Sampling Graphs with Given Degrees

James Zhao
University of Southern California

11 August 2014

Fun Fact

- The mean Erdős Number is 4.7. [Erdős Number Project]
- Is this surprising? Does it provide any insight into mathematicians' publishing habits, or is it purely a graph-theoretic phenomenon?
- We can answer this question statistically by comparing to simulated mean Erdős Numbers. This requires generating **labelled graphs with given degree sequence**.
- Many other similar problems in statistics (eg. Darwin's finches, degrees of separation, network motifs).

Fun Fact

- The mean Erdős Number is 4.7. [Erdős Number Project]
- Is this surprising? Does it provide any insight into mathematicians' publishing habits, or is it purely a graph-theoretic phenomenon?
- We can answer this question statistically by comparing to simulated mean Erdős Numbers. This requires generating **labelled graphs with given degree sequence**.
- Many other similar problems in statistics (eg. Darwin's finches, degrees of separation, network motifs).

Fun Fact

- The mean Erdős Number is 4.7. [Erdős Number Project]
- Is this surprising? Does it provide any insight into mathematicians' publishing habits, or is it purely a graph-theoretic phenomenon?
- We can answer this question statistically by comparing to simulated mean Erdős Numbers. This requires generating **labelled graphs with given degree sequence**.
- Many other similar problems in statistics (eg. Darwin's finches, degrees of separation, network motifs).

Fun Fact

- The mean Erdős Number is 4.7. [Erdős Number Project]
- Is this surprising? Does it provide any insight into mathematicians' publishing habits, or is it purely a graph-theoretic phenomenon?
- We can answer this question statistically by comparing to simulated mean Erdős Numbers. This requires generating **labelled graphs with given degree sequence**.
- Many other similar problems in statistics (eg. Darwin's finches, degrees of separation, network motifs).

Combinatorial Sampling

- Many useful applications for sampling from families of graphs, and from families of combinatorial structures in general.
- Easy for simple families, such as subsets of a given set, lattice points in a box, graphs with a given vertex set.
- Harder for more complicated families, such as subsets with a given size, lattice points with a given magnitude, graphs with a given degree sequence.
- Complicated families are often subsets of simple families derived by imposing additional constraints.
- A sampling algorithm for the simple superset can be used to produce one for the complicated subset.

Combinatorial Sampling

- Many useful applications for sampling from families of graphs, and from families of combinatorial structures in general.
- Easy for simple families, such as subsets of a given set, lattice points in a box, graphs with a given vertex set.
- Harder for more complicated families, such as subsets with a given size, lattice points with a given magnitude, graphs with a given degree sequence.
- Complicated families are often subsets of simple families derived by imposing additional constraints.
- A sampling algorithm for the simple superset can be used to produce one for the complicated subset.

Combinatorial Sampling

- Many useful applications for sampling from families of graphs, and from families of combinatorial structures in general.
- Easy for simple families, such as subsets of a given set, lattice points in a box, graphs with a given vertex set.
- Harder for more complicated families, such as subsets with a given size, lattice points with a given magnitude, graphs with a given degree sequence.
- Complicated families are often subsets of simple families derived by imposing additional constraints.
- A sampling algorithm for the simple superset can be used to produce one for the complicated subset.

Combinatorial Sampling

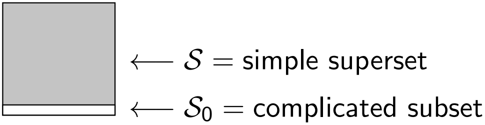
- Many useful applications for sampling from families of graphs, and from families of combinatorial structures in general.
- Easy for simple families, such as subsets of a given set, lattice points in a box, graphs with a given vertex set.
- Harder for more complicated families, such as subsets with a given size, lattice points with a given magnitude, graphs with a given degree sequence.
- Complicated families are often subsets of simple families derived by imposing additional constraints.
- A sampling algorithm for the simple superset can be used to produce one for the complicated subset.

Combinatorial Sampling

- Many useful applications for sampling from families of graphs, and from families of combinatorial structures in general.
- Easy for simple families, such as subsets of a given set, lattice points in a box, graphs with a given vertex set.
- Harder for more complicated families, such as subsets with a given size, lattice points with a given magnitude, graphs with a given degree sequence.
- Complicated families are often subsets of simple families derived by imposing additional constraints.
- A sampling algorithm for the simple superset can be used to produce one for the complicated subset.

Expanding to a Superset

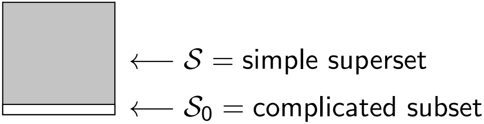
- Suppose $\mathcal{S}_0 \subset \mathcal{S}$. We can sample from $\mathcal{S}_0$ by sampling from $\mathcal{S}$ and rejecting until we observe a member of $\mathcal{S}_0$.



- This works very well if $\mathcal{S}_0$ is not too small compared to $\mathcal{S}$. For example, $GL_n(\mathbb{F}_q)$, the invertible matrices over a finite field.
- However, most of the time, this takes too long.
- Idea: instead of rejecting "bad" samples, we can try to modify them into "good" samples.

Expanding to a Superset

- Suppose $\mathcal{S}_0 \subset \mathcal{S}$. We can sample from $\mathcal{S}_0$ by sampling from $\mathcal{S}$ and rejecting until we observe a member of $\mathcal{S}_0$.



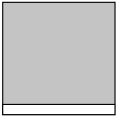
- This works very well if $\mathcal{S}_0$ is not too small compared to $\mathcal{S}$. For example, $GL_n(\mathbb{F}_q)$, the invertible matrices over a finite field.

- However, most of the time, this takes too long.

- Idea: instead of rejecting “bad” samples, we can try to modify them into “good” samples.

Expanding to a Superset

- Suppose $\mathcal{S}_0 \subset \mathcal{S}$. We can sample from $\mathcal{S}_0$ by sampling from $\mathcal{S}$ and rejecting until we observe a member of $\mathcal{S}_0$.



$\longleftarrow \mathcal{S} =$ simple superset

$\longleftarrow \mathcal{S}_0 =$ complicated subset

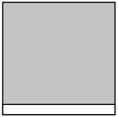
- This works very well if $\mathcal{S}_0$ is not too small compared to $\mathcal{S}$. For example, $GL_n(\mathbb{F}_q)$, the invertible matrices over a finite field.

- However, most of the time, this takes too long.

- Idea: instead of rejecting “bad” samples, we can try to modify them into “good” samples.

Expanding to a Superset

- Suppose $\mathcal{S}_0 \subset \mathcal{S}$. We can sample from $\mathcal{S}_0$ by sampling from $\mathcal{S}$ and rejecting until we observe a member of $\mathcal{S}_0$.



- This works very well if $\mathcal{S}_0$ is not too small compared to $\mathcal{S}$. For example, $GL_n(\mathbb{F}_q)$, the invertible matrices over a finite field.

- However, most of the time, this takes too long.

- Idea: instead of rejecting "bad" samples, we can try to modify them into "good" samples.

Expand and Contract

- We will need 3 ingredients:
  1. A sampling algorithm for $\mathcal{S}$.
  2. A Markov chain Q on $\mathcal{S}$.
  3. A graded partition $\mathcal{S} = \mathcal{S}_0 \sqcup \mathcal{S}_1 \sqcup \cdots \sqcup \mathcal{S}_k$. This defines a notion of "badness": any $x \in \mathcal{S}_i$ has badness i.

- The algorithm:
  1. Start at a random sample of $\mathcal{S}$.
  2. Run the chain Q, rejecting moves that increase the badness.
  3. Stop when an element of $\mathcal{S}_0$ is reached.

Expand and Contract

- We will need 3 ingredients:
  1. A sampling algorithm for $\mathcal{S}$.
  2. A Markov chain Q on $\mathcal{S}$.
  3. A graded partition $\mathcal{S} = \mathcal{S}_0 \sqcup \mathcal{S}_1 \sqcup \cdots \sqcup \mathcal{S}_k$. This defines a notion of “badness”: any $x \in \mathcal{S}_i$ has badness i.

- The algorithm:
  1. Start at a random sample of $\mathcal{S}$.
  2. Run the chain Q, rejecting moves that increase the badness.
  3. Stop when an element of $\mathcal{S}_0$ is reached.

Expand and Contract

- We will need 3 ingredients:
  1. A sampling algorithm for $\mathcal{S}$.
  2. A Markov chain Q on $\mathcal{S}$.
  3. A graded partition $\mathcal{S} = \mathcal{S}_0 \sqcup \mathcal{S}_1 \sqcup \cdots \sqcup \mathcal{S}_k$. This defines a notion of "badness": any $x \in \mathcal{S}_i$ has badness i.

- The algorithm:
  1. Start at a random sample of $\mathcal{S}$.
  2. Run the chain Q, rejecting moves that increase the badness.
  3. Stop when an element of $\mathcal{S}_0$ is reached.

Expand and Contract

- We will need 3 ingredients:
  1. A sampling algorithm for $\mathcal{S}$.
  2. A Markov chain Q on $\mathcal{S}$.
  3. A graded partition $\mathcal{S} = \mathcal{S}_0 \sqcup \mathcal{S}_1 \sqcup \cdots \sqcup \mathcal{S}_k$. This defines a notion of "badness": any $x \in \mathcal{S}_i$ has badness i.

- The algorithm:
  1. Start at a random sample of $\mathcal{S}$.
  2. Run the chain Q, rejecting moves that increase the badness.
  3. Stop when an element of $\mathcal{S}_0$ is reached.

Expand and Contract

- We will need 3 ingredients:
  1. A sampling algorithm for $\mathcal{S}$.
  2. A Markov chain Q on $\mathcal{S}$.
  3. A graded partition $\mathcal{S} = \mathcal{S}_0 \sqcup \mathcal{S}_1 \sqcup \cdots \sqcup \mathcal{S}_k$. This defines a notion of "badness": any $x \in \mathcal{S}_i$ has badness i.

- The algorithm:
  1. Start at a random sample of $\mathcal{S}$.
  2. Run the chain Q, rejecting moves that increase the badness.
  3. Stop when an element of $\mathcal{S}_0$ is reached.

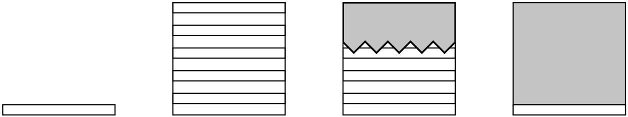
Expand and Contract

- We will need 3 ingredients:
  1. A sampling algorithm for $\mathcal{S}$.
  2. A Markov chain Q on $\mathcal{S}$.
  3. A graded partition $\mathcal{S} = \mathcal{S}_0 \sqcup \mathcal{S}_1 \sqcup \cdots \sqcup \mathcal{S}_k$. This defines a notion of "badness": any $x \in \mathcal{S}_i$ has badness i.

- The algorithm:
  1. Start at a random sample of $\mathcal{S}$.
  2. Run the chain Q, rejecting moves that increase the badness.
  3. Stop when an element of $\mathcal{S}_0$ is reached.

Expand and Contract

- In the ideal scenario, uniformity is gained through expansion and preserved through contraction.



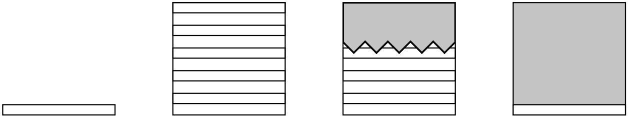
- If uniformity is not preserved exactly, reverse coupling arguments can show asymptotic uniformity.

- In this talk:

  1. A few motivating examples for Expand and Contract.
  2. Existing algorithms for graphs with given degrees.
  3. Expand and Contract for graphs with given degrees.

Expand and Contract

- In the ideal scenario, uniformity is gained through expansion and preserved through contraction.

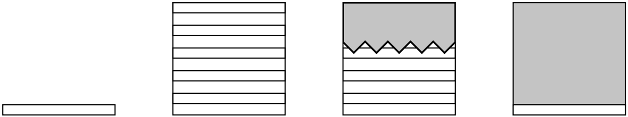


- If uniformity is not preserved exactly, reverse coupling arguments can show asymptotic uniformity.

- In this talk:
  1. A few motivating examples for Expand and Contract.
  2. Existing algorithms for graphs with given degrees.
  3. Expand and Contract for graphs with given degrees.

Expand and Contract

- In the ideal scenario, uniformity is gained through expansion and preserved through contraction.



- If uniformity is not preserved exactly, reverse coupling arguments can show asymptotic uniformity.

- In this talk:
  1. A few motivating examples for Expand and Contract.
  2. Existing algorithms for graphs with given degrees.
  3. Expand and Contract for graphs with given degrees.

Example 1: Subsets of a Given Size

- Let $\mathcal{S}_0$ be the subsets of $\{1, \ldots, n\}$ which have cardinality k. Assuming $k = \Theta(n)$, best existing algorithms take $O(n)$ steps. [Knuth 1969, Pak 1998]
- Expand and Contract:
  1. Include each element of $\{1, \ldots, n\}$ independently w.p. $\frac{k}{n}$;
  2. Pick an element of $\{1, \ldots, n\}$ randomly and include/exclude it at random, rejecting if the cardinality moves further from k;
  3. Repeat until the cardinality is exactly k.
- At each step, $\Theta(1)$ probability of cardinality getting closer to k, so Step 2 is repeated $O(\sqrt{n})$ times. Runtime is $O(n)$.
- Initial state is uniform on subsets of each given size, and each move preserves this uniformity, so output is also uniform.

Example 1: Subsets of a Given Size

- Let $\mathcal{S}_0$ be the subsets of $\{1, \ldots, n\}$ which have cardinality k. Assuming $k = \Theta(n)$, best existing algorithms take $O(n)$ steps. [Knuth 1969, Pak 1998]
- Expand and Contract:
  1. Include each element of $\{1, \ldots, n\}$ independently w.p. $\frac{k}{n}$;
  2. Pick an element of $\{1, \ldots, n\}$ randomly and include/exclude it at random, rejecting if the cardinality moves further from k;
  3. Repeat until the cardinality is exactly k.
- At each step, $\Theta(1)$ probability of cardinality getting closer to k, so Step 2 is repeated $O(\sqrt{n})$ times. Runtime is $O(n)$.
- Initial state is uniform on subsets of each given size, and each move preserves this uniformity, so output is also uniform.

Example 1: Subsets of a Given Size

- Let $\mathcal{S}_0$ be the subsets of $\{1, \ldots, n\}$ which have cardinality k. Assuming $k = \Theta(n)$, best existing algorithms take $O(n)$ steps. [Knuth 1969, Pak 1998]
- Expand and Contract:
  1. Include each element of $\{1, \ldots, n\}$ independently w.p. $\frac{k}{n}$;
  2. Pick an element of $\{1, \ldots, n\}$ randomly and include/exclude it at random, rejecting if the cardinality moves further from k;
  3. Repeat until the cardinality is exactly k.
- At each step, $\Theta(1)$ probability of cardinality getting closer to k, so Step 2 is repeated $O(\sqrt{n})$ times. Runtime is $O(n)$.
- Initial state is uniform on subsets of each given size, and each move preserves this uniformity, so output is also uniform.

Example 1: Subsets of a Given Size

- Let $\mathcal{S}_0$ be the subsets of $\{1, \ldots, n\}$ which have cardinality k. Assuming $k = \Theta(n)$, best existing algorithms take $O(n)$ steps. [Knuth 1969, Pak 1998]
- Expand and Contract:
  1. Include each element of $\{1, \ldots, n\}$ independently w.p. $\frac{k}{n}$;
  2. Pick an element of $\{1, \ldots, n\}$ randomly and include/exclude it at random, rejecting if the cardinality moves further from k;
  3. Repeat until the cardinality is exactly k.
- At each step, $\Theta(1)$ probability of cardinality getting closer to k, so Step 2 is repeated $O(\sqrt{n})$ times. Runtime is $O(n)$.
- Initial state is uniform on subsets of each given size, and each move preserves this uniformity, so output is also uniform.

Example 2: General Linear Group

- Let $\mathcal{S}_0 = GL_n(\mathbb{F}_q)$, the invertible $n \times n$ matrices over $\mathbb{F}_q$.
  Best existing algorithms run in time $O(n^3)$. [Pak 1998]

- Expand and Contract:
  1. Pick each entry independently and uniformly at random;
  2. Pick a linearly dependent column and replace each entry independently and uniformly at random;
  3. Repeat until the matrix has full rank.

- Each move takes $O(n^3)$ steps and has $\Theta(1)$ probability of increasing rank, so runtime is $O(n^3)$.

- Initial state is uniform on matrices with same RREF.
  Each move preserves this, hence output is uniform.

Example 2: General Linear Group

- Let $\mathcal{S}_0 = GL_n(\mathbb{F}_q)$, the invertible $n \times n$ matrices over $\mathbb{F}_q$.
  Best existing algorithms run in time $O(n^3)$. [Pak 1998]

- Expand and Contract:
  1. Pick each entry independently and uniformly at random;
  2. Pick a linearly dependent column and replace each entry independently and uniformly at random;
  3. Repeat until the matrix has full rank.

- Each move takes $O(n^3)$ steps and has $\Theta(1)$ probability of increasing rank, so runtime is $O(n^3)$.

- Initial state is uniform on matrices with same RREF.
  Each move preserves this, hence output is uniform.

Example 2: General Linear Group

- Let $\mathcal{S}_0 = GL_n(\mathbb{F}_q)$, the invertible $n \times n$ matrices over $\mathbb{F}_q$. Best existing algorithms run in time $O(n^3)$. [Pak 1998]

- Expand and Contract:
  1. Pick each entry independently and uniformly at random;
  2. Pick a linearly dependent column and replace each entry independently and uniformly at random;
  3. Repeat until the matrix has full rank.

- Each move takes $O(n^3)$ steps and has $\Theta(1)$ probability of increasing rank, so runtime is $O(n^3)$.

- Initial state is uniform on matrices with same RREF. Each move preserves this, hence output is uniform.

Example 2: General Linear Group

- Let $\mathcal{S}_0 = GL_n(\mathbb{F}_q)$, the invertible $n \times n$ matrices over $\mathbb{F}_q$.
  Best existing algorithms run in time $O(n^3)$. [Pak 1998]

- Expand and Contract:
  1. Pick each entry independently and uniformly at random;
  2. Pick a linearly dependent column and replace each entry independently and uniformly at random;
  3. Repeat until the matrix has full rank.

- Each move takes $O(n^3)$ steps and has $\Theta(1)$ probability of increasing rank, so runtime is $O(n^3)$.

- Initial state is uniform on matrices with same RREF.
  Each move preserves this, hence output is uniform.

Example 3: Lattice Points on a Sphere

- Let $\mathcal{S}_0$ be the set of points $(a_1, \ldots, a_n) \in \mathbb{N}^n$ with sum of squares $a_1^2 + \cdots + a_n^2 = E = \Theta(n)$. Quantum mechanical system with a given energy E. [Chatterjee-Diaconis 2013]
- Expand and Contract:
  1. Pick entries independently with $\mathbb{P}(a_i = k) = \frac{1}{Z} e^{-ck^2}$, where c is chosen so that $\mathbb{E}[a_i^2] = E/n$;
  2. Randomly change an entry by ± 1, rejecting if either the sum of squares moves further from E or if the entry becomes 0;
  3. Repeat until the sum of squares is exactly E.
- Similarly to subsets example, runtime is $O(n)$.
- Initial state is uniform on each hypersphere. Unfortunately, each move increases TV distance by $O(n^{-1/2})$, and there are $O(\sqrt{n})$ moves, so we can only get a TV bound of $O(1)$.

Example 3: Lattice Points on a Sphere

- Let $\mathcal{S}_0$ be the set of points $(a_1, \ldots, a_n) \in \mathbb{N}^n$ with sum of squares $a_1^2 + \cdots + a_n^2 = E = \Theta(n)$. Quantum mechanical system with a given energy E. [Chatterjee-Diaconis 2013]
- Expand and Contract:
  1. Pick entries independently with $\mathbb{P}(a_i = k) = \frac{1}{Z} e^{-ck^2}$, where c is chosen so that $\mathbb{E}[a_i^2] = E/n$;
  2. Randomly change an entry by ± 1, rejecting if either the sum of squares moves further from E or if the entry becomes 0;
  3. Repeat until the sum of squares is exactly E.
- Similarly to subsets example, runtime is $O(n)$.
- Initial state is uniform on each hypersphere. Unfortunately, each move increases TV distance by $O(n^{-1/2})$, and there are $O(\sqrt{n})$ moves, so we can only get a TV bound of $O(1)$.

Example 3: Lattice Points on a Sphere

- Let $\mathcal{S}_0$ be the set of points $(a_1, \ldots, a_n) \in \mathbb{N}^n$ with sum of squares $a_1^2 + \cdots + a_n^2 = E = \Theta(n)$. Quantum mechanical system with a given energy E. [Chatterjee-Diaconis 2013]
- Expand and Contract:
  1. Pick entries independently with $\mathbb{P}(a_i = k) = \frac{1}{Z} e^{-ck^2}$, where c is chosen so that $\mathbb{E}[a_i^2] = E/n$;
  2. Randomly change an entry by ± 1, rejecting if either the sum of squares moves further from E or if the entry becomes 0;
  3. Repeat until the sum of squares is exactly E.
- Similarly to subsets example, runtime is $O(n)$.
- Initial state is uniform on each hypersphere. Unfortunately, each move increases TV distance by $O(n^{-1/2})$, and there are $O(\sqrt{n})$ moves, so we can only get a TV bound of $O(1)$.

Example 3: Lattice Points on a Sphere

- Let $\mathcal{S}_0$ be the set of points $(a_1, \ldots, a_n) \in \mathbb{N}^n$ with sum of squares $a_1^2 + \cdots + a_n^2 = E = \Theta(n)$. Quantum mechanical system with a given energy E. [Chatterjee-Diaconis 2013]
- Expand and Contract:
  1. Pick entries independently with $\mathbb{P}(a_i = k) = \frac{1}{Z} e^{-ck^2}$, where c is chosen so that $\mathbb{E}[a_i^2] = E/n$;
  2. Randomly change an entry by ± 1, rejecting if either the sum of squares moves further from E or if the entry becomes 0;
  3. Repeat until the sum of squares is exactly E.
- Similarly to subsets example, runtime is $O(n)$.
- Initial state is uniform on each hypersphere. Unfortunately, each move increases TV distance by $O(n^{-1/2})$, and there are $O(\sqrt{n})$ moves, so we can only get a TV bound of $O(1)$.

Example 4: Magic Squares

- Let $\mathcal{S}_0$ be the $n \times n$ matrices with entries a permutation of $1, \ldots, n^2$ whose row/column/diagonal sums are $\frac{1}{2}n(n^2+1)$.
- Define **badness** of a $n \times n$ matrix as the L^1 distance of the row/column/diagonal sums from $\frac{1}{2}n(n^2+1)$.
- Expand and Contract:
  1. Pick entries via a random permutation of $\{1, \ldots, n^2\}$;
  2. Randomly swap 2 entries, rejecting moves that increase badness by i w.p. $1 - e^{-\beta i}$ for some $0 < \beta < \infty$;
  3. Repeat until a magic square is obtained.
- Difficult to say anything about either runtime or uniformity. Can generate up to 50×50 magic squares.

Example 4: Magic Squares

- Let $\mathcal{S}_0$ be the $n \times n$ matrices with entries a permutation of $1, \ldots, n^2$ whose row/column/diagonal sums are $\frac{1}{2}n(n^2+1)$.
- Define **badness** of a $n \times n$ matrix as the L^1 distance of the row/column/diagonal sums from $\frac{1}{2}n(n^2+1)$.
- Expand and Contract:
  1. Pick entries via a random permutation of $\{1, \ldots, n^2\}$;
  2. Randomly swap 2 entries, rejecting moves that increase badness by i w.p. $1 - e^{-\beta i}$ for some $0 < \beta < \infty$;
  3. Repeat until a magic square is obtained.
- Difficult to say anything about either runtime or uniformity. Can generate up to 50×50 magic squares.

Example 4: Magic Squares

- Let $\mathcal{S}_0$ be the $n \times n$ matrices with entries a permutation of $1, \ldots, n^2$ whose row/column/diagonal sums are $\frac{1}{2}n(n^2+1)$.
- Define **badness** of a $n \times n$ matrix as the L^1 distance of the row/column/diagonal sums from $\frac{1}{2}n(n^2+1)$.
- Expand and Contract:
  1. Pick entries via a random permutation of $\{1, \ldots, n^2\}$;
  2. Randomly swap 2 entries, rejecting moves that increase badness by i w.p. $1 - e^{-\beta i}$ for some $0 < \beta < \infty$;
  3. Repeat until a magic square is obtained.
- Difficult to say anything about either runtime or uniformity. Can generate up to 50×50 magic squares.

Example 4: Magic Squares

- Let $\mathcal{S}_0$ be the $n \times n$ matrices with entries a permutation of $1, \ldots, n^2$ whose row/column/diagonal sums are $\frac{1}{2}n(n^2+1)$.
- Define **badness** of a $n \times n$ matrix as the L^1 distance of the row/column/diagonal sums from $\frac{1}{2}n(n^2+1)$.
- Expand and Contract:
  1. Pick entries via a random permutation of $\{1, \ldots, n^2\}$;
  2. Randomly swap 2 entries, rejecting moves that increase badness by i w.p. $1 - e^{-\beta i}$ for some $0 < \beta < \infty$;
  3. Repeat until a magic square is obtained.
- Difficult to say anything about either runtime or uniformity. Can generate up to 50×50 magic squares.

Graphs With Given Degrees

- Let $(d_1, \ldots, d_n)$ be a degree sequence, and let $\mathcal{S}_0$ be the graphs with vertex set $\{1, \ldots, n\}$ where vertex i has degree d_i.

- Let $\mathcal{S} \supset \mathcal{S}_0$ be the set of multigraphs with the given degree sequence $(d_1, \ldots, d_n)$. Sample from $\mathcal{S}$ by considering each vertex as a set of half-edges, and picking a perfect matching of half-edges. [Bender-Canfield 1978, Bollobás 1980]

- For example, degree sequence $1, 1, 1, 2, 2, 2, 3, 3, 3$.

Graphs With Given Degrees

- Let $(d_1, \ldots, d_n)$ be a degree sequence, and let $\mathcal{S}_0$ be the graphs with vertex set $\{1, \ldots, n\}$ where vertex i has degree d_i.

- Let $\mathcal{S} \supset \mathcal{S}_0$ be the set of multigraphs with the given degree sequence $(d_1, \ldots, d_n)$. Sample from $\mathcal{S}$ by considering each vertex as a set of half-edges, and picking a perfect matching of half-edges. [Bender-Canfield 1978, Bollobás 1980]

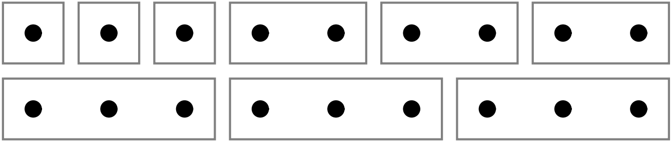
- For example, degree sequence $1, 1, 1, 2, 2, 2, 3, 3, 3$.

Graphs With Given Degrees

- Let $(d_1, \ldots, d_n)$ be a degree sequence, and let $\mathcal{S}_0$ be the graphs with vertex set $\{1, \ldots, n\}$ where vertex i has degree d_i.

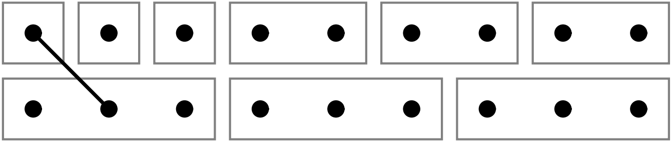
- Let $\mathcal{S} \supset \mathcal{S}_0$ be the set of multigraphs with the given degree sequence $(d_1, \ldots, d_n)$. Sample from $\mathcal{S}$ by considering each vertex as a set of half-edges, and picking a perfect matching of half-edges. [Bender-Canfield 1978, Bollobás 1980]

- For example, degree sequence $1, 1, 1, 2, 2, 2, 3, 3, 3$.



Graphs With Given Degrees

- Let $(d_1, \ldots, d_n)$ be a degree sequence, and let $\mathcal{S}_0$ be the graphs with vertex set $\{1, \ldots, n\}$ where vertex i has degree d_i.
- Let $\mathcal{S} \supset \mathcal{S}_0$ be the set of multigraphs with the given degree sequence $(d_1, \ldots, d_n)$. Sample from $\mathcal{S}$ by considering each vertex as a set of half-edges, and picking a perfect matching of half-edges. [Bender-Canfield 1978, Bollobás 1980]
- For example, degree sequence $1, 1, 1, 2, 2, 2, 3, 3, 3$.

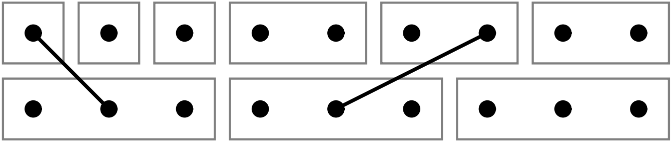


Graphs With Given Degrees

- Let $(d_1, \ldots, d_n)$ be a degree sequence, and let $\mathcal{S}_0$ be the graphs with vertex set $\{1, \ldots, n\}$ where vertex i has degree d_i.

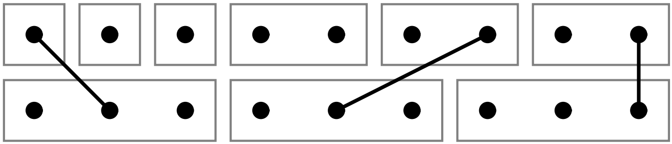
- Let $\mathcal{S} \supset \mathcal{S}_0$ be the set of multigraphs with the given degree sequence $(d_1, \ldots, d_n)$. Sample from $\mathcal{S}$ by considering each vertex as a set of half-edges, and picking a perfect matching of half-edges. [Bender-Canfield 1978, Bollobás 1980]

- For example, degree sequence $1, 1, 1, 2, 2, 2, 3, 3, 3$.



Graphs With Given Degrees

- Let $(d_1, \dots, d_n)$ be a degree sequence, and let $\mathcal{S}_0$ be the graphs with vertex set $\{1, \dots, n\}$ where vertex i has degree d_i.
- Let $\mathcal{S} \supset \mathcal{S}_0$ be the set of multigraphs with the given degree sequence $(d_1, \dots, d_n)$. Sample from $\mathcal{S}$ by considering each vertex as a set of half-edges, and picking a perfect matching of half-edges. [Bender-Canfield 1978, Bollobás 1980]
- For example, degree sequence $1, 1, 1, 2, 2, 2, 3, 3, 3$.

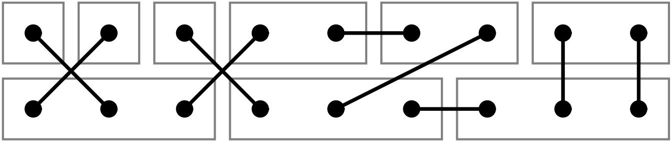


Graphs With Given Degrees

- Let $(d_1, \ldots, d_n)$ be a degree sequence, and let $\mathcal{S}_0$ be the graphs with vertex set $\{1, \ldots, n\}$ where vertex i has degree d_i.

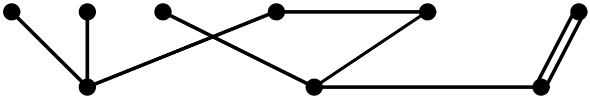
- Let $\mathcal{S} \supset \mathcal{S}_0$ be the set of multigraphs with the given degree sequence $(d_1, \ldots, d_n)$. Sample from $\mathcal{S}$ by considering each vertex as a set of half-edges, and picking a perfect matching of half-edges. [Bender-Canfield 1978, Bollobás 1980]

- For example, degree sequence $1, 1, 1, 2, 2, 2, 3, 3, 3$.



Graphs With Given Degrees

- Let $(d_1, \dots, d_n)$ be a degree sequence, and let $\mathcal{S}_0$ be the graphs with vertex set $\{1, \dots, n\}$ where vertex i has degree d_i.
- Let $\mathcal{S} \supset \mathcal{S}_0$ be the set of multigraphs with the given degree sequence $(d_1, \dots, d_n)$. Sample from $\mathcal{S}$ by considering each vertex as a set of half-edges, and picking a perfect matching of half-edges. [Bender-Canfield 1978, Bollobás 1980]
- For example, degree sequence $1, 1, 1, 2, 2, 2, 3, 3, 3$.



Matching-Based Sampling Methods

$n = \#$ vertices $\quad m = \#$ edges $\quad d = $ max degree

- If we discard non-simple graphs, the result is uniform among simple graphs. Works well when $d = O(1)$. [Wormald 1984]
- Can adjust the multigraph to obtain a simple graph when $d = o(m^{1/4})$. But this is slow. [McKay-Wormald 1990]
- Can reject moves that create bad edges up to degree $d = O(n^{1/11})$. [Steger-Wormald 1999]
- Smarter rejection works up to degree $d = O(m^{1/4-\epsilon})$. [Bayati-Kim-Saberi 2010]

Matching-Based Sampling Methods

$n = \#$ vertices	$m = \#$ edges	$d = $ max degree

- If we discard non-simple graphs, the result is uniform among simple graphs. Works well when $d = O(1)$. [Wormald 1984]
- Can adjust the multigraph to obtain a simple graph when $d = o(m^{1/4})$. But this is slow. [McKay-Wormald 1990]
- Can reject moves that create bad edges up to degree $d = O(n^{1/11})$. [Steger-Wormald 1999]
- Smarter rejection works up to degree $d = O(m^{1/4-\epsilon})$. [Bayati-Kim-Saberi 2010]

Matching-Based Sampling Methods

$n = \#$ vertices	$m = \#$ edges	$d =$ max degree

- If we discard non-simple graphs, the result is uniform among simple graphs. Works well when $d = O(1)$. [Wormald 1984]
- Can adjust the multigraph to obtain a simple graph when $d = o(m^{1/4})$. But this is slow. [McKay-Wormald 1990]
- Can reject moves that create bad edges up to degree $d = O(n^{1/11})$. [Steger-Wormald 1999]
- Smarter rejection works up to degree $d = O(m^{1/4-\epsilon})$. [Bayati-Kim-Saberi 2010]

Matching-Based Sampling Methods

$n = \#$ vertices $\quad m = \#$ edges $\quad d =$ max degree

- If we discard non-simple graphs, the result is uniform among simple graphs. Works well when $d = O(1)$. [Wormald 1984]
- Can adjust the multigraph to obtain a simple graph when $d = o(m^{1/4})$. But this is slow. [McKay-Wormald 1990]
- Can reject moves that create bad edges up to degree $d = O(n^{1/11})$. [Steger-Wormald 1999]
- Smarter rejection works up to degree $d = O(m^{1/4-\epsilon})$. [Bayati-Kim-Saberi 2010]

Matching-Based Sampling Methods

$n = \#$ vertices	$m = \#$ edges	$d =$ max degree

- If we discard non-simple graphs, the result is uniform among simple graphs. Works well when $d = O(1)$. [Wormald 1984]

- Can adjust the multigraph to obtain a simple graph when $d = o(m^{1/4})$. But this is slow. [McKay-Wormald 1990]

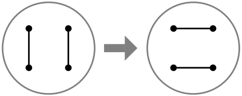
- Can reject moves that create bad edges up to degree $d = O(n^{1/11})$. [Steger-Wormald 1999]

- Smarter rejection works up to degree $d = O(m^{1/4-\epsilon})$. [Bayati-Kim-Saberi 2010]

Markov Chain Monte Carlo

$n = \#$ vertices	$m = \#$ edges	$d = $ max degree

- Start with a graph, and run a Markov chain until it mixes.
- Easiest move is a **2-swap**.

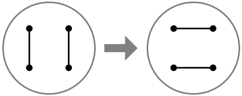


- Polynomial runtime for $d \leq \sqrt{n/2}$. [Jerrum-Sinclair 1990]
- Polynomial runtime for all inputs with a warm start. [Bezáková-Bhatnagar-Vigoda 2006]
- However, best runtime bound is $O(n^4 m^3 d)$, and the warm start is a complicated process involving bootstrapping weights.

Markov Chain Monte Carlo

$n = \#$ vertices $\quad m = \#$ edges $\quad d = $ max degree

- Start with a graph, and run a Markov chain until it mixes.
- Easiest move is a **2-swap**.

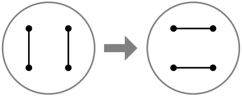


- Polynomial runtime for $d \leq \sqrt{n/2}$. [Jerrum-Sinclair 1990]
- Polynomial runtime for all inputs with a warm start. [Bezáková-Bhatnagar-Vigoda 2006]
- However, best runtime bound is $O(n^4 m^3 d)$, and the warm start is a complicated process involving bootstrapping weights.

Markov Chain Monte Carlo

$n = \#$ vertices	$m = \#$ edges	$d =$ max degree

- Start with a graph, and run a Markov chain until it mixes.
- Easiest move is a **2-swap**.

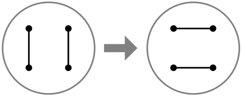


- Polynomial runtime for $d \leq \sqrt{n/2}$. [Jerrum-Sinclair 1990]
- Polynomial runtime for all inputs with a warm start. [Bezáková-Bhatnagar-Vigoda 2006]
- However, best runtime bound is $O(n^4 m^3 d)$, and the warm start is a complicated process involving bootstrapping weights.

Markov Chain Monte Carlo

$n = \#$ vertices	$m = \#$ edges	$d =$ max degree

- Start with a graph, and run a Markov chain until it mixes.
- Easiest move is a **2-swap**.

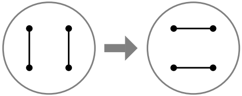


- Polynomial runtime for $d \leq \sqrt{n/2}$. [Jerrum-Sinclair 1990]
- Polynomial runtime for all inputs with a warm start. [Bezáková-Bhatnagar-Vigoda 2006]
- However, best runtime bound is $O(n^4 m^3 d)$, and the warm start is a complicated process involving bootstrapping weights.

Markov Chain Monte Carlo

$n = \#$ vertices	$m = \#$ edges	$d =$ max degree

- Start with a graph, and run a Markov chain until it mixes.
- Easiest move is a **2-swap**.



- Polynomial runtime for $d \leq \sqrt{n/2}$. [Jerrum-Sinclair 1990]
- Polynomial runtime for all inputs with a warm start. [Bezáková-Bhatnagar-Vigoda 2006]
- However, best runtime bound is $O(n^4 m^3 d)$, and the warm start is a complicated process involving bootstrapping weights.

Sequential Importance Sampling

- Instead of trying to generate uniformly, if we can determine the amount of non-uniformity, then we can make unbiased estimates of any statistic.

  [Chen-Diaconis-Holmes-Liu 2005, Blitzstein-Diaconis 2010]

- However, unbiased estimates can still behave badly if it the measure is too far from uniform.

- All non-MCMC algorithms behave exponentially badly for the double-star degree sequence $(d, d, 1, 1, \ldots, 1)$.

  [Bezáková-Sinclair-Štefankovič-Vigoda 2011]

Sequential Importance Sampling

- Instead of trying to generate uniformly, if we can determine the amount of non-uniformity, then we can make unbiased estimates of any statistic.

  [Chen-Diaconis-Holmes-Liu 2005, Blitzstein-Diaconis 2010]

- However, unbiased estimates can still behave badly if it the measure is too far from uniform.

- All non-MCMC algorithms behave exponentially badly for the double-star degree sequence $(d, d, 1, 1, \ldots, 1)$.

  [Bezáková-Sinclair-Štefankovič-Vigoda 2011]

Sequential Importance Sampling

- Instead of trying to generate uniformly, if we can determine the amount of non-uniformity, then we can make unbiased estimates of any statistic.
  [Chen-Diaconis-Holmes-Liu 2005, Blitzstein-Diaconis 2010]

- However, unbiased estimates can still behave badly if it the measure is too far from uniform.

- All non-MCMC algorithms behave exponentially badly for the double-star degree sequence $(d, d, 1, 1, \ldots, 1)$.

[Bezáková-Sinclair-Štefankovič-Vigoda 2011]

Comparison of Algorithms

Algorithm	Runtime	Sparseness
Perfect Matching	$O(me^{(d^2-1)/4})$	All
Bezáková-Bhatnagar-Vigoda	$O(n^4m^3d)$	All
Blitzstein-Diaconis	$O(n^2m)$	SIS
Chen-Diaconis-Holmes-Liu	$O(n^3)$	SIS
McKay-Wormald	$O(m^2d^2)$	$d = o(m^{1/4})$
Bayati-Kim-Saberi	$O(md)$	$d = o(m^{1/4})$
Expand and Contract	$O(m)$	$d = o(m^{1/4})$

Table of runtime against sparseness constraint required for provable asymptotic uniformity.

Expand and Contract

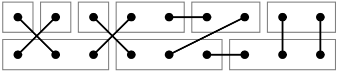
The algorithm:

1. Generate a random multigraph by perfect matching method.
2. Perform a **3-swap** on 1 **bad edge** and 2 **good edges**, rejecting moves that create any new bad edges.
3. Stop when there are no more bad edges.

Expand and Contract

The algorithm:

1. Generate a random multigraph by perfect matching method.



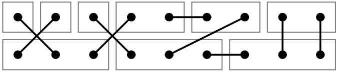
2. Perform a **3-swap** on 1 **bad edge** and 2 **good edges**, rejecting moves that create any new bad edges.

3. Stop when there are no more bad edges.

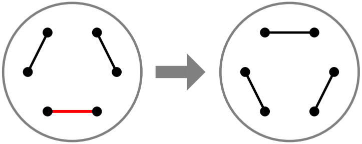
Expand and Contract

The algorithm:

1. Generate a random multigraph by perfect matching method.



2. Perform a **3-swap** on 1 **bad edge** and 2 **good edges**, rejecting moves that create any new bad edges.

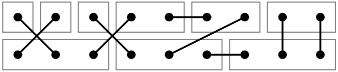


3. Stop when there are no more bad edges.

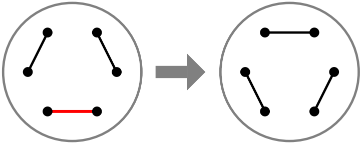
Expand and Contract

The algorithm:

1. Generate a random multigraph by perfect matching method.



2. Perform a **3-swap** on 1 **bad edge** and 2 **good edges**, rejecting moves that create any new bad edges.



3. Stop when there are no more bad edges.

Runtime

- **Theorem**: Assuming $d = O(m^{1/3})$, runtime is $O(m)$.
- *Proof:* Let (a, b) be the bad edge chosen. Want to count number of good edges (a', b') and (a'', b'') so that a 3-swap does not create any new bad edges.

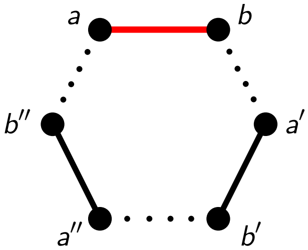
a b

b'' a'

a'' b'

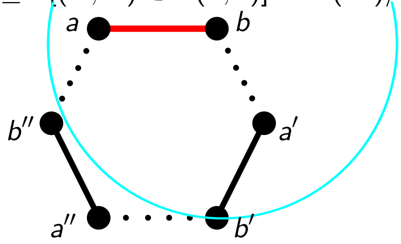
Runtime

- **Theorem**: Assuming $d = O(m^{1/3})$, runtime is $O(m)$.
- *Proof:* Let (a, b) be the bad edge chosen. Want to count number of good edges (a', b') and (a'', b'') so that a 3-swap does not create any new bad edges.



Runtime

- There are $O(d^2)$ edges in the ball $B(b, 2)$ of radius 2 at b.
  $\mathbb{P}[(b, a') \text{ is bad}] \leq \mathbb{P}[(a', b') \in B(b, 2)] = O(d^2)/m = o(1)$.

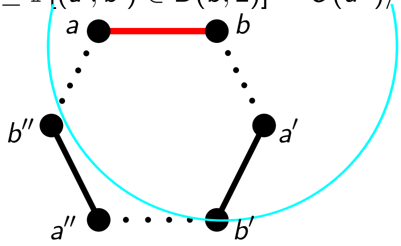


- Similarly, (b', a'') and (b'', a) are also good with high probability. Hence, almost every move removes a bad edge.

- **Lemma:** There are $O(d^2)$ bad edges.
  This implies runtime is $O(m + d^3)$. □

Runtime

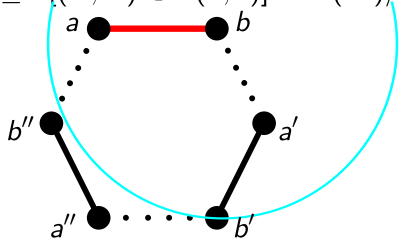
- There are $O(d^2)$ edges in the ball $B(b,2)$ of radius 2 at b. $\mathbb{P}[(b,a') \text{ is bad}] \leq \mathbb{P}[(a',b') \in B(b,2)] = O(d^2)/m = o(1)$.



- Similarly, (b',a'') and (b'',a) are also good with high probability. Hence, almost every move removes a bad edge.
- **Lemma:** There are $O(d^2)$ bad edges. This implies runtime is $O(m+d^3)$. □

Runtime

- There are $O(d^2)$ edges in the ball $B(b,2)$ of radius 2 at b.
  $\mathbb{P}[(b,a') \text{ is bad}] \leq \mathbb{P}[(a',b') \in B(b,2)] = O(d^2)/m = o(1)$.



- Similarly, (b',a'') and (b'',a) are also good with high probability. Hence, almost every move removes a bad edge.
- **Lemma:** There are $O(d^2)$ bad edges.
  This implies runtime is $O(m+d^3)$. □

Uniformity

- **Theorem**: If the maximum degree is $d = o(m^{1/4})$, then the output is asymptotically uniform in total variation.

- *Proof*: Define a **bad edge set** $B = \{(a_i, b_i, n_i)\}$ to be the set of multigraphs with prescribed multiplicities n_i at vertex pairs (a_i, b_i), and no bad edges anywhere else.

  For example, $B = \{(1,1,1), (1,2,2)\}$ is the set of multigraphs with a loop at vertex 1, a double edge between vertices 1 and 2, and no other bad edges.

- Let $B_0, B_1, B_2, \ldots$ be a sequence of bad edge sets.

Uniformity

- **Theorem**: If the maximum degree is $d = o(m^{1/4})$, then the output is asymptotically uniform in total variation.
- *Proof*: Define a **bad edge set** $B = \{(a_i, b_i, n_i)\}$ to be the set of multigraphs with prescribed multiplicities n_i at vertex pairs (a_i, b_i), and no bad edges anywhere else.

  For example, $B = \{(1,1,1),(1,2,2)\}$ is the set of multigraphs with a loop at vertex 1, a double edge between vertices 1 and 2, and no other bad edges.
- Let $B_0, B_1, B_2, \ldots$ be a sequence of bad edge sets.

Uniformity

- **Theorem**: If the maximum degree is $d = o(m^{1/4})$, then the output is asymptotically uniform in total variation.

- *Proof*: Define a **bad edge set** $B = \{(a_i, b_i, n_i)\}$ to be the set of multigraphs with prescribed multiplicities n_i at vertex pairs (a_i, b_i), and no bad edges anywhere else.

  For example, $B = \{(1,1,1),(1,2,2)\}$ is the set of multigraphs with a loop at vertex 1, a double edge between vertices 1 and 2, and no other bad edges.

- Let $B_0, B_1, B_2, \ldots$ be a sequence of bad edge sets.

Uniformity

- Let $X_0, X_1, X_2, \ldots$ be the states of the chain, and let

$$\mathcal{L}\big(X_t \mid X_0 \in B_0, \ldots, X_t \in B_t\big) = u_t U_{B_t} + (1 - u_t) E_t,$$

  where u_t is the largest number so that E_t is a signed measure of absolute mass 1. Think of u_t as the uniformity at time t given a particular sample path of bad edge sets.

- **Lemma:** Every graph with the same bad edge set has the same initial probability. Thus, $u_0 = 1$.

- Want to show $u_\infty = \lim_{t \to \infty} u_t = 1 - o(1)$.

- If $B_{t+1} = B_t$, then $u_{t+1} \geq u_t$, since the transition probabilities within B_t are symmetric, so the uniform part stays uniform.

Uniformity

- Let $X_0, X_1, X_2, \ldots$ be the states of the chain, and let

$$\mathcal{L}\big(X_t \mid X_0 \in B_0, \ldots, X_t \in B_t\big) = u_t U_{B_t} + (1 - u_t) E_t,$$

  where u_t is the largest number so that E_t is a signed measure of absolute mass 1. Think of u_t as the uniformity at time t given a particular sample path of bad edge sets.

- **Lemma:** Every graph with the same bad edge set has the same initial probability. Thus, $u_0 = 1$.

- Want to show $u_\infty = \lim_{t\to\infty} u_t = 1 - o(1)$.

- If $B_{t+1} = B_t$, then $u_{t+1} \geq u_t$, since the transition probabilities within B_t are symmetric, so the uniform part stays uniform.

Uniformity

- Let $X_0, X_1, X_2, \ldots$ be the states of the chain, and let

$$\mathcal{L}\big(X_t \mid X_0 \in B_0, \ldots, X_t \in B_t\big) = u_t U_{B_t} + (1 - u_t) E_t,$$

  where u_t is the largest number so that E_t is a signed measure of absolute mass 1. Think of u_t as the uniformity at time t given a particular sample path of bad edge sets.

- **Lemma:** Every graph with the same bad edge set has the same initial probability. Thus, $u_0 = 1$.

- Want to show $u_\infty = \lim_{t \to \infty} u_t = 1 - o(1)$.

- If $B_{t+1} = B_t$, then $u_{t+1} \geq u_t$, since the transition probabilities within B_t are symmetric, so the uniform part stays uniform.

Uniformity

- Let $X_0, X_1, X_2, \ldots$ be the states of the chain, and let

$$\mathcal{L}\big(X_t \mid X_0 \in B_0, \ldots, X_t \in B_t\big) = u_t U_{B_t} + (1 - u_t) E_t,$$

  where u_t is the largest number so that E_t is a signed measure of absolute mass 1. Think of u_t as the uniformity at time t given a particular sample path of bad edge sets.

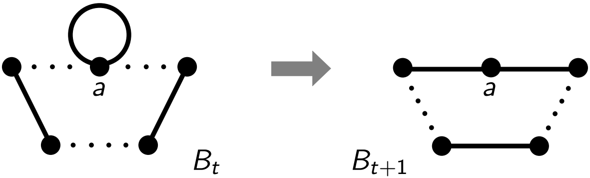
- **Lemma:** Every graph with the same bad edge set has the same initial probability. Thus, $u_0 = 1$.

- Want to show $u_\infty = \lim_{t \to \infty} u_t = 1 - o(1)$.

- If $B_{t+1} = B_t$, then $u_{t+1} \geq u_t$, since the transition probabilities within B_t are symmetric, so the uniform part stays uniform.

Uniformity

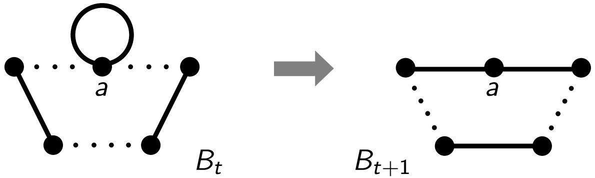
- Suppose the transition from B_t to B_{t+1} removes a loop (a, a).



- Each multigraph in B_t is connected to $(2m - O(d^2))^2$ multigraphs in B_{t+1}.
- Each multigraph in B_{t+1} is connected to $\binom{e_a}{2}(2m - O(d^2))$ multigraphs in B_t, where e_a is the number of non-loop edges out of a. Note that e_a depends only on B_{t+1} and a, and not on the multigraph that is chosen.

Uniformity

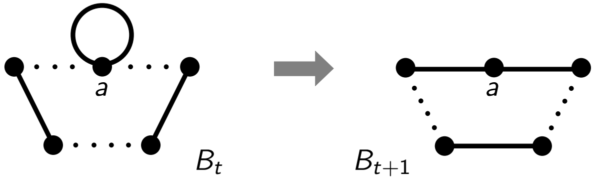
- Suppose the transition from B_t to B_{t+1} removes a loop (a, a).



- Each multigraph in B_t is connected to $(2m - O(d^2))^2$ multigraphs in B_{t+1}.
- Each multigraph in B_{t+1} is connected to $\binom{e_a}{2}(2m - O(d^2))$ multigraphs in B_t, where e_a is the number of non-loop edges out of a. Note that e_a depends only on B_{t+1} and a, and not on the multigraph that is chosen.

Uniformity

- Suppose the transition from B_t to B_{t+1} removes a loop (a, a).

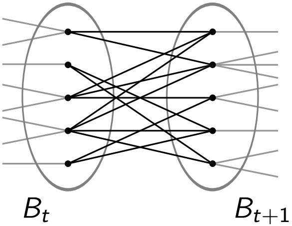


- Each multigraph in B_t is connected to $(2m - O(d^2))^2$ multigraphs in B_{t+1}.

- Each multigraph in B_{t+1} is connected to $\binom{e_a}{2}(2m - O(d^2))$ multigraphs in B_t, where e_a is the number of non-loop edges out of a. Note that e_a depends only on B_{t+1} and a, and not on the multigraph that is chosen.

Uniformity

- Variation in connectivity between B_t and B_{t+1} is a factor of $1 + O(d^2/m) = 1 + o(m^{-1/2})$.

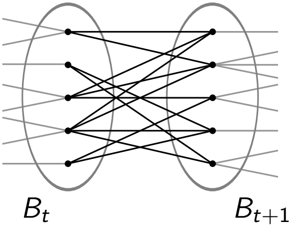


- Starting with uniform in B_t, variation in probability of each connection is $1 + o(m^{-1/2})$. Number of connections entering each point in B_{t+1} also varies by $1 + o(m^{-1/2})$.

  Hence, $u_{t+1} = (1 - o(m^{-1/2}))u_t$.

Uniformity

- Variation in connectivity between B_t and B_{t+1} is a factor of $1 + O(d^2/m) = 1 + o(m^{-1/2})$.



- Starting with uniform in B_t, variation in probability of each connection is $1 + o(m^{-1/2})$. Number of connections entering each point in B_{t+1} also varies by $1 + o(m^{-1/2})$.

  Hence, $u_{t+1} = (1 - o(m^{-1/2}))u_t$.

Uniformity

- Similar result if B_{t+1} reduces multiplicity of an edge by 1.
- When B_{t+1} removes a double edge $(a, b, 2)$, a single edge $(a, b, 1)$ remains; need to count this as a bad edge.
- When this remaining edge is removed, discard the triple $(a, b, 0)$ so that multiplicity between a and b is no longer prescribed. The proportion of graphs that contain this edge is $O(d_a d_b/m)$, so again $u_{t+1} = (1 - o(m^{-1/2}))u_t$.
- Since there are $O(d^2) = o(m^{1/2})$ bad edges,

$$u_\infty = (1 - o(m^{-1/2}))^{o(m^{1/2})} = 1 - o(1). \qquad \square$$

Uniformity

- Similar result if B_{t+1} reduces multiplicity of an edge by 1.
- When B_{t+1} removes a double edge $(a, b, 2)$, a single edge $(a, b, 1)$ remains; need to count this as a bad edge.
- When this remaining edge is removed, discard the triple $(a, b, 0)$ so that multiplicity between a and b is no longer prescribed. The proportion of graphs that contain this edge is $O(d_a d_b/m)$, so again $u_{t+1} = (1 - o(m^{-1/2}))u_t$.
- Since there are $O(d^2) = o(m^{1/2})$ bad edges,

$$u_\infty = (1 - o(m^{-1/2}))^{o(m^{1/2})} = 1 - o(1). \qquad \square$$

Uniformity

- Similar result if B_{t+1} reduces multiplicity of an edge by 1.
- When B_{t+1} removes a double edge $(a, b, 2)$, a single edge $(a, b, 1)$ remains; need to count this as a bad edge.
- When this remaining edge is removed, discard the triple $(a, b, 0)$ so that multiplicity between a and b is no longer prescribed. The proportion of graphs that contain this edge is $O(d_a d_b/m)$, so again $u_{t+1} = (1 - o(m^{-1/2}))u_t$.
- Since there are $O(d^2) = o(m^{1/2})$ bad edges,

$$u_\infty = (1 - o(m^{-1/2}))^{o(m^{1/2})} = 1 - o(1). \qquad \square$$

Uniformity

- Similar result if B_{t+1} reduces multiplicity of an edge by 1.
- When B_{t+1} removes a double edge $(a, b, 2)$, a single edge $(a, b, 1)$ remains; need to count this as a bad edge.
- When this remaining edge is removed, discard the triple $(a, b, 0)$ so that multiplicity between a and b is no longer prescribed. The proportion of graphs that contain this edge is $O(d_a d_b/m)$, so again $u_{t+1} = (1 - o(m^{-1/2}))u_t$.
- Since there are $O(d^2) = o(m^{1/2})$ bad edges,

$$u_\infty = (1 - o(m^{-1/2}))^{o(m^{1/2})} = 1 - o(1). \qquad \square$$

Uniformity

- With explicit constants, the TV distance to uniformity is

$$\frac{d^2\bar{d}^2}{4m}, \qquad \text{where } \bar{d} = \frac{1}{2m}\sum_i d_i(d_i - 1).$$

- Examples with power law distribution:

Vertices	Max Degree	Exponent	TV Bound
10^6	100	2.5	0.125
10^3	12	2.5	0.141
10^9	959	2.5	0.125
10^6	100	2.0	0.526
10^6	100	3.0	0.017
10^6	200	2.5	0.500
10^6	50	2.5	0.014
10^6	20	2.5	0.001

Uniformity

- With explicit constants, the TV distance to uniformity is

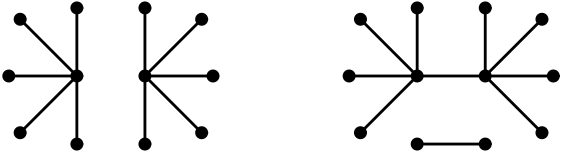
$$\frac{d^2\bar{d}^2}{4m}, \qquad \text{where } \bar{d} = \frac{1}{2m}\sum_i d_i(d_i - 1).$$

- Examples with power law distribution:

Vertices	Max Degree	Exponent	TV Bound
10^6	100	2.5	0.125
10^3	12	2.5	0.141
10^9	959	2.5	0.125
10^6	100	2.0	0.526
10^6	100	3.0	0.017
10^6	200	2.5	0.500
10^6	50	2.5	0.014
10^6	20	2.5	0.001

Multi-Star Graphs

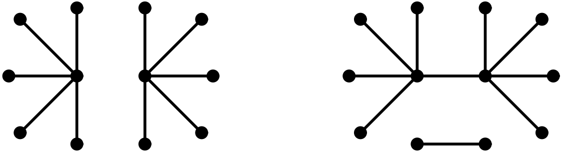
- Multi-star degree sequences are $(1, 1, \ldots, 1, d_1, \ldots, d_k)$. Counterexample to all existing non-MCMC algorithms.



- **Definition**: Probability ratio metric for measures on a finite set S is $d(\mu, \nu) = \max_{x \in S} \left| \log \mu(x) - \log \nu(x) \right|$.
- **Theorem**: Taking a sample $O(\log m)$ steps after a simple graph is reached yields an asymptotically uniform graph.
- Runtime is again $O(m)$.

Multi-Star Graphs

- Multi-star degree sequences are $(1, 1, \ldots, 1, d_1, \ldots, d_k)$. Counterexample to all existing non-MCMC algorithms.



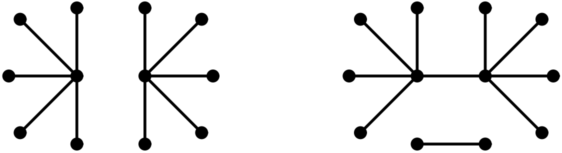
- **Definition**: Probability ratio metric for measures on a finite set S is $d(\mu, \nu) = \max_{x \in S} \left| \log \mu(x) - \log \nu(x) \right|$.

- **Theorem**: Taking a sample $O(\log m)$ steps after a simple graph is reached yields an asymptotically uniform graph.

- Runtime is again $O(m)$.

Multi-Star Graphs

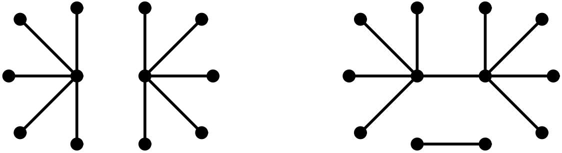
- Multi-star degree sequences are $(1, 1, \ldots, 1, d_1, \ldots, d_k)$. Counterexample to all existing non-MCMC algorithms.



- **Definition**: Probability ratio metric for measures on a finite set S is $d(\mu, \nu) = \max_{x \in S} \big| \log \mu(x) - \log \nu(x) \big|$.
- **Theorem**: Taking a sample $O(\log m)$ steps after a simple graph is reached yields an asymptotically uniform graph.
- Runtime is again $O(m)$.

Multi-Star Graphs

- Multi-star degree sequences are $(1, 1, \ldots, 1, d_1, \ldots, d_k)$. Counterexample to all existing non-MCMC algorithms.



- **Definition**: Probability ratio metric for measures on a finite set S is $d(\mu, \nu) = \max_{x \in S} \left| \log \mu(x) - \log \nu(x) \right|$.
- **Theorem**: Taking a sample $O(\log m)$ steps after a simple graph is reached yields an asymptotically uniform graph.
- Runtime is again $O(m)$.

Erdős Numbers

- Some properties of the collaboration graph: $n = 253{,}339$; $m = 496{,}489$; $d = 502$; and $\bar{d} = 38$.
- One large connected component containing Erdős. Mean distance to Erdős within this component is 4.7.
- Generating a graph with the same degrees takes 0.2 seconds, compared to 0.9 seconds to compute mean Erdős number.
- For 10,000 samples, sample mean was 4.119 with standard deviation 0.025. Thus, the real-world mean Erdős number is 22 standard deviations above the simulated mean.

Erdős Numbers

- Some properties of the collaboration graph: $n = 253{,}339$; $m = 496{,}489$; $d = 502$; and $\bar{d} = 38$.
- One large connected component containing Erdős. Mean distance to Erdős within this component is 4.7.
- Generating a graph with the same degrees takes 0.2 seconds, compared to 0.9 seconds to compute mean Erdős number.
- For 10,000 samples, sample mean was 4.119 with standard deviation 0.025. Thus, the real-world mean Erdős number is 22 standard deviations above the simulated mean.

Erdős Numbers

- Some properties of the collaboration graph: $n = 253{,}339$; $m = 496{,}489$; $d = 502$; and $\bar{d} = 38$.
- One large connected component containing Erdős. Mean distance to Erdős within this component is 4.7.
- Generating a graph with the same degrees takes 0.2 seconds, compared to 0.9 seconds to compute mean Erdős number.
- For 10,000 samples, sample mean was 4.119 with standard deviation 0.025. Thus, the real-world mean Erdős number is 22 standard deviations above the simulated mean.

Erdős Numbers

- Some properties of the collaboration graph: $n = 253{,}339$; $m = 496{,}489$; $d = 502$; and $\bar{d} = 38$.
- One large connected component containing Erdős. Mean distance to Erdős within this component is 4.7.
- Generating a graph with the same degrees takes 0.2 seconds, compared to 0.9 seconds to compute mean Erdős number.
- For 10,000 samples, sample mean was 4.119 with standard deviation 0.025. Thus, the real-world mean Erdős number is 22 standard deviations above the simulated mean.

Conclusion

- For graphs with given degree sequence, we obtained best possible runtime under a typical sparseness constraint.
- The fact that it works well for multi-star suggests sparseness constraint may not be essential. Can we weaken it?
- Possible place for improvement: current proof does not require choosing the bad edges randomly, only the good edges.
- The strategy applies to many other combinatorial structures. However, our other examples are either trivial or intractable. Can we apply it to anything else in an interesting way?

Conclusion

- For graphs with given degree sequence, we obtained best possible runtime under a typical sparseness constraint.
- The fact that it works well for multi-star suggests sparseness constraint may not be essential. Can we weaken it?
- Possible place for improvement: current proof does not require choosing the bad edges randomly, only the good edges.
- The strategy applies to many other combinatorial structures. However, our other examples are either trivial or intractable. Can we apply it to anything else in an interesting way?

Conclusion

- For graphs with given degree sequence, we obtained best possible runtime under a typical sparseness constraint.
- The fact that it works well for multi-star suggests sparseness constraint may not be essential. Can we weaken it?
- Possible place for improvement: current proof does not require choosing the bad edges randomly, only the good edges.
- The strategy applies to many other combinatorial structures. However, our other examples are either trivial or intractable. Can we apply it to anything else in an interesting way?

Conclusion

- For graphs with given degree sequence, we obtained best possible runtime under a typical sparseness constraint.
- The fact that it works well for multi-star suggests sparseness constraint may not be essential. Can we weaken it?
- Possible place for improvement: current proof does not require choosing the bad edges randomly, only the good edges.
- The strategy applies to many other combinatorial structures. However, our other examples are either trivial or intractable. Can we apply it to anything else in an interesting way?