Micro and Macro Views of Discrete-State Markov Models and their Application to Efficient Simulation with Phase-type Distributions

Philipp Reinecke, Miklós Telek, and Katinka Wolter

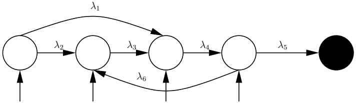
HP Labs, Bristol, UK and Freie Universität Berlin

BME Budapest

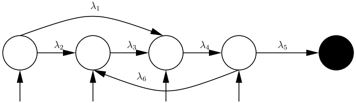
Newcastle University, UK and Freie Universität Berlin

August 25, 2012

Phase-Type (PH) Distributions

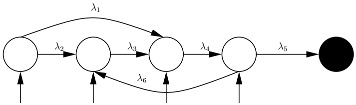


Phase-Type (PH) Distributions



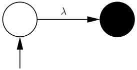
- A PH distribution is the distribution of the time to absorption in a Markov chain with one absorbing state

Phase-Type (PH) Distributions



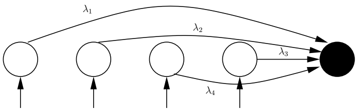
- A PH distribution is the distribution of the time to absorption in a Markov chain with one absorbing state
- Examples:

Phase-Type (PH) Distributions



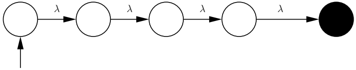
- A PH distribution is the distribution of the time to absorption in a Markov chain with one absorbing state
- Examples:
  - Exponential distribution

Phase-Type (PH) Distributions



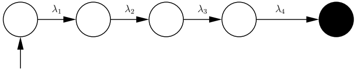
- A PH distribution is the distribution of the time to absorption in a Markov chain with one absorbing state
- Examples:
  - Exponential distribution
  - Hyperexponential distribution

Phase-Type (PH) Distributions



- A PH distribution is the distribution of the time to absorption in a Markov chain with one absorbing state
- Examples:
  - Exponential distribution
  - Hyperexponential distribution
  - Erlang distribution

Phase-Type (PH) Distributions



- A PH distribution is the distribution of the time to absorption in a Markov chain with one absorbing state
- Examples:
  - Exponential distribution
  - Hyperexponential distribution
  - Erlang distribution
  - Hypoexponential distribution

PH Distributions: Notation

- Size: $n \geq 1$
- Initial vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$
- Subgenerator matrix

$$\mathbf{Q} = \begin{pmatrix} -\lambda_{11} & \lambda_{12} & \ldots & \lambda_{1n} \\ \lambda_{21} & \ddots & & \vdots \\ \vdots & & & \\ \lambda_{n1} & \ldots & & -\lambda_{nn} \end{pmatrix}$$

- Markovian representation:

$$\begin{aligned} \boldsymbol{\alpha} &\geq \mathbf{0} \\ \boldsymbol{\alpha}\mathbb{1} &= 1 \\ \lambda_{ii} &> 0,\ i = 1, \ldots, n \\ \lambda_{ij} &\geq 0,\ i \neq j \end{aligned}$$

PH Distributions: Properties

- Support: $t \in [0, \infty)$
- Density function:

$$f(t) = \boldsymbol{\alpha} e^{\mathbf{Q}t}(-\mathbf{Q}\mathbb{1})$$

- The density is strictly positive: $f(t) > 0$ for $t > 0$
- Cumulative density function:

$$F(t) = 1 - \boldsymbol{\alpha} e^{\mathbf{Q}t}\mathbb{1}$$

- kth moment:

$$E[X^k] = k!\boldsymbol{\alpha}(-\mathbf{Q})^{-k}\mathbb{1}$$

- Bound on the squared coefficient of variation (SCV) [1]:

$$cv^2 \geq \frac{1}{n}$$

Equality holds for the Erlang distribution.

Similarity Transformations

Similarity Transformations

- The $(\boldsymbol{\alpha}, \mathbf{Q})$ representation is not unique

Similarity Transformations

- The $(\boldsymbol{\alpha}, \mathbf{Q})$ representation is not unique
- Let $(\boldsymbol{\alpha}, \mathbf{Q})$ be a PH distribution of size n and let $\mathbf{S} \in \mathbb{R}^{n \times n}$ be non-singular and $\mathbf{S}\mathbb{1} = \mathbb{1}$.

Similarity Transformations

- The $(\boldsymbol{\alpha}, \mathbf{Q})$ representation is not unique
- Let $(\boldsymbol{\alpha}, \mathbf{Q})$ be a PH distribution of size n and let $\mathbf{S} \in \mathbb{R}^{n \times n}$ be non-singular and $\mathbf{S}\mathbb{1} = \mathbb{1}$.
- $(\boldsymbol{\alpha}\mathbf{S}, \mathbf{S}^{-1}\mathbf{Q}\mathbf{S})$ represents the same distribution:

Similarity Transformations

- The $(\boldsymbol{\alpha}, \mathbf{Q})$ representation is not unique
- Let $(\boldsymbol{\alpha}, \mathbf{Q})$ be a PH distribution of size n and let $\mathbf{S} \in \mathbb{R}^{n \times n}$ be non-singular and $\mathbf{S}\mathbb{1} = \mathbb{1}$.
- $(\boldsymbol{\alpha}\mathbf{S}, \mathbf{S}^{-1}\mathbf{Q}\mathbf{S})$ represents the same distribution:

$$F(t) = 1 - \boldsymbol{\alpha}\mathbf{S}\mathrm{e}^{\mathbf{S}^{-1}\mathbf{Q}t\mathbf{S}}\mathbb{1}$$

Similarity Transformations

- The $(\boldsymbol{\alpha}, \mathbf{Q})$ representation is not unique
- Let $(\boldsymbol{\alpha}, \mathbf{Q})$ be a PH distribution of size n and let $\mathbf{S} \in \mathbb{R}^{n \times n}$ be non-singular and $\mathbf{S}\mathbb{1} = \mathbb{1}$.
- $(\boldsymbol{\alpha}\mathbf{S}, \mathbf{S}^{-1}\mathbf{Q}\mathbf{S})$ represents the same distribution:

$$
\begin{aligned}
F(t) &= 1 - \boldsymbol{\alpha}\mathbf{S}\mathrm{e}^{\mathbf{S}^{-1}\mathbf{Q}t\mathbf{S}}\mathbb{1} \\
&= 1 - \boldsymbol{\alpha}\mathbf{S}\mathbf{S}^{-1}\mathrm{e}^{\mathbf{Q}t}\mathbf{S}\mathbb{1}
\end{aligned}
$$

Similarity Transformations

- The $(\boldsymbol{\alpha}, \mathbf{Q})$ representation is not unique
- Let $(\boldsymbol{\alpha}, \mathbf{Q})$ be a PH distribution of size n and let $\mathbf{S} \in \mathbb{R}^{n \times n}$ be non-singular and $\mathbf{S}\mathbb{1} = \mathbb{1}$.
- $(\boldsymbol{\alpha}\mathbf{S}, \mathbf{S}^{-1}\mathbf{Q}\mathbf{S})$ represents the same distribution:

$$\begin{aligned} F(t) &= 1 - \boldsymbol{\alpha}\mathbf{S}\mathrm{e}^{\mathbf{S}^{-1}\mathbf{Q}t\mathbf{S}}\mathbb{1} \\ &= 1 - \boldsymbol{\alpha}\mathbf{S}\mathbf{S}^{-1}\mathrm{e}^{\mathbf{Q}t}\mathbf{S}\mathbb{1} \\ &= 1 - \boldsymbol{\alpha}\mathrm{e}^{\mathbf{Q}t} \end{aligned}$$

- Can be used to compute a new initialisation vector for a new representation

Similarity Transformations

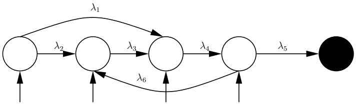
- The $(\boldsymbol{\alpha}, \mathbf{Q})$ representation is not unique
- Let $(\boldsymbol{\alpha}, \mathbf{Q})$ be a PH distribution of size n and let $\mathbf{S} \in \mathbb{R}^{n \times n}$ be non-singular and $\mathbf{S}\mathbb{1} = \mathbb{1}$.
- $(\boldsymbol{\alpha}\mathbf{S}, \mathbf{S}^{-1}\mathbf{Q}\mathbf{S})$ represents the same distribution:

$$\begin{aligned} F(t) &= 1 - \boldsymbol{\alpha}\mathbf{S}\mathrm{e}^{\mathbf{S}^{-1}\mathbf{Q}t\mathbf{S}}\mathbb{1} \\ &= 1 - \boldsymbol{\alpha}\mathbf{S}\mathbf{S}^{-1}\mathrm{e}^{\mathbf{Q}t}\mathbf{S}\mathbb{1} \\ &= 1 - \boldsymbol{\alpha}\mathrm{e}^{\mathbf{Q}t} \end{aligned}$$

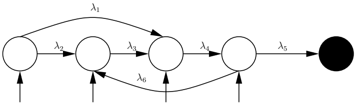
- Can be used to compute a new initialisation vector for a new representation
- Solve:

$$\begin{aligned} \mathbf{Q}' &= \mathbf{S}^{-1}\mathbf{Q}\mathbf{S} \\ \mathbf{S}\mathbb{1} &= \mathbb{1} \end{aligned}$$

General PH distributions

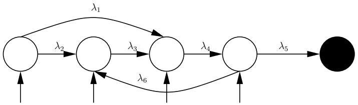


General PH distributions



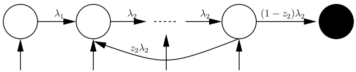
- General PH distributions may have cycles

General PH distributions



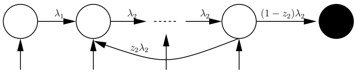
- General PH distributions may have cycles
- Every general PH distribution has a monocyclic representation [11]

General PH distributions



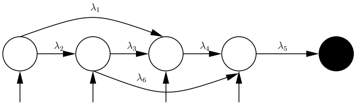
- General PH distributions may have cycles
- Every general PH distribution has a monocyclic representation [11]
- Monocyclic representation: Feedback-Erlang blocks on the diagonal, ordered by dominant eigenvalues

General PH distributions

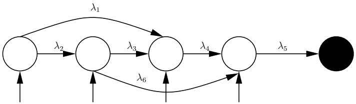


- General PH distributions may have cycles
- Every general PH distribution has a monocyclic representation [11]
- Monocyclic representation: Feedback-Erlang blocks on the diagonal, ordered by dominant eigenvalues
- Representation: Feedback blocks $\boldsymbol{\Upsilon} = ((b_1, z_1, \lambda_1), \ldots, (b_m, z_m, \lambda_m))$, initial vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$

Acyclic Phase-type distributions

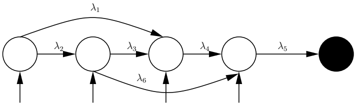


Acyclic Phase-type distributions



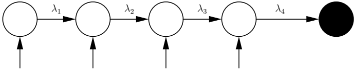
- Acyclic PH distributions (ACPH) have a representation without cycles

Acyclic Phase-type distributions



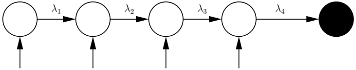
- Acyclic PH distributions (ACPH) have a representation without cycles
- CF-1: Every acyclic PH distribution has a bi-diagonal representation of the same size [6]

Acyclic Phase-type distributions



- Acyclic PH distributions (ACPH) have a representation without cycles
- CF-1: Every acyclic PH distribution has a bi-diagonal representation of the same size [6]
- Phase-type in CF-1 form: n rates $\lambda_1 \leq \cdots \leq \lambda_n$, n initial probabilities $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$.

Acyclic Phase-type distributions



- Acyclic PH distributions (ACPH) have a representation without cycles
- CF-1: Every acyclic PH distribution has a bi-diagonal representation of the same size [6]
- Phase-type in CF-1 form: n rates $\lambda_1 \leq \cdots \leq \lambda_n$, n initial probabilities $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$.
- Representation: Rate vector $\boldsymbol{\Lambda} = (\lambda_1, \ldots, \lambda_n)$, initial vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$

PH-Distributions in System Evaluation

PH-Distributions in System Evaluation

- Use PH distributions to model delays, response-times, failure-times, etc. in test-beds, simulations, and abstract models

PH-Distributions in System Evaluation

- Use PH distributions to model delays, response-times, failure-times, etc. in test-beds, simulations, and abstract models
- Approach:
  - Obtain samples from measurements or simulation
  - Fit PH distribution to samples
  - Draw random variates from PH distribution

PH-Distributions in System Evaluation

- Use PH distributions to model delays, response-times, failure-times, etc. in test-beds, simulations, and abstract models
- Approach:
  - Obtain samples from measurements or simulation
  - Fit PH distribution to samples
  - Draw random variates from PH distribution
- Advantages over other distributions:

PH-Distributions in System Evaluation

- Use PH distributions to model delays, response-times, failure-times, etc. in test-beds, simulations, and abstract models
- Approach:
  - Obtain samples from measurements or simulation
  - Fit PH distribution to samples
  - Draw random variates from PH distribution
- Advantages over other distributions:
  - Flexibility $\rightarrow$ Capture important system properties by fitting PH distributions to measurements

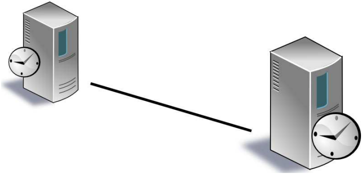
PH-Distributions in System Evaluation

- Use PH distributions to model delays, response-times, failure-times, etc. in test-beds, simulations, and abstract models
- Approach:
  - Obtain samples from measurements or simulation
  - Fit PH distribution to samples
  - Draw random variates from PH distribution
- Advantages over other distributions:
  - Flexibility $\rightarrow$ Capture important system properties by fitting PH distributions to measurements
  - Generic representations $\rightarrow$ Catch-all routines for random-variate generation

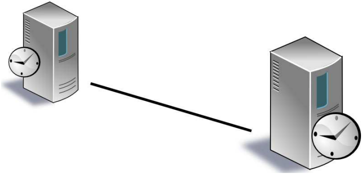
PH-Distributions in System Evaluation

- Use PH distributions to model delays, response-times, failure-times, etc. in test-beds, simulations, and abstract models
- Approach:
  - Obtain samples from measurements or simulation
  - Fit PH distribution to samples
  - Draw random variates from PH distribution
- Advantages over other distributions:
  - Flexibility $\rightarrow$ Capture important system properties by fitting PH distributions to measurements
  - Generic representations $\rightarrow$ Catch-all routines for random-variate generation
  - Markovian representations $\rightarrow$ Suitable for analytical approaches

Frequency-Synchronisation in Mobile Backhaul Networks

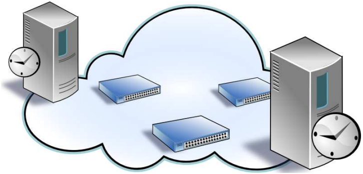


Frequency-Synchronisation in Mobile Backhaul Networks

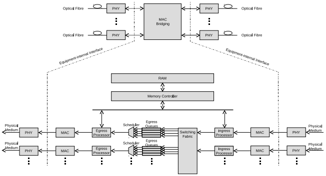


- Network service providers need guarantees in order to provide services, e.g. on frequency synchronisation of base-stations

Frequency-Synchronisation in Mobile Backhaul Networks

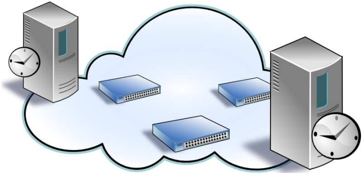


- Network service providers need guarantees in order to provide services, e.g. on frequency synchronisation of base-stations
- Bit-synchronous connection networks are being replaced by packet-switched networks



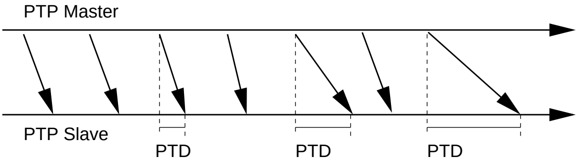
- Network service providers need guarantees in order to provide services, e.g. on frequency synchronisation of base-stations
- Bit-synchronous connection networks are being replaced by packet-switched networks

Frequency-Synchronisation in Mobile Backhaul Networks



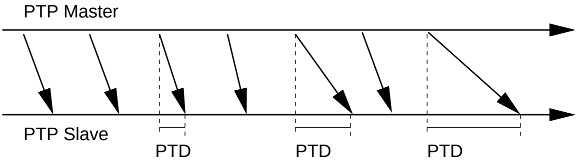
- Network service providers need guarantees in order to provide services, e.g. on frequency synchronisation of base-stations
- Bit-synchronous connection networks are being replaced by packet-switched networks

Frequency-Synchronisation in Mobile Backhaul Networks



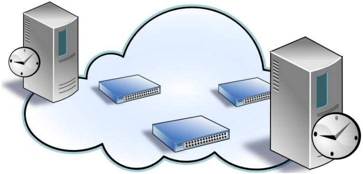
- Network service providers need guarantees in order to provide services, e.g. on frequency synchronisation of base-stations
- Bit-synchronous connection networks are being replaced by packet-switched networks
- Precision Time Protocol (PTP) provides frequency synchronisation

Frequency-Synchronisation in Mobile Backhaul Networks



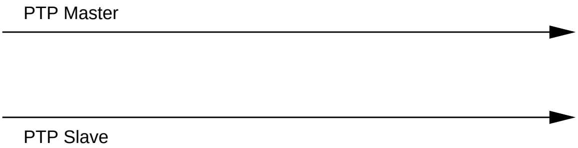
- Network service providers need guarantees in order to provide services, e.g. on frequency synchronisation of base-stations
- Bit-synchronous connection networks are being replaced by packet-switched networks
- Precision Time Protocol (PTP) provides frequency synchronisation
- PTP cannot tolerate packet-delay variation (PDV) above $216\mu s$

Frequency-Synchronisation in Mobile Backhaul Networks

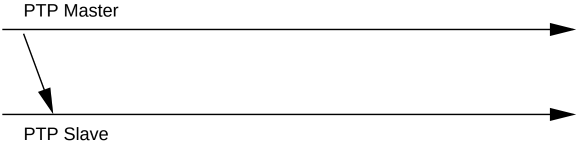


- Network service providers need guarantees in order to provide services, e.g. on frequency synchronisation of base-stations
- Bit-synchronous connection networks are being replaced by packet-switched networks
- Precision Time Protocol (PTP) provides frequency synchronisation
- PTP cannot tolerate packet-delay variation (PDV) above $216\mu s$
- Will PTP work?

Precision Time Protocol (PTP)

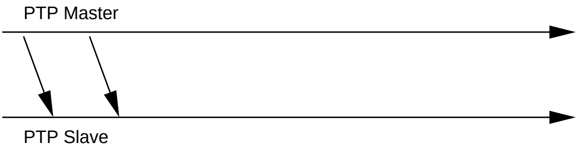


Precision Time Protocol (PTP)



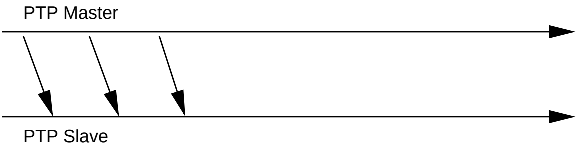
- PTP Master transmits Sync packets at clock steps

Precision Time Protocol (PTP)



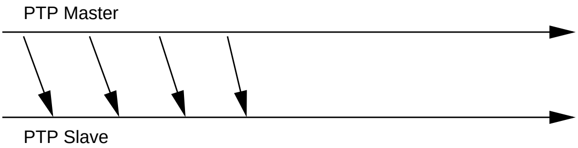
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)

Precision Time Protocol (PTP)



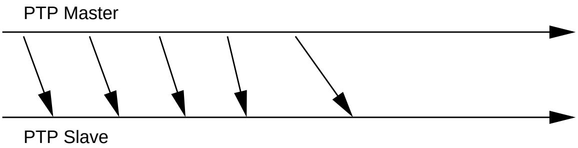
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)
- Constant delays do not matter

Precision Time Protocol (PTP)



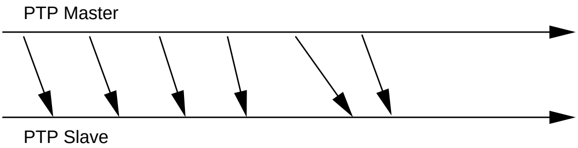
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)
- Constant delays do not matter

Precision Time Protocol (PTP)



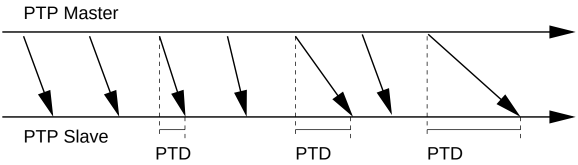
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)
- Constant delays do not matter
- ... but variation does

Precision Time Protocol (PTP)



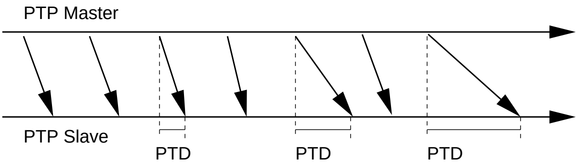
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)
- Constant delays do not matter
- . . . but variation does

Precision Time Protocol (PTP)



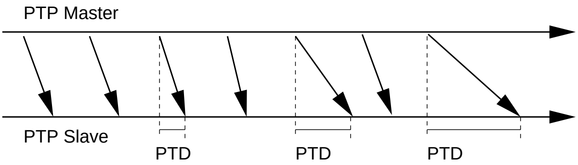
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)
- Constant delays do not matter
- . . . but variation does

Precision Time Protocol (PTP)



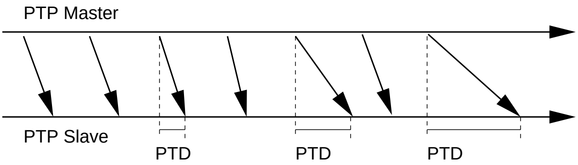
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)
- Constant delays do not matter
- . . . but variation does
- Metrics:

Precision Time Protocol (PTP)



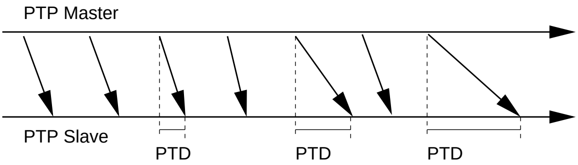
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)
- Constant delays do not matter
- . . . but variation does
- Metrics:
  - PDV: $PDV = PTD - PTD_{min}$

Precision Time Protocol (PTP)



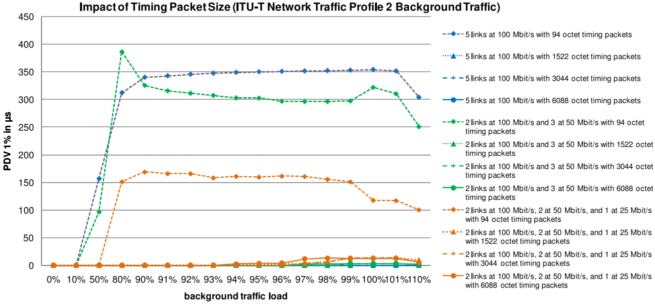
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)
- Constant delays do not matter
- . . . but variation does
- Metrics:
  - PDV: $PDV = PTD - PTD_{min}$
  - Peak-to-Peak PDV: $p2pPDV = PTD_{max} - PTD_{min}$

Precision Time Protocol (PTP)



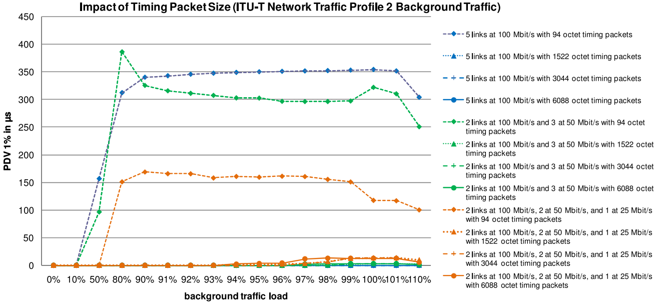
- PTP Master transmits Sync packets at clock steps
- PTP Slave derives clock frequency from the interarrival-times of the fastest packets (1% quantile)
- Constant delays do not matter
- ... but variation does
- Metrics:
  - PDV: $PDV = PTD - PTD_{min}$
  - Peak-to-Peak PDV: $p2pPDV = PTD_{max} - PTD_{min}$
  - 1% quantile of PDV

Insights in PTP analysis



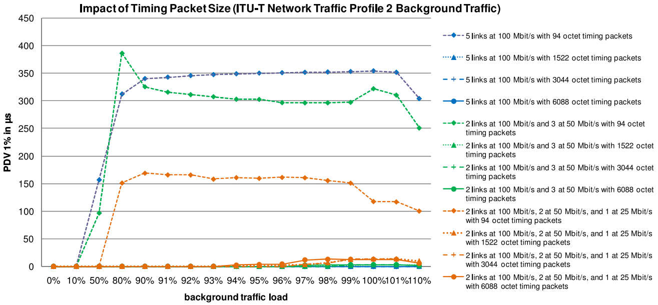
- Delay variation is highest for fast links and small PTP packets

Insights in PTP analysis



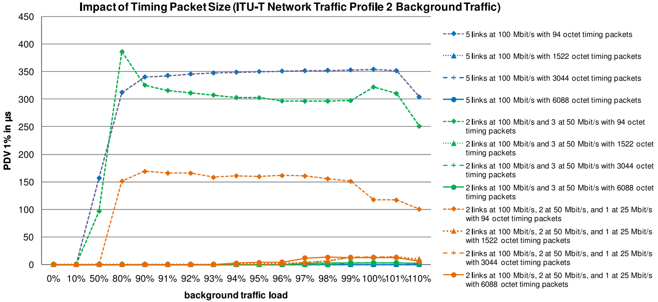
- Delay variation is highest for fast links and small PTP packets
- Delay variation is lower the slower the links,

Insights in PTP analysis



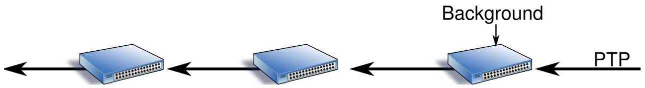
- Delay variation is highest for fast links and small PTP packets
- Delay variation is lower the slower the links, more important:

Insights in PTP analysis



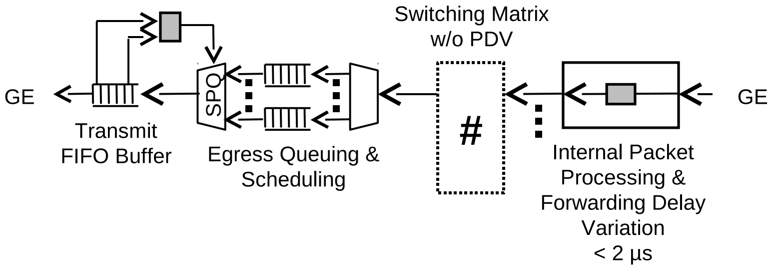
- Delay variation is highest for fast links and small PTP packets
- Delay variation is lower the slower the links, more important:
- PDV can be minimised by increasing PTP packet size

Simulation for Mobile Backhaul Network Evaluation



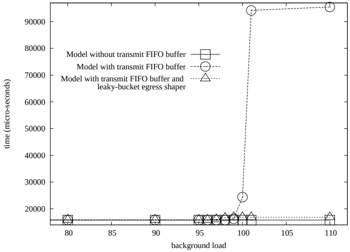
- Discrete-event simulations using ns-2

Simulation for Mobile Backhaul Network Evaluation



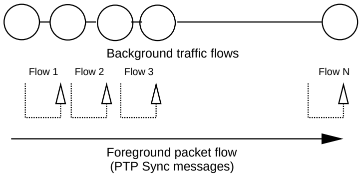
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment

Simulation for Mobile Backhaul Network Evaluation



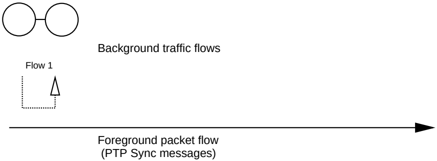
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects

Simulation for Mobile Backhaul Network Evaluation



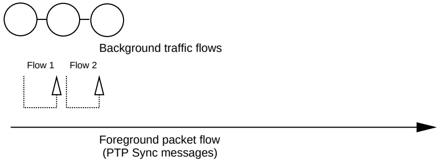
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic

Simulation for Mobile Backhaul Network Evaluation



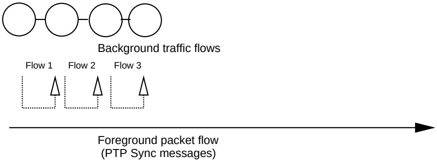
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic
  - 10 000 PTP packets $\Leftrightarrow$ 312.5 s simulated time (32 PTP packets per second)
  - One link $\Rightarrow$ 1883.25 s runtime

Simulation for Mobile Backhaul Network Evaluation



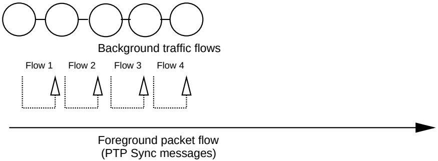
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic
  - 10 000 PTP packets $\Leftrightarrow$ 312.5 s simulated time (32 PTP packets per second)
  - One link $\Rightarrow$ 1883.25 s runtime
  - 2 links = 3815.63s,

Simulation for Mobile Backhaul Network Evaluation



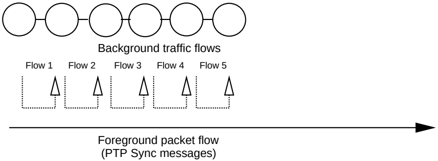
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic
  - 10 000 PTP packets $\Leftrightarrow$ 312.5 s simulated time (32 PTP packets per second)
  - One link $\Rightarrow$ 1883.25 s runtime
  - 2 links = 3815.63s, 3 links = 5822.63s,

Simulation for Mobile Backhaul Network Evaluation



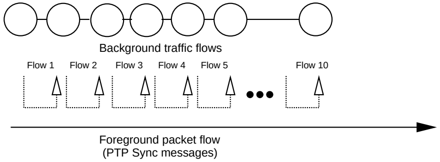
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic
  - 10 000 PTP packets $\Leftrightarrow$ 312.5 s simulated time (32 PTP packets per second)
  - One link $\Rightarrow$ 1883.25 s runtime
  - 2 links = 3815.63s, 3 links = 5822.63s, 4 links = 7516.72s,

Simulation for Mobile Backhaul Network Evaluation



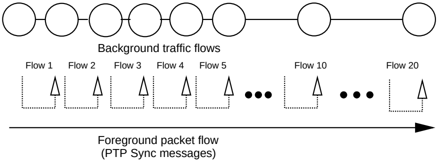
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic
  - 10 000 PTP packets $\Leftrightarrow$ 312.5 s simulated time (32 PTP packets per second)
  - One link $\Rightarrow$ 1883.25 s runtime
  - 2 links = 3815.63s, 3 links = 5822.63s, 4 links = 7516.72s, 5 links = 9718.97s,

Simulation for Mobile Backhaul Network Evaluation



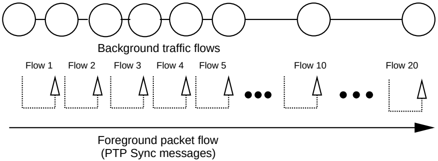
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic
  - 10 000 PTP packets $\Leftrightarrow$ 312.5 s simulated time (32 PTP packets per second)
  - One link $\Rightarrow$ 1883.25 s runtime
  - 2 links = 3815.63s, 3 links = 5822.63s, 4 links = 7516.72s, 5 links = 9718.97s, 10 links = 19,616.97s,

Simulation for Mobile Backhaul Network Evaluation



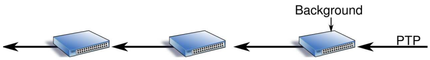
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic
  - 10 000 PTP packets ⇔ 312.5 s simulated time (32 PTP packets per second)
  - One link ⇒ 1883.25 s runtime
  - 2 links = 3815.63s, 3 links = 5822.63s, 4 links = 7516.72s, 5 links = 9718.97s, 10 links = 19,616.97s, 20 links 36,519.38s.

Simulation for Mobile Backhaul Network Evaluation



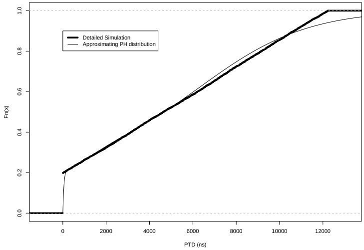
- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic
  - 10 000 PTP packets ⇔ 312.5 s simulated time (32 PTP packets per second)
  - One link ⇒ 1883.25 s runtime
  - 2 links = 3815.63s, 3 links = 5822.63s, 4 links = 7516.72s, 5 links = 9718.97s, 10 links = 19,616.97s, 20 links 36,519.38s.
  - Drawback: Simulation-times become prohibitively large

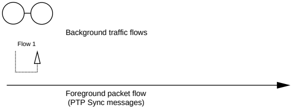
Simulation for Mobile Backhaul Network Evaluation



- Discrete-event simulations using ns-2
  - Highly-detailed models for typical network equipment
  - Simplified simulation skips important effects
  - Consider independent background traffic
  - 10 000 PTP packets ⇔ 312.5 s simulated time (32 PTP packets per second)
  - One link ⇒ 1883.25 s runtime
  - 2 links = 3815.63s, 3 links = 5822.63s, 4 links = 7516.72s, 5 links = 9718.97s, 10 links = 19,616.97s, 20 links 36,519.38s.
  - Drawback: Simulation-times become prohibitively large
- Solution: Approximate delay distributions of complex nodes

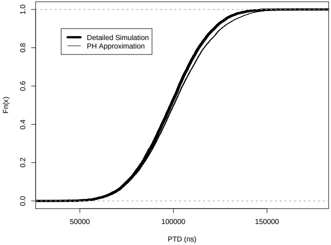
Highly-detailed Simulation

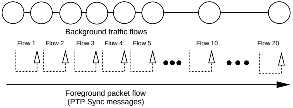




- Fit one link result using PhFit. Important feature: 1%quantile

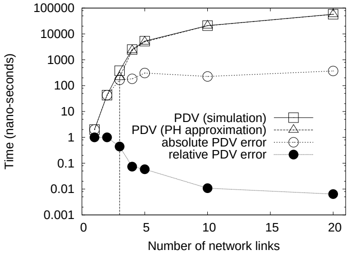
Highly-detailed Simulation





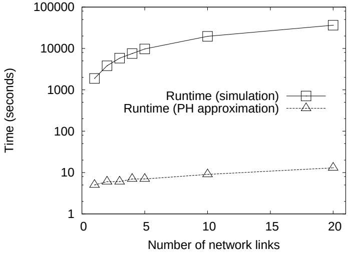
- Fit one link result using PhFit. Important feature: 1%quantile
- Use 20 PH RVs. Result still good for low quantiles

Highly-detailed Simulation



- Fit one link result using PhFit. Important feature: 1%quantile
- Use 20 PH RVs. Result still good for low quantiles
- Error reasonably small

Highly-detailed Simulation



- Fit one link result using PhFit. Important feature: 1%quantile
- Use 20 PH RVs. Result still good for low quantiles
- Error reasonably small
- Run time reduced by 2-3 orders of magnitude, analytical folding might achieve more.

The Libphprng Library

The Libphprng Library

- A library for generating random variates from PH distributions

The Libphprng Library

- A library for generating random variates from PH distributions
- Part of the Butools collection
  `http://webspn.hit.bme.hu/~butools`

The Libphprng Library

- A library for generating random variates from PH distributions
- Part of the Butools collection
  `http://webspn.hit.bme.hu/~butools`
- Advantages:

The Libphprng Library

- A library for generating random variates from PH distributions
- Part of the Butools collection
  `http://webspn.hit.bme.hu/~butools`
- Advantages:
  - easy to use

The Libphprng Library

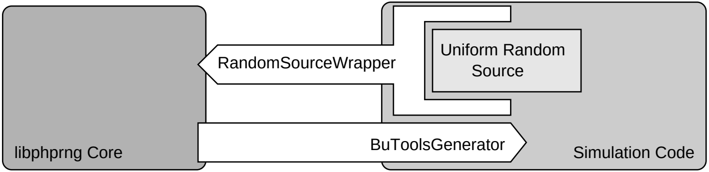
- A library for generating random variates from PH distributions
- Part of the Butools collection
  `http://webspn.hit.bme.hu/~butools`
- Advantages:
  - easy to use
  - portable between simulators

The Libphprng Library

- A library for generating random variates from PH distributions
- Part of the Butools collection
  `http://webspn.hit.bme.hu/~butools`
- Advantages:
  - easy to use
  - portable between simulators
  - fast

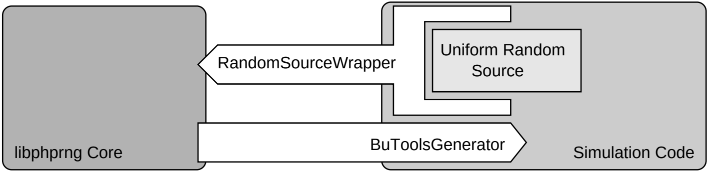
Libphprng Features

Libphprng Features



- Shared library with small wrapper code for the uniform random number stream

Libphprng Features



- Shared library with small wrapper code for the uniform random number stream
- Libphprng implements efficient algorithms and optimises the structure for random-variate generation

Libphprng Application

- Link simulator code with libphprng.so
- Changes to the code:

Libphprng Application

- Link simulator code with libphprng.so
- Changes to the code:
  1. Create `BuToolsGenerator` object for the distribution

Libphprng Application

- Link simulator code with libphprng.so
- Changes to the code:
  1. Create `BuToolsGenerator` object for the distribution
  2. Register uniform random number stream

Libphprng Application

- Link simulator code with libphprng.so
- Changes to the code:
  1. Create `BuToolsGenerator` object for the distribution
  2. Register uniform random number stream
  3. Draw random variates
- Wrappers exist for NS-2 and OMNeT++
- For other simulators: Write your own wrapper

Wrapper implementation

- Implement `UniformRandomSourceWrapper` interface
- Class must implement a method that returns a uniform random number in $(0, 1)$ drawn using the simulator's random number stream

Summary

- Phase-type distributions enable efficient simulation
- Several tools exist for PH fitting:
  - PhFit
  - G-FIT
  - Hyper-*
- The libphprng library allows integration of PH distributions into simulation

The Magic Behind the Scenes

- Fitting phase-type distributions to data sets
- Analytical evaluation using phase-type distributions
- Generating random variates from phase-type distributions

PH Fitting: General problem

PH Fitting: General problem

- Find a Markovian tuple $(\boldsymbol{\alpha}, \mathbf{Q})$ that describes the distribution of the data well

PH Fitting: General problem

- Find a Markovian tuple $(\boldsymbol{\alpha}, \mathbf{Q})$ that describes the distribution of the data well
- Different criteria may be applied

PH Fitting: General problem

- Find a Markovian tuple $(\boldsymbol{\alpha}, \mathbf{Q})$ that describes the distribution of the data well
- Different criteria may be applied
- Special structures of $(\boldsymbol{\alpha}, \mathbf{Q})$...

PH Fitting: General problem

- Find a Markovian tuple $(\boldsymbol{\alpha}, \mathbf{Q})$ that describes the distribution of the data well
- Different criteria may be applied
- Special structures of $(\boldsymbol{\alpha}, \mathbf{Q})$...
  - may reduce fitting to sub-classes

PH Fitting: General problem

- Find a Markovian tuple $(\boldsymbol{\alpha}, \mathbf{Q})$ that describes the distribution of the data well
- Different criteria may be applied
- Special structures of $(\boldsymbol{\alpha}, \mathbf{Q})$...
  - may reduce fitting to sub-classes
  - may improve fitting efficiency and fitting quality

PH Fitting: General problem

- Find a Markovian tuple $(\boldsymbol{\alpha}, \mathbf{Q})$ that describes the distribution of the data well
- Different criteria may be applied
- Special structures of $(\boldsymbol{\alpha}, \mathbf{Q})$...
  - may reduce fitting to sub-classes
  - may improve fitting efficiency and fitting quality
  - may enable more efficient evaluation

PH Fitting: General problem

- Find a Markovian tuple $(\boldsymbol{\alpha}, \mathbf{Q})$ that describes the distribution of the data well
- Different criteria may be applied
- Special structures of $(\boldsymbol{\alpha}, \mathbf{Q})\ldots$
  - may reduce fitting to sub-classes
  - may improve fitting efficiency and fitting quality
  - may enable more efficient evaluation
- Many approaches exist

Approaches

Approaches

- Moment-matching: Match moments of the PH to empirical moments

Approaches

- Moment-matching: Match moments of the PH to empirical moments
- Expectation-Maximisation (EM): Maximise (log-)likelihood

Approaches

- Moment-matching: Match moments of the PH to empirical moments
- Expectation-Maximisation (EM): Maximise (log-)likelihood
- Optimisation: Minimise a distance function

Approaches

- Moment-matching: Match moments of the PH to empirical moments
- Expectation-Maximisation (EM): Maximise (log-)likelihood
- Optimisation: Minimise a distance function
- Splitting the data set: break up the data set, then fit with simpler distributions

Moment-Matching

Moment-Matching

- Derive parameters from explicit expressions for the moments:

$$E\left[X^k\right] = k!\boldsymbol{\alpha}(-\mathbf{Q})^{-k}\mathbb{1}.$$

Moment-Matching

- Derive parameters from explicit expressions for the moments:

$$E\left[X^k\right] = k!\boldsymbol{\alpha}(-\mathbf{Q})^{-k}\mathbb{1}.$$

- Examples:

Moment-Matching

- Derive parameters from explicit expressions for the moments:

$$E\left[X^k\right] = k!\boldsymbol{\alpha}(-\mathbf{Q})^{-k}\mathbb{1}.$$

- Examples:
  - [19]: Match first three moments with an APH(2)

Moment-Matching

- Derive parameters from explicit expressions for the moments:

$$E\left[X^k\right] = k!\boldsymbol{\alpha}(-\mathbf{Q})^{-k}\mathbb{1}.$$

- Examples:
  - [19]: Match first three moments with an APH(2)
  - [7]: Match first five moments with PH(3)

Moment-Matching

- Derive parameters from explicit expressions for the moments:

$$E\left[X^k\right] = k!\boldsymbol{\alpha}(-\mathbf{Q})^{-k}\mathbb{1}.$$

- Examples:
  - [19]: Match first three moments with an APH(2)
  - [7]: Match first five moments with PH(3)
  - [5]: Uses moment-matching in MAP matching

Example: Moment-Matching for APH(2) [19]

Example: Moment-Matching for APH(2) [19]

- Use canonical form:

$$\boldsymbol{\alpha} = (\alpha, 1-\alpha)$$
$$\mathbf{Q} = \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & -\lambda_2 \end{pmatrix}$$

Example: Moment-Matching for APH(2) [19]

- Use canonical form:

$$\begin{aligned} \boldsymbol{\alpha} &= (\alpha, 1-\alpha) \\ \mathbf{Q} &= \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & -\lambda_2 \end{pmatrix} \end{aligned}$$

- Explicit expressions for the moments:

$$\begin{aligned} m_1 &= \frac{\lambda_1 + \alpha\lambda_2}{\lambda_1\lambda_2} \\ m_2 &= \frac{2(\lambda_1^2 + \alpha\lambda_1\lambda_2 + \alpha\lambda_2^2)}{\lambda_1^2\lambda_2^2} \\ m_3 &= \frac{6(\lambda_1^3 + \alpha\lambda_1^2 + \alpha\lambda_1\lambda_2^2 + \alpha\lambda_2^3)}{\lambda_1^3\lambda_2^3} \end{aligned}$$

Example: Moment-Matching for APH(2) [19]

- Use canonical form:

$$\begin{aligned} \boldsymbol{\alpha} &= (\alpha, 1-\alpha) \\ \mathbf{Q} &= \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & -\lambda_2 \end{pmatrix} \end{aligned}$$

- Explicit expressions for the moments:

$$\begin{aligned} m_1 &= \frac{\lambda_1 + \alpha\lambda_2}{\lambda_1\lambda_2} \\ m_2 &= \frac{2(\lambda_1^2 + \alpha\lambda_1\lambda_2 + \alpha\lambda_2^2)}{\lambda_1^2\lambda_2^2} \\ m_3 &= \frac{6(\lambda_1^3 + \alpha\lambda_1^2 + \alpha\lambda_1\lambda_2^2 + \alpha\lambda_2^3)}{\lambda_1^3\lambda_2^3} \end{aligned}$$

- Compute empirical moments of the data set

Example: Moment-Matching for APH(2) [19]

- Use canonical form:

$$\begin{aligned}\boldsymbol{\alpha} &= (\alpha, 1-\alpha)\\ \mathbf{Q} &= \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0 & -\lambda_2 \end{pmatrix}\end{aligned}$$

- Explicit expressions for the moments:

$$\begin{aligned}m_1 &= \frac{\lambda_1 + \alpha\lambda_2}{\lambda_1\lambda_2}\\ m_2 &= \frac{2(\lambda_1^2 + \alpha\lambda_1\lambda_2 + \alpha\lambda_2^2)}{\lambda_1^2\lambda_2^2}\\ m_3 &= \frac{6(\lambda_1^3 + \alpha\lambda_1^2 + \alpha\lambda_1\lambda_2^2 + \alpha\lambda_2^3)}{\lambda_1^3\lambda_2^3}\end{aligned}$$

- Compute empirical moments of the data set
- Set parameters using the explict expressions

Moment-Matching

Moment-Matching

- Advantages:

Moment-Matching

- Advantages:
  - Fast

Moment-Matching

- Advantages:
  - Fast
  - Exact match possible

Moment-Matching

- Advantages:
  - Fast
  - Exact match possible
- Disadvantages:

Moment-Matching

- Advantages:
  - Fast
  - Exact match possible
- Disadvantages:
  - Only matches moments; shape can differ significantly

Moment-Matching

- Advantages:
  - Fast
  - Exact match possible
- Disadvantages:
  - Only matches moments; shape can differ significantly
  - Exact match is only possible if the moments are within the bounds of the selected sub-class. E.g. PH(2) cannot match data sets with $cv^2 < \frac{1}{2}$ [1] (approximate matching may be used)

Expectation-Maximisation

Expectation-Maximisation

- Let $\boldsymbol{\theta}$ be the parameters of a phase-type distribution

Expectation-Maximisation

- Let $\boldsymbol{\theta}$ be the parameters of a phase-type distribution
- Maximise likelihood $\prod f_{\boldsymbol{\theta}}(t_i)$ or log-likelihood $\ln \sum f_{\boldsymbol{\theta}}(t_i)$

Expectation-Maximisation

- Let $\boldsymbol{\theta}$ be the parameters of a phase-type distribution
- Maximise likelihood $\prod f_{\boldsymbol{\theta}}(t_i)$ or log-likelihood $\ln \sum f_{\boldsymbol{\theta}}(t_i)$
- Steps:

Expectation-Maximisation

- Let $\boldsymbol{\theta}$ be the parameters of a phase-type distribution
- Maximise likelihood $\prod f_{\boldsymbol{\theta}}(t_i)$ or log-likelihood $\ln \sum f_{\boldsymbol{\theta}}(t_i)$
- Steps:
  - Estimate unknown parameters

Expectation-Maximisation

- Let $\boldsymbol{\theta}$ be the parameters of a phase-type distribution
- Maximise likelihood $\prod f_{\boldsymbol{\theta}}(t_i)$ or log-likelihood $\ln \sum f_{\boldsymbol{\theta}}(t_i)$
- Steps:
  - Estimate unknown parameters
  - Compute new parameter vector $\boldsymbol{\theta}$ to maximise likelihood

Expectation-Maximisation

- Let $\boldsymbol{\theta}$ be the parameters of a phase-type distribution
- Maximise likelihood $\prod f_{\boldsymbol{\theta}}(t_i)$ or log-likelihood $\ln \sum f_{\boldsymbol{\theta}}(t_i)$
- Steps:
  - Estimate unknown parameters
  - Compute new parameter vector $\boldsymbol{\theta}$ to maximise likelihood
- Examples:

Expectation-Maximisation

- Let $\boldsymbol{\theta}$ be the parameters of a phase-type distribution
- Maximise likelihood $\prod f_{\boldsymbol{\theta}}(t_i)$ or log-likelihood $\ln \sum f_{\boldsymbol{\theta}}(t_i)$
- Steps:
  - Estimate unknown parameters
  - Compute new parameter vector $\boldsymbol{\theta}$ to maximise likelihood
- Examples:
  - G-FIT [20]: Fit Hyper-Erlang distributions

Expectation-Maximisation

- Let $\boldsymbol{\theta}$ be the parameters of a phase-type distribution
- Maximise likelihood $\prod f_{\boldsymbol{\theta}}(t_i)$ or log-likelihood $\ln \sum f_{\boldsymbol{\theta}}(t_i)$
- Steps:
  - Estimate unknown parameters
  - Compute new parameter vector $\boldsymbol{\theta}$ to maximise likelihood
- Examples:
  - G-FIT [20]: Fit Hyper-Erlang distributions
  - EMPHT [2]: Fit general PH distributions

EM-Algorithm in G-FIT

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions
- Parameters of Hyper-Erlang distributions:

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions
- Parameters of Hyper-Erlang distributions:
  - Number of branches m

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions
- Parameters of Hyper-Erlang distributions:
  - Number of branches m
  - Branch lengths $b_1, \ldots, b_m$

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions
- Parameters of Hyper-Erlang distributions:
  - Number of branches m
  - Branch lengths $b_1, \ldots, b_m$
  - Branch probabilities $\beta_1, \ldots, \beta_m$

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions
- Parameters of Hyper-Erlang distributions:
  - Number of branches m
  - Branch lengths $b_1, \ldots, b_m$
  - Branch probabilities $\beta_1, \ldots, \beta_m$
  - Branch rates $\lambda_1, \ldots, \lambda_m$

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions
- Parameters of Hyper-Erlang distributions:
  - Number of branches m
  - Branch lengths $b_1, \ldots, b_m$
  - Branch probabilities $\beta_1, \ldots, \beta_m$
  - Branch rates $\lambda_1, \ldots, \lambda_m$
- Selection of m and $b_1, \ldots, b_m$:

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions
- Parameters of Hyper-Erlang distributions:
  - Number of branches m
  - Branch lengths $b_1, \ldots, b_m$
  - Branch probabilities $\beta_1, \ldots, \beta_m$
  - Branch rates $\lambda_1, \ldots, \lambda_m$
- Selection of m and $b_1, \ldots, b_m$:
  - Manual

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions
- Parameters of Hyper-Erlang distributions:
  - Number of branches m
  - Branch lengths $b_1, \ldots, b_m$
  - Branch probabilities $\beta_1, \ldots, \beta_m$
  - Branch rates $\lambda_1, \ldots, \lambda_m$
- Selection of m and $b_1, \ldots, b_m$:
  - Manual
  - Automatic (enumeration)

EM-Algorithm in G-FIT

- G-FIT [20] fits Hyper-Erlang distributions
- Parameters of Hyper-Erlang distributions:
  - Number of branches m
  - Branch lengths $b_1, \ldots, b_m$
  - Branch probabilities $\beta_1, \ldots, \beta_m$
  - Branch rates $\lambda_1, \ldots, \lambda_m$
- Selection of m and $b_1, \ldots, b_m$:
  - Manual
  - Automatic (enumeration)
- $\beta_1, \ldots, \beta_m$ and $\lambda_1, \ldots, \lambda_m$ fitted by EM algorithm

EM-Algorithm in G-FIT

EM-Algorithm in G-FIT

- Fix number of branches m and branch lengths $b_1, \ldots, b_m$.

EM-Algorithm in G-FIT

- Fix number of branches m and branch lengths $b_1, \ldots, b_m$.
- Choose initial parameters $\hat{\boldsymbol{\theta}} = (\hat{\beta}_1, \ldots, \hat{\beta}_m, \hat{\lambda}_1, \ldots, \hat{\lambda}_m)$

EM-Algorithm in G-FIT

- Fix number of branches m and branch lengths $b_1, \dots, b_m$.
- Choose initial parameters $\hat{\boldsymbol{\theta}} = (\hat{\beta}_1, \dots, \hat{\beta}_m, \hat{\lambda}_1, \dots, \hat{\lambda}_m)$
- (E-Step): Estimate probability of sample assignments to branches

$$q(i|x_k, \hat{\boldsymbol{\theta}}) := \frac{\hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}{\sum_{i=1}^{m} \hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}$$

EM-Algorithm in G-FIT

- Fix number of branches m and branch lengths $b_1, \ldots, b_m$.
- Choose initial parameters $\hat{\boldsymbol{\theta}} = (\hat{\beta}_1, \ldots, \hat{\beta}_m, \hat{\lambda}_1, \ldots, \hat{\lambda}_m)$
- (E-Step): Estimate probability of sample assignments to branches

$$q(i|x_k, \hat{\boldsymbol{\theta}}) := \frac{\hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}{\sum_{i=1}^{m} \hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}$$

- (M-Step): Compute new parameter vector $\boldsymbol{\theta}$ that maximises the log-likelihood:

EM-Algorithm in G-FIT

- Fix number of branches m and branch lengths $b_1, \ldots, b_m$.
- Choose initial parameters $\hat{\boldsymbol{\theta}} = (\hat{\beta}_1, \ldots, \hat{\beta}_m, \hat{\lambda}_1, \ldots, \hat{\lambda}_m)$
- (E-Step): Estimate probability of sample assignments to branches

$$q(i|x_k, \hat{\boldsymbol{\theta}}) := \frac{\hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}{\sum_{i=1}^{m} \hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}$$

- (M-Step): Compute new parameter vector $\boldsymbol{\theta}$ that maximises the log-likelihood:

$$\beta_i := \frac{1}{K} \sum_{k=1}^{K} q(i|x_k, \hat{\boldsymbol{\theta}}) \quad (1)$$

$$\lambda_i := b_i \frac{\sum_{k=1}^{K} q(i|x_k, \hat{\boldsymbol{\theta}})}{\sum_{k=1}^{K} (q(i|x_k, \hat{\boldsymbol{\theta}}) x_k)} \quad (2)$$

EM-Algorithm in G-FIT

- Fix number of branches m and branch lengths $b_1, \ldots, b_m$.
- Choose initial parameters $\hat{\boldsymbol{\theta}} = (\hat{\beta}_1, \ldots, \hat{\beta}_m, \hat{\lambda}_1, \ldots, \hat{\lambda}_m)$
- (E-Step): Estimate probability of sample assignments to branches
$$q(i|x_k, \hat{\boldsymbol{\theta}}) := \frac{\hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}{\sum_{i=1}^{m} \hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}$$
- (M-Step): Compute new parameter vector $\boldsymbol{\theta}$ that maximises the log-likelihood:
$$\beta_i := \frac{1}{K} \sum_{k=1}^{K} q(i|x_k, \hat{\boldsymbol{\theta}}) \quad (1)$$
$$\lambda_i := b_i \frac{\sum_{k=1}^{K} q(i|x_k, \hat{\boldsymbol{\theta}})}{\sum_{k=1}^{K} (q(i|x_k, \hat{\boldsymbol{\theta}}) x_k)} \quad (2)$$
- Replace old parameter vector: $\hat{\boldsymbol{\theta}} := \boldsymbol{\theta}$

EM-Algorithm in G-FIT

- Fix number of branches m and branch lengths $b_1, \ldots, b_m$.
- Choose initial parameters $\hat{\boldsymbol{\theta}} = (\hat{\beta}_1, \ldots, \hat{\beta}_m, \hat{\lambda}_1, \ldots, \hat{\lambda}_m)$
- (E-Step): Estimate probability of sample assignments to branches
$$q(i|x_k, \hat{\boldsymbol{\theta}}) := \frac{\hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}{\sum_{i=1}^{m} \hat{\beta}_i f_i(x_k|\hat{\lambda}_i)}$$
- (M-Step): Compute new parameter vector $\boldsymbol{\theta}$ that maximises the log-likelihood:
$$\beta_i := \frac{1}{K} \sum_{k=1}^{K} q(i|x_k, \hat{\boldsymbol{\theta}}) \quad (1)$$
$$\lambda_i := b_i \frac{\sum_{k=1}^{K} q(i|x_k, \hat{\boldsymbol{\theta}})}{\sum_{k=1}^{K} (q(i|x_k, \hat{\boldsymbol{\theta}}) x_k)} \quad (2)$$
- Replace old parameter vector: $\hat{\boldsymbol{\theta}} := \boldsymbol{\theta}$
- Repeat until convergence occurs

EM-Algorithm in G-FIT

EM-Algorithm in G-FIT

- Advantages:

EM-Algorithm in G-FIT

- Advantages:
  - Fast fitting, easy to automate

EM-Algorithm in G-FIT

- Advantages:
  - Fast fitting, easy to automate
  - Little configuration required for good results

EM-Algorithm in G-FIT

- Advantages:
  - Fast fitting, easy to automate
  - Little configuration required for good results
  - Well-suited for simulation

EM-Algorithm in G-FIT

- Advantages:
  - Fast fitting, easy to automate
  - Little configuration required for good results
  - Well-suited for simulation
- Disadvantages:

EM-Algorithm in G-FIT

- Advantages:
  - Fast fitting, easy to automate
  - Little configuration required for good results
  - Well-suited for simulation
- Disadvantages:
  - No graphical user-interface

EM-Algorithm in G-FIT

- Advantages:
  - Fast fitting, easy to automate
  - Little configuration required for good results
  - Well-suited for simulation
- Disadvantages:
  - No graphical user-interface
  - Configuration (if required) may become difficult

Optimisation

Optimisation

- Find a parameter vector $\boldsymbol{\theta}$ that minimises a distance function $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ (or, equivalently, with F)

Optimisation

- Find a parameter vector $\boldsymbol{\theta}$ that minimises a distance function $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ (or, equivalently, with F)
- Example distance functions:

Optimisation

- Find a parameter vector $\boldsymbol{\theta}$ that minimises a distance function $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ (or, equivalently, with F)
- Example distance functions:
  - Relative Entropy: $\int_0^\infty f(t) \ln \frac{f(t)}{f_{\boldsymbol{\theta}}(t)} dt$

Optimisation

- Find a parameter vector $\boldsymbol{\theta}$ that minimises a distance function $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ (or, equivalently, with F)
- Example distance functions:
  - Relative Entropy: $\int_0^\infty f(t) \ln \frac{f(t)}{f_{\boldsymbol{\theta}}(t)} dt$
  - PDF Area Distance: $\int_0^\infty |f_{\boldsymbol{\theta}}(t) - f(t)| dt$

Optimisation

- Find a parameter vector $\boldsymbol{\theta}$ that minimises a distance function $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ (or, equivalently, with F)
- Example distance functions:
  - Relative Entropy: $\int_0^\infty f(t) \ln \frac{f(t)}{f_{\boldsymbol{\theta}}(t)} dt$
  - PDF Area Distance: $\int_0^\infty |f_{\boldsymbol{\theta}}(t) - f(t)| dt$
- Non-linear optimisation problem

Optimisation

- Find a parameter vector $\boldsymbol{\theta}$ that minimises a distance function $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ (or, equivalently, with F)
- Example distance functions:
  - Relative Entropy: $\int_0^\infty f(t) \ln \frac{f(t)}{f_{\boldsymbol{\theta}}(t)} dt$
  - PDF Area Distance: $\int_0^\infty |f_{\boldsymbol{\theta}}(t) - f(t)| dt$
- Non-linear optimisation problem
- May apply different methods from non-linear optimisation

Optimisation

- Find a parameter vector $\boldsymbol{\theta}$ that minimises a distance function $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ (or, equivalently, with F)
- Example distance functions:
  - Relative Entropy: $\int_0^\infty f(t) \ln \frac{f(t)}{f_{\boldsymbol{\theta}}(t)} dt$
  - PDF Area Distance: $\int_0^\infty |f_{\boldsymbol{\theta}}(t) - f(t)| dt$
- Non-linear optimisation problem
- May apply different methods from non-linear optimisation
- Examples:

Optimisation

- Find a parameter vector $\boldsymbol{\theta}$ that minimises a distance function $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ (or, equivalently, with F)
- Example distance functions:
  - Relative Entropy: $\int_0^\infty f(t) \ln \frac{f(t)}{f_{\boldsymbol{\theta}}(t)} dt$
  - PDF Area Distance: $\int_0^\infty |f_{\boldsymbol{\theta}}(t) - f(t)| dt$
- Non-linear optimisation problem
- May apply different methods from non-linear optimisation
- Examples:
  - PhFit [8]: Frank/Wolfe method – linearisation and then linear optimisation to find the optimal direction. Supports APH in CF-1 form.

Optimisation

- Find a parameter vector $\boldsymbol{\theta}$ that minimises a distance function $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ (or, equivalently, with F)
- Example distance functions:
  - Relative Entropy: $\int_0^\infty f(t) \ln \frac{f(t)}{f_{\boldsymbol{\theta}}(t)} dt$
  - PDF Area Distance: $\int_0^\infty |f_{\boldsymbol{\theta}}(t) - f(t)| dt$
- Non-linear optimisation problem
- May apply different methods from non-linear optimisation
- Examples:
  - PhFit [8]: Frank/Wolfe method – linearisation and then linear optimisation to find the optimal direction. Supports APH in CF-1 form.
  - MonoFit: Nelder/Mead algorithm – direct optimisation without computing derivatives. Supports PH in FE-diagonal form (or in Monocyclic form).

Optimisation in PhFit

Optimisation in PhFit

- APH in CF-1 form

Optimisation in PhFit

- APH in CF-1 form
- Parameter vector: $\boldsymbol{\theta} = (\alpha_1, \ldots, \alpha_n, \lambda_1, \ldots, \lambda_n)$

Optimisation in PhFit

- APH in CF-1 form
- Parameter vector: $\boldsymbol{\theta} = (\alpha_1, \ldots, \alpha_n, \lambda_1, \ldots, \lambda_n)$
- Optimisation problem: Minimise $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ subject to

$$\boldsymbol{\alpha} \geq \mathbf{0} \quad (3)$$

$$\boldsymbol{\alpha}\mathbb{1} = 1 \quad (4)$$

$$\lambda_i > 0 \quad (5)$$

$$\lambda_i \leq \lambda_{i+1} \quad (6)$$

Optimisation in PhFit

- APH in CF-1 form
- Parameter vector: $\boldsymbol{\theta} = (\alpha_1, \ldots, \alpha_n, \lambda_1, \ldots, \lambda_n)$
- Optimisation problem: Minimise $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ subject to

$$\boldsymbol{\alpha} \geq \mathbf{0} \quad (3)$$

$$\boldsymbol{\alpha}\mathbb{1} = 1 \quad (4)$$

$$\lambda_i > 0 \quad (5)$$

$$\lambda_i \leq \lambda_{i+1} \quad (6)$$

- Apply Frank/Wolfe method to linearise in a small neighbourhood

Optimisation in PhFit

- APH in CF-1 form
- Parameter vector: $\boldsymbol{\theta} = (\alpha_1, \ldots, \alpha_n, \lambda_1, \ldots, \lambda_n)$
- Optimisation problem: Minimise $\mathcal{D}(f, f_{\boldsymbol{\theta}})$ subject to

$$\boldsymbol{\alpha} \geq \mathbf{0} \quad (3)$$

$$\boldsymbol{\alpha}\mathbb{1} = 1 \quad (4)$$

$$\lambda_i > 0 \quad (5)$$

$$\lambda_i \leq \lambda_{i+1} \quad (6)$$

- Apply Frank/Wolfe method to linearise in a small neighbourhood
- Additional constraint: Do not leave the neighbourhood

Optimisation in PhFit

Optimisation in PhFit

- Linearise in a small neighbourhood around the current parameter vector $\boldsymbol{\theta}$: Compute partial derivatives

$$\frac{\partial \mathcal{D}(f, f_{\boldsymbol{\theta}})}{\partial \theta_i}, i = 1, \ldots, 2n$$

Optimisation in PhFit

- Linearise in a small neighbourhood around the current parameter vector $\boldsymbol{\theta}$: Compute partial derivatives

$$\frac{\partial \mathcal{D}(f, f_{\boldsymbol{\theta}})}{\partial \theta_i}, i = 1, \ldots, 2n$$

- Total derivative is linear in $d\theta$:

$$\mathsf{d}\mathcal{D} = \sum_{i=1}^{2n} \frac{\partial \mathcal{D}(f, f_{\boldsymbol{\theta}})}{\partial \theta_i} \mathsf{d}\theta_i$$

Optimisation in PhFit

- Linearise in a small neighbourhood around the current parameter vector $\boldsymbol{\theta}$: Compute partial derivatives

$$\frac{\partial \mathcal{D}(f, f_{\boldsymbol{\theta}})}{\partial \theta_i}, i = 1, \ldots, 2n$$

- Total derivative is linear in $d\theta$:

$$\mathsf{d}\mathcal{D} = \sum_{i=1}^{2n} \frac{\partial \mathcal{D}(f, f_{\boldsymbol{\theta}})}{\partial \theta_i} \mathsf{d}\theta_i$$

- Minimise total derivative using Simplex method. This gives the direction of steepest descent of $\mathcal{D}$

Optimisation in PhFit

- Linearise in a small neighbourhood around the current parameter vector $\boldsymbol{\theta}$: Compute partial derivatives

$$\frac{\partial \mathcal{D}(f, f_{\boldsymbol{\theta}})}{\partial \theta_i}, i = 1, \ldots, 2n$$

- Total derivative is linear in $d\theta$:

$$\mathsf{d}\mathcal{D} = \sum_{i=1}^{2n} \frac{\partial \mathcal{D}(f, f_{\boldsymbol{\theta}})}{\partial \theta_i} \mathsf{d}\theta_i$$

- Minimise total derivative using Simplex method. This gives the direction of steepest descent of $\mathcal{D}$
- Follow this direction to find the next point

PhFit

PhFit

- Advantages:

PhFit

- Advantages:
  - Good fitting results

PhFit

- Advantages:
  - Good fitting results
  - Mixed body/tail fitting for long tails

PhFit

- Advantages:
  - Good fitting results
  - Mixed body/tail fitting for long tails
  - Well-suited for simulation

PhFit

- Advantages:
  - Good fitting results
  - Mixed body/tail fitting for long tails
  - Well-suited for simulation
  - Graphical user-interface

PhFit

- Advantages:
  - Good fitting results
  - Mixed body/tail fitting for long tails
  - Well-suited for simulation
  - Graphical user-interface
- Disadvantages:

PhFit

- Advantages:
  - Good fitting results
  - Mixed body/tail fitting for long tails
  - Well-suited for simulation
  - Graphical user-interface
- Disadvantages:
  - Fitting can be slow with large PH

PhFit

- Advantages:
  - Good fitting results
  - Mixed body/tail fitting for long tails
  - Well-suited for simulation
  - Graphical user-interface
- Disadvantages:
  - Fitting can be slow with large PH
  - Configuration can be difficult

Clustering/Segmentation

Clustering/Segmentation

- Goal: Make fitting more efficient/accurate/user-friendly

Clustering/Segmentation

- Goal: Make fitting more efficient/accurate/user-friendly
- Approach:

Clustering/Segmentation

- Goal: Make fitting more efficient/accurate/user-friendly
- Approach:
  - Split the data set S into subsets $S_1, \ldots, S_m$

Clustering/Segmentation

- Goal: Make fitting more efficient/accurate/user-friendly
- Approach:
  - Split the data set S into subsets $S_1, \ldots, S_m$
  - Fit each subset by a distribution with density $f_i(t),\ i = 1, \ldots, m$

Clustering/Segmentation

- Goal: Make fitting more efficient/accurate/user-friendly
- Approach:
  - Split the data set S into subsets $S_1, \ldots, S_m$
  - Fit each subset by a distribution with density $f_i(t),\ i = 1, \ldots, m$
  - Combine densities:

Clustering/Segmentation

- Goal: Make fitting more efficient/accurate/user-friendly
- Approach:
  - Split the data set S into subsets $S_1, \ldots, S_m$
  - Fit each subset by a distribution with density $f_i(t),\ i = 1, \ldots, m$
  - Combine densities:

$$f(t) = \sum_{i=1}^{m} \beta_i f_i(t) \tag{7}$$

$$\beta_i = \frac{|S_i|}{|S|} \tag{8}$$

Clustering/Segmentation

- Goal: Make fitting more efficient/accurate/user-friendly
- Approach:
  - Split the data set S into subsets $S_1, \ldots, S_m$
  - Fit each subset by a distribution with density $f_i(t), \; i = 1, \ldots, m$
  - Combine densities:

$$f(t) = \sum_{i=1}^{m} \beta_i f_i(t) \tag{7}$$

$$\beta_i = \frac{|S_i|}{|S|} \tag{8}$$

- Segmentation Approaches

Clustering/Segmentation

- Goal: Make fitting more efficient/accurate/user-friendly
- Approach:
  - Split the data set S into subsets $S_1, \ldots, S_m$
  - Fit each subset by a distribution with density $f_i(t),\ i = 1, \ldots, m$
  - Combine densities:

$$f(t) = \sum_{i=1}^{m} \beta_i f_i(t) \quad (7)$$

$$\beta_i = \frac{|S_i|}{|S|} \quad (8)$$

- Segmentation Approaches
- Clustering Approaches

Segmentation Approaches ([21], etc)

Segmentation Approaches ([21], etc)

- Whole family of methods

Segmentation Approaches ([21], etc)

- Whole family of methods
- Goal: Handle heavy-tailed data

Segmentation Approaches ([21], etc)

- Whole family of methods
- Goal: Handle heavy-tailed data
- Sort data, split such that the segments have a specified maximal coefficient of variation cv

Segmentation Approaches ([21], etc)

- Whole family of methods
- Goal: Handle heavy-tailed data
- Sort data, split such that the segments have a specified maximal coefficient of variation cv
- Fit each segment by. . .

Segmentation Approaches ([21], etc)

- Whole family of methods
- Goal: Handle heavy-tailed data
- Sort data, split such that the segments have a specified maximal coefficient of variation cv
- Fit each segment by...
  - an exponential distribution [17, 18]

Segmentation Approaches ([21], etc)

- Whole family of methods
- Goal: Handle heavy-tailed data
- Sort data, split such that the segments have a specified maximal coefficient of variation cv
- Fit each segment by...
  - an exponential distribution [17, 18]
  - a Hyper-Erlang distribution [22, 21]

Segmentation Approaches ([21], etc)

- Whole family of methods
- Goal: Handle heavy-tailed data
- Sort data, split such that the segments have a specified maximal coefficient of variation cv
- Fit each segment by...
  - an exponential distribution [17, 18]
  - a Hyper-Erlang distribution [22, 21]
- Fitting for the segments [21]: BEM and AEM algorithms

Segmentation Approaches ([21], etc)

- Whole family of methods
- Goal: Handle heavy-tailed data
- Sort data, split such that the segments have a specified maximal coefficient of variation cv
- Fit each segment by...
  - an exponential distribution [17, 18]
  - a Hyper-Erlang distribution [22, 21]
- Fitting for the segments [21]: BEM and AEM algorithms
- Build mixture of individual distributions.

Segmentation Approaches ([21], etc)

- Whole family of methods
- Goal: Handle heavy-tailed data
- Sort data, split such that the segments have a specified maximal coefficient of variation cv
- Fit each segment by...
  - an exponential distribution [17, 18]
  - a Hyper-Erlang distribution [22, 21]
- Fitting for the segments [21]: BEM and AEM algorithms
- Build mixture of individual distributions.
- Advantage: Requires only specification of maximal cv

Segmentation Approaches ([21], etc)

- Whole family of methods
- Goal: Handle heavy-tailed data
- Sort data, split such that the segments have a specified maximal coefficient of variation cv
- Fit each segment by...
  - an exponential distribution [17, 18]
  - a Hyper-Erlang distribution [22, 21]
- Fitting for the segments [21]: BEM and AEM algorithms
- Build mixture of individual distributions.
- Advantage: Requires only specification of maximal cv
- Disadvantage: Results depend heavily on choice of appropriate cv

Clustering (Hyper-*, [15])

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres
- Fit samples in each cluster by a user-specified PH class and method

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres
- Fit samples in each cluster by a user-specified PH class and method
  - Moment-Matching for Erlang distributions

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres
- Fit samples in each cluster by a user-specified PH class and method
  - Moment-Matching for Erlang distributions
  - PhFit, G-FIT, or other external tools

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres
- Fit samples in each cluster by a user-specified PH class and method
  - Moment-Matching for Erlang distributions
  - PhFit, G-FIT, or other external tools
  - Mathematica modules, ...

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres
- Fit samples in each cluster by a user-specified PH class and method
  - Moment-Matching for Erlang distributions
  - PhFit, G-FIT, or other external tools
  - Mathematica modules, ...
- Build mixture of individual distributions

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres
- Fit samples in each cluster by a user-specified PH class and method
  - Moment-Matching for Erlang distributions
  - PhFit, G-FIT, or other external tools
  - Mathematica modules, ...
- Build mixture of individual distributions
- Advantages:

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres
- Fit samples in each cluster by a user-specified PH class and method
  - Moment-Matching for Erlang distributions
  - PhFit, G-FIT, or other external tools
  - Mathematica modules, ...
- Build mixture of individual distributions
- Advantages:
  - Good fitting, especially with Erlang distributions for the clusters

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres
- Fit samples in each cluster by a user-specified PH class and method
  - Moment-Matching for Erlang distributions
  - PhFit, G-FIT, or other external tools
  - Mathematica modules, ...
- Build mixture of individual distributions
- Advantages:
  - Good fitting, especially with Erlang distributions for the clusters
  - Intuitive configuration

Clustering (Hyper-*, [15])

- Goal: Fit empirical distributions with peaks well
- Use k-means algorithm to create clusters
- User selects cluster centres
- Fit samples in each cluster by a user-specified PH class and method
  - Moment-Matching for Erlang distributions
  - PhFit, G-FIT, or other external tools
  - Mathematica modules, ...
- Build mixture of individual distributions
- Advantages:
  - Good fitting, especially with Erlang distributions for the clusters
  - Intuitive configuration
- Disadvantage: Fitted distributions can become very large

Comparison of Fitting Tools

Comparison of Fitting Tools

- Three data sets

Comparison of Fitting Tools

- Three data sets
  - APH distribution

Comparison of Fitting Tools

- Three data sets
  - APH distribution
  - Packet-delivery ratios from the DES-Testbed [3]

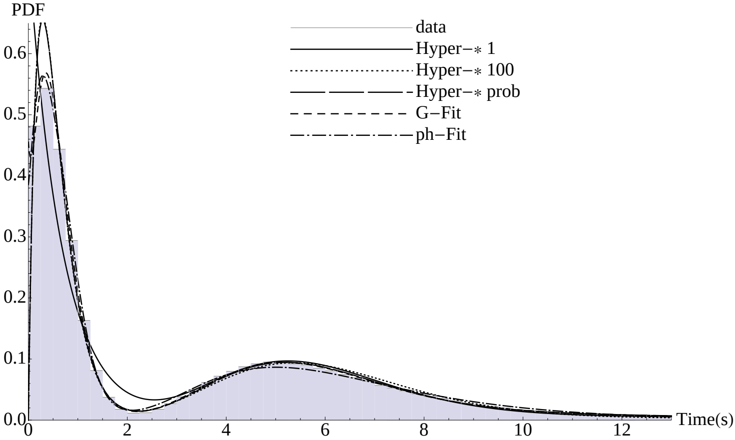
Comparison of Fitting Tools

- Three data sets
  - APH distribution
  - Packet-delivery ratios from the DES-Testbed [3]
  - PTP packet transmission delays

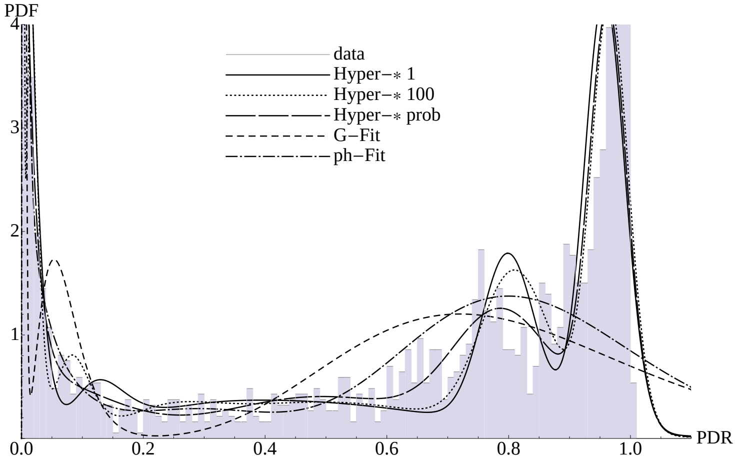
Comparison of Fitting Tools

- Three data sets
  - APH distribution
  - Packet-delivery ratios from the DES-Testbed [3]
  - PTP packet transmission delays
- Parameters chosen similarly, if possible

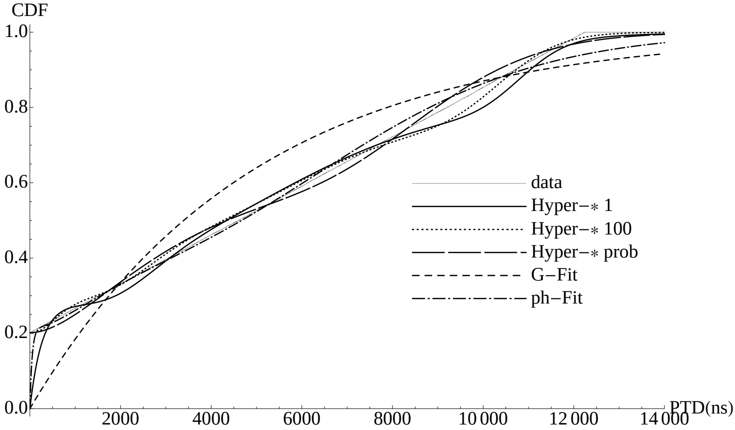
APH distribution



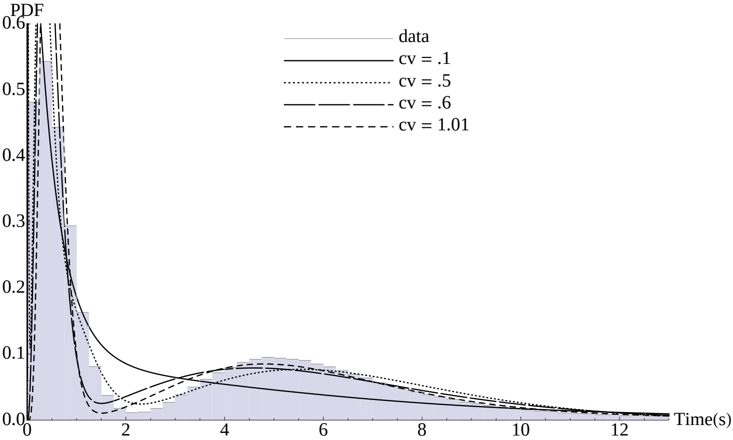
Packet-delivery ratio distribution



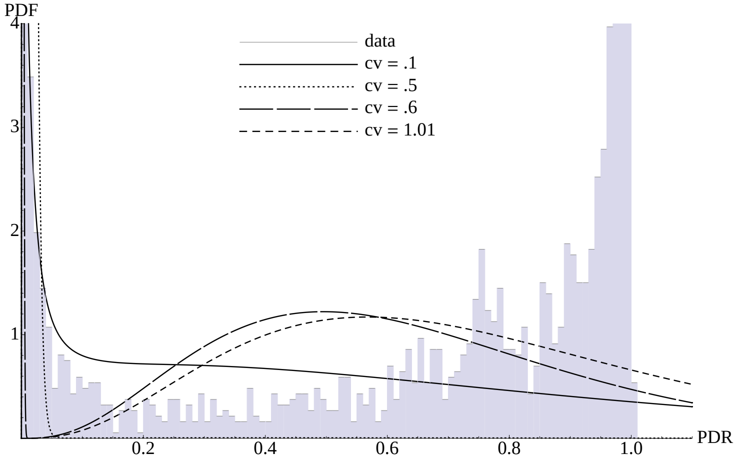
PTD distribution



APH distribution (Segmentation approach)



Packet-delivery ratio distribution (Segmentation approach)



Summary

Summary

- Many different approaches to PH fitting exist

Summary

- Many different approaches to PH fitting exist
- Suitability of approaches depends on

Summary

- Many different approaches to PH fitting exist
- Suitability of approaches depends on
  - Required quality of fit

Summary

- Many different approaches to PH fitting exist
- Suitability of approaches depends on
  - Required quality of fit
  - Shape of the empirical density

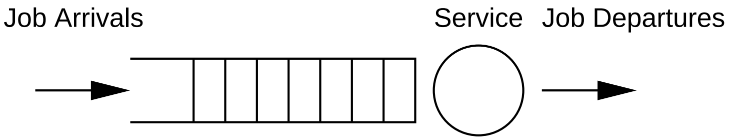
Summary

- Many different approaches to PH fitting exist
- Suitability of approaches depends on
  - Required quality of fit
  - Shape of the empirical density
  - Intended application of the distribution

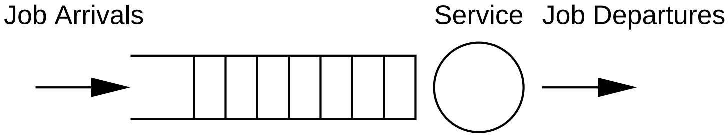
Summary

- Many different approaches to PH fitting exist
- Suitability of approaches depends on
  - Required quality of fit
  - Shape of the empirical density
  - Intended application of the distribution
  - Expertise of the user and user-friendliness of the tool

Queueing Theory

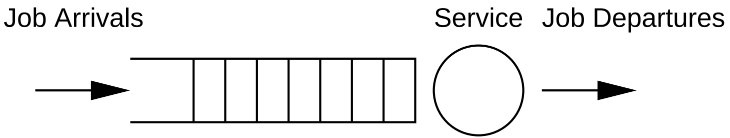


Queueing Theory



- Jobs arrive, are processed, and leave

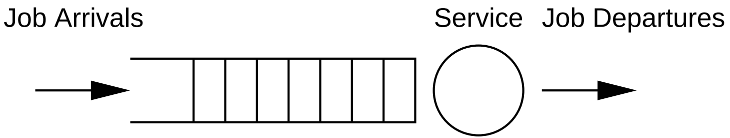
Queueing Theory



- Jobs arrive, are processed, and leave
- Kendall notation:

  Arrival process/Service process/Number of servers(/ ...)

Queueing Theory

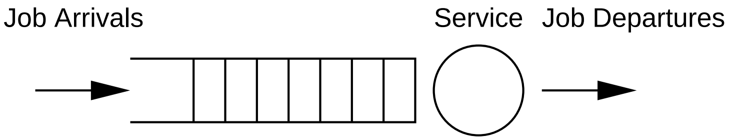


- Jobs arrive, are processed, and leave
- Kendall notation:

  Arrival process/Service process/Number of servers(/ . . .)

- Examples
  - $M/M/1$
  - $M/PH/1$
  - $PH/PH/1$

Queueing Theory



- Jobs arrive, are processed, and leave
- Kendall notation:

  Arrival process/Service process/Number of servers$(/\ldots)$

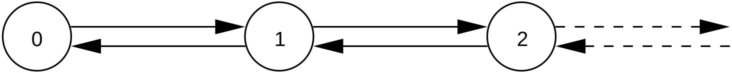
- Examples
  - $M/M/1$
  - $M/PH/1$
  - $PH/PH/1$
- Typical questions:
  - Average number of jobs in the system?
  - Quantiles of the queue-length distribution?

Analysis

- Queue only changes on arrivals or departures

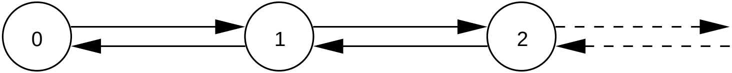
Analysis

- Queue only changes on arrivals or departures → 'Birth/Death process':

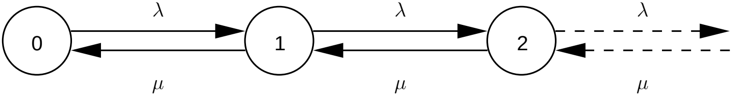


Analysis

- Queue only changes on arrivals or departures → 'Birth/Death process':

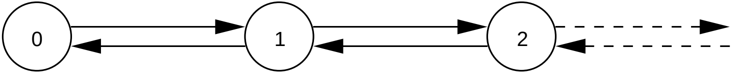


- For the $M/M/1$ queue, this is a CTMC with infinite state-space:

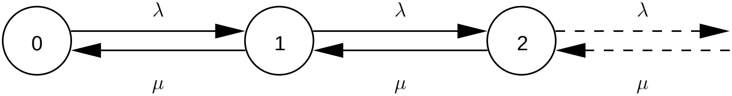


Analysis

- Queue only changes on arrivals or departures → 'Birth/Death process':



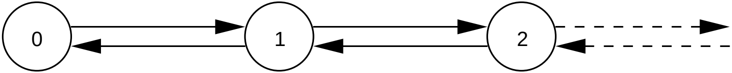
- For the $M/M/1$ queue, this is a CTMC with infinite state-space:



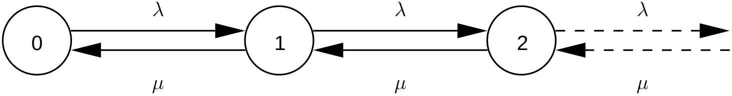
- For the $M/PH/1$ queue, things get a bit more interesting:
  - Infinite state-space and phase-transitions
  - Finite number of phases for any number of jobs

Analysis

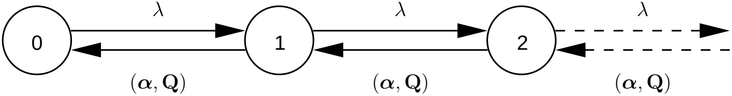
- Queue only changes on arrivals or departures → 'Birth/Death process':



- For the $M/M/1$ queue, this is a CTMC with infinite state-space:



- For the $M/PH/1$ queue, things get a bit more interesting:
  - Infinite state-space and phase-transitions
  - Finite number of phases for any number of jobs
- Block-transitions → 'Quasi-Birth/Death process':



What can we do?

What can we do?

- Compute transient measures, e.g. time until we first have m jobs in the queue

What can we do?

- Compute transient measures, e.g. time until we first have m jobs in the queue
- Compute steady-state distribution, i.e. stochastic vector $\mathbf{x}$ such that

$$\mathbf{x}\mathbf{Q} = \mathbf{0} \quad (9)$$

$$\mathbf{x}\mathbb{1} = 1 \quad (10)$$

What can we do?

- Compute transient measures, e.g. time until we first have m jobs in the queue
- Compute steady-state distribution, i.e. stochastic vector $\mathbf{x}$ such that

$$\mathbf{xQ} = \mathbf{0} \quad (9)$$

$$\mathbf{x1\!\!I} = 1 \quad (10)$$

- Prerequisite for steady-state solution: Queue must be stable, i.e. jobs must not arrive faster than they can be served:

$$\rho = \frac{E[S]}{E[A]} < 1$$

Matrix-Geometric Methods

Generator matrix of the CTMC:

$$\mathbf{Q} = \begin{pmatrix} -\lambda & \lambda\boldsymbol{\alpha} & & & & \\ \mathbf{q} & (\mathbf{Q} - \lambda\mathbf{I}) & \lambda\mathbf{I} & & & \\ & \mathbf{q}\boldsymbol{\alpha} & (\mathbf{Q} - \lambda\mathbf{I}) & \lambda\mathbf{I} & & \\ & & \mathbf{q}\boldsymbol{\alpha} & (-\mathbf{Q} - \lambda\mathbf{I}) & \lambda\mathbf{I} & \\ & & & & & \ddots \end{pmatrix}$$

Matrix-Geometric Methods

Generator matrix of the CTMC:

$$\mathbf{Q} = \begin{pmatrix} -\lambda & \lambda\boldsymbol{\alpha} & & & \\ \mathbf{q} & (\mathbf{Q}-\lambda\mathbf{I}) & \lambda\mathbf{I} & & \\ & \mathbf{q}\boldsymbol{\alpha} & (\mathbf{Q}-\lambda\mathbf{I}) & \lambda\mathbf{I} & \\ & & \mathbf{q}\boldsymbol{\alpha} & (-\mathbf{Q}-\lambda\mathbf{I}) & \lambda\mathbf{I} \\ & & & & \ddots \end{pmatrix}$$

...nice, regular structure, leading to

$$\mathbf{xQ} = \mathbf{0} \quad \Leftrightarrow \quad \begin{cases} x_0(-\lambda) + \mathbf{x}_1\mathbf{q} = 0 \\ x_0(\lambda\boldsymbol{\alpha}) + \mathbf{x}_1(\mathbf{Q}-\lambda\mathbf{I}) + \mathbf{x}_2(\mathbf{q}\boldsymbol{\alpha}) = \mathbf{0} \\ \mathbf{x}_{i-1}(\lambda\mathbf{I}) + \mathbf{x}_i(\mathbf{Q}-\lambda\mathbf{I}) + \mathbf{x}_{i+1}(\mathbf{q}\boldsymbol{\alpha}) = \mathbf{0} \quad i \geq 2, \end{cases}$$

where

$$\mathbf{x} = (x_0, \mathbf{x}_1, \mathbf{x}_2, \dots)$$

gives the steady-state probabilities.

Solution for M/PH/1

Theorem 3.2.1 in [13]:

$$\begin{aligned} \rho &= \lambda E[S] \\ x_0 &= 1 - \rho \\ \mathbf{x}_i &= (1-\rho)\boldsymbol{\beta}\mathbf{R}^i \;\; i \geq 1, \end{aligned}$$

where

$$\mathbf{R} = \lambda(\lambda\mathbf{I} - \lambda\mathbf{e}\boldsymbol{\beta} - \mathbf{Q})^{-1}$$

Solution for M/PH/1

Theorem 3.2.1 in [13]:

$$\begin{aligned} \rho &= \lambda E[S] \\ x_0 &= 1-\rho \\ \mathbf{x}_i &= (1-\rho)\boldsymbol{\beta}\mathbf{R}^i \quad i \geq 1, \end{aligned}$$

where

$$\mathbf{R} = \lambda(\lambda\mathbf{I} - \lambda\mathbf{e}\boldsymbol{\beta} - \mathbf{Q})^{-1}$$

Note:

- $\mathbf{x}$: steady-state distribution of number of jobs in system and phase of the job in service
- Phases have no physical interpretation with a fitted phase-type distribution $\rightarrow$ We are only interested in the distribution of the number of jobs in the system:

$$\bar{\mathbf{x}} = (x_0, \mathbf{x}_1 \mathbb{1}, \mathbf{x}_2 \mathbb{1}, \dots)$$

Summary

- Closed-form expressions allow analytical approaches
- Efficient solution methods due to special structures of the resulting models
- In queueing-analysis, matrix-geometric methods utilise block structures
- Solutions for more general systems are available, e.g. $PH/PH/1$, or queues with bounded queue size

Random-variate Generation

Random-variate Generation

- Goal: Efficiently generate random variates from a given PH distribution

Random-variate Generation

- Goal: Efficiently generate random variates from a given PH distribution
- Different methods:

Random-variate Generation

- Goal: Efficiently generate random variates from a given PH distribution
- Different methods:
  - Inversion methods

Random-variate Generation

- Goal: Efficiently generate random variates from a given PH distribution
- Different methods:
  - Inversion methods
  - Acceptance/Rejection methods

Random-variate Generation

- Goal: Efficiently generate random variates from a given PH distribution
- Different methods:
  - Inversion methods
  - Acceptance/Rejection methods
  - Characterisation/Play methods

Elementary and Atomic Operations

Elementary and Atomic Operations

- Uniform random number in $(0, 1)$: u

Elementary and Atomic Operations

- Uniform random number in $(0,1)$: u
- Random variate from the geometric distribution on $0,1,\ldots$:

$$t_{\mathsf{Geo}(p)} := \left\lfloor \frac{\ln(u)}{\ln(p)} \right\rfloor$$

Elementary and Atomic Operations

- Uniform random number in $(0,1)$: u
- Random variate from the geometric distribution on $0, 1, \ldots$:

$$t_{\mathsf{Geo}(p)} := \left\lfloor \frac{\ln(u)}{\ln(p)} \right\rfloor$$

- Random variate from the exponential distribution with rate λ:

$$t_{\mathsf{Exp}(\lambda)} := -\frac{1}{\lambda} \ln u$$

Elementary and Atomic Operations

- Uniform random number in $(0,1)$: u
- Random variate from the geometric distribution on $0, 1, \ldots$:

$$t_{\mathsf{Geo}(p)} := \left\lfloor \frac{\ln(u)}{\ln(p)} \right\rfloor$$

- Random variate from the exponential distribution with rate λ:

$$t_{\mathsf{Exp}(\lambda)} := -\frac{1}{\lambda} \ln u$$

- Matrix exponential e^{Q}:

Elementary and Atomic Operations

- Uniform random number in $(0,1)$: u
- Random variate from the geometric distribution on $0, 1, \ldots$:

$$t_{\mathsf{Geo}(p)} := \left\lfloor \frac{\ln(u)}{\ln(p)} \right\rfloor$$

- Random variate from the exponential distribution with rate λ:

$$t_{\mathsf{Exp}(\lambda)} := -\frac{1}{\lambda} \ln u$$

- Matrix exponential e^{Q}:
  - Many different methods (‘19 dubious ways. . . ’ [12])

Elementary and Atomic Operations

- Uniform random number in $(0,1)$: u
- Random variate from the geometric distribution on $0,1,\ldots$:

$$t_{\mathsf{Geo}(p)} := \left\lfloor \frac{\ln(u)}{\ln(p)} \right\rfloor$$

- Random variate from the exponential distribution with rate λ:

$$t_{\mathsf{Exp}(\lambda)} := -\frac{1}{\lambda} \ln u$$

- Matrix exponential e^{Q}:
  - Many different methods ('19 dubious ways. . . ' [12])
  - Can be reduced to computation of n scalar exponentials

Atomic Operations and Cost Metrics

Atomic Operations and Cost Metrics

- Computation of a uniform random number

Atomic Operations and Cost Metrics

- Computation of a uniform random number
- Computation of a scalar exponential

Atomic Operations and Cost Metrics

- Computation of a uniform random number
- Computation of a scalar exponential
- Computation of a logarithm

Atomic Operations and Cost Metrics

- Computation of a uniform random number
- Computation of a scalar exponential
- Computation of a logarithm
- Cost metrics:

Atomic Operations and Cost Metrics

- Computation of a uniform random number
- Computation of a scalar exponential
- Computation of a logarithm
- Cost metrics:
  - Number of uniforms, #uni

Atomic Operations and Cost Metrics

- Computation of a uniform random number
- Computation of a scalar exponential
- Computation of a logarithm
- Cost metrics:
  - Number of uniforms, #uni
  - Number of scalar exponentials, #exp

Atomic Operations and Cost Metrics

- Computation of a uniform random number
- Computation of a scalar exponential
- Computation of a logarithm
- Cost metrics:
  - Number of uniforms, #uni
  - Number of scalar exponentials, #exp
  - Number of logarithms, #ln

Atomic Operations and Cost Metrics

- Computation of a uniform random number
- Computation of a scalar exponential
- Computation of a logarithm
- Cost metrics:
  - Number of uniforms, #uni
  - Number of scalar exponentials, #exp
  - Number of logarithms, #ln
- . . . for the worst case and for the average case

Inversion method

Inversion method

- $F(t) \sim U(0,1) \Rightarrow t = F^{-1}(u) \sim F$

Inversion method

- $F(t) \sim U(0,1) \Rightarrow t = F^{-1}(u) \sim F$
- Example: Exponential distribution

Inversion method

- $F(t) \sim U(0,1) \Rightarrow t = F^{-1}(u) \sim F$
- Example: Exponential distribution

$$\begin{aligned} u &= F(t) = 1 - \mathrm{e}^{-\lambda t} && (11) \\ \Leftrightarrow t &= -\frac{1}{\lambda} \ln(1-u), && (12) \end{aligned}$$

Inversion method

- $F(t) \sim U(0,1) \Rightarrow t = F^{-1}(u) \sim F$
- Example: Exponential distribution

$$u = F(t) = 1 - \mathrm{e}^{-\lambda t} \tag{11}$$

$$\Leftrightarrow t = -\frac{1}{\lambda} \ln(1-u), \tag{12}$$

and, since $u \sim U(0,1) \Rightarrow (1-u) \sim U(0,1)$, we can simplify:

$$t = -\frac{1}{\lambda} \ln u. \tag{13}$$

Inversion

Inversion

- Direct inversion of

$$F(t) = 1 - \boldsymbol{\alpha} \exp^{\mathbf{Q}t} \mathbb{1}$$

impossible for $n > 1$

Inversion

- Direct inversion of

$$F(t) = 1 - \boldsymbol{\alpha} \exp^{\mathbf{Q}t} \mathbb{1}$$

  impossible for $n > 1$
- Numerical inversion [4]: Identify t close to $F(u)$ by binary search:

Inversion

- Direct inversion of

$$F(t) = 1 - \boldsymbol{\alpha} \exp^{\mathbf{Q}t} \mathbb{1}$$

  impossible for $n > 1$
- Numerical inversion [4]: Identify t close to $F(u)$ by binary search:
  - Let $[a, b]$ be the interval, with center $t = \frac{a+b}{2}$

Inversion

- Direct inversion of

$$F(t) = 1 - \boldsymbol{\alpha} \exp^{\mathbf{Q}t} \mathbb{1}$$

  impossible for $n > 1$
- Numerical inversion [4]: Identify t close to $F(u)$ by binary search:
  - Let $[a, b]$ be the interval, with center $t = \frac{a+b}{2}$
  - If $F(t) > F(u)$, set $a := t$, else set $b := t$.

Inversion

- Direct inversion of

$$F(t) = 1 - \boldsymbol{\alpha} \exp^{\mathbf{Q}t} \mathbb{1}$$

  impossible for $n > 1$
- Numerical inversion [4]: Identify t close to $F(u)$ by binary search:
  - Let $[a, b]$ be the interval, with center $t = \frac{a+b}{2}$
  - If $F(t) > F(u)$, set $a := t$, else set $b := t$.
  - Stop and return t if $F(t) \sim F(t)$

Inversion (ctd.)

Inversion (ctd.)

- Valid for Matrix-Exponential and PH distributions in any form

Inversion (ctd.)

- Valid for Matrix-Exponential and PH distributions in any form
- Costs:

Inversion (ctd.)

- Valid for Matrix-Exponential and PH distributions in any form
- Costs:
  - Number of steps: $\log \frac{1}{\delta}$ for accuracy of δ

Inversion (ctd.)

- Valid for Matrix-Exponential and PH distributions in any form
- Costs:
  - Number of steps: $\log \frac{1}{\delta}$ for accuracy of δ
  - [4]: $\delta = 10^{-6} \Rightarrow 19$ steps

Inversion (ctd.)

- Valid for Matrix-Exponential and PH distributions in any form
- Costs:
  - Number of steps: $\log \frac{1}{\delta}$ for accuracy of δ
  - [4]: $\delta = 10^{-6} \Rightarrow 19$ steps
  - One matrix exponential for each step

Inversion (ctd.)

- Valid for Matrix-Exponential and PH distributions in any form
- Costs:
  - Number of steps: $\log \frac{1}{\delta}$ for accuracy of δ
  - [4]: $\delta = 10^{-6} \Rightarrow 19$ steps
  - One matrix exponential for each step
  - If the matrix exponential is computed from scalar exponentials, n scalar exponentials for each step

Inversion (ctd.)

- Valid for Matrix-Exponential and PH distributions in any form
- Costs:
  - Number of steps: $\log \frac{1}{\delta}$ for accuracy of δ
  - [4]: $\delta = 10^{-6} \Rightarrow 19$ steps
  - One matrix exponential for each step
  - If the matrix exponential is computed from scalar exponentials, n scalar exponentials for each step
  - One uniform random number

Acceptance/Rejection [9]

Acceptance/Rejection [9]

- Split the density into parts with positive and parts with negative coefficients:

$$f(t) = \boldsymbol{\alpha} \mathrm{e}^{\mathbf{Q}t}(-\mathbf{Q}\mathbb{1})$$

Acceptance/Rejection [9]

- Split the density into parts with positive and parts with negative coefficients:

$$\begin{aligned} f(t) &= \boldsymbol{\alpha} \mathrm{e}^{\mathbf{Q}t}(-\mathbf{Q}\mathbb{1}) \\ &= \sum_{i \in \mathcal{A}_+} \alpha_i g_i(t) + \sum_{i \in \mathcal{A}_-} \alpha_i g_i(t) \end{aligned}$$

Acceptance/Rejection [9]

- Split the density into parts with positive and parts with negative coefficients:

$$\begin{aligned} f(t) &= \boldsymbol{\alpha} e^{\mathbf{Q}t}(-\mathbf{Q}\mathbb{1}) \\ &= \sum_{i \in \mathcal{A}_+} \alpha_i g_i(t) + \sum_{i \in \mathcal{A}_-} \alpha_i g_i(t) \\ &= f_+(t) + f_-(t) \end{aligned}$$

Acceptance/Rejection [9]

- Split the density into parts with positive and parts with negative coefficients:

$$\begin{aligned} f(t) &= \boldsymbol{\alpha}\mathrm{e}^{\mathbf{Q}t}(-\mathbf{Q}\mathbb{1}) \\ &= \sum_{i\in\mathcal{A}_+}\alpha_i g_i(t) + \sum_{i\in\mathcal{A}_-}\alpha_i g_i(t) \\ &= f_+(t) + f_-(t) \end{aligned}$$

- $f_+(t)$ can be normalised to a PH density:

$$\hat{f}(t) = \frac{1}{\sum_{i\in\mathcal{A}_+}\alpha_i} f_+(t).$$

Acceptance/Rejection [9]

- Split the density into parts with positive and parts with negative coefficients:

$$\begin{aligned} f(t) &= \boldsymbol{\alpha} \mathrm{e}^{\mathbf{Q}t}(-\mathbf{Q}\mathbb{1}) \\ &= \sum_{i \in \mathcal{A}_+} \alpha_i g_i(t) + \sum_{i \in \mathcal{A}_-} \alpha_i g_i(t) \\ &= f_+(t) + f_-(t) \end{aligned}$$

- $f_+(t)$ can be normalised to a PH density:

$$\hat{f}(t) = \frac{1}{\sum_{i \in \mathcal{A}_+} \alpha_i} f_+(t).$$

- A sample x from $\hat{f}(t)$ is accepted with

$$p = \frac{f_+(x) + f_-(x)}{f_+(x)}$$

Acceptance/Rejection (ctd.)

Acceptance/Rejection (ctd.)

- Supports Matrix-Exponential and Phase-type distributions

Acceptance/Rejection (ctd.)

- Supports Matrix-Exponential and Phase-type distributions
- Support PH distributions in non-Markovian representation

Acceptance/Rejection (ctd.)

- Supports Matrix-Exponential and Phase-type distributions
- Support PH distributions in non-Markovian representation
- Enables efficient algorithms for PH:

Acceptance/Rejection (ctd.)

- Supports Matrix-Exponential and Phase-type distributions
- Support PH distributions in non-Markovian representation
- Enables efficient algorithms for PH:
  - Transform PH to e.g. Hyper-Feedback-Erlang form

Acceptance/Rejection (ctd.)

- Supports Matrix-Exponential and Phase-type distributions
- Support PH distributions in non-Markovian representation
- Enables efficient algorithms for PH:
  - Transform PH to e.g. Hyper-Feedback-Erlang form
  - ... which may be non-Markovian

Acceptance/Rejection (ctd.)

- Supports Matrix-Exponential and Phase-type distributions
- Support PH distributions in non-Markovian representation
- Enables efficient algorithms for PH:
  - Transform PH to e.g. Hyper-Feedback-Erlang form
  - . . . which may be non-Markovian
  - Draw random variates using Acceptance/Rejection

Acceptance/Rejection (ctd.)

- Supports Matrix-Exponential and Phase-type distributions
- Support PH distributions in non-Markovian representation
- Enables efficient algorithms for PH:
  - Transform PH to e.g. Hyper-Feedback-Erlang form
  - ...which may be non-Markovian
  - Draw random variates using Acceptance/Rejection
- Costs:

Acceptance/Rejection (ctd.)

- Supports Matrix-Exponential and Phase-type distributions
- Support PH distributions in non-Markovian representation
- Enables efficient algorithms for PH:
  - Transform PH to e.g. Hyper-Feedback-Erlang form
  - ...which may be non-Markovian
  - Draw random variates using Acceptance/Rejection
- Costs:
  - Number of steps: $\frac{1}{p}$

Acceptance/Rejection (ctd.)

- Supports Matrix-Exponential and Phase-type distributions
- Support PH distributions in non-Markovian representation
- Enables efficient algorithms for PH:
  - Transform PH to e.g. Hyper-Feedback-Erlang form
  - ...which may be non-Markovian
  - Draw random variates using Acceptance/Rejection
- Costs:
  - Number of steps: $\frac{1}{p}$
  - Number of uniforms and number of logarithms depends on the method used for drawing from $\hat{f}$

Characterisation methods

Characterisation methods

- Create random variates using the CTMC representation

Characterisation methods

- Create random variates using the CTMC representation
- Costs depend on

Characterisation methods

- Create random variates using the CTMC representation
- Costs depend on
  - Number of traversed states

Characterisation methods

- Create random variates using the CTMC representation
- Costs depend on
  - Number of traversed states
  - Costs per state

Characterisation methods

- Create random variates using the CTMC representation
- Costs depend on
  - Number of traversed states
  - Costs per state
- Methods support different classes and representations:

Characterisation methods

- Create random variates using the CTMC representation
- Costs depend on
  - Number of traversed states
  - Costs per state
- Methods support different classes and representations:
  - General PH: `Play`, `Count`

Characterisation methods

- Create random variates using the CTMC representation
- Costs depend on
  - Number of traversed states
  - Costs per state
- Methods support different classes and representations:
  - General PH: `Play`, `Count`
  - PH in FE-diagonal form: `FE-Diagonal`

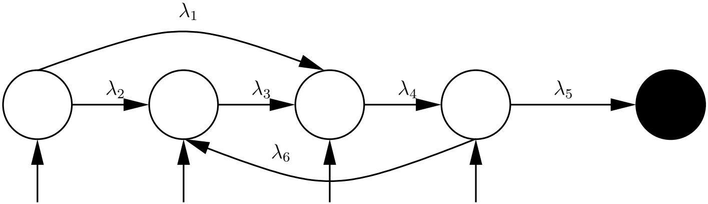
Characterisation methods

- Create random variates using the CTMC representation
- Costs depend on
  - Number of traversed states
  - Costs per state
- Methods support different classes and representations:
  - General PH: `Play`, `Count`
  - PH in FE-diagonal form: `FE-Diagonal`
  - APH in bi-diagonal form: `SimplePlay`

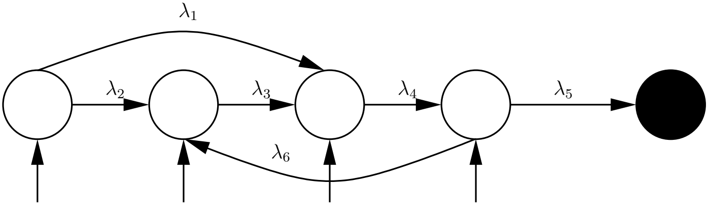
Characterisation methods

- Create random variates using the CTMC representation
- Costs depend on
  - Number of traversed states
  - Costs per state
- Methods support different classes and representations:
  - General PH: `Play`, `Count`
  - PH in FE-diagonal form: `FE-Diagonal`
  - APH in bi-diagonal form: `SimplePlay`
  - HErD in HErD form: `SimpleCount`

Play

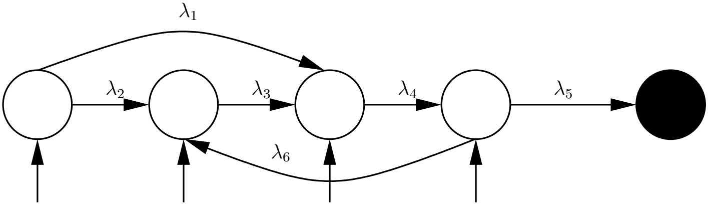


Play



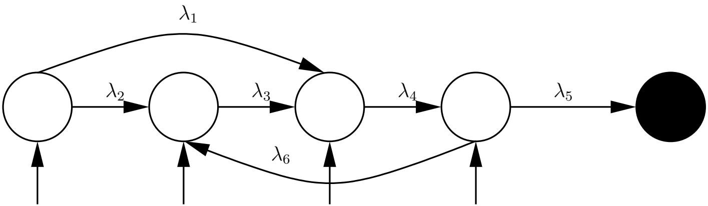
- Play the Markov chain: Select an initial state, then select successive states until the absorbing state is reached. Draw one exponential random variate for each visited state.

Play



- Play the Markov chain: Select an initial state, then select successive states until the absorbing state is reached. Draw one exponential random variate for each visited state.
- Worst-case number of traversals: Not defined

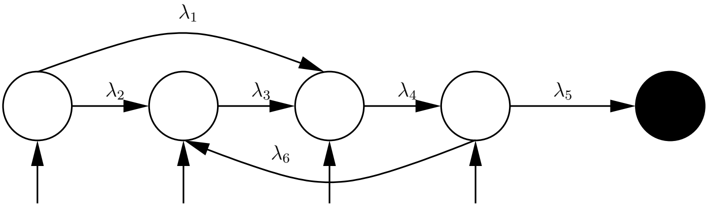
Play



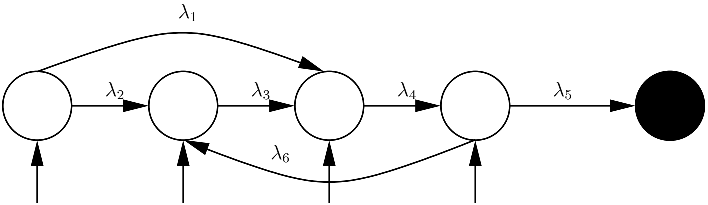
- Play the Markov chain: Select an initial state, then select successive states until the absorbing state is reached. Draw one exponential random variate for each visited state.
- Worst-case number of traversals: Not defined
- Average-case number of traversals:

$$n^* = \boldsymbol{\alpha}(\text{diag}(\mathbf{Q})^{-1}\mathbf{Q})^{-1}\mathbb{1}$$

Play (ctd.)

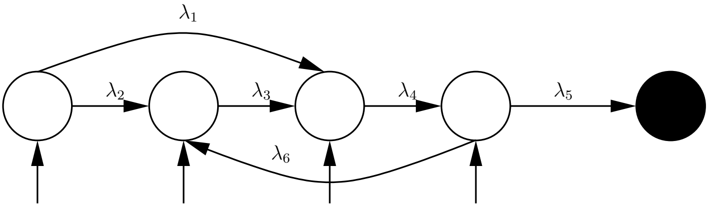


Play (ctd.)



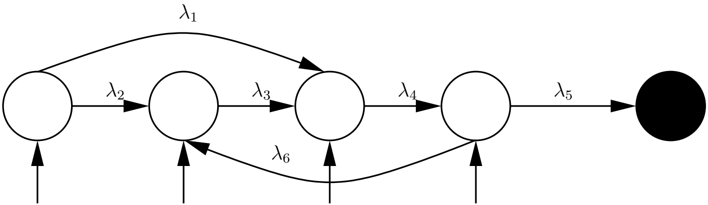
- Costs:

Play (ctd.)



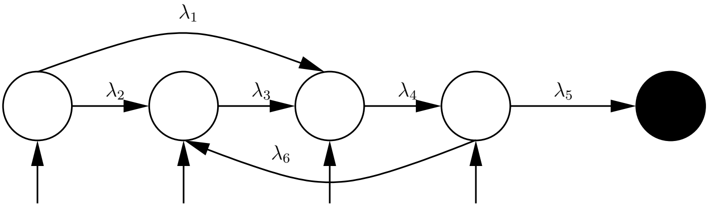
- Costs:
  - 1 uniform for initial selection

Play (ctd.)



- Costs:
  - 1 uniform for initial selection
  - 2 uniforms for each visit to a state

Play (ctd.)



- Costs:
  - 1 uniform for initial selection
  - 2 uniforms for each visit to a state
  - 1 logarithm for each visit to a state

Count [14]

Count [14]

- Observation: k visits to the same state are equal to drawing an Erlang-k distribution.

Count [14]

- Observation: k visits to the same state are equal to drawing an Erlang-k distribution.
- Idea: Use $\sum \ln = \ln \prod$

Count [14]

- Observation: k visits to the same state are equal to drawing an Erlang-k distribution.
- Idea: Use $\sum \ln = \ln \prod$
- Algorithm: Play Markov chain, count numbers of visits, draw n Erlangs with appropriate lengths

Count [14]

- Observation: k visits to the same state are equal to drawing an Erlang-k distribution.
- Idea: Use $\sum \ln = \ln \prod$
- Algorithm: Play Markov chain, count numbers of visits, draw n Erlangs with appropriate lengths
- State traversals: Same as Play

Count [14]

- Observation: k visits to the same state are equal to drawing an Erlang-k distribution.
- Idea: Use $\sum \ln = \ln \prod$
- Algorithm: Play Markov chain, count numbers of visits, draw n Erlangs with appropriate lengths
- State traversals: Same as Play
- Costs:

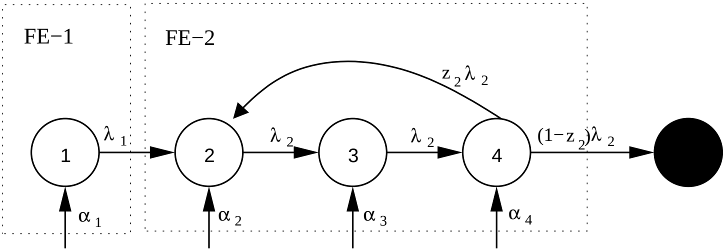
Count [14]

- Observation: k visits to the same state are equal to drawing an Erlang-k distribution.
- Idea: Use $\sum \ln = \ln \prod$
- Algorithm: Play Markov chain, count numbers of visits, draw n Erlangs with appropriate lengths
- State traversals: Same as Play
- Costs:
  - Worst-Case: $\#ln = n, \#uni = \infty$

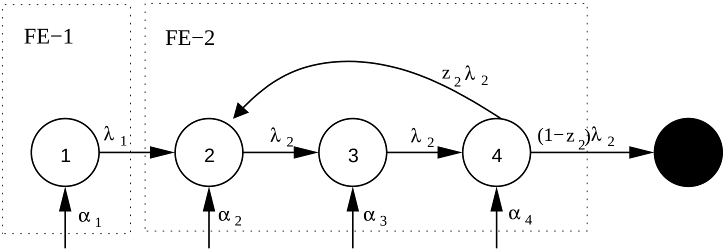
Count [14]

- Observation: k visits to the same state are equal to drawing an Erlang-k distribution.
- Idea: Use $\sum \ln = \ln \prod$
- Algorithm: Play Markov chain, count numbers of visits, draw n Erlangs with appropriate lengths
- State traversals: Same as Play
- Costs:
  - Worst-Case: $\#ln = n, \#uni = \infty$
  - Average: $\#ln = n, \#uni = 1 + n^*$

FE-diagonal Algorithm

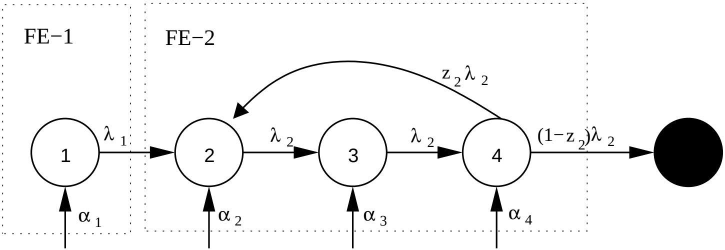


FE-diagonal Algorithm



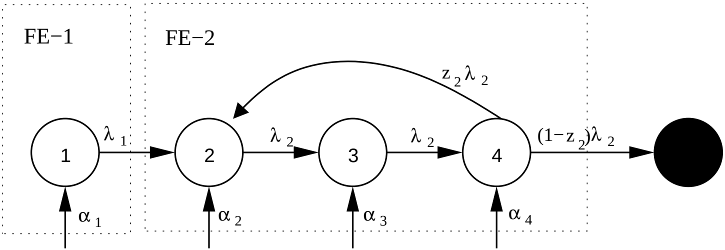
- Use FE-diagonal form

FE-diagonal Algorithm



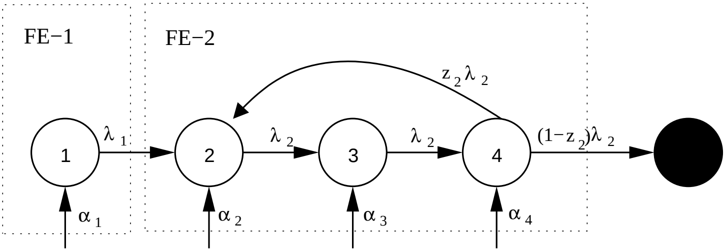
- Use FE-diagonal form
- Select an initial state according to α. This state belongs to block $1 \leq i \leq m$.

FE-diagonal Algorithm



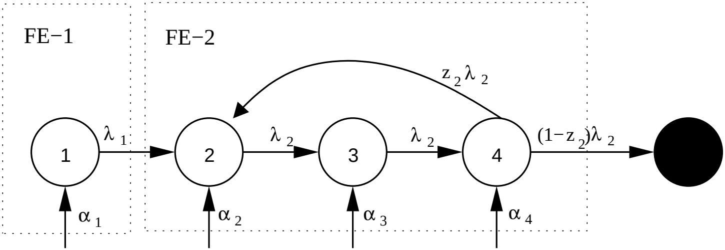
- Use FE-diagonal form
- Select an initial state according to $\boldsymbol{\alpha}$. This state belongs to block $1 \leq i \leq m$.
- $0 \leq l \leq b_i$ states have to be traversed before the next feedback loop

FE-diagonal Algorithm

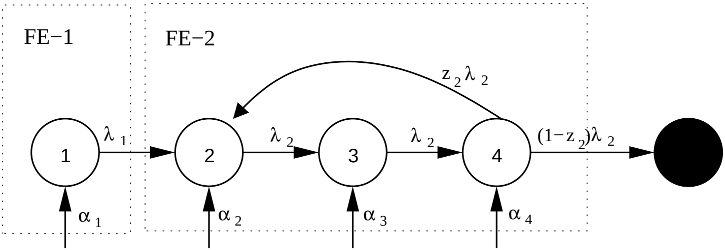


- Use FE-diagonal form
- Select an initial state according to $\boldsymbol{\alpha}$. This state belongs to block $1 \leq i \leq m$.
- $0 \leq l \leq b_i$ states have to be traversed before the next feedback loop
- The number of loops c follows a geometric distribution with parameter z_i

FE-diagonal Algorithm (ctd.)

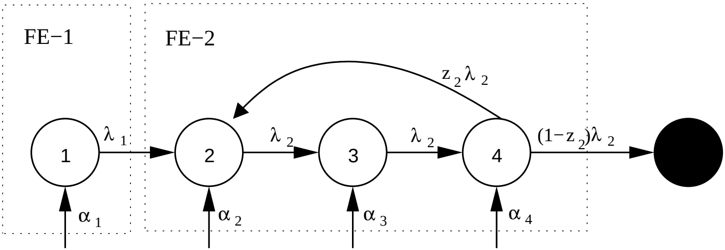


FE-diagonal Algorithm (ctd.)



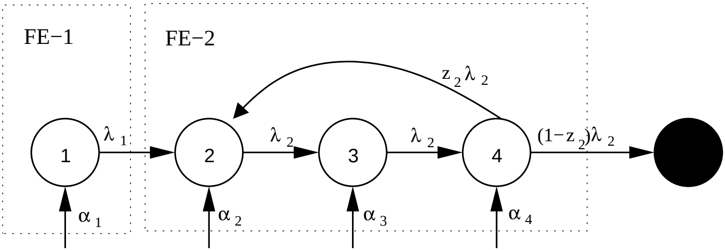
- Note: All rates in a block are identical

FE-diagonal Algorithm (ctd.)



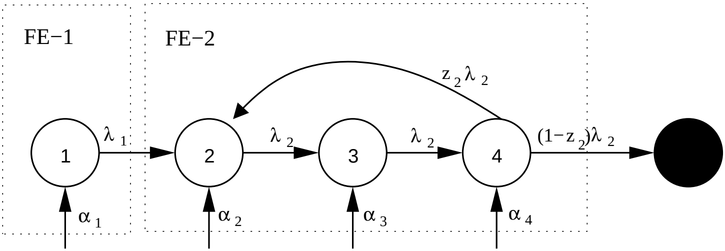
- Note: All rates in a block are identical . . . draw one Erlang-$(c \cdot b_i + l)$-distributed sample

FE-diagonal Algorithm (ctd.)



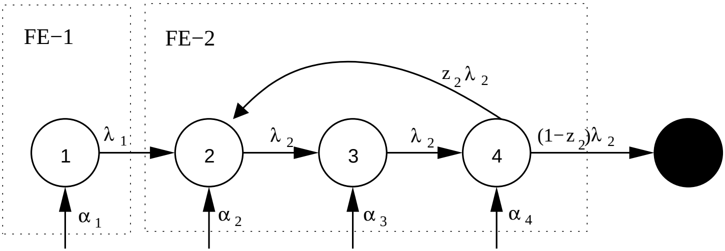
- Note: All rates in a block are identical . . . draw one Erlang-$(c \cdot b_i + l)$-distributed sample
- Repeat for the remaining blocks $j = i + 1, \ldots, m$, with $l := b_j$

FE-diagonal Algorithm (ctd.)



- Note: All rates in a block are identical . . . draw one Erlang-$(c \cdot b_i + l)$-distributed sample
- Repeat for the remaining blocks $j = i + 1, \ldots, m$, with $l := b_j$
- Costs: 1 uniform for initial state, 1 uniform for each visit, 1 uniform and 3 logarithms for each block

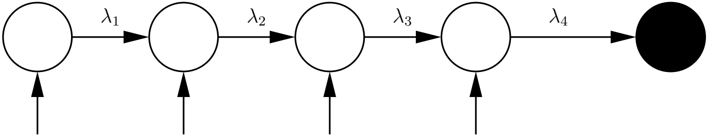
FE-diagonal Algorithm (ctd.)



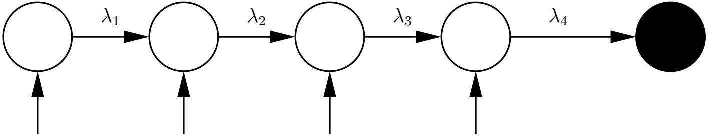
- Note: All rates in a block are identical ... draw one Erlang-$(c \cdot b_i + l)$-distributed sample
- Repeat for the remaining blocks $j = i+1, \ldots, m$, with $l := b_j$
- Costs: 1 uniform for initial state, 1 uniform for each visit, 1 uniform and 3 logarithms for each block
- Average number of traversed blocks:

$$\ell^* = \overline{\boldsymbol{\alpha}} \left(m, m-1, \ldots, 1\right)^{\mathsf{T}}$$

SimplePlay

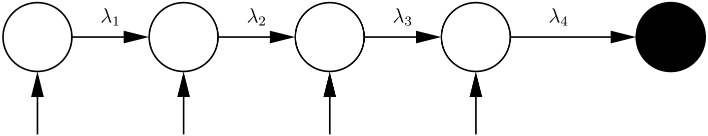


SimplePlay



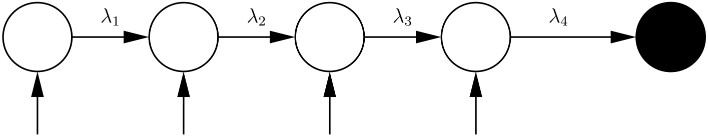
- Bi-diagonal form: Blocks of length 1, no feedbacks

SimplePlay



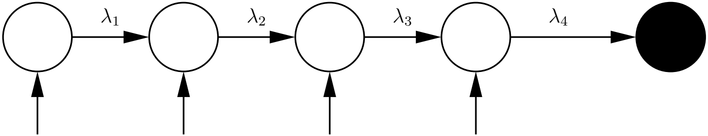
- Bi-diagonal form: Blocks of length 1, no feedbacks
- Draw initial state, then sum up exponential random variates until the absorbing state is reached

SimplePlay

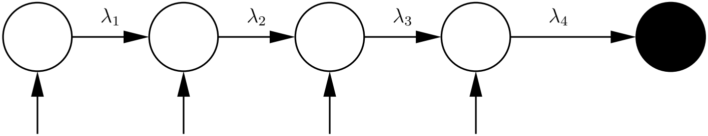


- Bi-diagonal form: Blocks of length 1, no feedbacks
- Draw initial state, then sum up exponential random variates until the absorbing state is reached
- Advantage: No random numbers for state selection required

SimplePlay (ctd.)

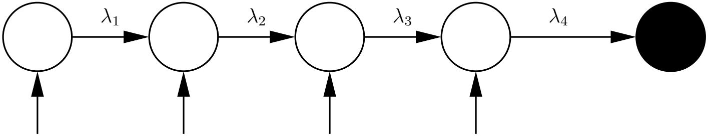


SimplePlay (ctd.)



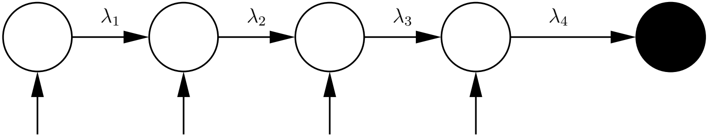
- Worst-Case Costs:
  - $\#uni = 1 + n$
  - $\#ln = n$

SimplePlay (ctd.)



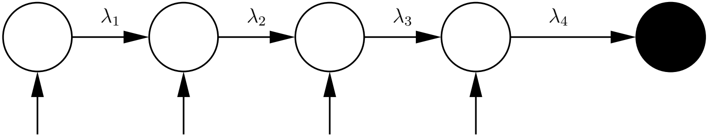
- Worst-Case Costs:
  - $\#uni = 1 + n$
  - $\#ln = n$
- Average Costs:

SimplePlay (ctd.)



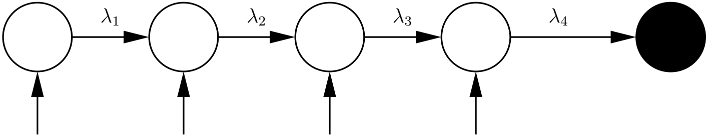
- Worst-Case Costs:
  - $\#uni = 1 + n$
  - $\#ln = n$
- Average Costs:
  - $n^* = \boldsymbol{\alpha}(n, n-1, \ldots, 1)^{\mathsf{T}}$

SimplePlay (ctd.)



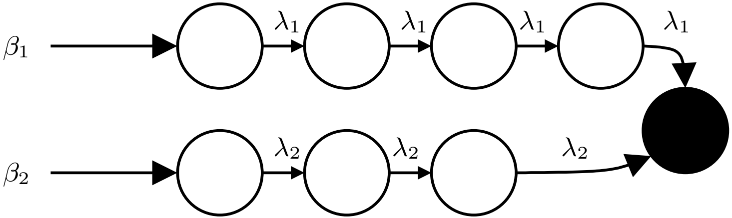
- Worst-Case Costs:
  - $\#uni = 1 + n$
  - $\#ln = n$
- Average Costs:
  - $n^* = \boldsymbol{\alpha}(n, n-1, \ldots, 1)^\mathsf{T}$
  - $\#uni = 1 + n^*$

SimplePlay (ctd.)

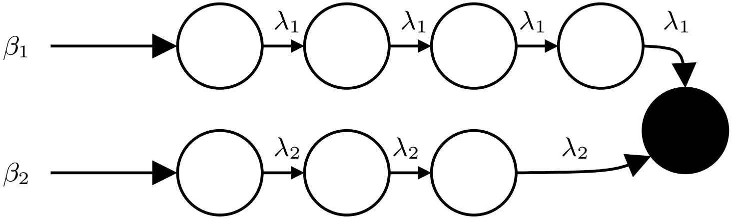


- Worst-Case Costs:
  - $\#uni = 1 + n$
  - $\#ln = n$
- Average Costs:
  - $n^* = \boldsymbol{\alpha}(n, n-1, \ldots, 1)^\mathsf{T}$
  - $\#uni = 1 + n^*$
  - $\#ln = n^*$

SimpleCount

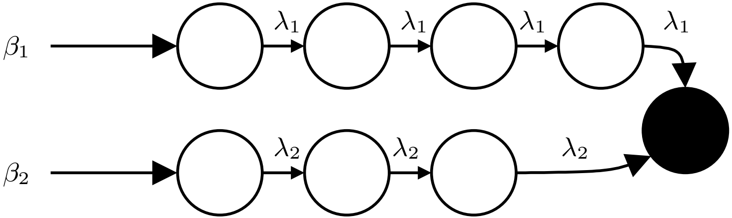


SimpleCount



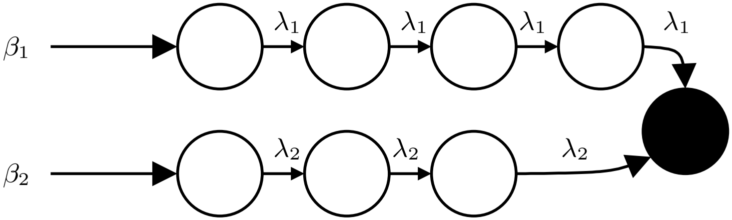
- Hyper-Erlang is a mixture of Erlangs

SimpleCount

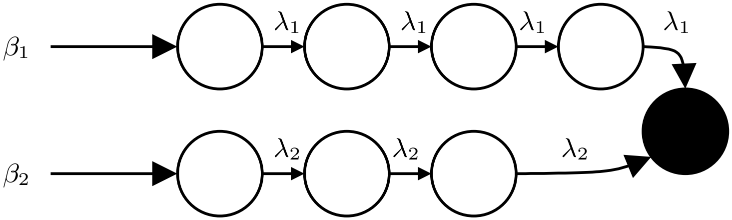


- Hyper-Erlang is a mixture of Erlangs
- Method: Select a branch, draw an Erlang sample

SimpleCount (ctd.)

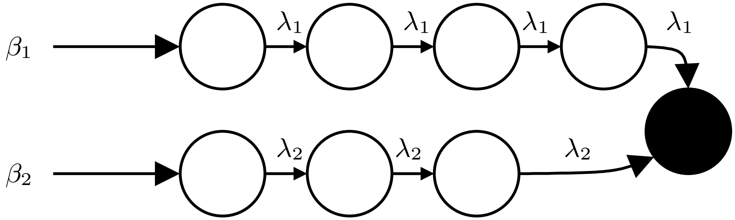


SimpleCount (ctd.)



- Worst-Case Costs:
  - $\#uni = 1 + \max\{b_1, \ldots, b_m\}$
  - $\#ln = 1$

SimpleCount (ctd.)



- Worst-Case Costs:
  - $\#uni = 1 + \max\{b_1, \ldots, b_m\}$
  - $\#ln = 1$
- Average Costs:
  - $n^* = \boldsymbol{\alpha}(b_1, \ldots, b_m)\mathsf{T}$
  - $\#uni = 1 + n^*$
  - $\#ln = 1$

Example: Costs

Example: Costs

- Hyper-Erlang distribution in Hyper-Erlang form:

$$\begin{aligned} \boldsymbol{\alpha} &= (0.1, 0, 0.9, 0, 0, 0) \\ \mathbf{Q} &= \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}. \end{aligned}$$

Example: Costs

- Hyper-Erlang distribution in Hyper-Erlang form:

$$\begin{aligned} \boldsymbol{\alpha} &= (0.1, 0, 0.9, 0, 0, 0) \\ \mathbf{Q} &= \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}. \end{aligned}$$

- Same distribution in CF-1 form:

$$\begin{aligned} \boldsymbol{\alpha}' &= (0.0125, 0.0375, 0.925, 0.025, 0) \\ \mathbf{Q}' &= \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 \\ 0 & 0 & 0 & -2 & 2 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}. \end{aligned}$$

Example: Worst-Case Costs

Method	Worst Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$

Example: Worst-Case Costs

Method	Worst Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$

Example: Worst-Case Costs

Method	Worst Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	7	3	11	5

Example: Worst-Case Costs

Method	Worst Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	7	3	11	5
`Count`	7	5	11	5

Example: Worst-Case Costs

Method	Worst Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	7	3	11	5
`Count`	7	5	11	5
`FE-diagonal`	–	–	8	6

Example: Worst-Case Costs

Method	Worst Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	7	3	11	5
`Count`	7	5	11	5
`FE-diagonal`	–	–	8	6
`SimplePlay`	–	–	6	5

Example: Worst-Case Costs

Method	Worst Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	7	3	11	5
`Count`	7	5	11	5
`FE-diagonal`	–	–	8	6
`SimplePlay`	–	–	6	5
`SimpleCount`	4	1	–	–

Example: Average Costs

Method	Average Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$

Example: Average Costs

Method	Average Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$

Example: Average Costs

Method	Average Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	6.8	2.9	7.075	3.0375

Example: Average Costs

Method	Average Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	6.8	2.9	7.075	3.0375
`Count`	6.8	5	7.075	5

Example: Average Costs

Method	Average Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	6.8	2.9	7.075	3.0375
`Count`	6.8	5	7.075	5
`FE-diagonal`	–	–	5.0875	3.15

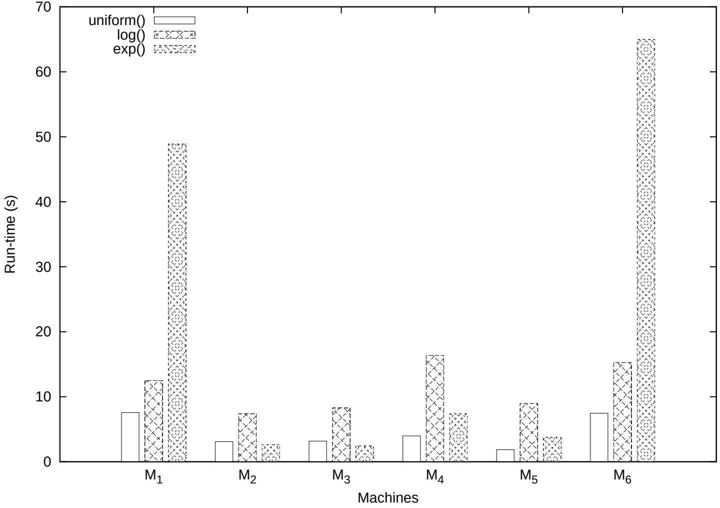
Example: Average Costs

Method	Average Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	6.8	2.9	7.075	3.0375
`Count`	6.8	5	7.075	5
`FE-diagonal`	–	–	5.0875	3.15
`SimplePlay`	–	–	4.0375	3.0375

Example: Average Costs

Method	Average Case			
	$(\boldsymbol{\alpha}, \mathbf{Q})$		$(\boldsymbol{\alpha}', \mathbf{Q}')$	
	$\#uni$	$\#exp$	$\#uni$	$\#exp$
`NumericalInversion`	1	95	1	95
	$\#uni$	$\#ln$	$\#uni$	$\#ln$
`Play`	6.8	2.9	7.075	3.0375
`Count`	6.8	5	7.075	5
`FE-diagonal`	–	–	5.0875	3.15
`SimplePlay`	–	–	4.0375	3.0375
`SimpleCount`	3.9	1	–	–

Computational Costs



Run-time for 10^8 operations on different machines.

Observations

Observations

- Costs differ by method and representation

Observations

- Costs differ by method and representation
- Atomic operations have different costs

Observations

- Costs differ by method and representation
- Atomic operations have different costs
- . . . logarithms are expensive

Observations

- Costs differ by method and representation
- Atomic operations have different costs
- . . . logarithms are expensive
- Optimisation problem: Given a Markovian representation $(\boldsymbol{\alpha}, \mathbf{Q})$, find the (not necessarily minimal) Markovian representation that minimises the costs of random-variate generation

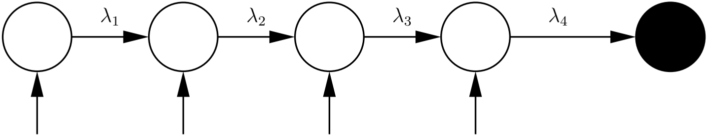
Observations

- Costs differ by method and representation
- Atomic operations have different costs
- ... logarithms are expensive
- Optimisation problem: Given a Markovian representation $(\boldsymbol{\alpha}, \mathbf{Q})$, find the (not necessarily minimal) Markovian representation that minimises the costs of random-variate generation
- Optimisation for bi-diagonal and FE-diagonal forms $\rightarrow$ cover APH and PH

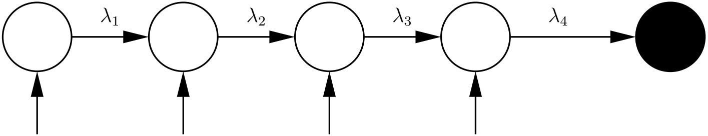
Observations

- Costs differ by method and representation
- Atomic operations have different costs
- ... logarithms are expensive
- Optimisation problem: Given a Markovian representation $(\boldsymbol{\alpha}, \mathbf{Q})$, find the (not necessarily minimal) Markovian representation that minimises the costs of random-variate generation
- Optimisation for bi-diagonal and FE-diagonal forms $\rightarrow$ cover APH and PH
- Focus on number of logarithms

Optimisation for APH

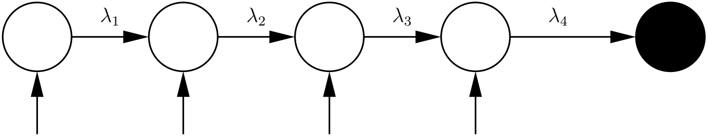


Optimisation for APH



- Every APH has a bi-diagonal representation (the CF-1 form, [6])

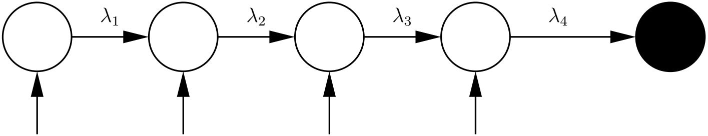
Optimisation for APH



- Every APH has a bi-diagonal representation (the CF-1 form, [6])
- Costs for `SimplePlay`:

$$\begin{aligned} \#uni &= n^* + 1 \\ \#ln &= n^* \end{aligned}$$

Optimisation for APH



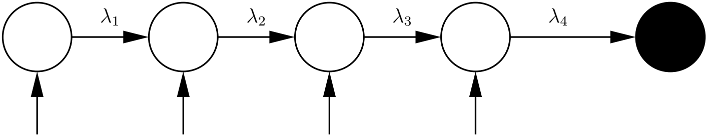
- Every APH has a bi-diagonal representation (the CF-1 form, [6])
- Costs for `SimplePlay`:

$$\begin{aligned} \#uni &= n^* + 1 \\ \#ln &= n^* \end{aligned}$$

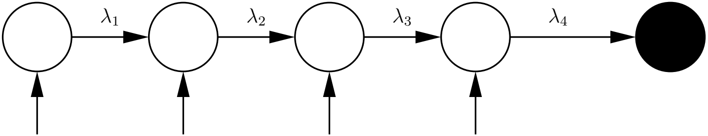
- State-transitions for bi-diagonal representations:

$$n^* = \sum_{i=1}^{n} \alpha_i \cdot (n - i + 1)$$

Optimisation for APH (ctd.)

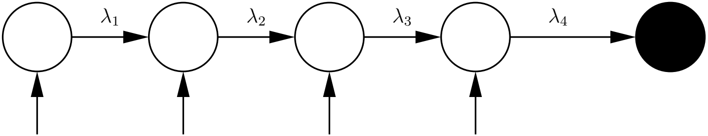


Optimisation for APH (ctd.)



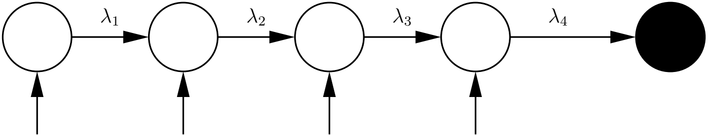
- Idea: Re-order rates along the diagonal – preserves eigenvalues

Optimisation for APH (ctd.)



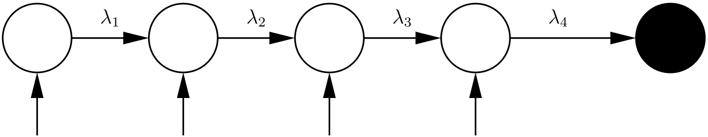
- Idea: Re-order rates along the diagonal – preserves eigenvalues
- Express by a similarity transformation – we keep the same distribution

Optimisation for APH (ctd.)



- Idea: Re-order rates along the diagonal – preserves eigenvalues
- Express by a similarity transformation – we keep the same distribution
- Successive pairwise swappings can construct any ordering (Steinhaus/Johnsohn/Trotter, [10])

Optimisation for APH (ctd.)



- Idea: Re-order rates along the diagonal – preserves eigenvalues
- Express by a similarity transformation – we keep the same distribution
- Successive pairwise swappings can construct any ordering (Steinhaus/Johnsohn/Trotter, [10])
- Check all $n!$ permutations?

The Swap Operator

The Swap Operator

- `Swap`$(i, i+1)$ exchanges the ith, and $(i+1)$th rates

The Swap Operator

- $\texttt{Swap}(i, i+1)$ exchanges the ith, and $(i+1)$th rates
- Similarity transformation:

$$\begin{aligned}\mathbf{Q}' &= \mathbf{S}^{-1}\mathbf{Q}\mathbf{S} \\ \boldsymbol{\alpha}' &= \boldsymbol{\alpha}\mathbf{S}\end{aligned}$$

The Swap Operator

- `Swap`$(i, i+1)$ exchanges the ith, and $(i+1)$th rates
- Similarity transformation:

$$\begin{aligned} \mathbf{Q}' &= \mathbf{S}^{-1}\mathbf{Q}\mathbf{S} \\ \boldsymbol{\alpha}' &= \boldsymbol{\alpha}\mathbf{S} \end{aligned}$$

- Exchange of adjacent rates λ_i, λ_{i+1}:

$$\mathbf{S} = \begin{pmatrix} \ddots & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{\lambda_i - \lambda_{i+1}}{\lambda_i} & \frac{\lambda_{i+1}}{\lambda_i} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \ddots \end{pmatrix}$$

- Local effect on initialisation vector:

$$\begin{aligned}\boldsymbol{\alpha}'_j &= \boldsymbol{\alpha}_j \text{ for } j \neq i, i+1 \\ \boldsymbol{\alpha}'_i &= \boldsymbol{\alpha}_i + \frac{\lambda_i - \lambda_{i+1}}{\lambda_i}\boldsymbol{\alpha}_{i+1} \\ \boldsymbol{\alpha}'_{i+1} &= \frac{\lambda_{i+1}}{\lambda_i}\boldsymbol{\alpha}_{i+1}\end{aligned}$$

- Local effect on initialisation vector:

$$\begin{aligned} \boldsymbol{\alpha}'_j &= \boldsymbol{\alpha}_j \ \text{ for } j \neq i, i+1 \\ \boldsymbol{\alpha}'_i &= \boldsymbol{\alpha}_i + \frac{\lambda_i - \lambda_{i+1}}{\lambda_i}\boldsymbol{\alpha}_{i+1} \\ \boldsymbol{\alpha}'_{i+1} &= \frac{\lambda_{i+1}}{\lambda_i}\boldsymbol{\alpha}_{i+1} \end{aligned}$$

- Effect on the number of traversed states:

$$\begin{aligned} n^{*\prime} &= n^* + \alpha_{i+1}\left(1 - \frac{\lambda_{i+1}}{\lambda_i}\right) \\ n^{*\prime} \leq n^* &\Leftrightarrow \lambda_{i+1} > \lambda_i \end{aligned}$$

- Local effect on initialisation vector:

$$\begin{aligned}
\boldsymbol{\alpha}'_j &= \boldsymbol{\alpha}_j \text{ for } j \neq i, i+1 \\
\boldsymbol{\alpha}'_i &= \boldsymbol{\alpha}_i + \frac{\lambda_i - \lambda_{i+1}}{\lambda_i}\boldsymbol{\alpha}_{i+1} \\
\boldsymbol{\alpha}'_{i+1} &= \frac{\lambda_{i+1}}{\lambda_i}\boldsymbol{\alpha}_{i+1}
\end{aligned}$$

- Effect on the number of traversed states:

$$\begin{aligned}
n^{*\prime} &= n^* + \alpha_{i+1}\left(1 - \frac{\lambda_{i+1}}{\lambda_i}\right) \\
n^{*\prime} \leq n^* &\Leftrightarrow \lambda_{i+1} > \lambda_i
\end{aligned}$$

- $\Rightarrow$ costs can be reduced by moving larger rates to the left

Optimality result for bi-diagonal representations

Theorem ([16])

Given a Markovian representation $(\boldsymbol{\alpha}, \mathbf{Q})$ in CF-1 form, the representation $(\boldsymbol{\alpha}^*, \mathbf{Q}^*)$ that reverses the order of the rates is optimal with respect to n^* if $\boldsymbol{\alpha}^*$ is a stochastic vector. In this case, all bi-diagonal representations constructed by the Swap *operator are Markovian.*

Proof.

Follows from the fact that costs can only be reduced by moving larger rates to the left. □

Caveat: The reversed CF-1 is not always Markovian

Caveat: The reversed CF-1 is not always Markovian

- Consider

$$\begin{aligned}\boldsymbol{\Lambda} &= (1, 2, 3, 4) \\ \boldsymbol{\alpha} &= (0.5, 0.4, 0.05, 0.05)\end{aligned}$$

Caveat: The reversed CF-1 is not always Markovian

- Consider

$$\begin{aligned}\boldsymbol{\Lambda} &= (1,2,3,4)\\ \boldsymbol{\alpha} &= (0.5,0.4,0.05,0.05)\end{aligned}$$

- Reversed CF-1:

$$\begin{aligned}\boldsymbol{\Lambda}' &= (4,3,2,1)\\ \boldsymbol{\alpha}' &= (-0.6,1.4,0,0.2)\end{aligned}$$

Caveat: The reversed CF-1 is not always Markovian

- Consider

$$
\begin{aligned}
\mathbf{\Lambda} &= (1, 2, 3, 4) \\
\boldsymbol{\alpha} &= (0.5, 0.4, 0.05, 0.05)
\end{aligned}
$$

- Reversed CF-1:

$$
\begin{aligned}
\mathbf{\Lambda}' &= (4, 3, 2, 1) \\
\boldsymbol{\alpha}' &= (-0.6, 1.4, 0, 0.2)
\end{aligned}
$$

... not Markovian

Caveat: The reversed CF-1 is not always Markovian

- Consider

$$\begin{aligned}\boldsymbol{\Lambda} &= (1, 2, 3, 4) \\ \boldsymbol{\alpha} &= (0.5, 0.4, 0.05, 0.05)\end{aligned}$$

- Reversed CF-1:

$$\begin{aligned}\boldsymbol{\Lambda}' &= (4, 3, 2, 1) \\ \boldsymbol{\alpha}' &= (-0.6, 1.4, 0, 0.2)\end{aligned}$$

  ... not Markovian
- Optimal Markovian representation:

$$\begin{aligned}\boldsymbol{\Lambda}^* &= (2, 4, 3, 1) \\ \boldsymbol{\alpha}^* &= (0.1, 0.7, 0, 0.2)\end{aligned}$$

Optimisation Algorithms

Optimisation Algorithms

- BubbleSortOptimise:

Optimisation Algorithms

- BubbleSortOptimise:
  - Modified BubbleSort algorithm

Optimisation Algorithms

- BubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort rates in descending order

Optimisation Algorithms

- BubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort rates in descending order
  - Stop if no new Markovian representations can be found (or the reversed CF-1 is reached)

Optimisation Algorithms

- BubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort rates in descending order
  - Stop if no new Markovian representations can be found (or the reversed CF-1 is reached)
- FindMarkovian:

Optimisation Algorithms

- BubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort rates in descending order
  - Stop if no new Markovian representations can be found (or the reversed CF-1 is reached)
- FindMarkovian:
  - Start from reversed CF-1 form

Optimisation Algorithms

- BubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort rates in descending order
  - Stop if no new Markovian representations can be found (or the reversed CF-1 is reached)
- FindMarkovian:
  - Start from reversed CF-1 form
  - Sort rates in ascending order

Optimisation Algorithms

- BubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort rates in descending order
  - Stop if no new Markovian representations can be found (or the reversed CF-1 is reached)
- FindMarkovian:
  - Start from reversed CF-1 form
  - Sort rates in ascending order
  - Stop if a Markovian representation is found

Optimisation: `BubbleSortOptimise`

Algorithm `BubbleSortOptimise`$(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$:

for $i = 1, \ldots, n-1$ **do**
  for $j = 1, \ldots, n-1$ **do**
    $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}') :=$ `Swap`$(\boldsymbol{\alpha}, \boldsymbol{\Lambda}, i)$
    if $\boldsymbol{\Lambda}[j] < \boldsymbol{\Lambda}[j+1] \ \wedge \ \boldsymbol{\alpha}' \geq \mathbf{0}$ **then**
      $(\boldsymbol{\alpha}, \boldsymbol{\Lambda}) := (\boldsymbol{\alpha}', \boldsymbol{\Lambda}')$
    else
      break
    end if
  end for
end for
return $(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$

Optimisation: `FindMarkovian`

Algorithm `FindMarkovian`$(\boldsymbol{\alpha}, \boldsymbol{\Lambda})$:
Let $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}')$ be the reversed CF-1 of $(\boldsymbol{\alpha}, \boldsymbol{\Lambda}')$
while $\neg(\boldsymbol{\alpha}' \geq \mathbf{0})$ **do**
  $i := \mathrm{argmin}_i \{\alpha'_i < 0\}$
  $i := \max\{2, i\}$
  while $\neg(\boldsymbol{\alpha}' \geq \mathbf{0}) \wedge \exists k \ : \ \boldsymbol{\Lambda}[k] \geq \boldsymbol{\Lambda}[k+1]$ **do**
    $k := \mathrm{argmin}_j \{j \mid i-1 \leq j \leq n-1 \wedge \boldsymbol{\Lambda}[j] \geq \boldsymbol{\Lambda}[j+1]\}$
    $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}') := \mathtt{Swap}(\boldsymbol{\alpha}', \boldsymbol{\Lambda}', k)$
  end while
end while
return $(\boldsymbol{\alpha}', \boldsymbol{\Lambda}')$

Optimisation: Examples

Optimisation: Examples

- Generalised Erlang:

Optimisation: Examples

- Generalised Erlang:
  - $\boldsymbol{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$
  - $n^* = 3.49$

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$
  - $n^* = 3.49$
  - Reversed CF-1: $\mathbf{\Lambda}' = (4, 3, 2, 1), \boldsymbol{\alpha}' = (0.46, 0.12, 0.18, 0.24)$

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$
  - $n^* = 3.49$
  - Reversed CF-1: $\mathbf{\Lambda}' = (4, 3, 2, 1), \boldsymbol{\alpha}' = (0.46, 0.12, 0.18, 0.24)$
  - $n^{*\prime} = 2.8$

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$
  - $n^* = 3.49$
  - Reversed CF-1: $\mathbf{\Lambda}' = (4, 3, 2, 1), \boldsymbol{\alpha}' = (0.46, 0.12, 0.18, 0.24)$
  - $n^{*\prime} = 2.8$
- APH with non-Markovian reversed CF-1:

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$
  - $n^* = 3.49$
  - Reversed CF-1: $\mathbf{\Lambda}' = (4, 3, 2, 1), \boldsymbol{\alpha}' = (0.46, 0.12, 0.18, 0.24)$
  - $n^{*\prime} = 2.8$
- APH with non-Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.5, 0.4, 0.05, 0.05)$

Optimisation: Examples

- Generalised Erlang:
  - $\boldsymbol{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\boldsymbol{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$
  - $n^* = 3.49$
  - Reversed CF-1: $\boldsymbol{\Lambda}' = (4, 3, 2, 1), \boldsymbol{\alpha}' = (0.46, 0.12, 0.18, 0.24)$
  - $n^{*\prime} = 2.8$
- APH with non-Markovian reversed CF-1:
  - $\boldsymbol{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.5, 0.4, 0.05, 0.05)$
  - $n^* = 3.35$

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$
  - $n^* = 3.49$
  - Reversed CF-1: $\mathbf{\Lambda}' = (4, 3, 2, 1), \boldsymbol{\alpha}' = (0.46, 0.12, 0.18, 0.24)$
  - $n^{*\prime} = 2.8$
- APH with non-Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.5, 0.4, 0.05, 0.05)$
  - $n^* = 3.35$
  - Reversed CF-1: $\mathbf{\Lambda} = (4, 3, 2, 1), \boldsymbol{\alpha}' = (-0.6, 1.4, 0, 0.2)$

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$
  - $n^* = 3.49$
  - Reversed CF-1: $\mathbf{\Lambda}' = (4, 3, 2, 1), \boldsymbol{\alpha}' = (0.46, 0.12, 0.18, 0.24)$
  - $n^{*\prime} = 2.8$
- APH with non-Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.5, 0.4, 0.05, 0.05)$
  - $n^* = 3.35$
  - Reversed CF-1: $\mathbf{\Lambda} = (4, 3, 2, 1), \boldsymbol{\alpha}' = (-0.6, 1.4, 0, 0.2)$
  - Optimum: $\mathbf{\Lambda}'' = (2, 4, 3, 1), \boldsymbol{\alpha}'' = (0.1, 0.7, 0, 0.2),$

Optimisation: Examples

- Generalised Erlang:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (1, 0, 0, 0)$
  - $n^* = 4$ for every ordering
- APH with Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.7, 0.15, 0.09, 0.06)$
  - $n^* = 3.49$
  - Reversed CF-1: $\mathbf{\Lambda}' = (4, 3, 2, 1), \boldsymbol{\alpha}' = (0.46, 0.12, 0.18, 0.24)$
  - $n^{*\prime} = 2.8$
- APH with non-Markovian reversed CF-1:
  - $\mathbf{\Lambda} = (1, 2, 3, 4), \boldsymbol{\alpha} = (0.5, 0.4, 0.05, 0.05)$
  - $n^* = 3.35$
  - Reversed CF-1: $\mathbf{\Lambda} = (4, 3, 2, 1), \boldsymbol{\alpha}' = (-0.6, 1.4, 0, 0.2)$
  - Optimum: $\mathbf{\Lambda}'' = (2, 4, 3, 1), \boldsymbol{\alpha}'' = (0.1, 0.7, 0, 0.2)$,
  - $n^*(\boldsymbol{\alpha}'', \mathbf{\Lambda}'') = 2.7$

Summary for APH Optimisation

Summary for APH Optimisation

- Optimisation is possible purely by modification of the ordering of the rates

Summary for APH Optimisation

- Optimisation is possible purely by modification of the ordering of the rates
- Moving a larger rate to the left reduces costs

Summary for APH Optimisation

- Optimisation is possible purely by modification of the ordering of the rates
- Moving a larger rate to the left reduces costs
- The reversed CF-1 is optimal *if it is Markovian*.

Summary for APH Optimisation

- Optimisation is possible purely by modification of the ordering of the rates
- Moving a larger rate to the left reduces costs
- The reversed CF-1 is optimal *if it is Markovian*.
- Efficient optimisation algorithms

Summary for APH Optimisation

- Optimisation is possible purely by modification of the ordering of the rates
- Moving a larger rate to the left reduces costs
- The reversed CF-1 is optimal *if it is Markovian*.
- Efficient optimisation algorithms
- Only valid for APH $\rightarrow$ can we extend it to PH?

Optimisation for general PH

Optimisation for general PH

- Use the FE-diagonal form

Optimisation for general PH

- Use the FE-diagonal form
  - Every PH has an FE-diagonal representation (the Monocyclic form, [11])

Optimisation for general PH

- Use the FE-diagonal form
  - Every PH has an FE-diagonal representation (the Monocyclic form, [11])
  - Elegant expression for the number of logarithms

Optimisation for general PH

- Use the FE-diagonal form
  - Every PH has an FE-diagonal representation (the Monocyclic form, [11])
  - Elegant expression for the number of logarithms
- Costs for FE-diagonal representations:

$$\#ln \quad = \quad 3\ell^*$$

Optimisation for general PH

- Use the FE-diagonal form
  - Every PH has an FE-diagonal representation (the Monocyclic form, [11])
  - Elegant expression for the number of logarithms
- Costs for FE-diagonal representations:

$$\#ln \quad = \quad 3\ell^*$$

- Block visits for FE-diagonal representations:

$$\ell^* = \sum_{i=1}^{n} \alpha_i \cdot (m - i + 1)$$

Optimisation for general PH

- Use the FE-diagonal form
  - Every PH has an FE-diagonal representation (the Monocyclic form, [11])
  - Elegant expression for the number of logarithms
- Costs for FE-diagonal representations:

$$\#ln \quad = \quad 3\ell^*$$

- Block visits for FE-diagonal representations:

$$\ell^* = \sum_{i=1}^{n} \alpha_i \cdot (m - i + 1)$$

- Idea: Re-order blocks along the diagonal – preserves eigenvalues

Optimisation for general PH

- Use the FE-diagonal form
  - Every PH has an FE-diagonal representation (the Monocyclic form, [11])
  - Elegant expression for the number of logarithms
- Costs for FE-diagonal representations:

$$\#ln \quad = \quad 3\ell^*$$

- Block visits for FE-diagonal representations:

$$\ell^* = \sum_{i=1}^{n} \alpha_i \cdot (m - i + 1)$$

- Idea: Re-order blocks along the diagonal – preserves eigenvalues
- Express by a similarity transformation

Optimisation for general PH

- Use the FE-diagonal form
  - Every PH has an FE-diagonal representation (the Monocyclic form, [11])
  - Elegant expression for the number of logarithms
- Costs for FE-diagonal representations:

$$\#ln \quad = \quad 3\ell^*$$

- Block visits for FE-diagonal representations:

$$\ell^* = \sum_{i=1}^{n} \alpha_i \cdot (m - i + 1)$$

- Idea: Re-order blocks along the diagonal – preserves eigenvalues
- Express by a similarity transformation
- Successive pairwise swappings can construct any ordering

The GSwap Operator

The `GSwap` Operator

- `GSwap`$(i, i+1)$ exchanges the ith and $(i+1)$th FE-blocks along the diagonal

The `GSwap` Operator

- `GSwap`$(i, i+1)$ exchanges the ith and $(i+1)$th FE-blocks along the diagonal
- Similarity Transformation:

$$\mathbf{S} = \begin{pmatrix} \mathbf{I}_{\nu\times\nu} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{S}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{\mu\times\mu} \end{pmatrix},$$

The GSwap Operator

- $\texttt{GSwap}(i, i+1)$ exchanges the ith and $(i+1)$th FE-blocks along the diagonal
- Similarity Transformation:

$$\mathbf{S} \;=\; \begin{pmatrix} \mathbf{I}_{\nu\times\nu} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{S}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{\mu\times\mu} \end{pmatrix},$$

- $\hat{\mathbf{S}}$ is block-lower-triangular ... but does not have a nice, general explicit structure

The GSwap Operator

- GSwap$(i, i+1)$ exchanges the ith and $(i+1)$th FE-blocks along the diagonal
- Similarity Transformation:

$$\mathbf{S} = \begin{pmatrix} \mathbf{I}_{\nu\times\nu} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{S}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I}_{\mu\times\mu} \end{pmatrix},$$

- $\hat{\mathbf{S}}$ is block-lower-triangular … but does not have a nice, general explicit structure
- $\hat{\mathbf{S}}$ needs to be computed for each possible swap as the solution of

$$\begin{pmatrix} \mathbf{F}_i & -\mathbf{F}_i \mathbb{1}\mathbf{e}_1 \\ \mathbf{0} & \mathbf{F}_{i+1} \end{pmatrix} \hat{\mathbf{S}} = \hat{\mathbf{S}} \begin{pmatrix} \mathbf{F}_{i+1} & -\mathbf{F}_{i+1} \mathbb{1}\mathbf{e}_1 \\ \mathbf{0} & \mathbf{F}_i \end{pmatrix}$$

$$\hat{\mathbf{S}}\mathbb{1} = \mathbb{1}.$$

Conjecture

The optimal ordering is achieved by computing the reversed Monocyclic form.

Counterexample

Counterexample

- Consider

$$\begin{aligned} \boldsymbol{\Upsilon} &= ((1, 0.1, 0), (3, 1.5, 0.5), (3, 1, 0)) \\ \boldsymbol{\Upsilon}' &= ((1, 0.1, 0), (3, 1, 0), (3, 1, 0.5)) \end{aligned}$$

Counterexample

- Consider

$$\begin{aligned} \Upsilon &= ((1, 0.1, 0), (3, 1.5, 0.5), (3, 1, 0)) \\ \Upsilon' &= ((1, 0.1, 0), (3, 1, 0), (3, 1, 0.5)) \end{aligned}$$

- Consider two initial vectors:

Counterexample

- Consider

$$\begin{aligned}\mathbf{\Upsilon} &= ((1, 0.1, 0), (3, 1.5, 0.5), (3, 1, 0)) \\ \mathbf{\Upsilon}' &= ((1, 0.1, 0), (3, 1, 0), (3, 1, 0.5))\end{aligned}$$

- Consider two initial vectors:

$$\boldsymbol{\alpha}_1 = (0.09 \mid 0.1, 0.3, 0.31 \mid 0.1, 0.1, 0)$$

Counterexample

- Consider

$$\begin{aligned}\boldsymbol{\Upsilon} &= ((1, 0.1, 0), (3, 1.5, 0.5), (3, 1, 0)) \\ \boldsymbol{\Upsilon}' &= ((1, 0.1, 0), (3, 1, 0), (3, 1, 0.5))\end{aligned}$$

- Consider two initial vectors:

$$\begin{aligned}\boldsymbol{\alpha}_1 &= (0.09 \mid 0.1, 0.3, 0.31 \mid 0.1, 0.1, 0) \\ \boldsymbol{\alpha}_2 &= (0.09 \mid 0.1, 0.3, 0.31 \mid 0.2, 0, 0)\end{aligned}$$

Counterexample

- Consider

$$\begin{aligned} \boldsymbol{\Upsilon} &= ((1, 0.1, 0), (3, 1.5, 0.5), (3, 1, 0)) \\ \boldsymbol{\Upsilon}' &= ((1, 0.1, 0), (3, 1, 0), (3, 1, 0.5)) \end{aligned}$$

- Consider two initial vectors:

$$\begin{aligned} \boldsymbol{\alpha}_1 &= (0.09 \mid 0.1, 0.3, 0.31 \mid 0.1, 0.1, 0) \\ \boldsymbol{\alpha}_2 &= (0.09 \mid 0.1, 0.3, 0.31 \mid 0.2, 0, 0) \end{aligned}$$

- Costs:

$$\ell_1^* = \ell_2^* = 1.89$$

Counterexample (ctd.)

Counterexample (ctd.)

- Initial vectors after swapping:

Counterexample (ctd.)

- Initial vectors after swapping:

$$\boldsymbol{\alpha}_1' \quad = (0.09 \quad | \; 0.141852, 0.289630.271111$$

Counterexample (ctd.)

- Initial vectors after swapping:

$$\begin{aligned} \boldsymbol{\alpha}'_1 &= (0.09 && \mid 0.141852, 0.289630.271111 \\ & && \mid 0.118519, 0.0888889, 0) \\ \boldsymbol{\alpha}'_2 &= (0.09 && \mid 0.0492593, 0.426667, 0.315556 \\ & && \mid 0.118519, 0, 0) \end{aligned}$$

Counterexample (ctd.)

- Initial vectors after swapping:

$$
\begin{aligned}
\boldsymbol{\alpha}'_1 &= (0.09 & &\mid 0.141852, 0.289630.271111 \\
& & &\mid 0.118519, 0.0888889, 0) \\
\boldsymbol{\alpha}'_2 &= (0.09 & &\mid 0.0492593, 0.426667, 0.315556 \\
& & &\mid 0.118519, 0, 0)
\end{aligned}
$$

- Costs:

Counterexample (ctd.)

- Initial vectors after swapping:

$$\begin{aligned} \boldsymbol{\alpha}'_1 &= (0.09 && \mid 0.141852, 0.289630.271111 \\ & && \mid 0.118519, 0.0888889, 0) \\ \boldsymbol{\alpha}'_2 &= (0.09 && \mid 0.0492593, 0.426667, 0.315556 \\ & && \mid 0.118519, 0, 0) \end{aligned}$$

- Costs:

$$\ell^{*\prime}_1 \quad = \quad 1.8825939 < \ell^*_1 = 1.89$$

Counterexample (ctd.)

- Initial vectors after swapping:

$$\begin{aligned}\boldsymbol{\alpha}'_1 &= (0.09 & | \; 0.141852, 0.289630.271111 \\ & & | \; 0.118519, 0.0888889, 0) \\ \boldsymbol{\alpha}'_2 &= (0.09 & | \; 0.0492593, 0.426667, 0.315556 \\ & & | \; 0.118519, 0, 0)\end{aligned}$$

- Costs:

$$\begin{aligned}{\ell^*}'_1 &= 1.8825939 < \ell^*_1 = 1.89 \\ {\ell^*}'_2 &= 1.9714836 > \ell^*_2 = 1.89\end{aligned}$$

Counterexample (ctd.)

- Initial vectors after swapping:

$$
\begin{aligned}
\boldsymbol{\alpha}'_1 &= (0.09 & | \, 0.141852, 0.289630.271111 \\
& & | \, 0.118519, 0.0888889, 0) \\
\boldsymbol{\alpha}'_2 &= (0.09 & | \, 0.0492593, 0.426667, 0.315556 \\
& & | \, 0.118519, 0, 0)
\end{aligned}
$$

- Costs:

$$
\begin{aligned}
{\ell^*}'_1 &= 1.8825939 < \ell^*_1 = 1.89 \\
{\ell^*}'_2 &= 1.9714836 > \ell^*_2 = 1.89
\end{aligned}
$$

- $\Rightarrow$ Effect of the swap depends on the initialisation vector

Optimisation Algorithms

Optimisation Algorithms

- GBubbleSortOptimise:

Optimisation Algorithms

- GBubbleSortOptimise:
  - Modified BubbleSort algorithm

Optimisation Algorithms

- GBubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort blocks in descending order

Optimisation Algorithms

- GBubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort blocks in descending order
  - Stop if no Markovian representations can be found (or the reversed CF-1 is reached)

Optimisation Algorithms

- GBubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort blocks in descending order
  - Stop if no Markovian representations can be found (or the reversed CF-1 is reached)
- GFindMarkovian:

Optimisation Algorithms

- GBubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort blocks in descending order
  - Stop if no Markovian representations can be found (or the reversed CF-1 is reached)
- GFindMarkovian:
  - Start from reversed Monocyclic form

Optimisation Algorithms

- GBubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort blocks in descending order
  - Stop if no Markovian representations can be found (or the reversed CF-1 is reached)
- GFindMarkovian:
  - Start from reversed Monocyclic form
  - Sort blocks in ascending order

Optimisation Algorithms

- GBubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort blocks in descending order
  - Stop if no Markovian representations can be found (or the reversed CF-1 is reached)
- GFindMarkovian:
  - Start from reversed Monocyclic form
  - Sort blocks in ascending order
  - Stop if a Markovian representation is found

Optimisation Algorithms

- GBubbleSortOptimise:
  - Modified BubbleSort algorithm
  - Sort blocks in descending order
  - Stop if no Markovian representations can be found (or the reversed CF-1 is reached)
- GFindMarkovian:
  - Start from reversed Monocyclic form
  - Sort blocks in ascending order
  - Stop if a Markovian representation is found
- Order determined by a heuristic

Optimisation: BubbleSortOptimise

```
Algorithm GBubbleSortOptimise(α, ϒ):
for i = 1, ..., m − 1 do
   for j = 1, ..., m − 1 do
      (α′, ϒ′) := Swap(α, ϒ, i)
      if ComparisonHeuristic(α, ϒ, j) = true ∧ α′ ≥ 0 then
         (α, ϒ) := (α′, ϒ′)
      else
         break
      end if
   end for
end for
return (α, ϒ)
```

Optimisation: FindMarkovian

Let $(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}')$ be the reversed Monocyclic form of $(\boldsymbol{\alpha}, \boldsymbol{\Upsilon}')$
r:=0
while $\neg(\boldsymbol{\alpha}' \geq \mathbf{0})$ **do**
  $i := \text{argmin}_i \{\alpha_i' < 0\}$
  $i := \max\{2, i\}$
  while $\neg(\boldsymbol{\alpha}' \geq \mathbf{0}) \wedge \exists k$:
  ComparisonHeuristic$(\boldsymbol{\Upsilon}[k], \boldsymbol{\Upsilon}[k+1])$ = false **do**
    $k := \text{argmin}_j \{j \mid i-1 \leq j \leq m-1 \wedge \boldsymbol{\Upsilon}[j] \geq \boldsymbol{\Upsilon}[j+1]\}$
    $(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}') :=$ Swap$(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}', k)$
    if $(\boldsymbol{\alpha}', \boldsymbol{\Upsilon}')$ is a new representation **then**
      $r++$
    end if
    if $r = m!$ **then**
      goto END
    end if
  end while

Swapping Criteria for APH

Swapping Criteria for APH

- Assume blocks of length 1 (bi-diagonal case)

Swapping Criteria for APH

- Assume blocks of length 1 (bi-diagonal case)
- We swap blocks $i, i+1$ if $\lambda_i < \lambda_{i+1}$

Swapping Criteria for APH

- Assume blocks of length 1 (bi-diagonal case)
- We swap blocks $i, i+1$ if $\lambda_i < \lambda_{i+1}$
- Equivalent to

Swapping Criteria for APH

- Assume blocks of length 1 (bi-diagonal case)
- We swap blocks $i, i+1$ if $\lambda_i < \lambda_{i+1}$
- Equivalent to
  - Block i has dominant eigenvalue of smaller magnitude than block $i+1$:
$$|r_i| < |r_{i+1}| \Leftrightarrow \lambda_i < \lambda_{i+1}$$

Swapping Criteria for APH

- Assume blocks of length 1 (bi-diagonal case)
- We swap blocks $i, i+1$ if $\lambda_i < \lambda_{i+1}$
- Equivalent to
  - Block i has dominant eigenvalue of smaller magnitude than block $i+1$:

$$|r_i| < |r_{i+1}| \Leftrightarrow \lambda_i < \lambda_{i+1}$$

  - Block i has larger mean than block $i+1$:

$$M_i > M_{i+1} \Leftrightarrow \frac{1}{\lambda_i} > \frac{1}{\lambda_{i+1}} \Leftrightarrow \lambda_i < \lambda_{i+1}$$

Swapping Criteria for APH

- Assume blocks of length 1 (bi-diagonal case)
- We swap blocks $i, i+1$ if $\lambda_i < \lambda_{i+1}$
- Equivalent to
  - Block i has dominant eigenvalue of smaller magnitude than block $i+1$:

$$|r_i| < |r_{i+1}| \Leftrightarrow \lambda_i < \lambda_{i+1}$$

  - Block i has larger mean than block $i+1$:

$$M_i > M_{i+1} \Leftrightarrow \frac{1}{\lambda_i} > \frac{1}{\lambda_{i+1}} \Leftrightarrow \lambda_i < \lambda_{i+1}$$

  - Block i has smaller exit-rate:

$$\lambda_i < \lambda_{i+1}$$

Swapping Criteria for APH

- Assume blocks of length 1 (bi-diagonal case)
- We swap blocks $i, i+1$ if $\lambda_i < \lambda_{i+1}$
- Equivalent to
  - Block i has dominant eigenvalue of smaller magnitude than block $i+1$:
  $$|r_i| < |r_{i+1}| \Leftrightarrow \lambda_i < \lambda_{i+1}$$
  - Block i has larger mean than block $i+1$:
  $$M_i > M_{i+1} \Leftrightarrow \frac{1}{\lambda_i} > \frac{1}{\lambda_{i+1}} \Leftrightarrow \lambda_i < \lambda_{i+1}$$
  - Block i has smaller exit-rate:
  $$\lambda_i < \lambda_{i+1}$$
  - The determinant of the transformation matrix is larger than 1:
  $$\left|\hat{\mathbf{S}}\right| = \frac{\lambda_{i+1}}{\lambda_i} > 1$$

Swapping Criteria for PH

Swapping Criteria for PH

- Criteria are different for the FE-diagonal case:

Swapping Criteria for PH

- Criteria are different for the FE-diagonal case:
  - Eigenvalues:

$$\left|-\left(1 - z_i^{\frac{1}{b_i}}\right)\right| < \left|-\left(1 - z_{i+1}^{\frac{1}{b_{i+1}}}\right)\right|$$

Swapping Criteria for PH

- Criteria are different for the FE-diagonal case:
  - Eigenvalues:

$$\left| -\left(1 - z_i^{\frac{1}{b_i}}\right) \right| < \left| -\left(1 - z_{i+1}^{\frac{1}{b_{i+1}}}\right) \right|$$

  - Means:

$$\begin{aligned} \text{Start at the first state: } \hat{M}_i &= \mathbf{e}_1(-\mathbf{F}_i)^{-1}\mathbb{1} \\ \text{Start at all states: } M_i &= \frac{\boldsymbol{\alpha}_i}{\boldsymbol{\alpha}_i\mathbb{1}}(-\mathbf{F}_i)^{-1}\mathbb{1} \end{aligned}$$

Swapping Criteria for PH

- Criteria are different for the FE-diagonal case:
  - Eigenvalues:

$$\left| -\left(1 - z_i^{\frac{1}{b_i}}\right) \right| < \left| -\left(1 - z_{i+1}^{\frac{1}{b_{i+1}}}\right) \right|$$

  - Means:

$$\begin{aligned} \text{Start at the first state: } \hat{M}_i &= \mathbf{e}_1(-\mathbf{F}_i)^{-1}\mathbb{1} \\ \text{Start at all states: } M_i &= \frac{\boldsymbol{\alpha}_i}{\boldsymbol{\alpha}_i\mathbb{1}}(-\mathbf{F}_i)^{-1}\mathbb{1} \end{aligned}$$

  - Exit-rates:

$$(1 - z_i)\lambda_i < (1 - z_{i+1})\lambda_{i+1}$$

Swapping Criteria for PH

- Criteria are different for the FE-diagonal case:
  - Eigenvalues:

$$\left| -\left(1 - z_i^{\frac{1}{b_i}}\right) \right| < \left| -\left(1 - z_{i+1}^{\frac{1}{b_{i+1}}}\right) \right|$$

  - Means:

$$\begin{aligned} \text{Start at the first state: } \hat{M}_i &= \mathbf{e}_1(-\mathbf{F}_i)^{-1}\mathbb{1} \\ \text{Start at all states: } M_i &= \frac{\boldsymbol{\alpha}_i}{\boldsymbol{\alpha}_i \mathbb{1}}(-\mathbf{F}_i)^{-1}\mathbb{1} \end{aligned}$$

  - Exit-rates:

$$(1 - z_i)\lambda_i < (1 - z_{i+1})\lambda_{i+1}$$

  - Determinant:

$$\left| \hat{\mathbf{S}} \right| > 1$$

Heuristics are not perfect

				Correct?	
	$\mathbf{F}_1$	$\mathbf{F}_2$	Swap?	$\boldsymbol{\alpha}_1$	$\boldsymbol{\alpha}_2$

Heuristics are not perfect

				Correct?	
	$\mathbf{F}_1$	$\mathbf{F}_2$	Swap?	$\boldsymbol{\alpha}_1$	$\boldsymbol{\alpha}_2$
Eigenvalue	-0.3095	-1	yes	✓	✗

Heuristics are not perfect

	$\mathbf{F}_1$	$\mathbf{F}_2$	Swap?	Correct? $\boldsymbol{\alpha}_1$	$\boldsymbol{\alpha}_2$
Eigenvalue	-0.3095	-1	yes	✓	✗
Mean (first state)	4	3	yes	✓	✗
Mean (all states, $\boldsymbol{\alpha}_1$)	4	1.7042	yes	✓	✗
Mean (all states, $\boldsymbol{\alpha}_2$)	2.5	1.7042	yes	✓	✗

Heuristics are not perfect

	$\mathbf{F}_1$	$\mathbf{F}_2$	Swap?	Correct? $\boldsymbol{\alpha}_1$	$\boldsymbol{\alpha}_2$
Eigenvalue	-0.3095	-1	yes	✓	✗
Mean (first state)	4	3	yes	✓	✗
Mean (all states, $\boldsymbol{\alpha}_1$)	4	1.7042	yes	✓	✗
Mean (all states, $\boldsymbol{\alpha}_2$)	2.5	1.7042	yes	✓	✗
Exit rate	0.75	1	yes	✓	✗

Heuristics are not perfect

				Correct?	
	$\mathbf{F}_1$	$\mathbf{F}_2$	Swap?	$\boldsymbol{\alpha}_1$	$\boldsymbol{\alpha}_2$
Eigenvalue	-0.3095	-1	yes	✓	✗
Mean (first state)	4	3	yes	✓	✗
Mean (all states, $\boldsymbol{\alpha}_1$)	4	1.7042	yes	✓	✗
Mean (all states, $\boldsymbol{\alpha}_2$)	2.5	1.7042	yes	✓	✗
Exit rate	0.75	1	yes	✓	✗
Determinant	0.208		no	✗	✓

Example

Example

- Generate 100 random PH distributions

Example

- Generate 100 random PH distributions
- Compute Monocyclic form

Example

- Generate 100 random PH distributions
- Compute Monocyclic form
- Apply exhaustive search for the optimum

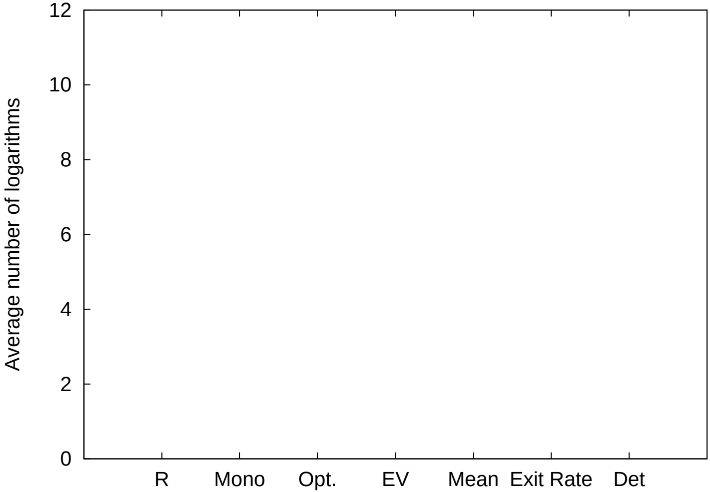
Example

- Generate 100 random PH distributions
- Compute Monocyclic form
- Apply exhaustive search for the optimum
- Apply heuristics in BubbleSort algorithm

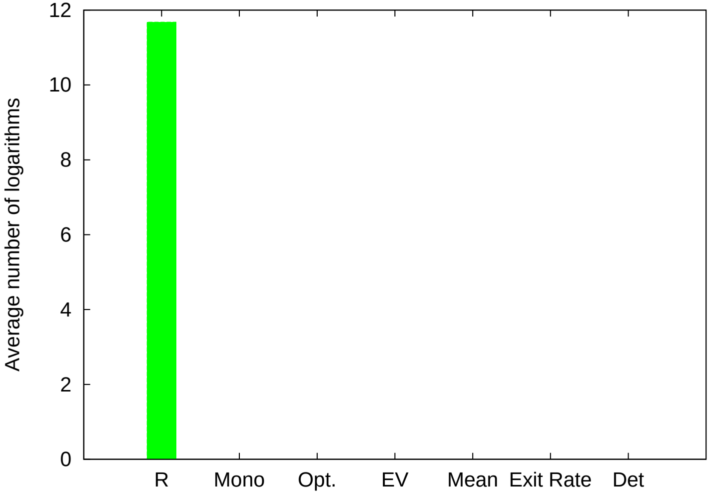
Example

- Generate 100 random PH distributions
- Compute Monocyclic form
- Apply exhaustive search for the optimum
- Apply heuristics in BubbleSort algorithm
- Results shown here: $n = 6$

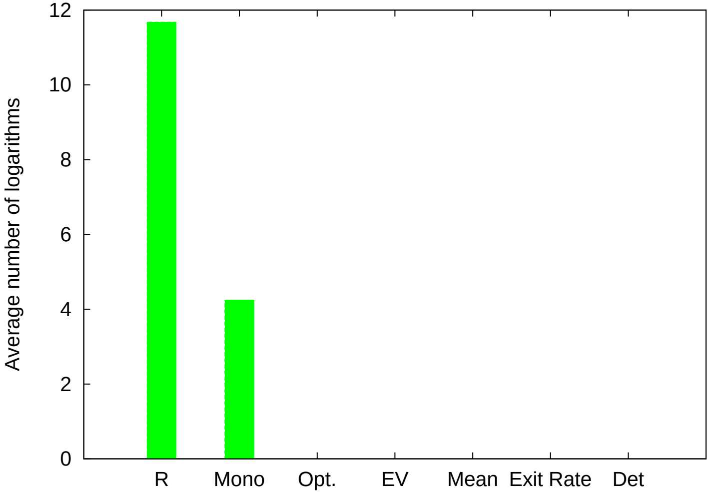
Some Empirical Results



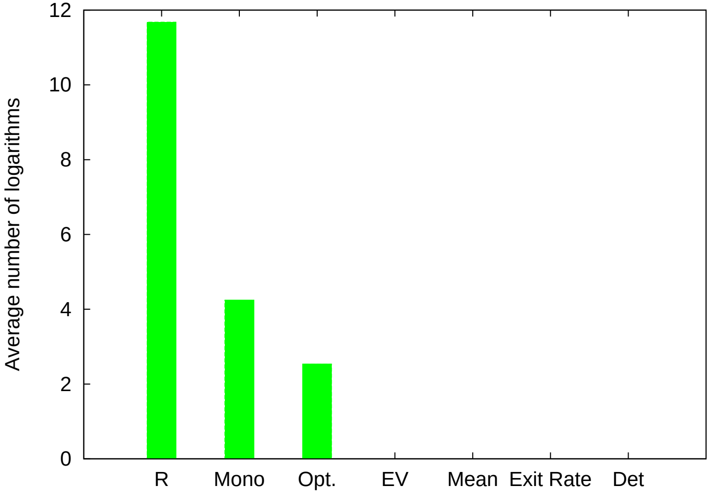
Some Empirical Results



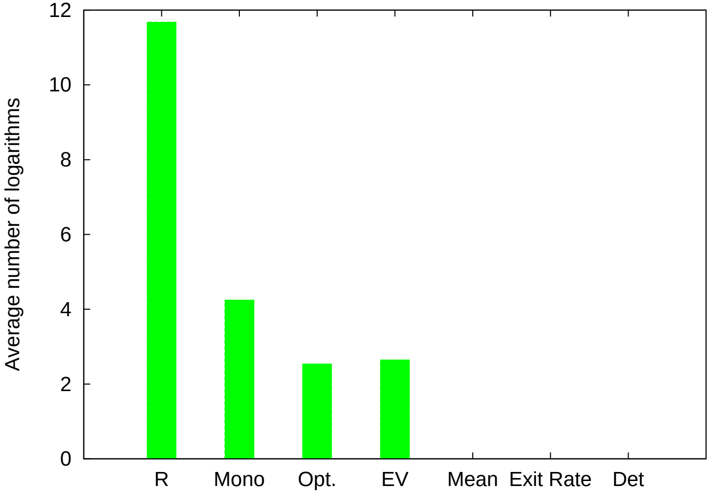
Some Empirical Results



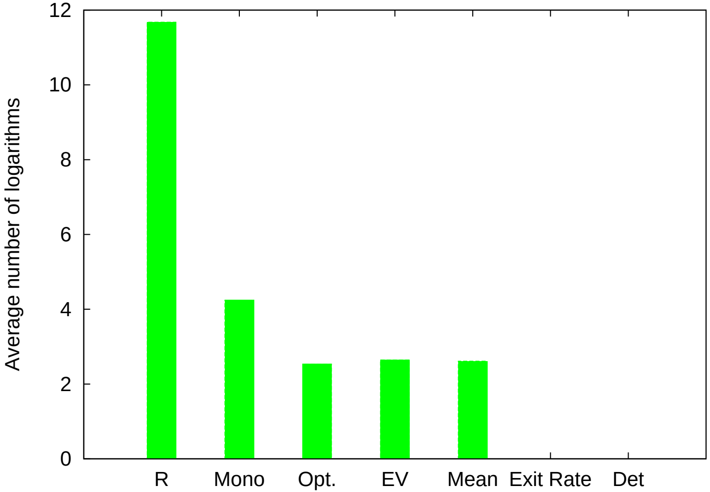
Some Empirical Results



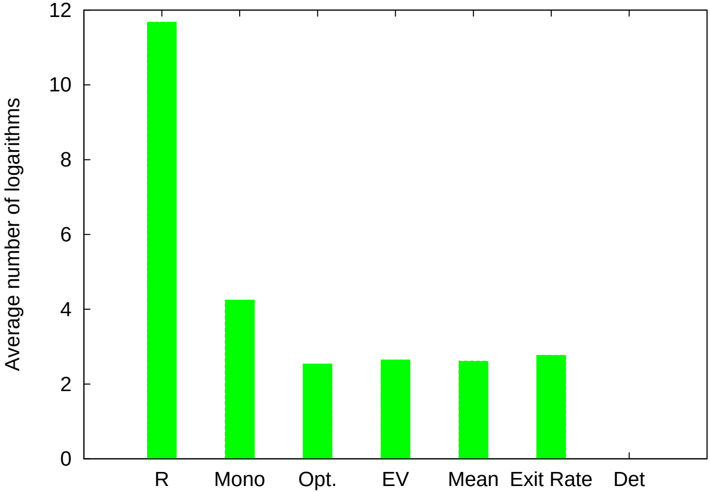
Some Empirical Results



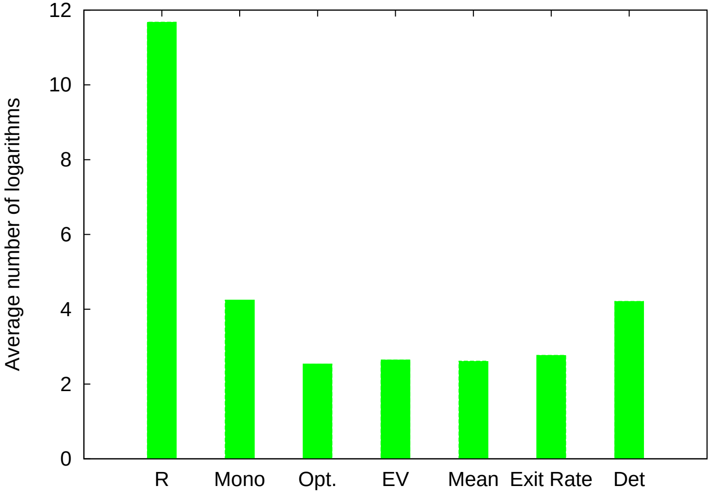
Some Empirical Results



Some Empirical Results



Some Empirical Results



Summary

Summary

- Efficiency of random-variate generation depends on

Summary

- Efficiency of random-variate generation depends on
  - Representation

Summary

- Efficiency of random-variate generation depends on
  - Representation
  - Algorithm

Summary

- Efficiency of random-variate generation depends on
  - Representation
  - Algorithm
- Canonical representations are efficient and allow optimisation

Summary

- Efficiency of random-variate generation depends on
  - Representation
  - Algorithm
- Canonical representations are efficient and allow optimisation
- Optimisation of canonical representations:

Summary

- Efficiency of random-variate generation depends on
  - Representation
  - Algorithm
- Canonical representations are efficient and allow optimisation
- Optimisation of canonical representations:
  - General optimum for APH

Summary

- Efficiency of random-variate generation depends on
  - Representation
  - Algorithm
- Canonical representations are efficient and allow optimisation
- Optimisation of canonical representations:
  - General optimum for APH
  - No general optimum for PH, but heuristics exist

fin.

D. Aldous and L. Shepp.
The least variable phase-type distribution is erlang.
Stochastic Models, 3:467–473, 1987.

S. Asmussen, O. Nerman, and M. Olsson.
Fitting Phase-Type Distribution Via the EM Algorithm.
Scand. J. Statist., 23:419–441, 1996.

B. Blywis, M. Günes, F. Juraschek, O. Hahm, and N. Schmittberger.
Properties and Topology of the DES-Testbed (2nd Extended Revision).
Technical Report TR-B-11-04, Freie Universität Berlin, July 2011.

E. F. Brown.

A distribution-free random number generator via a matrix-exponential representation.
In *Proceedings of the 1992 ACM/SIGAPP symposium on Applied computing: technological challenges of the 1990's*, SAC '92, pages 960–969, New York, NY, USA, 1992. ACM.

G. Casale, E. Z. Zhang, and E. Smirni.
Kpc-toolbox: Simple yet effective trace fitting using markovian arrival processes.
In *Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems*, pages 83–92, Washington, DC, USA, 2008. IEEE Computer Society.

A. Cumani.
On the Canonical Representation of Homogeneous Markov Processes Modelling Failure-time Distributions.
Microelectronics and Reliability, 22:583–602, 1982.

A. Horváth, S. Rácz, and M. Telek.
Moments characterization of order 3 matrix exponential distributions.
In *ASMTA '09: Proceedings of the 16th International Conference on Analytical and Stochastic Modeling Techniques and Applications*, pages 174–188, Berlin, Heidelberg, 2009. Springer-Verlag.

A. Horváth and M. Telek.
PhFit: A General Phase-Type Fitting Tool.
In *TOOLS '02: Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, pages 82–91, London, UK, 2002. Springer-Verlag.

G. Horváth and M. Telek.

Acceptance-rejection methods for generating random variates from matrix exponential distributions and rational arrival processes.
In *Int. Conf. on Martix Analytic Methods (MAM)*, New York, New York, USA, june 2011.

S. M. Johnson.
Generation of Permutations by Adjacent Transposition.
Mathematics of Computation, 17(83):282–285, July 1963.

S. Mocanu and C. Commault.
Sparse Representations of Phase-type Distributions.
Commun. Stat., Stochastic Models, 15(4):759 – 778, 1999.

C. Moler and C. V. Loan.
Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later.

SIAM Review, 45(1):3–49, 2003.

M. F. Neuts.
Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach.
Dover Publications, Inc., New York, 1981.

M. F. Neuts and M. E. Pagano.
Generating random variates from a distribution of phase type.
In *WSC '81: Proceedings of the 13th Winter Simulation Conference*, pages 381–387, Piscataway, NJ, USA, 1981. IEEE Press.

P. Reinecke, T. Krau, K. Wolter, P. Reinecke, T. Krauß, and K. Wolter.
Cluster-based fitting of phase-type distributions to empirical data.

Computers & Mathematics with Applications, (0):–, 2012.
To appear.

P. Reinecke, M. Telek, and K. Wolter.
Reducing the Costs of Generating APH-Distributed Random Numbers.
In B. Müller-Clostermann, K. Echtle, and E. Rathgeb, editors, *MMB & DFT 2010*, number 5987 in LNCS, pages 274–286. Springer-Verlag Berlin Heidelberg, 2010.

A. Riska, V. Diev, and E. Smirni.
Efficient fitting of long-tailed data sets into phase-type distributions.
SIGMETRICS Perform. Eval. Rev., 30:6–8, December 2002.

R. Sadre and B. Haverkort.

Fitting heavy-tailed http traces with the new stratified em-algorithm.
In *4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks (IT-NEWS)*, pages 254–261, Los Alamitos, February 2008. IEEE Computer Society Press.

M. Telek and A. Heindl.
Matching Moments for Acyclic Discrete and Continous Phase-Type Distributions of Second Order.
International Journal of Simulation Systems, Science & Technology, 3(3–4):47–57, Dec. 2002.

A. Thümmler, P. Buchholz, and M. Telek.
A Novel Approach for Phase-Type Fitting with the EM Algorithm.
IEEE Trans. Dependable Secur. Comput., 3(3):245–258, 2006.

J. Wang, J. Liu, and C. She.
Segment-based adaptive hyper-erlang model forlong-tailed network traffic approximation.
The Journal of Supercomputing, 45:296–312, 2008.
10.1007/s11227-008-0173-5.

J. Wang, H. Zhou, F. Xu, and L. Li.
Hyper-erlang based model for network traffic approximation.
In Y. Pan, D. Chen, M. Guo, J. Cao, and J. Dongarra, editors, *Parallel and Distributed Processing and Applications*, volume 3758 of *Lecture Notes in Computer Science*, pages 1012–1023. Springer Berlin / Heidelberg, 2005.
10.1007/11576235_101.