Downloaded from www.sacemaquarterly.com | The quarterly update on epidemiology from the South African Centre for Epidemiological Modelling and Analysis (SACEMA).

Published: June 2015

Speeding up agent-based modelling of sexually transmitted diseases

Nathan Geffen - PhD Student, Centre for Social Science Research, University of Cape Town.

Agent-based modelling, also called microsimulation, is a way of modelling epidemics that is growing in popularity. Instead of the traditional way of modelling using differential equations, an agent-based model consists of, perhaps, thousands of agents, each representing a person, and each behaving according to a simple set of rules (1). Instead of outputs such as infection and mortality rates being derived from equations, they are derived from the interactions of the agents over many iterations. These models are providing rich insights into the HIV epidemic (2, 3).

Agent-based models are more computationally intensive than traditional equation-based models, but they have some characteristics that make them attractive. They are stochastic: randomness is built into them. While simple models are usually easier to implement as a set of differential equations, as the number of variables increases and models become increasingly complex, equation-based models become unwieldy and ultimately impractical. Yet the rules of each agent in an agent-based model can be made quite complex without much difficulty, and the model, if designed well, will scale. Agent-based models also allow researchers to examine what happens to specific agents or groups of agents more easily than equation-based models. Equation-based models do allow a population to be broken down into different compartments, but having multiple compartments can quickly become unmanageable.

How agent-based models work

Events in agent-based models can be simulated in discrete or continuous time. Here we are primarily concerned with discrete microsimulation, but key principles are for the most part the same between them.

At the beginning of the simulation you initialise a population of agents with information. So if simulating the HIV epidemic perhaps you can give each agent an initial age, HIV infection status, sex and a parameter describing how often they form new sexual partnerships.

Then, for a discrete model, decide on a time-step, say one day, one week or one month. Also decide how many time-steps to run the simulation for. So if the time-step is one month, to execute the simulation for 20 years, 240 iterations are needed. Each iteration involves interactions of agents and events. Examples of events are *DIE*, which determines if it is an agent's turn to die, *INFECT* which determines if it is an agent's turn to contract, say, HIV, *MATCH*, which finds a new sexual partner for an agent, or *BIRTH* which decides whether or not an agent is to give birth to a new agent. So on each iteration, the simulation steps through every agent and applies events to it. During and at the end of the simulation, we calculate interesting information such as life-expectancy of the simulated population and the changes in the HIV prevalence and incidence rates over time.

This is a typical structure of a discrete microsimulation. Variations are possible, including swapping lines 2 and 3.

```
1 for each time-step
2 for each event E
3 for each agent A
4 if E should be applied to A
5 apply E to A
```

We are particularly interested in lines 3 to 5. The event applied in line 5 can either be a simple operation that considers only agent A, or it can be more complicated and consider a range of agents (maybe even all the other agents) with respect to A. The former will typically be fast, the latter is slow.

Speed and agent-based models

Speed is a problem for agent-based modelling. Only in the last fifteen years or so have personal computers reached a point where it is convenient to do agent-based disease modelling on them. While running a single simulation for a few minutes might not be inconvenient, we often need to run thousands of simulations, especially if we wish to do sensitivity testing and uncertainty analysis.

Let's say we want to execute an agent-based simulation on a standard mid-range laptop. There are several ways to speed it up:

- Code it in a faster lower-level language, such as C or C++ and be sure to use the compiler optimisations.
- Do multi-threading programming, so that all processors in your computer are put to use.
- Improve algorithms that are slow.

Currently I am doing all three of these as part of my PhD research. But it is improving the partner matching event by finding better algorithms that is proving the most interesting and effective.

Fast versus slow events

Let's consider the *DIE* event. On an iteration, it will be checked to see if it must be applied to every living agent. Assume a particular agent has a risk of dying in a given time step of 1%, or 0.01. Then the DIE event generates a uniform pseudo-random number on the range [0,1). If the random number is less than 0.1, the agent dies, else it lives for another time step.

The time that it takes *DIE* to execute on an iteration of the simulation is proportional to *n*, the number of agents. We write:
$T \propto n$

Now consider the partner matching event, *MATCH*. We want to match agent, *A*, with some other suitable agent in our simulation. In a first naïve approximation of how we code this, which we call *BruteForceMatch*, we could examine every other agent in the simulation to find a partner, *B*, with whom to match *A*. Here is pseudo-code to do this for every agent:

BruteForceMatch(agents)
// agents are stored in an array or list
Place all the agents, randomly, in a queue
For each agent, A, in the queue
For each agent, B, behind A in the queue
Determine if it's a suitable partner
Remove A and its chosen partner from the queue

In general, assuming all agents need to be matched, the execution time, *T*, over *n* agents, for this algorithm is $T \propto [(n-1)+(n-2)+(n-3)+...+3+2+1]$. Thus T equals $(n^2-n)/2$.

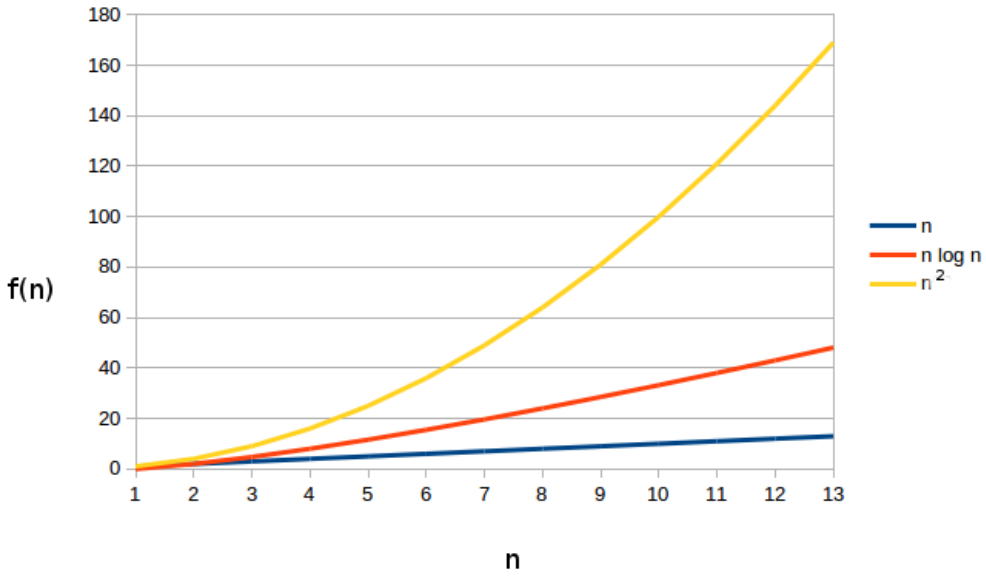
Examine this table to see how much faster the DIE *event is than the* MATCH *event.*

Agents (n)	***n***	***$(n^2-n)/2$***
10	*10*	*45*
100	*100*	*4,950*
1,000	*1,000*	*499,500*
10,000	*10,000*	*49,995,000*
100,000	*100,000*	*4,999,950,000*
1,000,000	*1,000,000*	*499,999,500,000*

For simplicity, we can say that we have this relationship between *T* and *n*:
$T \propto n^{2.}$

In computer science we call *DIE* an *O(n)* algorithm and *MATCH* an $O(n^2)$ (4). Of practical importance too are O(*n log n*) algorithms.

This graph depicts the time differences between algorithms in these three efficiency classes:



Let's say we have a simulation with one million agents and it takes nearly six days to execute an $O(n^2)$ event. If we can find an algorithm to execute that event in $O(n)$ or $O(n \log n)$ time, we can expect our event to execute in several seconds. This, intuitively (and with a bit of over-simplification), is why improving algorithms is often the most effective way to speed up a program.

Speeding up partner matching algorithms

Partner matching is a vital part of agent-based modelling. In a model developed by Leigh Johnson, the *MATCH* algorithm (which is a bit different and more complicated than the one described above) was at one point $O(n^2)$. Various modifications have sped it up (5). However, for my PhD I am researching generally applicable fast algorithms that other modellers can use easily.

Thus far I have identified three possibly useful algorithms. I describe here just one of them, *ClusterShuffleMatch*. The other two, *WeightedShuffleMatch* and *RandomMatchK* are described on the webpage. ClusterShuffleMatch depends on a distance measure and a cluster measure.

We define a distance measure between the agents. The smaller the distance between any two agents, the more suitable a match they are. Examples of characteristics informing the distance measure could be risk behaviour, geographic location, age and socio-economic status.

Usually when we think of distance measures, we think of Euclidean distance. There are good algorithms for finding suitable matches between points in Euclidean space. Unfortunately for technical reasons explained on the webpage, it is not usually possible to use a Euclidean metric for matching sexual partners.

We also need a clustering measure that ensures that agents with similar characteristics are clustered together. Examples of distance and cluster measures are given on the webpage.

Our algorithm works as follows: First we assign every agent a value, the cluster measure, such that agents that are likely to be matched hopefully have a similar value. Then we sort the agents into a queue based on their cluster value. We divide the agents into equally sized clusters (log n for the size of each cluster appears to be a good value in *ad hoc* testing) and shuffle them within their clusters so as to introduce stochastic behaviour into partner matching. We then look at each agent's k nearest neighbours (where k is a user-defined value, typically equal to a factor of $\log n$ from 1 to 5). The partner selected is the most suitable match determined by the distance calculation of the k neighbours of the agent under consideration.

Here is the pseudocode

```
ClusterShuffleMatch(Agents, c, k)
// Agents is an array of agents
// c is the number of clusters
// k is the number of neighbours to search
for each agent, a, in Agents
a.weight = cluster(a)
sort agents in weight order
cluster_size = number of agents / c
i = 0
for each cluster
first = i * cluster_size
last = first + cluster_size
shuffle Agents[first ... last - 1]
++i
find_best_of_k_neighbours(Agents, k)
```

Several details are omitted which can be found on the webpage.

The *ClusterShuffleMatch* algorithm is $O(n \log n)$, assuming k and c are constant factors of $\log n$. We would expect it to be much faster than *BruteForceMatch*. The question is how much quality in partner selection are we sacrificing for this increase in speed?

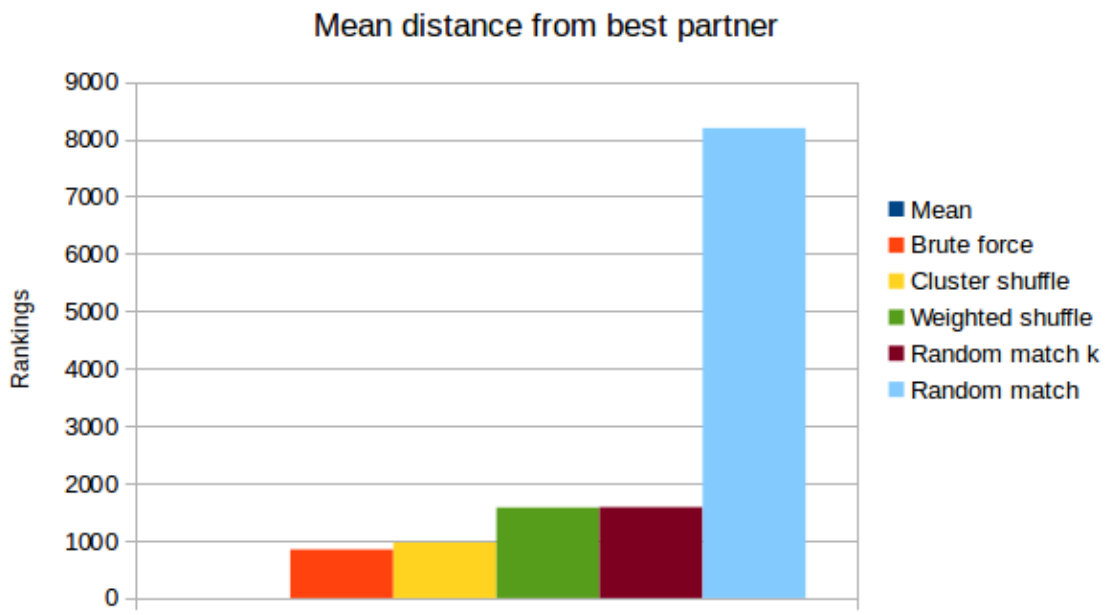
To get a preliminary answer to this question for this and the other algorithms mentioned here, they were compared for speed and suitability to an algorithm that always finds the best partner for every agent, and a thoroughly useless algorithm which merely randomly matches partners. The one that finds the very best partner is unsuitable for simulation for several reasons (it's not stochastic, partner selection is often not reciprocal and it is painfully slow). The random matching algorithm, despite its uselessness, is blindingly fast.

Hundreds of partner matching iterations were done using 16,384 agents.

This table shows the mean speeds in milliseconds of the algorithms (the perfect match algorithm is not included in this table because we are not interested in its time; it is not a suitable algorithm for simulation, but it took about 13 seconds per iteration):

	Brute force	*Random match*	*Random match k*	*Cluster shuffle*	*Weighted shuffle*
Mean (ms)	*2,337*	*2*	*20*	*21*	*22*

This is how the five algorithms compared to the perfect matching algorithm in quality of partner selection:



More details and precision are provided on the webpage.

As can be seen, *ClusterShuffleMatch*, appears to offer the best trade off in terms of speed and time (though perhaps there are practical situations where *RandomMatchK* – an algorithm not described here -- will be better). The challenge now will be to use *ClusterShuffleMatch* in a simulation of an epidemic and determine if it's much faster speed makes it a suitable replacement of slower algorithms.

This is an edited and shortened version of an article I've placed online: http://nathangeffen.webfactional.com/partnermatching/partnermatching.html.

Nathan Geffen *- PhD Student, Centre for Social Science Research, University of Cape Town. Research interests: human rights aspects of HIV and TB, as well as modelling HIV.*
nathangeffen@gmail.com

References:

1. Macal CM, North MJ. Tutorial on agent-based modelling and simulation. J Simul. 2010 Sep;4(3):151–62.
2. Orroth KK, Freeman EE, Bakker R, Buvé A, Glynn JR, Boily M-C, et al. Understanding the differences between contrasting HIV epidemics in east and west Africa: results from a simulation model of the Four Cities Study. Sex Transm Infect. 2007;83 Suppl 1:i5–16.
3. Hontelez JAC, Lurie MN, Bärnighausen T, Bakker R, Baltussen R, Tanser F, et al. Elimination of HIV in South Africa through expanded access to antiretroviral therapy: a model comparison study. PLoS Med. 2013;10(10):e1001534.
4. Levitin A. Introduction to the Design and Analysis of Algorithms. 3 edition. Boston: Pearson; 2011. 592 p.
5. Johnson L, Geffen N. A comparison of microsimulation and deterministic approaches to modelling of sexually transmitted infection dynamics. STI and AIDS World Congress, Vienna, Austria, 14-17 July 2013. Abstract P3227.