Edge-Critical Strongly Connected Directed Graphs

Abhinav Kumar Verma[1] and Akrati Saxena[2][0000−0002−7151−6309]

[1] Department of Electronics and Electrical Communications, Indian Institute of Technology, Kharagpur, India
[2] The Leiden Institute of Advanced Computer Science, Universiteit Leiden, the Netherlands a.saxena@liacs.leidenuniv.nl

Abstract. We propose an iterative algorithm for generation of all edge-critical strongly connected directed graphs. A proof of correctness of the algorithm is also provided.

Keywords: edge-criticality · strongly connected · directed graph · spanning tree.

1 Introduction

A directed graph is strongly connected if there exists a path between any pair of nodes. It is edge-critical, if the strongly connected digraph ceases to be strongly connected when any of its edge is removed. The Tarjan algorithm is used to test the strong connectivity of a digraph. The algorithm determines strong connectedness in $\mathcal{O}(|E|)$. Assuming self loops and multiple edges are not allowed, there exists only one digraph of two nodes that is strongly connected. Coincidentally, it is edge-critical too.

Our Contributions. We propose an algorithm to test the edge criticality of a strongly connected digraph in $\mathcal{O}(|E|^2)$. Then, we propose an iterative algorithm to construct a set of all unlabelled edge-critical strongly connected directed graphs with N+1 nodes provided a set of all such graphs with N nodes. The edge-criticality of the graphs constructed by the algorithm is self-evident. We provide a proof for the completeness of the algorithm i.e. all edge-critical strongly connected digraphs of N+1 nodes are constructed by the algorithm.

2 Algorithms

2.1 Edge-Criticality of a Strongly Connected Digraph

To test if a strongly connected digraph is edge-critical, Tarjan algorithm is employed repetitively with removal of an edge with replacement from the digraph. If the edge-removed digraph remains strongly connected, then the original digraph is not edge-critical. A pseudo-code for this algorithm is provided below.

Algorithm 1 Test Edge-Criticality

Require: A strongly connected directed graph $G = (V, E)$
Ensure: Whether G is edge-critical
for each edge $e \in E$ **do**
  Remove edge e from G to create G'
  Test if G' is strongly connected using Tarjan's algorithm
  if G' is not strongly connected **then**
    Restore edge e in G
  else
    return G is not edge-critical
  end if
end for
return G is edge-critical

2.2 Construction of All N+1 Nodes Edge-Critical Strongly Connected Digraphs

Let us denote the set of all edge-critical strongly connected digraphs with N nodes as $\mathcal{P}_N$. We initiate an empty set $\mathcal{Q}$ to store the constructed digraphs with N+1 nodes. A digraph $\mathcal{G}$ is popped from $\mathcal{P}_N$ and an unconnected node $\mathcal{V}_{new}$ is added to the digraph. A set $\mathcal{L} = Nodes(G) \times Nodes(G)$ of all possible pairs of nodes is constructed.

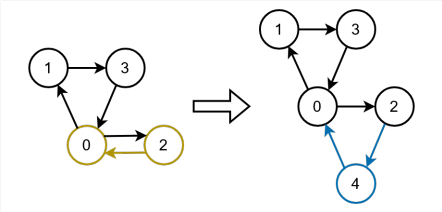
Iteratively, a pair (a, b) is popped from the set $\mathcal{L}$. If $(a, b) \in Edge(G)$, then the edge (a,b) is removed and edges (a, $\mathcal{V}_{new}$) and ($\mathcal{V}_{new}$, b) are added to construct $\mathcal{G}_{(a,b)}$, an edge-critical strongly connected digraph of N+1 edges. If this graph and its isomorphs are not a member of $\mathcal{Q}$, then the graph is added to the set $\mathcal{Q}$.

If $(a, b) \notin Edge(G)$, then the edges (a, $\mathcal{V}_{new}$) and ($\mathcal{V}_{new}$, b) are added and the resulting graph is checked for edge-criticality. If this graph is edge-critical and neither the graph nor its isomorphs are a member of $\mathcal{Q}$, then the graph is added to the set $\mathcal{Q}$. A pseudo-code for this algorithm is provided below.

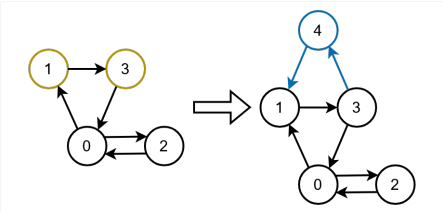
3 Proofs

3.1 Edge-Criticality Algorithm

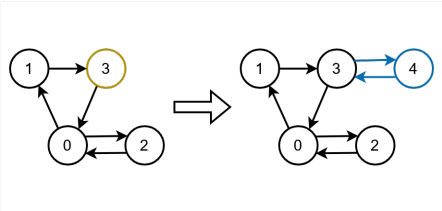
The correctness of edge-criticality algorithm can be directly verified using the definition of an edge-critical digraph. A strongly connected digraph is not edge-critical if it maintains strong connectivity after the removal of any single edge. Conversely, if the removal of any edge destroys the strong connectivity of the digraph, then the digraph is edge-critical. The algorithm directly leads to this definition.



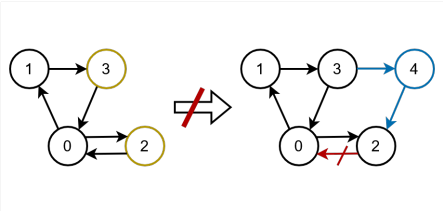
(a) a = 2, b = 0, and (2, 0) ∈ Edge($\mathcal{G}$). When (a, b) ∈ Edge($\mathcal{G}$), the extended digraph is always edge-critical.



(b) a = 3, b = 1, and (3, 1) ∉ Edge($\mathcal{G}$). Constructed digraph $\mathcal{G}_{(3,1)}$ is edge-critical.



(c) a = 3, b = 3, and (3, 3) ∉ Edge($\mathcal{G}$). Constructed digraph $\mathcal{G}_{(3,3)}$ is edge-critical.



(d) a = 3, b = 2, and (3, 2) ∉ Edge($\mathcal{G}$). Constructed digraph $\mathcal{G}_{(a,b)}$ is not edge-critical, thus it is not added to the set $\mathcal{Q}$.

Fig. 1: Extension of an edge-critical strongly connected digraph of 4 nodes to 5 node digraph by the algorithm. Nodes of the existitng digraph are numbered from 0 to 3 and $\mathcal{V}_{\text{new}} = 4$ for all four cases. Yellow highlights indicate nodes and edges affected during the extension process, while blue highlights show both newly added edges and the nodes involved in these new connections. The red edge with a stroke is redundant and its removal doesn't affect strong connectedness of digraph.

Algorithm 2 Construct Edge-Critical Strongly Connected Digraphs

Require: Set of edge-critical strongly connected digraphs $\mathcal{P}_N$ with N nodes
Ensure: Set $\mathcal{Q}$ of edge-critical strongly connected digraphs with $N+1$ nodes

```
Initialize empty set Q
for each digraph G ∈ P_N do
    Construct set L = Nodes(G) × Nodes(G)
    for each pair (a, b) ∈ L do
        Initialize G_(a,b) = G
        Add unconnected node V_new to G_(a,b)
        Add edges (a, V_new) and (V_new, b) to G_(a,b)
        if (a, b) ∈ Edge(G) then
            Remove edge (a, b) from G_(a,b)
        else
            if graph G_(a,b) is not edge-critical then
                continue
            end if
        end if
        if G_(a,b) and its isomorphs ∉ Q then
            Add G_(a,b) to Q
        end if
    end for
end for
return Q
```

3.2 Correctness of the Construction Algorithm

In this section, we prove that any digraph in the set $\mathcal{Q}$ is necessarily edge-critical strongly connected digraph. We proceed by proving two theorems that correspond to the scenarios, $(a,b) \notin \text{Edge}(\mathcal{G})$ and $(a,b) \in \text{Edge}(\mathcal{G})$.

We use these notations and description of a digraph $\mathcal{G}_{(a,b)}$ for the following theorems and the proofs.

$$
\begin{aligned}
&\mathcal{V} = Nodes(\mathcal{G});\\
&e_{in} = edge(a, \mathcal{V}_{\text{new}});\\
&e_{out} = edge(\mathcal{V}_{\text{new}}, b);\\
&Nodes(\mathcal{G}_{(a,b)}) = \mathcal{V} \cup \mathcal{V}_{\text{new}} = \mathcal{W};\\
&Edges(\mathcal{G}_{(a,b)}) = Edges(\mathcal{G}) \cup e_{in} \cup e_{out};
\end{aligned}
$$

Theorem 1. *If $\mathcal{G}$ is an edge-critical strongly connected digraph with nodes a and b such that (a, b) $\notin$ Edges($\mathcal{G}$) then the graph $\mathcal{G}_{(a,b)}$ is a strongly connected digraph.*

Proof. We observe that there exists a path from any node in set $\mathcal{V}$ to any other node in the same set because the original graph $\mathcal{G}$ is strongly connected. $a \in \mathcal{V}$,

$b \in \mathcal{V}$, and there exists a path from $\mathcal{V}_{\text{new}}$ to the node b and from the node a to node $\mathcal{V}_{\text{new}}$. Thus, there must exist a path from $\mathcal{V}_{\text{new}}$ to any node in $\mathcal{V}$, from any node in $\mathcal{V}$ to $\mathcal{V}_{\text{new}}$, and from $\mathcal{V}_{\text{new}}$ to $\mathcal{V}_{\text{new}}$. So, there exists a path from any node in $\mathcal{W}$ to any other node in the same set. Hence, $\mathcal{G}_{(a,b)}$ is strongly connected.

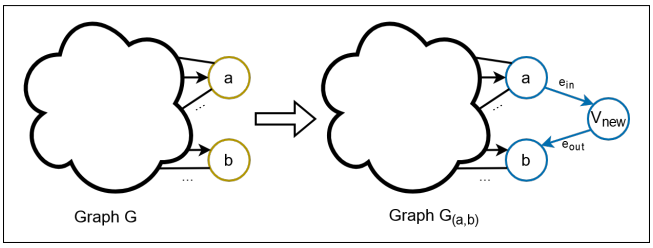


Fig. 2: Transformation of a edge-critical strongly connected digraph $\mathcal{G}$ to a strongly connected digraph $\mathcal{G}_{(a,b)}$ when (a, b) $\notin$ Edges($\mathcal{G}$).

Note: The digraph $\mathcal{G}_{(a,b)}$ is not necessarily edge-critical and therefore we require to check the edge-criticality explicitly in the algorithm for $(a, b) \notin \text{Edges}(\mathcal{G})$ scenario.

In addition we use these notations and description of a digraph $\mathcal{G}'_{(a,b)}$ for the next theorem and its proof.

$$
\begin{aligned}
&e_{-} = edge(a, b);\\
&Nodes(\mathcal{G}'_{(a,b)}) = Nodes(\mathcal{G}_{(a,b)});\\
&Edges(\mathcal{G}'_{(a,b)}) = Edges(\mathcal{G}_{(a,b)}) \setminus e_{-}.
\end{aligned}
$$

Theorem 2. *If $\mathcal{G}$ is an edge-critical strongly connected digraph with nodes a and b such that (a, b) $\in$ Edges($\mathcal{G}$) then the graph $\mathcal{G}'_{(a,b)}$ is also an edge-critical strongly connected digraph.*

In this scenario, the edge $e_{-} = (a, b)$ is replaced by two edges e_{in} and e_{out} preserving the connectivity from node a to b.

Proof. As the original graph $\mathcal{G}$ was strongly connected, there must exist a path from any node in $\mathcal{V}$ to any other node in the same set on graph $\mathcal{G}'_{(a,b)}$. If the path contained edge e_{-} on $\mathcal{G}$ then it would instead contain e_{in} and e_{out} on $\mathcal{G}_{(a,b)}$. Also there exists a path from from $\mathcal{V}_{\text{new}}$ to the node b and from the node a to node $\mathcal{V}_{\text{new}}$, so there must exist a path from $\mathcal{V}_{\text{new}}$ to any node in $\mathcal{V}$ and from any node in $\mathcal{V}$ to $\mathcal{V}_{\text{new}}$. Furthermore, the path $e_{out} \cup Path(b, a) \cup e_{in}$ connects $\mathcal{V}_{\text{new}}$ to itself. So, there exists a path from any node in $\mathcal{W}$ to another node in the set. Hence, $\mathcal{G}'_{(a,b)}$ is strongly connected.

If the edge e_{in} is removed, then there doesn't exist any path from the node a to node $\mathcal{V}_{\text{new}}$. Similarly, if the edge e_{out} is removed, then there doesn't exist any

path from the node $\mathcal{V}_{\text{new}}$ to node b. Thus with removal of either of these edges, the graph $\mathcal{G'}_{(a,b)}$ ceases to be strongly connected.

The graph $\mathcal{G}$ was edge-critical. If any edge $e = (x, y) \neq e_{-}$ is removed from the graph $\mathcal{G}$, then there exists no path from the node x to the node y on the graph $\mathcal{G}$. This must be the case otherwise the edge e would be redundant. Now consider the edge e on the new graph $\mathcal{G'}_{(a,b)}$. Here too, if the edge e is removed, there must not exist any path from the node x to the node y because if such a path existed on $\mathcal{G'}_{(a,b)}$, then the path would exist on $\mathcal{G}$ too with possible replacement of $\{e_{in}\} \cup \{e_{out}\}$ by $\{e_{-}\}$. Thus removal of any edge from the graph $\mathcal{G'}_{(a,b)}$, dissolves its strong connectedness. Hence, $\mathcal{G'}_{(a,b)}$ is edge-critical strongly connected digraph.

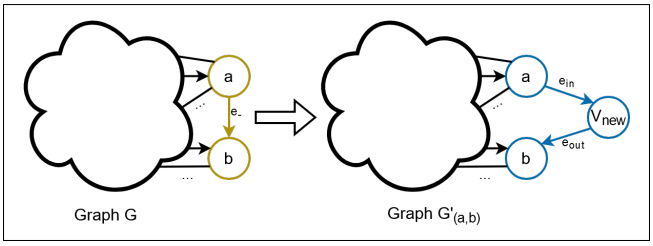


Fig. 3: Transformation of a edge-critical strongly connected digraph $\mathcal{G}$ to another edge-critical strongly connected digraph $\mathcal{G'}_{(a,b)}$ when (a, b) $\in$ Edges($\mathcal{G}$).

With these theorems, the correctness of algorithm is established i.e. each element of the output set $\mathcal{Q}$ is edge-critical strongly connected digraph. Now, we proceed to prove the completeness of the algorithm i.e. the set $\mathcal{Q}$ contains all possible edge-critical strongly connected digraph of N+1 nodes, if it receives a set of all possible edge-critical strongly connected digraph of N ndoes as its input.

3.3 Reduction of Edge-Critical Strongly Connected Digraphs to Smaller Node Counts

Theorem 3. *If an edge-critical strongly connected digraph $\mathcal{G}$ has a node V_{red} with one in-edge (e_{in}) and one out-edge (e_{out}) only, then the digraph can be reduced to another edge-critical strongly connected digraph with one fewer node by performing one of the following operations:*

1. *Removing the node V_{red} and the edges e_{in} and e_{out}; or*
2. *Removing the node V_{red}, removing the edges e_{in} and e_{out}, and adding an edge from the pre-node of e_{in} to the post-node of e_{out}.*

Proof. Let us assume that the pre-node of e_{in} is a and the post-node of e_{out} is b on the graph $\mathcal{G}$. Now, we construct a graph $\mathcal{H}$ from the graph $\mathcal{G}$ by following the

second operation i.e. removing the node V_{red}, removing the edges e_{in} and e_{out}, and adding an edge $e = (a, b)$. Now let $\mathcal{V} = Nodes(\mathcal{G})$ and $\mathcal{W} = Nodes(\mathcal{H})$.

The digraph $\mathcal{G}$ is strongly connected so it is possible to reach from any node in the set $\mathcal{V}$ to another node in the set on $\mathcal{G}$. Since $\mathcal{W} \subset \mathcal{V}$, a path exists from any node in $\mathcal{W}$ to another node in the set on the graph $\mathcal{H}$ with the possibility of including the edge $\{e\}$ in the path instead of $\{e_{in}\} \cup \{e_{out}\}$, if the latter is a component of the path on the graph $\mathcal{G}$. Thus, the graph $\mathcal{H}$ is strongly connected.

The graph $\mathcal{G}$ is edge-critical. If any edge $e = (x, y)$ other than e_{in} and e_{out} is removed from the graph $\mathcal{G}$, then there exists no path from the node x to the node y on the graph $\mathcal{G}$. Now consider the edge e on the graph $\mathcal{H}$. Here too, if the edge e is removed, there must not exist any path from the node x to the node y because if such a path existed on $\mathcal{H}$, then the path would exist on $\mathcal{G}$ too with possible replacement of $\{e_{in}\} \cup \{e_{out}\}$ by $\{e\}$. Thus removal of any edge from the graph $\mathcal{H}$ except possibly (a, b), dissolves its strong connectedness.

Now, we construct a graph $\mathcal{H}'$ by removing the edge e from the graph $\mathcal{H}$. This construction corresponds to the first operation. If the graph $\mathcal{H}'$ is strongly connected then it is also edge-critical following the above-mentioned argument. However, if it is not strongly connected, then $\mathcal{H}$ is necessarily edge-critical as removal of any of its edge including edge e dissolves the strong connectedness of the graph. Hence, either the graph $\mathcal{H}'$ composed by first operation or the graph $\mathcal{H}$ composed by second operation must be an edge-critical strongly connected digraph.

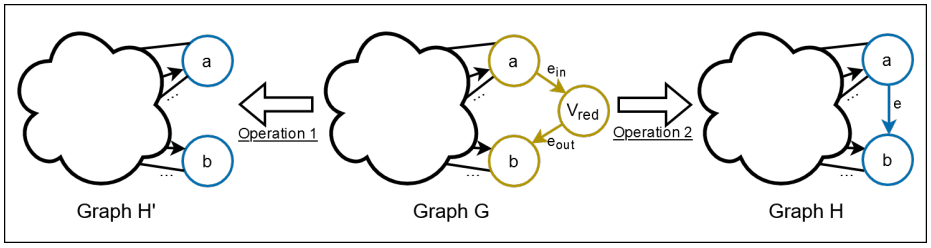


Fig. 4: Demonstration of Operation 1 and Operation 2 on a edge-critical strongly connected digraph $\mathcal{G}$ resulting into digraphs $\mathcal{H}'$ and $\mathcal{H}$ respectively. Either $\mathcal{H}'$ or $\mathcal{H}$ is also edge-critical strongly connected digraph.

We shall now prove the most important theorem regarding edge-critical strongly connected digraphs.

3.4 Completeness of the Construction Algorithm

Theorem 4. *Any edge-critical strongly connected digraph must contain a node with degree 2.*

Proof. Let us assume that there exist an edge-critical strongly connected digraph ($\mathcal{G}$) of N nodes, such that all its node have degree 3. We shall now show that the assumption leads to a contradiction.

Select any arbitrary node A of the graph $\mathcal{G}$ and construct a directed acyclic tree rooted at the node A. Add a node n and an edge e only if $e = (n, m) and m \in Nodes(DAGconstructedsofar)$ i.e we add a node if there exists an edge pointing from the node to another node in the DAG. We end up with a DAG such that there exists one and unique path from each node to the node A. We know that such a DAG exists since $\mathcal{G}$ is stringly connected, thus it is possible to reach any node (A in this case) from any other node. Also all the edges on the DAG correspond to an edge on the graph $\mathcal{G}$ but not all edges of the graph $\mathcal{G}$ are present on the DAG.

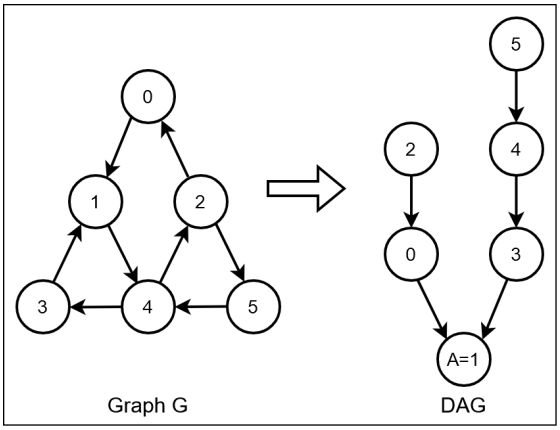


Fig. 5: Conversion of an edge-critical strongly connected digraph to its DAG equivalent with root at $A = 1$.

Since the graph $\mathcal{G}$ has two more nodes, the DAG must contain at least one leaf. We shall concentrate on it and try to prove how it is impossible to have degree 3 more less for all nodes. The leaves of the DAG only have one out-edge so they must have at least two other edges that are not a part of the DAG. Also the node A must have one out-edge, otherwise it is impossible to reach any other node from the node A.

Now, one of the following three scenarios must be true about the edges (other than the out-edge on DAG) of any leaf of the DAG:

1. The leaf has two or more out-edges only.
2. The leaf has two or more in-edges.
3. The leaf has one in-edge and one or more out-edges.

Scenario 1: A leaf has two or more out-edges only This is clearly not possible as such a leaf will have out-edges only and no in-edge rendering it impossible to reach from any node on the graph $\mathcal{G}$ to this node.

Scenario 2: A leaf has two or more in-edges Let's name the leaf node l and assume the base case i.e. the leaf has two in-edges $e_1 = (x, l)$ and $e_2 = (y, l)$ on the graph $\mathcal{G}$.

Now, let us consider the case when either $x = A$ or $y = A$ and without loss of generality, we assume $x = A$ such that $e_1 = (A, l)$. Now, consider the path - $P_1 = Path_{DAG}(y, A) \cup e_1$. Clearly $e_2 \notin P_1$ since $e_2 \notin Edges(DAG)$ and $e_1 \neq e_2$. So there exists a path from y to l rendering the edge e_2 redundant - a contradiction since $\mathcal{G}$ is an edge-critical strongly connected digraph.

Having established that $x \neq A$ or $y \neq A$, we proceed with the more general scenario. For l to be reachable from A, there must exist one or more paths from A to x. Then either of the two cases must hold true about these paths:

1. There exists a path on $\mathcal{G}$ from A to x that does not contain the edge $e_2 = (y, l)$.
2. All paths from A to x on $\mathcal{G}$ contain the edge $e_2 = (y, l)$.

In the first case, let's name the path on $\mathcal{G}$ from A to x that does not contain the edge e_2, P_2. Now, consider the path - $P_3 = Path_{DAG}(y, A) \cup P_2 \cup e_1$. Clearly $e_2 \notin P_3$, but P_3 is a path from node y to the node l, thus edge e_2 is redundant - a contradiction since $\mathcal{G}$ is an edge-critical strongly connected digraph.

In the second case, it is evident that there exists a path P_4 from A to l, since there exists a path from A to x on $\mathcal{G}$ that contains the edge $e_2 = (y, l)$. But, there also exists a path from x to A on the DAG. Thus the path - $P_5 = Path_{DAG}(x, a) \cup P_4$ connects the node x to l, and $e_1 \notin P_5$, so the edge e_1 is redundant - a contradiction since $\mathcal{G}$ is an edge-critical strongly connected digraph.

Since the base case is violated, we have proved that it is impossible for a leaf of the DAG to have two or more in-edges (irrespective of the count of out-edges) on the edge-critical strongly connected digraph $\mathcal{G}$.

Scenario 3: A leaf has one in-edge and one or more out-edges Since, the first two scenarios are impossible all leafs on the DAG must have one in-edge and at least one-edge (apart from the out-edge that is part of DAG). Let's assume that there are a total of m leafs and name them as $l_1, l_2, ..., l_m$, their corresponding in-edges as $ein_1, ein_2, ..., ein_m$, the pre-nodes corresponding to these in-edges as $x_1, x_2, ..., x_m$. Also, let's name the out-edges as $eout_1, eout_2, ..., eout_m$, and the post-nodes corresponding to these out-edges as $y_1, y_2, ..., y_m$. Let $P_1, P_2, ...P_m$ be the paths from the leaf nodes $l_1, l_2, ..., l_m$ to the root node A on the DAG. Also, let $\mathcal{Z}_m = \{1, 2, ..., m\}$

$\forall i \in \mathcal{Z}_m$ the path from any node (including the node A) to l_i on $\mathcal{G}$ must contain the edge ein_i, since it is the only in-edge for the node l_i.

Let the node y_i fall on path P_j following some map from $\mathcal{Z}_m \to \mathcal{Z}_m$. Then, all paths from A to l_j must contain the edge $eout_i = (l_i, y_i)$, otherwise the path - $R_1 = Path_{DAG}(l_i, A) \cup Path(A, l_j) \cup Path_{DAG}(l_j, y_i)$ does not contain the edge $eout_i$ but connects the node l_i to the node y_i on $\mathcal{G}$, violating the edge-criticality of graph $\mathcal{G}$. For the corner condition $y_i = l_j$, the statements holds too.

So, we have essentially proved that the path $S = A \to l_1 \to y_1$ must lead to node l_i such that $y_1 \in P_i$. The node $y_1 \notin P_1$ as there there exists a path from l_1 to any node in P_1. Moreover, this chain propagates to y_i, and $y_i \notin P_1$, $y_i \notin P_i$. This holds because the path - $R_2 = Path_{DAG}(l_i, A) \cup Path(A, l_a) \cup Path_{DAG}(l_a, y_i)$ for l_a being a node on the path S, connects node l_i to y_i and it doesn't contain the edge $eout_i = (l_i, y_i)$ violating the edge-criticality of graph $\mathcal{G}$. Note the path $Path(A, l_a)$ in R_2 is contained in the path S.

Continuing the construction of the path $S = A \to l_1 \to y_1 \to l_i \to y_i \to l_j \to ... \to y_k \to l_{final}$ we observe that when all leaf nodes are visited, it is possible for the y_{final} to not lie on any of the paths as all nodes must lie on at least one path. Thus, we have shown that it is impossible for all leaf nodes to have one in-edge and one or more out edge.

Summary We have shown that none of the three scenarios for a leaf on the constructed spanning tree is feasible. Thus, our assumption that there exists an edge-critical strongly connected digraph ($\mathcal{G}$) of N nodes, such that all its nodes have degree 3 or more is incorrect. All edge-critical strongly connected digraph must contain a node of degree 2.

4 Results

The number of non-labeled edge-critical strongly connected digraphs grows quickly (beyond-polynomial) with the number of node N.

Table 1: Total Prime Digraphs by Number of Nodes.

Number of Nodes	Total Prime Digraphs
2	1
3	2
4	5
5	15
6	63
7	288
8	1526
9	8627
10	52021

References

1. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM Journal on Computing **1**(2), 146–160 (1972)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. 3rd edn. MIT Press, Cambridge (2009)
3. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
4. Eswaran, K.P., Tarjan, R.E.: Improved algorithms for finding minimal cycle bases. SIAM Journal on Computing **4**(4), 604–616 (1975)
5. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications. Springer Monographs in Mathematics, Berlin (2007)
6. Even, S.: Graph Algorithms. 2nd edn. Cambridge University Press, Cambridge (2011)
7. Diestel, R.: Graph Theory. 4th edn. Springer, Heidelberg (2010)