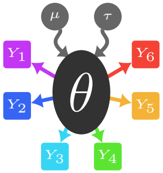


MultiBUGS
A parallel implementation of the BUGS modelling framework for faster Bayesian inference

Robert Goudie and Andrew Thomas (and the many past contributors to the BUGS project)
MRC Biostatistics Unit, University of Cambridge
Advances and challenges in Machine Learning Languages, 21 May 2019

BUGS is general-purpose Bayesian modelling software that implements Markov chain Monte Carlo (MCMC).

Two key ideas:

- The declarative BUGS language (Thomas, 2006), through which the user specifies the graphical model (Lauritzen *et al.*, 1990)
- Markov Chain Monte Carlo simulation (MCMC) for approximating the posterior distribution

Thomas, A. (2006). “The BUGS Language”. *R News* **6**, 17–21.
Lauritzen, S. L. *et al.* (1990). “Independence Properties of Directed Markov Fields”. *Networks* **20**, 491–505.

Background Timeline of BUGS

1989	BUGS project started.
1991	Prototype for random effects model (Thomas *et al.*, 1992).
1993	Version 0.1 released.
1994	Spatial smoothing at 5th Valencia (Spiegelhalter *et al.*, 1996).
1997	WinBUGS released, and Metropolis-Hastings implemented.
2004	Open source OpenBUGS started.
2018	MultiBUGS 1.0 released (Goudie *et al.*, ?2019) `https://www.multibugs.org`.

Thomas, A. *et al.* (1992). “BUGS: A program to perform Bayesian inference using Gibbs Sampling”. In: *Bayesian Statistics 4*. Oxford, UK: Oxford University Press, pp. 837–842.
Spiegelhalter, D. J. *et al.* (1996). “Computation on Bayesian graphical models”. In: *Bayesian Statistics 5*. Oxford, UK: Oxford University Press, pp. 407–425.
Goudie, R. J. B. *et al.* (?2019). “MultiBUGS: A Parallel Implementation of the BUGS Modelling Framework for Faster Bayesian Inference”. *Journal of Statistical Software*. https://arxiv.org/abs/1704.03216.

Specifying a model in BUGS

BUGS code is equivalent to algebraic statements defining a statistical model

e.g. linear regression

$$\begin{aligned} y_i &\sim N(\mu_i, \sigma^2) \\ \mu_i &= \alpha + \beta x_i \\ i &= 1, \ldots, n \end{aligned}$$

plus priors on α, β, σ

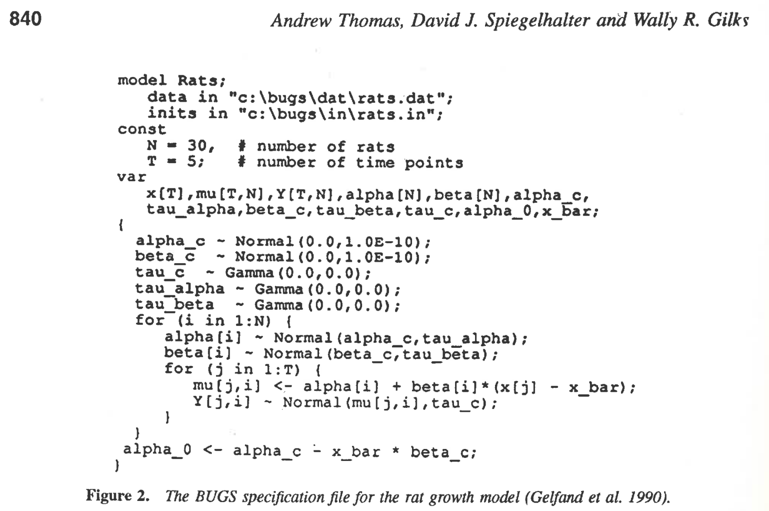
- Unknown parameters are α, β, σ
- Known data are y_i, x_i
  (and parameters of priors for α, β, σ)

```
model {
 for (i in 1:n) {
   y[i]  ~ dnorm(mu[i], tau)
   mu[i] <- alpha + beta*x[i]
 }
 alpha   ~ dunif(-100,100)
 beta    ~ dunif(-100,100)
 sigma   ~ dunif(0, 100)
 tau     <- 1/(sigma*sigma)
}
```

`~` stochastic relation: used for
- models for data
- priors for parameters

`<-` logical/deterministic relation

Thomas *et al.* (1992)



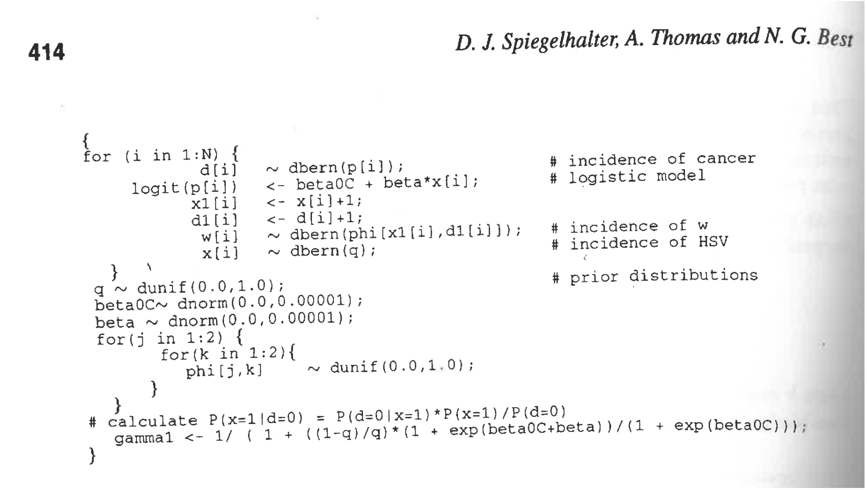
840 *Andrew Thomas, David J. Spiegelhalter and Wally R. Gilks*

```
model Rats;
   data in "c:\bugs\dat\rats.dat";
   inits in "c:\bugs\in\rats.in";
const
   N = 30,   # number of rats
   T = 5;    # number of time points
var
   x[T],mu[T,N],Y[T,N],alpha[N],beta[N],alpha_c,
   tau_alpha,beta_c,tau_beta,tau_c,alpha_0,x_bar;
{
  alpha_c ~ Normal(0.0,1.0E-10);
  beta_c  ~ Normal(0.0,1.0E-10);
  tau_c   ~ Gamma(0.0,0.0);
  tau_alpha ~ Gamma(0.0,0.0);
  tau_beta  ~ Gamma(0.0,0.0);
  for (i in 1:N) {
     alpha[i] ~ Normal(alpha_c,tau_alpha);
     beta[i] ~ Normal(beta_c,tau_beta);
     for (j in 1:T) {
        mu[j,i] <- alpha[i] + beta[i]*(x[j] - x_bar);
        Y[j,i] ~ Normal(mu[j,i],tau_c);
     }
  }
 alpha_0 <- alpha_c - x_bar * beta_c;
}
```

Figure 2. *The BUGS specification file for the rat growth model (Gelfand et al. 1990).*

Thomas, A. *et al.* (1992). "BUGS: A program to perform Bayesian inference using Gibbs Sampling". In: *Bayesian Statistics 4*. Oxford, UK: Oxford University Press, pp. 837–842.

Spiegelhalter *et al.* (1996)



414 *D. J. Spiegelhalter, A. Thomas and N. G. Best*

```
{
for (i in 1:N) {
           d[i]   ~ dbern(p[i]);                      # incidence of cancer
     logit(p[i])  <- beta0C + beta*x[i];              # logistic model
          x1[i]   <- x[i]+1;
          d1[i]   <- d[i]+1;
           w[i]   ~ dbern(phi[x1[i],d1[i]]);          # incidence of w
           x[i]   ~ dbern(q);                         # incidence of HSV
   }
 q ~ dunif(0.0,1.0);                                  # prior distributions
 beta0C~ dnorm(0.0,0.00001);
 beta ~ dnorm(0.0,0.00001);
 for(j in 1:2) {
       for(k in 1:2){
          phi[j,k]    ~ dunif(0.0,1.0);
       }
   }
 # calculate P(x=1|d=0) = P(d=0|x=1)*P(x=1)/P(d=0)
   gamma1 <- 1/ ( 1 + ((1-q)/q)*(1 + exp(beta0C+beta))/(1 + exp(beta0C)));
 }
```

Spiegelhalter, D. J. *et al.* (1996). “Computation on Bayesian graphical models”. In: *Bayesian Statistics 5*. Oxford, UK: Oxford University Press, pp. 407–425.

Background & motivation

Impossible to use OpenBUGS with a large amount of data, partly because it is single-threaded.

Background & motivation

Impossible to use OpenBUGS with a large amount of data, partly because it is single-threaded.

Aim: to make the speed-ups of multi-core computation available to applied statisticians using BUGS for general models, without requiring any knowledge of parallel programming (and without needing to anything manually)

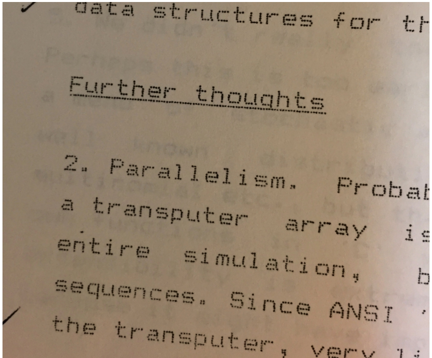
Note: not aiming to change mixing properties of the Markov chain, simply to run it faster

Background & motivation

Impossible to use OpenBUGS with a large amount of data, partly because it is single-threaded.

Aim: to make the speed-ups of multi-core computation available to applied statisticians using BUGS for general models, without requiring any knowledge of parallel programming (and without needing to anything manually)

Note: not aiming to change mixing properties of the Markov chain, simply to run it faster



✓ data structures for t

Further thoughts

2. Parallelism. Proba

a transputer array i

entire simulation, b

sequences. Since ANSI

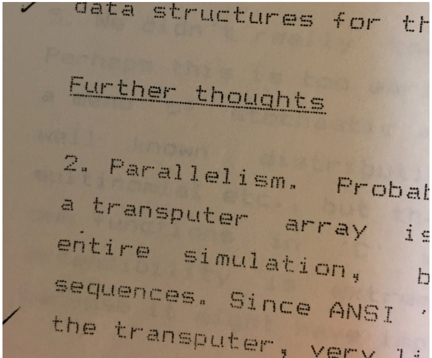
✓ the transputer, very

Background & motivation

Impossible to use OpenBUGS with a large amount of data, partly because it is single-threaded.

Aim: to make the speed-ups of multi-core computation available to applied statisticians using BUGS for general models, without requiring any knowledge of parallel programming (and without needing to anything manually)

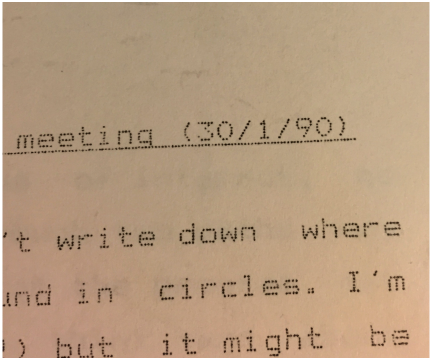
Note: not aiming to change mixing properties of the Markov chain, simply to run it faster



✓ data structures for th

Further thoughts

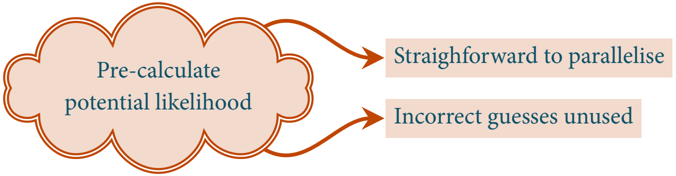
2. Parallelism. Probab
a transputer array is
entire simulation, b
sequences. Since ANSI
✓ the transputer, very li



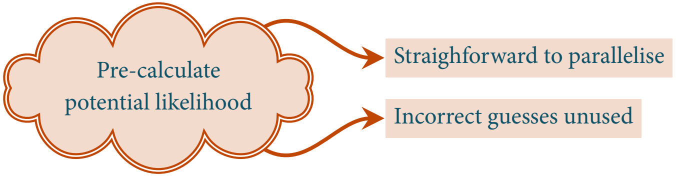
meeting (30/1/90)

t write down where
und in circles. I'm
) but it might be

Selected other parallelisation approaches


Pre-calculate potential likelihood
Straighforward to parallelise
Incorrect guesses unused

Selected other parallelisation approaches



Selected other parallelisation approaches

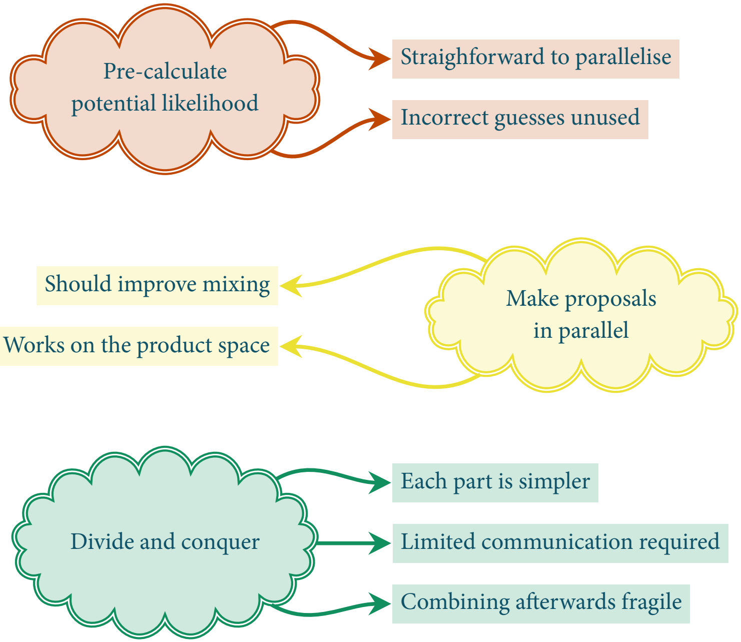
Pre-calculate potential likelihood

- Straighforward to parallelise
- Incorrect guesses unused

Make proposals in parallel

- Should improve mixing
- Works on the product space

Selected other parallelisation approaches


Pre-calculate potential likelihood
Straighforward to parallelise
Incorrect guesses unused
Make proposals in parallel
Should improve mixing
Works on the product space
Divide and conquer
Each part is simpler
Limited communication required
Combining afterwards fragile

MultiBUGS implements two levels of parallelisation.

Bradford, R. and Thomas, A. (1996). “Markov Chain Monte Carlo Methods for Family Trees Using a Parallel Processor”. *Statistics and Computing* **6**, 67–75.
Wilkinson, D. (2006). “Parallel Bayesian Computation”. In: *Handbook of Parallel Computing and Statistics*. Ed. by Kontoghiorghes, E. Boca Raton, FL: Chapman and Hall/CRC, pp. 477–508.

MultiBUGS implements two levels of parallelisation.

Simple approach – run each of multiple, independent MCMC chains on a separate CPU or core (Bradford and Thomas, 1996)

- Useful for assessing convergence e.g. the Brooks-Gelman-Rubin diagnostic
- Burn-in time isn't shortened

Bradford, R. and Thomas, A. (1996). "Markov Chain Monte Carlo Methods for Family Trees Using a Parallel Processor". *Statistics and Computing* **6**, 67–75.

Wilkinson, D. (2006). "Parallel Bayesian Computation". In: *Handbook of Parallel Computing and Statistics*. Ed. by Kontoghiorghes, E. Boca Raton, FL: Chapman and Hall/CRC, pp. 477–508.

Parallelising the BUGS algorithm | Types of parallelisation

MultiBUGS implements two levels of parallelisation.

Simple approach – run each of multiple, independent MCMC chains on a separate CPU or core (Bradford and Thomas, 1996)

- Useful for assessing convergence e.g. the Brooks-Gelman-Rubin diagnostic
- Burn-in time isn't shortened

More complicated approach – use multiple CPUs/cores for a single MCMC chain

- Aim to shorten the per-iteration computation time by identifying tasks that can be calculated in parallel (Wilkinson, 2006)
- MultiBUGS parallelises the following tasks:
    1. "Likelihood" computation
    2. Sampling of conditionally-independent components

Bradford, R. and Thomas, A. (1996). "Markov Chain Monte Carlo Methods for Family Trees Using a Parallel Processor". *Statistics and Computing* **6**, 67–75.
Wilkinson, D. (2006). "Parallel Bayesian Computation". In: *Handbook of Parallel Computing and Statistics*. Ed. by Kontoghiorghes, E. Boca Raton, FL: Chapman and Hall/CRC, pp. 477–508.

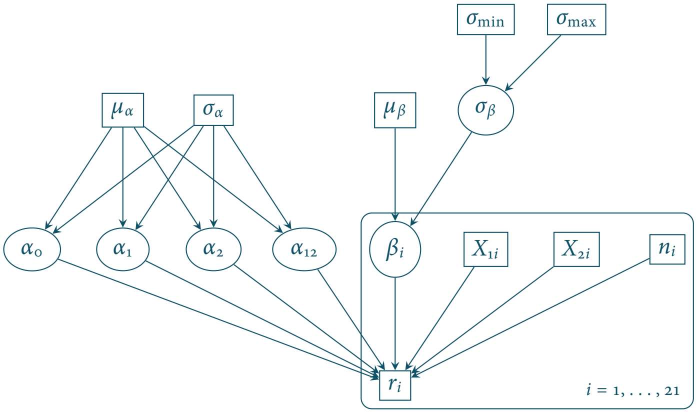
Trivial illustrative example The "seeds" model

A random-effects logistic regression without outcome r_i and covariates X_{1i} and X_{2i} (21 observations)

$$r_i \sim \mathrm{Bin}(p_i, n_i)$$

$$\mathrm{logit}(p_i) = \alpha_0 + \alpha_1 X_{1i} + \alpha_2 X_{2i} + \alpha_{12} X_{1i} X_{2i} + \beta_i \qquad \alpha_0, \alpha_1, \alpha_2, \alpha_{12} \sim \mathrm{N}(\mu_\alpha, \sigma_\alpha^2)$$

$$\beta_i \sim \mathrm{N}(\mu_\beta, \sigma_\beta^2) \qquad \sigma_\beta \sim \mathrm{Unif}(\sigma_{\min}, \sigma_{\max})$$



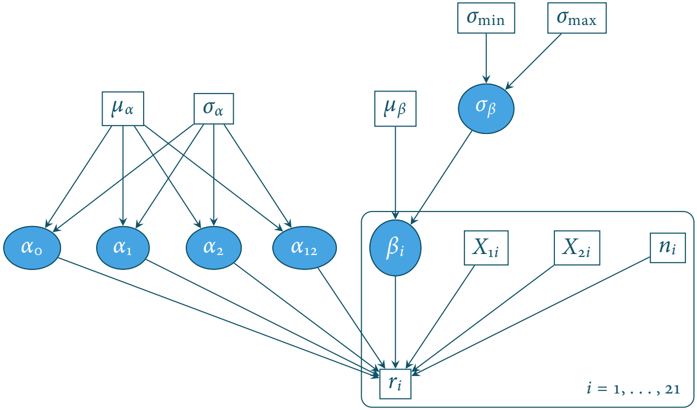
Trivial illustrative example | The "seeds" model

A random-effects logistic regression without outcome r_i and covariates X_{1i} and X_{2i} (21 observations)

$$r_i \sim \text{Bin}(p_i, n_i)$$

$$\text{logit}(p_i) = \alpha_0 + \alpha_1 X_{1i} + \alpha_2 X_{2i} + \alpha_{12} X_{1i} X_{2i} + \beta_i \qquad \alpha_0, \alpha_1, \alpha_2, \alpha_{12} \sim \text{N}(\mu_\alpha, \sigma_\alpha^2)$$

$$\beta_i \sim \text{N}(\mu_\beta, \sigma_\beta^2) \qquad \sigma_\beta \sim \text{Unif}(\sigma_{\min}, \sigma_{\max})$$



At each MCMC iteration, BUGS does the following:

```
for $v$ in $S$ do
    Do something involving $p(v \mid V_{-v})$
end for
```

At each MCMC iteration, BUGS does the following:

```
for $v$ in $S$ do
    Do something involving $p(v \mid V_{-v})$
end for
```

The conditional distribution $p(v \mid V_{-v})$ of a node $v \in S$, given the other nodes V_{-v}, is

$$\begin{aligned} p(v \mid V_{-v}) &\propto p(v \mid \mathrm{pa}(v)) \times \prod_{u \in \mathrm{ch}(v)} p(u \mid \mathrm{pa}(u)) \\ &= p(v \mid \mathrm{pa}(v)) \times L(v) \\ &= \text{“prior” term} \times \text{“likelihood” term} \end{aligned}$$

At each MCMC iteration, BUGS does the following:

```
for $v$ in $S$ do
    Evaluate the "prior" $p(v \mid \mathrm{pa}(v)$
    for $u \in \mathrm{ch}(v)$ do
        Evaluate "likelihood" component $p(u \mid \mathrm{pa}(u))$
    end for
    etc ...
end for
```

The conditional distribution $p(v \mid V_{-v})$ of a node $v \in S$, given the other nodes V_{-v}, is

$$\begin{aligned} p(v \mid V_{-v}) &\propto p(v \mid \mathrm{pa}(v)) \times \prod_{u \in \mathrm{ch}(v)} p(u \mid \mathrm{pa}(u)) \\ &= p(v \mid \mathrm{pa}(v)) \times L(v) \\ &= \text{“prior” term} \times \text{“likelihood” term} \end{aligned}$$

When a parameter has many children, the likelihood is the product of many terms.

$$L(v) = \prod_{u \in \mathrm{ch}(v)} p(u \mid \mathrm{pa}(u))$$

When a parameter has many children, the likelihood is the product of many terms.

$$L(v) = \prod_{u \in \mathrm{ch}(v)} p(u \mid \mathrm{pa}(u))$$

But, with a partition of the children $\mathrm{ch}(v) = \left\{ \mathrm{ch}^{(1)}(v), \ldots, \mathrm{ch}^{(C)}(v) \right\}$,

$$L(v) = \underbrace{\left[\prod_{u \in \mathrm{ch}^{(1)}(v)} p(u \mid \mathrm{pa}(u)) \right]}_{\text{Core 1}} \times \underbrace{\left[\prod_{u \in \mathrm{ch}^{(2)}(v)} p(u \mid \mathrm{pa}(u)) \right]}_{\text{Core 2}} \times \ldots \times \underbrace{\left[\prod_{u \in \mathrm{ch}^{(C)}(v)} p(u \mid \mathrm{pa}(u)) \right]}_{\text{Core C}}$$

Parameters that do not directly depend on each other can be updated simultaneously

Parameters that do not directly depend on each other can be updated simultaneously

More precisely, parameters in a mutually conditionally-independent set $W \subseteq S$ can be updated simultaneously. That is, W satisfying

$$\text{all } w_1, w_2 \in W \ (w_1 \neq w_2) \text{ satisfy } w_1 \perp\!\!\!\perp w_2 \mid V \setminus W$$

If not all parameters can be collated into a single W, form a series of Ws and sample in turn.

Trivial illustrative example Identifying mutually conditionally-independent sets W

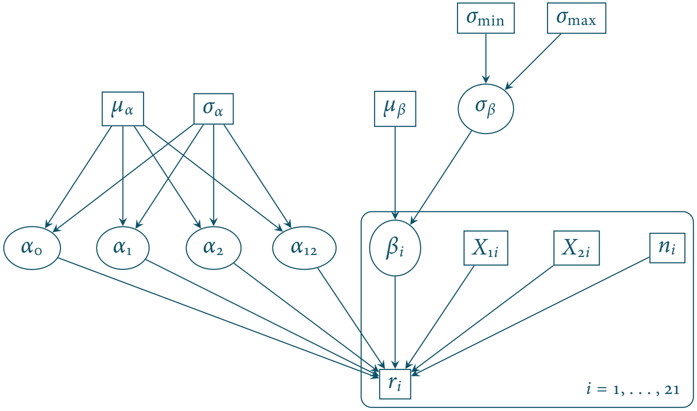
Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

$$d(v) = \begin{cases} 0 & \text{if } \mathrm{pa}(v) = \varnothing \\ 1 + \max_{u \in \mathrm{pa}(v)} d(u) & \text{otherwise} \end{cases}$$

Trivial illustrative example | Identifying mutually conditionally-independent sets W

Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

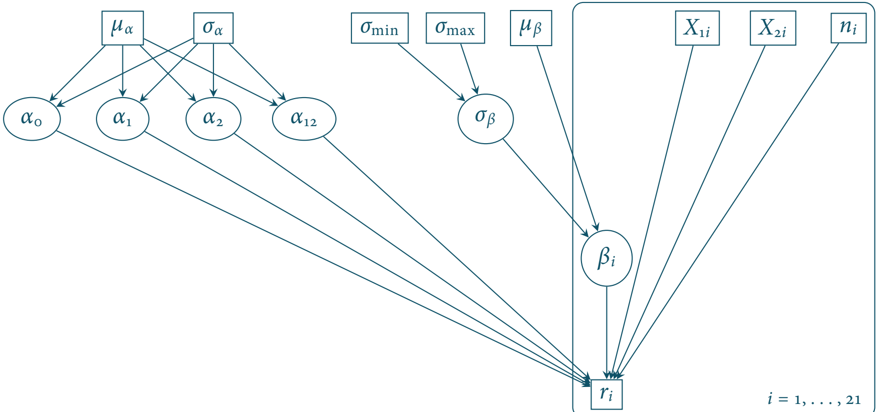
$$d(v) = \begin{cases} 0 & \text{if } \mathrm{pa}(v) = \emptyset \\ 1 + \max_{u \in \mathrm{pa}(v)} d(u) & \text{otherwise} \end{cases}$$



Identifying mutually conditionally-independent sets W

Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

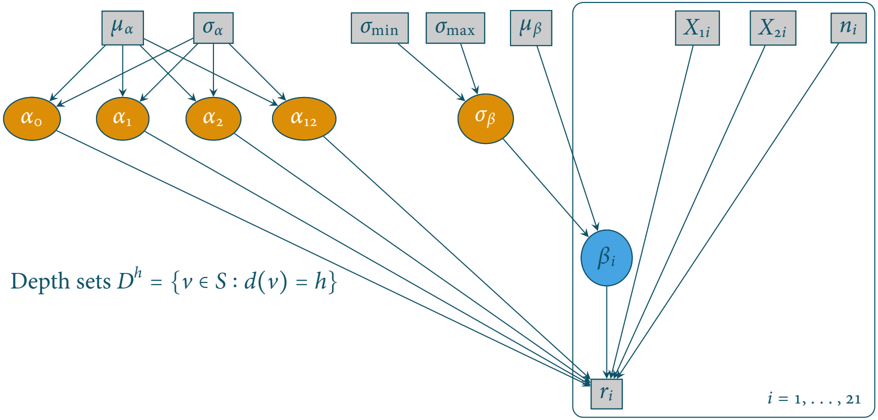
$$d(v) = \begin{cases} 0 & \text{if } \mathrm{pa}(v) = \varnothing \\ 1 + \max_{u \in \mathrm{pa}(v)} d(u) & \text{otherwise} \end{cases}$$



Trivial illustrative example | Identifying mutually conditionally-independent sets W

Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

$$d(v) = \begin{cases} 0 & \text{if } \mathrm{pa}(v) = \varnothing \\ 1 + \max_{u \in \mathrm{pa}(v)} d(u) & \text{otherwise} \end{cases}$$

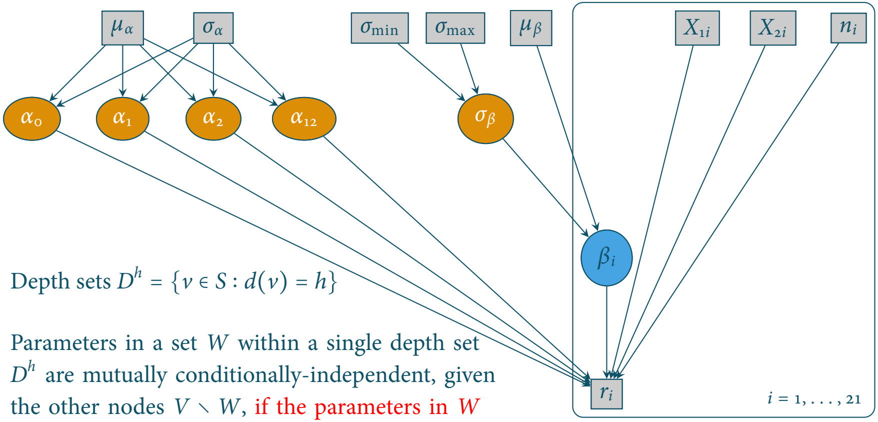


Depth sets $D^h = \{v \in S : d(v) = h\}$

Trivial illustrative example — Identifying mutually conditionally-independent sets W

Define the topological depth of a node $v \in V$ recursively, starting from the nodes with no parents.

$$d(v) = \begin{cases} 0 & \text{if } \mathrm{pa}(v) = \emptyset \\ 1 + \max_{u \in \mathrm{pa}(v)} d(u) & \text{otherwise} \end{cases}$$



Depth sets $D^h = \{v \in S : d(v) = h\}$

Parameters in a set W within a single depth set D^h are mutually conditionally-independent, given the other nodes $V \setminus W$, if the parameters in W have no child node in common.

MultiBUGS aims to

- Parallelise the evaluation of the "likelihood" of 'fixed effect'-like parameters
- Parallelise sampling of 'random effect'-like parameters

Selecting parallelisation type for each parameter

MultiBUGS aims to

- Parallelise the evaluation of the "likelihood" of 'fixed effect'-like parameters
- Parallelise sampling of 'random effect'-like parameters

Let $\overline{\text{ch}} = \text{mean}_{v \in S} |\text{ch}(v)|$ be the mean number of children across parameters

Heuristic algorithm:
Consider each depth set D^h in turn, starting with the 'deepest' set

```
**if** a parameter has more than $2 \times \overline{\text{ch}}$ children **then**
    Parallelise evaluation of this parameter's "likelihood"
**else**
    Sample this parameter in parallel, if possible
**end if**
```

Selecting parallelisation type for each parameter

MultiBUGS aims to

- Parallelise the evaluation of the “likelihood” of ‘fixed effect’-like parameters
- Parallelise sampling of ‘random effect’-like parameters

Let $\overline{\text{ch}} = \text{mean}_{v \in S} |\text{ch}(v)|$ be the mean number of children across parameters

Heuristic algorithm:

Consider each depth set D^h in turn, starting with the ‘deepest’ set

```
if a parameter has more than $2 \times \overline{\text{ch}}$ children then
    Parallelise evaluation of this parameter’s “likelihood”
else
    Sample this parameter in parallel, if possible
end if
```

Users need only specify how many cores they wish to use.

The heuristic is deterministic – makes reproducing the chain easy.

Trivial illustrative example Computation schedule, with 4 cores

There are 26 stochastic parameters.

Trivial illustrative example | Computation schedule, with 4 cores

There are 26 stochastic parameters.

Topological depths:

- $d(\beta_1) = \cdots = d(\beta_{21}) = 2$
- $d(\alpha_0) = \cdots = d(\alpha_{12}) = d(\sigma_\beta) = 1$

Trivial illustrative example | Computation schedule, with 4 cores

There are 26 stochastic parameters.

Topological depths:

- $d(\beta_1) = \cdots = d(\beta_{21}) = 2$
- $d(\alpha_0) = \cdots = d(\alpha_{12}) = d(\sigma_\beta) = 1$

1. Parameters $\beta_1, \ldots, \beta_{21}$
   - Likelihood evaluation not parallelised these parameter have only 1 child and $\overline{\text{ch}} \approx 4.8$.
   - But, $\beta_1, \ldots, \beta_{21}$ are mutually conditionally-independent and so sampling can be parallelised
   - $21 \bmod 4 \neq 0$ so we will have idle cores

	Core			
Row	1	2	3	4
1	β_1	β_2	β_3	β_4
2	β_5	β_6	β_7	β_8
3	β_9	β_{10}	β_{11}	β_{12}
4	β_{13}	β_{14}	β_{15}	β_{16}
5	β_{17}	β_{18}	β_{19}	β_{20}
6	β_{21}			

Trivial illustrative example | Computation schedule, with 4 cores

There are 26 stochastic parameters.

Topological depths:

- $d(\beta_1) = \cdots = d(\beta_{21}) = 2$
- $d(\alpha_0) = \cdots = d(\alpha_{12}) = d(\sigma_\beta) = 1$

1. Parameters $\beta_1, \ldots, \beta_{21}$
   - Likelihood evaluation not parallelised these parameter have only 1 child and $\overline{\text{ch}} \approx 4.8$.
   - But, $\beta_1, \ldots, \beta_{21}$ are mutually conditionally-independent and so sampling can be parallelised
   - $21 \bmod 4 \neq 0$ so we will have idle cores
2. Parameters $\alpha_0, \alpha_1, \alpha_2, \alpha_{12}$ and σ_β
   - All of these parameters have 21 children $\overline{\text{ch}} \approx 4.8$, so likelihood computation is parallelised

	Core			
Row	1	2	3	4
1	β_1	β_2	β_3	β_4
2	β_5	β_6	β_7	β_8
3	β_9	β_{10}	β_{11}	β_{12}
4	β_{13}	β_{14}	β_{15}	β_{16}
5	β_{17}	β_{18}	β_{19}	β_{20}
6	β_{21}			
7	α_{12}	α_{12}	α_{12}	α_{12}
8	α_1	α_1	α_1	α_1
9	α_2	α_2	α_2	α_2
10	α_0	α_0	α_0	α_0
11	σ_β	σ_β	σ_β	σ_β

Implementation notes

Each core keeps

- The complete DAG, the computation schedule, and associated sampling algorithms
- A copy of the current state of the MCMC
- Two pseudo-random number generation (PRNG) streams
    1. "Core-specific" stream, initialised with a different seed for each core
        - Used when we wish to sample independently across cores
    2. "Common" stream, initialised using the same seed on all cores.
        - Used when we wish to obtain the same samples across cores

Implementation notes

Each core keeps

- The complete DAG, the computation schedule, and associated sampling algorithms
- A copy of the current state of the MCMC
- Two pseudo-random number generation (PRNG) streams
    1. "Core-specific" stream, initialised with a different seed for each core
        - Used when we wish to sample independently across cores
    2. "Common" stream, initialised using the same seed on all cores.
        - Used when we wish to obtain the same samples across cores

Most existing code can be reused!

Implementation notes

Each core keeps

- The complete DAG, the computation schedule, and associated sampling algorithms
- A copy of the current state of the MCMC
- Two pseudo-random number generation (PRNG) streams
    1. "Core-specific" stream, initialised with a different seed for each core
        - Used when we wish to sample independently across cores
    2. "Common" stream, initialised using the same seed on all cores.
        - Used when we wish to obtain the same samples across cores

Most existing code can be reused!

Likelihood parallelisation: Just delete the children whose likelihood contribution is calculated elsewhere. MPI function `Allreduce` used to aggregate.

For Metropolis-Hastings: the prior, the sampling of new value, and Metropolis test (redundantly) replicated on every core, using "common" PRNG stream

Implementation notes

Each core keeps

- The complete DAG, the computation schedule, and associated sampling algorithms
- A copy of the current state of the MCMC
- Two pseudo-random number generation (PRNG) streams
    1. "Core-specific" stream, initialised with a different seed for each core
        - Used when we wish to sample independently across cores
    2. "Common" stream, initialised using the same seed on all cores.
        - Used when we wish to obtain the same samples across cores

Most existing code can be reused!

Likelihood parallelisation: Just delete the children whose likelihood contribution is calculated elsewhere. MPI function `Allreduce` used to aggregate.

For Metropolis-Hastings: the prior, the sampling of new value, and Metropolis test (redundantly) replicated on every core, using "common" PRNG stream

Sampling parallelisation: Just delete the nodes that are sampled elswhere from the list of nodes to be updated. Sample using "core-specific" PRNG stream, and propagate new values across cores using `Allgather`.

Hierarchical regression example The "methadone" model

Based on an analysis linked database of methadone prescriptions given to opioid dependent patients in Scotland (Gao *et al.*, 2016)

425,112 observations, with the following structure:

- Geographic regions ($i = 1, \ldots, 8$)
  - Containing patients (20410 in total)
    - Each of whom may have multiple prescriptions

For some measurements patient-level identifiers are available:

$$y_{ijk} = \sum_{m=1}^{4} \beta_m \times \underbrace{x_{mij}}_{\text{covariates}} + \underbrace{u_i}_{\substack{\text{region-}\\\text{specific}\\\text{intercept}}} + \underbrace{v_i}_{\substack{\text{region-}\\\text{specific}\\\text{slope}}} \times \underbrace{r_{ijk}}_{\text{covariate}} + \underbrace{w_{ij}}_{\substack{\text{patient-}\\\text{level}\\\text{intercept}}} + \varepsilon_{ijk}$$

For other measurements no patient-level identifier is available:

$$z_{il} = \lambda + \underbrace{u_i}_{\substack{\text{region-}\\\text{specific}\\\text{intercept}}} + \underbrace{v_i}_{\substack{\text{region-}\\\text{specific}\\\text{slope}}} \times \underbrace{s_{il}}_{\text{covariate}} + \eta_{il}$$

Gao, L. *et al.* (2016). "Risk-Factors for Methadone-Specific Deaths in Scotland's Methadone-Prescription Clients between 2009 and 2013". *Drug and Alcohol Dependence* **167**, 214–223.

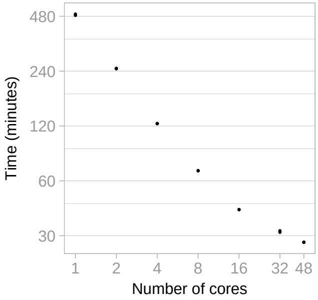
Hierarchical regression example | Run times on 2.4Ghz AMD Operon 6378

In OpenBUGS, running chains 2 for 15,000 iterations takes about 32 hours.

JAGS 4.0.0 took 9 hours.

In MultiBUGS 1.0:

- Sampling of pairs of random-effect means and variances parallelised;
- Sampling of person-level random effects w_{ij} parallelised, except for the component corresponding to the person with the most observations (176 observations)
- The likelihood computation of all the other parameters in the model is parallelised



Outlook

MultiBUGS 1.0 is available – `https://www.multibugs.org`

MultiBUGS 2.0[1] uses a more efficient system for communicating partial models to cores

Released version requires Windows, but MultiBUGS 2.0 will run on Linux too

Other software:

- JAGS - Martyn Plummer is adopting a similar ideas (using OpenMP)
- NIMBLE? Not sure
- Stan - map/reduce with user-selected sharding

[1] `https://github.com/MultiBUGS/MultiBUGS`