MA4J5 Project - Comparing the performance of Gillespie and HKO methods on modelling Diseases

U2009076

January 7, 2024

1 Introduction

Modelling rule-based systems in an efficient manner can be an incredibly difficult task. Many systems have large numbers of rules and variables and often require lots of linear algebra and array manipulation. In these cases, picking efficient algorithms is of great importance, such that we can run simulations in the quickest time possible.

In this short paper I will be comparing the efficiency of 2 different algorithms for simulating rule-based systems. These are the very well known Gillespie algorithm and the less well known Hu-Kang-Otter algorithms respectively.

2 The algorithms

Before we look further at the performance of these 2 algorithms, it's important we define what they are, such that there's no confusion as to methodology. Below is the Gillespie algorithm.

Algorithm 1 Gillespie algorithm

```
Require: $n \geq 0$
Ensure: $y = x^n$
1: Set $t = 0$ and calculate $\kappa_i$ and $\kappa$ for the initial stage of the system where $\kappa_i$ is the propensities of each rule and $\kappa$ is the sum of propensities of all rules
2: Set $r_1$ and $r_2$ as 2 independent uniformly distributed random numbers in the interval $[0, 1]$
3: Set $\tau = \frac{1}{\kappa} \ln r_1$
4: Set $t = t + \tau$
5: Determine which rule is to be exectued at time t by finding $\mu$ for which
       $$\frac{1}{\kappa}\sum_{l=1}^{\mu-1} \kappa_l < r_2 \leq \frac{1}{\kappa}\sum_{l=1}^{\mu} \kappa_l$$
6: Execute rule $\mu$
7: Update propensities $\kappa_i$ and $\kappa$
8: if $t < t_{max}$ then
9:     Return to step 1
10: end if
```

Below is the lesser-known HKO algorithm. The HKO algorithm has many similarities to the Gillespie algorithm, but its main difference is the implementation of subrules. The idea is that using subrules we can change the complexity of simulating the task.

Algorithm 2 HKO algorithm

```
1: Set $t = 0$ and calculate $\kappa_{i,j}$, $\kappa_i$ and $\kappa$ for the initial stage of the system where $\kappa_{i,j}$ is the propensity of subrule j of rule i, $\kappa_i$ is the sum of all the propensities of the subrules of rule i and $\kappa$ is the sum of propensities of all rules
2: Set $r_1$ and $r_2$ as 2 independent uniformly distributed random numbers in the interval $[0, 1]$
3: Set $\tau = \frac{1}{\kappa} \ln r_1$
4: Set $t = t + \tau$
5: Determine which rule is to be exectued at time t by finding $\mu$ for which
       $$\frac{1}{\kappa}\sum_{l=1}^{\mu-1}\kappa_l < r_2 \leq \frac{1}{\kappa}\sum_{l=1}^{\mu}\kappa_l$$
6: Set $r_2$ to $r_2 - \frac{1}{\kappa}\sum_{l=1}^{\mu-1}\kappa_l$
7: Determine which subrule is to be exectued at time t by finding $\nu$ for which
       $$\frac{1}{\kappa}\sum_{l=1}^{\nu-1}\kappa_{\mu,l} < r_2 \leq \frac{1}{\kappa}\sum_{l=1}^{\nu}\kappa_{\mu,l}$$
8: Execute subrule $\nu$ of rule $\mu$
9: Update propensities $\kappa_{i,j}$, $\kappa_i$ and $\kappa$
10: if $t < t_{max}$ then
11:     Return to step 1
12: end if
```

We will now have a brief look at the complexities of each algorithm before we test them. Say that for our system the maximum number of time steps is T. In our system we have R rules, each of which has α_i subrules for each rule i.

If we're using the Gillespie algorithm we just treat the rules as subrules, so we would have $\sum_{i=1}^{R} \alpha_i$ rules, giving us and order of complexity of $\mathcal{O}(T \sum_{i=1}^{R} \alpha_i)$

If we're using the HKO algorithm we have to pick from our R rules and then from one of those pick a subrule. This means that our order of complexity is at most $\mathcal{O}(T(R + \max_{i\in 1\ldots R} \alpha_i))$. This means that for many cases where we're dealing with a large amount of rules we'd expect the HKO algorithm to be far quicker.

3 The Model we're using

To Study these two methods and their relative efficiency we need to pick a model for them to simulate.

In our case we're using a 'SIRD' model to model the spread of disease. This model has 6 basic rules:

- birth

- infection
- recovery
- death of susceptible individuals
- death of infected individuals
- death of recovered individuals

In our model we also have a variety of age classes. This adds new rules for susceptible, infected, and recovered individuals where they may age. Members in one age classes will be subject to different recovery rates, different death rates, different infection rates...etc. We could treat all combinations of our base SIRD rules with different age classes each as different rules. However, these different age classes make for obvious subrules. For example in our system we may have one rule for recovery of an infected individual:

$$I \longrightarrow R$$

Then for this general rule, we have the following subrules for the different age classes:

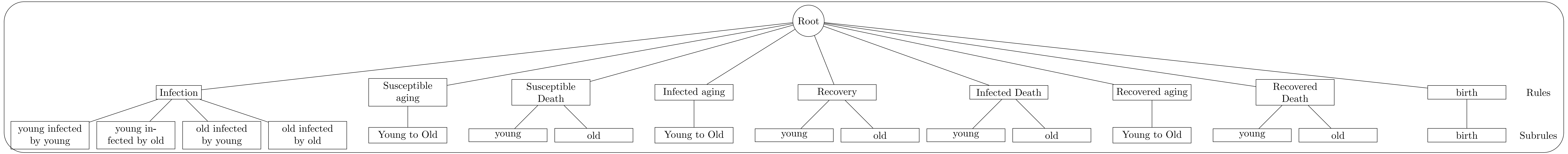
$$I_Y \longrightarrow R_Y$$

$$I_O \longrightarrow R_O$$

Where Y and O denote young and old members of the population respectively.

We can even create a visual representation of these subrules using an HKO tree. This is a Hierarchical hypergraph, with one level of the graph representing the rules and another level the subrules. In our case [1] it would be:

[1]The following figure was created with the help of [1] as a starting point


Root
Infection
Susceptible aging
Susceptible Death
Infected aging
Recovery
Infected Death
Recovered aging
Recovered Death
birth
Rules
young infected by young
young infected by old
old infected by young
old infected by old
Young to Old
young
old
Young to Old
young
old
young
old
Young to Old
young
old
birth
Subrules

This obvious use of subrules makes this version of the SIRD model very appealing to use for our testing. I have coded up the Gillespie and the HKO method with 2 and 3 age classes [2] and below you can see an example of the 3 age class HKO model running:

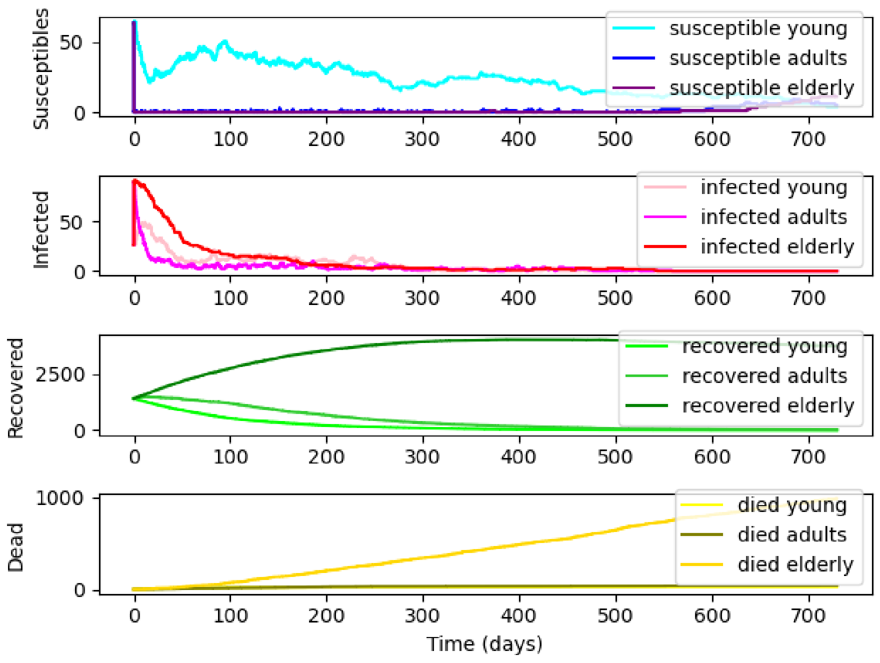


Figure 1: HKO (3 classes) population graphs

4 First testing

Our first tests [3] used either 2 or 3 age classes to test the efficiency of 2 methods. They were tested on a variety of different population numbers, as this varies the total number of rule executions required. Each time we ran the model we used a time period of 2 years. For each population size used, we ran the test 10 times and took an average. Below you can see the plots produced.

[2]You can find these 4 Jupyter notebooks attached and create similar plots for each

[3]You can recreate these tests for yourself with the 2 efficiency study Jupyter Notebooks attached. They even have an unused method (Tau Leap) one can look at

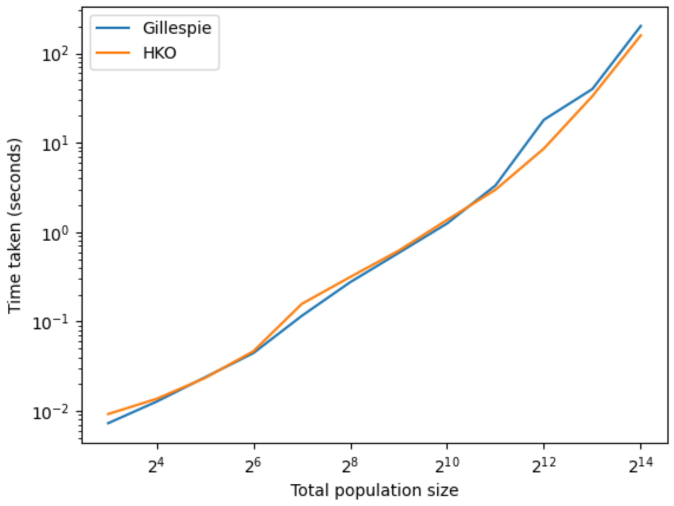


Figure 2: Model speeds with 2 age classes

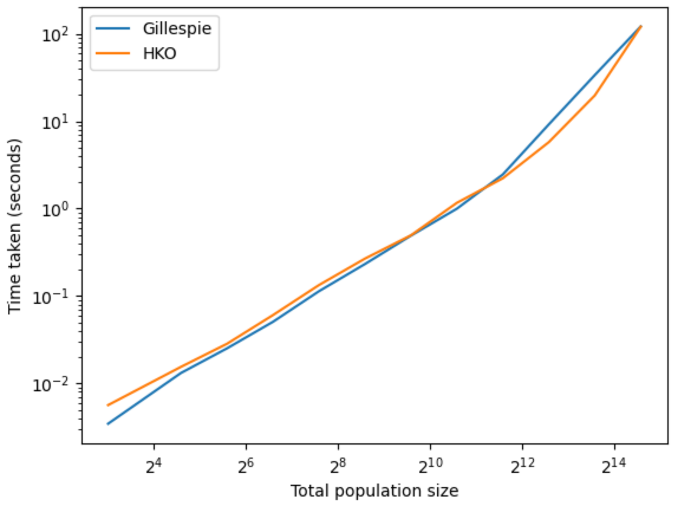


Figure 3: Model speeds with 3 age classes

You will notice in both of these cases the HKO method tends to be slower, but this difference diminishes with time. One may think this appearance is due to the logarithmic scale the graph is using, however, this is not the case. For a population of 2^{14} with 2 age classes, each run of the model took on average 202.20s for the Gillespie model and 158.89s for the HKO method. This is a very noticeable difference between the two in terms of efficiency and appears to only grow larger with larger population sizes, as for 2^{13} we have times of 39.83 and 33.20 for Gillespie and HKO respectively. However, on my machine, it's hard to get large amounts of data for these higher populations, so gathering data for a population of size 2^{15} is not possible.

What is interesting to note is that for the same population, when we have 3 age classes it seems that the HKO algorithm still reigns supreme, albeit by a very small amount - in this case it's 121.59s for Gillespie vs 120.76s for the HKO method.

On top of this, it's worth looking at the standard deviation produced. For 2 age classes note the variance of Gillespie on the population of size 2^{14} is 12.99, whereas for the HKO method it is a noticeably less consistent 14.81. For 3 age classes we have 11.75 and 11.03 as our standard deviation for Gillespie and HKO respectively. .

Either way though we can clearly see that the HKO is a more consistent method for the more complex 3 age class system and for very complex tasks appears to better the Gillespie algorithm. Though the Gillespie algorithm is better for less complex tasks.

Below are some Histograms for the data produced from trying out the 2 methods on the population of size 2^{14} for 2 and 3 age classes respectively. Hopefully these give you a better idea of the consistency of these methods.

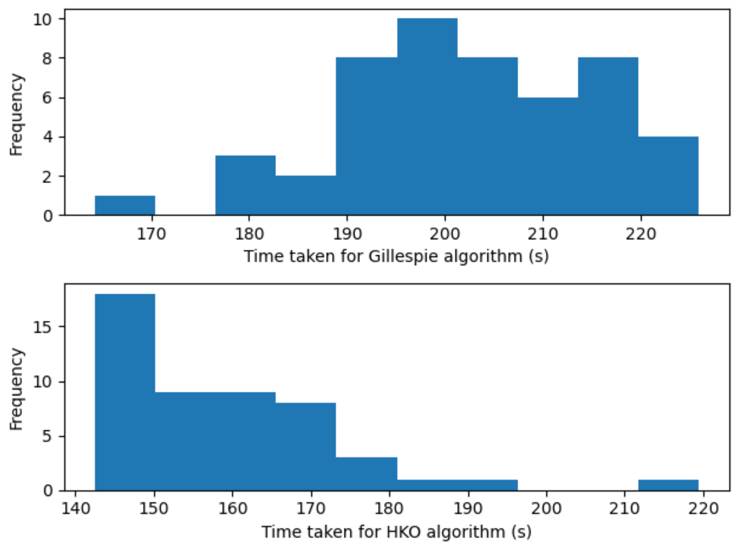


Figure 4: Histogram showing distribution of the time taken for the 2 Methods for 3 age classes

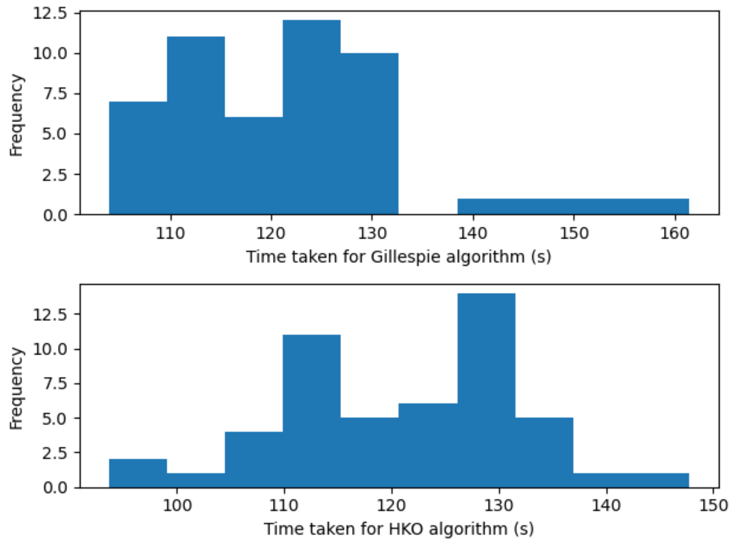


Figure 5: Histogram showing distribution of the time taken for the 2 Methods for 3 age classes

5 Conclusion

Overall we can see that although not universally an improvement, the HKO method proves itself incredibly valuable for modeling complex systems, where we can easily split rules into subrules. For very comnplex systems with carefully chosen subrules the HKO method is therefore an obvious choiec over the Gillespie method. It would be interesting to see how far ahead the HKO method pulls ahead compared to Gillespie for even larger population numbers, but doing this in a way where we can repeat the experiment many times seems impossible, given the power of my laptop.

One thing to note is how the subrules are decided and distributed could have a huge impact on the efficiency of the method. We wouldn't expect the HKO method to have any advantage over the Gillespie method if each rule was split into one subrule, as we'd just be increasing complexity of the method. Therefore the HKO method could have very different performance based on how our HKO tree is structured. In this case I tried to do it in an intuitive feeling way, however maybe a method where the amount of subrules for each rule is more consistent would be even more efficient.

6 Extension

With the notion of subrules in the stystem proving rather useful, one may wonder why we don't have "subsubrules". It would be very easy to extend our HKO algorithm to allow for this:

Algorithm 3 Deep HKO algorithm

```
1: Set $t = 0$ and calculate $\kappa_{i,j,s}$, $\kappa_{i,j}$, $\kappa_i$ and $\kappa$ for the initial stage of the system where $\kappa_{i,j,s}$ is the propensity of subsubrule s of subrule j of rule i, $\kappa_{i,j}$ is the propensity of subrule j of rule i, $\kappa_i$ is the sum of all the propensities of the subrules of rule i and $\kappa$ is the sum of propensities of all rules
2: Set $r_1$ and $r_2$ as 2 independent uniformly distributed random numbers in the interval $[0, 1]$
3: Set $\tau = \frac{1}{\kappa} \ln r_1$
4: Set $t = t + \tau$
5: Determine which rule is to be exectued at time t by finding $\mu$ for which
       $$\frac{1}{\kappa}\sum_{l=1}^{\mu-1} \kappa_l < r_2 \leq \frac{1}{\kappa}\sum_{l=1}^{\mu} \kappa_l$$
6: Set $r_2$ to $r_2 - \frac{1}{\kappa}\sum_{l=1}^{\mu-1} \kappa_l$
7: Determine which subrule is to be exectued at time t by finding $\nu$ for which
       $$\frac{1}{\kappa}\sum_{l=1}^{\nu-1} \kappa_{\mu,l} < r_2 \leq \frac{1}{\kappa}\sum_{l=1}^{\nu} \kappa_{\mu,l}$$
8: Set $r_2$ to $r_2 - \frac{1}{\kappa}\sum_{l=1}^{\nu-1} \kappa_{\mu,l}$
9: Determine which subsubrule is to be exectued at time t by finding $\xi$ for which
       $$\frac{1}{\kappa}\sum_{l=1}^{\xi-1} \kappa_{\mu,\nu,l} < r_2 \leq \frac{1}{\kappa}\sum_{l=1}^{\xi} \kappa_{\mu,\nu,l}$$
10: Execute subsubrule $\xi$ of subrule $\nu$ of rule $\mu$
11: Update propensities $\kappa_{i,j,s}$, $\kappa_{i,j}$, $\kappa_i$ and $\kappa$
12: if $t < t_{max}$ then
13:     Return to step 1
14: end if
```

To adequetely test this method, we would need to add a "subsubclass" that individuals can fall into. One obvious way of doing this would be via location. One could easily implement a patch model in a very similar way to age and risk structures and that is exactly what I shall do. To keep this from getting overly complicated to implement, I will use the model with 2 ages as a starting point and I will only have 2 locations.

On top of this with location, we have the ability to add new rules surrounding migration of individuals from one location to another. This adds even more complexity to the model, which I believe will make it even more suited to using the "Deep HKO" method.

I have coded up this "Deep HKO" method [4] and below are the plots I created from it with 4000 individuals:

[4] All this code and the plot are in the notebook "Deep HKO method (2x2 classes)"

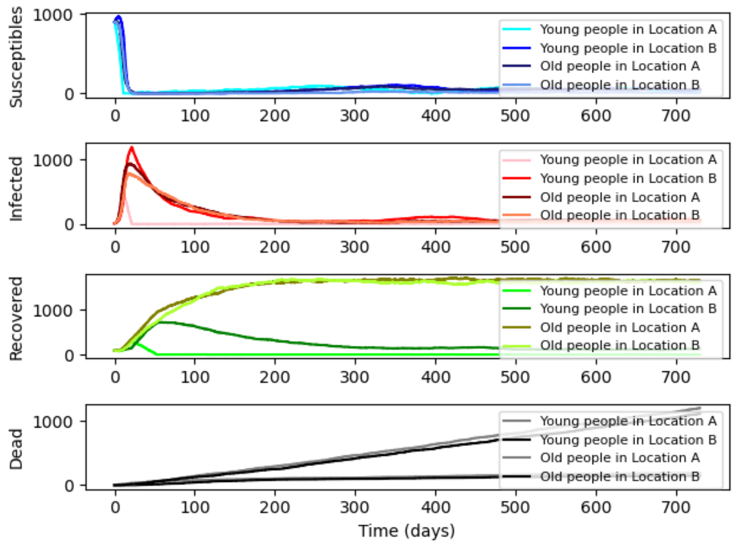


Figure 6: Plots for the Deep HKO algorithm

7 Conclusion 2

In the notebook there are more details about the coding of the method. Just to note here the method's incredibly high complexity, along with it feeling a bit beyond the scope of the project, is why I decided to not create a third efficiency study that included the "Deep HKO" method.

I would imagine performance-wise wise it would follow a similar trend to the HKO method, but being even more extreme. One would expect it to do very poorly for small populations and less complex scenarios, but once the simulation becomes very complex with huge population sizes it may start to be more efficient than even the HKO method.

However I hope the notebook proves that using this method may be feasible for other tasks in the future, but more investigation of its effectiveness is likely required.

References

[1] Manuel Kirsch. Example: Merge sort recursion tree, Dec 2009. URL `https://texample.net/tikz/examples/merge-sort-recursion-tree/`.