“Omnithermal” perfect simulation for $M/G/c$ queues

Stephen Connor
stephen.connor@york.ac.uk

University of York

MCQMC, Stanford
August 2016

Dominated CFTP in a nutshell

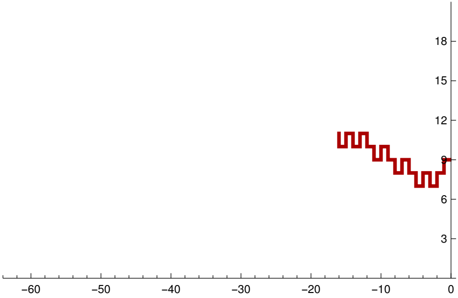
Suppose that we're interested in simulating from the equilbrium distribution of some ergodic Markov chain X.

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: $\tilde{X}_0 \sim \pi$;
- dominated CFTP (domCFTP) tries to determine the value of $\tilde{X}_0$ by looking into the past only a *finite* number of steps;
- do this by identifying a time in the past such that *all earlier starts from x lead to the same result at time zero*.

Basic ingredients:

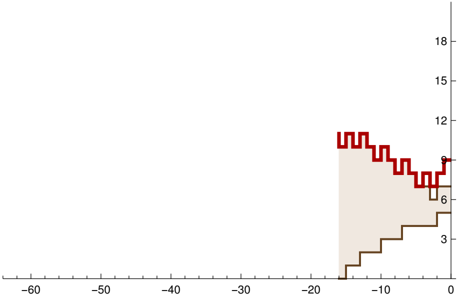
- *dominating process*
  - draw from equilibrium
  - simulate backwards in time



Basic ingredients:

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

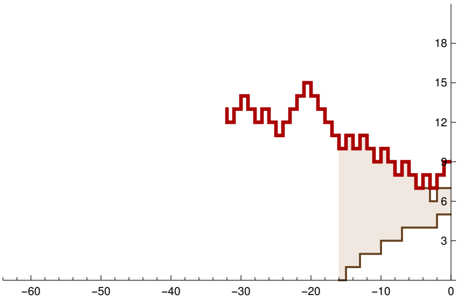
$$\text{Lower}_{\text{late}} \preccurlyeq \text{Lower}_{\text{early}} \preccurlyeq \ldots \preccurlyeq \text{Target} \preccurlyeq \ldots \preccurlyeq \text{Upper}_{\text{early}} \preccurlyeq \text{Upper}_{\text{late}}$$



Basic ingredients:

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

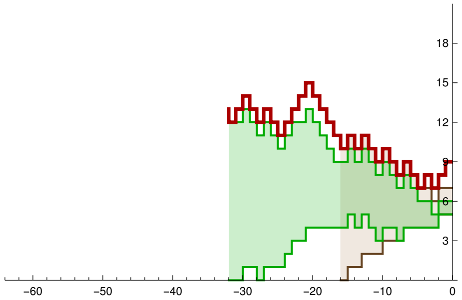
  $\text{Lower}_{\text{late}} \preccurlyeq \text{Lower}_{\text{early}} \preccurlyeq \ldots \preccurlyeq \text{Target} \preccurlyeq \ldots \preccurlyeq \text{Upper}_{\text{early}} \preccurlyeq \text{Upper}_{\text{late}}$



Basic ingredients:

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

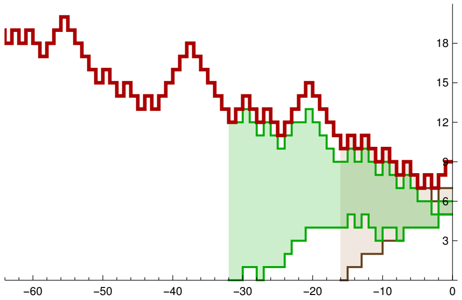
$$\text{Lower}_{\text{late}} \preccurlyeq \text{Lower}_{\text{early}} \preccurlyeq \ldots \preccurlyeq \text{Target} \preccurlyeq \ldots \preccurlyeq \text{Upper}_{\text{early}} \preccurlyeq \text{Upper}_{\text{late}}$$



Basic ingredients:

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

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

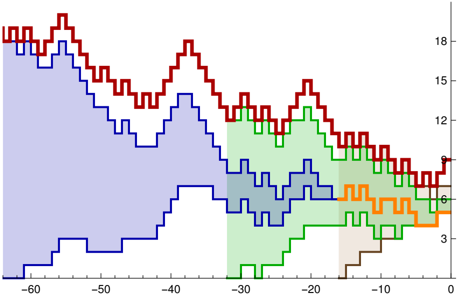


Basic ingredients:

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

  $\text{Lower}_{\text{late}} \preccurlyeq \text{Lower}_{\text{early}} \preccurlyeq \ldots \preccurlyeq \text{Target} \preccurlyeq \ldots \preccurlyeq \text{Upper}_{\text{early}} \preccurlyeq \text{Upper}_{\text{late}}$
- *coalescence*

  eventually a Lower and an Upper process must coalesce



$M/G/c$ queue

- 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.

domCFTP for $M/G/c$ queues

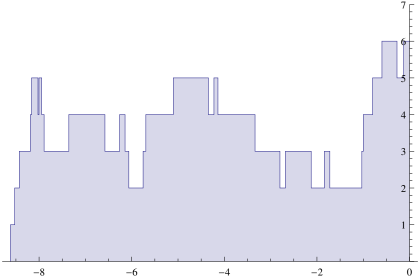
- Sigman (2011) showed how to do this for **super-stable** queues:
  - (stable) $M/G/1$ can be simulated in reverse-time by changing to *processor-sharing* discipline
  - so identify time $\tau < 0$ when $M/G/1$ is empty, and then use path of $M/G/1$ to dominate $M/G/c$ over $[\tau, 0]$

domCFTP for $M/G/c$ queues

- Sigman (2011) showed how to do this for **super-stable** queues:
  - (stable) $M/G/1$ can be simulated in reverse-time by changing to *processor-sharing* discipline
  - so identify time $\tau < 0$ when $M/G/1$ is empty, and then use path of $M/G/1$ to dominate $M/G/c$ over $[\tau, 0]$
- C. & Kendall (2015) extended this to **stable** $M/G/c$:
  - dominate $M/G/c\,[FCFS]$ by $M/G/c\,[RA]$, where **RA = Random Assignment**
  - important to assign service duration S_n to the n^{th} **initiation of service** in order to maintain sample-path domination
  - two possible algorithms presented ...

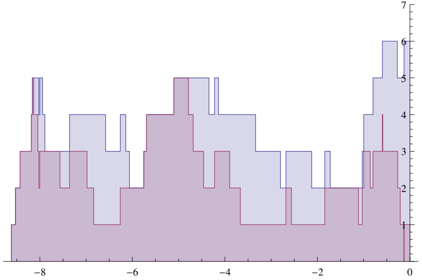
Algorithm 1: wait for dominating process to empty

1. Simulate $M/G/c\,[RA]$ backwards, in equilibrium, until it empties at time $\tau < 0$



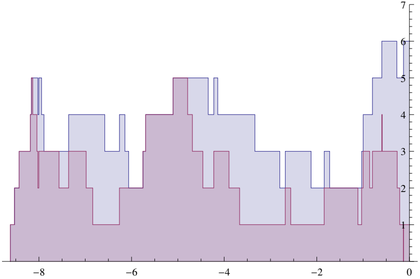
Algorithm 1: wait for dominating process to empty

1. Simulate $M/G/c\,[RA]$ backwards, in equilibrium, until it empties at time $\tau < 0$
2. Use this to evolve an $M/G/c\ [FCFS]$ process, over $[\tau, 0]$, started from empty
3. Return X_0



Algorithm 1: wait for dominating process to empty

1. Simulate $M/G/c\,[RA]$ backwards, in equilibrium, until it empties at time $\tau < 0$
2. Use this to evolve an $M/G/c\,[FCFS]$ process, over $[\tau, 0]$, started from empty
3. Return X_0



(Simple, but inefficient!)

Algorithm 2: backoff and sandwich

1. Set $T = -1$.
2. Generate a path of a stationary $M/G/c$ $[RA]$ process Y over $[T, 0]$

Algorithm 2: backoff and sandwich

1. Set $T = -1$.
2. Generate a path of a stationary $M/G/c\,[RA]$ process Y over $[T, 0]$
3. Construct upper and lower **sandwiching processes**, U and L:
   - these are workload vectors of $M/G/c\,[FCFS]$ queues
   - L starts from empty
   - U is instantiated using **residual workloads** from Y_T

Algorithm 2: backoff and sandwich

1. Set $T = -1$.
2. Generate a path of a stationary $M/G/c\,[RA]$ process Y over $[T, 0]$
3. Construct upper and lower **sandwiching processes**, U and L:
   - these are workload vectors of $M/G/c\,[FCFS]$ queues
   - L starts from empty
   - U is instantiated using **residual workloads** from Y_T
4. Check for **coalescence of workload vectors**; if $L(0) \neq U(0)$ then set $T \leftarrow 2T$ and return to step 2 (binary backoff). Else return $U(0)$.

Algorithm 2: backoff and sandwich

1. Set $T = -1$.
2. Generate a path of a stationary $M/G/c\,[RA]$ process Y over $[T, 0]$
3. Construct upper and lower **sandwiching processes**, U and L:
   - these are workload vectors of $M/G/c\,[FCFS]$ queues
   - L starts from empty
   - U is instantiated using **residual workloads** from Y_T
4. Check for **coalescence of workload vectors**; if $L(0) \neq U(0)$ then set $T \leftarrow 2T$ and return to step 2 (binary backoff). Else return $U(0)$.

(Much more efficient!)

Omnithermal domCFTP

Question

Is it possible to carry out domCFTP *simultaneously* for systems with varying numbers of servers?

In other words, can we use our algorithm for the c-server system to also obtain perfect draws for the $M/G/(c+j)$ system, for $j = 1, 2, \ldots$?

Omnithermal domCFTP

Question

Is it possible to carry out domCFTP *simultaneously* for systems with varying numbers of servers?

In other words, can we use our algorithm for the c-server system to also obtain perfect draws for the $M/G/(c+j)$ system, for $j = 1, 2, \ldots$?

- This is trivial if we use **Algorithm 1**: once we have identified $\tau < 0$ at which the $M/G/c\,[RA]$ is empty, we can (carefully) run the $M/G/(c+j)\,[FCFS]$ from empty at time τ.
- But what about using **Algorithm 2**?

Suppose that we have implemented **Algorithm 2** with c servers:

- call the upper and lower $M/G/c\,[FCFS]$ bounding processes used U^c and L^c respectively;
- recall that L^c is started from **empty**, and that U^c is instantiated using **residual workloads** from the dominating RA process at time $T < 0$.

Suppose that we have implemented **Algorithm 2** with c servers:

- call the upper and lower $M/G/c\,[FCFS]$ bounding processes used U^c and L^c respectively;
- recall that L^c is started from **empty**, and that U^c is instantiated using **residual workloads** from the dominating RA process at time $T < 0$.

Observation 1

If we instantiate U_T^{c+j} using the **same set of residual workloads**, and define $L_T^{c+j} = 0$, then these FCFS processes will sandwich our $M/G/(c+j)$ process of interest.

Suppose that we have implemented **Algorithm 2** with c servers:

- call the upper and lower $M/G/c\,[FCFS]$ bounding processes used U^c and L^c respectively;
- recall that L^c is started from **empty**, and that U^c is instantiated using **residual workloads** from the dominating RA process at time $T < 0$.

Observation 1

If we instantiate U_T^{c+j} using the **same set of residual workloads**, and define $L_T^{c+j} = 0$, then these FCFS processes will sandwich our $M/G/(c+j)$ process of interest.

Moreover, $U_t^{c+j}(i+j) \leq U_t^c(i)$, for $i = 1, \ldots, c$ and $t \in [T, 0]$.

Suppose that we have implemented **Algorithm 2** with c servers:

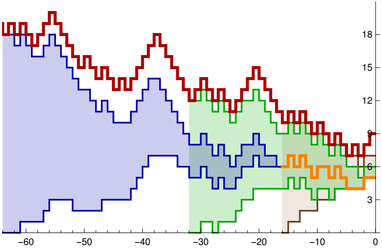
- call the upper and lower $M/G/c\,[FCFS]$ bounding processes used U^c and L^c respectively;
- recall that L^c is started from **empty**, and that U^c is instantiated using **residual workloads** from the dominating RA process at time $T < 0$.

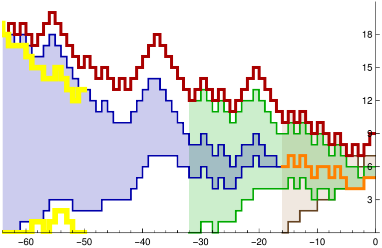
Observation 1

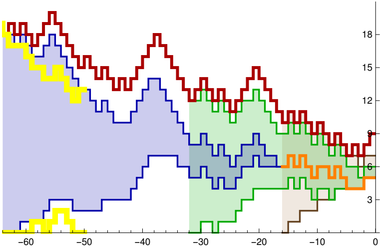
If we instantiate U_T^{c+j} using the **same set of residual workloads**, and define $L_T^{c+j} = 0$, then these FCFS processes will sandwich our $M/G/(c+j)$ process of interest.

Moreover, $U_t^{c+j}(i+j) \leq U_t^c(i)$, for $i = 1, \ldots, c$ and $t \in [T, 0]$.

But will U^{c+j} and L^{c+j} necessarily coalesce before time 0?


18
15
12
9
6
3
−60
−50
−40
−30
−20
−10
0


18
15
12
9
6
3
−60
−50
−40
−30
−20
−10
0



- Let $D^c = U^c - L^c$;
- termination of the algorithm means we have established a backoff time T such that D^c hits zero in the interval $[T, 0]$;
- write T^c for this coalescence time:

$$T^c = \inf\{t > T \, : \, D^c_t = 0\} < 0\,.$$

We seek conditions (which can be checked using only output from **Algorithm 2** for the *c*-server system) that ensure $T^{c+j} \leq T^c$.

It will be important to consider the set of coordinates in which U^c and L^c *agree*:

$$\mathcal{A}_t^c = \{k \leq c \, : \, D_t^c(k) = 0\} \, .$$

It will be important to consider the set of coordinates in which U^c and L^c *agree*:

$$\mathcal{A}_t^c = \{k \leq c \,:\, D_t^c(k) = 0\} \,.$$

Now write C_t^c for the remaining time (at time t) until coalescence of U_t^c and L_t^c **under the assumption of no more arrivals**:

$$\boxed{C_t^c = \max_{k \notin \mathcal{A}_t^c} U_t^c(k) = U_t^c(m_t^c)}$$

where $m_t^c = \max\{1 \leq k \leq c \,:\, k \notin \mathcal{A}_t^c\}$.

It will be important to consider the set of coordinates in which U^c and L^c *agree*:

$$\mathcal{A}_t^c = \{k \leq c \,:\, D_t^c(k) = 0\} \,.$$

Now write C_t^c for the remaining time (at time t) until coalescence of U_t^c and L_t^c **under the assumption of no more arrivals**:

$$C_t^c = \max_{k \notin \mathcal{A}_t^c} U_t^c(k) = U_t^c(m_t^c)$$

where $m_t^c = \max\{1 \leq k \leq c \,:\, k \notin \mathcal{A}_t^c\}$.

Observation 2

Since **(i)** $C_T^{c+j} \leq C_T^c$ **(ii)** $C_{T^c}^c = 0$ **(iii)** $C_t^{c+j} = 0$ iff $D_t^{c+j} = 0$, we will be assured of coalescence if we can show that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.

Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;

Evolution of C_t^c

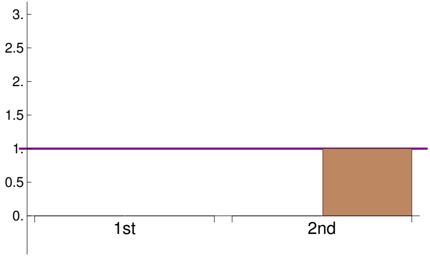
Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.

Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

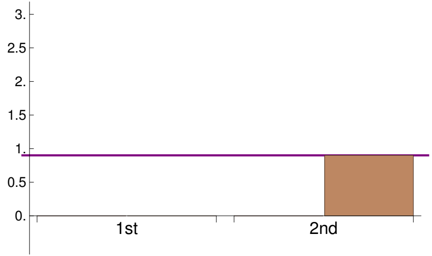
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

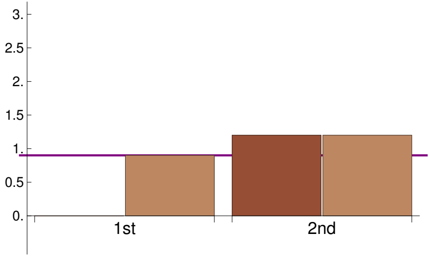
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

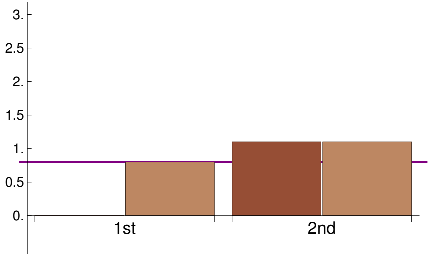
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

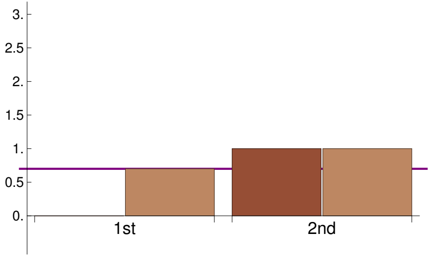
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

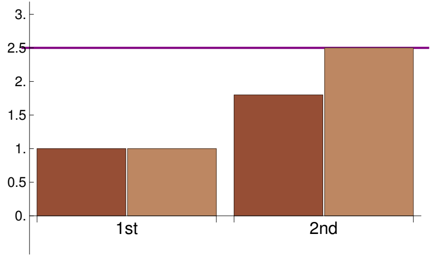
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

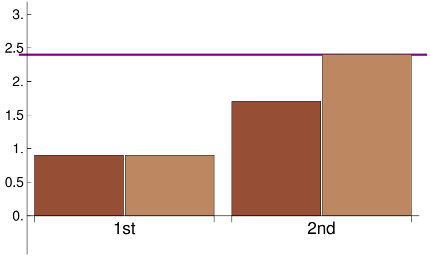
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

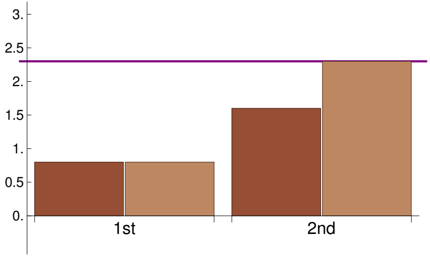
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

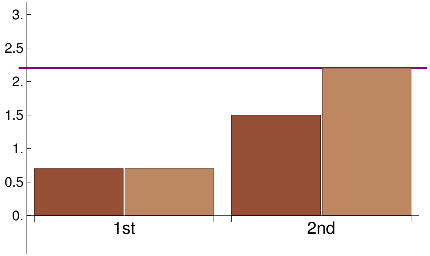
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

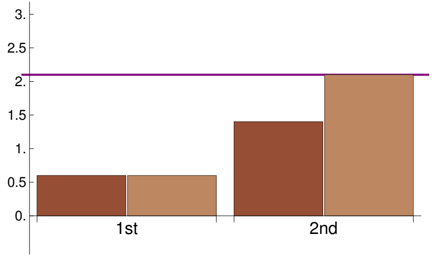
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

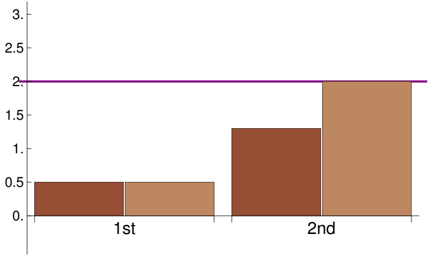
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

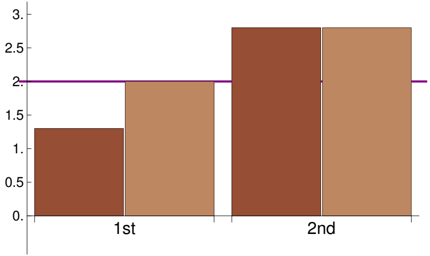
- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



Evolution of C_t^c

Suppose that we see an arrival (in both U^c and L^c) at time t, with associated workload S.

- if $1 \in \mathcal{A}_{t-}^c$ then arrival does not affect the time to coalescence;
- if $1 \notin \mathcal{A}_{t-}^c$ then we will see an increase in the time to coalescence iff the new service is placed at some coordinate $k \geq m_{t-}^c$ in U^c.



It follows that

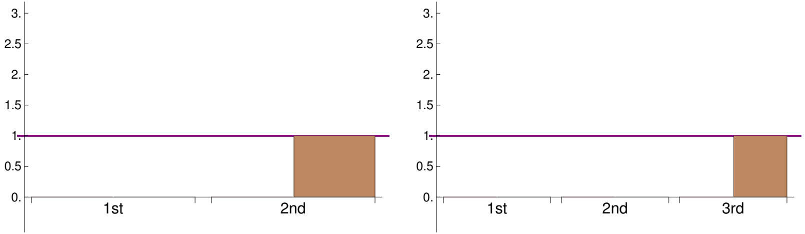
$$C_t^c = \max\left\{ C_{t-}^c\,,\ (U_{t-}^c(1) + S)\mathbf{1}_{[1 \notin \mathcal{A}_{t-}^c]} \right\}$$

It follows that

$$C_t^c = \max\left\{ C_{t-}^c\,,\; (U_{t-}^c(1) + S)\mathbf{1}_{[1 \notin \mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.

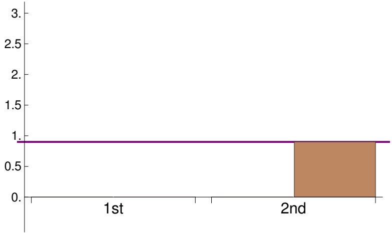


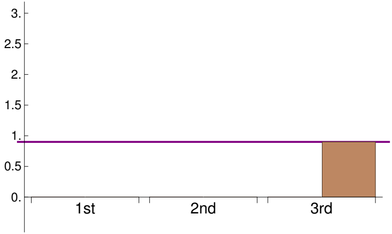
It follows that

$$C_t^c = \max\left\{ C_{t-}^c\,,\; (U_{t-}^c(1) + S)\mathbf{1}_{[1\notin\mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \le C_t^c$ for all $t \in [T, T^c]$.



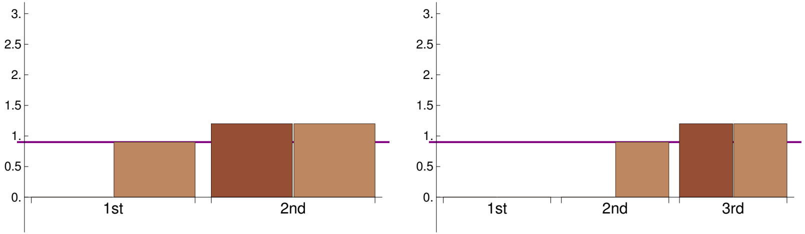


It follows that

$$C_t^c = \max\left\{ C_{t-}^c \, , \; (U_{t-}^c(1) + S)\mathbf{1}_{[1 \notin \mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.

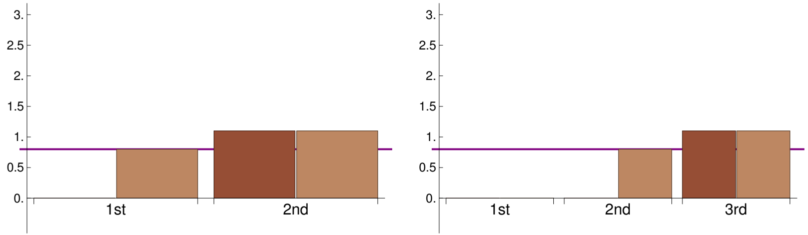


It follows that

$$C_t^c = \max\left\{ C_{t-}^c\,,\ (U_{t-}^c(1)+S)\mathbf{1}_{[1\notin\mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.

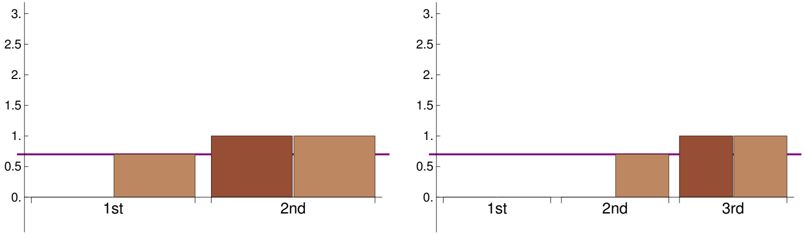


It follows that

$$C_t^c = \max\left\{ C_{t-}^c \,,\ (U_{t-}^c(1) + S)\mathbf{1}_{[1 \notin \mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.

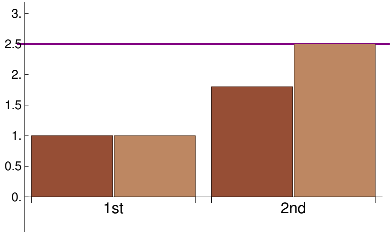


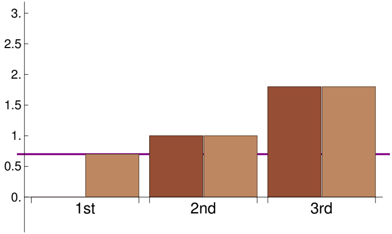
It follows that

$$C_t^c = \max\left\{ C_{t-}^c \,,\ (U_{t-}^c(1) + S)\mathbf{1}_{[1 \notin \mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.



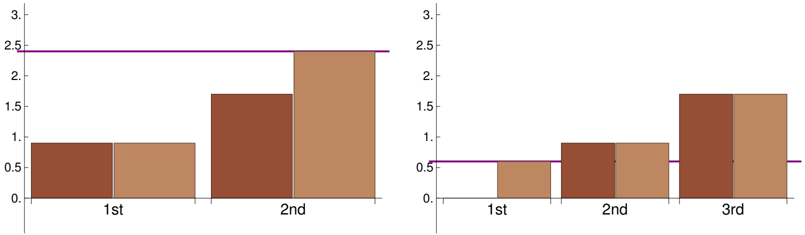


It follows that

$$C_t^c = \max\left\{ C_{t-}^c \,,\; (U_{t-}^c(1) + S)\mathbf{1}_{[1 \notin \mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.

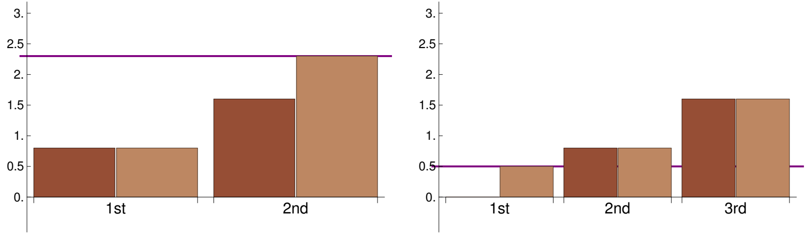


It follows that

$$C_t^c = \max\left\{ C_{t-}^c \,,\; (U_{t-}^c(1) + S)\mathbf{1}_{[1 \notin \mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.

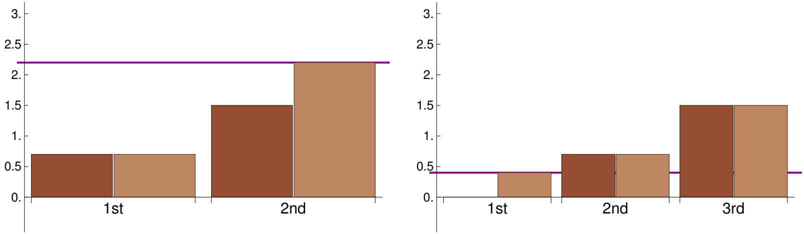


It follows that

$$C_t^c = \max\left\{ C_{t-}^c\,,\ (U_{t-}^c(1) + S)\mathbf{1}_{[1 \notin \mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.

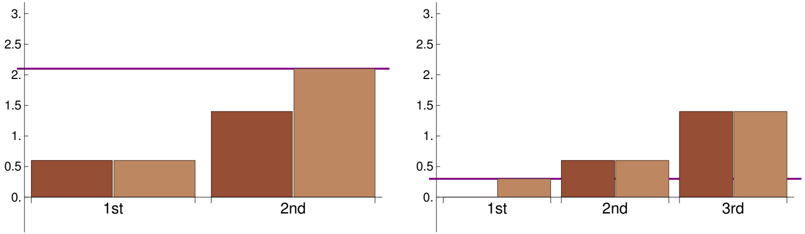


It follows that

$$C_t^c = \max\left\{ C_{t-}^c\,,\ (U_{t-}^c(1) + S)\mathbf{1}_{[1\notin\mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \le C_t^c$ for all $t \in [T, T^c]$.

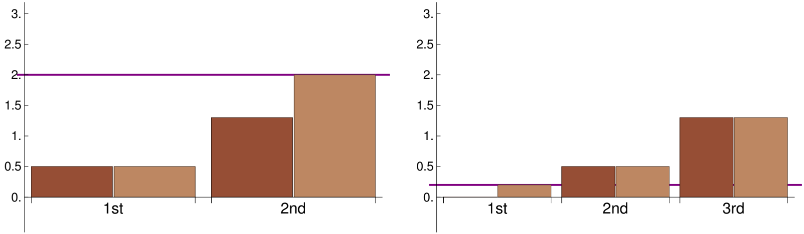


It follows that

$$C^c_t = \max\left\{ C^c_{t-}\,,\ (U^c_{t-}(1)+S)\mathbf{1}_{[1\notin\mathcal{A}^c_{t-}]} \right\}$$

Observation 3

It is **not** true in general that $C^{c+j}_t \leq C^c_t$ for all $t \in [T, T^c]$.

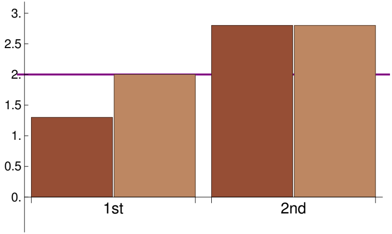


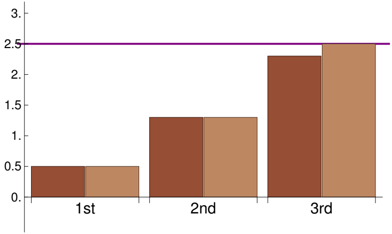
It follows that

$$C_t^c = \max\left\{ C_{t-}^c\,,\; (U_{t-}^c(1) + S)\mathbf{1}_{[1 \notin \mathcal{A}_{t-}^c]} \right\}$$

Observation 3

It is **not** true in general that $C_t^{c+j} \leq C_t^c$ for all $t \in [T, T^c]$.





However...

Theorem

If **no** *customer arriving during the interval* $[T, T^c]$ *finds* $1 \in \mathcal{A}^c$ *with* $U^c(1) > 0$, *then* $C_t^{c+j} \leq C_t^c$ *for all* $j \geq 0$ *and for all* $t \in [T, T^c]$. *In particular,* $T^{c+j} \leq T^c < 0$.

However...

Theorem

If **no** *customer arriving during the interval* $[T, T^c]$ *finds* $1 \in \mathcal{A}^c$ *with* $U^c(1) > 0$*, then* $C_t^{c+j} \leq C_t^c$ *for all* $j \geq 0$ *and for all* $t \in [T, T^c]$*. In particular,* $T^{c+j} \leq T^c < 0$*.*

This gives us a method for performing omnithermal domCFTP:

1. for a given run of **Algorithm 2** with c servers, check to see whether the condition of the Theorem holds. If not, backoff further ($T \leftarrow 2T$) and keep doing this until the condition is satisfied.
2. then run $M/G/(c+j)$ (for any choice of $j \geq 0$) over $[T, 0]$ and return state at time 0.

What does this cost?

How expensive is this in practice? Not very!

- Simulations seem to indicate that the condition is satisfied $> 95\%$ of the time
- In addition, runs in which the condition fails typically don't require significant extension

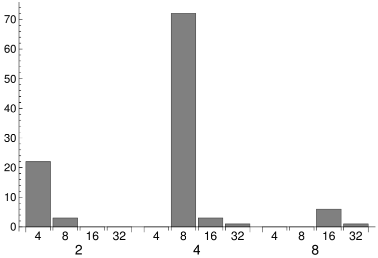
What does this cost?

How expensive is this in practice? Not very!

- Simulations seem to indicate that the condition is satisfied $> 95\%$ of the time
- In addition, runs in which the condition fails typically don't require significant extension

e.g. 5,000 runs of $M/M/c$ with $\lambda = 10$, $\mu = 2$ and $c = 10$:

- 108 (2.1%) runs needed extending
- none more than 3 times



Extensions

This idea can be applied in other settings.

1. Consider keeping c fixed, but increasing the rate at which servers work (this corresponds to decreasing the arrival rate). Same analysis as above holds.

2. Moreover, there's no need to restrict attention to Poisson arrivals! Blanchet, Pei & Sigman (2015) show how to implement domCFTP for $GI/GI/c$ queues, again using a random assignment dominating process.

Conclusions

- It is highly feasible to produce **perfect** simulations of stable $GI/GI/c$ queues using domCFTP
- Furthermore, with minimal additional effort this can be accomplished in an **omnithermal** way, allowing us to simultaneously sample from the equilibrium distribution when
  - using $c + j$ servers, $j = 1, 2, \dots$
  - increasing the service rate
  - or both.