Uniform generation of random regular graphs

Jane Gao

Joint work with Nick Wormald

15th January, 2016
27th June, 2016
Why?

- A classical TCS problem;
- Intimate connection with enumeration;
- Testing algorithms with random input;
- Coping with “big data”.
Commonly used methods

- Rejection algorithm
- Boltzmann sampler
- MCMC
- Coupling from the past
- Switching algorithm

Uniform generation of random regular graphs
Generating random $d$-regular graphs

- Tinhofer '79 – non-uniform random generation.
- Rejection algorithm – uniform sampler for small $d$ ($d = O(\sqrt{\log n})$).
  - A. Békéssy, P. Békéssy and Komlós '72;
  - Bender and Canfield '78;
  - Bollobás '80.
- MCMC – approximate sampler.
  - Jerrum and Sinclair '90 – for any $d$, FPTAS, but no explicit bound on the time complexity;
  - Cooper, Dyer and Greenhill '07 – mixing time bounded by $d^{24} n^9 \log n$;
  - Greenhill '15 – non-regular case, mixing time bounded by $\Delta^{14} M^{10} \log M$.
- Switching algorithm – fast, uniform sampler.
  - McKay and Wormald '90 – for $d = O(n^{1/3})$.
  - Gao and Wormald '15 (NEW) – for $d = o(n^{1/2})$. 
Generating random $d$-regular graphs

- Tinhofer '79 – non-uniform random generation.
- Rejection algorithm – uniform sampler for small $d$ ($d = O(\sqrt{\log n})$).
  - A. Békéssy, P. Békéssy and Komlós '72;
  - Bender and Canfield '78;
  - Bollobás '80.
- MCMC – approximate sampler.
  - Jerrum and Sinclair '90 – for any $d$, FPTAS, but no explicit bound on the time complexity;
  - Cooper, Dyer and Greenhill '07 – mixing time bounded by $d^{24}n^9 \log n$;
  - Greenhill '15 – non-regular case, mixing time bounded by $\Delta^{14}M^{10} \log M$.
- Switching algorithm – fast, uniform sampler.
  - McKay and Wormald '90 – for $d = O(n^{1/3})$.
  - Gao and Wormald '15 (NEW) – for $d = o(n^{1/2})$. 

Uniform generation of random regular graphs
Generating random $d$-regular graphs

- Tinhofer ‘79 – non-uniform random generation.
- Rejection algorithm – uniform sampler for small $d$ ($d = O(\sqrt{\log n})$).
  - A. Békéssy, P. Békéssy and Komlós ’72;
  - Bender and Canfield ’78;
  - Bollobás ’80.
- MCMC – approximate sampler.
  - Jerrum and Sinclair ’90 – for any $d$, FPTAS, but no explicit bound on the time complexity;
  - Cooper, Dyer and Greenhill ’07 – mixing time bounded by $d^{24}n^9 \log n$;
  - Greenhill ’15 – non-regular case, mixing time bounded by $\Delta^{14}M^{10} \log M$.
- Switching algorithm – fast, uniform sampler.
  - McKay and Wormald ’90 – for $d = O(n^{1/3})$.
  - Gao and Wormald ’15 (NEW) – for $d = o(n^{1/2})$. 
Generating random $d$-regular graphs

- Tinhofer '79 – non-uniform random generation.
- Rejection algorithm – uniform sampler for small $d$ ($d = O(\sqrt{\log n})$).
  - A. Békéssy, P. Békéssy and Komlós '72;
  - Bender and Canfield '78;
  - Bollobás '80.
- MCMC – approximate sampler.
  - Jerrum and Sinclair '90 – for any $d$, FPTAS, but no explicit bound on the time complexity;
  - Cooper, Dyer and Greenhill '07 – mixing time bounded by $d^{24} n^9 \log n$;
  - Greenhill '15 – non-regular case, mixing time bounded by $\Delta^{14} M^{10} \log M$.
- Switching algorithm – fast, uniform sampler.
  - McKay and Wormald '90 – for $d = O(n^{1/3})$.
  - Gao and Wormald '15 (NEW) – for $d = o(n^{1/2})$. 

Uniform generation of random regular graphs
Generating random $d$-regular graphs

- Tinhofer '79 – non-uniform random generation.
- Rejection algorithm – uniform sampler for small $d$ ($d = O(\sqrt{\log n})$).
  - A. Békéssy, P. Békéssy and Komlós '72;
  - Bender and Canfield '78;
  - Bollobás '80.
- MCMC – approximate sampler.
  - Jerrum and Sinclair '90 – for any $d$, FPTAS, but no explicit bound on the time complexity;
  - Cooper, Dyer and Greenhill '07 – mixing time bounded by $d^{24} n^9 \log n$;
  - Greenhill '15 – non-regular case, mixing time bounded by $\Delta^{14} M^{10} \log M$.
- Switching algorithm – fast, uniform sampler.
  - McKay and Wormald '90 – for $d = O(n^{1/3})$.
  - Gao and Wormald '15 (NEW) – for $d = o(n^{1/2})$. 

Uniform generation of random regular graphs
Other methods – asymptotic approximate sampler.

- Steger and Wormald ’99 – for $d = n^{1/28}$;
- Kim and Vu ’06 – for $d \leq n^{1/3-\epsilon}$;
- Bayati, Kim and Saberi ’10 – for $d \leq n^{1/2-\epsilon}$;
- Zhao ’13 – for $d = o(n^{1/3})$.

These algorithms are fast (linear time). However, unlike MCMC, the approximation error depends on $n$ and cannot be reduced by running the algorithm longer.
Other methods – asymptotic approximate sampler.

- Steger and Wormald '99 – for $d = n^{1/28}$;
- Kim and Vu '06 – for $d \leq n^{1/3-\epsilon}$;
- Bayati, Kim and Saberi '10 – for $d \leq n^{1/2-\epsilon}$;
- Zhao '13 – for $d = o(n^{1/3})$.

These algorithms are fast (linear time). However, unlike MCMC, the approximation error depends on $n$ and cannot be reduced by running the algorithm longer.
Rejection algorithm

\[ n = 6, \quad d = 3 \]
Rejection algorithm

\[ n = 6, \, d = 3 \]

Uniform generation of random regular graphs
Rejection algorithm

\[ n = 6, d = 3 \]
Rejection algorithm

$n = 6, d = 3$

reject!
Rejection algorithm

\[ n = 6, \ d = 3 \]
Rejection algorithm

$n = 6, d = 3$

Uniform generation of random regular graphs
Rejection algorithm

$n = 6, d = 3$
Rejection algorithm

\[ n = 6, \ d = 3 \]

accept!

Uniform generation of random regular graphs
Rejection algorithm

The time complexity is \textit{exponential} in $d^2$. 
Switching algorithm (McKay and Wormald ’90): DEG

\[ n = 6, \, d = 3 \]
Switching algorithm (McKay and Wormald ’90): DEG

$n = 6, d = 3$

Uniform generation of random regular graphs
Switching algorithm (McKay and Wormald '90): DEG

For $d = O(n^{1/3})$, with a positive probability (bounded away from 0), there are no double-loops, or multiple edges with multiplicity greater than 2.
So we only need to worry about loops and double-edges.
Switching algorithm (McKay and Wormald '90): DEG

$n = 6, d = 3$

Uniform generation of random regular graphs
Switching algorithm (McKay and Wormald '90): DEG

\[ n = 6, \quad d = 3 \]
Switching algorithm (McKay and Wormald ’90): DEG

switch away double-edges

switch away double-edges
Switching algorithm (McKay and Wormald '90): DEG

Let $S_i$ denote the set of pairings containing exactly $i$ loops (correspondingly, $i$ double-edges, in the phase for double-edge reduction). A switching converts a pairing $P \in S_i$ to $P' \in S_{i-1}$. 
Switching algorithm (McKay and Wormald '90): DEG

Let \( S_i \) denote the set of pairings containing exactly \( i \) loops (correspondingly, \( i \) double-edges, in the phase for double-edge reduction).

A switching converts a pairing \( P \in S_i \) to \( P' \in S_{i-1} \).
Switching algorithm (McKay and Wormald '90): DEG

Let $S_i$ denote the set of pairings containing exactly $i$ loops (correspondingly, $i$ double-edges, in the phase for double-edge reduction).

A switching converts a pairing $P \in S_i$ to $P' \in S_{i-1}$.
Switching algorithm (McKay and Wormald '90): DEG

Uniform generation of random regular graphs
Switching algorithm (McKay and Wormald '90): DEG

- If $P$ is uniformly distributed in $S_i$, is $P'$ uniformly distributed in $S_{i-1}$?
- No.
- Because
  - The number $(N(P))$ of ways to perform a switching to $P \in S_i$ is not uniformly the same for all $P \in S_i$;
  - The number $(N'(P'))$ of ways to reach $P' \in S_{i-1}$ via a switching is not uniformly the same for all $P' \in S_{i-1}$. 
Switching algorithm (McKay and Wormald '90): DEG

- If $P$ is uniformly distributed in $S_i$, is $P'$ uniformly distributed in $S_{i-1}$?
- No.
- Because
  - The number ($N(P)$) of ways to perform a switching to $P \in S_i$ is not uniformly the same for all $P \in S_i$;
  - The number ($N'(P')$) of ways to reach $P' \in S_{i-1}$ via a switching is not uniformly the same for all $P' \in S_{i-1}$.
Switching algorithm (McKay and Wormald ’90): DEG

- If $P$ is uniformly distributed in $S_i$, is $P'$ uniformly distributed in $S_{i-1}$?
- No.
- Because
  - The number $(N(P))$ of ways to perform a switching to $P \in S_i$ is not uniformly the same for all $P \in S_i$;
  - The number $(N'(P'))$ of ways to reach $P' \in S_{i-1}$ via a switching is not uniformly the same for all $P' \in S_{i-1}$.
Switching algorithm (McKay and Wormald '90): DEG

- Use rejection to enforce uniformity of $P' \in S_{i-1}$.
- Once a random switching $(P, P')$ is chosen.
  - perform an f-rejection to equalise the probability of a switching for all $P \in S_i$;
  - perform a b-rejection to equalise the probability that $P'$ is reached for all $P' \in S_{i-1}$. 
Switching algorithm (McKay and Wormald '90): DEG

- Use rejection to enforce uniformity of $P' \in S_{i-1}$.
- Once a random switching $(P, P')$ is chosen.
  - perform an f-rejection to equalise the probability of a switching for all $P \in S_i$;
  - perform a b-rejection to equalise the probability that $P'$ is reached for all $P' \in S_{i-1}$.

Uniform generation of random regular graphs
Switching algorithm (McKay and Wormald '90): DEG

\[ f\text{-rejected with probability } 1 - \frac{N(P)}{\max_{P'' \in S_i} \{N(P'')\}}; \]

\[ b\text{-rejected with probability } 1 - \frac{\min_{P'' \in S_{i-1}} \{N'(P'')\}}{N'(P')} ; \]
Switching algorithm (McKay and Wormald '90): DEG

- Inductively, the final pairing $P'$ is uniformly distributed in $S_0$, if no rejection has occurred.
- $P'$ represents a uniformly random $d$-regular graph.
- The probability of a rejection is away from 1 if $d = O(n^{1/3})$. 

Switching algorithm (McKay and Wormald ’90): DEG

- Inductively, the final pairing $P'$ is uniformly distributed in $S_0$, if no rejection has occurred.
- $P'$ represents a uniformly random $d$-regular graph.
- The probability of a rejection is away from 1 if $d = O(n^{1/3})$. 

Uniform generation of random regular graphs
Switching algorithm (McKay and Wormald ’90): DEG

- Inductively, the final pairing $P'$ is uniformly distributed in $S_0$, if no rejection has occurred.
- $P'$ represents a uniformly random $d$-regular graph.
- The probability of a rejection is away from 1 if $d = O(n^{1/3})$. 

Uniform generation of random regular graphs
For $d \gg n^{1/3}$

For $d \gg n^{1/3}$, DEG is inefficient as the probability of a rejection during the algorithm is very close to 1.

This is because

- variation of $N(P)$ is too big for $P \in S_i$, causing big f-rejection probability;
- variation of $N'(P')$ is too big for $P \in S_{i-1}$, causing big b-rejection probability.

REG reduces the probabilities of f and b-rejections, and thus runs efficiently for larger $d$. 
For $d \gg n^{1/3}$

For $d \gg n^{1/3}$, DEG is inefficient as the probability of a rejection during the algorithm is very close to 1.

This is because
- variation of $N(P)$ is too big for $P \in S_i$, causing big $f$-rejection probability;
- variation of $N'(P')$ is too big for $P \in S_{i-1}$, causing big $b$-rejection probability.

REG reduces the probabilities of $f$ and $b$-rejections, and thus runs efficiently for larger $d$. 
For $d \gg n^{1/3}$

For $d \gg n^{1/3}$, DEG is inefficient as the probability of a rejection during the algorithm is very close to 1.

This is because

- variation of $N(P)$ is too big for $P \in S_i$, causing big f-rejection probability;
- variation of $N'(P')$ is too big for $P \in S_{i-1}$, causing big b-rejection probability.

REG reduces the probabilities of f and b-rejections, and thus runs efficiently for larger $d$. 
REG: reduce f-rejection

To reduce f-rejection, we allow some switchings that were forbidden before, these switchings are categorised into different classes (e.g. class B).
Class A: the typical one

Uniform generation of random regular graphs
Class B: the one forbidden by DEG
Reduce b-rejection

To reduce b-rejection, we perform occasionally some other types of switchings that will boost the pairings that were underrepresented, i.e. with small value of $N'(P')$. 
Type I

Uniform generation of random regular graphs
Type I

Uniform generation of random regular graphs
Type II

Uniform generation of random regular graphs
New features

- Each switching is associated with a type $\tau \in \mathcal{T}$ and a class $\alpha \in \mathcal{A}$;
- The Markov chain among $(S_i)_{i \geq 0}$ cycles;
- The algorithm needs to decide which type of switchings to be performed in each step.
New features

- Each switching is associated with a type $\tau \in \mathcal{T}$ and a class $\alpha \in \mathcal{A}$;
- The Markov chain among $(S_i)_{i \geq 0}$ cycles;
- The algorithm needs to decide which type of switchings to be performed in each step.
New features

- Each switching is associated with a type $\tau \in \mathcal{T}$ and a class $\alpha \in \mathcal{A}$;
- The Markov chain among $(S_i)_{i \geq 0}$ cycles;
- The algorithm needs to decide which type of switchings to be performed in each step.
New features

- Each switching is associated with a type $\tau \in \mathcal{T}$ and a class $\alpha \in \mathcal{A}$;
- The Markov chain among $(S_i)_{i \geq 0}$ cycles;
- The algorithm needs to decide which type of switchings to be performed in each step.
Transitions into $S_j$
New elements of the analysis

- We no longer equalise the transition probabilities out of or into pairings in a certain $S_i$, instead we equalise the expected number of times each pairing is visited in a given $S_i$.
- This equalisation is done within each switching class $\alpha \in A$.
- For each $P \in S_i$, the algorithm probabilistically determines which switching type $\tau$ to be performed. There can be a t-rejection if no type is chosen.
New elements of the analysis

- We no longer equalise the transition probabilities out of or into pairings in a certain $S_i$, instead we equalise the expected number of times each pairing is visited in a given $S_i$.
- This equalisation is done within each switching class $\alpha \in \mathcal{A}$.
- For each $P \in S_i$, the algorithm probabilistically determines which switching type $\tau$ to be performed. There can be a t-rejection if no type is chosen.
New elements of the analysis

- We no longer equalise the transition probabilities out of or into pairings in a certain $S_i$, instead we equalise the expected number of times each pairing is visited in a given $S_i$.
- This equalisation is done within each switching class $\alpha \in \mathcal{A}$.
- For each $P \in S_i$, the algorithm probabilistically determines which switching type $\tau$ to be performed. There can be a $t$-rejection if no type is chosen.
New elements of the analysis

- We no longer equalise the transition probabilities out of or into pairings in a certain $S_i$, instead we equalise the expected number of times each pairing is visited in a given $S_i$.
- This equalisation is done within each switching class $\alpha \in A$.
- For each $P \in S_i$, the algorithm probabilistically determines which switching type $\tau$ to be performed. There can be a t-rejection if no type is chosen.
Algorithm framework

Given $P \in S_i$,

(i) If $i = 0$, output $P$.
(ii) Choose a type: choose $\tau$ with probability $\rho_{\tau}(i)$, and with the remaining probability, $1 - \sum_{\tau} \rho_{\tau}(i)$, perform a t-rejection. Then select u.a.r. one of the type $\tau$ switchings that can be performed on $P$.
(iii) Let $P'$ be the element that the selected switching would produce if applied to $P$, let $\alpha$ be the class of the selected switching and let $i' = S(P')$. Perform an f-rejection with probability $1 - N_{\tau}(P)/\max\{N_{\tau}(P'')\}$ and then perform a b-rejection with probability $1 - \min\{N'_{\alpha}(P'')\}/N'_{\alpha}(P')$;
(iv) if no rejection occurred, replace $P$ with $P'$.
(v) Repeat until $i = 0$. 
The rest of the definition

- The algorithm is defined after we fix all parameters $\rho_\tau(i)$, $\tau \in \{I, II\}$.
- These probabilities are designed to equalise the expected number of times each pairing is visited in a certain $S_i$.
- Thereby we deduce a system of equations that $\rho_\tau(i)$ must satisfy.
- We describe an efficient scheme to find a desirable solution to the system of equations.
- done!
The rest of the definition

- The algorithm is defined after we fix all parameters $\rho_{\tau}(i)$, $\tau \in \{I, II\}$.
- These probabilities are designed to equalise the expected number of times each pairing is visited in a certain $S_i$.
- Thereby we deduce a system of equations that $\rho_{\tau}(i)$ must satisfy.
- We describe an efficient scheme to find a desirable solution to the system of equations.
- done!
The rest of the definition

- The algorithm is defined after we fix all parameters $\rho_\tau(i)$, $\tau \in \{I, II\}$.
- These probabilities are designed to equalise the expected number of times each pairing is visited in a certain $S_i$.
- Thereby we deduce a system of equations that $\rho_\tau(i)$ must satisfy.
- We describe an efficient scheme to find a desirable solution to the system of equations.
- done!
The rest of the definition

- The algorithm is defined after we fix all parameters $\rho_\tau(i)$, $\tau \in \{I, II\}$.
- These probabilities are designed to equalise the expected number of times each pairing is visited in a certain $S_i$.
- Thereby we deduce a system of equations that $\rho_\tau(i)$ must satisfy.
- We describe an efficient scheme to find a desirable solution to the system of equations.
- done!
The rest of the definition

- The algorithm is defined after we fix all parameters $\rho_\tau(i)$, $\tau \in \{I, II\}$.
- These probabilities are designed to equalise the expected number of times each pairing is visited in a certain $S_i$.
- Thereby we deduce a system of equations that $\rho_\tau(i)$ must satisfy.
- We describe an efficient scheme to find a desirable solution to the system of equations.
- done!
The rest of the definition

- The algorithm is defined after we fix all parameters $\rho_\tau(i)$, $\tau \in \{I, II\}$.
- These probabilities are designed to equalise the expected number of times each pairing is visited in a certain $S_i$.
- Thereby we deduce a system of equations that $\rho_\tau(i)$ must satisfy.
- We describe an efficient scheme to find a desirable solution to the system of equations.
- done!
Time complexity

**Theorem (Gao and Wormald ’15)**

REG generates $d$-regular graphs uniformly at random. For $d = o(\sqrt{n})$, the expected running time of REG for generating one graph is $O(d^3 n)$. 

Uniform generation of random regular graphs
Approximate sampler: REG*

Most of the running time of REG is spent on computing the probability of a b-rejection in each step. If we ignore b-rejections, we get a linear-running-time approximate sampler REG*.

Theorem (Gao and Wormald ’15)

For \( d = o(\sqrt{n}) \), REG* generates a random \( d \)-regular graph whose total variation distance from the uniform distribution is \( o(1) \). The expected running time of REG* is \( O(dn) \).
Approximate sampler: REG*

Most of the running time of REG is spent on computing the probability of a b-rejection in each step. If we ignore b-rejections, we get a linear-running-time approximate sampler REG*.

Theorem (Gao and Wormald ’15)

For $d = o(\sqrt{n})$, REG* generates a random $d$-regular graph whose total variation distance from the uniform distribution is $o(1)$. The expected running time of REG* is $O(dn)$.
Approximate sampler: REG*

Most of the running time of REG is spent on computing the probability of a b-rejection in each step. If we ignore b-rejections, we get a linear-running-time approximate sampler REG*.

Theorem (Gao and Wormald ’15)

For $d = o(\sqrt{n})$, REG* generates a random $d$-regular graph whose total variation distance from the uniform distribution is $o(1)$. The expected running time of REG* is $O(dn)$. 
Future directions

- Larger $d$.
- General degree sequences.
- Heavily-tailed degree sequences such as power-law sequences.
- New switchings.