On Generating Random Network Structures: Trees*

Alexey S. Rodionov[1] and Hyunseung Choo[2]

[1] Institute of Computational Mathematics and Mathematical Geophysics
Siberian Division of the Russian Academy of Science
Novosibirsk, RUSSIA +383-2-396211
alrod@rav.sscc.ru

[2] School of Information and Communication Engineering
Sungkyunkwan University
440-746, Suwon, KOREA +82-31-290-7145
choo@ece.skku.ac.kr

Abstract. Random trees (RTs) are widely used for testing various algorithms on tree-type networks and also for generating connected graphs similar to real nets. While random topologies based on RTs are generally accepted as network models, the task of their generation is almost unexplored. In this paper we discuss the set of basic algorithms for generating random trees. The fast algorithms with proven properties are presented for generating random trees under conditions for given restrictions, such as a limited node degree, fixed node degrees, and different probabilities of edge existence. Generating random graphs similar to physical networks are underway.

1 Introduction

In this paper we present the set of basic algorithms for generating random trees (RTs). RTs are used for testing various algorithms on tree-structural networks and as a base for further algorithms for generating connected graphs. Random graphs are widely used as a model of communication networks, usually for the testing of control algorithms [1,2,3]. Using real network structures for an algorithm testing is impossible with a simple reason: to achieve reliable results the large number of different structures with given properties is necessary while actual data for real network structures are usually inaccessible and the number of such real structures is small. While random trees and graphs are widely used as a network model, the task of their generation is almost unexplored. It seems that most authors who use random trees and graphs as a model consider the task as an obvious one. Still there are a lot of problems in the task of random graphs generation, especially when complex limitations are loaded on the graph structure. The usual requirements are to guarantee the uniformity on the given

* This paper was partially supported by BK21 program. Dr. H.Choo is the correspondent author

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2658, pp. 879–887, 2003.

space of graphs (to the enumeration) and attainability of all graphs from the given space. Hereafter we discuss some base approaches for the generation of RTs and present some effective algorithms. The rest of the paper is organized as follows. Section 2 is devoted to the base approaches, notations and concepts. In Section 3 we discuss about the generation algorithms of RTs for equal probabilities of edge existence. Section 4 presents the algorithm for generating RTs with different probabilities of edges existence. Section 5 is a brief conclusion.

2 Preliminaries

The variable types are clear from the context usually, or are stated in comments. The following procedures are used in the algorithms descriptions:

- `INC(n)`, `DEC(n)` – increment and decrement operators;
- `FindPlace(x,V,OK)` – integer function, returns the index of the first element in vector `V` equal to `x`. OK is TRUE if such element is found and FALSE in the opposite case;
- `remove(x,V)` – removes element `x` from vector `V` (the last element replace `x`). If `x` is not belongs to `V` then no action is done;
- `rand()` – random number uniformly distributed on $(0, 1)$;
- `random(n,m)` – integer random number uniformly distributed on $[n, n + 1, \ldots, m]$, $n \leq m$;
- `random(P,n,m)` – integer random number distributed on $[n, n + 1, \ldots, m]$, $n \leq m$ with probabilities $p_i = P(x_i = n + i - 1)$, $i = 1, 2, \ldots, m - n + 1$;
- `Call Alg(Input,Output)` – Call the algorithm or procedure `Alg` with input `Input` and output `Output`.

Let us denote arbitrary non-oriented graph with N nodes and M edges and without multiple edges as $G(N, M)$. Several approaches can be used for the random graphs generation. First and most inefficient approach (but still widely used) is the *trial method*. A random graph with given number of nodes (and possibly edges) is generated and then its properties are checked to answer the given limitations. As the number of possible $G(N, N-1)$ graphs (with all possible enumerations of nodes) is

$$S = \left(\begin{array}{c} \frac{N(N-1)}{2} \\ N-1 \end{array} \right) \frac{N(N-1)}{2}! = \frac{\left[\frac{N(N-1)}{2}! \right]^2}{(N-1)! \left[\frac{N(N-1)}{2} - N + 1 \right]!}, \tag{1}$$

while the total number of all N-node trees (with all possible enumerations of nodes also) is only N^{N-1}, the trial method is obviously ineffective for RT generation.

Next basic approach is the "sequential growth": each new element is added to the generated graph without violating the limitation and with guaranteeing its properties. Once more there are two possibilities: either to use trial or "smart" method on each step. We use the sampling without repetition for trial method.

As a "smart" method we use the *method of admissible choice (MAC)*. This is the method in which on each step the choice is done only from the set of elements that 1) keeps the graph in a given class and 2) does not allow to break any limitation. Let us denote:

- X_i – set of nodes in the generated graph on the i-th step;
- E_i – set of edges in the generated graph on the i-th step;
- A_i – set of edges that are allowed to be added to the generated graph on the i-th step;
- In_i – set of edges that are to be added to A_i before the $(i+1)$-st step;
- Ex_i – set of edges that are to be excluded from A_i before the $(i+1)$-st step;
- v_i – i-th node (vertex) of a graph;
- e_{ij} – edge that connects v_i and v_j;
- $X(G), E(G)$ – sets of nodes and edges of the graph G;
- $C(N)$ – the complete graph with N nodes.

Thus, $A_{i+1} = A_i \setminus Ex_i \cup In_i$. Some limitations can lead to the empty A_{i+1}. In this case the rollback is needed by one step. If e_{st} is the last edge chosen before the deadlock then it transfers from A_i to Ex_i.

The simple example is to generate random trees in the case of equal probability of an edge existence. If the trial method is used, then after each new edge selection the test is done if addition of this edge makes a cycle or multi-edge in a tree. If yes then the edge is rejected and choice repeats. If MAC is used then first we choose uniformly any node as a root, let it be v_f. Thus

$$A_1 = \{e_{f\,1}, e_{f\,2}, \ldots, e_{f\,f-1}, e_{f\,f+1}, \ldots, e_{f\,N}\}. \tag{2}$$

If on the i-th step the edge e_{st} is selected (t-th node is new) then

$$Ex_i = \{e_{ct} | v_c \in X_i\}, \quad In_i = \{e_{td} | v_d \in \bar{X}_i\}. \tag{3}$$

3 Algorithms

First we formally describe the algorithm for generating random trees sketched above and then discuss about more complex cases of random trees with limitations.

3.1 The Algorithm for Generating Random N-Vertex Trees

For generating random N-vertex trees (the isomorphism is possible) the following fast algorithm is offered:

Fast Random Tree Generating Algorithm.[1]
Input: N is the number of nodes.
Output: $OU[1..N-1]$ and $IO[1..N-1]$ are the vectors of numbers starting and ending nodes for the resulting tree edges (i-th edge connects node OU_i with IO_i).

[1] Some algorithms discussed in this paper including this one were first programmed for the application package Graph-ES, see [4,5,6]

```
1. for i:=1 to N do
2.   I[i]:=i;                \\ Initial vector of node numbers
3. endfor;
4. l:=random(1,N);           \\ Choose the root
5. K:=I[N]; I[N]:=I[l]; I[l]:=K;\\ Replacement with the last node
6. for m:=1 to N-1 do
7.   t:=random(1,m);         \\ New node
8.   s:=random(m+1,N);       \\ Node of "growth"
9.   OU[m]:=I[t]; IO[m]:=I[s]; \\ Edge registration
10.  K:=I[t]; I[t]:=I[N-m];  \\ Replacement: from N-m+1 to N -
     I[N-m]:=K;              \\ nodes that are in a tree already
11.endfor.
```

Remark: It is clear that this algorithm needs only $2N-1$ calls to the random number generator and uses the minimum memory ($N+4$ integers for the case of equal probabilities and $N+1$ reals additionally in the case of different probabilities of edge existence).

The following theorem is proved for this algorithm.

Theorem 1. *Fast Random Tree Generating Algorithm creates any of N-vertex tree with equal probability $1/N^{N-1}$.*

Proof is rather obvious but tiresome. At first the equal probability of entire results is proved and then it is proved that the number of different results is equal to N^{N-1}. As it is the well-known number for different N-node trees (all renumbering included), the attainability is proved also. For the complete proof, refer to [7].

3.2 Generating Trees with the Restriction on a Maximum Node Degree

Let us have the limitation on the maximum degree of nodes in a tree: $deg(v_i) \leq Deg$. This leads to the new definition of Ex_i in comparison with the previous case:

$$Ex_i = \begin{cases} \{e_{ct} | v_c \in X_i\} & \text{if } deg(v_s) \leq Deg, \\ \{e_{ct} | v_c \in X_i\} \bigcup \{e_{sd} | v_d \in \bar{X}_i\} & \text{otherwise.} \end{cases} \quad (4)$$

The addition of new edges is similar to the case of the unconditional random tree generation, but the set of nodes that *are in the tree already* is divided into two parts: those with maximum degree (N_m) and those with degrees under it ($N_<$). New edge must be chosen to connect some free node with the one in $N_<$. The algorithm looks as follows:

Algorithm for Generating Trees with Limited Node Degrees (GTD).

Input: N is the number of nodes; Deg is the possible maximum degree of nodes. *Output*: $OU[1..N-1]$ and $IO[1..N-1]$ are the vectors of starting and ending nodes for the resulting tree edges, respectively (i-th edge connect node OU_i with IO_i), $Degrees(1 \times N)$ is the vector for the degrees of nodes.

```
1. for i:=1 to N do
2.   I[i]:=i; Degree[i]:=0;   \\Initial vectors
3. endfor;
4. l:=random(1,N);            \\ Choose the root
5. R:=I[N];I[N]:=I[l];I[l]:=R;\\ Replacement with the last element
6. m:=1;                 \\ current numbers of nodes in a tree and
5. k:=0;                 \\ nodes restricted for further choice
8. while m<N do               \\ Main cycle
9.   s:=random(N-m+1,N-k);    \\ For admissible node in a tree
10.  t:=random(1,N-m);        \\ For new node
11.  IO[m]:=I[t]; OU[m]:=I[s];\\ Registration
12.  INC(Degree[I[t]]); INC(Degree[I[s]]);
    \\ Now let's Check for the maximum possible degree
    \\ If YES then restrict the node for future choice
13.  if Degree[I[s]]=Deg then
14.    I[s]:=I[N-k]; INC(k);
15.  endif;
16.  R:=I[t];      \\ Rearrange the working vector: from N-k+1 to
     I[t]:=I[N-m]; \\ N - nodes restricted for further selection
     I[N-m]:=R;
17.  INC(m);                  \\ Working step is done
18.endwhile.
```

Remark: There cannot be a deadlock in this algorithm: as $2 \leq Deg \leq N-1$, the subset $X_i^* = \{v_j | v_j \in X_i \& deg(v_j) < Deg\}$ is always non-empty (at least the last node added to the generated tree always has degree one). And A_{i+1} includes all the edges that connects these nodes with free nodes.

From this we can simply construct the algorithm for generating unbalanced binary tree (B-tree) with the number of nodes $N > 4$. Note that node degree in B-tree is limited to 3. So we can use the following generating scheme:

Algorithm for Generating B-trees.
Input: N is the number of nodes.
Output: $OU[1..N-1]$ and $IO[1..N-1]$ are the vectors of numbers starting and ending nodes of the resulting tree edges (i-th edge connect node OU_i with IO_i), *Root* is the number of root node.

```
1.  N1:=random(1,N+1); N2:=N-N1+1;
2.  Call GTD(N1,3,OU1,IO1,Degree1);
3.  Call GTD(N2,3,OU2,IO2,Degree2);
4.  Using vectors Degree1 and Degree2 the nodes s and t are chosen
    from nodes with degree 1 in both trees, respectively;
5.  Renumbering of nodes: the node s becomes node number N1 in
    first tree, the node t becomes node number N1 in second tree.
6.  Root:=N1;                      \\ Root is assigned
7.  for i:=1 to N1-1 do
```

```
8.   IO[i]:=IO1[i];OU[i]:=OU1[i]; \\ First tree is the left part
9.  endfor;
10. for i:=N1 to N1+N2-1 do       \\ Second tree is the right part
11.  IO[i]:=IO2[i]+N1-1; OU[i]:=OU2[i]+N1-1;
12. endfor.
```

Remark: If we assume the possibility of one-side B-tree then $N2$ can be 0. In this case we simply generate one tree with restriction $Deg = 3$ and then choose a root from all nodes with a degree of 1 or 2.

3.3 Generating Trees with Given Node Degrees

Now let us have the strict values for $deg(v_i)$ in the resulting tree. This is more complicated case, because there can be a deadlock in direct use of the basic algorithm. In fact, it is enough that $\sum_j deg(v_j|v_j \in X_i) = 2(i-1)$. For example, if some v_s and v_t both have degree one where v_s is selected as a root on the preliminary step, and e_{st} is the first choice, then we have the deadlock on the very first step of the algorithm. To avoid the deadlock we employ the following rule: each time when we have $\left(\sum_j deg(v_j|v_j \in X_i) = 2(i-1)-1\right) \&(i < N-2)$ we add all edges that connects nodes from X_i with free nodes with prescribed degree one to the Ex_i.

Let us prove the correctness of this rule based on the proof by contradiction. Assume that $\sum_j deg(v_j|v_j \in X_i) = 2(i-1)-1$. The deadlock will take place if and only if next node has degree one. Assume that all free nodes (their number is $N-i$) have the prescribed degree one. So the total sum of degrees by all nodes is $2(i-1)-1+N-i = N+i-3$ while for a tree it must be $2(N-1)$. We have a contradiction, so our rule is proved to be correct.

4 Generating Trees with Different Probabilities of Edge Existence

This case differs from the previous one only in the procedure of a new edge choice. In the case of equal probability the set A_i is virtual and choice is made between nodes, and now we need to have it in some form that allows us to make choice of new edge as a numbered pair of nodes. Note that in general case the sum of probabilities of the edge existence is not equal to 1 as these events are not alternative. Thus for the choice of *one* edge from the set we use normalized probabilities $p_i^* = p_i / \sum_{j=1}^{n} p_j$.

Let us denote $\frac{N(N-1)}{2}$ as n. In hereafter we use the presentation of undirected graph with N nodes by the up-diagonal part of its adjacency matrix VV unfolded into a vector of the length n. We denote this vector as $D(G)$ or simply D.

Obviously there is one-to-one correspondence between any element VV_{ij} and some element D_l. Let us derive a formula for l through i, j and N.

Each element VV_{ij} has $i-1$ rows above it and k-th row contains $N-k$ elements that belong to the up-diagonal part. The i-th row contains $j-i-1$ elements to the left of VV_{ij} which belong to this up-diagonal part. Thus

$$l = [(N-1)+(N-2)+\ldots+(N-i+1)]+j-i = \frac{(2N-i)(i-1)}{2}+j-i. \quad (5)$$

To obtain i and j from the values of l and N the following primitive algorithm is provided.

```
1. procedure RowCol(N,l : integer; var Row,Col: integer);
2. var m : integer;
3. begin i:=1;
4.   while (i<N) and (m>0) do
5.     m:=m-N+i; INC(i);
6.   endwhile;
7.   Row:=i-1;
8.   Col:=N+m;
9. end RowCol;
```

Note that if we don't need to save the computer memory it is simpler to use the array of pairs (i, j) instead of transformation described above. Now we go on to the algorithms.

As any node inevitably belongs to the spanning tree, we can choose any one as a root on the preliminary step with equal probability. Then we construct the initial list (vector) of possible edges as list of edges incident to the root.

It's simple to obtain the maximum possible length L_m of the list of possible edges: recollect that with each new i-th edge chosen we eliminate from it $i-1$ edges that connected new node with the nodes from X_i and include $N-i$ edges that connect new node with the nodes from $\bar{X}_i$. Thus, after completion the i-th step we have

$$L = (N-1)-1+(N-2)-2+(N-3)-3+\ldots-(i-1)+(N-i) = (N-i)i, \quad (6)$$

and $\lceil L_m = N^2/4 \rceil$. Any additional limitation will only decrease this number.

The pseudo code of the algorithm looks as follows.

Algorithm for Generating Tree with Different Probabilities of Edge Existence and Limited Maximum Node Degree.

Input: $N > 1$ is the number of nodes, $Prob[1..n]$ is the vector of probabilities of edges existence (this vector corresponds to the $\|\varepsilon_{ij} = P(e_{ij}\text{exists})\|$), Deg is the possible maximum node degree.

Output: $OU[1..N-1]$ and $IO[1..N-1]$ are the vectors of the resulting tree edges starting and ending nodes numbers (i-th edge connect node OU_i with IO_i), $Degrees(1 \times N)$ is the vector of the nodes degrees.

Working arrays and variables: $X[1..N-1]$ is the vector for numbers of nodes that are in a generated tree already, $W[1..K]$, $P[1..K]$ – vectors of numbers of edges

admissible for choice on a step and their normalized probabilities, respectively. M is current number of nodes that are in a tree already and K is a number of edges that are admissible for choice on a current step.

```
1.  n:=N*(N-1)/2;                \\ Whole number of possible edges
    \\ Initialization of the vector for edge numbers
2.  for i:=1 to n do I[i]:=i; endfor;
    \\ Initialization of the vector for node degrees
3.  for i:=1 to N do Degree[i]:=0; endfor;
4.  l:=random(P,1,n); X[1]:=l;  \\ Root is selected
5.  S:=0.0; j:=0; M:=1;  \\ M - current number of selected nodes,
    K:=N-1;              \\ K - current number of free nodes
    \\ Now let's obtain the vector for admissible edges
    \\ and Sum of probabilities for future normalization
6.  for v:=1 to l-1,l+1 to N do
7.    INC(j); W[j]:=I[InUpDiag(j,l)];
8.    S:=S+Prob[InUpDiag(N,v,l)];
9.  endfor;
10. for i:=1 to N-1 do           \\ Main cycle
      \\ Probabilities normalization
11.   for v:=1 to K do P[v]:=P[v]/S; endfor;
12.   j:=W[random(P,1,m)];INC(M);OldS:=S; \\ New edge is selected
      \\ Let's obtain the numbers of adjusted nodes and increment
      \\ thier degrees
13.   Call RowCol(N,j,s,t); INC(Degree[s]); INC(Degree[t]);
      \\ Now let us define which node is new one
14.   w:=FindPlace(s,X,OK);
15.   if OK then NewNode:=t else NewNode:=s; s:=t; endif;
      \\ s is the "old" node in the selected edge
16.   X[M]:=NewNode;             \\ Registration of a new node
17.   if Degree[s]=Deg then      \\ Check for maximum degree
18.     for t:=1 to N do
    \\ All free edges connected with a node with maximum possible
    \\ degree must be removed from the set of admissible
19.       w:=FindPlace(t,X,OK);
20.       if not OK then
            q:=InUpDiag(N,t,s); S:=S-P[q]; remove(q,W); DEC(K);
21.   endif; endfor; endif;
22. \\ Now let us exclude all edge that connects new node with
    \\ old ones from the set of admissible and include edges that
    \\ connect new node with free ones into it
23.   SS:=0.0;
24.   for v:=1 to NewNode-1,NewNode+1 to N do
25.     t:=InUpDiag(X[v],NewNode); w:=FindPlace(t,W,OK);
26.     if OK then S:=S-P[w]; remove(t,W); DEC(K);
27.     else SS:=SS+Prob[t]; INC(K); W[K]:=t;
```

```
28.   endif endfor;
    \\ Let us recalculate a sum for normalization
29.   S:=SS+S*OldS;
30. endfor.
```

The procedure `InUpDiag` (line 8) is for expression (5).

5 Conclusion

We have presented general methods and computer algorithms for generating random trees with many essential properties. It is shown that the use of trial method is mostly ineffective. Meanwhile the time complexity of the proposed algorithms based on the consequential tree growth are proven to be effective in terms of the number of operations and the memory use. Random trees are widely used in different applications by themselves, and furthermore can be efficiently employed for generating various connected graphs. As graphs used in applications have different limitations, their spanning trees must satisfy them too. Our basic algorithms allow those. In the future we present algorithms for generating connected graphs and graphs "similar to real nets." The last task assumes definitions of some special limitations that are typical for such graphs.

References

1. M. Doar, Multicast in the ATM environment. PhD thesis, Cambridge Univ., Computer Lab., September 1993.
2. M. Doar, "A Better Mode for Generating Test Networks," *Proc. Global Telecommunication Conf. GLOBECOM'96*, pp. 86–93, 1996.
3. Chai-Keong Toh, "Performance Evaluation of Crossover Switch Discovery Algorithms for Wireless ATM LANs," *Proc. of INFOCOM'96*, pp. 1380–1387, 1993.
4. A.S. Rodionov, L.N. Postnikova, and O.K. Rodionova, "Program Complex for Graph Generating in the GRAPH-ES Package," *Materials of the Conf. "Computers and System Analysis," Novosibirsk, 1982*, pp. 13–15., 1982. (in Russian)
5. A.S. Rodionov, "Graph Generating in the GRAPH-ES Package," *Materials of the Conf. "Methods and Programs for Solving Optimization Problems on Graphs and Nets," Novosibirsk, 1982, Part 1*, pp. 170–171., 1982. (in Russian)
6. M.I. Netchepurenko, V.K. Popkov, S.M. Mainagashev, etc, *Algorithms and Programs for Solving Optimization Problems on Graphs and Networks*. Novosibirsk, Nauka P.A., 1990, 515 p. (in Russian).
7. G. Omarova, "Program Complex for Graph Generating," *Proc. of Young Scientists Annual Conf., Institute of Computational Mathematics and Mathematical Geophysics of Siberian Division of the Russian Academy of Science, Novosibirsk*, pp. 174–181, 1998. (in Russian)