Perfect simulation for the $M/G/c$ queue

Stephen Connor
University of York

Joint work with Wilfrid Kendall (University of Warwick)

Statistics Seminar
Durham University
15 Feb 2015

Introduction

Markov chain Monte Carlo (MCMC)

- **AIM**: to obtain a sample from a particular distribution π
- **METHOD**:
  - (i) design a Markov chain with stationary distribution π
  - (ii) run chain until near equilibrium
  - (iii) sample from the chain
- **PROBLEM**: How long is the 'burn-in' period? i.e. how long should we wait in step (ii)?

Markov chain Monte Carlo (MCMC)

- **AIM**: to obtain a sample from a particular distribution π
- **METHOD**:
  - (i) design a Markov chain with stationary distribution π
  - (ii) run chain until near equilibrium
  - (iii) sample from the chain
- **PROBLEM**: How long is the 'burn-in' period? i.e. how long should we wait in step (ii)?
- **POSSIBLE SOLUTIONS**:
  - guess from simulation output
  - estimate it analytically

Markov chain Monte Carlo (MCMC)

- **AIM**: to obtain a sample from a particular distribution π
- **METHOD**:
  - (i) design a Markov chain with stationary distribution π
  - (ii) run chain until near equilibrium
  - (iii) sample from the chain
- **PROBLEM**: How long is the 'burn-in' period? i.e. how long should we wait in step (ii)?
- **POSSIBLE SOLUTIONS**:
  - guess from simulation output
  - estimate it analytically

Or use *perfect simulation!*

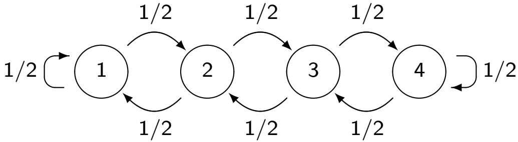
Modify an MCMC algorithm so as to produce an *exact* draw from π, at the cost of a random length run-time

Heuristic idea

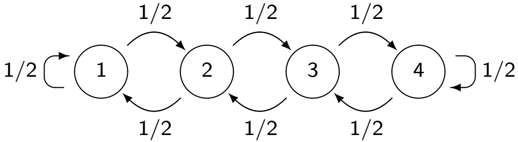
Think of a (hypothetical) version of the chain, $\tilde{X}$, which was started by your (presumably distant) ancestor from some state x at time $-\infty$:

- at time zero this chain is in equilibrium: $\tilde{X}_0^{x,-\infty} \sim \pi$
- most perfect simulation algorithms try to determine the value of $\tilde{X}_0^{x,-\infty}$ by looking into the past only a *finite* number of steps...

Simple example



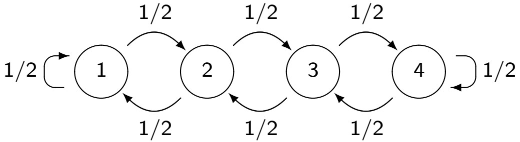
Simple example



Run chains from all states using a common update function f (and the same source of randomness u for all chains):

$$f(x,u) = \begin{cases} \min(x+1,4) & \text{if } u \leq 1/2 \\ \max(x-1,1) & \text{if } u > 1/2\,. \end{cases}$$

Simple example

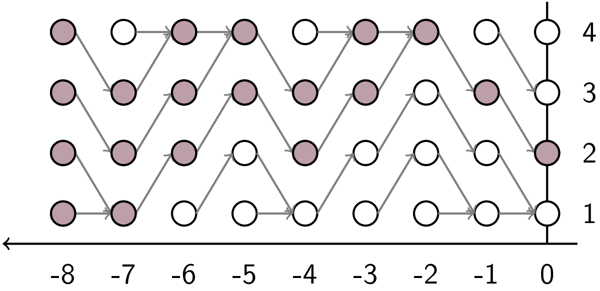


Run chains from all states using a common update function f (and the same source of randomness u for all chains):

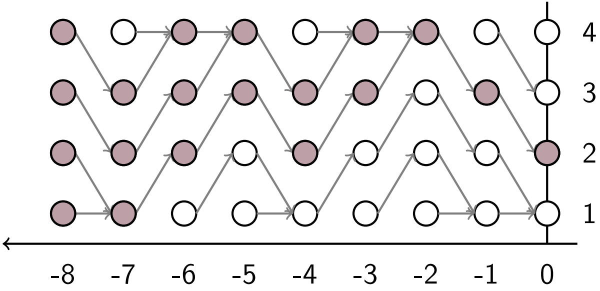
$$f(x,u) = \begin{cases} \min(x+1,4) & \text{if } u \le 1/2 \\ \max(x-1,1) & \text{if } u > 1/2\,. \end{cases}$$

Algorithm:

- set $n = 1$;
- run chains $X^{x,-n}$ for all $x = 1,2,3,4$ up to time 0;
  - if $X_0^{x,-n} = X_0$ **for all** x, return X_0;
  - else set $n \leftarrow 2n$ and repeat, re-using randomness over $[-n, 0]$.



For this realisation, when $n = 8$ is reached, all of the target chains have the same value at time zero: $X_0^{x,-8} = 2$ in this case. Coalescence time is $T^* = 7$.



For this realisation, when $n = 8$ is reached, all of the target chains have the same value at time zero: $X_0^{x,-8} = 2$ in this case. Coalescence time is $T^* = 7$.

Claim

$$X_0 := X_0^{x,-T^*} \sim \pi$$

This is **Coupling From The Past!**

Dominated CFTP

Dominated CFTP

This really only works when the state space is (essentially) bounded. (Foss & Tweedie, 1998: CFTP is theoretically possible $\Leftrightarrow$ X is *uniformly ergodic*.)

The idea is to identify a time in the past from which *"chains from all possible starting states have coalesced by time zero"*.

Dominated CFTP

This really only works when the state space is (essentially) bounded. (Foss & Tweedie, 1998: CFTP is theoretically possible $\Leftrightarrow X$ is *uniformly ergodic*.)
The idea is to identify a time in the past from which *"chains from all possible starting states have coalesced by time zero"*.

But we could also obtain a sample from π by identifying a time in the past such that *"all earlier starts from a specific state x lead to the same result at time zero"*.

Dominated CFTP

This really only works when the state space is (essentially) bounded. (Foss & Tweedie, 1998: CFTP is theoretically possible $\Leftrightarrow X$ is *uniformly ergodic*.)
The idea is to identify a time in the past from which *"chains from all possible starting states have coalesced by time zero"*.

But we could also obtain a sample from π by identifying a time in the past such that *"all earlier starts from a specific state x lead to the same result at time zero"*.

Main idea

Replace upper and lower processes by *random* processes in statistical equilibrium ('envelope processes')

Example (adapted from Kendall, 1997)

- X is nonlinear immigration-death process:
  $X \to X - 1$ at rate μX;
  $X \to X + 1$ at rate α_X, where $\alpha_X \leq \alpha_\infty < \infty$.
  No max means not uniformly ergodic, so no classic CFTP!

Example (adapted from Kendall, 1997)

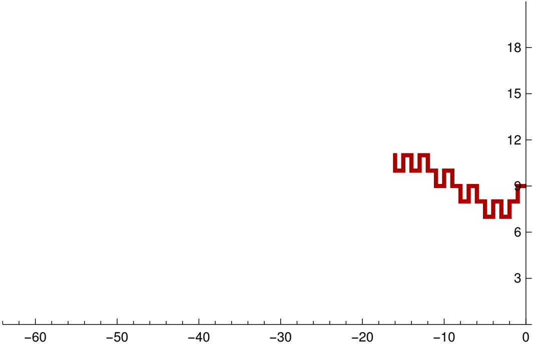
- X is nonlinear immigration-death process:
  $X \to X - 1$ at rate μX;
  $X \to X + 1$ at rate α_X, where $\alpha_X \leq \alpha_\infty < \infty$.
  No max means not uniformly ergodic, so no classic CFTP!
- Bound by *linear* immigration-death process Y:
  $Y \to Y - 1$ at rate μY;
  $Y \to Y + 1$ at rate α_∞.

Example (adapted from Kendall, 1997)

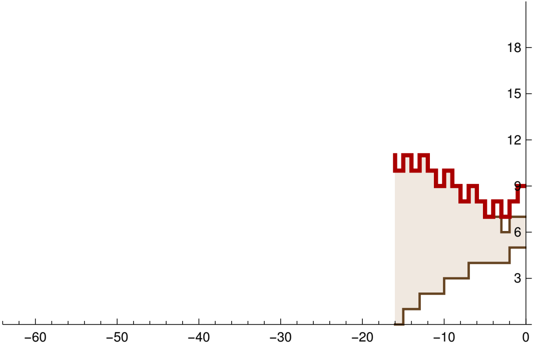
- X is nonlinear immigration-death process:
  $X \to X - 1$ at rate μX;
  $X \to X + 1$ at rate α_X, where $\alpha_X \leq \alpha_\infty < \infty$.
  No max means not uniformly ergodic, so no classic CFTP!
- Bound by *linear* immigration-death process Y:
  $Y \to Y - 1$ at rate μY;
  $Y \to Y + 1$ at rate α_∞.
- Produce X from Y by *censoring* births and deaths:
  if $Y \to Y - 1$ then $X \to X - 1$ with cond. prob. X/Y;
  if $Y \to Y + 1$ then $X \to X + 1$ with cond. prob. α_X/α_∞.

- Because Y is *reversible*, with *known equilibrium* (via detailed balance), we can simulate Y *backwards*.
- Given a (forwards) trajectory of Y over $[-n, 0]$, we can build trajectories of X starting at every $0 \leq X_{-n} \leq Y_{-n}$ *staying below* Y until time 0.
- These can be checked for coalescence!

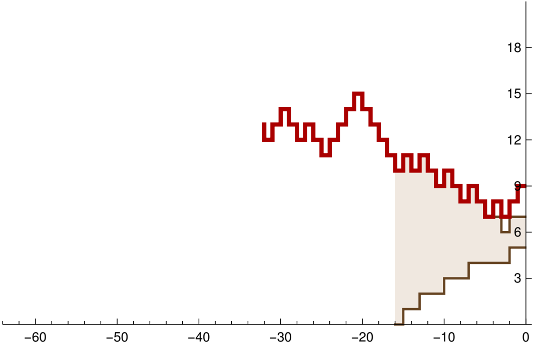
- Because Y is *reversible*, with *known equilibrium* (via detailed balance), we can simulate Y *backwards*.
- Given a (forwards) trajectory of Y over $[-n, 0]$, we can build trajectories of X starting at every $0 \leq X_{-n} \leq Y_{-n}$ *staying below Y* until time 0.
- These can be checked for coalescence!



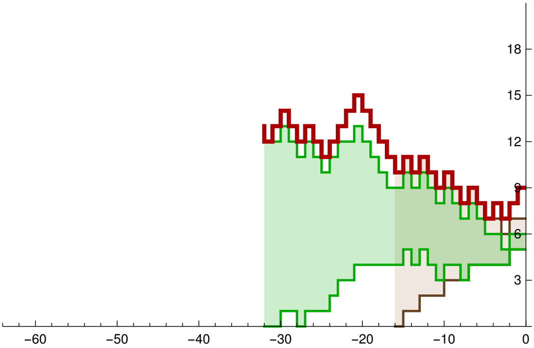
- Because Y is *reversible*, with *known equilibrium* (via detailed balance), we can simulate Y *backwards*.
- Given a (forwards) trajectory of Y over $[-n, 0]$, we can build trajectories of X starting at every $0 \leq X_{-n} \leq Y_{-n}$ *staying below* Y until time 0.
- These can be checked for coalescence!



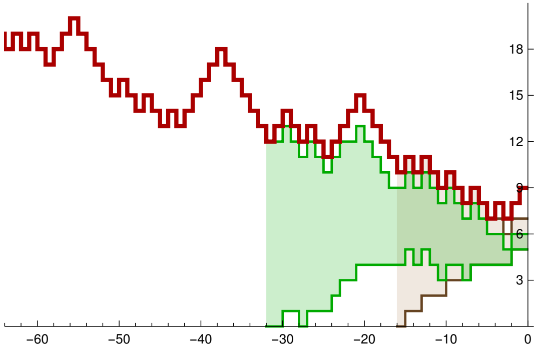
- Because Y is *reversible*, with *known equilibrium* (via detailed balance), we can simulate Y *backwards*.
- Given a (forwards) trajectory of Y over $[-n, 0]$, we can build trajectories of X starting at every $0 \leq X_{-n} \leq Y_{-n}$ *staying below Y* until time 0.
- These can be checked for coalescence!



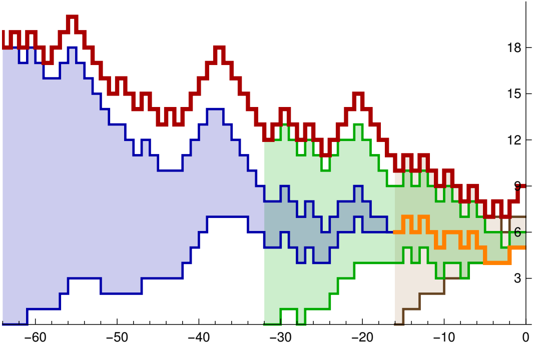
- Because Y is *reversible*, with *known equilibrium* (via detailed balance), we can simulate Y *backwards*.
- Given a (forwards) trajectory of Y over $[-n, 0]$, we can build trajectories of X starting at every $0 \leq X_{-n} \leq Y_{-n}$ *staying below* Y until time 0.
- These can be checked for coalescence!



- Because Y is *reversible*, with *known equilibrium* (via detailed balance), we can simulate Y *backwards*.
- Given a (forwards) trajectory of Y over $[-n, 0]$, we can build trajectories of X starting at every $0 \leq X_{-n} \leq Y_{-n}$ *staying below* Y until time 0.
- These can be checked for coalescence!



- Because Y is *reversible*, with *known equilibrium* (via detailed balance), we can simulate Y *backwards*.
- Given a (forwards) trajectory of Y over $[-n, 0]$, we can build trajectories of X starting at every $0 \leq X_{-n} \leq Y_{-n}$ *staying below* Y until time 0.
- These can be checked for coalescence!



Dominated CFTP summary

Basic ingredients:

- *dominating process*
  - draw from equilibrium
  - simulate backwards in time
- *sandwiching*

  $$\text{Lower}_{\text{late}} \preccurlyeq \text{Lower}_{\text{early}} \preccurlyeq \ldots \preccurlyeq \text{Targets} \preccurlyeq \ldots \preccurlyeq \text{Upper}_{\text{early}} \preccurlyeq \text{Upper}_{\text{late}}$$

- *coalescence*

  eventually a Lower and an Upper process must coalesce

Dominated CFTP summary

Basic ingredients:

- *dominating process*
  - draw from equilibrium
  - simulate backwards in time
- *sandwiching*

$$\text{Lower}_{\text{late}} \preccurlyeq \text{Lower}_{\text{early}} \preccurlyeq \ldots \preccurlyeq \text{Targets} \preccurlyeq \ldots \preccurlyeq \text{Upper}_{\text{early}} \preccurlyeq \text{Upper}_{\text{late}}$$

- *coalescence*
  eventually a Lower and an Upper process must coalesce

Surprisingly general!

- domCFTP has been applied in numerous practical settings
- Kendall (2004) shows domCFTP possible *in principle* for all geometrically ergodic (GE) chains
- C. & Kendall (2007) extend this to a class of non-GE positive-recurrent chains

Queues

$M/G/c$ Queues

- Customers arrive at times of a Poisson process: interarrival times $T_n \sim \text{Exp}(\lambda)$
- Service durations S_n are i.i.d. with $\mathbb{E}[S] = 1/\mu$ (and we assume that $\mathbb{E}\left[S^2\right] < \infty$)
- Customers are served by c servers, on a First Come First Served (FCFS) basis

Queue is *stable* iff $\lambda/\mu < c$, and *super-stable* if $\lambda/\mu < 1$.

The (ordered) workload vector just before the arrival of the n^{th} customer satisfies the *Kiefer-Wolfowitz* recursion:

$$\mathbf{W}_{n+1} = R(\mathbf{W}_n + S_n\delta_1 - T_n\mathbf{1})^+ \quad \text{for } n \geq 0$$

- add workload S_n to first coordinate of $\mathbf{W}_n$ (server currently with least work)
- subtract T_n from every coordinate (work done between arrivals)
- reorder the coordinates in increasing order
- replace negative values by zeros.

Our goal is to sample from the equilibrium distribution of this workload vector.

Super-stable $M/G/c$ and domCFTP

Sigman (2011) pioneered domCFTP for multiserver queues. Key step: find amenable dominating process.

- Restrict to **super-stable case**

Super-stable $M/G/c$ and domCFTP

Sigman (2011) pioneered domCFTP for multiserver queues. Key step: find amenable dominating process.

- Restrict to **super-stable case**
- Workload of $M/G/c$ dominated by that of $M/G/1$

Super-stable $M/G/c$ and domCFTP

Sigman (2011) pioneered domCFTP for multiserver queues. Key step: find amenable dominating process.

- Restrict to **super-stable case**
- Workload of $M/G/c$ dominated by that of $M/G/1$
- Time-reversal: same $M/G/1$ workload if use **PS** not **FCFS**

Super-stable $M/G/c$ and domCFTP

Sigman (2011) pioneered domCFTP for multiserver queues. Key step: find amenable dominating process.

- Restrict to **super-stable case**
- Workload of $M/G/c$ dominated by that of $M/G/1$
- Time-reversal: same $M/G/1$ workload if use **PS** not **FCFS**
- But $M/G/1\,[PS]$ is **dynamically reversible** (so we can reverse time in equilibrium)

Super-stable $M/G/c$ and domCFTP

Sigman (2011) pioneered domCFTP for multiserver queues. Key step: find amenable dominating process.

- Restrict to **super-stable case**
- Workload of $M/G/c$ dominated by that of $M/G/1$
- Time-reversal: same $M/G/1$ workload if use **PS** not **FCFS**
- But $M/G/1\,[PS]$ is **dynamically reversible** (so we can reverse time in equilibrium)
- Recover $M/G/1\,[FCFS]$ from workload of $M/G/1\,[PS]$

Super-stable $M/G/c$ and domCFTP

Sigman (2011) pioneered domCFTP for multiserver queues. Key step: find amenable dominating process.

- Restrict to **super-stable case**
- Workload of $M/G/c$ dominated by that of $M/G/1$
- Time-reversal: same $M/G/1$ workload if use **PS** not **FCFS**
- But $M/G/1\,[PS]$ is **dynamically reversible** (so we can reverse time in equilibrium)
- Recover $M/G/1\,[FCFS]$ from workload of $M/G/1\,[PS]$
- $M/G/c\,[FCFS]$ workload smaller than $M/G/1\,[FCFS]$

Super-stable $M/G/c$ and domCFTP

Sigman (2011) pioneered domCFTP for multiserver queues. Key step: find amenable dominating process.

- Restrict to **super-stable case**
- Workload of $M/G/c$ dominated by that of $M/G/1$
- Time-reversal: same $M/G/1$ workload if use **PS** not **FCFS**
- But $M/G/1\,[PS]$ is **dynamically reversible** (so we can reverse time in equilibrium)
- Recover $M/G/1\,[FCFS]$ from workload of $M/G/1\,[PS]$
- $M/G/c\,[FCFS]$ workload smaller than $M/G/1\,[FCFS]$
- Coalescence forced when $M/G/1\,[PS]$ empties.
  (Finite mean if finite second moment of service time.)

However...

This idea is great, but has some drawbacks.

1. Coalescence is achieved by running backwards in time until $M/G/1\,[PS]$ empties.
   This will be inefficient if the target $M/G/c$ workload is such that $M/G/1$ is nearly unstable.

However...

This idea is great, but has some drawbacks.

1. Coalescence is achieved by running backwards in time until $M/G/1\,[PS]$ empties.
   This will be inefficient if the target $M/G/c$ workload is such that $M/G/1$ is nearly unstable.
2. Worse, the interesting case for $M/G/c$ is *exactly* when the $M/G/1$ is **not** stable!

However...

This idea is great, but has some drawbacks.

1. Coalescence is achieved by running backwards in time until $M/G/1\,[PS]$ empties.
   This will be inefficient if the target $M/G/c$ workload is such that $M/G/1$ is nearly unstable.
2. Worse, the interesting case for $M/G/c$ is *exactly* when the $M/G/1$ is **not** stable!
3. Sigman (2012) uses an importance-sampling approach for the *stable* case, but this algorithm has a run-time with infinite mean!

Dominated CFTP for *stable* $M/G/c$ queues

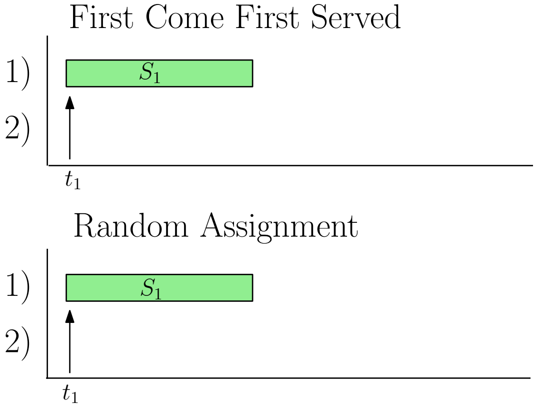
We need to find a dominating process for our $M_\lambda/G/c$ $[FCFS]$ queue X.

C. & Kendall (2015): dominate with $M/G/c$ $[RA]$

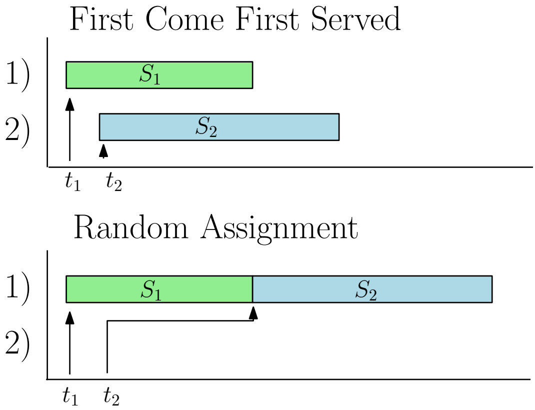
- RA = **random assignment**, so c independent copies of $M_{\lambda/c}/G/1$.
- Evidently stable iff $M/G/c$ is stable.
- Easy to simulate in equilibrium, and in reverse.
- Care needed with domination arguments...

Let Y be a $M/G/c\,[RA]$ queue. If Y uses the *same arrival times* and *service durations* as X (our $M/G/c\,[FCFS]$ queue), even though its allocation rule is less efficient it **doesn't** follow that the number of customers who have departed from X by time t will be at least as big as the number who have departed from Y...

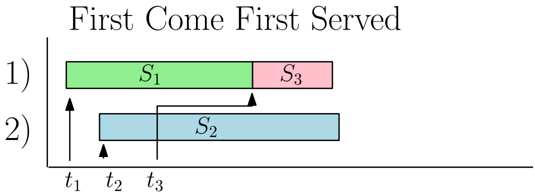
Let Y be a $M/G/c\,[RA]$ queue. If Y uses the *same arrival times* and *service durations* as X (our $M/G/c\,[FCFS]$ queue), even though its allocation rule is less efficient it **doesn't** follow that the number of customers who have departed from X by time t will be at least as big as the number who have departed from Y...

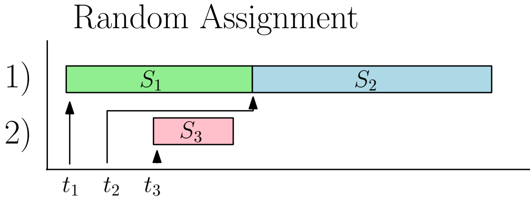


Let Y be a $M/G/c\,[RA]$ queue. If Y uses the *same arrival times* and *service durations* as X (our $M/G/c\,[FCFS]$ queue), even though its allocation rule is less efficient it **doesn't** follow that the number of customers who have departed from X by time t will be at least as big as the number who have departed from Y...



Let Y be a $M/G/c\,[RA]$ queue. If Y uses the *same arrival times* and *service durations* as X (our $M/G/c\,[FCFS]$ queue), even though its allocation rule is less efficient it **doesn't** follow that the number of customers who have departed from X by time t will be at least as big as the number who have departed from Y...

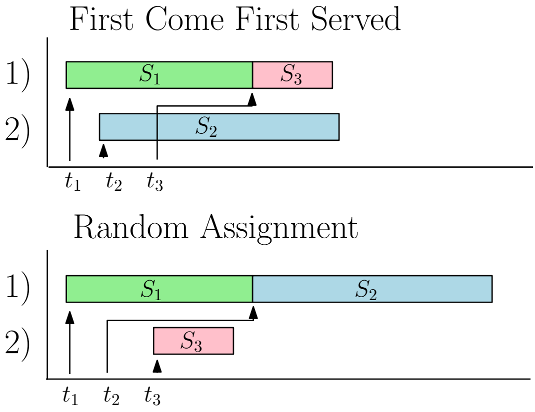




It **is** true that queue length under FCFS is stochastically dominated by that under RA. But the result does **not** hold for sample path domination!

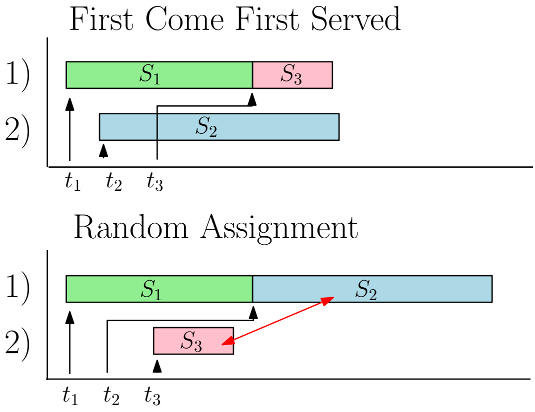
It **is** true that queue length under FCFS is stochastically dominated by that under RA. But the result does **not** hold for sample path domination!

But we **can** get this domination if we assign service S_n to the n^{th} **initiation of service**!



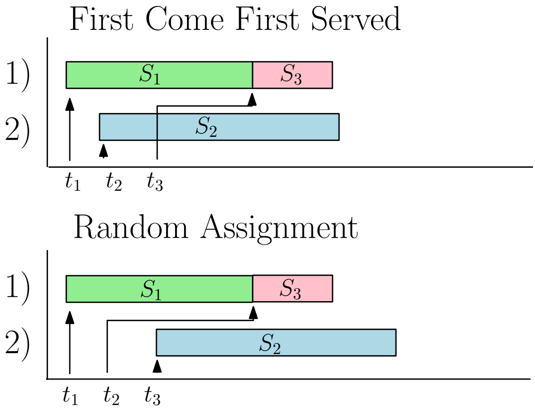
It **is** true that queue length under FCFS is stochastically dominated by that under RA. But the result does **not** hold for sample path domination!

But we **can** get this domination if we assign service S_n to the n^{th} **initiation of service**!



It **is** true that queue length under FCFS is stochastically dominated by that under RA. But the result does **not** hold for sample path domination!

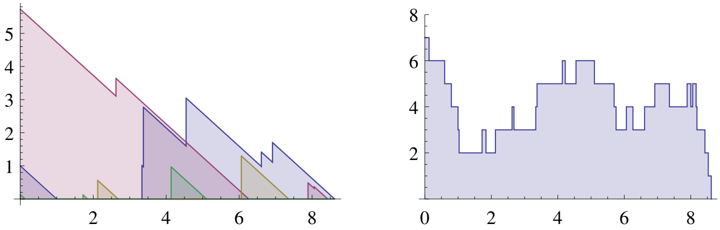
But we **can** get this domination if we assign service S_n to the n^{th} **initiation of service**!



Algorithm 1

(Processes run backwards are crowned with a hat.)

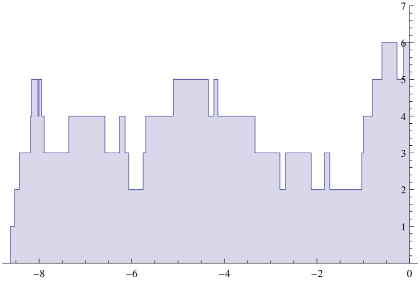
1. Simulate a $[M/G/1\ PS]^c$ process $\hat{Y}$, in statistical equilibrium, until it first empties (at time $\hat{\tau}$)



Algorithm 1

(Processes run backwards are crowned with a hat.)

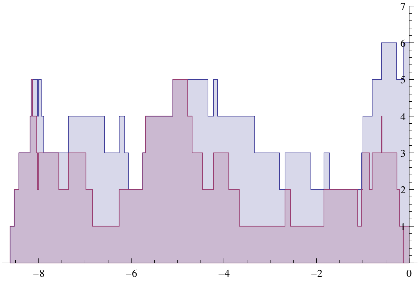
1. Simulate a $[M/G/1\ PS]^c$ process $\hat{Y}$, in statistical equilibrium, until it first empties (at time $\hat{\tau}$)
2. Set $\tau = -\hat{\tau}$, and use the path of $\hat{Y}$ to construct its (dynamic) time reversal: thus build $(Y(t) : \tau \leq t \leq 0)$, an $M/G/c\,[RA]$ process



Algorithm 1

(Processes run backwards are crowned with a hat.)

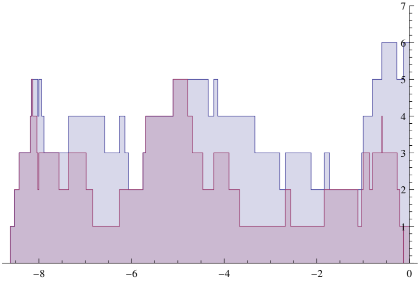
1. Simulate a $[M/G/1\ PS]^c$ process $\hat{Y}$, in statistical equilibrium, until it first empties (at time $\hat{\tau}$)
2. Set $\tau = -\hat{\tau}$, and use the path of $\hat{Y}$ to construct its (dynamic) time reversal: thus build $(Y(t) : \tau \leq t \leq 0)$, an $M/G/c\,[RA]$ process
3. Use Y to evolve X, an $M/G/c\ [FCFS]$ process, over $[\tau, 0]$, started from empty



Algorithm 1

(Processes run backwards are crowned with a hat.)

1. Simulate a $[M/G/1\,PS]^c$ process $\hat{Y}$, in statistical equilibrium, until it first empties (at time $\hat{\tau}$)
2. Set $\tau = -\hat{\tau}$, and use the path of $\hat{Y}$ to construct its (dynamic) time reversal: thus build $(Y(t) : \tau \leq t \leq 0)$, an $M/G/c\,[RA]$ process
3. Use Y to evolve X, an $M/G/c\ [FCFS]$ process, over $[\tau, 0]$, started from empty
4. Return X_0



Algorithm 1

- works!
- has finite mean run-time (time taken for $\hat{Y}$ to empty is finite iff $\mathbb{E}\left[S^2\right] < \infty$)
- is inefficient: if $1 \ll \rho < c$ then $\hat{Y}$ will take a long time to empty completely

Algorithm 1

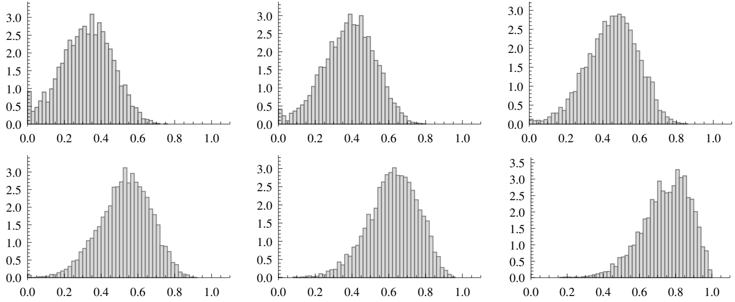
- works!
- has finite mean run-time (time taken for $\hat{Y}$ to empty is finite iff $\mathbb{E}\left[S^2\right] < \infty$)
- is inefficient: if $1 \ll \rho < c$ then $\hat{Y}$ will take a long time to empty completely

We can do better than this by simulating our dominating process $\hat{Y}$ until **each server** has emptied at least once, and then using **sandwiching processes** to try to establish coalescence much faster.

Algorithm 2

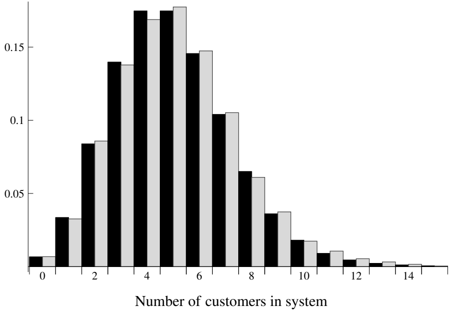
1. Set $\hat{T} = 1$.
2. Simulate a $[M/G/1\ PS]^c$ process $\hat{Y}$, in statistical equilibrium as follows: evolve the queue for server j (independently of all other servers) until the first time $\hat{\tau}_j \geq \hat{T}$ that **this server** is empty, for $j = 1, \ldots, c$.
3. Construct Y_j, an $M/G/1\ [FCFS]$ process over $[-\hat{\tau}_j, 0]$, for $j = 1, \ldots, c$.
4. Construct upper and lower **sandwiching processes**, $U_{[T,0]}$ and $L_{[T,0]}$. ($M/G/c\,[FCFS]$ queues.)
5. Check for **coalescence of workload vectors**; if $L_{[T,0]}(0) \neq U_{[T,0]}(0)$ then set $\hat{T} \leftarrow 2\hat{T}$ and repeat

Simulation output: workload at busiest six servers



Equilibrium distribution of final 6 coordinates of Kiefer-Wolfowitz workload vector: $\lambda = c = 25$, $S \sim \text{Uniform}[0, 1]$.
(5,000 draws, Algorithm 2)

Simulation output: number of people in system

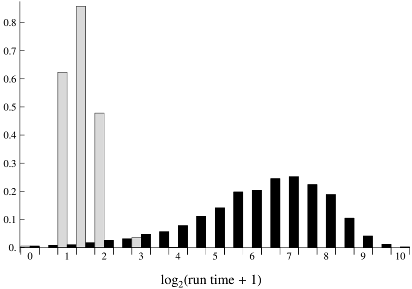


Number of customers for $M/M/c$ queue in equilibrium.
$\lambda = 10$, $\mu = 2$, $c = 10$.

Black bars show theoretical number of customers in system; grey bars give results of 5,000 draws using Algorithm 2.
χ^2-test: p-value 0.62.

Algorithm performance

$M/M/c$ queue. (5,000 runs, $\lambda = 10$, $\mu = 2$, $c = 10$.)



Black bars show $\log_2(\tilde{\tau} + 1)$ for Algorithm 1
($\tilde{\tau}$ = first time at which $\tilde{Y}$ empties).
Grey bars show distribution of $\log_2(\tilde{T} + 1)$ for Algorithm 2
($\tilde{T}$ = smallest time needed to detect coalescence using binary back-off).

We can bound run-times using

- *alternating renewal process theory* for Algorithm 1
- *supermartingale* ideas for Algorithm 2 (heuristic for $M/M/c$ queues only)

We can bound run-times using

- *alternating renewal process theory* for Algorithm 1
- *supermartingale* ideas for Algorithm 2 (heuristic for $M/M/c$ queues only)

Mean run-time:

λ	c	ρ	**lower bound** Algorithm 1	**upper bound** Algorithm 2
10	10	5		
20	20	10		
30	30	15		
40	40	20		
50	50	25		
30	30	5		
30	30	10		
30	30	20		
30	30	25		
30	30	29.5		

We can bound run-times using

- *alternating renewal process theory* for Algorithm 1
- *supermartingale* ideas for Algorithm 2 (heuristic for $M/M/c$ queues only)

Mean run-time:

λ	c	ρ	**lower bound** Algorithm 1	**upper bound** Algorithm 2
10	10	5	102	
20	20	10	52429	
30	30	15	3.58×10^7	
40	40	20	2.75×10^{10}	
50	50	25	2.25×10^{13}	
30	30	5		
30	30	10		
30	30	20		
30	30	25		
30	30	29.5		

We can bound run-times using

- *alternating renewal process theory* for Algorithm 1
- *supermartingale* ideas for Algorithm 2 (heuristic for $M/M/c$ queues only)

Mean run-time:

λ	c	ρ	**lower bound** Algorithm 1	**upper bound** Algorithm 2
10	10	5	102	5
20	20	10	52429	10
30	30	15	3.58×10^{7}	15
40	40	20	2.75×10^{10}	20
50	50	25	2.25×10^{13}	25
30	30	5		
30	30	10		
30	30	20		
30	30	25		
30	30	29.5		

We can bound run-times using

- *alternating renewal process theory* for Algorithm 1
- *supermartingale* ideas for Algorithm 2 (heuristic for $M/M/c$ queues only)

Mean run-time:

λ	c	ρ	**lower bound** Algorithm 1	**upper bound** Algorithm 2
10	10	5	102	5
20	20	10	52429	10
30	30	15	3.58×10^{7}	15
40	40	20	2.75×10^{10}	20
50	50	25	2.25×10^{13}	25
30	30	5	7.88	
30	30	10	6392	
30	30	20	6.86×10^{12}	
30	30	25	7.37×10^{21}	
30	30	29.5	7.37×10^{51}	

We can bound run-times using

- *alternating renewal process theory* for Algorithm 1
- *supermartingale* ideas for Algorithm 2 (heuristic for $M/M/c$ queues only)

Mean run-time:

λ	c	ρ	**lower bound** Algorithm 1	**upper bound** Algorithm 2
10	10	5	102	5
20	20	10	52429	10
30	30	15	3.58×10^7	15
40	40	20	2.75×10^{10}	20
50	50	25	2.25×10^{13}	25
30	30	5	7.88	1
30	30	10	6392	5
30	30	20	6.86×10^{12}	41
30	30	25	7.37×10^{21}	132
30	30	29.5	7.37×10^{51}	4854

Conclusions

Conclusions

- It is highly feasible to produce *perfect* simulations of stable $M/G/c$ queues using domCFTP
  - mean run-time is finite iff $\mathbb{E}\left[S^2\right] < \infty$
  - Algorithm 1 is inefficient when the queue is not super-stable
  - Algorithm 2 is more complex to implement, but more efficient
- More recent work (Blanchet, Dong & Pei, 2015) uses domCFTP to sample from equilibrium of $GI/GI/c$ queues: finite expected run-time requires $2 + \varepsilon$ moments