Volume 3, Issue 4, April 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Pre-determined Equations for Efficient Search Algorithms in a Search Space of the VLSI layout

S.T.Sonia
Student
Department of Computer Science,
Bharath University, India.

Dr.Nalini
Professor,
Department of Computer Science,
Bharath University, India.

Abstract - This research is concerned with incorporating the algorithms that are used inside the VLSI layout design. Area, Speed, Power dissipation, Design time and Testability are the important entities to be optimized when a VLSI circuit is designed [8]. But the complexity is simply too high. The two concepts Hierarchy and Abstraction are helpful to deal with the complexity. Hierarchy shows the structure of the design and the Abstraction hides the lower level details. This research mainly focuses the area of the Physical (or Layout) domain. VLSI layout designs are typically represented as graphs. Generally vertex orderings is induced by the traversals such as depth-first search or breadth-first search. The performance of one traversal algorithm may be better than the performance of any other traversal algorithms for a particular vertex of a graph or a general tree. So it is necessary to determine a best suitable algorithm to find a goal vertex instead of using any one of them randomly. This research will give a solution for this problem. The standard algorithms, Breadth First Search (BFS) and Depth First Search (DFS) are used in this research. So the best suitable algorithm for a particular vertex will be chosen from these two algorithms without traversing.

Keywords - VLSI Layout design, Search Algorithms – Breadth First & Depth First, Heuristics Algorithm

I. Introduction

1.1 Motivation and Goal

The floor plan of an Integrated Circuit can be represented hierarchically [8]: cells are built from other cells, except for those cells that are at the lowest level of the hierarchy. These lowest-level cells are called leaf cells. Cells that are made from leaf cells are called composite cells. The direct sub cells of a composite cell are called its children. Every cell, except for the one representing the complete circuit, has a parent cell. The following figure 1.3.1(A) shows the Schematic of some circuit at the register transfer level that consists of an ALU (Arithmetic Logic Unit), three registers (RA, RB, RC), two multiplexers(M1, M2) and a controller(Cont). VLSI layout designs are typically represented as graphs. A basic “layout” problem is to arrange the nodes of the graph in a limited area using net list so as to ensure that no edges cross. A net list hyper graph H(V,E) consists of a set of modules (vertices) $V = \{v_1,v_2, . . ., v_n\}$ and a set of nets (hyper edges) $E = \{e_1,e_2, . . ., e_m\}$. A cluster Ci is a nonempty subset of V, and a k-way clustering P^k is a set of k clusters such that every vi $\in$V *belongs* to exactly one cluster in P^k. The vertex ordering that capture the clustering structure of a net list hyper graph, such that the vertices in any contiguous subset of the ordering form a “good” cluster[1]. For this purpose, vertex orderings is induced by the traversals such as depth-first search or breadth-first search. Generally the search algorithms like Breadth First Search (BFS), Depth First Search (DFS), and Depth First Iterative Deepening (DFID) etc are used to traverse any search-space like graph. The net list hyper graph and the general tree are some examples for graph. So any improvement in the search algorithms will be very useful in vertex ordering. The performance of one search algorithm may be better than the performance of any other search algorithms for a particular vertex of a graph or a general tree. The algorithms like Breadth First Search (BFS), Depth First Search (DFS), and Depth First Iterative Deepening (DFID) can be used to traverse any tree entirely. But it necessary to determine a best suitable algorithm to search the goal vertex instead of using any one of them randomly. Then the identified algorithm can be used to traverse the tree to reach the goal vertex.

1.2 Methods used for the research

1) Deriving new equations for basic search algorithms
2) Comparative Analysis of the existing algorithms with/ without the equations in terms of time complexity.

The identified algorithm can be used to traverse the tree for better search of the goal-vertex. So the unsuitable search algorithms will be ignored in the searching of a particular goal-vertex. Also the search becomes faster

1.3 Background Survey

The background contains numerous case studies of graph algorithms and VLSI layout design using graph algorithms. The background also contains the study of the optimal path without growing the tree [2] and the exact path length problem [3]. The background also suggests that, in the last few years, the VLSI layout design uses the graphical representation to represent the integrated circuits (IC_s). This type of representation is quite effective for a wide variety of real-world graphs, including graphs from finite-element meshes, circuits, street maps, router connectivity, and web links [4]. In addition to significantly reducing the memory requirements, our implementation of the representation is faster than standard representations for queries [4].

1.3.1 Tree Representation for a Floor plan

Any hardware structure can be mapped with a floor plan. Figure 1.3.1(A) shows the schematic of some circuit at the register-transfer level that consists of an ALU (arithmetic logic unit), three registers (RA, RB and RC) two multiplexers (M1 and M2) and a controller.

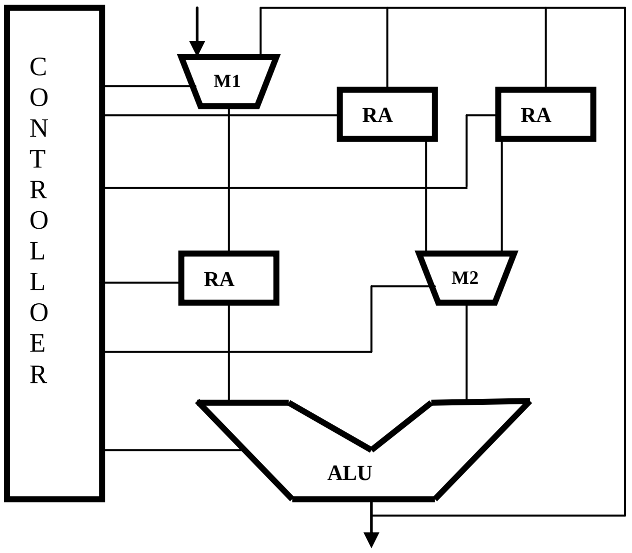

Figure 1.3.1(A): A structural description of some circuit

Figure 1.3.1(B) shows a possible floor plan annotated with the names of the cells and nets of the schematic. The sub blocks have different shapes. This is one of the main characteristics of floor plan-based design.

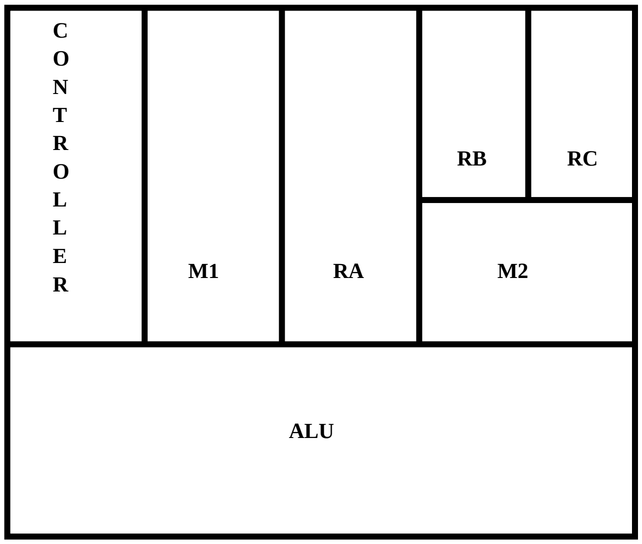

Figure 1.3.1(B): The floor plan for the circuit in figure 1.3.1

The figure 1.3.1(C) shows the slicing tree for the floor plan of figure 1.3.1(B). In the figure the node labeled with an 'H' was obtained by horizontal composition and a node labeled with a 'V' by vertical composition. The ordering of a composite node's children is relevant: 'from left to right' in horizontal composition and 'from bottom to top' in vertical composition.

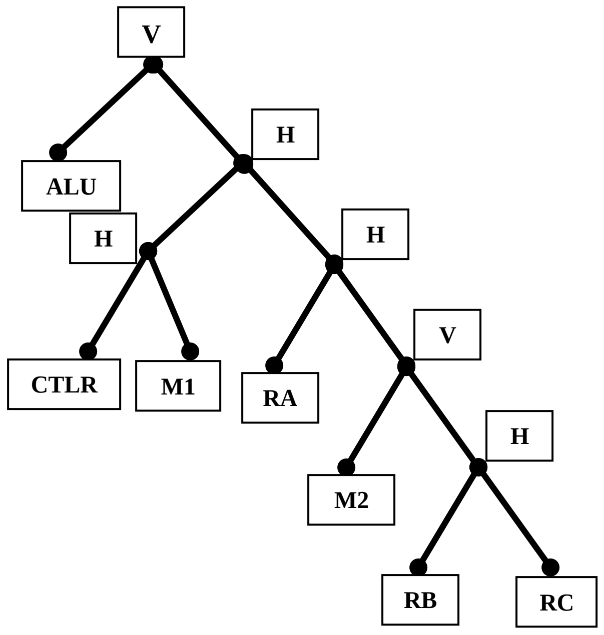

Figure 1.3.1(C): The slicing tree for the floor plan of figure 1.3.1(B)

1.3.2 Introduction to Graph Traversal Algorithms

A tree is a graph with vertices or nodes and links or edges and without cycles as shown in figure 1.3.2(A). Also a tree will have n number of nodes and (n-1) number of edges.

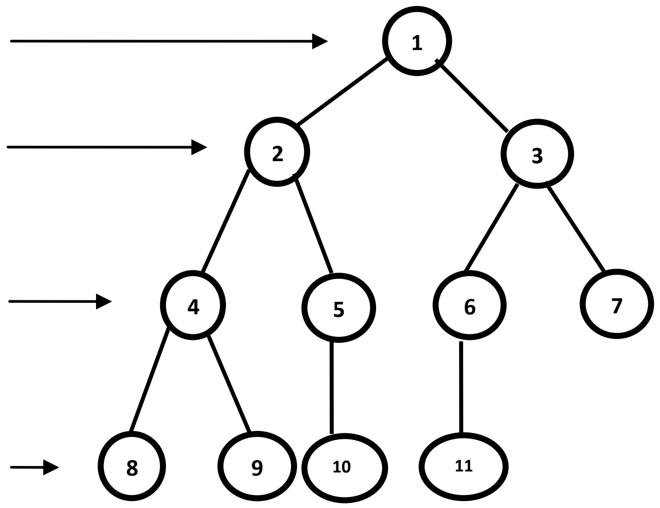

Figure 1.3.2(A): A general tree with 11 nodes

A tree which has nodes with maximum two children of each node is called binary tree. If there is no limitation in the number of children, the tree is called general tree. A tree can be traversed using some of the traversal methods like depth first search (DFS), breadth first search (BFS), uniform cost search (UCS), depth limited search (DLS) and depth first iterative deepening (DFID) [5]. The two standard methods of traversing through a graph are DFS and BFS.

1.4 Problem Definition

The background survey for the representation of VLSI layout design using the standard traversal algorithms says that the algorithms are used randomly to traverse the graph. So the algorithm chosen is not always suitable, sometimes for some goal nodes the suitable algorithm is not used even though it is available. The suitable algorithm for a particular goal node should be pre determined without doing any traversal. Then the determined algorithm can be used to traverse the tree to reach the goal node. The suitability of the algorithm is calculated from the length of the path of the goal node from the root node. The path is the number of nodes to be visited to reach the goal node. The equations to find the number of nodes to be visited to reach the goal node are to be derived.

1.5 Organization of this research

This research consists of two chapter including introduction as the first chapter. Apart from the introduction chapter-1 contains the background survey for the representation of VLSI layout design using the standard traversal algorithms says that the algorithms are chosen randomly and are used to traverse the graph. The chapter-1 also contains the problem definition of the standard traversal algorithm predetermination details. The research presentation is discussed in the chapter-2. The chapter-2 also contains the implementation of modified algorithms for standard traversal patterns using a linear array named as parent array. The conclusion obtains from the implemented data structure and algorithm is also carried out in this chapter. The derivation of equations for DFS and BFS are given in chapter-3. The chapter-3 also contains the comparative analysis of the search techniques with and without using the equations. The results of the research are given in chapter-4.

1.6 Proposed System

The new search algorithms are using the parent array representation of a tree. But the best algorithm for a particular node has to be predetermined by using any effective method without traverse the tree. For which two equations are derived and the best algorithm for searching a goal node is determined easily. Information about the nodes in the tree is the region. So this research can be further enhanced by using the characteristics of the regions in the tree.

II. A General Tree Representation

2.1 Introduction

The existing search algorithms are using two dimensional matrixes to represent any graph. The two dimensional data structure occupies more memory space to represent the graph. So to represent a tree with n nodes we need n × n memory spaces. Also the search algorithms using this data structure will have to use more variables for indexing. So the size of the programming will be increased automatically. To avoid such disadvantages a new single dimensional data structure is used in this research. So the algorithms for searching using this single dimensional array are developed.

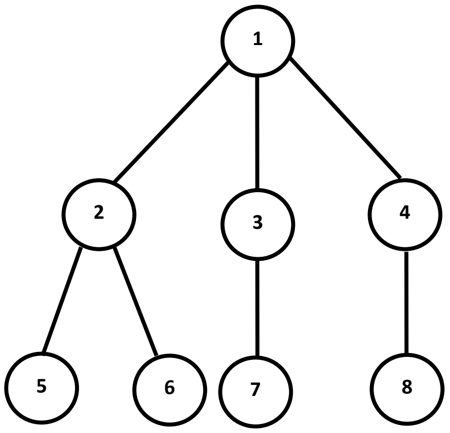

Figure 2.1: A general tree with 8 nodes

2.2 General Tree Representations

The tree is used as a search space and is generated using a random tree generator. The tree in figure 2.1 has 8 nodes and 7 links. This tree can be represented using a two dimensional array as shown in figure 2.2

1		*	*	*				
2	*				*	*		
3	*						*	
4	*							*
5		*						
6		*						
7			*					
8				*				

Figure 2.2: Two dimensional representation for the tree in figure 2.1

The memory spaces required for the representation is 8×8 = 64. The indices represent the nodes and the cell represents the links. If any cell is marked, it represents the corresponding indices have a link. There are 7 edges in the tree. So 7 cells are marked with * symbol. Also the images of the all links are marked in the array. The memory spaces required for the representation is 8×8 = 64. The indices represent the nodes and the cell represents the links. If any cell is marked, it represents the corresponding indices have a link. There are 7 edges in the tree. So 7 cells are marked with * symbol. Also the images of the all links are marked in the array.

2.3 Parent Array Representation

The requirement of memory locations is highly reduced by using parent array representation. The parent array is a single dimensional array in which only the parents of a tree are stored and the children are used as indices. The array size is the number of nodes in the tree. The root node will not have parent node. So the first location is empty. This can be used for some other purpose. The tree in figure 2.1 can be represented as shown in figure 2.2 with 8 memory locations by using the parent array representation method.

0	**1**	**1**	**1**	**2**	**2**	**3**	**3**

Figure 2.3: Parent array representation for the tree in figure 2.1

2.4 Tree Traversal using Parent Array Representation

Since the data structure is changed, the algorithm for the standard tree traversal patterns have to be developed. The algorithm for the Depth first search is as follows. The algorithm needs the Goal node, Root node and Parent array as inputs. The output is the depth first traversal pattern.

```
INPUT    : Goal_node, Root_node, Parent_array
OUTPUT : //DFS pattern for given tree
dfs(int Goal_node,int Root_node,int Parent_array[])
begin
        INTEGER : i ,total_nodes
        Total_nodes ← Parant_array[0]
        if(Root_node == Goal_node)          Parant_array[1] ← 1
        for(i=Root_node; i<= total_nodes;i++)
        if(Parent_array[i] == Root_node)
        begin
                        if(Parent_array[1] == 0)
                        OUTPUT  Root_node
                        CALL  dfs(Goal_node,i,Parent_array)
        end
        if(Parent_array[1] == 0)
        begin
                OUTPUT  Root_node
        end
end
```

The algorithm for breadth first search is given below. The input for the algorithm is Goal node and the parent array. The output is the breadth first traversal pattern.

```
INPUT    : Parent_array,Goal_node
OUTPUT : //BFS pattern for given tree
bfs(int Parent_array[],int Goal_node)
begin
        INTEGER i
        if(Goal_node <= Parent_array[0])
        begin
                for(i=1;i<Goal_node;i++)
                        OUTPUT  i
        end
else
        begin
                OUTPUT "Goal node not found: message"
        end
end
```

2.5 Algorithms used for this research

1. Depth First Search

2. Breadth First Search

2.5.1 Depth First Search

A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path as shown in figure 2.5.1. For example, after searching 1, then 2, then 4, then 8, the search backtracks and tries another path from 4 likewise the nodes are explored in the order 1 2 4 8 9 5 10 3 6 11 7. But some of the nodes are re-visited when back-tracking. So the actual search order will be 1 2 4 8 4 9 4 2 5 10 5 2 1 3 6 11 6 3 7. So the algorithm visits all of the nodes in a connected graph by expanding the search from the currently visited node [6]. Initially, we set visited (i) = 0 for all nodes i. After visiting each node, it is marked as visited. The algorithm is initially called from any initial node of the graph.

Depth first search (DFS) is useful for

- Find a path from one vertex to another
- Whether or not graph is connected

Computing a spanning tree of a connected graph.

2.5.2 Depth First Search Algorithm [7]

Algorithm starts at a specific vertex S in G, which becomes current vertex. Then algorithm traverse graph by any edge (*u*, *v*) incident to the current vertex *u*. If the edge (*u*, *v*) leads to an already visited vertex v, then we backtrack to current vertex u. If, on other hand, edge (u, *v*) leads to an unvisited vertex *v*, then we go to *v* and *v* becomes our current vertex. We proceed in this manner until we reach to "dead-end". At this point we start back tracking. The process terminates when backtracking leads back to the start vertex. Edges leads to new vertex are called discovery or tree edges and edges lead to already visit are called back edges.

DFS (G, *v*)
Input: A graph G and a vertex v.
Output: Edges labeled as discovery and back edges in the connected component.

```
For all edges e incident on v do
    If edge e is unexplored then
        w ← opposite (v, e) // return the end point of e distant to v
        If vertex w is unexplained then
            - mark e as a discovery edge
            - Recursively call DFS (G, w)
        else
            - mark e as a back edge
```

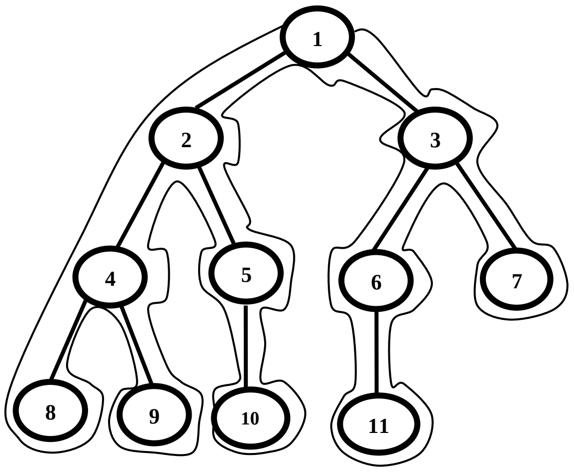

Figure 2.5.1: DFS for the tree in figure 1.3.2(A)

2.5.3 Breadth First Search

A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away as shown in figure 1.3.5(A). For example, after searching 1, then 2, then 3, the search proceeds with 4, 5, 6, 7 likewise the nodes are explored in the order 1 2 3 4 5 6 7 8 9 10 11 as shown in figure 1.3. Breadth First Search Always visits an unvisited node that is closest to the starting vertex or root node.

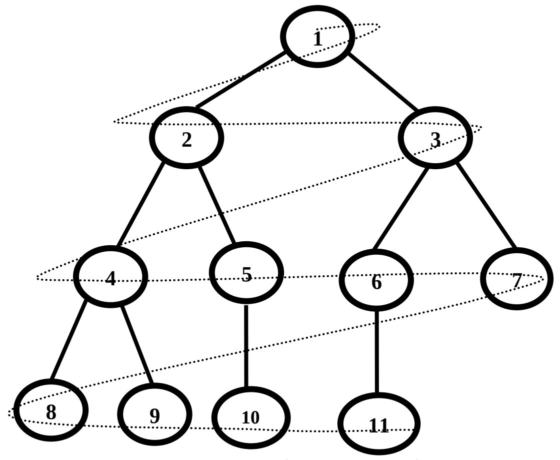

Figure 2.5.3: BFS for the tree in figure 1.3.2(A)

Breadth First Search algorithm is used in

- Prim's MST algorithm.
- Dijkstra's single source shortest path algorithm.

Like depth first search, BFS traverse a connected component of a given graph and defines a spanning tree.

2.5.4 Breadth First Search Algorithm [7]

BFS starts at a given vertex, which is at level 0. In the first stage, all vertices at level 1 are visited. In the second stage, all vertices at second level are visited. These new vertices, which are adjacent to level 1 vertex, and so on. The BFS traversal terminates when every vertex has been visited.

BFS (G, S)
Input: A graph G and a vertex.
Output: Edges labeled as discovery and cross edges in the connected component.

```
Create a Queue Q.
ENQUEUE (Q, S)    // Insert S into Q.
While Q is not empty do
    for each vertex v in Q do
        for all edges e incident on v do
            if edge e is unexplored then
                let w be the other endpoint of e.
                if vertex w is unexpected then
                    - mark e as a discovery edge
                    - insert w into Q
                else
                    mark e as a cross edge
```

BFS label each vertex by the length of a shortest path (in terms of number of edges) from the start vertex.

2.5.5 Heuristics

The idea of a "heuristic" is a technique, which sometimes will work, but not always. It is sort of like a rule of thumb. Most of what we do in our daily lives involves heuristic solutions to problems. They usually work, or they usually work well enough, when they don't work, we then deal with that. We often use the word "heuristic" not just to describe cases where a solution might not be found, but to describe cases where we want to find the best solution (according to some way to measure best-ness). A heuristic might help us to find solutions, which are good, but perhaps not the very best they can be. This is the idea of heuristics that we will talk about today. Obviously the measure of goodness, and the assessment of a heuristic technique, is going to be relative to the domain, and to the specific job that problem solving is going to be applied to in that domain.

2.5.6 Heuristic Search

This last heuristic for the driving domain is the basic idea behind most "heuristic search" techniques in artificial intelligence. If you can somehow estimate which states are "closest" to a goal state, using some numerical measure of closeness, this estimate can be used to decide which states to consider applying operators to next. Best-First Search, Hill Climbing, Minimizing Cost and A* Search are the best heuristic search techniques.

III. Derivation Of Predetermined Equations

3.1 Introduction

The new search algorithms for the most standard traversal patterns DFS and BFS are using the parent array representation of a tree. But the best algorithm for a particular node has to be predetermined by using any effective method without traverse the tree. For which two predetermined equations are derived using a general tree. These equations are tested for all nodes of many general trees and the best algorithm for searching a goal node will be determined easily.

3.2 Derivation of the Equation for BFS

The numbering of the nodes in a general tree is given level by level. So the parent array representation of a tree contains the parent of each node. The index of each location in the parent array is the child node of the node which is stored in the same location. Also the numbering of the node is breadth-wise. So the number of nodes to be traversed to reach the goal node is the order of the node immediately previous to the goal node. For example to reach the goal node, 8 in the figure 3.1, the number of nodes to be visited is 7. (i.e. 8-1 = 7). Thus the predetermined equation for BFS can be derived as follows.

$$\mathbf{n(N_V) = O(N_G) - 1}$$

Where, $\mathbf{N_V}$ - Visited nodes
$\mathbf{N_G}$ - Goal node
$\mathbf{n(N_V)}$ - Number of visited nodes
$\mathbf{O(N_G)}$ - Order of the Goal node

Let the goal node is 'n'. In BFS the number of visited nodes before reaching the goal node 'n' is n-1.

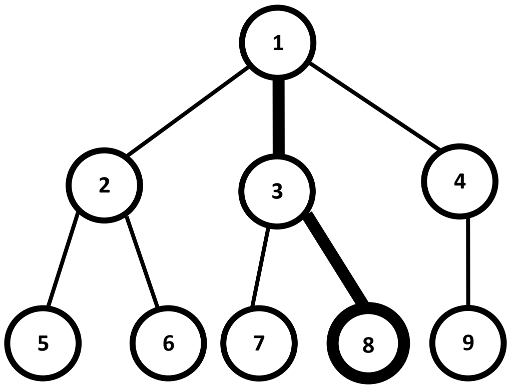

Figure 3.1: A general tree with 9 nodes

3.3 Derivation of the Equation for DFS

The Depth First Search pattern contains the nodes in depth-wise as explained in the chapter 1. The DFS uses backtracking technique, when it fails to find the goal node in the left most leaf. By using backtracking it can go back to the parent of the leaf. Then it starts the search through the next immediate right leaf of the parent node. So each parent node in the left leaf is revisited as much as the number of children is has. So the nodes in the leaf are visited two times of the number of children. If the goal node found in any of the leaf then the search is stopped. The leaf which contains the goal node is called **root path**. So the children in the root path are not revisited. This idea is used to derive the equation for DFS.

In DFS the number of child nodes before the root path are visited two times. The nodes in the root path are visited only one time. So the equation can be derived as follows

$$\mathbf{n(N_V) = n(N_R) + (2 \times n(N_C)}$$

Where, $\mathbf{N_V}$ - Visited nodes
$\mathbf{N_R}$ - node in root
$\mathbf{N_C}$ - child node
$\mathbf{n(N_V)}$ - Number of visited nodes
$\mathbf{n(N_R)}$ - Number of nodes in root path
$\mathbf{n(N_C)}$ - Number of child nodes before the root path

For example to reach the goal node, 8, the root path is 1, 3, 8. The node in the root path except the goal node is 2. The child node before the root path is 4. So the nodes to be visited for the goal node 8 is 2+ (2 × 4) = 10. This equation needs some data about the tree. The required data can be collected easily from the parent array of a tree.

3.4 Comparative Analysis of BFS and DFS

Any general tree can be divided in to four regions like left top (LT), right top (RT), left bottom (LB) and right bottom (RB) as shown in figure 3.2. The nodes in each region have its own characteristics as shown in figure 3.3. BFS and DFS are good for the left top region nodes. BFS is good for the right top region nodes. DFS is good for the left bottom region nodes. Both BFS and DFS are poor for the right bottom nodes.

A searching of a node without any knowledge is tedious and not time consumable. Also some traversal algorithm will give best solution for some of the nodes. But for other nodes the same algorithm will not give the better. For example a node 'n' can be reached only after visiting few nodes in BFS and the same node can be reached by visiting more number of nodes in DFS. So in this case the BFS is best for search 'n'. If the DFS is chosen for the search of the node 'n' the result will not be good. The calculation time for the equation of BFS is the execution time of a subtraction operation. So it is negligible when comparing with the time consumption of blind search. The calculation time for the equation of DFS involves the construction time of boundary arrays and root path. Since it is common for all the nodes and also it is a onetime calculation, this time can be shared by the number of nodes in the tree. So it also becomes negligible when comparing with the time consumption of blind search. A general tree with 2500 nodes is generated using the random tree generator for this research. Since the calculation time slice is very small, the tree with minimum number of nodes cannot give efficient result. Even the tree with 2500 nodes also gives a very small quantum of time. For the selected nodes, the result is generated and which is compared with the time slices of the searches with and without the equations. The result and report are given in chapter 4.

IV. Results

```
              ORIGINAL ARRAY
            8 0 1 1 1 2 2 3 4

BFS
      Goal
      2: 1
      3: 1 2
      4: 1 2 3
      5: 1 2 3 4
      6: 1 2 3 4 5
      7: 1 2 3 4 5 6
      8: 1 2 3 4 5 6 7

DFS
      Goal
      2: 1
      3: 1 2 5 2 6 2 1
      4: 1 2 5 2 6 2 1 3 7 3 1
      5: 1 2
      6: 1 2 5 2
      7: 1 2 5 2 6 2 1 3
      8: 1 2 5 2 6 2 1 3 7 3 1 4
```

V. Conclusion

The predetermined equations are used to determine the best algorithm for a particular node in a tree. The tree is generated using the Random tree generator. Instead of using blind search technique, the search with the prior knowledge of the tree and the algorithms used, the objective node can be reached easily and the time complexity is also reduced.

References

[1] C.J.Alpert and A.B.Kahng UCLA Computer Science Department, Los Angeles, CA 90024-1596 **"A General Framework for Vertex Orderings, with Applications to Net list Clustering"**

[2] Danny Z. Chen, Ovidiu Daescu, Xiaobo (Sharon) Hu and Jinhui Xu Journal of Algorithms Volume 49, Issue 1, October 2003, Pages 13-41 **"Finding an optimal path without growing the tree"**

[3] Matti Nykänen and Esko Ukkonen, Department of Computer Science, University of Helsinki, P.O. Box 26, (Teollisuuskatu 23), 00014, Helsinki, Finland Journal of Algorithms Volume 42, Issue 1 , January 2002, Pages 41-53 **"The Exact Path Length Problem"**

[4] Daniel K. Blandford Guy E. Blelloch Ian A. Kash, Computer Science Department Carnegie Mellon University, Pittsburgh, PA 15213, fblandford,
blelloch,iakg@cs.cmu.edu **"An Experimental Analysis of a Compact Graph Representation"**

[5] **"www.cs.uni.edu/~schafer/courses/161/sessions/s09.ppt"**

[6]**"http://www.ugrad.cs.ubc.ca/~cs320/Lectures/cs320lec5.pdf"**

[7] **"http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html"**

[8] Sabih H.Gerez.(1999) John Wiley & Sons **"Algorithms for VLSI Design Automation"**

[9] N.A. Sherwani. (1999) Tata Mc Hill "**Algorithm for VLSI Physical Design Automation"**

[10] Ellis Horowitz and Sartaj Sahni, (1999) Tata Mc Hill **"Fundamentals of Computer Algorithms"**