Fast generation of spatially embedded random networks

Eric Parsonage and Matthew Roughan
eric@eparsonage.com
matthew.roughan@adelaide.edu.au

UoA

July 18, 2015
Random Graphs

- Graph: $G(N, E)$
  - $N = \text{set of nodes (vertices)}$
  - $E = \text{set of edges (links)}$

- Motivation
  - simulations to test new network protocols
  - models for structured connections in an epidemic
  - ...

- Canonical example: Gilbert-Erdős-Rényi (GER) [1, 2]
  - two cases:
    - $G(n, e)$: put $e$ edges on random node pairs ($n$ nodes)
    - $G(n, p)$: put edge between each node pair with probability $p$
SERNs
Spatially Embedded Random Networks

- GER is too simple
  - many ways to generalise
- One approach is a SERN
  - generate random points in some metric space
  - generate links between node pairs independently with probability $p_{ij}$
  
  $$p_{i,j} = f\left(d(n_i, n_k)\right)$$

  - NB: links are not independent, because of distance dependencies
- Motivation:
  - real actors are often in some space
  - often some “cost” to a link that depends on distance
    - e.g., computer network, you have to run a cable
    - e.g., epidemic, spread of infection requires transport of vector
SERN variations

- Many choices for metric space and point generation
  - typically points uniformly distributed over a unit square
  - many obvious generalisations of space and measure

- Many choices for distance functions
  common cases:
  - Random Plane Networks [3]:
    \[ f(d) = I(d \leq r) \]
  - Waxman [4]:
    \[ f(d) = qe^{-sd} \]
Simulation

- Uses for these graphs often require simulation
  - for testing protocols
  - in estimation, e.g., ABC
- Often (in the past)
  - simulation toolkits couldn’t handle huge networks
  - we didn’t have large-scale data anyway
  
  **but neither of these features holds anymore**
- I want to be able to generate graphs
  - with thousands to millions or even billions of nodes
  - I want to generate large numbers of them
- Most existing graph generation toolkits (for cases I deal with) use $O(n^2)$ algorithms
  - usually in time
  - sometimes also in memory

  **but most real graphs are sparse** $O(e) \ll O(n^2)$
The history of the Gilbert-Erdös-Rényi (GER) is illustrative

- Almost all code for generating GERs
  - $O(n^2)$ Bernoulli trials [5, 6, 7]

Algorithm 1: Naive Waxman generation

```
Input: n, q, s  // parameters of the graph
Output: E = set of edges
1 for i = 1..n do
2   for j = i+1..n do
3       calculate $d_{ij}$
4       calculate $p_{ij} = q \exp(-sd_{ij})$
5       generate $r \sim U[0, 1]$
6       if $r \leq p_{ij}$ then
7           add (i, j) to E
8       end
9   end
10 end
```

- In 2005 Batagelj and Brandes [8] came up with an $O(e)$ algorithm
- Only two sets of software (I can find) use this: NetworkX and igraph
  - None have better than $O(n^2)$ for a SERN [9]
Batagelj and Brandes algorithm

Their approach is based on the following insight

- Think of the possible edges in a list
  - order doesn’t matter
- The actual edges are selected (notionally) by Bernoulli trials
  - we can instead just do geometric jumps between edges
- Just requires the idea of homogeneous memoryless renewal process

But it doesn’t work for a SERN because not all links are equal

- we might be able to transform, but
- we don’t want to even calculate all of the distances!
Fast Waxman 1

We can apply the same idea as follows

\[ p_{ij} = qe^{-sd_{ij}} \leq q \]

Hence, the GER random graph \( G(n, q) \) provides an “upper bound” graph

- that suggests an algorithm

```
Input: n, q, s  // parameters of the graph
Output: E = set of edges
1 Construct a GER(n,q) graph \( G_1(N, E_1) \) using geometric jumps
2 forall the \((i, j) \in E_1\) do
3     calculate \(d_{ij}\)
4     calculate \(p_{ij} = \exp(-sd_{ij})\)
5     generate \(r \sim U[0, 1]\)
6     if \(r \leq p_{ij}\) then
7         add \((i, j)\) to \(E\)
8     end
9 end
```

**Algorithm 2:** \( q \)-jumping
How good is it?

- Algorithm complexity is $O(e_1)$ where $e_1$ is edges in the $GER(n, q)$
  - efficiency depends on how close $e$ is to $e_1$

\[
\begin{align*}
\mathbb{E}[e_1] &= \frac{n\bar{k}}{2} \\
\mathbb{E}[e] &= \frac{n\bar{k}\tilde{G}(s)}{2}
\end{align*}
\]

- $\bar{k}$ is average node degree
- $\tilde{G}(s)$ is Laplace transform of PDF of the line-picking problem

  - so we have an $O(e)$ algorithm, but how close to optimal optimal?

- Efficiency depends on $\tilde{G}(s)$
  - $\tilde{G}(0) = 1$
  - $\tilde{G}(s) \to 0$ for large $s$
  - efficiency is its good for small $s$
  - but for large $s$ we have $\mathbb{E}[e_1] = \mathbb{E}[e]/\tilde{G}(s)$
What can we do for large $s$

Consider breaking the region into $M^2$ “buckets”, e.g.,

We can put a lower bound $D_{IJ} \leq d_{ij}$ on the distance between nodes $i$ and $j$ in buckets $I$ and $J$, respectively.
GER skipping algorithm didn’t depend on the order of the potential edges, or even that we generated them all at once

Group potential edges into bucket-pairs \((I, J)\)

Perform skipping to create

\[
GER(n_{IJ}, q \exp(-sD_{IJ}))
\]

upper-bound subgraph for each bucket pair

Calculate the exact distance, and filter with probability

\[
p_{ij} = \exp(-s(d_{ij} - D_{IJ}))
\]

Put all the edges back together
Coding

- This isn’t quite trivial
  - the time to create a link in this code isn’t much longer than the time to access the relevant memory
  - buckets can’t be calculated on the fly
  - can’t sort the points into buckets (sorting $O(n \log n)$)
  - controlling the memory allocated has to be done carefully

- The algorithm parallelises
  - only other similar example on GER [10]
  - we have a multi-thread implementation
  - its hard to avoid blocking, so speedup limited
Results: small $s = 0.1$, fixed $\bar{k}$

![Graph showing time vs. n for different M values and methods: naive, matlab.](image-url)
Results: large $s = 10$, fixed $\bar{k}$
Results: fixed $n = 1,000,000$
Results: fixed $n = 1,000,000$
Conclusion

- Random graphs
  - current generation techniques often naive
  - we can do better

- SERNs
  - showed how to do Waxman
  - not too hard to see how to generalise to many other cases

- There are some problems
  - what about non-convex regions
  - what about non-monotonic distance functions


