NISTIR 8066

Improving the Computational Efficiency of the Blitzstein-Diaconis Algorithm for Generating Random Graphs of Prescribed Degree

Elizabeth Moseman

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.8066

National Institute of Standards and Technology
U.S. Department of Commerce

NISTIR 8066

Improving the Computational Efficiency of the Blitzstein-Diaconis Algorithm for Generating Random Graphs of Prescribed Degree

Elizabeth Moseman
Applied and Computational Mathematics Division
Information Technology Laboratory

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.8066

June 2015

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Under Secretary of Commerce for Standards and Technology and Director

Abstract

When generating a random graph, if more structure is desired than is given in the popular Erdős–Renyi model, one method is to generate a degree sequence first then create a graph with this degree sequence. Blitzstein and Diaconis[1] (among others) developed a sequential algorithm to create a random graph from a degree sequence. This algorithm is assured to always terminate in a graph with the desired degree sequence; unfortunately, it is slow. This work focuses on the subroutine of the previous algorithm which determines the candidate edges, improving the runtime of the overall algorithm from $O(mn^2)$ to $O(mn)$.

Key words: random graphs; degree sequence; graph algorithms

Contents

For modeling purposes, one frequently wishes to generate a random graph. Many models for random graph generation, including the frequently used Erdős–Renyi model, will generate a graph on a specified number of vertices, but the degrees of the vertices in the resulting graph will be concentrated among very few values. This makes such models of limited use in modeling real world networks, which often exhibit degree distributions that are not concentrated in the same manner. In an effort to generate random graphs which more closely resemble real world graphs, effort has been made on creating random graphs with a specified degree sequence.

The simplest (and most computationally efficient) algorithm is that of Bayati, Kim and Saberi[2]. In this algorithm, each edge is added sequentially from among the allowable edges. In the analysis of the algorithm, it is shown that this algorithm has running time $O(md_{\max})$ where m is the number of edges and $d_{\max}$ is the maximum degree of any vertex. One of the main drawbacks of this algorithm is that it will not always generate a graph; sometimes, the order of the edge selection results in no edges being allowable before all the necessary edges have been generated. The probability of this occurring is shown to be asymptotically small when the maximum degree is bounded as a function of the total number of edges. However, from the standpoint of actually using such an algorithm, asymptotics are unsatisfying. In addition, this result only holds for graphs where $d_{\max} = O(m^{1/4-\tau})$ with τ a positive constant, and does not hold for the degree sequences that arise in many real world networks.

There are three algorithms that generate random graphs of prescribed degree that ensure successful termination. Mihail and Vishnoi[3] created an algorithm that transforms the problem to the well-studied problem of finding a maximum matching. This is done at the expense of making the problem bigger: the matching problem is performed on graph with $O(n^2)$ vertices, where the original graph had only n.

The most widely used algorithm is that of Gkantsidis, Mihail, and Zegura[4], which is based on a Monte Carlo Markov chain. In it, a graph is created which realizes the degree sequence, and then edge swaps are performed in order to randomize the graph. The main drawback of this method is that there are no results on the mixing time (the number of swaps to perform) in general.

This technical report focuses on the algorithm of Blitzstein and Diaconis[1]. This algorithm also adds edges to the graph sequentially, at each step only a few edges are allowed. More details of the algorithm will be given in the following section. The contribution here is a more efficient method of calculating the candidates in this algorithm, improving the runtime from $O(n^2m)$ to $O(nm)$.

1 The Blitzstein–Diaconis Algorithm

Beginning with a degree sequence, it is graphical if there is some graph which realizes the degree sequence. As edges are added to a graph sequentially, each vertex has a desired degree (the degree given by the degree sequence), a current degree (how many edges already contain that vertex) and a residual degree (the difference between the desired degree and the current degree). The main idea of the Blitzstein–Diaconis algorithm is that if edges are added to the vertex of smallest residual degree, and the remaining residual degree sequence is graphical, then the graph can always be completed with the desired degree sequence. This is codified as Algorithm 1.

In this algorithm and later sections, we use the notation $\ominus_{i,j}d$ to indicate the degree sequence obtained from d by subtracting 1 from d_i and d_j but leaving the other entries unchanged. The

Algorithm 1: Blitzstein–Diaconis algorithm for generating a random graph with a given degree sequence

Input: a graphical degree sequence $d = (d_1, \ldots, d_n)$
Output: the edge list E for a graph with degree sequence d

$E \leftarrow \emptyset$
while $d \neq 0$ **do**
  Choose i with d_i a minimal nonzero entry.
  while $d_i > 0$ **do**
    Compute a candidate list $J = \{j \neq i \mid \{i,j\} \notin E \text{ and } \ominus_{i,j} d \text{ is graphical}\}$.
    Pick $j \in J$ with probability proportional to its degree in d.
    $E \leftarrow E \cup \{\{i,j\}\}$
    $d \leftarrow \ominus_{i,j} d$.
  end
end

largest bottleneck in this algorithm comes from calculating the candidate list, J, before each edge is added. Since this is also where the improvements come in, some time should be spent on this step.

To test for graphicality, Blitzstein and Diaconis use the Erdős–Gallai conditions for graphicality.

Definition 1.1. *For a monotone decreasing sequence $d = (d_1, \ldots, d_n)$, the kth Erdős–Gallai condition is*

$$k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i) - \sum_{i=1}^{k} d_i \geq 0. \tag{1}$$

This definition differs from the standard in the ordering of the terms, but this usage will make later definitions easier. In general, an integer sequence is graphical if it satisfies all of the Erdős–Gallai conditions for $k = 1, \ldots n$.

In their algorithm, Blitzstein and Diaconis utilize a $\Theta(n)$ subroutine to test all the Erdős–Gallai conditions for each candidate, one at a time. Since there are n potential candidates, the running time for generating the candidate list is $O(n^2)$ and since a candidate list is generated for every edge this makes the total running time $O(n^2 m)$. Anecdotally, Blitzstein and Diaconis note that it took 13 seconds to generate graphs for a specified 33 node degree sequence on a 1.33 GHz machine when the algorithm was coded in R. Matlab[1] code for the same procedure took 0.05 seconds on the same 33 node degree sequence.

2 Creating the Candidate List Efficiently

To improve the running time of the candidate calculations, we need to observe that the values obtained from calculations of the Erdős–Gallai conditions for d will be very similar to the values

[1]Certain commercial equipment, instruments, or materials are identified in this paper to foster understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

of the same condition for $\ominus_{i,j}d$. We will see later that the values differ by at most two, and it is easy to tell by exactly how much they differ. But first, we need to address the issue of order in the degree sequence.

In testing the Erdős–Gallai conditions efficiently, we assume that the sequence fed into the tester is ordered from greatest to least. This is important because there is a $\Theta(n \log n)$ lower bound on a sorting algorithms when nothing is known of the sequence. However, the sorted order of d and the sorted order of $\ominus_{i,j}d$ may differ. Assume that d is in sorted order. Notice that trailing zeros do not effect whether d is graphical or not, so we assume that n is the number of nonzero entries and thus d_n is a minimal nonzero entry.

Definition 2.1. *Let d be a sorted, graphical degree sequence with no trailing zeros. We say that $j \leq n-1$ is a* **candidate** *if $\ominus_{j,n}d$ is graphical.*

We first address the case where $\ominus_{j,n}d$ is not in sorted order. Notice that since there are no entries less than d_n, decrementing to $d_n - 1$ makes no difference to the sorting, so the only possibility is $d_j - 1 < d_{j+1}$. Since $d_j \geq d_{j+1}$, this forces $d_j = d_{j+1}$. If, however, $j = n-1$, then even $d_j = d_{j+1} = d_n$ will not cause $\ominus_{j,n}d$ to be unsorted.

Lemma 2.2. *If $d_j = d_{j+1}$ with $j < n-1$, then j is a candidate if and only if $j+1$ is a candidate.*

Proof. This is a necessary consequence of the sorted order of $\ominus_{j,n}d$. In particular, in sorted order $\ominus_{j,n}d$ and $\ominus_{j+1,n}d$ are the same sequence so if the common sequence is graphical then both are candidates and otherwise both are not. □

The following theorem addresses the candidacy of the remaining j values.

Theorem 2.3. *Let $d = (d_1, \ldots, d_n)$ be a sorted, graphical degree sequence with $d_n > 0$. Fix k with $1 \leq k < n$ and $1 \leq j < n$. Define $\tilde{d} = \ominus_{j,n}d$,*

$$e_k = k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i) - \sum_{i=1}^{k} d_i \qquad \text{and} \tag{2}$$

$$\tilde{e}_k = k(k-1) + \sum_{i=k+1}^{n} \min(k, \tilde{d}_i) - \sum_{i=1}^{k} \tilde{d}_i. \tag{3}$$

Suppose, without losing any generality via Lemma 2.2, $d_j > d_{j+1}$ or $j = n-1$ so that $\tilde{d}$ is also sorted. Then

$$\tilde{e}_k - e_k = \begin{cases} 1 & j \leq k \text{ and } d_n > k \\ 0 & j \leq k \text{ and } d_n \leq k \\ 0 & j > k \text{ and } d_n > k \\ -1 & j > k \text{ and } d_j > k \geq d_n \\ -2 & j > k \text{ and } k \geq d_j \end{cases} \tag{4}$$

Proof. We observe that $\tilde{e}_k$ and e_k are very similar. We consider two cases.

Case $j \leq k$: Then

$$\tilde{e}_k - e_k = \min(k, d_n - 1) - \min(k, d_n) + 1 \tag{5}$$

where the 1 is due to the change in d_j to $d_j - 1$. If $d_n > k$, we have $\min(k, d_n - 1) = \min(k, d_n) = k$ so that $\tilde{e}_k - e_k = 1$. Otherwise, $\min(k, d_n - 1) = d_n - 1$ and $\min(k, d_n) = d_n$ so that $\tilde{e}_k - e_k = 0$.

Case $j > k$: We see that

$$\tilde{e}_k - e_k = \min(k, d_n - 1) - \min(k, d_n) + \min(k, d_j - 1) - \min(k, d_j). \tag{6}$$

When $d_n > k$, the sorted nature of d ensures $d_j \geq d_n > k$ so that $\min(k, d_n - 1) = \min(k, d_n) = \min(k, d_j - 1) = \min(k, d_j) = k$ and $\tilde{e}_k - e_k = 0$. When $d_j > k \geq d_n$, we have $\min(k, d_n - 1) = d_n - 1$, $\min(k, d_n) = d_n$, and $\min(k, d_j - 1) = \min(k, d_j) = k$ so that $\tilde{e}_k - e_k = -1$. Finally, when $k \geq d_j \geq d_n$, we have $\min(k, d_j - 1) = d_j - 1$ and $\min(k, d_j) = d_j$ so that $\tilde{e}_k - e_k = -2$. □

Algorithm 2: Calculating the max candidate.

Input: a sorted, graphical degree sequence $d = (d_1, \ldots, d_n)$, with $d_n > 0$
Output: a max candidate j' with $\ominus_{j,n} d$ graphical for any $j \leq j'$

```
Calculate e_k for k = 1, ... n
for k = 1 to n do
    if e_k ≥ 2 then
        j_k ← n
    else if e_k = 1 then
        j_k ← max(k, min(j | d_j ≤ k))
    else if e_k = 0 then
        if d_n ≤ k then
            j_k ← k
        else
            j_k ← max(k, min(j | d_j ≤ k))
        end
    end
end
j' ← min_k(j_k)
while d_{j'} = d_{j'+1} do
    j' ← j' − 1
end
```

Each condition will have a maximum index j so that $\ominus_{j,n} d$ passes that condition, and the minimum of these indices will be the maximum candidate j'. Then the candidates are $J = \{j \leq j' \mid \{j, n\} \notin E \text{ and } d_j = d_{j'+1}\}$. This process is shown in Algorithm 2. This is then a $\Theta(n)$ process for calculating the candidate list. If we use the theorem to adjust the e_k once a candidate has been selected, the constant may be improved but we cannot calculate e_k in better than linear time because all of them may need to be updated.

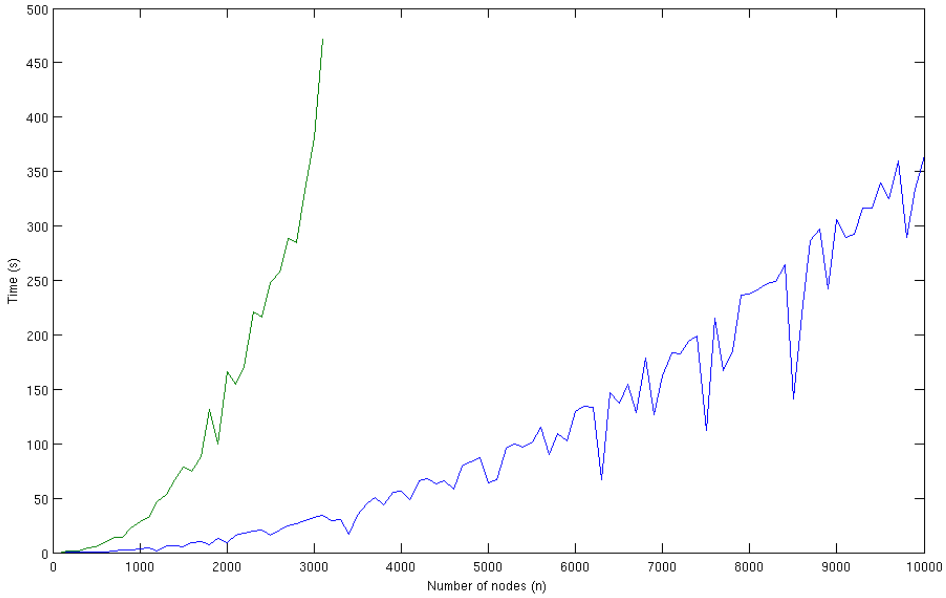

Figure 1: Running time for trials with the unaltered (green) and altered (blue) algorithms. Average of 10 trials at each n. As a reference, a single trial with the unaltered algorithm with $n = 10000$ ran in 5300 s, or approximately 88 minutes.

3 Impact

To show the practicality of the algorithm, degree sequences were chosen to follow a power law, so that the probability that a given vertex has degree k is proportional to $k^{-\beta}$ for some constant β. We chose to work with $\beta = 2.1$ since many graphs found in practice seem to have this exponent. Elementary calculations yield the results that the expected degree on these graphs is $\frac{\beta-1}{2-\beta}$ and the expected maximum degree is at least $n^{1/(\beta-1)}$ and thus the Bayati, Kim and Saberi algorithm [2] will converge for $\beta > 5$, but not in the range that we are interested in. Figure 1 shows the average of 10 running times for random degree sequences generated according to a power law with $\beta = 2.1$.[2]

[2]Code was generated in Matlab and run on a 2.4GHz AMD Athlon 64 processor.

References

[1] J. Blitzstein and P. Diaconis (2010). "A sequential importance sampling algorithm for generating random graphs with prescribed degrees". *Internet Mathematics* **6** (4), 489. http://dx.doi.org/10.1080/15427951.2010.557277.

[2] M. Bayati, J. H. Kim, and A. Saberi (2010). "A sequential algorithm for generating random graphs". *Algorithmica* **58** (4), 860–910. http://dx.doi.org/10.1007/s00453-009-9340-1.

[3] M. Mihail and N. Vishnoi (2002). "On generating graphs with prescribed vertex degrees for complex network modelling". In ARACNE.

[4] C. Gkantsidis, M. Mihail, and E. Zegura (2003). "The markov chain simulation method for generating connected power law random graphs". In ALENEX.