A Scalable Acyclic Graph Generator

Sandeep Gupta[1]

I. INTRODUCTION

Data generators play an important role in algorithm design and optimization engineering. They are an important tool for modeling, benchmarking, scalability, and cost estimation. Graph generators have been an extensive area of research in last decade. With the unprecedented growth in Internet, WWW, and, social networks there was an immediate need to model and subsequently build generators that produce graphs reflecting structures of these domains. It was discovered in [6], [7] that such networks are fractal in nature (scale-free) and can be described via physical phenomena of “rich gets richer” or “preferential attachment” [1]. The mathematical model that produces such fractal graphs is based upon R-MAT [2] or Kronecker Product [8] and is the central theory underlying the scale-free generators. A large body of work exist that utilize either of the mathematical models to develop scalable scale-free graphs [2], [10]. These works have contributed significantly towards development of network protocols, algorithms, and, architecture design.

The community has paid little attention to generation of acyclic graphs. Acyclic graphs, much like scale-free graphs, appear in many areas of computation and engineering. Knowledge representation, binary decision diagram, dependency graphs, semantic web, and, binaries of computer program to name of few. In the field of life-sciences and bio-informatics such structures are used to create ontologies that represent the compendium of factual information. In recent years the number and the sizes of ontologies has been growing steadily. It is important for the database and the computing world to be able to develop algorithms over such workloads to be better address the needs of the domain science. Unlike social networks and WWW, the workloads in life-science and knowledge engineering disciplines are much more complex and include reachability and pattern queries, and, lowest common ancestors [4].

The size of acyclic graphs is not as large as the social-network graphs (upwards of multi-billion edges) but they can potentially be many orders of magnitude larger then those present currently. This is particularly true in life-sciences, bio-informatics and knowledge representation domain. This is because disparate ontologies can reference entities across each other. In fact, the Gene Ontology, has been constructed by combining ontologies of sub-species. In this field we are witnessing almost exponential increase in number of ontologies due to technological growth in bio-informatics and capabilities in life-sciences research.

Hence in order to drive and evaluate the next generation of information engines and tools an scalable graph generator that can build acyclic graphs of varying characteristics is of immense importance. To this end we develop a generator that given a set of distribution criteria over the acyclic graph can produce a stochastic graph having those characteristics. In particular we consider the in-degree distribution, out-degree distribution, the depth distribution, and the path-length distribution (distribution of random walks from source to sink). Our work differs from previous R-MAT based graph generator work in that our’s is generic acyclic graph generator and doesn’t try model a particular structure. Our generator is parametrized by the distribution functions. The generator simply uses these distribution function to guide the structure of the final graph. For the case of R-MAT generators the goal was to produce synthetic graphs that follows the structure of scale-free graphs i.e the distribution is implicit within the generator. In our work the distribution is explicit and is an input parameter for the generator.

A. Collapsing Nature of Acyclic Graphs

The most obvious scheme for generating acyclic acyclic graphs is to start with a tree and add edges between nodes \((u,v)\) if level of \(u\) is less then the level of \(v\). The node pair \(u\) and \(v\) are selected randomly. We argue that such scheme and other schemes that select edge pairs randomly produces degenerate graphs i.e. the depth distribution of the resulting acyclic graph collapses to less or equal to one. Alternatively, with very high probability they yield bipartite graphs irrespective of the initial tree structure and the type of random number generator.

We demonstrate this phenomena for binary trees. Consider a binary tree \(T\) with \(n\) nodes and \(n-1\) edges. Let \(td_v\) denote the tree depth of node \(v\). We follow the convention that root is at level 0. The level is in ascending order along the child/parent axis. To this tree we add \(c \cdot (n + \epsilon) \cdot n\) edges for some fixed \(c\) and \(0 < \epsilon < 1\). Each edge \(e = (u,v)\) is directed from the source \(u\) to \(v\) if \(td_u < td_v\). Let \(dd_u\) denote the depth of the graph post-addition random edges and is described as \(dd_u = \min\{dd_u + 1 | u\ \text{is a parent of v}\}\)

**Claim 1.1:** With high probability \(dd_u < \frac{c}{2}n\).

**Proof:** Sketch: Perform the generation in \(c\) rounds. In each round add \(n + \epsilon \cdot n\). For large enough \(\epsilon\), each node \(v\) with high probability (w.h.p) is a target node in at least one of the inserted edges. Let \(e = (u,v)\) be one of the edges. By definition of depth, this edge bounds the depth of \(v\) in the resulting dag i.e.

\[
dd_v \leq dd_u + 1
\]

Further, probability that \(td_u < td_v - 1\) is 1/2. This is because number of nodes at depth \(td_v - 1\) equals total number of nodes with depth less than \(td_v - 1\) (by the property of binary tree). Hence with at least 1/2 probability \(v\) has a new parent \(u\) whose
After completion of a round, applying the same logic on \( u \) implies that there exists \( u' \) such that \((u', u)\) is an edge and \( td_{u'} < td_u - 1 \). By recurrence, we can claim that every node suffers a decrease in depth by at least 1. This implies that for any node \( v \):
\[
d_{d,v} \leq td_v / 2
\]

after completion of a round.

Applying \( c \) rounds of edge insertion we can claim that
\[
d_{d,v} \leq td_v / 2^c
\]

Although we have shown the collapsing nature for binary trees the phenomena occurs irrespective of initial shape of the tree.

Since the depth of the tree is \( \log n \) it follows that if we add merely \( n \log n \) the entire tree will collapse to a single depth bipartite graphs. Benchmarks based upon such generators would simply test set-membership and set intersections and would not stress any graph-centric capabilities of the applications, kernels and algorithms. Many real-world acyclic graphs have average degree of more than 1. For e.g. arxiv, pubmed, yago, and go-uniprot have average degree of 11.12, 4.45, 6.38, and 4.99 respectively [12].

An alternative approach for producing acyclic graphs would be modify or post-process existing random generator and R-MAT generator to produce acyclic graphs. It is feasible to do so by finding strongly connected components and reduce a connected component to a single node. Let \( C_1, C_2, \ldots, C_c \) be the maximally strongly connected components (SCC) of digraph \( G \). Create a DAG \( D \) with \( V(D) = v_1, v_2, \ldots, v_c \) where each \( v_i \) corresponds to SCC \( C_i \). Edges in \( D \) reflect existence of edge between components i.e. if there is an edge \((c_i, c_j)\) where \( c_i \in C_i \) and \( c_j \in C_j \) then \((v_i, v_j)\) \( \in E(D) \).

We call \( D \) the reduced-DAG of digraph \( G \). However, random generator and R-MAT generator produce graphs that have one large giant component [3]. This phenomena is referred as “six degrees of separation” in social sciences and popular media. Hence the generated acyclic graph would have few nodes irrespective of the size of the original digraph.

In [11] authors presented a linear programming based approach towards generating acyclic graphs. In their approach each node is a variable in the linear programming formulation. Therefore it does not scale to multi-millions node acyclic graphs. On the other hand our approach is designed to scale to very large DAGs. This is because in our method instead of generating a graph that exactly satisfies all the parameter, it performs best effort to find the suitable graph and if not generates a graph that is close to required characteristics.

The algorithm is local and greedy. In each iteration a node is added to the graph and best effort is made to satisfy its in-degree and existing nodes’ out-degree requirements.

### B. Graph Generators

The simplest graph generator is due to Erdos-Renyi [5] which generates a random graph. Given \( n, p \) as number of nodes and edge probability respectively, this generator creates a graph with \( n \) nodes and (expected) \( np \) number of edges. An edge is created by picking two nodes at random and joining them. Erdos showed that even such a simple generator exhibit an interesting phenomena of phase change i.e. there exists a narrow range for \( p \) values for which number of connected components drop significantly for very small increment in \( p \).

Planar triangulated graphs [9] are another class of random graph generators in which the points are embedded on a Euclidean plane. The delaunay triangulation of the points yields a random graph. As mentioned earlier the R-MAT or Kronecker product based graph generators are the most widely used generators. Our work fills an important gap in this family of generators. Acyclic graphs are important class in many emerging life-sciences disciplines and are steadily growing in size and occurrences.

## II. Approach

We begin with few definitions

### A. Notations, Terminologies, and, Definitions

Given a directed graph \( G = (V(G), E(G)) \), let \( m, n \) denote number of nodes and edges respectively.

A path is a sequence of edges, \( P = \langle (v_1 = u, v_2), (v_2, v_3), \ldots, (v_{t-2}, v_{t-1}), (v_{t-1}, v_t = v) \rangle \). Alternatively, the same path can be represented as a sequence of vertices \( P = \langle v_1, v_2, \ldots, v_t \rangle \). A path is simple if a node appears at most once in the sequence; else the path has cycles.

Graph \( G \) is a directed acyclic graph (or DAG) if all paths are simple. Let \( G = (V(G), E(G)) \) denote a directed graph. Edge \( e \in E(G) \) is said to be out-incident on node \( u \) and in-incident on node \( v \) if \( e = (u, v) \). If \((u, v)\) is an edge in the DAG, then \( u \) is termed as in-incident neighbor of \( v \) and \( v \) is the out-incident neighbor of \( u \). The number of out-incident and in-incident are termed as in-degree and out-degree respectively.

Nodes with in-degree zero are termed as root nodes while nodes with out-degree zero are leaf nodes. We call the rest of the nodes as dag nodes. Given a node \( u \) let \( O(u) \) and \( I(u) \) denote its out- and in-degree respectively.

Our generator based upon the topological (or partial) ordering present naturally in acyclic graphs. We create \( s \) slots where \( s >> d \) (the depth of the resulting dag). The \( n \) nodes are distributed over these \( s \) slot based upon input distribution criteria. This distribution now defines the initial topological ordering of the nodes. Give two nodes \( n_i \) and \( n_j \) appearing in slots \( i \) and \( j \) the generator will guarantee that \( T(n_i) < T(n_j) \) if \( i < j \). Nodes at slot 0 form the roots of the dag and nodes at slot \( s \) form the leaf. Nodes in other slots can either be root, leaf, or, dag-nodes.

Once we have slotted the nodes we pick nodes starting from the lowest slot and build its out-edges. Let say we are at slot \( s_i \) and building out-edges for node \( n_i \). We select \( j > i \) based upon a distribution criteria. From this slot \( s_j \), we then randomly choose a node \( n_j \). After the addition of edge \((n_i, n_j)\) \( n_i \) has one out-degree satisfied and \( n_j \) has one in-degree satisfied. In order to satisfy all the out-degree requirements we repeat this process \( O(n_i) \) times. Our implementation
guarantees that each iteration picks a unique out-incidence node for \( n_i \).

In order to satisfy in-degree requirements of node \( n_j \) we place \( I(n_j) \) copies at its designated slot \( j \). Algorithms 1,2,3 provides the full description of the generator which use following notations. Let \( X_o, X_i, X_s, \) and, \( X_d \) be the fanout, fanin, slot, and, the depth distribution function. Further let \( x_o, x_i, x_s, \) and \( x_d \) be the outcomes from their respective distributions upon a invocation i.e \( x = X() \). Let \( S_i \) be the list of nodes shelved at slot \( i \). Let \( P \) be permutation of numbers from \([0 : n]\) and \( P[i] \) denote the \( i \)th value from this permutation. Let \( adj(u) \) be the adjacencies (out-incident) nodes of \( u \).

Algorithm 3 builds the acyclic graph given the various distribution functions. It calls algorithm shelf_nodes which distributes the nodes into shelves based upon the shelving function \( X_s \). Each node \( u \) is duplicated \( I(u) \) times where \( i \) is sampled from the in-degree distribution function \( X_i \). We randomly permute the nodes in each shelf so every node has equally likely possibility of being the target node. This ensure that resultin fan-in distribution has the same shape as \( X_i \).

Algorithm build_out_edges builds the edges for a given node \( u \). It first selects a slot based upon \( X_s \) and then picks a node \( v \) to form directed edge \( e = (P[u], P[v]) \). The number of out-edges incident of node \( u \) is sampled from degree distribution \( X_d \). In order to build the complete graph the build_out_edges is called for each node \( u \) in ascending order of the depth.

**Algorithm 1 shelf_nodes\((X_s, X_i)\)**

Require: \( max_s, x_s, X_s, X_i \) the maximum slot value, slot, and fan-in distributions respectively

1: for \( i \in \{0 : n\} \) do
2: \( x_s \leftarrow X_s() \)
3: \( x_i \leftarrow X_i() \)
4: for \( j \in \{0 : x_i\} \) do
5: \( \text{append}(S_{x_s, i}) \)
6: end for
7: end for
8: for \( i \in \{0 : D\} \) do
9: \( \text{random_shuffle}(S_d) \)
10: end for

**Algorithm 2 build_out_edges\((u, X_o, X_d)\)**

Require: \( X_d, P \) the depth distribution and permutation vector respectively

1: for \( i \in \{0 : d\} \) do
2: \( x_d \leftarrow X_d() \)
3: \( v \leftarrow \text{pop}(S_{x_d}) \)
4: \( \text{append}(adj(P[u]), P[v]) \)
5: end for

B. Choosing Distribution Functions

The choice of distribution function affects the shape and various characteristics of the resulting dag. In fact, some combination of distribution function may not be feasible or may yield degenerate graph. Here we provide distribution functions that lead to non-degenerate graphs whose shapes can be controlled by tuning the parameters of the distribution function. The distribution functions we use are in accordance with those used in other statistical and graph generators.

We consider \( X_i \) and \( X_o \) to be gaussian distribution with mean \( m \) and variance \( \mu = m/5 \). Figures 1(a),1(b),1(c),1(d) shows the gaussian distribution for this setup with \( m = 5, 10, 15, \) and 20. The slot distribution is set as discrete geometric function \( p \) i.e \( Pr(X = k) = (1-p)^{k-1}p \). The depth distribution is a scale free distribution and is parametrized by scale \( \gamma \) i.e \( Pr(k) \approx c k^{-\gamma} \). Figure 1(e) and 1(f) show the frequency distribution resulting from the geometric and scale-free distribution for \( p = 0.001 \) and \( \gamma = 0.04 \) respectively.

Our scale free distribution is the same as those used in R-MAT graph generator. The difference is that in R-MAT this distribution is used as joint-probability distribution to pick both the source and the destination. Here we use this distribution for a fixed source to arrive at the slot of target node. We then choose the target randomly from the set of all nodes present at the target slot.

For current experiment study we fixed \( p = 0.001 \) for discrete geometric function and \( \gamma = 0.02 \) for scale-free distribution. We found that lower values leads to degenerate acyclic graphs and higher value have little influence on resulting depth distribution. The in- and out- degree distribution is orthogonal to slot \( X_s \) and depth \( X_d \) distribution.

III. EXPERIMENTAL RESULTS

We used our proposed generator and choice of distribution function generate large acyclic graphs and study its various characteristics. We validate our model across following three criteria:

- Ability to produce non-degenerate graphs as we scale the number of nodes and edges.
- The shape of the in- and out- degree frequency distribution of the resulting graph
- Shape of the path length distribution as described below.

a) Path Length Frequency Distribution: We characterize acyclic graphs based upon the frequency distribution of the length of random walks starting from roots to the destination. This distribution characterizes the “richness” in the resulting structure. This attribute of acyclic graph becomes important since the depth parameter does not reveal the degree of irregularity. Two acyclic graphs with same depth value can have very different random path length frequency distribution.
We can illustrate this by constructing a dag $D = (V, E)$ where nodes in $V$ can be partitioned into $V_1, V_2, \ldots, V_t$ and edges into $E_1, E_2, \ldots, E_{t-1}$ sets s.t. an edge $e = (u, v) \in E_i$ be such that $u \in V_i$ and $v \in V_{i+1}$. This graph is a dag with roots as $V_1$ and leafs as $V_t$. It can be visualized as $t$ bi-partite graphs stacked over each other. The depth of the graph is $t$. However path length frequency distribution contains only a singleton value $t$ i.e every path in $D$ has length $t$. Such a graph will not be very meaningful for benchmarking and characterization even though it may satisfy other acyclic graph properties.

We generated graphs with $2^{23}$ nodes. The number of edges is controlled by the $X_i$ and $X_o$ which are parametrized by $m$ ($\text{meandegree} = m$ and $\text{variance} = m/5$). We studied generated graphs by varying $m = 5, 10, 15, 20$. Figures 1(a), 1(b), 1(c), 1(d) shows the shape of the $X_i$ and $X_o$ for these set of values. Figures 1(e) and 1(f) shows the geometric frequency distribution (with $p=0.001$) and scale-free distribution with $d = 0.04$ used as slot distribution and depth distribution respectively.

Figures 2(b), 2(f), 2(j), 2(n), 2(a), 2(e), 2(i), 2(m) show the resulting in-degree and out-degree frequency distribution. We see that the graph generated is able to capture the input shape. The depth distribution illustrated in figures 2(c), 2(g), 2(k), 2(o) show that our graph does not collapse under increased degree. In fact comparing depth distribution for $m = 5$ and $m = 20$ we see that the maximum depth increases while increasing the number of edges in the graph. Figures 2(d), 2(h), 2(l), 2(p) show the random path-
Fan-Out degree distribution

(i) Out-degree distribution (m=15)

Fan-In degree distribution

(j) In-degree distribution (m=15)

Depth Distribution (m=15)

(k)

Path Length Distribution (m=15)

(l)

Out-degree distribution (m=20)

(m)

In-degree distribution (m=20)

(n)

Depth Distribution (m=20)

(o)

Path Length Distribution (m=20)

(p)
length distribution which follows a power law distribution. The steepness of the distribution i.e. the slope of the exponent decreases as we increase m.

IV. Conclusion

References