le i < j \le n$, $0 \le k < j-i$, can construct the OST=OT(0,n,0).

Let COT(i,j,k) denote the cost of an average search in OT(i,j,k). Let M(i,j,k) denote the index of the largest search probability p(l) in the subsequence [i,j,k]. The weight of the subsequence [i,j,k] is

$$W(i,j,k) = \sum_{p(l) \in [i,j,k]} p(l) + \sum_{l=1}^{j-1} q(l)$$

The root of the subtree OT(i,j,k) contains the node value K(M(i,j,k)) and the split value K(SP(i,j,k)).

Now we present the dynamic programming procedure OST for finding an optimum split tree.

```
Procedure OST

1. (Initialization of M)

for i = 0, n-1  do M(i,i+1,0) = i+1. M(i,i+1,1) = 0

        for j = i+2, n  do

                if p(j) > p(M(i,j-1,0)) then M(i,j,0) = j

                                        else M(i,j,0) = M(i,j-1,0).

                for k = 1,j-i-1  do

                        if p(j) > p(M(i,j-1,k-1)) then M(i,j,k) = M(i,j-1,k-1)

                                                else if p(j) > p(M(i,j-1,k))

                                                                then M(i,j,k) = j

                                                                else M(i,j,k) = M(i,j-1,k)

                                        end

                        end

                end

2. (Initialization of W)

    W(n,n+1,0) = q(n). W(n,n+1,1) = q(n)

    for i = 0, n-1  do  W(i,i+1,0) = q(i)+p(i+1). W(i,i+1,1) = q(i).

                for j = i+2,n+1  do W(i,j,0) = W(i,j-1,0) + q(j-1) + p(j).

                        for k = 1,j-i  do

if p(j) > p(M(i,j-1,k-1)) then W(i,j,k) = W(i,j-1,k-1) + q(j-1)

                        else W(i,j,k) = W(i,j-1,k) + q(j-1) + p(j).

                                end

                        end

                end
```

3. (Computing the cost COT and the split values SP of the optimum split subtrees.)

for i = 0, n do COT(i,i+1,0) = W(i,i+1,0)

COT(i,i+1,1) = W(i,i+1,1)

end

for d = 2, n+1 do

for i = 0, n+1−d do

for k = 0,d do

COT(i,i+d,k) = W(i,i+d,k) + MIN[COT(i,l,n) + COT(l, i+d,k−n)]

where MIN is taken over i <l<i+d and MAX(0,k+l−i−d) ≤n≤MIN(l−i,k)

SP(i,i+d,k) = l minimizing COT(i,i+d,k)

end

end

end

end OST

The validity of this dynamic programming is straightforward after applying the principle of optimality.

The complexity of this procedure is clearly $O(n^5)$ while the storage requirement is $O(n^3)$.

3. Improving the Complexity of the Procedure

Although the OST procedure is polynomial its complexity $O(n^5)$ is quite high. In this section we present an improvement reducing the complexity to $O(n^4)$.

In [K2], Knuth reduces the complexity of the dynamic programming procedure for constructing an optimum FOBS tree from $O(n^3)$ to $O(n^2)$ using the following observation: Adding a new key, which is larger, in the lexical order, than all other keys to the tree (i.e., adding it to the right of the tree) never forces the root of the optimum tree to move to the left. He uses this observation to bound the search for the root of the subtree of [i,j] between the roots of the subtrees of [i,j−1] and [i+1,j]

Naturally we try to modify this observation for our procedure. But SP(i,j,k)>SP(i,j−1,k) is not always true. However we have the following observations.

Lemma 2: If $p(j)<M(i,j-1,k-1)$ then $SP(i,j,k) \geq SP(i,j-1,k)$
else $SP(i,j,k) \geq SP(i,j-1,k-1)$

Proof: In case $P(j) \in [i,j,k]$ Knuth's observation implies that $SP(i,j,k) > SP(i,j-1,k)$. But this is not necessarily true in case $P(j) \notin [i,j,k]$ where there exists an index l, $i<l<j$, such that $p(l) \in [i,j,k]-[i,j-1,k]$. But in case $p(j) \notin [i,j,k]$ we have $SP(i,j,k) \geq SP(i,j-1,k-1)$, since then $[i,j,k] = [i,j-1,k-1] \cup \{q(j-1)\}$. Note that $p(j) \in [i,j,k]$ if and only if $p(j)<M(i,j-1,k-1)$.

Hence if $p(j)<M(i,j-1,k-1)$ then $SP(i,j,k) \geq SP(i,j-1,k)$
else $SP(i,j,k) \geq SP(i,j-1,k-1)$. ■

Similarly we obtain:

Lemma 3: If $p(i) \leq M(i+1,j,k-1)$ then $SP(i,j,k) \leq SP(i+1,j,k)$
else $SP(i,j,k) \leq SP(i+1,j,k-1)$.

Lemmas 2 and 3 imply:

Theorem 4: The OST procedure can be modified by replacing the range for l, $i<l<i+d$, in minimizing $COT(i,i+d,k)$ in Step 3, by $L_1 \leq l \leq L_2$ where L_1 and L_2 are computed as follows.

If $p(i+d) < M(i, i+d-1,k-1)$ then $L_1 = SP(i,i+d-1,k)$
else $L_1 = SP(i,i+d-1,k-1)$

if $p(i) < M(i+1,i+d,k-1)$ then $L_2 = SP(i+1,i+d,k)$
else $L_2 = SP(i+1,i+d,k-1)$.

Corollary 5: The length of the range of l is bounded by

$$\begin{aligned}&MAX[SP(i+1,i+d,k),\ SP(i+1,i+d,k-1)] - MIN[SP(i,i+d-1,k),\ SP(i,i+d-1,k-1)] + 1\\ &<1 + SP(i+1,i+d,k) - SP(i,i+d-1,k) + SP(i+1,i+d,k-1) - SP(i,i+d-1,k-1).\end{aligned}$$

In order to evaluate the complexity of the improved procedure we count the number of operations required for computing the values $COT(i,i+d,k)$ for $i = 0,n+1-d$ and fixed values of d and k. The sum of the lengths of the ranges of l for computing these values is bounded by

$$\sum_{i=0}^{n+1-d} 1+SP(i+1,i+d,k)-SP(i,i+d-1,k)+SP(i+1,i+d,k-1)-SP(i,i+d-1,k-1) =$$

$$=n-d+2+\sum_{i=0}^{n+1-d} SP(i+1,i+d,k)-SP(i,i+d-1,k)+\sum_{i=0}^{n+1-d} SP(i+1,i+d,k-1)-SP(i,i+d-1,k-1) =$$

$$=n-d+2+SP(n+2-d,n+1,k) - SP(0,d-1,k) + SP(n+2-d,n+1,k-1) - SP(0,d-1,k-1) < 3n$$

We used the fact that the two series are telescoping series. Taking in account the ranges of d,k and n we obtain $O(n^4)$ complexity.

4. Equiprobably keys are permitted

In this section we present a modification of the OST algorithm for the case where equiprobably keys are permitted.

First we show the necessity of such a modification. Suppose, for example, that the c most frequent keys are equiprobable. Then each of these c keys can be chosen as the node value for the root of the tree. Each of the c choices may yield a tree of different cost. See for example the sequence of keys K(1), K(2),...,K(9) with the probabilities P(i), $1 \le i \le 9$, 0.05, 0.1, 0.25, 0.08, 0.04, 0.08, 0.25, 0.09, 0.06 and q(i)=0 $0 \le i \le 9$. The same applies for every interval [i,j,k]. Thus, by permitting equiprobably keys we lose the advantage of choosing uniquely the node values.

Hence, in order to obtain an optimum split tree we have to consider any possible choice of a most frequent key of [i,j,k] as the node value K(M(i,j,k)), for all intervals [i,j,k] containing equiprobably most frequent keys. Let [i,j,k] contain c most frequent keys. Then we have to consider c choices for M(i,j,k) and for the interval [i,j,k+1] obtained from [i,j,k] by omitting M(i,j,k). For each of these c intervals [i,j,k+1] there are c−1 possible choices for M(i,j,k+1) and the interval [i,j,k+2]. But actually there are not c(c−1) but $\binom{c}{2}$ choices of [i,j,k+2] to be considered since two of the c most frequent key were omitted in this interval. Similarly there are $\binom{c}{[c/2]}$ choices of the interval [i,j,k+[c/2]] to be considered and this is the maximum number of choices for any interval [i,j,k+d], $1 \le d \le c$.

Fortunately there is only one possible choice of [i,j,k+c] where all c most frequent keys are omitted from [i,j,k]. Hence we are ready at this point for a similar treatment of [i,j,k+c] if it contains several most frequent keys. This treatment is independent with the choices in processing [i,j,k+d], $0 \le d < c$.

Let c be the maximum number of equiprobable keys. Then the maximum number of choices of any interval is $\binom{c}{[c/2]}$. Hence we can modify the OST algorithm to handle equiprobable keys by keeping any possible choice of the variables for each interval [i,j,k] and optimize over all possible choices of the variables whenever relevant. This modification yield an extra factor of at most $\binom{c}{[c/2]} = O(2^c/c^{1/2})$ to the time and space complexity of the algorithm. In most practical applications c is a small constant and the modified algorithm is polynomial.

Acknowledgement

We wish to thank the referee for pointing out the difficulty with equiprobable keys.

References

[CFP] Choueka Y., Fraenkel A. S. and Perl Y., "Polynomial Construction of Optimal Prefix Tables for Text Compression" in the Proceedings of the Nineteenth Annual Allerton Conference on Communication Control and Computing, 1981, 762-768.

[DL] Dreyfus S. E. and Law A. M., The Art and Theory of Dynamic Programming, Academic Press, New York 1977.

[K1] Knuth D. E., The Art of Computer Programming Vol 3: Sorting and Searching, Addison-Wesley, Reading, Mass. 1969.

[K2] Knuth D. E., "Optimum Binary Search Trees," Acta Informatica 1(1971), 14-25.

[S] Sheil B. A., "Median Split Trees: A Fast Lookup Technique for Frequently Occurring Keys," CACM 21(1978) 947-958.

[WG] Walker W. A. and Gotlieb C. C., Graph Theory and Computing, Academic press, 1972.