Generating Random Graphs without Short Cycles

Mohsen Bayati, Andrea Montanari, Amin Saberi

Anmol Kagrecha
Karan Taneja

April 4, 2019

Introduction

Motivation

- Low Density Parity Check (LDPC) codes are a class of capacity achieving codes.
- LDPC codes can be represented as a bipartite graph between parity check nodes and variable nodes.

Motivation

- Low Density Parity Check (LDPC) codes are a class of capacity achieving codes.
- LDPC codes can be represented as a bipartite graph between parity check nodes and variable nodes.
- It is known that short cycles in this graph degrades performance.
- This paper proposes an algorithm for generating random graphs without short cycles and provides performance guarantees for the algorithm.

Motivation

- Low Density Parity Check (LDPC) codes are a class of capacity achieving codes.
- LDPC codes can be represented as a bipartite graph between parity check nodes and variable nodes.
- It is known that short cycles in this graph degrades performance.
- This paper proposes an algorithm for generating random graphs without short cycles and provides performance guarantees for the algorithm.
- The authors also propose an algorithm for generating random bipartite graphs.

Terminology and Notation

- Girth of a graph G is the length of the shortest cycle.

Terminology and Notation

- Girth of a graph G is the length of the shortest cycle.
- $G_{n,m}$ is the set of all graphs with n nodes and m edges.
- $G_{n,m,k}$ is the set of all graphs with n nodes, m edges and girth greater than k.

Terminology and Notation

- Girth of a graph G is the length of the shortest cycle.
- $G_{n,m}$ is the set of all graphs with n nodes and m edges.
- $G_{n,m,k}$ is the set of all graphs with n nodes, m edges and girth greater than k.
- K_n is a complete graph with vertex set $[n] = 1, 2, \ldots, n$.
- (ij) is an undirected edge between distinct nodes i and j.

Algorithm

Basic Idea

- Input is n, m, k
- Output is an element of $G_{n,m,k}$ or ‘FAIL’.

Basic Idea

- Input is n, m, k
- Output is an element of $G_{n,m,k}$ or 'FAIL'.
- Initalize with an empty graph G_0 on n vertices.
- G_t is the graph at time t.

Basic Idea

- Input is n, m, k
- Output is an element of $G_{n,m,k}$ or 'FAIL'.
- Initalize with an empty graph G_0 on n vertices.
- G_t is the graph at time t.
- $Q(G_t)$ is set of edges (ij) that don't create a cycle of length $\leq k$ when added to G_t.

Basic Idea

- Input is n, m, k
- Output is an element of $G_{n,m,k}$ or 'FAIL'.
- Initalize with an empty graph G_0 on n vertices.
- G_t is the graph at time t.
- $Q(G_t)$ is set of edges (ij) that don't create a cycle of length $\leq k$ when added to G_t.
- $G_{t+1} = G_t \cup (ij)$ only if $(ij) \in Q(G_t)$.
- If $Q(G_t)$ is empty for $t < m$, report 'FAIL'.

What do we want?

- A random sample from $G_{n,m,k}$.

What do we want?

- A random sample from $G_{n,m,k}$.
- More precisely, a uniformly random sample from $G_{n,m,k}$.

Selecting an edge (ij) from $Q(G_t)$

- Probability of selecting edge (ij) at time t is

$$p(ij|G_t) = \frac{1}{Z(G_t)} e^{-E_k(G_t, ij)}$$

- $Z(G_t)$ is a normalizing term.

Selecting an edge (ij) from $Q(G_t)$

- $E_k(G_t, ij)$ is given by

$$E_k(G_t, ij) = \sum_{r=3}^{k} \sum_{l=0}^{r-2} N_{r,l}^{G_t,ij} q_t^{r-1-l}$$

- $N_{r,l}^{G_t,ij}$ is the number of simple cycles in K_n, have length r, includes (ij) and include exactly l edges of G_t.
- $q_t = \frac{m-t}{{}^{n}C_2 - t}$

Algorithm

Algorithm 1 `RandGraph`

```
Input: n, m, k
Output: An element of G_{n,m,k} or FAIL
set G_0 to be a graph over vertex set [n] and with no edges.
for t = 0 to m − 1 do
    if |Q(G_t)| = 0 then
        stop and return FAIL
    else
        sample an edge (ij) with probability p(ij|G_t).
        set G_{t+1} = G_t ∪ (ij).
    end if
end for
if the algorithm does not FAIL before t = m-1 then
    return G_m
end if
```

Remarks

- By construction, if `RandGraph` outputs a graph G, it will be valid.

Remarks

- By construction, if `RandGraph` outputs a graph G, it will be valid.
- If `RandGraph` outputs FAIL, algorithm is repeated till it produces a graph.
- The probability of failing vanishes asymptotically.

`RandGraph` **Guarantees**

Theorem 1.
For $m = O(n^{1+\alpha})$, $m \geq n$, *and a constant* $k \geq 3$ *such that* $\alpha \leq 1/[2k(k+3)]$, *the failure probability of* `RandGraph` *asymptotically vanishes and the graphs generated by* `RandGraph` *are approximately uniform. In particular,*

$$\mathbb{P}_{RG}(FAIL) = O(n^{-1/2+k(k+3)\alpha})$$

$$d_{TV}(\mathbb{P}_{RG}, \mathbb{P}_U) = O(n^{-1/2+k(k+3)\alpha})$$

where, $\mathbb{P}_U = 1/|G_{n,m,k}| \; \forall \; G \in G_{n,m,k}$ *is the uniform distribution.*

`RandGraph` **Time Complexity**

Theorem 2.
Let n, m, and k satisfy the conditions of Theorem 1. For all n large enough, there exist an implementation of `RandGraph` *that uses asymptotically $O(n^2 m)$ operations in expectation.*

Intuition behind `RandGraph`

Simple `RandGraph`

- Forget about the constraint of not having short cycles! Let us construct a graph by sequentially adding m edges to a empty graph with n nodes, a *simple* `RandGraph`.

Simple `RandGraph`

- Forget about the constraint of not having short cycles! Let us construct a graph by sequentially adding m edges to a empty graph with n nodes, a *simple* `RandGraph`.
- What is the *execution tree* T of simple `RandGraph`? It's a tree that represents the process of adding m edges to an empty graph.
- The root of T, at level 0, corresponds to an empty graph. Level t contains all pairs (G_t, Π_t) where G_t is a graph with t edges and Π_t is the ordering in which t edges were added.

Simple `RandGraph`

- Forget about the constraint of not having short cycles! Let us construct a graph by sequentially adding m edges to a empty graph with n nodes, a *simple* `RandGraph`.
- What is the *execution tree* T of simple `RandGraph`? It's a tree that represents the process of adding m edges to an empty graph.
- The root of T, at level 0, corresponds to an empty graph. Level t contains all pairs (G_t, Π_t) where G_t is a graph with t edges and Π_t is the ordering in which t edges were added.
- Any path from the root to a leaf at level m of T corresponds to one possible way of generating a random graph in $G_{n,m}$.

Naïve Approach

- A *valid* leaf is a leaf node in T that has girth greater than k.
- Each valid leaf corresponds to a graph in $G_{n,m,k}$

Naïve Approach

- A *valid* leaf is a leaf node in T that has girth greater than k.
- Each valid leaf corresponds to a graph in $G_{n,m,k}$
- A naïve approach is to keep generating a simple `RandGraph` till you get a valid leaf.
- When $m = O(n^{1+\alpha})$, fraction of valid leaves is of the order $O(e^{-n^{\alpha}})$.

Naïve Approach

- A *valid* leaf is a leaf node in T that has girth greater than k.
- Each valid leaf corresponds to a graph in $G_{n,m,k}$
- A naïve approach is to keep generating a simple `RandGraph` till you get a valid leaf.
- When $m = O(n^{1+\alpha})$, fraction of valid leaves is of the order $O(e^{-n^{\alpha}})$.
- Naïve approach works well if m = O(n) as a constant fraction of leaves of T are valid.

How to sample G_{t+1}?

- We want to uniformly randomly generate a valid leaf of T.

How to sample G_{t+1}?

- We want to uniformly randomly generate a valid leaf of T.
- `RandomGraph` chooses (ij) with probability proportional to number of valid leaves of T among the descendants of (G_{t+1}, Π_{t+1}) where $G_{t+1} = G_t \cup (ij)$ and $\Pi_{t+1} = \left[\Pi_t \; (ij)\right]$.
- Call this probability $p_{true}(G_{t+1}, \Pi_{t+1})$.

Finding $p_{true}(G_{t+1}, \Pi_{t+1})$

- Let $n_k(G_{t+1}, \Pi_{t+1})$ denote the number of cycles of length at most k in a leaf chosen uniformly at random among descendants of (G_{t+1}, Π_{t+1}) in T.

Finding $p_{true}(G_{t+1}, \Pi_{t+1})$

- Let $n_k(G_{t+1}, \Pi_{t+1})$ denote the number of cycles of length at most k in a leaf chosen uniformly at random among descendants of (G_{t+1}, Π_{t+1}) in T.
- Note that $p_{true}(G_{t+1}, \Pi_{t+1}) = \mathbb{P}(n_k(G_{t+1}, \Pi_{t+1}) = 0)$.

Finding $p_{true}(G_{t+1}, \Pi_{t+1})$

- Let $n_k(G_{t+1}, \Pi_{t+1})$ denote the number of cycles of length at most k in a leaf chosen uniformly at random among descendants of (G_{t+1}, Π_{t+1}) in T.
- Note that $p_{true}(G_{t+1}, \Pi_{t+1}) = \mathbb{P}(n_k(G_{t+1}, \Pi_{t+1}) = 0)$.
- It is known that $n_k(G_{t+1}, \Pi_{t+1})$ is approximately Poisson.
- Hence,

$$\mathbb{P}(n_k(G_{t+1}, \Pi_{t+1}) = 0) \approx \exp(-\mathbb{E}[n_k(G_{t+1}, \Pi_{T+1})])$$

Finding $p_{true}(G_{t+1}, \Pi_{t+1})$

- Let $n_k(G_{t+1}, \Pi_{t+1})$ denote the number of cycles of length at most k in a leaf chosen uniformly at random among descendants of (G_{t+1}, Π_{t+1}) in T.
- Note that $p_{true}(G_{t+1}, \Pi_{t+1}) = \mathbb{P}(n_k(G_{t+1}, \Pi_{t+1}) = 0)$.
- It is known that $n_k(G_{t+1}, \Pi_{t+1})$ is approximately Poisson.
- Hence,

$$\mathbb{P}(n_k(G_{t+1}, \Pi_{t+1}) = 0) \approx \exp(-\mathbb{E}[n_k(G_{t+1}, \Pi_{T+1})])$$

- Finally, we need to control the accumulated error.

$$\prod_{t=0}^{m-1} \frac{p(G_{t+1}, \Pi_{t+1})}{p_{true}(G_{t+1}, \Pi_{t+1})}$$

Proof of Theorem 1

Proof Outline

- Core idea: $\mathbb{P}_{\mathsf{RG}}(G)$ is asymptotically larger than $\mathbb{P}_{\mathsf{U}}(G)$.
- This idea will be formalized in Lemma 1.

Proof Outline

- Core idea: $\mathbb{P}_{\mathsf{RG}}(G)$ is asymptotically larger than $\mathbb{P}_{\mathsf{U}}(G)$.
- This idea will be formalized in Lemma 1.
- Using Lemma 1, Theorem 1 will be proved.

Lemma 1

There exist positive constants c_1 and c_2 such that

$$\mathbb{P}_{\mathsf{RG}}(G) \geq [1 - c_1 n^{-1/2+k(k+3)\alpha}]\mathbb{P}_{\mathsf{U}}(G)$$

for every n, m, k satisfying the conditions of Theorem 1, and all $G \in \mathbb{G}_{n,m,k}$ except for a subset of graphs in $\mathbb{G}_{n,m,k}$ of size $c_2 e^{-n^{k\alpha}} |\mathbb{G}_{n,m,k}|$.

Proof using Lemma 1

- Total variation distance between two probability distributions $\mathbb{P}$ and $\mathbb{Q}$ on a set X is defined by

$$d_{TV}(\mathbb{P}, \mathbb{Q}) := \sup\left\{ |\mathbb{P}(A) - \mathbb{Q}(A)| : A \subset X \right\}$$

- Using triangle inequality, we obtain

$$d_{TV}(\mathbb{P}_{\mathsf{RG}}(G), \mathbb{P}_{\mathsf{U}}(G)) \leq \sum_{G \in \mathbb{G}_{n,m,k}} |\mathbb{P}_{\mathsf{RG}}(G) - \mathbb{P}_{\mathsf{U}}(G)|$$

Proof using Lemma 1

- We'll bound $|\mathbb{P}_{\mathsf{RG}}(G) - \mathbb{P}_{\mathsf{U}}(G)|$ depending on whether $\mathbb{P}_{\mathsf{RG}}(G) \geq \mathbb{P}_{\mathsf{U}}(G)$ or $\mathbb{P}_{\mathsf{RG}}(G) < [1 - c_1 n^{-1/2+k(k+3)\alpha}]\mathbb{P}_{\mathsf{U}}(G)$.

Proof using Lemma 1

- We'll bound $|\mathbb{P}_{\mathsf{RG}}(G) - \mathbb{P}_{\mathsf{U}}(G)|$ depending on whether $\mathbb{P}_{\mathsf{RG}}(G) \geq \mathbb{P}_{\mathsf{U}}(G)$ or $\mathbb{P}_{\mathsf{RG}}(G) < [1 - c_1 n^{-1/2+k(k+3)\alpha}]\mathbb{P}_{\mathsf{U}}(G)$.
- Let $\mathbb{B}_{n,m,k} \subset \mathbb{G}_{n,m,k}$ be the set of all graphs G with $\mathbb{P}_{\mathsf{RG}}(G) < \mathbb{P}_{\mathsf{U}}(G)$.
- Let $\mathbb{D}_{n,m,k} \subseteq \mathbb{B}_{n,m,k}$ be graphs with $\mathbb{P}_{\mathsf{RG}}(G) < [1 - c_1 n^{-1/2+k(k+3)\alpha}]\mathbb{P}_{\mathsf{U}}(G)$

Proof using Lemma 1

- We'll bound $|\mathbb{P}_{\mathsf{RG}}(G) - \mathbb{P}_{\mathsf{U}}(G)|$ depending on whether $\mathbb{P}_{\mathsf{RG}}(G) \geq \mathbb{P}_{\mathsf{U}}(G)$ or $\mathbb{P}_{\mathsf{RG}}(G) < [1 - c_1 n^{-1/2+k(k+3)\alpha}]\mathbb{P}_{\mathsf{U}}(G)$.
- Let $\mathbb{B}_{n,m,k} \subset \mathbb{G}_{n,m,k}$ be the set of all graphs G with $\mathbb{P}_{\mathsf{RG}}(G) < \mathbb{P}_{\mathsf{U}}(G)$.
- Let $\mathbb{D}_{n,m,k} \subseteq \mathbb{B}_{n,m,k}$ be graphs with $\mathbb{P}_{\mathsf{RG}}(G) < [1 - c_1 n^{-1/2+k(k+3)\alpha}]\mathbb{P}_{\mathsf{U}}(G)$
- For ease of notation, we'll drop the subscripts n, m, k in $\mathbb{D}_{n,m,k}, \mathbb{B}_{n,m,k}, \mathbb{G}_{n,m,k}$.

Proof using Lemma 1

- Assuming Lemma 1 holds, $|\mathbb{D}| = c_2 e^{-n^{k\alpha}} |\mathbb{G}|$ and for $G \in \mathbb{B}/\mathbb{D}$

$$|\mathbb{P}_{\mathsf{RG}}(G) - \mathbb{P}_{\mathsf{U}}(G)| = \mathbb{P}_{\mathsf{U}}(G) - \mathbb{P}_{\mathsf{RG}}(G) \leq c_1 n^{-1/2+k(k+3)\alpha} \mathbb{P}_{\mathsf{U}}(G)$$

Proof using Lemma 1

$$
\begin{aligned}
\sum_{G\in\mathbb{G}} |\mathbb{P}_{\mathrm{RG}}(G) - \mathbb{P}_{\mathrm{U}}(G)| &= \sum_{G\in\mathbb{G}} [\mathbb{P}_{\mathrm{RG}}(G) - \mathbb{P}_{\mathrm{U}}(G)] + 2\sum_{G\in\mathbb{B}} |\mathbb{P}_{\mathrm{RG}}(G) - \mathbb{P}_{\mathrm{U}}(G)| \\
&= \sum_{G\in\mathbb{G}} [\mathbb{P}_{\mathrm{RG}}(G) - \mathbb{P}_{\mathrm{U}}(G)] + 2\sum_{G\in\mathbb{B}/D} |\mathbb{P}_{\mathrm{RG}}(G) - \mathbb{P}_{\mathrm{U}}(G)| \\
&\quad + 2\sum_{G\in\mathbb{D}} |\mathbb{P}_{\mathrm{RG}}(G) - \mathbb{P}_{\mathrm{U}}(G)| \\
&\leq \sum_{G\in\mathbb{G}} \mathbb{P}_{\mathrm{RG}}(G) - \sum_{G\in\mathbb{G}} \mathbb{P}_{\mathrm{U}}(G) + 2c_1 n^{-1/2+k(k+3)\alpha} \\
&\quad \times \sum_{G\in\mathbb{B}/\mathbb{D}} \mathbb{P}_{\mathrm{U}}(G) + 4\sum_{G\in\mathbb{D}} \mathbb{P}_{\mathrm{U}}(G) \\
&\leq 1 - \mathbb{P}_{RG}(FAIL) - 1 + 2c_1 n^{-1/2+k(k+3)\alpha} + 4\frac{|\mathbb{D}|}{|\mathbb{G}|} \\
&\leq 2c_1 n^{-1/2+k(k+3)\alpha} + 4c_2 n^{-n^{k\alpha}} - \mathbb{P}_{RG}(FAIL)
\end{aligned}
$$

Finishing the proof

Using lemma 1, we've proved that

$$
\begin{aligned}
d_{TV}(\mathbb{P}_{\mathsf{RG}}(G), \mathbb{P}_{\mathsf{U}}(G)) + \mathbb{P}_{RG}(FAIL) &\leq \sum_{G \in \mathbb{G}} |\mathbb{P}_{\mathsf{RG}}(G) - \mathbb{P}_{\mathsf{U}}(G)| + \mathbb{P}_{RG}(FAIL) \\
&= O(n^{-1/2+k(k+3)\alpha})
\end{aligned}
$$

This finishes our proof.

Proof of Theorem 2

Running Time of $O(n^2 m)$

- We shall define surrogate quantities for probabilities $p(ij|G_t)$ which takes order of n^2 operations for each m. These quantities are efficiently computable using sparse matrix multiplications.
- By definition, $p(ij|G_t)$ is weighted sum over simple cycles.

Running Time of $O(n^2m)$

- We shall define surrogate quantities for probabilities $p(ij|G_t)$ which takes order of n^2 operations for each m. These quantities are efficiently computable using sparse matrix multiplications.
- By definition, $p(ij|G_t)$ is weighted sum over simple cycles.
- It is known that we can count all cycles of a graph via matrix multiplication of its adjacency matrix. They've proved that the contribution of non-simple cycles will be negligible.

Approximate `RandGraph`

- During the execution of `RandGraph`, after adding t edges, let $\mathbf{M}_t$ and $\mathbf{M}_t^{(c)}$ be the adjacency matrices of the partially constructed graph G_t and its complement $G_t^{(c)}$ respectively.

Approximate `RandGraph`

- During the execution of `RandGraph`, after adding t edges, let $\mathbf{M}_t$ and $\mathbf{M}_t^{(c)}$ be the adjacency matrices of the partially constructed graph G_t and its complement $G_t^{(c)}$ respectively.
- In addition, let $\mathbf{Q}_t$ be the adjacency matrix of the graph obtained by all edges (ij) such that $G_t \cup (ij) \in G_{n,t+1,k}$.

Approximate `RandGraph`

- During the execution of `RandGraph`, after adding t edges, let $\mathbf{M}_t$ and $\mathbf{M}_t^{(c)}$ be the adjacency matrices of the partially constructed graph G_t and its complement $G_t^{(c)}$ respectively.
- In addition, let $\mathbf{Q}_t$ be the adjacency matrix of the graph obtained by all edges (ij) such that $G_t \cup (ij) \in G_{n,t+1,k}$.
- We modify `RandGraph` so that it selects the $(t+1)^{th}$ edge from all pairs (ij) with probability $p'(ij|G_t)$ that is equal to (i,j) entry of the symmetric matrix $\mathbf{P}'_{G_t}$, defined below.

$$\mathbf{P}'_{G_t} \equiv [p'(ij|G_t)] \equiv \frac{1}{Z'(G_t)}\mathbf{Q}_t \odot \widehat{\exp}\left[-\sum_{r=2}^{k-1}\left(\mathbf{M}_t + \frac{m-t}{\binom{n}{2}-t}\mathbf{M}_t^{(c)}\right)^r\right]$$

Approximate `RandGraph`

- During the execution of `RandGraph`, after adding t edges, let $\mathbf{M}_t$ and $\mathbf{M}_t^{(c)}$ be the adjacency matrices of the partially constructed graph G_t and its complement $G_t^{(c)}$ respectively.
- In addition, let $\mathbf{Q}_t$ be the adjacency matrix of the graph obtained by all edges (ij) such that $G_t \cup (ij) \in G_{n,t+1,k}$.
- We modify `RandGraph` so that it selects the $(t+1)^{th}$ edge from all pairs (ij) with probability $p'(ij|G_t)$ that is equal to (i,j) entry of the symmetric matrix $\mathbf{P}'_{G_t}$, defined below.

$$\mathbf{P}'_{G_t} \equiv [p'(ij|G_t)] \equiv \frac{1}{Z'(G_t)} \mathbf{Q}_t \odot \widehat{\exp}\left[-\sum_{r=2}^{k-1} \left(\mathbf{M}_t + \frac{m-t}{\binom{n}{2}-t} \mathbf{M}_t^{(c)} \right)^r \right]$$

- One can see that $\mathbf{Q}_t = \mathbf{J}_n - \widehat{sign}(\sum_{r=0}^{k-1} \mathbf{M}_t^r)$ where $\mathbf{J}_n$ is a $n \times n$ matrix of all ones.

Why approximating $p'(ij|G_t)$ works?

They have proved that, for any non-zero probability term $p'(ij|G_t)$,

$$p'(ij|G_t) \geq \frac{1}{Z(G_t)} e^{-E_k(G_t, ij) - O(n^{k(k+3)\alpha - 2})}$$

where $Z(G_t) = \sum_{rs \in Q} e^{-E_k(G_t, rs)}$ is the normalization term.

Efficient Computation

- Algorithm mainly relies on efficient computation of following two quantities.

$$\sum_{r=2}^{k-1} \left(\mathbf{M}_t + \frac{m-t}{\binom{n}{2} - t} \mathbf{M}_t^{(c)} \right)^r \text{ and } \sum_{r=0}^{k-1} \mathbf{M}_t^r$$

Efficient Computation

- Algorithm mainly relies on efficient computation of following two quantities.

$$\sum_{r=2}^{k-1}\left(\mathbf{M}_t + \frac{m-t}{\binom{n}{2}-t}\mathbf{M}_t^{(c)}\right)^r \text{ and } \sum_{r=0}^{k-1}\mathbf{M}_t^r$$

- Naïve computation can be done in $k \times n^3$ operations leading to a complexity of $O(n^3 m)$.

Efficient Computation

- Algorithm mainly relies on efficient computation of following two quantities.

$$\sum_{r=2}^{k-1}\left(\mathbf{M}_t + \frac{m-t}{\binom{n}{2}-t}\mathbf{M}_t^{(c)}\right)^r \text{ and } \sum_{r=0}^{k-1}\mathbf{M}_t^r$$

- Naïve computation can be done in $k \times n^3$ operations leading to a complexity of $O(n^3 m)$.
- A better algorithm exploits the sparsity of $\mathbf{M}_{t+1} - \mathbf{M}_t$. It stores the matrix prodcuts $\mathbf{M}_t, \mathbf{M}_t^2, \ldots, \mathbf{M}_t^{k-1}$ and uses the equation below to calculate $\mathbf{M}_{t+1}, \mathbf{M}_{t+1}^2, \ldots, \mathbf{M}_{t+1}^{k-1}$.

$$\mathbf{M}_{t+1}^r = [\mathbf{M}_t + (\mathbf{M}_{t+1} - \mathbf{M}_t)]^r = \mathbf{M}_t^r + \mathbf{L}$$

Efficient Computation

- Algorithm mainly relies on efficient computation of following two quantities.

$$\sum_{r=2}^{k-1}\left(\mathbf{M}_t + \frac{m-t}{\binom{n}{2}-t}\mathbf{M}_t^{(c)}\right)^r \text{ and } \sum_{r=0}^{k-1}\mathbf{M}_t^r$$

- Naïve computation can be done in $k \times n^3$ operations leading to a complexity of $O(n^3 m)$.
- A better algorithm exploits the sparsity of $\mathbf{M}_{t+1} - \mathbf{M}_t$. It stores the matrix prodcuts $\mathbf{M}_t, \mathbf{M}_t^2, \ldots, \mathbf{M}_t^{k-1}$ and uses the equation below to calculate $\mathbf{M}_{t+1}, \mathbf{M}_{t+1}^2, \ldots, \mathbf{M}_{t+1}^{k-1}$.

$$\mathbf{M}_{t+1}^r = [\mathbf{M}_t + (\mathbf{M}_{t+1} - \mathbf{M}_t)]^r = \mathbf{M}_t^r + \mathbf{L}$$

- A similar argument can be used for calculating $\left[\mathbf{M}_t + \frac{m-t}{\binom{n}{2}-t}\mathbf{M}_t^{(c)}\right]^r$ using sparsity of $\mathbf{M}_{t+1} - \mathbf{M}_t$ and $\mathbf{M}_{t+1}^{(c)} - \mathbf{M}_t^{(c)}$.

Finishing the proof

- Note that $\mathbf{L}$ is simply a matrix addition of r matrices which can be done in order of n^2 operations.
- Hence, the complexity of calculating $p'(ij|G_t)$ has order $O(n^2)$, leading to a $O(n^2 m)$ algorithm.

Extension to Bipartite Graphs with Given Degrees

Random Bipartite Graphs

- Random bipartite graphs with given node degrees define the standard model for irregular LDPC codes.
- Consider two ordered sequences of positive integers $\overline{r} = (r_1, ..., r_{n_1})$ and $\overline{c} = (c_1, ..., c_{n_2})$ such that $m = \sum_{i=1}^{n_1} r_i = \sum_{j=1}^{n_2} c_j$.

Random Bipartite Graphs

- Random bipartite graphs with given node degrees define the standard model for irregular LDPC codes.
- Consider two ordered sequences of positive integers $\overline{r} = (r_1, ..., r_{n_1})$ and $\overline{c} = (c_1, ..., c_{n_2})$ such that $m = \sum_{i=1}^{n_1} r_i = \sum_{j=1}^{n_2} c_j$.
- We would like to generate a random bipartite graph $G(V_1, V_2), V_1 = [n_1]$ and $V_2 = [n_2]$, with girth greater than k and degree sequence $(\overline{r}, \overline{c})$.

Random Bipartite Graphs

- Random bipartite graphs with given node degrees define the standard model for irregular LDPC codes.
- Consider two ordered sequences of positive integers $\overline{r} = (r_1, ..., r_{n_1})$ and $\overline{c} = (c_1, ..., c_{n_2})$ such that $m = \sum_{i=1}^{n_1} r_i = \sum_{j=1}^{n_2} c_j$.
- We would like to generate a random bipartite graph $G(V_1, V_2), V_1 = [n_1]$ and $V_2 = [n_2]$, with girth greater than k and degree sequence $(\overline{r}, \overline{c})$.
- Assume that k is an even number.
- Denote the set of all such graphs by $\mathbb{G}_{\overline{r},\overline{c},k}$.

Algorithm

Algorithm 2 `BipRandGraph`: Algorithm for generating bipartite graph with no short cycles

Input: Degree sequence $(\overline{r}, \overline{c})$ and k
Output: An element of $G_{\overline{r},\overline{r},k}$ or FAIL
set G_0 to be a graph over $V_1 = [n_1], V_2 = [n_2]$ and with no edges.
Let $\hat{r} = (\hat{r}_1, ..., \hat{r}_n)$ and $\hat{c} = (\hat{c}_1, ..., \hat{c}_m)$ be arrays; let $\hat{r} = \overline{r}$ and $\hat{c} = \overline{c}$
for $t = 0$ to $m - 1$ **do**
    if adding any edge to G_t creates a cycle of length at most k **then**
        stop and return FAIL
    else
        sample an edge (ij) with probability $p''(ij|G_t)$.
        set $G_{t+1} = G_t \cup (ij)$.
        $\hat{r}_i = \hat{r}_i - 1$ and $\hat{c}_j = \hat{c}_j - 1$
    end if
end for
if the algorithm does not FAIL before $t = m - 1$ **then**
    return G_m

Finding probability $p''(ij|G_t)$

- $p''(ij|G_t)$ is an approximation to the probability that a uniformly random extension of graph $G_t \cup (ij)$ has girth greater than k.
- Poisson-type approximation for $p''(ij|G_t)$,

$$p''(ij|G_t) \equiv \frac{\hat{r}_i \hat{c}_j e^{-E_k''(G_t,ij)}}{Z''(G_t)}$$

Finding probability $p''(ij|G_t)$

- $p''(ij|G_t)$ is an approximation to the probability that a uniformly random extension of graph $G_t \cup (ij)$ has girth greater than k.
- Poisson-type approximation for $p''(ij|G_t)$,

$$p''(ij|G_t) \equiv \frac{\hat{r}_i \hat{c}_j e^{-E_k''(G_t,ij)}}{Z''(G_t)}$$

- $E_k''(G_t, ij) \equiv \sum_{r=1}^{k/2} \sum_{\gamma \in \mathcal{C}_{2r}, (ij) \in \gamma} p_{i,j}^t(\gamma)$, where $\mathcal{C}_{2r}$ is the set of all simple cycles of length $2r$ in the complete bipartite graph on vertices of V_1 and V_2.

Finding probability $p''(ij|G_t)$

- $p''(ij|G_t)$ is an approximation to the probability that a uniformly random extension of graph $G_t \cup (ij)$ has girth greater than k.
- Poisson-type approximation for $p''(ij|G_t)$,

$$p''(ij|G_t) \equiv \frac{\hat{r}_i \hat{c}_j e^{-E''_k(G_t, ij)}}{Z''(G_t)}$$

- $E''_k(G_t, ij) \equiv \sum_{r=1}^{k/2} \sum_{\gamma \in \mathcal{C}_{2r}, (ij) \in \gamma} p^t_{i,j}(\gamma)$, where $\mathcal{C}_{2r}$ is the set of all simple cycles of length $2r$ in the complete bipartite graph on vertices of V_1 and V_2.
- $p^t_{i,j}(\gamma)$ is approximately the probability that γ is in a random extension of G_t to a random bipartite graph with degree sequence $(\overline{r}, \overline{c})$.

Conclusion

Conclusion

- The paper provides an algorithm to construct random graphs without short cycles.
- For super linear number of edges, the paper provides guarantees on probability of generation of a valid graph and the time complexity of the algorithm.

Conclusion

- The paper provides an algorithm to construct random graphs without short cycles.
- For super linear number of edges, the paper provides guarantees on probability of generation of a valid graph and the time complexity of the algorithm.
- The paper also provides an algorithm to generate random bipartite graphs without short cycles. This algorithm is of importance in generating LDPC codes.

Conclusion

- The paper provides an algorithm to construct random graphs without short cycles.
- For super linear number of edges, the paper provides guarantees on probability of generation of a valid graph and the time complexity of the algorithm.
- The paper also provides an algorithm to generate random bipartite graphs without short cycles. This algorithm is of importance in generating LDPC codes.
- Cool ideas: execution tree, Poisson approximation, total variation distance, sparse matrix multiplication!