Limiting The Search for 2-Dimensional Optimal Alphabetic Trees

Ahmed Belal*
Email: belala@usa.net
Dept. of Computer Science &
Automatic Control,
Faculty of Engineering,
Alexandria University,
Egypt.

Magdy A. Ahmed
Email: magdya@alex.eun.eg
Dept. of Computer Science &
Automatic Control,
Faculty of Engineering,
Alexandria University,
Egypt.

Shymaa M. Arafat
Email: shymaa@alex.eun.eg
Informatics Institution,
Mubarak City for Scientific
Research,
Alexandria,
Egypt.

Abstract

Two-dimensional alphabetic trees have many applications in a wide variety of diverse fields. Although, a relatively fast algorithm that finds an approximate optimal alphabetic tree (OAT) does exist, yet, the only way to find the exact one is to use dynamic programming. In dynamic programming the (OAT) is found by simply examining all nodes in the two dimensional array of weights as possible roots (*cuts*) at each level of the tree and finding the one with minimal cost. In this paper, we introduce the concept of for each cut to limit the search for an optimal solution. The measure of goodness is a value we call *the expense of the cut*, only cuts with expense less than a given limit L are considered *good cuts*. Then at each level of the tree, only good cuts are tested as possible candidates for an optimal solution.

Key Words: alphabetic tree - optimal tree - dynamic programming -two dimensional alphabetic trees- branch and bound.

1 Introduction

With the advent of technology and information exchange all around the world, the ability to store and quickly retrieve large amounts of information became a persistent need. A large information system is essential to any corporation; digital libraries and dictionaries are common in any institution. Consequently, handling information systems has become a major computer application and a wide area for continuous research.

One kind of information systems is probabilistic retrieval systems where fast retrieval is guaranteed to high frequency items. Alphabetic trees are a suitable structure for such systems. When data need to be stored in a multi dimension structure, like in multimedia and visualization for example, multi-dimensional alphabetic trees are used.

A tree is considered optimal if its cost is minimal over all possible trees for the same set of weights [1,2]. The problem of obtaining the OAT for the *M* x *N* two dimensional array of weights is equivalent to the problem of minimizing the cost of cutting the array into individual cells using planar cuts along the edges, where the cost of a cut is equal to the sum of the weights of its two parts. The root of the tree corresponds to the first cut. In what follows, we will interchangeably use the words root and cut. Similarly, by an optimal tree we will mean an optimal alphabetic tree.

Fast *O(n log n)* algorithms for finding the one dimensional OAT were found early in the seventies [1,3]. However, no fast algorithm was developed for the two dimensional problem at that time. Then, and for the existence of new applications, recent research was directed to the OAT problem in the last few years [4,5,6]. Some results for finding an approximate two-dimensional OAT were reported in [7]. Other results were also reported for the one dimensional problem [8,9].

Although this two dimensional problem can be solved in polynomial time ($O(N^4)$) when $M=O(N)$ using dynamic programming, this is not fast enough for many applications. Also, the computation becomes more excessive when the dimensions increase; that is, the power of the polynomial function increases with the number of dimensions k ($O(N^{k+2})$). In this paper, we use a measure of goodness for each cut to limit the number of feasible solutions to the problem and thus, reduce the number of possible choices for the root of the optimal tree. Here, we examine this method for the two-dimensional case, but the concept is applicable to higher dimensions as well.

In section 2 for each cut and how its value is computed. In section 3, the expected number of good cuts is discussed using experimental results. In section 4, an algorithm for finding the OAT for an arbitrary *MxN* two dimensional array is introduced. Section 5 contains experiments that empirically show the effectiveness of our technique. The paper is concluded in section 6.

2 The Expense of a Cut

The expense of a cut is defined as *the value this cut has contributed to the deviation from the optimal solution*. A cut is considered if its expense is less than a limit L. First, we define a lower bound on the cost of OAT, then we define the limit L. Finally, we explain the way of computing the expense of each cut.

2.1 Bound

If $T(R_i)$, $T(C_j)$ denote the cost of the optimal trees for row i and column j respectively, then the cost of the optimal tree for the two dimensional array is bounded by:

$$T_{\text{optimal}} \geq \text{Bound} = \sum_{i=1}^{M} T(R_i) + \sum_{j=1}^{N} T(C_j) \qquad (1)$$

This lower bound is achieved when all the rows, or all the columns, have the same optimal tree.

2.2 The Limit L

The limit L is set to be:

$$L = T_{\text{approx}} - \text{Bound} \qquad (2)$$

Where T_{approx} is obtained using a fast greedy algorithm for finding an approximate OAT [7].

Substituting from Eq.(1), the limit L becomes:

$$L = T_{\text{approx}} - \left(\sum_{i=1}^{M} T(R_i) + \sum_{j=1}^{N} T(C_j) \right)$$

2.3 Computing the Expense

Suppose that a vertical cut divides an MxN array into a left array and a right array. If each of the resulting arrays can be solved such that their respective lower bounds are achieved, then we have:

$$C(\text{left}) = \text{Cost of tree of left part} = \sum T(R_i^{'}) + \sum T(C_j^{'})$$

$$C(\text{right}) = \text{Cost of tree of right part} = \sum T(R_i^{''}) + \sum T(C_j^{''})$$

Where R',C' are the rows and columns of the left part, and R", C" are the respective ones for the right part.
The resulting tree obtained by making this vertical cut will have the cost:
Cost of tree = C(left) + C(right) + W
where W is the weight of the original array
The expense (Ex) of the cut is defined as the deviation of this cost from the lower bound of the original array. Thus,

$$Ex = \sum T(R_i^{'}) + \sum T(C_j^{'}) + \sum T(R_i^{''}) + \sum T(C_j^{''}) + W - \sum T(R_i) - \sum T(C_j)$$

Giving

$$Ex(\text{vertical cut}) = \sum T(R_i^{'}) + \sum T(R_i^{''}) + W - \sum T(R_i) \qquad (3)$$

Similarly, if the cut is horizontal, we get

$$Ex(\text{horizontal cut}) = \sum T(C_j^{'}) + \sum T(C_j^{''}) + W - \sum T(C_j) \quad (4)$$

As an example, for the array with two rows R_1, R_2 and *N* columns the horizontal cut has zero expense, using Eq.(4) and noting that in this case T(C_j')=T(C_j")=0 for j=1,2 and $\sum T(C_j) = W$. The optimal tree in this case has the lower bound of Eq.(1) as its cost. This is expected since in the *2xN* arrays, all the columns have the same optimal tree.

3 Good Cuts

Before carrying this concept any further, we need to have some idea about the number of good cuts we are going to have at each step. First, we present experimental results on the number of good cuts at the first level and their nature.

3.1 Experimental Results

Although we do not have a theoretical bound on the number of good cuts, sample runs can give us some indication. Through more than 1000 runs for each of 25 different values of M&N, the number of good cuts never exceeded half the possible cuts, and it is about one third in the average. Also, through all these runs good cuts where always adjacent to each other; we never encountered a case where a bad cut is between two good cuts. Figure 1 shows the minimum, maximum and average number of good cuts at the first step for an *n x n* square array.

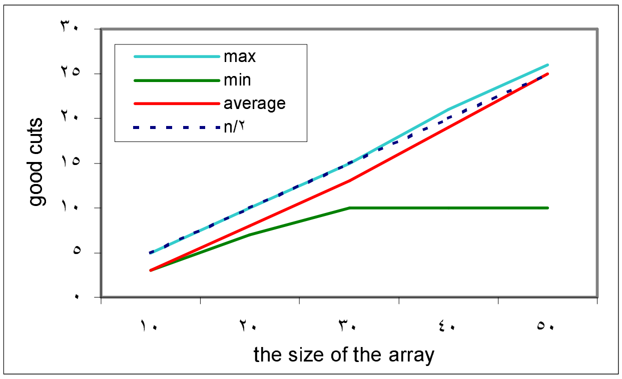

Figure 1: the number of good cuts at the first level

Further more, we expect the number of good cuts to decrease at each step.

4 Proposed Method

In this section, we describe the proposed algorithm, which uses the concept of goodness, then we follow by an example to demonstrate the process.

4.1 Algorithm

Given an MxN array of weights, the limit L is computed from Eq.(2) then the expense of all possible (M-1)+(N-1) cuts is computed using Eq.s(3),(4). Only the cuts with expense less than L are retained as candidates to an optimal solution. Then for a given candidate with expense Ex<L, the two parts generated by the cut are explored for candidate cuts by looking for a pair of cuts, one for each part, whose sum of expenses is less than the new limit (L-Ex). Finally, when the size along one of the dimensions reaches the value4, no further exploration is needed and dynamic programming can be used to obtain the optimal tree for this array.

4.2 Example

For the purpose of illustration, the procedure is applied to the simple 5x5 array of weights shown in figure 3.

260	62	244	183	6
230	140	127	49	93
249	168	116	s63	241
115	194	91	179	126
28	169	50	126	269

Y_1 Y_2 Y_3 Y_4

X_1 X_2 X_3 X_4

Figure 3 : a 5x5 array

The cost of the optimal tree for each row and column is obtained and this sum determines the bound of equation (1): Bound = 15892.

The cost of the approximate optimal tree is determined and found to be: T_{approx} = 16236 and the limit is set to the value: L = 16236-15892 = 344.

Using Eq.s (3), (4) we now determine all vertical and horizontal cuts with expense $<$ 344. There is only one such cut at X_2 with expense Ex(X2) = 72. This is the only possible cut that may result in a tree with cost less than T_{approx} = 16236.

The left part produced by the cut X_2 is an array with two columns and thus the cut at X_1 will have zero expense.

We now have to look for a cut of the 5x3 right array whose expense is less than

L - Ex(X_2) = 344 - 72 = 272. The only possible cut is Y_3 with expense Ex(Y_3) = 103.

The resulting two parts will be cut respectively at Y_4 with zero expense and X_3 with expense Ex(X_3)=125.

The optimal tree is now obtained, and it will have a total cost:

$$T_{optimal} = \text{Bound} + 72 + 103 + 125$$
$$= \text{Bound} + 300 = 16192$$

5 Experiments

Since we do not have a theoretical estimate on the number of good cuts, we can not reach a closed form time complexity of the algorithm. We only know the added complexity in finding T_{approx} which is *O(MNlogN)*. The summations calculated for finding the bound are needed in further steps whether in the proposed algorithm or in the dynamic programming method. The reduction in further steps depends on the number of good cuts. Thus, we measured the performance of the algorithm using experimental results. Run time is measured for the proposed algorithm and the dynamic programming as the only existing optimal algorithm for finding OAT. Figure 4 presents the curves and shows the advantage of using the proposed method.

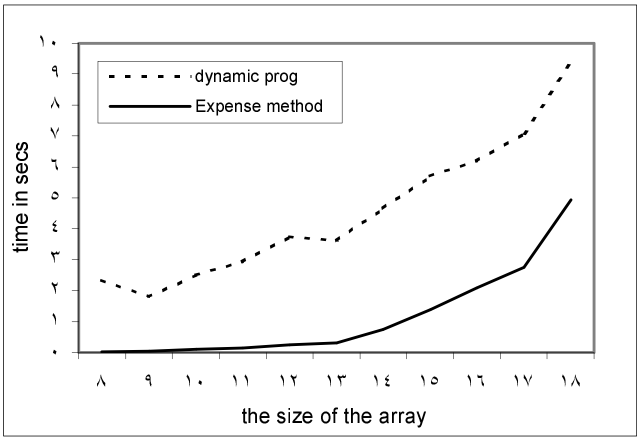

Figure 4: Runtime of the proposed algorithm versus dynamic programming

6 Conclusion

In this paper, we have introduced a new concept for limiting the search for a two-dimensional OAT. A value we call *the expense of the cut* is used to measure the of each possible cut (root). Only cuts with expense less than a given limit L are considered *good cuts.* An algorithm is suggested as a possible way to benefit from such concept where cuts are tested for goodness, then at each level only good cuts are tested as possible candidates for an optimal solution. The performance of the algorithm is tested and compared to the dynamic programming ; results show improvement in run time.

References

[1] T. C. Hu and A. C. Tucker "Optimal computer search trees and variable length alphabetical codes", *SIAM Journal on Applied Mathematics*, 21(4) :514-532, 1971.

[2] Alon Itai "Optimal alphabetic trees", *SIAM Journal on Computing*, 5(1):9-18, March 1976.

[3] A. M. Garsia and M. L. Wachs "A new algorithm for minimum cost binary trees", *SIAM Journal on Computing*, 6(4) :622-642, 1977.

[4] P. Ramanan "Testing the optimality of alphabetic trees", Theoretical Computer Science, 93, 1992.

[5] Teresa M. Przytycka and Lawrence L. Larmore "The optimal alphabetic tree problem revisited", In Serge Abiteboul and Eli Shamir, editors, *Automata, Languages and Programming, 21st International Colloquium*, volume 820 of *Lecture Notes in Computer Science*, pages 251-262, Jerusalem, Israel, 11-14 July 1994. Springer-Verlag.

[6] Lawrence L. Larmore and Teresa M. Przytycka " The optimal alphabetic tree problem revisited", *Journal of Algorithms*, 28(1):1-20, July 1998.

[7] M. A. Ahmed, A. A. Belal and K.M. Ahmed "Optimal insertion in two-dimensional arrays", *International Journal of Information Sciences*, 99(1/2) :1-20, June 1997.

[8] Lawrence L. Larmore and Teresa M. Przytycka "A parallel algorithm for optimum height-limited alphabetic binary trees", Journal of Parallel and Distributed Computing, 35(1):49-56, May 1996.

[9] B. M. Mumey "Some new results for constructing optimal alphabetic binary trees", Master's thesis, University of British Colombia, 1992.