Graph Searching Algorithms

Amal Byju
Department of Computer Science and Engineering,
National Institute of Technology Karnataka Surathkal,
Mangalore, India
16co205.amal@nitk.edu.in

Edwin Thomas
Department of Computer Science and Engineering,
National Institute of Technology Karnataka Surathkal,
Mangalore,India
16co218.edwin@nitk.edu.in

***Abstract*-This document provides an insight into the various graph searching algorithms as a repercussion of systematic literature review of conventional and trending graph searching algorithms. Through this paper we intend to highlight the significance of machine learning as a useful tool that can be incorporated in various graph searching algorithms that can reduce its complexity. We classify the graph searching techniques as subsets or modifications of two major conventional graph searching algorithms namely BFS(Breadth First Search) and DFS(Depth First Search). It is confounding that only few research papers explore the application of machine learning to the aforementioned graph searching algorithms. Hence, it is evident that there exists scope for future research on this topic and we aim to suggest directions for the same.**

Keywords--DFS(DepthFirstSearch),BFS(Breadth First Search) ,RHS(Reverse Heriarchical Searching Algorithm),IDDFS(Iterative Deepening Depth First Search algorithm).

1.Introduction

Graphs in the field of computer science are considered as mathematical models defined as a set of vertices and edges. It is a data structure that consists of finite set of ordered pairs, called edges or arcs that connects entities known as nodes.

A graph searching algorithm focuses on a series of steps to be followed in finding an item with the given properties from a group/collection of other items. There exists many graph searching algorithms in today's world of computer science each with its own advantages and disadvantages in a particular context. The applications of the several graph searching algorithms are tremendous which includes GPS Navigation systems, Computer networks,Webcrawlers, Social Networking websites , peer to peer networking etc.
Taking into account this wide umbrella of application domain of the graph searching algorithms it becomes imperative that better and more efficient methodologies to reduce its complexity must be developed as time progresses.
We have found out few research papers that elucidate the various trending graph searching algorithms with specified scope of improvement. In this paper we present a literature review of the analysis of the Depth First Search algorithm and its modifications, and Breadth first search and its modifications. Moreover, we propose the novel idea of Logistic Regression as a useful tool that reduces the graph searching algorithm complexities.
The remainder of the paper is organized as follows: Section 2 provides summarization of various DFS based graph searching algorithms, Section 3 provides summarization of various BFS based graph searching algorithms, Sections 4 to 13 discuss some notable graph searching algorithms and their advantages and disadvantages in specified contexts, Section 14 present the idea of logistic regression to reduce the complexity of graph searching algorithms, and finally section 15 summarizes the above mentioned algorithms, the conclusions drawn from those algorithms and scope for future work.

II. DFS(DEPTH FIRST SEARCH ALGORITHMS)

A. Conventional Depth First Search Algorithm

This section deals with the conventional Depth First Search Algorithm and elaborates on the pseudo code and the asymptotic analysis of the algorithm followed by the version of the code that displays all the paths from source to destination. The following text is a summarization of paper[2],[3].
The conventional depth first search algorithm is a recursive graph traversing mechanism based on backtracking. It deals with an exhaustive search of all the nodes of a graph by going ahead.
If in the process of moving forward no more nodes are found out then it backtracks to the previous calling node to continue the process of all its branches recursively. This process continues until all the nodes of a particular graph has been visited or marked as visited using a tri-colour mechanism.
The nodes that are already visited must be marked or else this may lead to fatal error of infinite looping. The recursive implementation makes use of the program stack of recursion to keep track of immediate previous node whose branch is being cross-checked.

Complexity Analysis:
If the graph is implemented using adjacency lists, wherein each node maintains a list of all its adjacent edges, then, for each node, one could discover all its neighbours by traversing its adjacency list just once in liner time.
Here there arises two different cases:

1)For a directed graph, the sum of the sizes of the adjacency lists of all the nodes is E (total number of edges). So, the complexity of DFS is O(V) + O(E) = O (V + E).
2)For an undirected graph, each edge will appear twice. Once in the adjacency list of either end of the edge. So, the overall complexity will be O(V) + O (2E) ~ O (V + E).

But the propose conventional DFS algorithm is implemented to print all the nodes in a particular graph G by an exhaustive search. However, the same algorithm poses an exponential complexity for printing all paths of a given graph.

Complexity Analysis:

The above algorithm in worst case has a complexity of O((V-2)!) where V is the number of vertices in the graph.

The factor of two in the above asymptotic analysis have arrised from the fact that the we exclude the source and the destination nodes.

As observed the complexity is exponential as

$(V-2)!=O(V^V)$

Hence, and improvement in the above search algorithm to print all the paths is imperative and is discussed in section 4 using logistic regression to reduce the complexity to polynomial time.

B. Iterative Deepening Depth First Search Algorithm

This section deals with an algorithm that implements and improvement to the Depth First algorithm in terms of space complexity and also combines notable features of BFS(Breadth First Search Algorithm) and DFS(Depth First Search Algorithm) referenced from paper[3].

There exists a variation of the conventional DFS algorithm known as the Iterative Deepening Depth First Algorithm. The core idea behind this algorithm is that the depth first search is applied repeatedly by incrementing the depth successively after each iteration.

The logical steps followed by this graph searching algorithm is as follows:

Step1: For the first iteration the depth is 0. Frontier is initialized as the source node. We remove that node from the frontier and this node is not the goal node and is also is at maximum depth bound. So we don't expand source node and move to the next iteration,

Step2: For the second iteration we consider the depth bound 1 and visit the nodes connected to the source node of the directed graph which is considered to be at depth 1. Now we start can expand the start node as it is not the goal and also is not at maximum depth (now depth is at one). So we find all the nodes from left to the right at same depth and repeat Step1 for each of those nodes.

Note: Here when we consider the frontier from left to right the nodes from left to right are taken one by one and each node is expanded first up to maximum depth rather than moving to the next node at the frontier.

Time complexity Analysis:

The IDDFS algorithm combines the benefits of BFS(Breadth first Algorithm) in terms of access speed and the DFS(Depth First Search) algorithm in terms of efficiency of space.

Suppose the number of children in each node also known as the branch factor is equal to b and its at depth d then there are b^d nodes in total.

In this algorithm the nodes at the bottom most level are expanded once , the nodes on the next level are expanded twice and the pattern continues such that the nodes at the i^{th} level are expanded i times and the nodes at the bottom most level are expanded depth+1(d+1) times.

Hence its concluded according to the research paper that the total number of expansion are calculated as follows:

$(d+1) + db + (d-1)b^2 + \ldots\ldots + 3b^{(d-2)} + 2b^{(d-2)} + b^d$

$= \sum[0 <= i <= d] (d+1-i)*b^i = O(b^d)$

Hence, the IDDFS algorithm takes the same time complexity as the Depth First Search algorithm but serves a better space complexity of O(bd) as compared to the conventional DFS algorithm.

C .Reverse Hierarchical Search Algorithm

This section deals presents the idea of RHS known as Reverse Heriarchical Search algorithm proposed in the paper[4] as an improvement on the conventional Depth First Search Algorithm. The paper claims that the proposed algorithm has a better space and time complexity with respect to the tree data structures and can also be extended to graph searching algorithms.

The RHS algorithm proposed in the paper focuses on reducing the amount of time complexity with regards to the load on backtracking mechanism of the DFS/conventional Depth First Search Algorithm. Considering the visualization of a graph as consisting of a root at depth 0 and its children at depth 1 and so on until the terminating nodes, it is correct to say that the number of possible combinations to be checked from root node to the terminating children node will be $O(2^n - 1)$. Hence the RHS algorithm was proposed which suggests search from the child node all up to the root node. The advantage of the proposed algorithm is that the volume of search decreases to 2^{n-1} elements accounting for the elements in the nth level. Moreover, it eliminates the need of backtracking by limiting the depth of the search form individual nodes to the root.

Complexity Analysis:

For a given input number of nodes of size N, the complexity is analysed as:

$\log(N)*\log(N) + \log(N) = \log(N^2) + \log(N)$

Considering the fact that asymptotic analysis is being used as a technique of analysis the constant factor of 3 can be neglected in the above equation and hence the equation simplifies to log(N).

Hence its concluded that the above algorithm runs in a logarithmic time which is considered superior compared to the conventional DFS algorithm and the Iterative Deepening Depth First Algorithm(IDDFS) that runs in linear time.

III.BFS(Breadth First Search Algorithms)

A. Conventional Breadth First Search Algorithm:

This section discusses the conventional BFS algorithm and analyses its complexity referenced in paper[7]. Moreover the variation of the conventional BFS algorithm to print all the paths from a given source to destination node is discussed and complexity analyzed.

Breadth first search is a graph searching algorithm where nodes are explored "by expanding the frontier". It checks all the nodes that are d edges away from the source vertex before moving on to a node that is d+1 nodes away from the source vertex. Frontier refers to all the nodes that are d nodes away from the source vertex.

BFS achieves this by enqueueing the adjacent nodes of the source vertex into a queue, say q. Then it marks the source vertex as "visited". The algorithm then dequeues an element from the queue as long as it is not empty. This element is marked as "visited" and then the neighboring vertices of this vertex that are not marked as "visited" are enqueued. This process is carried out repeatedly until the algorithm traverses the entire graph.

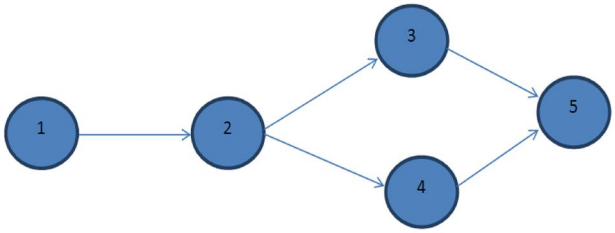
BFS can be modified to find the shortest path in a graph. Another variation of BFS prints all the possible paths between two vertices in a graph. Faster and memory efficient variations of BFS have sprung up such as parallel BFS, direction optimised BFS and so on.

Complexity Analysis:

The time complexity of Breadth First Search is O(N+E), where N is number of nodes and E is number of edges. The algorithm also requires information about the number of nodes in the graph. Also, to mark nodes as “visited”, an appropriate data structure must be used. The space complexity is O(|V|), where |V| is the number of elements in the set of vertices.

Breadth First Search algorithm to print all the paths from a given source to destination:

Given a graph G = (V,E) and a source vertex s, breadth first search can be modified to print all paths to a destination vertex



Here, the conventional breadth first search algorithm is modified to generate the new algorithm. This algorithm works for both directed and undirected graphs.

G is a two dimensional array where each row is a queue and corresponds to a particular node in the graph. Each queue is filled with the numbers of adjacent nodes to the node whose number is the row index where the queue is placed. Let path be a one dimensional queue to store the current path. Let q be a two dimensional queue which stores all the paths.

Complexity Analysis:

Finding the time complexity of this problem is a NP complete problem. The maximum limit on the number of paths that can exist between two nodes in a graph is (V-2)!.

B. Direction optimized Breadth First Search algorithm:

This section discusses idea proposed in paper [8]. The conventional breadth first search algorithm scans the graph by "expanding the frontier" and performs the same number of operations as its worst case, every single time. In direction optimised BFS, however, the approach is different. Here, the direction in which the nodes are scanned (from the bottom to top or from top to bottom) depend on the current status of the search. For instance, suppose y nodes are separated from the source node by d edges and y' nodes is separated from the source node by (d-1) edges. Suppose that one or more nodes in the set y' are connected to the destination node. The conventional DFS algorithm does not take advantage of this situation and performs (y'-1) extra checks. However, the bottom up algorithm searches the nodes from bottom to top reducing the overall number of operations.

In the conventional BFS, as the frontier grows, more and more nodes are visited. At each new node arrived, the neighboring nodes that are not marked as visited are stored separately. The next node to be visited is a node chosen from this storage location. It can be agreed upon that conventional BFS always creates paths from source to destination.

In direction optimised BFS, paths can be drawn from source to destination as well as from destination to source. These two processes can occur concurrently during the execution of the algorithm. When a path is traversed from destination to source, the neighboring vertices of the destination vertex are stored. Then the algorithm checks whether any of the neighboring nodes are part of the frontier or not. If a node is part of the frontier, the check for examinations of the neighboring nodes can be stopped. If such a node is not found this operation cn be carried out repeatedly until a node which is part of the frontier is found. To increase the performance of BFS, an approach that is used is to run the conventional BFS till the current node is somewhat midway between the surce node and the destination node. Here, the frontier normally will be at its largest. Using the bottom up BFS at this point will reduce the total number of operations performed. Nowadays, computers are multicore systems. New variants of BFS have evolve to work more efficiently on these systems.

C. Boolean Expression Based Algorithm to print all paths:

This section discusses the Boolean expression algorithm as described in paper[6].We use operators such as ‘.’ (logical AND) and ‘+’ (logical OR) to represent graphs as Boolean equations. Now, observe the graph below :

Case 1:

From the perspective of the root node, the children nodes are associated with it by the OR relation.

Case 2:

From the perspective of the children node, its parent is associated with it by the AND relation.

An example of case 1 would be the relation of node '2' to children nodes '3' and '4'. From the viewpoint of '2', nodes '3' and '4' are in 'OR' relation. As in case 2, from the viewpoint of node '3' or '4', node '2' is related to them by the relation 'AND'. Using these basic relations, the Boolean representation of the graph above is :
G = (1.(2.((3.5)+(4.5))))

Complexity Analysis of the above algorithm:
The worst case complexity of the above pseudocode is $O(b^d)$ where branching factor is b and d is the depth of the traversal. The variables in the Boolean expression can take 2 values: 0 or 1. After the variables in the boolean expression of a graph are replaced by their boolean values, the result of the expression specifies whether the combination of boolean values builds a simple path in the graph.
For example, if combination (1,0,1,0,1) is used in the Boolean expression, the result obtained is FALSE, indicating that this combination does not yield a path in the graph. On the other hand, if the combination (1,1,0,1,1) is used, then the result is TRUE which indicates a possible path from 1 to 5.The algorithm for finding all paths between two nodes of a graph is given below:

'Start' is the first variable of a Boolean expression and 'Goal' is the destination.

```
if Start = Goal then
        Print path
else
        Count substrings with AND relation
        If Count >= 1 then
                Do for each substring
                        FindAllPath (substring)
                end for
        end if
end if
```

So, far the different algorithms for graphs searching as mentioned in several research papers have been analyzed and presented. The next section deals with the proposed novel idea.

IV CWRS algorithm.

This section discusses the CWRS algorithm for workflow graphs verification. This algorithm offers an advantage over other graph searching algorithms in terms of combining the benefit of reduced graph searching complexity and considerable amount of graph reduction.
In the CWRS algorithm is particularly used in domains that requires to find loops in the graph , to reduce existing loops and also verify claimed graphs in a graph.
The algorithm uses a conventional DFS graph algorithm and a stack structure to push current node into the stack. If the same node is seen at some point in the graph , a loop is detected and claimed. In case the AND split nodes does not match the merge nodes then the algorithm raises an error. The loop verification segment of the algorithm cleverly takes care of nested node pairs and handles them within considerably less asymptotic time period.
Moreover, the algorithm also has a reduce loop segment that takes into consideration the OR split nodes and produces a new workflow graph and recursively applies the reduction segment. This help to convert the graph to an acyclic graph.

The complexity of the CWRS algorithm for workflow graph verification is $O(E^2)$. This asymptotic computational complexity displays great potential and performance as compared to other conventional graph algorithms like the BFS(Breadth First Search).

V Genetic Algorithm based Heuristic Search on Graphs.

This section discusses the very intuitive analogy for relating the Genetic algorithm to implement an efficient graph search. The core concept behind the idea is to use the LaPaugh's Lemma which describes the property of monotonicity in graph search algorithms. It uses the idea that the edges in the graph can be represented as permutations of edges.
The analogy used in the aforementioned presented novel idea is to consider the domain on the graph that is set of edges E and set of vertices E as the intruded location by intruders. The edges of the graph are known to be contaminated if the intruders stay plunged into these edges and the edges are known to clean when the prescribed counter agents traverse the corresponding edge and claims to remove the contamination.
For undirected graphs with the existence of cycles there exists a scenario of looping. To represent this an analogy is given. If the vertices of the graph between which a loop exists are names as m and n , then if an intruder occupies any one of the edges and is considerably fast , the only way to catch the intruder is to have two agents. The first agent stays in m and waits for the intruder who might come from n to m and back through the next edge. The next agent takes a walk from m to n to clean the intruder and to prevent recontamination of the loop causing edge. So, the task represented by the above algorithm is to find the maximum number of agents to clean the edges in the graph.
The advantage of the above proposed algorithm is that conventional genetic algorithms could only be applied on optimal path detection and edge detection strategies and not easily on an exhaustive graph search, whereas this algorithm provided an optimal sub structure to cover all the edges of the graph using genetic operators using crossover operator for two or more parent solutions.
The paper provides a exchange method proof of optimal solution construction to find the best sequence for agent number and positioning to implement graph search. The complexity of the above algorithm O(|E|), which is lesser than search algorithm mentioned in sections 1 and 2 but fails in

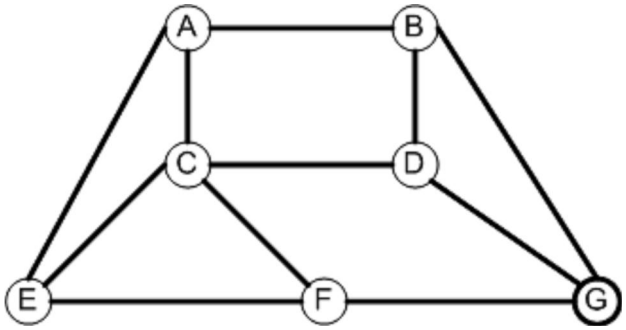
supremacy as compared to the Reverse hierarchical graph search algorithm.

V1.Heuristic Path Algorithm.

This section discusses the Heuristic Path Algorithm also abbreviated as HPA referenced from paper [10],[11].An efficient algorithm can apply heuristics to efficiently find the paths between two nodes in a graph. From the start node, a function, say H(x) can be applied to each and every node along the path which gives an estimate of the distance between the current node and the destination. This algorithm is used in directed graphs only. Here, we map each node X to its power set, i.e, P(X). The size of the power set is always finite that is 2^x. Let the mapping for each node be represented as gamma(x). Let S represent the nodes already visited and expanded. We incrementally add more nodes to S as we traverse the graph. We place the start node in S and calculate gamma(S) which we append to a new set S'. Let g be a function that enumerates the number of edges from the start node to the destination node. If the current node is present in gamma(start node), then g(current node) = 1. Then we calculate the heuristic at the current node (one step away from the start node). We choose that node in S' such that the heuristic is minimum. We repeat this process over and over until we reach the destination node.

VII.Optimized Algorithm for Graph Traversal using Adjacency Matrix

Here, the graph is represented as an adjacency matrix. Another array called the sum matrix is also maintained which stores the sum of each row in the adjacency matrix. Another array called visited array whose length is the number of nodes in the graph is also maintained. Initially, all nodes in the visited array are set to NULL.



We start with node A. Now, the adjacent nodes to A are B, C and E. Going by the row corresponding to row A in the adjacency matrix, all the elements in the columns B, C and E are set to 0. Now, the vertices A, B, C and E are appended to the visited matrix. We update the sum array after this. Now we check the vertex corresponding to the maximum value in the sum array. We repeat the previously mentioned steps until the adjacency matrix becomes a zero matrix and the visited array is full. The time complexity of this algorithm is O(|V|+|E|) and the space complexity is $O(|V|^2)$.

IX .Branch and Bound search algorithm.

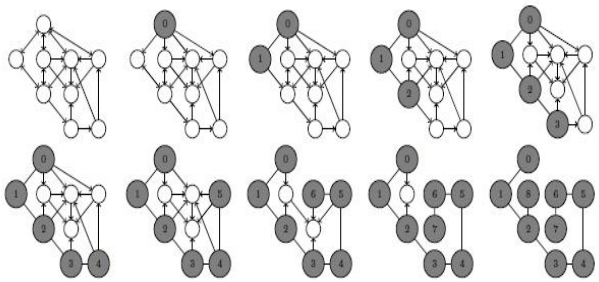
This section discusses the Branch and Bound Algorithm referenced from paper [13],[14]. The power of branch and bound algorithm can be added to depth first search algorithm. Here, a path is found from the start node to the destination node and is set as the bound. Now if another path is traversed from the start node to the destination node and the bound is exceeded, (we know this when at a particular node, C(current node) + H(current node) > bound, where C(node) stands for the cost of node and H(node) stands for the heuristic function at that particular node) then the path is pruned or purged. A new path is considered. Branch and bound is often used with heuristic search to minimize the elapsed time. Initially, the bound is set to infinity. The algorithm returns an empty set and bound as infinity if no paths are found.

X. British Museum Algorithm.

This section discusses the British Museum Algorithm for searching graphs which is supreme over the conventional DFS and BFS search algorithm mentioned in section 2 and 3.However,this is not an optimal search algorithm. The algorithm starts from the start node, say s. Let the destination node be d. Two arrays, a visited array and path array are maintained. Path array store the current path. Visited array has a length that is the number of nodes and all the nodes have a status of false in the visited array. In DFS and BFS, the nodes were traversed in a particular order; that is in DFS, the graph was traversed such that each branch is traversed till the end before backtracking, whereas in BFS, all the neighboring nodes of the current node are visited before continuing the search to the next level. However, in British museum search, there is no such rule. More and more nodes are visited as a path is created and the values of these nodes in the visited array are set to true. If the current node is the destination node, then the path is added to the existing set of paths found for the problem.

XI Parallel DFS by elimination of backward arcs.

This section discusses parallel DFS algorithm for graph search by elimination of backward arcs referenced from papers [12],[14],[15]. Conventional depth first searching algorithms use outgoing paths (forward paths) to reach neighboring nodes and finally reach the destination. However, this algorithm pays fine attention to the incoming arcs to a node and eliminates those arcs (in parallel) which are no longer relevant for search. Suppose DFS algorithm is executed from a starting vertex a. The vertices reachable from a are traversed in DFS manner and are numbered simultaneously. Each hop to a new vertex removes all incoming edges to that vertex and then recursively, DFS numbers the subtree from the first non-eliminated arc out of the starting vertex.



XII. Parallel Breadth First Search by elimination of backward arcs.

This section describes the advantages of Parallel Breadth First Search algorithm over conventional Breadth First Search.
This algorithm follows the same idea of the conventional BFS algorithm, however it performs parallel elimination of unvisited nodes which are appended to the queue for the next level. For each vertex of a particular level in the queue, the non-eliminated arcs are considered in order that is determined by how the graph is represented. Now, when the next level is traversed, the incoming arcs to the unvisited vertices are again discarded. There exist alternate approaches to the parallel BFS problem like, for example, repeatedly squaring the adjacency matrix representing the graph during the min-plus semiring of the element wise operations. Such an algorithm computes the result in O(NlogN) time but however requires $O(N^3)$ processors, which is not practical.
Such algorithms have an application in computer systems where each processor has its own private memory. The complexity of parallel BFS algorithms is different by a constant factor to the conventional BFS algorithm. This constant factor measures the efficiency of such an algorithm.

XIII Beam Search Algorithm.

This section discusses the Beam Search Algorithm for searching directed and undirected graphs from paper [15] and [11]. Conventional algorithms like DFS, BFS and A* cannot be used on large search spaces. Beam search focuses on generating paths starting with the optimal solution (path with minimum number of nodes). Here, only the most promising b nodes are retained for further branching instead of all the nodes. Therefore, beam search focuses on providing the optimal or close to optimal solution and consumes less memory space compared to best first search. Let vertex_set be a set containing only one vertex, the source node. We repeatedly choose the best node from this set until we reach the destination. After choosing the best node, we add the neighboring vertices of the best node to vertex_set. Meanwhile, the parent node should be recorded. On reaching the destination, we backtrack back and record the path taken. If |vertex_set| > b, we select the best b nodes and remove the others from vertex_set.
However, it is possible for the algorithm to miss the goal node entirely even though there exists a path from the start node to the goal node. Therefore, the heuristic function should be accurate and the beam should be large enough that improve the beam search's chances of finding the goal.

XIV. Logistic Regression- A tool to decrease complexity of DFS algorithm.

This section of the paper proposes a novel idea/methodology to reduce the complexity of conventional graph searching algorithms such as DFS algorithm using a Machine Learning tool known as Logistic Regression used in classifier problems. The proposed idea is explained with respect to a case study on the Trip Advising Software.

A. Brief Description of the case study:
The software project (Trip Advising Software) aims at providing a flight booking interface based on optimized filtering options such as cost and other parameters.
This software uses graphs to search the available planes that could be taken corresponding to different flights of different registered airline companies. The conventional graph searching algorithm DFS(Depth First Search) gives an exponential search complexity for searching and displaying all the possible combinations of flights (at max three connection flights) from source to destination node entered by the user.
Hence it becomes imperative that a more efficient searching algorithm must replace the conventional DFS in this software scenario to produce faster and more accurate results catering to the convenience of the system end user.
A novel approach to the aforementioned problem will be to use a technique known as logistic regression used in classifier problems to reduce the complexity to polynomial time.

Logistic Regression:

It is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured using a dichotomous variable in which there are only two possible outcomes. The outcomes one and zero of the dichotomous variable in the above case study is represented as follows:
Let z be the computed theta equation:

if z == 1:
Then a possible flight route for the given parameters exists.
if z==0:
Then a possible flight route for the given source to destination node doesn't exist.

We define the theta function as follows:
$X0\theta0 + X1.\theta1 + X2.\theta2 + \ldots\ldots + Xn.\theta n + 1.\theta n+1 = z$

X_i: It is the mapping value corresponding to the ith plane among all the planes of all the plane companies.

θi: This theta value is a characteristic of the logistic regression algorithm and it indicates the weights given to each of these possible mapping values to indicate the probability of a plane(i) 's presence or absence at a particular node corresponding to the mapping table.
The mapping table described earlier is defined as follows:
Node A = 10
Node B = 11
Node C = 12
Node D = 13
……
……..
Node x = y

Here, the values written to the right of nodes A,B,C… is the numeric value given for any plane indicating its presence at that particular node or (station in reality) after the previous plane has landed into that particular node of the graph. In addition to the above values we present a inverse magnitude mapping value for each node indicating that the particular plane being considered has started from that particular node in the graph as journey from the source to destination.

Now we present the steps to be followed to obtain the theta equation using logistic regression:

Step1:
Run the DFS search algorithm from all possible source nodes to all the possible destination nodes to fill the data set matrix required by the logistic regression algorithm. The first parameter(x0) would be an equivalent unique ASCII mapping of [source, destination] nodes as numbers.
The parameters following that would be(xi) where 1<=i<=n, where n is number of aeroplanes. The (xi) parameter takes in the unique mapping values of the plane(i) that contains information of the location of the plane at any instant of time with respect to the node corresponding to the table mentioned above. The last parameter(xn+1) is used as a constant bias value = 1.
Step 2:
The DFS also fills the correct output matrix y based on the possibility of taking flights (xi) ->(xj) where 1<=i<=n-2 and n-i+1<=j<=n-1, in a sequence containing a maximum of three connection flights. If the sequence found out is a valid combination to reach from source to destination, the graph searching algorithm assigns the output matrix y corresponding to the current mapping values of the planes (pi) to 1 else 0.

Step3:
After we obtains the dataset matrix, the output matrix then the next step is to use the proposed Cost Function ,
$J(\theta) = (1/m)*\sum[1<=i<=m]*Cost(h_\theta(x^{(i)}),y^{(i)})$

Where the Cost function is defined as follows :
if y == 1:
$Cost(h_\theta(x^{(i)}),y^{(i)}) = -\log(h_\theta(x^{(i)}))$
if y==0:
$Cost(h_\theta(x^{(i)}),y^{(i)}) = -\log(1-h_\theta(x^{(i)}))$

The above split function can be summarized by one equation given as the follows:

$J(\theta) = (-1/m)*\sum[1<=i<=m]*(y^{(i)}*\log(h_\theta(x^{(i)})) +(1-y^{(i)})* \log(1-h_\theta(x^{(i)}))$

Step4:
Now since we have found out the cost function we use gradient descent algorithm to find out the optimized θ matrix values to satisfy the given dataset and also the output matrix y. For greater efficiency we present a vectorised equation for the implementation of the gradient descent algorithm until convergence is reached:

$\theta := \theta – (alpha/m)*X'(g(X*\theta) – y)$

X: This is the matrix storing the dataset.
y: This is vector storing the outputs of the combination of dataset.

Step 5:
Once the theta equation is obtained by training the theta matrix with data samples the software will be set to find out the possible paths from entered source to destination by just plugging in X(i) values to the equation and crosschecking the z values to see if the path exists.
Complexity Analysis:
Since we consider a maximum of three connection flights , we have to choose 3 planes from the source to destination that takes a complexity of ${}^nC_2$ and this is bounded by $O(n^2)$ where n stands for the number of planes registered to the software. Thus this methodology serves as supreme when compared to the conventional DFS algorithm as DFS takes exponential time to search and print all the paths. Moreover, it avoids the backtracking mechanism load on the software as it involves simple plug in of values into the equation to find all valid routes from the given source to destination.

XV. Conclusion

This paper attempt to do systematic literature review on various graph searching algorithm among which DFS and BFS based algorithms are discussed in details by capturing essential details from the various reference papers mentioned under the references sub-section. Moreover, the novel idea of logistic regression to reduce complexity of graph search to print all the paths is discussed which reduces the complexity from exponential to polynomial is discussed.

REFERENCES

[1] Rakesh Agrawal and H. V. Jagadish "*Algorithms for searching Massive Graphs*",IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 6, NO. 2, APRIL 1994.

[2] Akanmu T. A., Olabiyisi S. O., Omidiora E. O. ,Oyeleye C. A., Mabayoje M.A. and Babatunde A. O. *"Comparative Study of*

Complexity of Breadth-First Search and Depth First Search Algorithms using Software Complexity Measures", Proceedings of the World Congress on Engineering 2010 Vol I WCE 2010, June 30 - July 2, 2010, London, U.K.

[3] Robert Tarjan "Depth-First Search and Linear Graph Algorithms",Computer Science Department Stanford University Stanford, California.

[4] Deepak Garg Megha Tyagi, *"Comparitive Analysis of Dynamic Graph Techniques and Data Structures"*, International Journal of Computer Applications (0975 – 8887) Volume 45– No.5, May 2012.

[5] Rua Shi*, "Searching Algorithms Implementation and Comparison of Eight –puzzle problem"*, 20 II International Conference on Computer Science and Network Technology.

[6] Sankhadeep Chatterjee,Debarshi Banerjee "A Novel Boolean Expression based Algorithm to find all possible Simple Paths between two nodes of a Graph" , International Journal of Advanced Research in Computer Science Volume 5, No. 7, September-October 2014

[7] S. Beamer, K. Asanovic, and D. Patterson, "Direction-Optimizing Breadth-First Search," in Proceedings of the 2012 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC12). Salt Lake City, UT, United states: IEEE, 2012, pp. 1–10.

[8] Ankit Malhotra,Dipit Malhotra,Saksham Kashyap "Optimized Proposed Algorithm for Graph Traversal" , International Journal of Computer Applications (0975 – 8887),Volume 104 – No.9, October 2014

[9] Tianrui Li, Luole Qi, Da Ruan. *"An efficient Algorithm for the Single-Source Shortest Path Problem in Graph Theory."* Proceedings of 2008 3rd International Conference on Intelligent System and Knowledge Engineering

[10] Ratan K. Ghosh, G.P. Bhattacharjee "Parallel breadth-first search algorithms for trees and graphs". International Journal of Computer Mathematics, Volume 15, 1984 - Issue 1-4

[11] A Reinefeld and V Schnecke "WorkLoad Balancing in Highly Parallel Depth First Search". Proc Scalable High Performance Computing Conf SHPCC IEEE Comp. Sc. Press (1994), 773-780.

[12] Bernard Mansac Thierry Mautor Catherine Roucairol, "A parallel depth first search branch and bound algorithm for the quadratic assignment problem"

[13] Zhong-Weixu, Fengliu,Ying-Xinli "THE RESEARCH ON ACCURACY OPTIMIZATION OF BEAM SEARCH ALGORITHM". Computer-Aided Industrial Design and Conceptual Design, 2006. CAIDCD '06. 7th International Conference.

[14] L.Fu, D. Sun, L.R. Rilett "Heuristic shortest path algorithms for transportation applications: State of the art". Computers & Operations Research 33 (2006) 3324–3343, Elsevier.

[15] Lukasz Wrona,Bartosz Jaworski, "Application of Genetic Algorithms in Graph", Proceedings of the 2nd International Conference on Information Technology, ICIT 2010 • 28-30 June 2010, Gdansk, Poland.

[16] Kia Lu, Qiang Liu. "An Algorithm Combining Graph-Reduction And Graph-Search For Workflow Graphs Verification.Proceedings of the 2007 11th International Conference on Computer Supported Cooperative Work in Design.

[17] Scott Beamer, Krste Asanović, and David A. Patterson. Direction-optimizing breadth-first search. In International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2012.

[18] Scott Beamer, Aydın Buluç, Krste Asanović, and David Patterson. Distributed memory breadth-first search revisited: Enabling bottom-up search. In Workshop on Multithreaded Architectures and Applications (MTAAP), in conjunction with IPDPS. IEEE Computer Society, 2013.