ISSN: 0975-766X
CODEN: IJPTFI
Available Online through *Research Article*
www.ijptonline.com

APPLICATION OF DATA PREPROCESSING ON THE GIVEN DATA AND EFFICIENT CONSTRUCTION OF OPTIMAL BINARY SEARCH TREES USING POST DYNAMIC PROGRAMMING

S.Hrushikesava Raju*, Dr. M.Nagabhusana Rao
Professor, Department of CSE, SIETK, NarayanaVanam Road, Puttur, A.P.
Professor, Dept.of CSE, KL University, Vijayawada, A.P.
Email: hkesavaraju@gmail.com

Received on 06-08-2016 *Accepted on 10-09-2016*

Abstract

There are various methods of handling Optimal Binary search trees in order to improve the performance. One of the methods is Dynamic programming which incurs $O(n^3)$ time complexity to store involved computations in a table. The data mining technique called Data Preprocessing is used

to remove noise early in the data and enhance consistency of given data. The data postcomputing (opposite to Data Preprocessing) is applied using dynamic programming principle which starts with only required data and computes only the necessary attributes required to construct Optimal Binary Search Tree with time complexity O(n) if there are n identifiers / integers / any complex objects. This approach avoids computing all table attributes. Hence, the complexity or cost of Data post computing using Dynamic Programming is proved to be less than $O(n^3)$ or even less than specified in some cases with experimental results.

Keywords: Optimal Binary Search Tree (OBST), Data Preprocessing, Post computing, Dynamic Programming, Time Complexity.

I. Introduction

Optimal Binary search Tree is a variety of binary trees in which each node stores maximum of two children and it stores strings as identifiers within it or integers or any complex object as nodes of this binary tree. There were many methods such as greedy, recursion, memorizing are useful to do this work. One of those methods is that randomly drawing all possible binary trees and finds a particular binary tree whose cost is low and considers that result is an OBST. The name optimal binary search tree is titled because of simple reason that is finding a key in a tree incurs least number of comparisons. In this method, the optimal binary search tree is chosen in the possible binary trees

which involve minimum cost for searching a key in the tree. There are also Greedy, Memorizing, 0/1 knapsack problem, multiple chain multiplication used but they are not efficient because they are all use strategy recursion and another drawback is solutions are not helpful to compute larger problem solution. Dynamic programming is a popular and efficient method to construct binary search tree by following principles such as saving sub problem solutions, reusing sub problem solutions to construct solution for larger problems, and also it avoid using of recursion strategy. Dynamic programming unnecessarily stores all computations in memory and involves $O(n^3)$ complexity and it is applied on any data that is irrelevant or in improper manner. Hence, A data mining technique named data preprocessing is used to sort or clean the given sequence. Also, a technique is proposed titled Data Post computing using Dynamic Programming concept is performed that only computes required attributes which are required to construct optimal binary search trees. This second step leads to compute some more low level attributes in order to compute that particular attribute.

II. Related Work

There are many methods involved to construct the optimal Binary search trees (OBST). First approach [4,8,9] called randomization which constructs many binary search trees in order to find out an OBST that has minimum cost. The cost can be calculated by multiplying the key and frequency of that node. This approach has a drawback that wastage of time in computing unnecessary trees in addition to correct tree.

Second approach named Computing OBST using sets[6] arranges elements in a tree using recursive approach in which computing optimal substructures and overlapping sub-problems is became expensive(exponential nature) in terms of computing same sub-problems again and again. This issue is sorted in Using Dynamic Programming by a temporary array in a bottom up manner.

Third approach is Greedy recursive approach [7, 8] which initially finds root through best heuristic using recursion and it applies to remaining elements for computing sub trees. This leads sometimes not a OBST although root of each sub tree is optimal.

To overcome these flaws, Dynamic programming[8,9] is introduced which split the given problem into small instances, proposed algorithm applied to all, later integrate them into a large solution for the given problem. The advantage of dynamic programming is flexibility in integrating the sub problems and each solution takes a space in memory and solution is obtained using recurrence formulas. But still, some refinement is possible on this work that is refinement of DP algorithm is required which reduce the unnecessary computations. This reduces space in a table in

storing the elements as well as computations done in less time compared to other techniques. This work can be done in the proposed work where detailed steps are given.

The techniques are summarized in the following table.

Table 1: Details of each technique used for OBST.

Technique	Advantage	Disadvantage
Randomization	Simple, easy and mandatory task	Takes more time in giving right tree with minimum cost
Using sets	Optimal sub-structures	Expensive in avoiding overlapping sub-problems
Greedy	Guarantee the optimal in each case	Using Recursion cause
Traditional Dynamic Programming	Giving a tree optimally	Leads many Unnecessary computations
Proposed Dynamic Programming(Post DP)	Gives a tree optimally with little time complexity	NIL

III. Proposed Work

Consider the given data elements are as a_1,a_2,a_3, …., a_{n-1},and a_n. These labels denote the keywords in the given data. Suppose the raw data is given, which can be classified into the text keywords, individual characters, and numeric data. For numeric data, it is easy to construct binary search tree by using rules such as left node element should be less than root node and right sub tree element should be greater than root element and this procedure is repeated until last element in the given data is processed. For alphabets, it is also easy to construct binary search trees by using same logic but left sub tree element is alphabetically comes before the root element, the tight sub tree element is alphabetically comes after the root element, and this procedure is applied till last character is processed. For text keywords to make as an OBST, there are predefined procedures also called algorithms which pick the root node first by a notation t_{ij} which is the last level calculation that helps to pick the root node which can be indexed by the rank value calculated using DP_OBST algorithm. From this notation t_{ij}, root can be picked using the rank r which takes ar element in the given data. The sub trees in the next low level are taken based on this rank r. The first level sub trees are t_{ir-1} (left sub tree) and t_{rj} (right sub tree). The second level nodes can be estimated from t_{ir-1} and t_{rj}. The rank of entry (i,r-1=k) is r1 assume. The sub trees of this are t_{ir1-1}(left sub tree) and right sub tree t_{r1k} (right sub tree). Assume the rank of first level right sub tree is r2. The sub trees of $t_{r=ij}$ are t_{ir2-1} and t_{r2j}. This is repeated until all elements in the given data are taken a place in the tree.

The following are the procedures to process the data and construct the OBST from the category of data like numeric, individual characters (alphabets), and text words.

A. Irrespective of the category of data, this is going to be applied on the raw data.

Procedure DP_OBST(string text[])

Take three arrays one for numeric- num[ ], second for alphabets – letter[], third for text words – words[]

Take three indexes m,n,l

1. read the data based on spaces.
2. while reading, check the following conditions

```
if(sizeof(text[i])== 1 && isalpha(text[i]))
{
 letter[n]=text[i]
 n++;
 }
if(sizeof(text[i])==1 && isdigit(text[i]))
{
num[m]=text[i];
m++;
}
if(sizeof(text[i])>1)
{
 words[l]=text[i];
 l++;
}
```

3. Now, raw data classified into their allocated arrays.

A.1 For numeric, and alphabet data, the procedure to construct OBST is same as general approach which is given as follows:

Procedure OBST_Numeric_and_alphabets(num[] / letter[])

1. take first node as root.

2. take second element, compare it with root value. If it is same, override it. If it is less than the root, make as left child of it. if it is greater than the root, make it as right child. Assume num[i] / letter[i] is value. Use also pointer concept.

```
struct node
{
struct node *leftptr;
struct node *rightptr;
int or char value;
}
if(root==NULL)
root=value;
else
{
if(root==value)
skip this iteration
if(value < root )
{
root->leftptr=value;
OBST_Numeric_and_alphabets(root->leftptr);
}
if(value > root )
{
root->rightptr=value;
OBST_Numeric_and_alphabets(root->rightptr);
}
```

3. This is repeated until last element is processed.

A.2 But for the textual words, the optimal binary search tree is going to be constructed in the bottom up manner using the Dynamic programming formulae.

To do this, the normal calculation of Dynamic Programming is recapped here.

Procedure of DP()

The formula for computing weight, cost, and rank are:

$w(i,j) = p(j)+q(j)+w(i,j-1)$

$c(i,j) = w(i,j) + \min_{i<k<j} \{c[i,k-1]+c[k,j]\}$

$r(i,j) = k$ that minimizes the $c(i,j)$

1. Take the initial values of costs, weights, ranks are 0.

2. This procedure calculates many unnecessary attributes in which relevant attributes are also existing

```
For i:=0 to n-1 do

w[i,i] := q[i]; r[i,i] = 0; c[i,i] =0;
// Optimal tree with one node
  w[i,i+1] := q[i] + q[i+1] + p[i+1];
  r[i,i+1] := i+1
  c[i,i+1] := q[i] + q[i+1] + p[i+1];     }
  For m:=2 to n do
  // find optimal trees with m  nodes
   For i:=0 to n-m do    {
    j := i+m;
   w[i,j] := w[i,j-1] + p[j] +q[j];
    k := find(c,r,i,j);
  //it returns the value to the k  that minimizes the c
   c[i,j] := w[i,j] + c[i,k-1]+c[k,j];
   r[i,j] := k;     }         }
```

3. From the obtained table of entries, some entries are picked based on rank which has a relation to next level computations

4. Based on Knowledge of Proposed work, OBST for textual words is going to be constructed.

B. By taking this as base, the logic is rewritten in order to reduce unnecessary computations. The result obtained is from post dynamic technique. The following procedure reduces waste(unnecessary) computations;

1. Take the initial weights, costs, and ranks are zero.

```
w[i,i] := q[i]; r[i,i] = 0; c[i,i] =0;
```

2. Call rank of last level computation because which helps to pick root node.

Assume the sub procedure named finding_rank_forij

Procedure finding_rank_forij(i,j)

{

$w[i,j]= \sum_{i \& j >0} w[i,j-1]+p[j]+q[j];$

$c[i,j] = w[i,j]+ \min_{i<k<j} c[i,k-1] + c[k,j];$

return k

}

3. Call the calculation of weight and cost of only related entries (i,j)

The sub procedure named calculate_w&c(i,j)

Procedure calculate_w&c(i,j)

{

For i=r to 0 do

w[i,i+1] := q[i] + q[i+1] + p[i+1];

r[i,i+1] := i+1

c[i,i+1] := q[i] + q[i+1] + p[i+1]; }

4. Now only relevant weights and costs are calculated in bottom up fashion from present level to zeroth level to get final value for their attributes. These values helpful to know the rank and element with rank as index takes right position in the OBST. The following is the flowchart which demonstrates the OBST in bottom up fashion from last level to initial level which reduce unnecessary computations.

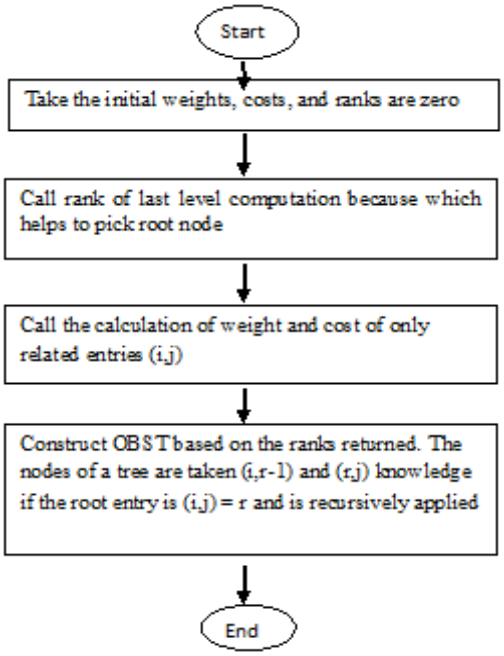

Fig. 1: Procedure for constructing OBST using Dynamic Programming knowledge.

IV. Experimental Results

The results are taken by considering some real sample example scenarios.

Example: Take a set (do, if, int, while), Let p(1: 4) = (3, 3, 1, 1) and q(0: 4) = (2, 3, 1, 1, 1).

The following table shows the comparison between traditional Dynamic Programming and Proposed Dynamic Programming using Data Preprocessing and Post Computing from bottom up manner.

Technique	**Space**	**Time**
Traditional Dynamic Programming	45 * sizeof(data)	O(15*3)=45
Proposed Dynamic Programming using Data preprocessing and post computing in bottom up manner	4 * sizeof(data)	O(4 * 3 + 5 for initial variables + some intermediate variables but not all variables) < 45

The following table shows the calculations using traditional Dynamic Programming

j ⟶ 0	**1**	**2**	**3**	**4**
i ↓ **0**: $w_{00} = 2$, $c_{00} = 0$, $r_{00} = 0$	$w_{11} = 3$, $c_{11} = 0$, $r_{11} = 0$	$w_{22} = 1$, $c_{22} = 0$, $r_{22} = 0$	$w_{33} = 1$, $c_{33} = 0$, $r_{33} = 0$	$w_{44} = 1$, $c_{44} = 0$, $r_{44} = 0$
1: **$w_{01} = 8$**, **$c_{01} = 8$**, **$r_{01} = 1$**	$w_{12} = 7$, $c_{12} = 7$, $r_{12} = 2$	$w_{23} = 3$, $c_{23} = 3$, $r_{23} = 3$	**$w_{34} = 3$**, **$c_{34} = 3$**, **$r_{34} = 4$**	
2: $w_{02} = 12$, $c_{02} = 19$, $r_{02} = 1$	$w_{13} = 9$, $c_{13} = 12$, $r_{13} = 2$	**$w_{24} = 5$**, **$c_{24} = 8$**, **$r_{24} = 3$**		
3: $w_{03} = 14$, $c_{03} = 25$, $r_{03} = 2$	$w_{14} = 11$, $c_{14} = 19$, $r_{14} = 2$			
4: **$w_{04} = 19$**, **$c_{04} = 32$**, **$r_{04} = 2$**				

From above table, only bolded entries are required to construct OBST. There are some attributes need to be computed in calculating the required attribute. In this way, computing of some unnecessary attributes are going be avoided using Proposed Dynamic Programming using Data Preprocessing and bottom up post computing method.

By observing a simple example, the complexity is more. If You the data set is large, huge number of entries to be computed in a dynamic programming table which not only leads to occupying space and also huge time. To avoid this overhead, Proposed technique named Modified Dynamic programming using Data Preprocessing and attribute post computing in bottom up fashion from last level to first level depending on required attributes.

The following is also a graph that shows efficiency of the traditional and proposed Techniques:

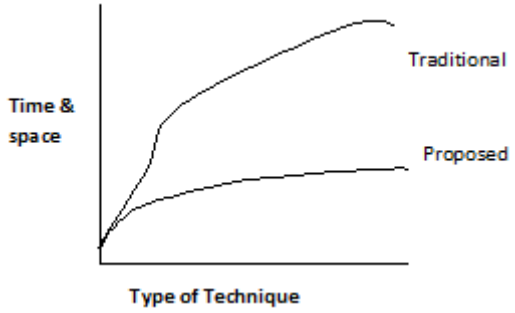

Fig.2: Efficiency of Traditional and Proposed Techniques.

V. Conclusion

This approach is very useful early in removing the redundancy in the dataset and later as saving space and time in terms of not to compute unnecessary attributes using Post Dynamic Programming Approach. The results are described through examples shown that unnecessary space is removed and also minimized the time to construct binary search tree. This approach may be extended in future using either backtracking or any latest designed algorithm to get better time efficiency.

References

1. "Data Mining: Concepts and Techniques" by Micheline Kamber, Hei, Second Edition.
2. iasri.res.in/ebook/win_school_aa/notes/Data_Preprocessing.pdf
3. www.mimuw.edu.pl/~son/datamining/DM/ preprocess.pdf
4. http://www.cse.yorku.ca/~andy/courses/3101/lecture-notes /OptBST.pdf
5. http://fileadmin.cs.lth.se/cs/Personal/Rolf_Karlsson/lect5.pdf
6. http://www.geeksforgeeks.org/dynamic-program ming-set-24-optimal-binary-search-tree/
7. http://www.sciencedirect.com/science/article/pii/ 0020019081901435
8. "Fundamentals of data structures in C++" By E.Horotiwz, S.Sahni, Dinesh Mehta, Second Edition.
9. http://vitconference.com/vit_mca/images/resources/DAOA/Fundamentals-of-Computer-Algorithms-By-Ellis-Horowitz-1984.pdf

10. https://en.wikipedia.org/wiki/Optimal_binary _search _tree

11. http://www.cs.ccsu.edu/~markov/ccsu_courses/ datamining-3.html

12. http://iasri.res.in/ebook/win_school_aa/notes/ Data_Preprocessing.pdf

13. http://staffwww.itn.liu.se/~aidvi/courses/06/dm/ lectures/lec2.pdf

14. "Data Preprocessing in Data Mining" by Salvadar Garcia, Julian Luengo, Francisco Hurrera, ISBN : 9783319102474 & 9783319102467.

15. http://www.enggjournals.com/ijcse/doc/IJCSE16-08-01-009.pdf

16. http://www.ijcse.net/issue.php?file=vol05issue1

17. http://journals.indexcopernicus.com/issue.php?id=737&id_issue=882666

Corresponding Author:

S.Hrushikesava Raju,

Email: *hkesavaraju@gmail.com*