International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 11, November 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

A Novel Path Enumeration Algorithm for Directed Acyclic Graphs

Sushobhit Singh[1], Ajay Kumar Saxena[2]

[1,2]*Department of Electrical Engineering, Dayalbagh Educational Institute,(India)*

ABSTRACT

Path enumeration is well studied and applied operation in all classes of graph problems. In directed acyclic graphs, traversal methods are typically used for enumerating paths. In this work, we have explored a novel approach to path enumeration for directed acyclic graphs, which is particularly useful to converge quickly for repeated path enumeration operations on the graph. We have defined a set of problem cases and compared our method with DFS based path enumeration. In our experiments we have found a dramatic impact of number of repeated operations on overall runtime.

Keyword: Depth First Search (DFS), Directed Acyclic Graph (DAG), From To Path (FTP), Path Enumeration, Through Path (ThP)

I. INTRODUCTION

Graph, in computer science is an abstract data type, which is used for representing the mathematical graph concept. A graph can be classified as directed or undirected, based upon the possibility of accessing the adjacent vertices from a given vertex in the graph. In directed graphs the vertices are connected with the edges which have direction associated with them, which mean that all the adjacent vertices of a given vertex are not accessible with the same cost. Directed acyclic graphs or (DAG) is an important class of graph structures which, is a finite directed graph with "no directed cycles". A DAG contains no vertex, v that can be reached to itself through a finite set of directed edges E [1]. Some of the important definitions, related to DAGs which we will be using throughout this paper are presented below:

1. Path – A path is a finite set of directed edges. Set of all paths is denoted by P.
2. Start Point – A start point is a vertex v, in the DAG which does not have any incoming edges to it. S is the set of all start points in the graph.
3. End Point – An end point is a vertex v, in the DAG which does not have any outgoing edges from it. E is a set of all end points in the graph.

In a lot of graph applications [2], [3], [4] and [5], it is required to find out all paths between a set of vertices, this all path finding problem can be generalized as, enumerating all the paths from a set of start points to a set of end points, where the start and end points of interest are known and is a sub-set of all start and end points of the graph.

In this work we have particularly focused on two sub-sets of path enumeration problem, which we call as FTP (from_to_paths) and ThP (through_paths) algorithms. These problems are formally defined below:

1. From_To_Path_Enumeration (FTP):

International Journal of Advance Research in Science and Engineering
Volume No.06, Issue No. 11, November 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

Given a graph G, find a set of all paths p ϵ P, between a given vertex pair (u, v) | u ϵ S, v ϵ E

2. Through_Path_Enumeration (ThP):

Given a graph G, find a set of paths p ϵ P, which contains the vertex t ϵ V, between each vertex pair (u, v) |

u ϵ S, v ϵ E.

Fig. 1 shows a sample DAG; we have shown inputs to both the FTP and ThP algorithms, and expected output for both problem types.

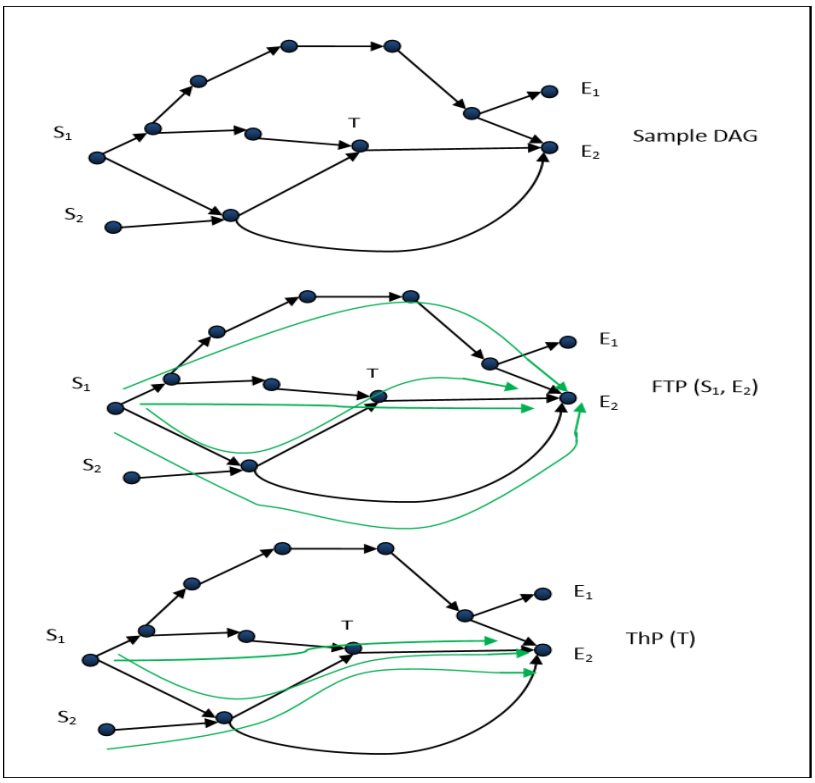

Fig.1 A sample Directed Acyclic Graph and paths resulting from FTP and ThP

Rest of the paper is organized as follows; we will describe our data structure and its related operations in section 2. Section 3 describes the path enumeration algorithms, both DFS based and point handle based. Section 4 describes the comparative experiments and results, and conclusive remarks are made.

II. POINT HANDLE

Point handle (PH), is a vertex bound structure, which has a unique handle per vertex and it keeps the track of splitting and merging of paths from vertices in a DAG. Fig. 2 describes a PH structure in detail; where we have enlisted the main structure elements and define the basic intent of each one of them.

1. The source vertex handle, keeps the track of source vertex from where the PH is originated.
2. Master PH, keeps the track of PH from which the current PH is generated, it ensures that in constant stime one can reach from a PH to its master PH, and can be reached to the origin point or a start point of the path.

3. Merged PH container contains a set of PHs which are merged into this PH and are getting propagated encapsulated in this PH.

In the next section we will describe operations to be performed on point handles; and the point handle propagation algorithm. We will further describe the state of the data structure after each of these operations. Before we delve into exploring individual operations let's describe a central property of the PH based algorithms that we will be presenting in later sections. A unique PH is created at each of the start points in the Graph and they are propagated on the paths originating from the vertices, but on each vertex of the graph we maintain only one PH explicitly. With this as the central property of the PH algorithms, let us now describe various operations and their intent for the algorithm.

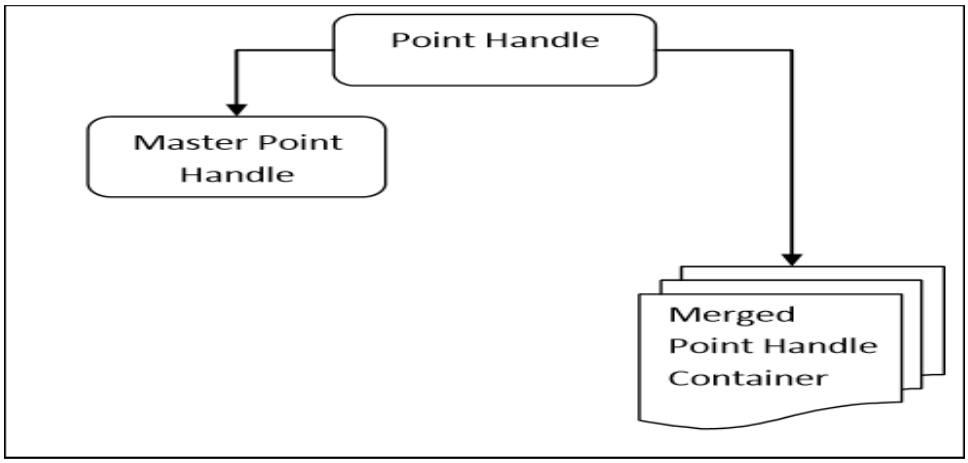

Fig.2 Point Handle Structure

2.1 Split Point Handle

Split operation on a PH comes into being when there is more than one vertex on the graph which contains an incoming edge from the give vertex. In such a case to ensure single vertex single PH property, point handles are effectively split, and a new PH is created at each of the sink vertices. Source vertex handle of each the newly created PHs points to the source vertex at which the split takes place. As a special case, if there is already a point handle present, PH merging takes place, we have described PH_Merging further.

OPERATION Split_PH:

Input: v ϵ V

Output: Set S of vertices

For each vertex u in the fan-out of v

If (PH[u] does not exist)

PH[u] = new PH

Master [PH[u]] = PH[v]

Add u to S

Else

Merge_PH (PH[u], PH[v]);

End.

2.2 Merge Point Handle

As a point handle propagates through a path there is a possibility that it comes across a situation where there is already a PH present on sink vertex of the path, in accordance with the central idea of PH structure, this PH cannot propagate any further, hence it is merged with the PH which is present on the sink vertex. The inputs to this operation will be two point handles to be merged; it inserts the source vertex point handle PH[u], into the merged PH container of sink vertex point handle PH[v].

OPERATION Merge_PH:
Input: PH[u], PH[v]
Output:
if PH[u] not uniquely equal to PH[v]
Merged_Container[PH[v]] += PH[u]

2.3 Test PH Merged

As a PH can exist on a node contained in the merged container for another PH, it is important to establish some method or operation to find out if a vertex contains a given PH. For this purpose, we have defined the operation called Test_Merged_PH, which will take two PH and check whether they are merged or not. The "is in merged container" is a costly operation, in that it involves searching for a PH in another. But, due to PH culmination at split and merge points, it's unlikely that a given PH will contain a big number of merged PHs.

OPERATION Test_Merged_PH:
Input: PH[u], PH[v]
Output: Status
Status ← false

if PH[u] equals PH[v] OR PH[u] exists in merged container of PH[v]
Status ← true

2.4 Propagate Point Handles

With the basic operations setup, let us now look at the tag propagation procedure, for graph sweep and establishing point handles at each of the vertices, which will then be used for path enumeration algorithms. For asserting point handles on each vertex in the design, path handles are propagated on the graph. At every start point a new path handle is created and propagation follows. Point handle propagation starts from a start point in the DAG, generally any point in the DAG can be used for tag propagation, but in this work we have the algorithm initiating from the start points. Propagation of the PHs on the downstream fan-out vertices, happens in breadth first manner.

We have maintained a queue of vertices which are to be explored further. There are couple of observations which we would like to draw attention to –

1. Point handles gets split into child point handles at the split point, and thus a point handle does not exist beyond the split point
2. A given point handle, or its children propagates only till the time it reaches a vertex which is already populated with a point handle, beyond which it is contained in the handle in which it is merged.

The aforementioned observations make the tag propagation exactly a O(V) algorithm, i.e. by a single sweep of all vertices of the graph we will be able to propagate the point handles on the graph, and this method maintains the previously stated invariant. Fig. 3 presents a DAG and its state after the point handle propagation. In this case we have given unique integer identifiers to point handles.

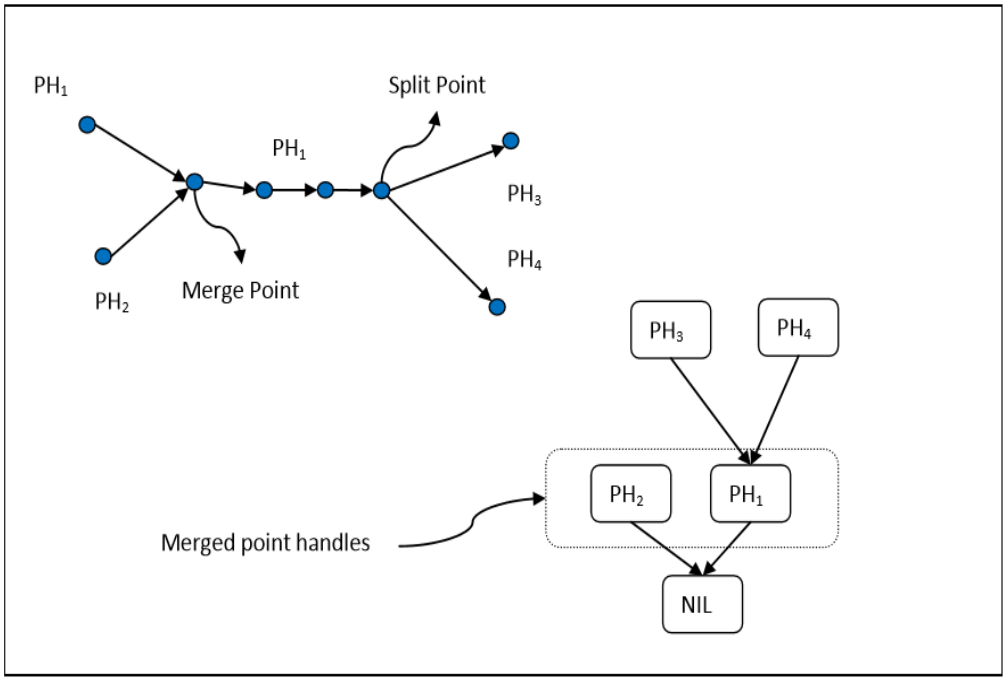

Fig.3 Merge and Split on the graph and point handle connectivity

III. PATH ENUMERATION ALGORITHM

With all the constituents established, we may now present the path enumeration algorithms. We will briefly touch upon the path enumeration using depth first search on the graph, which is a typical method used on DAGs. We will take special case of path enumeration which we are interested in this work, without the loss of generality. Next, we present the challenges faced by generic DAG path enumeration algorithms based on DFS in FTP and ThP enumeration problems.

3.1 DFS Based Path Enumeration on DAG

DFS is a standard graph algorithm, and we will not go into the details of it, any interested reader can go through [6] for understanding the DFS. There is however an important observation which we would like the readers to take a note of, given a vertex v in the DAG, all the paths out of the vertex can be easily enumerated, but the same cannot be said for the paths from incoming edges, because DAGs typically doesn't have linkages in the backward direction. Thus, DFS based solutions for path enumeration in case of FTP and ThP type of problems becomes impractical in terms of runtime. Several applications tweak the DAG to have incoming edges information also stored on all the vertices [7] to be able to go backward on the graph, which incurs serious space penalty, and is unnecessary for path enumeration. In this case, the DFS operation can be performed, after fan-in/fan-out cone coloring on the graph for secluding the section of graph on which prospective paths are present [8]. In the next section we will present our path enumeration algorithm which uses the point handles for performing path enumeration.

ALGORITHM Propagate_PH:

Input: v S | S is a set of start Points in Graph G

Output:

Create empty Queue Q

Q.enqueue (v)

while Q is not empty

```
    u = Q.dequeue ()
    if u E | E is a set of end points of graph
        Continue to start
        if (split needed at u)
            S Split_PH (u);
            for each s inS
                Q.enqueue (s)
            end
        else
            W = Next[u]
            if w has PH
                Merge_PH (PH[u] , PH[w])
            else
                Q.enqueue (w)
end
```

3.2 Point Handles Based Path Enumeration on DAG

Before we delve into the path enumeration we would like to point the readers to a residual structure we call as point handle graph. A point handle is a residual structure that is derived from the original DAG after the tag propagation as depicted in figure 3. This is a reduced graph which takes in account the merges and splits that have taken place on the path. There are a couple of points to be observed for point handle graph:

1. It is reversed DAG, with reduced points as compared to original graph, i.e. the start points of the point handle graph contain the end point tags of the original graph.
2. The merged point handles can be visualized as being on the same level or union, and any path converging from a point handle; will pass through all the point handles at the same level, shown in Fig. 3.

Algorithm Enumerate_PHPaths is used to get a set of PH paths, culminating at an end point, given an end point in the design, and algorithm Enumerate_DAGPath is used to enumerate a DAG given a PH path. These are the two major sub operations of the enumeration algorithms, which we will be presenting in the next section.

Enumerate PH paths does a point handle path enumeration, which is a reduced graph path enumeration, from which the original graph path enumeration is done. In PH path enumeration, DFS operation is performed on this is used to get all the paths culminating at an end point. We have used an operation Expand_PH which expands all the point handles which are merged in this PH or in which this PH is merged.

```
ALGORITHM Enumerate_PHPaths:
Inputs: Vertex Stack S, PH[v] | v ϵ V
Output: Set of End point handle paths
if PH[v] is non-existent

        Push PH[v] on Stack S
        while S is not empty

                PH ← POP(S)

                PH_S ← Expand_PH
                foreach PH ϵ PH_S
                        Enumerate_PHPaths(S, Master[PH])
                end
        end
else
        Add path in stack to point handle paths set and remove non-needed entries from stack
end
```

```
ALGORTIHM Enumerate_DAGPath:
Inputs: A given point handle path
Outputs: DAG path
PHT ← Remove_PH_Path_Top
v← Get PH Vertex [PHT] Add on path [v]
        while v is not a graph end point
                for each vertex u in the fan-out of v
                        if PH[u] is equal to PH_T
                                v← u
                                break the inner loop
                        else if Test_Merged_PH(PH[u], PHT)
                                PH_T ←    Remove_PH_Path_Top
                                v ← u
                                break the inner loop
                end
        end
```

3.2.1 Path Enumeration from FTP (From To Path)

In FTP path enumeration, as discussed earlier, the paths between a pair of start and end point on the graph are enumerated. Path handle based method for enumerating such paths in the design, gets all the end points and does a backward path enumeration. It has to then filter the paths which do not start from the points in from set.

3.2.2 Path Enumeration of ThP (Through Path)

Through path enumeration problem requires to enumerate all start point to end point paths through a given point in the graph. The path handle based method, does a BFS traversal from the given point and reaches to all the end points in the fan out of the through point. At the end points, path enumeration for all paths to the end points takes place, with the only distinction from FTP algorithm being filtering of the from points.

```
ALGORITHM Enumerate_FTP:

    Inputs: Set of from pins F ∈ S and to pins T ∈ E
    Outputs: Enumerated DAG Paths
    foreach e ∈ T
        P← Enumerate_PHPaths (e)
        foreach path p ∈ P
                if start point of p not part of F
                        Remove p from P
                end
        end
    foreach p in P
        Enumerate_DAGPath (p)
```

IV. RESULT AND CONCLUSION

In this section we have done a basic comparison of the DFS based path enumeration with point handle based path enumeration from the vantage point of FTP and ThP enumeration problems. DFS based algorithms have been gauged for two variants of graph implementation:

1. Every graph vertex has only the out-going directed edges, which means movement in one direction is possible
2. Every graph vertex has out-going and in-coming directed edges both, which makes the backward iteration possible, but leads to memory penalty.

We have generated random directed acyclic graphs using a simple C++ method reported by [9], interested readers or researchers may as well try to compare the performance of our methods with DFS based path enumeration, for the graphs representing their domain of problem. We have presented the runtime of performing K different path enumeration operations, both in case of FTP and ThP problems. On a set of variable size graphs where we define the size of E.V, we have done performance analysis, with varying number of FTP and ThP enumeration calls on both our method and DFS based method, one must note that the tag numbers are for enumerating tag paths only and from this we can enumerate the original graph paths in O (path length) time. Results of our experimentation are presented below, we have taken random sets of graph in the format G (V, E), varying in the size and then performed fixed number FTP and ThP experiments as shown, and generated CPU time for each experiment. We have not done memory analysis on our data set, as we do not see memory as a bottleneck on our experimentation; an extensive runtime vs memory correlation may give some more insight into the solution effectiveness. For DFS we have not considered the case where there are no incoming edges because in such cases the ThP becomes practically impossible, so we have discarded this out of our experimentation, an interested reader can perform such analysis.

Table: Experimental Result

		Time in milliseconds			
Graph (#Nodes, #Edges)	**#Commands**	**TAG_FT**	**DFS_FT**	**TAG_TH**	**DFS_TH**
362, 1924	100	270	280	270	264790
458, 3088	200	3500	4410	4380	51362200
346, 1733	300	570	580	1000	382140
450, 2922	400	4200	5810	5123	29803900
355, 1787	600	1100	1140	900	708600
346, 1751	700	1500	1140	1750	1018000

Conclusively, we can see that the convergence time for point handle based algorithms comes better compared to DFS based algorithms. Interested readers may use these algorithms in their problem space and see how effective this data structure is for their class of problems. Moreover, we would further like to explore parallel versions of these algorithms on path enumeration and see how much scalability and speedup we can derive out of this.

REFERENCES

[1] GIUSEPPE DI BATTISTA et al, DRAWING DIRECTED ACYCLIC GRAPHS: AN EXPERIMENTAL STUDY, Int. J. Comput. Geom. Appl. 10, 623 (2000). Reference of applications page where paths are needed

[2] Eppstein, 1998, D. Eppstein, Finding the *K* shortest paths, Journal of the Society for Industrial and Applied Mathematics, 28 (2) (1998), pp. 652-673

[3] Yau-Tsun Steven Li , Sharad Malik, Performance analysis of embedded software using implicit path enumeration, Proceedings of the 32nd annual ACM/IEEE Design Automation Conference, p.456-461, June 12-16, 1995, San Francisco, California, USA [doi>10.1145/217474.217570]

[4] Dial, R B, 1971 "A probabilistic multipath assignment model which obviates path enumeration" Transportation Research 5 (2) 83–111 Google Scholar, Crossref

[5] E. DeutschDyck path enumeration, Discrete Math., 204 (1999), pp. 167-202

[6] A. Marino, Analysis and Enumeration, Atlantis Studies in Computing 6, Atlantis Press and the authors 2015

[7] John J. Zasio, Kenneth C. Choy, Darrell R. Parham, "Static timing analysis of semiconductor digital circuits", US Parent US4924430 A, issued May 8, 1990

[8] Maria Gradinariu 1 Sébastien Tixeuil , "Self-stabilizing Vertex Coloring of Arbitrary Graphs", International conference on Principles of Distributed Systems (OPODIS 2000), Dec 2000, Paris, France. pp.55-70, 2000

[9] GuyMelançon, FabricePhilippe, "Generating connected acyclic digraphs uniformly at random", Information Processing Letters Elsevier, Vol 90 Issue 4, Pages 209-213, 2004