The microsimulation package

Mark Clements

2021-07-02

Who am I?

- Biostatistician
- Taught SAS by Ross Ihaka in 1990
- CRAN maintainer for `microsimulation`, `rstpm2` and six other packages
- Funders: Swedish Research Council, Swedish Cancer Society, Prostatacancerförbundet, Karolinska Institutet
- Collaborators: a long list, including Andreas Karlsson, Shuang Hao and Edna Keeney

Some possible questions for the end of this presentation

- How is this different to the descem package?
- Why did you develop the microsimulation package? Why not use the hesim package?
- Is an event-oriented discrete event simulation really that flexible? Why not use a Markov model?
- When is it worthwhile writing the simulation in C++?

Why did we develop the microsimulation package?

- We had a challenging research question: evaluate the cost-effectiveness of prostate cancer screening strategies:
  - No screening
  - Screening with different start and stop ages and re-screening intervals
  - Different screening tests
  - Different biopsy technologies
  - Genetic risk scores
  - Risk-stratified screening based on test levels

Why did we develop the microsimulation package?

- We had a challenging research question: evaluate the cost-effectiveness of prostate cancer screening strategies:
  - No screening
  - Screening with different start and stop ages and re-screening intervals
  - Different screening tests
  - Different biopsy technologies
  - Genetic risk scores
  - Risk-stratified screening based on test levels
- Model components
  1. Natural history (longitudinal biomarker, disease states)
  2. Screening (screening initiation, re-screening, biopsy)
  3. Clinical (diagnosis, treatment, management, palliative care)

Possible model classes

Discrete-time Markov models

- I have modelled cervical cancer screening used TreeAge Pro with clones, tunnel states and outlines
- This was difficult and required some model simplifications (e.g. how to combine natural history and screening states)
- Not recommended for this research question 😕

Possible model classes

Discrete-time Markov models

- I have modelled cervical cancer screening used TreeAge Pro with clones, tunnel states and outlines
- This was difficult and required some model simplifications (e.g. how to combine natural history and screening states)
- Not recommended for this research question 😕

Individual-based continuous time Markov/semi-Markov models

- This could be done using hesim
- Readily incorporates individual heterogeneity and different time scales 😎
- The model specification would need to combine natural history and screening states (difficult?)
- Unclear whether the current frameworks are sufficiently flexible

Possible model classes

Event-oriented discrete event simulation

- Well-established individual-based, continuous time simulation approach
- Exceedingly flexible
  - Easy to specify individual attributes (e.g. natural history, screening history, treatment history) that affect the simulation in different ways
  - Can allow for resource allocation (see `simmer`)
  - The approach is based on an event queue, where events are scheduled and the next event is processed

Event-oriented discrete event simulation

Data: n simulations, other parameters
Result: Report for expected QALYs and costs
Simulation initialization: set up report;
for $i \leftarrow 1$ **to** n **do**
  Individual initialization (`init()`): clear the event queue; set QALYs = 0, costs = 0; set individual attributes; insert initial events into the event queue (using `scheduleAt()`);
  while *Event queue not empty* **do**
    For the next event (`handleMessage()`): update the current time and respond to the event;
    - Update any simulation attributes (e.g. reporting);
    - Update any individual attributes (e.g. QALYs, costs);
    - Insert any new events (`scheduleAt()`);
    - Cancel events (`clear()` or `cancel()`);

Our requirements

- Event-oriented discrete event simulation
- Data management before and after the simulation in R
- Simulate in R or in C++
- For a probabilistic analysis in cancer screening with 10 strategies and 10^3 parameter sets and at least 10^6 individual simulations per set, we need at least 10^{10} individual simulations
- Some tools for HTA
- GPL license

Our solution: `microsimulation` package

- R-based simulations based on reference classes (semantically equivalent to descem)
- C++ based simulation used an existing C++ library (Ssim)
  - Fast! 10^{10} individual simulations takes less than three days on a 40 core node. The R version would be approximately 50 times slower ☹
  - But requires programming in C++ (two language problem)
- As per the `hesim` package, we make extensive use of the `Rcpp` library
- Important contributions:
  - Common random numbers: `C++` random number manipulation for `RngStream` random streams and sub-streams
  - Allows for inline model development for C++ code
- Now on CRAN

microsimulation: R API

Attribute/method	Description
`currentTime`, `now()`	Current time
`previousEventTime`	Previous event time
`scheduleAt()`	Insert a new event into the event queue
`clear()`	Remove all events from the event queue
`cancel()`	Test whether to remove events from the queue
`init()`	User-defined method for setting up the simulation
`handleMessage()`	User-defined method for responding to events
`final()`	User-defined method for end of simulation
`run()`	Run a single simulation

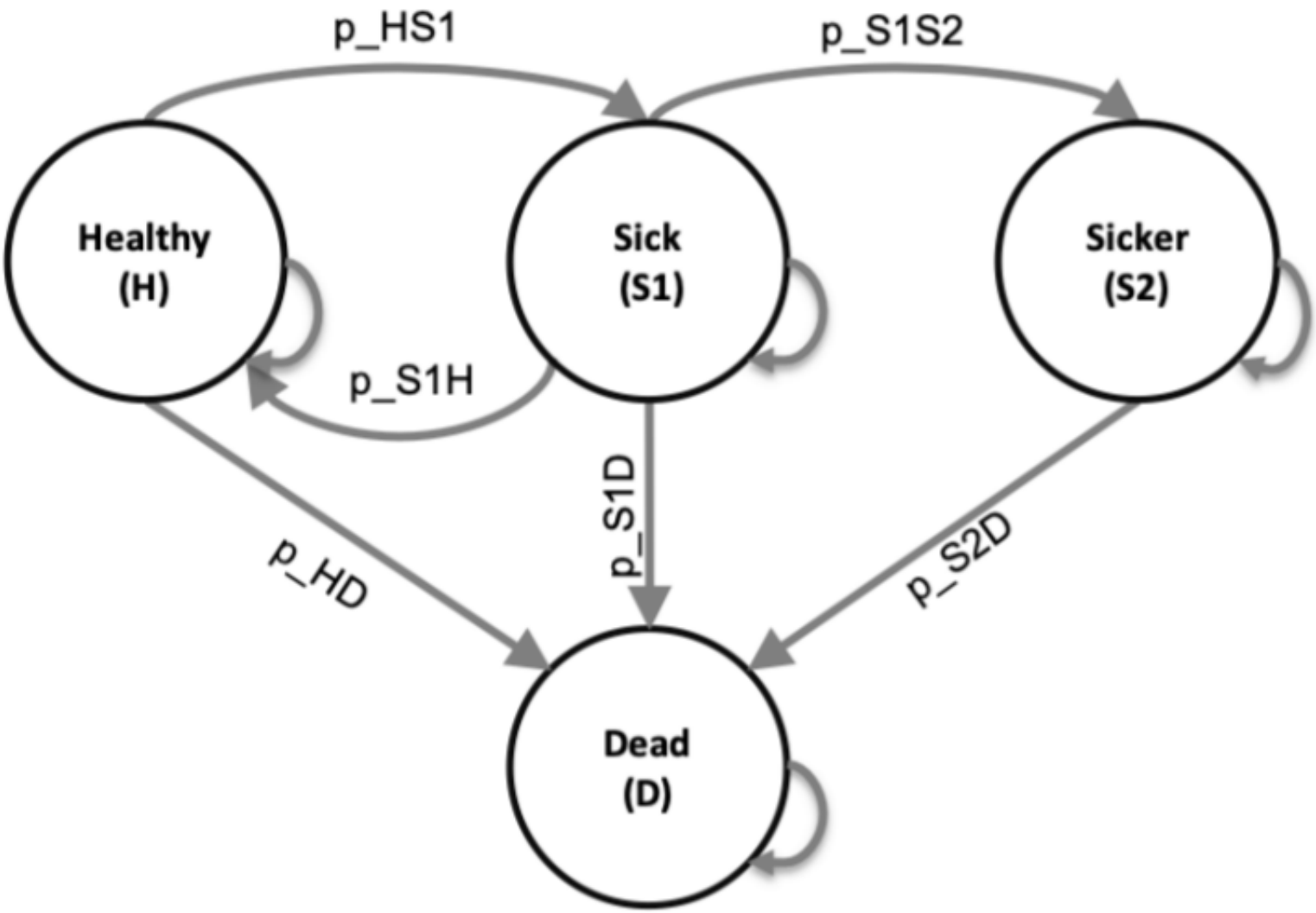
Simple example: Ticking bomb

```
library(microsimulation)
Ticking =
  setRefClass("Ticking",
              contains = "BaseDiscreteEventSimulation")
Ticking$methods(
  init = function() {
      scheduleAt(rexp(1,1), "tick")
      cat("begin\tend\t\tevent\n")
  },
  handleMessage = function(event) {
      cat(sprintf("%f\t%f\t%s\n", previousEventTime,
                  currentTime, event))
      clear()
      if (event != "explosion") {
          scheduleAt(currentTime+rexp(1,1),
                     if (event == "tick") "tock" else "tick")
          if (runif(1)<0.3)
              scheduleAt(currentTime+rexp(1,1), "explosion")
      }
  })
set.seed(1234)
Ticking$new()$run()
```

Simple example: Ticking bomb

```
begin end event
0.000000 2.501759 tick
2.501759 2.748517 tock
2.748517 3.029139 tick
3.029139 3.416321 tock
3.416321 4.240403 tick
4.240403 5.102813 tock
5.102813 5.940854 tick
5.940854 7.536959 tock
7.536959 9.287639 tick
9.287639 9.950329 tock
9.950329 9.964943 tick
9.964943 10.348485 explosion
```

Sick-Sicker example (Krijkamp et al 2018): diagram



Sick-Sicker example: parameters

```
param = within(list(), {
    p.HD = 0.005 # probability to die when healthy
    p.HS1 = 0.15 # probability to become sick when healthy
    p.S1H = 0.5 # probability to become healthy when sick
    p.S1S2 = 0.105 # probability to become sicker when sick
    rr.S1 = 3 # rate ratio of death when sick vs healthy
    rr.S2 = 10 # rate ratio of death when sicker vs healthy
    r.HD = -log(1-p.HD) # rate of death when healthy
    r.S1D = rr.S1 * r.HD # rate of death when sick
    r.S2D = rr.S2 * r.HD # rate of death when sicker
    p.S1D = 1-exp(-r.S1D) # probability to die when sick
    p.S2D = 1-exp(-r.S2D) # probability to die when sicker
    c_H = 2000 # cost of remaining one cycle healthy
    c_S1 = 4000 # cost of remaining one cycle sick
    c_S2 = 15000 # cost of remaining one cycle sicker
    c_Trt = 12000 # (additional) cost of treatment (per cycle)
    u_H = 1 # utility when healthy
    u_S1 = 0.75 # utility when sick
    u_S2 = 0.5 # utility when sicker
    u_Trt = 0.95 # utility when sick and being treated
    discountRate = 0.03 # discount rate
    Trt = FALSE # Treatment?
})
```

Sick-Sicker example: discrete to continuous time

```
library(expm)
## fix inconsistent transition
param = within(param, { p.HS2 = p.HS1*p.S1S2 })
## get transition probability matrix
Pmat = with(param,
            matrix(c(1-p.HD-p.HS1-p.HS2,p.HS1,p.HS2,p.HD,
                     p.S1H,1-p.S1H-p.S1S2-p.S1D,p.S1S2,p.S1D,
                     0,0,1-p.S2D,p.S2D,
                     0,0,0,1), 4, byrow=TRUE))
## get transition matrix
Qmat = expm::logm(Pmat) # matrix logarithm
## update parameters
param = within(param,
{ r_HS1 = Qmat[1,2]; r_HD = Qmat[1,4]
    r_S1H = Qmat[2,1]; r_S1S2 = Qmat[2,3]; r_S1D = Qmat[2,4]
    r_S2D = Qmat[3,4] })
```

Sick-Sicker example: define the class

```
library(microsimulation)
SickSicker =
    setRefClass("SickSicker",
                contains = "BaseDiscreteEventSimulation",
                fields = list(QALYs = "numeric", # cumulative
                              Costs = "numeric", # cumulative
                              cost = "numeric",
                              utility = "numeric",
                              state = "character"))
```

Sick-Sicker example: define init and cancelEvents method

```
SickSicker$methods(
                init = function() {
                    state <<- "Healthy"
                    QALYs <<- Costs <<- 0
                    cost <<- param$c_H
                    utility <<- param$u_H
                    scheduleAt(rexp(1,rate=param$r_HS1),"toS1")
                    scheduleAt(rexp(1,rate=param$r_HD),"toD")
                    scheduleAt(31.0,"toEOF") # end of follow-up
                },
                cancelEvents = function()
                    cancel(function(event)
                        event %in% c("toH","toS1","toS2","toD")))
```

Sick-Sicker example: define handleMessage() and final() methods

```
SickSicker$methods(
 handleMessage = function(event) {
     QALYs <<- QALYs + discountedInterval(utility, previousEventTime,
                                          now(), param$discountRate)
     Costs <<- Costs + discountedInterval(cost, previousEventTime,
                                          now(), param$discountRate)
     if (event == "toH") {
         state <<- "Healthy"
         utility <<- param$u_H
         cost <<- param$c_H
         cancelEvents()
         scheduleAt(now() + rexp(1,rate=param$r_HS1), "toS1")
         scheduleAt(now() + rexp(1,rate=param$r_HD), "toD")
     } else if (event == "toS1") {
         state <<- "Sick"
         utility <<- if (param$Trt) param$u_Trt else param$u_S1
         cost <<- param$c_S1 + param$Trt*param$c_Trt
         cancelEvents()
         scheduleAt(now() + rexp(1,rate=param$r_S1H), "toH")
         scheduleAt(now() + rexp(1,rate=param$r_S1S2), "toS2")
         scheduleAt(now() + rexp(1,rate=param$r_S1D), "toD")

     } else if (event == "toS2") {
```

Sick-Sicker example: define handleMessage() and final() methods

```
    } else if (event == "toS2") {
        state <<- "Sicker"
        utility <<- param$u_S2
        cost <<- param$c_S2 + (if (param$Trt) param$c_Trt else 0)
        cancelEvents()
        scheduleAt(now() + rexp(1,rate=param$r_S2D), "toD")
    } else if (event %in% c("toD","toEOF")) {
        clear() # end of simulation
    }
},
final = function() c(QALYs=QALYs, Costs=Costs))
```

Sick-Sicker example: running the simulations

```
set.seed(12345)
param$Trt = FALSE
simF = SickSicker()
simsF = t(replicate(1e3, simF$run()))
set.seed(12345)
param$Trt = TRUE
simT = SickSicker()
simsT = t(replicate(1e3, simT$run()))
su = function(x) c(mean=mean(x), se=sd(x)/sqrt(length(x)))
apply(simsF,2,su)
apply(simsT,2,su)
```

```
         QALYs     Costs
mean 14.224530 89891.640
se    0.164043  2078.324
         QALYs      Costs
mean 14.725614 167253.019
se    0.169923   3809.241
```

C++ implementations

- In-line C++ implementation for the Sick-Sicker example is described in the README at `https://github.com/mclements/microsimulation`
- We have also implemented an extensive prostate cancer model that builds on the `microsimulation` package. See `https://github.com/mclements/prostata`
- Some references:
  - `https://doi.org/10.1371/journal.pone.0211918`
  - `https://doi.org/10.1371/journal.pone.0246674`
  - `https://doi.org/10.1101/2021.03.31.21254617`

We are recruiting!

- I have two positions open for simulation modelling:
  - Post-doctoral researcher
  - Research assistant (which could lead to a PhD position)
- I also have a PhD position open investigating variational approximations in survival analysis

- Finally: thank you for your attention.

Additional slides

Model complexity for cancer screening

- For complex interventions such as cancer screening, we need to make strong assumptions
- One could use a simpler model with strong implicit assumptions
- One could use a more complex model with strong explicit assumptions
- Recommendation: a good natural history model provides a better foundation for comparing these complex interventions than simpler models

Rosetta for HTA

HTA DES implementations in multiple languages:

- R
- C++
- Visual Basic
- SAS
- Stata
- Julia
- JavaScript
- Java
- Python

I plan to use `https://github.com/mclements/rosetta`. See also `https://rosettacode.org/wiki/Priority_queue`