Unit 12 Branching Process Case Study

Armand Post, Andrew Abbott, Mustafa Sakarwala

March 24, 2018

## ABSTRACT

This project aims to explore The Monte Carlo method, simulation, and random number generation with built-in probability distribution generators. Certain computer processing tasks lend themselves to parallel execution across multiple CPUs. In such cases, it is common for such processes to create additional jobs which require the completion of their parent tasks before they can start running.

Random branching processes can be simulated fairly easily using certain parameters. The first parameter, lambda, is the rate of process creation. The second parameter, kappa, is the rate of process completion. The lifecycle of a random process can be modeled using a Poisson distribution with these two rates. One potential issue with simulations of random branching processes is that they can spawn for an infinite number of generations. In order to combat this and produce a complete family tree, constraints can be put on several parameters. The simplest method is to set a hard cap on the number of generations. Another option is to limit the maximum running time of the entire branching process tree. For the purposes of this case study, we will use the first method.

In Chapter 7 of “Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving” by Nolan and Lang, a custom graph is created to visualize a family tree. The objective of this case study is to modify that custom graph as detailed in exercise 6 of the same chapter.

## INTRODUCTION

Our aim is to create an interactive visualization to aid in the exploratory process that accompanies the creation of a Monte Carlo simulation for this problem. It will provide a convenient user experience whereby the effect of changes to the input constraints can be easily and repeatedly visualized in a branched tree diagram showing the resulting process generations, including the lifetime cycles of related birth-death relationships between each parent and child. Each result will represent a singular instance, potentially one of thousands of repetitions making up a Monte Carlo simulation. The process being studied is a variant of a branching process. An example of this is the birth and assassination process. Given various input paramaters, lambda and kappa, the above process is simulated and the number of offspring and number of generations a particular family survives is determined. When analyzing a branching process it can be helpful to use graphical plots. Being able to visually inspect the aspects of a family tree can present insights into the data that may be otherwise overlooked. In Chapter 7 of “Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving” by Nolan and Lang, a custom graph is created to visualize a family tree. The objective of this case study is to modify that custom graph as detailed in exercise 6 of the same chapter.

## LITERATURE REVIEW

## METHODS

The steps for for modeling a branching process are as follow 1.Ccreate a function that generates child processes given a parent process. We will denote birth time, completion time, rate for the inter-arrival of offspring, and a rate for the run times of the offspring with the parameters bTimes, cTimes, lambda, and kappa respectively. 2. Model the entire process. 3. Test the Model. 3) Visualize the model.

## RESULTS

The first step is is to create a function that generates child processes given a parent process. We will denote birth time, completion time, ID of the parent process, rate for the inter-arrival of offspring, and a rate for the run times of the offspring with the parameters bTimes, cTimes, parentID, lambda, and kappa respectively.

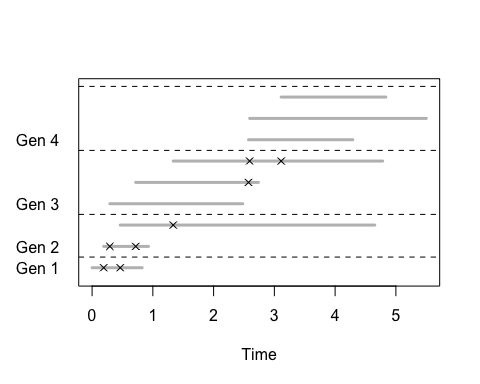
The second step is to model the entire process for which we will use the genKidsV function we created in the first step. We will avoid an infinite loop by passing a parameter to set the maximum number of generations to be modeled.

The next step is to test the model. We assign a seed value so that we can reproduce the results.

## [[1]]  
## parentID kidID births completes  
## 1 NA 1 0 0.8244658  
##   
## [[2]]  
## parentID kidID births completes  
## 1 1 1 0.1908525 0.9280697  
## 2 1 2 0.4631948 4.6525607  
##   
## [[3]]  
## parentID kidID births completes  
## 1 1 1 0.2921203 2.479967  
## 2 1 2 0.7177455 2.737410  
## 3 2 3 1.3348405 4.782500  
##   
## [[4]]  
## parentID kidID births completes  
## 1 2 1 2.574217 4.291288  
## 2 3 2 2.593132 5.500965  
## 3 3 3 3.111784 4.834751

### Alternative Visualization of a Family Tree Shown in Fig 7.4 of Data Science in R

All the data generated by the familyTree function can be analyzed using by plotting it.

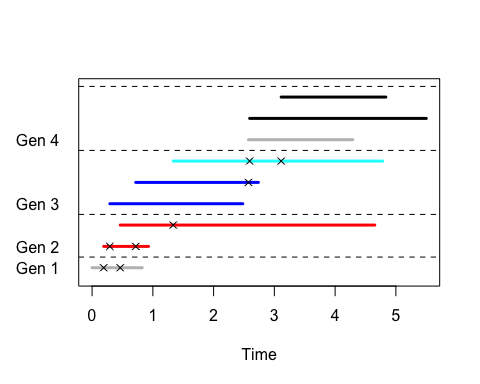


## [[1]]  
## NULL  
##   
## [[2]]  
## NULL  
##   
## [[3]]  
## NULL  
##   
## [[4]]  
## NULL

###### Figure 1 - Family Tree Graph

Figure 1 above plots the original custom graph from the Nolan and Lang book. The horizontal gray lines denote the lifetime of a process and the x’s denote the birth of child processes.

As we can see in the original graph, it is difficult to identify which parent processes started which child processes. Knowing that type of information is valuable to improve the ability to analyse the process. We can observe in the chart above that the first child born in a generation produces significantly more child processed than other processes in its generation. It might be observed that the last child process born in a generation never spawns any further child processes. In order to make it clearly visible, we will customize the visualisation by adding some code that will have different color for each line according to parentID. All processes that were spawned by the same parent process will be the same color. Since the first generation does not have a parent process, this line will remain gray. Also below we plot the same family tree in vertical format to provide a perhaps more intuitive representation of the process.



## [[1]]  
## NULL  
##   
## [[2]]  
## NULL  
##   
## [[3]]  
## NULL  
##   
## [[4]]  
## NULL

###### Figure 2 - Family tree graph color coded by Parent ID

As we see in Figure 2 above, every process within a generation is color-coded by its parent process ID. Specially looking at generation 4 we are now able to clearly identify that two processes were spawned at nearly the same time. Now it is apparent that the first process in generation 4 was spawned by the second process in generation 3 while the second and third processes in generation 4 were spawned by the third process in generation 3.

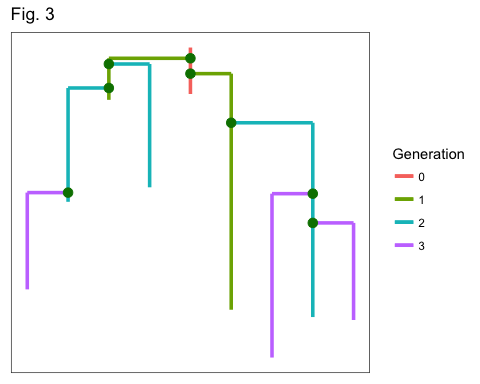
 ###### Figure 3 - Family tree graph color coded by Parent ID in vertical tree format

Figure 3 is an alternate visualization representing a complete cycle in a vertical tree format with several improvements not yet implemented in a Shiny App. It better maintains the visual relationship between parent and child process by grouping them together. Unfortunately, however, the algorithm required to determine the branch spacing is complicated to calculate dynamically. As such, this was not able to be implemented in a similar Shiny App. It remains future work to overcome that challenge.

## Conclusions and Future Work

The use of simulation, especially Monte Carlo simulation, has been increasing in recent years. Many models that are visible to the general public include at some level a Monte Carlo simulation. These include election results modeling, as well as the outcomes major sporting events. Fivethirtyeight.com’s models for both elections and for results of sports leagues include Monte Carlo simulations. The method has also been mentioned in the context of quantative finance, genetics and many other fields.

Graphical representation, charts, and visualizations improve the ability to analyze any type of process. It greatly improves ability to analyse in this type of random branching process. The original custom graph from the Nolan and Lang book goes a long way in enhancing the ability to examine the data produced by the familyTree function. However, it is difficult to distinguish which child processes were spawned by which parent processes. We can see in the modified version of the graph that colorizing a visualization can provide further insight into the data. In this specific example, the added color allows the reader to easily distinguish which child processes were spawned by which parent processes.

To improve on the current case study ,different formats such as a traditional dendrogram might reveal even more information that can’t be seen in the current graph format. The graph in this case study is useful for looking at a single family tree, but futher analysis might entail creating hundreds or thousands of family trees that would require a more aggregate analysis. More colors could be added to the charts by using advanced library like plotly to improve visualization and ability to analyse.