Analysis Of The Depth First Search Algorithms

Navneet kaur , Deepak Garg
Computer Science and Engineering Department,
Thapar University, Patiala

Abstract — *When the traditional Depth First Search(DFS) algorithm is used for searching an element in the Directed Acyclic Graphs (DAGs),then a lot of time is wasted in the back-tracking .But this discusses the Reverse Hierarchical Search (RHS) algorithm .For the DAG tree structure the RHS algorithm provides the better performance by avoiding the unnecessary search .This paper also presents a parallel formulation of the Depth First Search which retains the storage efficiency of the sequential depth first search and can be mapped on to any MIMD architecture. In this paper we have tried to improve the searching of any node in the DFS by combining the features of both the RHS algorithm and the parallel formulation of the DFS which helps in maintaining the storage efficiency and reduce the search which is unnecessary. In the RHS algorithm we use the previous node information (so that the duplicity can be prevented) to find the next nodes for searching. The main features that affect the parallel formulation is the dynamic work distribution technique which divides the work between different processors .The performance of the parallel formulation is strongly affected by the technique of work distribution and various features of the architecture such as presence or absence of shared memory, relative speed of the communication network. When we combine the features of both RHS algorithm and parallel formulation of DFS, it gives good performance*.

Keywords — RHS algorithm , parallel formulation ,DFS ,work distribution schemes, directed acyclic graphs , MIMD architecture.

I. INTRODUCTION

A search algorithm is a set of steps used for finding an item with specified properties among a collection of items. The items may be stored individually as records in a database; or may be elements of a search space defined by a mathematical formula or procedure.

There are many classes of the searching algorithms like searching algorithms for the virtual search spaces, for sub-structures of a given structure, for quantum computers. We use the different kind of algorithms for performing different kind of tasks .But there can be many algorithms, for performing the same kind of task, each having its own advantages and disadvantages depending upon for which kind of data structure it is being used ,its memory requirements, its time complexity etc.

So it is very difficult to decide which algorithm or the data structure is will be best for performing a specific task. The wrong choice of the data structure and algorithm for a task may lead to the production of the wrong result or to the results which are not optimal. So at the time of selection of an algorithm or the data structure for a task, we have to check the many trade-offs. A particular algorithm may be efficient and may provide the optimal result for a specific task but there is no algorithm that is best in all the aspects. So when we select an algorithm that produce the optimal results in most of the cases if not every time. First of all a brief definition of Depth First Search shows that Depth First Search is a general technique used in Artificial Intelligence for solving a variety of problems in planning ,decision making, theorem proving ,expert systems, etc . DFS is algorithm for traversing or searching a tree or graph. One starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

This paper represent the analysis of the Depth First Search algorithms. As we discussed above, there are also many algorithms for solving the Depth First Search problem, each having its own advantages and disadvantages**.** This paper represents that the problems that are caused by some algorithm can be solved by using another algorithm and we can make the algorithm more efficient by combining the features of some algorithms like the problems that may occurred in the simple DFS algorithm can be removed by using the Iterative Deepening Depth First Search algorithm, in Reverse Hierarchical Search (RHS) Algorithm the performance has been improved by eliminating need of backtracking it also solves the problem of DFS algorithm. Several aspects of parallel formulation DFS has been represented, which shows that by using some technique we can parallelize the DFS which will improve the performance.

II. Literature Review Of Tree Search Algorithms And Their Analysis

This section presents a review of the related work available in the literature on search algorithms in the domain of tree data structures and investigates any scope for improvement.

A. The Depth First Search Algorithm

DFS algorithm is a searching algorithm in which search progresses by expanding the first child node of the root that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it hasn't finished exploring. In a non-recursive implementation, all freshly expanded nodes are added to a stack for exploration. An example has been given below

For the following graph

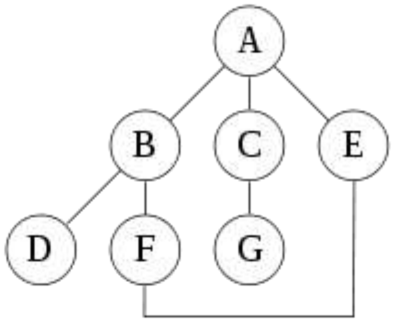

Fig. 1

It is assumed that a Depth First search starts at A , the left edges in the given graph are chosen before right edges, and the search remembers previously-visited nodes and it does not repeat them , then it will visit the nodes in the following order: A, B, D, F, E, C, G.

when the same search is performed by without remembering previously visited nodes, then it results in visiting nodes in the order A, B, D, F, E, A, B, D, F, E, etc. Forever caught in the A, B, D, F, E cycle and never reaching C or G. So sometimes there may be the problem of infinite loop and sometimes we are not able to reach the nodes to traverse them as discussed above.

These problems can be solved with the help of the Iterative Deepening Depth First Search algorithm.

When the tree is traversed, for space complexity evaluation, then the large memory will be required when the search reaches the maximum tree depth for the first time. Assume that, ‘b’ is the branching factor for each node, when a node at depth ‘d’ is inspected, the total number of nodes/vertices collected in memory is a combination of unexpanded nodes up to depth ‘d’ and the current node being studied. With *b*-1 number of unexpanded nodes at each level, the total amount of memory that is needed is d *(b -1) +1 i.e. O(|V|) , if the entire graph is traversed without repetition. To evaluate the time complexity, the time needed in searching a node is considered. In order to search a node that is located at the leftmost location at a depth *d* in a tree results in the total number of examined nodes to just being just *d* +1. On the other hand, the node at the extreme right location at depth d is being searched, then the total number of inspected nodes comprises of all the nodes in the tree and it is calculated as: $1 + b + b^2 + b^3 +\dots+ b^d = (b^{d+1} – 1) / (b – 1)$. In an average case scenario, the total number of examined nodes is evaluated as $[(d + 1) / 2] + [(b^{d+1} – 1) / (b – 1)] \approx [b (b^d + d)/2(b – 1)]$. Then the run-time complexity is evaluated as $O(b^d)$ where b is the branching factor and d is the depth.

B. Iterative Deepening Depth First Search (IDDFS) Algorithm

When the DFS algorithm is applied to the exceptionally large sized graph then it may results in non-termination issues that are based on the infinite path length in the search tree. So in a case, where the depth limit is not known in the starting, a variation of DFS algorithm can applied to solve the problem of infinite looping and in-accessible nodes. This variation is known as the Iterative Deepening Depth First Search (IDDFS).IDDFS combines Depth-First search's space-efficiency and Breadth-First search's completeness (when the branching factor is finite). The main idea behind this is that a depth-limited search is applied over and over again, by successively increment the depth in each iteration until the goal state is not found.

First, it perform a depth-first search to depth one. Then, it discards the nodes generated in the first search, start over by incrementing the depth level up to two and do a DFS to level two. Next, start over again by incrementing the depth level up to three and do a DFS to depth three, etc., this process continues until a goal state is reached. Since IDDFS expands all nodes at a given depth before expanding any nodes at a greater depth, it is guaranteed to find a shortest-length solution. At any given time it performs a depth-first search at a given depth d , and it never searches deeper than depth d. The disadvantage of IDDFS is that it performs computations which are prior to reaching the goal depth. In fact, at glance it seems very inefficient. But an analysis of the running time of IDDFS that shows that this wasted computation does not affect the asymptotic growth of the run time for exponential tree searches .An example of IDDFS's working is given below by following the fig. 1.

As discussed in the case of DFS algorithm there was a looping problem and the node C and D were inaccessible.

But it can be seen below that Iterative deepening prevents these loops and it is possible to reach the inaccessible nodes C and D on the following depths, assuming it proceeds left-to-right as above.0,1,2,3 all these represent the depth .

- 0: A
- 1: A (repeated), B, C, E

(Note that IDDFS can reach to C, when a conventional depth-first search did not.)

- 2: A, B, D, F, C, G, E, F

(Note that it still sees C, but that it came later. Also note that it sees E via a different path, and loops back to F twice.)

- 3: A, B, D, F, E, C, G, E, F, B

For this graph, as more depth is added, the two cycles "ABFE" and "AEFB" will simply get longer before the algorithm gives up and tries another branch.

The time complexity of IDDFS in well-balanced trees is same as Depth-first search, $O(b^d)$.

In case of IDDFS, the nodes on the bottom level are expanded once, those on the next to bottom level are expanded twice, and so on, up to the root of the search tree, which is expanded $d + 1$ times. So the total number of expansions in an iterative deepening search is:

The DFS algorithm can applied to solve the problem of infinite looping and in-accessible nodes. This variation is known as the Iterative Deepening Depth First Search (IDDFS).IDDFS combines Depth-First search's space-efficiency and Breadth-First search's completeness (when the branching factor is finite). The main idea behind this is that a depth-limited search is applied over and over again, by successively increment the depth in each iteration until the goal state is not found.

First, it perform a depth-first search to depth one .Then, it discards the nodes generated in the first search, start over by incrementing the depth level up to two and do a DFS to level two. Next, start over again by incrementing the depth level up to three and do a DFS to depth three, etc., this process continues until a goal state is reached. Since IDDFS expands all nodes at a given depth before expanding any nodes at a greater depth, it is guaranteed to find a shortest-length solution. At any given time it performs a depth-first search at a given depth d, and it never searches deeper than depth d. The disadvantage of IDDFS is that it performs computations which are prior to reaching the goal

$$(d+1)1 + (d)b + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + b^d$$
$$\sum_{i=0}^{d}(d+1-i)b^i$$

C. Reverse Hierarchical Search(RHS) Algorithm

This is a graph traversal algorithm for better time and space complexity. The RHS algorithm is an uninformed search similar to the normal DFS. But the DFS algorithm travels downstream from the root to leaf nodes, and it uses the concept the concept of backtracking. While the RHS algorithm does not use the concept of backtracking..This keeps a restriction on the search space boundary (upstream node till root) by limiting the depth of search from individual nodes to root. Thus it prevents indefinitely deep paths during traversals.

In the case of the complete binary tree of depth n, the worst-case search time complexity would be $O(2^n -1)$. Usually, in trees the root is a single element node while the maximum number of element occurs at the nth level. Therefore, a search commencing at the level "n" would represent a logical initiative in reducing the volume of search by $2^{(n-1)}$ elements (accounting for members at the nth level).

RHS algorithm is not only used for binary trees and can be used efficiently for non-binary versions. Because the search terminates at the root ,so no backtracking will be necessary and in this a record of solution space is kept and all the child-parent nodes found in the Reverse Hierarchical search are added up to the solution space. This saves the users from revisiting the same path for further target searches. It must be note that every single node that is traversed is added to the domain of the solution space. In this approach, special emphasis will be given for optimization of the algorithm for tree structures.

Informal Description of RHS Algorithm:

```
1. Select the vertex where the search should start
   and assign the maximum search depth
2. Verify if the current vertex is the goal state
   • If no: Go to Step 3
   • If yes: return
3.Verify if the current vertex is within the maximum search depth
   • If no: Do nothing
   • If yes:
      • Expand the vertex and save all of its predecessors in a stack
      • Call RHS recursively for all vertices of the stack and go back to step 2
```

D. The Parallel Formulation of Depth – First Search

This technique presents the idea of parallelizing the DFS by dividing the work of search to be done among the many processors. Each of the processor searches the disjoint part of the search space .when the processor has finished the searching of its own search part ,then it tries to get some more search parts from the other processors .when the goal node is found all the processors stop searching and quit. If the search space is finite and the goal node is not present in this search space then all the processors would run out of work and the search will terminate.

The part of search space of processor is represented by the stack. The depth of stack is the depth of the node that is currently explored. Each level of the stack represents the alternatives that are untried. When the local stack is empty it takes some untried alternatives from other. In some implementations, at the start of each iteration, all search space is given to one processor and other processors are given null space i.e. null stack. Then the other processors ask this processor to give them untried alternatives. Then the search space is divided and distributed among the other processors**.**

III. SCOPE OF IMPROVEMENT

The search time can be improved by combining the features of the RHS algorithm and the parallel formulation of the DFS algorithm. Because if the features of RHS algorithm are considered, then there will be no need to keep the record of backtracking. So the time for search will reduce. Because all child - parent nodes following a reverse hierarchy are added up to the solution space. This saves the users from revisiting the same path for further target searches and this keeps a limit on the depth of the tree. If the features of the parallel formulation of the DFS are considered then Depth First search can be made parallel as well. So in this technique the multiple processors are used. By using these all these features searching time, for a goal node, can be reduced.

IV. DESIGN

This type of technique can be mapped to any MIMD architecture that will have many processors and each processor will have stack. To implement this kind of algorithm, the shared memory model can be used. This model consists of a set of synchronous processors and a shared global memory through which they communicate. First of all the whole search space will be given to one processor (main processor), it will keep it in the stack (each processor has its own stack) and the other processors will have null search space or null stack and then they will request the processor which have the whole search space to provide the some search space and they can search the item by using the features of RHS algorithm (by not using backtracking, from individual nodes to root). So the when the processor has empty stack it will request another processors to provide some untried alternatives. If the target node (item) is found by any processor ,then all the processors stop searching further. If the search space is finite and the item is not found by any processor then the processors would run out of work and the search will terminated. .

V. WORKING

Consider a tree and a node is given from which search will be started. First of all the nodes at that level will be put in the stack of the main processor from left to right except the node from where the search is started, that node will be inserted at top of the stack and the other processors will have the empty stacks. Then the main processor will start the search and the other processor will ask it for providing the some untried alternatives and the main processor will provide the untried alternatives to them. Thus they can start searching in parallel. They will start their search by selecting the top node from their stacks and then explore it according to the RHSDFS procedure written below. If the node is found by any of the processors the processor then the search will be successful and all the processors will stop searching, otherwise the each of the processor will keep on trying the next alternative. If any of the processor it has empty stack then it will ask another processors to provide some more alternatives. If no processor has the enough untried alternatives then it will have to remain idle. If the goal node is not found after searching the whole search space, then the processors would run out of work and the search will terminate. The alternatives which have already been examined for searching the goal node are added to the solution space. Suppose we choose the node n which is at the depth d to start the search for a node m but if this node is not found even after trying the all alternatives, then all this information will be stored in the solution space. If search is again started from the node r which is at the more depth level than the node n, then the previous information will be used from the solution space for not searching the same paths again and again. Thus we can make the search more efficient. Its algorithms are described following:

Informal description of algorithms:

```
Parallel _ DFS : processors pI
    While (not terminated) do
        If (stack [I] = empty) then
            S = Getwork();
            Stack[ I ]=s
        While (stack[ I ]!=empty) do
            RHDFS (stack [I], depth , goal);
            Od
            Terminated _ test ();
            Od
```

In the Getwork() procedure if the processor gets the request of the another processor for untried alternatives ,If its stack is empty or it has only one entery and it is trying that entery then it will return nothing. Otherwise it will return the untried alternatives that are next to the top of the stack.
If the processor gets no alternative from the other processor then it check whether the search has been completed by using the Termination _ test () procedure.

Procedure used by each processor for searching the goal node:
RHDFS (node, depth, goal)
//node represent the node from where the search is started, depth is the depth of the node, goal represent the node which is to be find.

```
{   if (node=goal)
        Return node;
    Push _ in _ stack (node);
    While (stack is not empty)
        {   stack _counter++;
            If (depth>0)
               {  node _ recv  = expand (node);
                  If (node_ recv is not empty)
                        {   node = node_ recv ;
                            RHDFS (node,depth-1,goal)
                        }
               }

            Else //no operation
        }
}

expand (node)
   { while (node is not root)
        {   If (depth>0)
                {   nodep = node.parent();
                    //it will fetch the parent of the current node.

                }
        }

      return nodep.
}
```

VI. APPLICAIONS

The technique for improving the search time which is discussed above can be used for the trees which are very large and they have a lot of nodes, then by using this technique we can search in parallel the node and without any backtracking and visiting any path again and again for the search. But for the trees which are not much larger and have a small number of nodes this is not a good technique because in that case a lot of time will be wasted in distributing the work between the processors and keeping the track of that work. The processors will not have much work to do and some processors may be idle. So it will be costly technique for searching the node in such a case. There is also a case when there are only two processors and there are too many untried alternatives then the performance may not be so good. But when we increase the number of processors the performance starts improving than before but it will happen up to some extent, because if there are too many processors then it will cause in lot of overhead in managing the transfer of data between them.

VII. CONCLUSION

This paper presents various concepts of alternate search techniques based on the existing DFS algorithm known as IDDFS, Reverse Hierarchical Search (RHS) algorithm and the parallel formulation of DFS. During the process of searching for a node, the general DFS algorithm uses the concept of backtracking to a previous node and beginning the search in a new sub-tree, until and unless the search element is found and it does not use the concept of parallelism to search the search space parallel by the multiple processors. The solution space in DFS that maintains the list of traversed nodes may contain repeated nodes if the routes have been revisited. . The space and run-time complexity grows at the rate of O (N) for each in case of explicit graphs. The problems that occur in the simple DFS can be solved by the other algorithms that can efficiently solve the same problem. The performance of the algorithms can be improved by combining the features of different algorithms. The research carried out in this paper on various graphs and their run-time complexities reveals that backtracking is mostly responsible for degrading the performance of DFS,

REFERENCES

[1] Jon Freeman, Parallel Algorithms for Depth First Search, University of Pennsylvania,1991,Technical Report.

[2] V.Nageshwara Rao, Vipin Kumar, Parallel Depth First Search, part1 implementation, Department Computer Sciences, University of Texas, Austin,Texas 78712

[3] Mark E. Stickel and W. Mabry Tyson ,An Analysis of Consecutively Bounded Depth - First Search with Application in Automated deduction , Artificial Intelligence Center SRI International ,pp. 1074-1075.

[4] Richard E. Korf , Depth-First Iterative-Deepening: An Optimal Admissible Tree Search, Department of Computer Science, Columbia University,1985.

[5] Lopamudra Nayak, Design and Analysis of a Reverse Hierarchical Graph Search Algorithm, International Journal of Advanced Research in Computer Science, Volume 2, No. 4, July-August 2011.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 3rd Edition, Cambridge, MA: MIT Press, September 2009 .

[7] T.H.Lai and Sartaj Sahni. Anomalies in parallel branch and bound algorithms. In proceedings of International Conference on Parallel Processing, pages 183-190,1983.

[8] R.E.Korf.Depth-first iterative-deepening:An optimal admissible tree search.Artificial Intelligence,27:97-109

[9] J. Parallel algorithms for depth-first searches I. planar graphs. SIAM Journal on Computing, 15(3):814-830, August 1986.,1985ustin R. Smith.