Efficient Implementation of the Yule-Simon Stochastic Process for Modeling Internet and Software Development Activities

Roberto Tonelli, Giulio Concas, Mario Locci
Department of Electrical and Electronic Engineering
University of Cagliari
piazza d’Armi – 09123 Cagliari
ITALY
{roberto.tonelli, concas, mario.locci}@diee.unica.it
http://www.diee.unica.it

Abstract: - We develop three different algorithms for implementing the Preferential Attachment mechanism, with regards to the Yule process, able to describe how statistical power-law distributions, for various properties of OO software systems and of the internet, are generated. Since modern software systems have reached a huge dimension, counting tens, or hundreds of thousand, of units or modules, the efficiency of algorithms for their simulation is a critical issue. We discuss their efficiency for different parameters value, their scaling with system size, and analyze in which cases one is preferable with respect to the others.


1 Introduction
Power-law distributions are among the most common distributions found in natural and human-related phenomena. They are typically found in systems sharing common features in their dynamics, like continuous growth and the insertion of new elements according to a “rich get richer” scheme. These peculiarities are typical of complex systems and are widespread and ubiquitous – computer science, mathematics, physics, biology, social networks, graph theory among many others are all fields in which power-law distributions have been found. A power-law distribution, also called Pareto distribution, Zipf’s law, or scale-free distribution, implies that, while small values are far more common than large values, the probability to find a large value is not negligible. Moreover, it is possible to find samples whose values are as large as the sum of the values of many (or most) other samples. For example, the number of citations of the most cited paper is equal to the sum of the citations of hundreds, or thousands, other papers.

In order to explain the large diffusion of power-laws in different fields, many models have been proposed [1]. Among the others, one of the most convincing is the Yule-Simon process, introduced in the twenties of last century by G.U. Yule [2] in order to explain the distribution of genera and species in nature, and used by Simon in the fifties to model the frequency distribution of words in texts [3].

More recently, the “preferential attachment mechanism” which stays at the basis of Yule process, has been reintroduced in the modeling of the WWW growth dynamics [4].

As regards software systems, many of them have reached such a huge dimension that it looks sensible to treat them using stochastic approaches. Some researchers started to scrutinize the field of software, in the perspective of finding and studying scale-free and small-world behavior [5-7]. In fact, software is built-up out of many interacting units and subsystems at many levels of granularity (functions, classes, interfaces, libraries, source files, packages, etc.), and the various kinds of interactions among those pieces can be used to define graphs that form a skeletal description of a system. Moreover, these entities are characterized by features whose distribution in turn can be studied looking for scale-free behavior.

As examples, in order to illustrate the motivations for this work and get a flavor about the importance of modeling a Yule-Simon process governed by the preferential attachment, we consider the Internet, the WWW, and software development activities. In the Internet, the various connected computers must be identified uniquely. Every node must have a single
identifier, namely the "IP address". IP addresses have a structure of four decimal numbers, separated by dots, each ranging from 0 to 255. Thus the total number of possible addresses is $2^{32}$, more than 4 billions. It has been investigated that the Internet possesses a "scale-free" structure [4], that may be explained using the preferential attachment mechanism. In this context it means that larger hubs are more likely to receive more physical links than smaller hubs.

On the World Wide Web, according to NETCRAFT (http://news.netcraft.com), “In the December 2007 survey we received responses from 155,230,051 sites”. The total amount of web pages available to net-surfers are many billions. Again the WWW, like the Internet, possesses a link structure, that may be explained using the preferential attachment mechanism [4]. Here the most visited web pages are the more likely to be linked by other web pages.

Regarding software development, we know that modern software systems can be composed by tens of thousand different files, or modules. Concas et al. already shown that many properties of object-oriented (OO) software systems follow a power-law, including the number of in-links of OO software networks, the number of times an identifier is used to name a variable or a method, the number of subclasses of a given class [6]. Moreover, in another paper Concas et al. found that Yule-Simon process can be used precisely to stochastically model the generation of these properties [8].

Now the motivations for this paper have become clearer. On one side there is the need to create mathematical models to simulate complex behaviors and to test the suitability of the model with experimental data. On the other side, when dealing with real data, the involved quantities are often huge, millions or billions. The main issue is that, at each computation step, the Yule-Simon algorithm chooses among the system “entities” – which can be very many – proportionally to the actual value of their “property” that might be incremented. This choice can thus be very critical. Real-time then may become a critical issue and a faster algorithm may make the difference among being able to simulate a real process or not, to discern among different models which provide the same power-law distribution, and to identify the relevant features and variables representative of the entire process, especially when analytical results are not available.

In this paper we propose and compare some different algorithms for simulating a Yule-Simon process, focusing on their speed even when the number of entities of the system reaches numbers of the order of millions.

The article is organized as follows. In section 2 we briefly illustrate the generalities about the Yule-Simon process and the preferential attachment mechanism. In section 3 we present algorithms to simulate the simplest case, where growth through preferential attachment is the only process at play. In section 4 we analyze algorithms able to introduce deletions of class properties. Section 5 concludes the paper.

### 2 The Yule-Simon Process

The Yule-Simon process deals with a population of “entities”, each having a property, characterized by an integer numeric value – or number of “elements” – of the property. In the original work, entities are genera, and their properties (elements) are the number of species belonging to each genus. In Simon’s work, entities are single words, and their elements are the number of times each word is used in a text. The Yule-Simon process describes a mechanism for generating such a population, with successive addition of entities, and with a rule for incrementing the property value of existing entities. The key issue is that, if the entity whose property has to be modified is chosen with probability proportional to the size of this property, the resulting property distribution will tend to a power-law.

More formally, let us consider a population of $n$ entities, each having a property with integer value, $v_i$, $i = 1, 2, ..., n$. At the beginning of the process, there are no entities. As time flows, new entities are created, and existing entities are chosen for incrementing their properties by one unit. At each time-step there is a constant probability $a$ that a new entity is created, and a probability $1-a$ that the value of an existing entity is increased by one. The average number, $m$, of property increments in between the addition of two new entities, is related to $a$ by formula:

$$m = \frac{1-a}{a} ; \quad a = \frac{1}{1+m} . \quad (1)$$

For instance, if on average four entities are chosen for adding one element to their property values in between the addition of two new entities, then $m = 4$, and $a = 0.2$. If just one entity is chosen for adding one
element, on average every other addition of new entities, \( m \) will assume the value of 0.5, and \( a = 2/3 \).

The new entities have initial value of their property equal to \( k_0 \). When an existing entity has to be incremented by one, it is chosen in proportion to its current value of \( v_i \), plus a constant \( c \), i.e., \( i \)-th entity is chosen with probability \( p_i \):

\[
p_i = \frac{v_i + c}{\sum_{i=1}^{\alpha} v_i + nc}.
\]  

That is all – the Yule process depends only upon the values of these three parameters: \( k_0 \), \( m \) and \( c \). In the original Yule’s and Simon’s models, \( k_0 = 1 \), because any new genus has a single species, and any new word appears just once in the text; moreover, in these models entities are chosen only proportionally to their values \( v_i \), and thus \( c = 0 \).

The generalized process described above can be analyzed mathematically, using the master equation approach. The analysis is reported in detail in [1], though with slightly different assumptions than Simon’s. As \( n \to \infty \), the analysis yields an exact expression for the probability \( q_{k_0} \) that an entity has property left at the initial value of \( k_0 \):

\[
q_{k_0} = \frac{k_0 + c + m}{(m + 1)(k_0 + c) + m}.
\]  

The probability that an entity has property whose value is \( k \), is given instead by the following equation:

\[
q_k = \frac{B(k + c, \alpha)}{B(k_0 + c, \alpha)q_{k_0}},
\]  

where \( \alpha \) is related to the three parameters of the process according to the following formula:

\[
\alpha = 2 + \frac{k_0 + c}{m}
\]  

and \( B(a,b) \) is Legendre’s Beta function. This function has the property that it follows a power-law for large values of either of its arguments. In our case, for large values of \( a, B(a,b) \approx a^{-b} \), and consequently, the tail of the probability distribution given by eq. (4) is \( q_k \propto k^{-\alpha} \), neglecting the term \( c \), which is small with respect to the values of \( k \) in the tail.

### 3 Implementing Yule-Simon process

The key feature for implementing the process is the preferential attachment mechanism described by eq. 2. If we define the auxiliary variable \( x_i = v_i + c \), eq. 2 becomes:

\[
p_i = \frac{x_i}{\sum_{i=1}^{\alpha} x_i}.
\]

To perform a choice among \( n \) entities with probability \( p_i \), we represent entities as the cells of an array indicated by \( x \), containing the values \( x_i = v_i + c \), where \( v_i \) is an integer denoting the number of elements. The distribution of the values inside the cells is unbalanced, due to preferential attachment that, on average, will make higher the properties \( v_i \) of the entities which were created first. For instance, when \( m \) is very large, almost all the elements will be contained in the first few cells, while the remaining cells will contain only few elements. The distribution is less skewed for smaller \( m \): for \( m \approx 0 \) very few elements will be inserted through preferential attachment in existing entities and most cells will contain just \( k_0 \) elements, presenting an almost uniform distribution.

Preferential attachment is implemented by mapping the cells to a segment of length \( \sum x_i \), with each cell \( i \) corresponding to adjacent sub-segments of length proportional to cell value \( x_i \). In order to select an entity with probability proportional to the elements it possesses we extract a random variable, \( r \), uniformly distributed with value between zero and \( \sum x_i \). The random number is mapped to a point in the segment \( S \), and the probability for this point to fall in a given sub-segment is proportional to the sub-segment size, and thus to the amount of elements of the corresponding entity. Fig. 1 shows the segment and its sub-segments.

![Fig. 1: Scheme of segment representing entities and used to implement the preferential attachment.](image)

### 3.1 Algorithm 1

The first algorithm we present is very simple and takes advantage of the properties of the power-law distribution. In this algorithm, new cells correspond-
ing to new entities are inserted in array \( x \) in the order of creation. When an entity \( i \) is chosen to have its elements incremented by one, its cell \( x_i \) is immediately available, and its value is simply increased by one. More in detail, the algorithm is:

1. \( x[1] := k_0 + c; \ i := 1; \ s := x[1]; \)
2. Extract random variable \( r \) uniformly distributed between 0 and 1;
3. If \( r < a \) then // new entity
   \( i := i + 1; \ x[i] := k_0 + c; \ s := s + x[i]; \)
4. else // element addition
   Extract random variable \( r \) uniformly distributed between 0 and \( s \);
   \( k := 1; \ s' := x[1]; \)
   while \( r > s' \) do
      \( k := k + 1; \ s' := s' + x[k] \)
   end while
   \( x[k] := x[k] + 1; \)
end if
5. If \( i > i_{\text{max}} \) END;
6. goto (2)

where \( s \) is the length of the segment, \( i_{\text{max}} \) is the maximum number of entities allowed, \( s' \) is the sum of consecutive sub-segment lengths during the computation of preferential attachment.

However, it does not easily accommodate deletion of entities, because they would imply to shift entire portions of array \( x \) to the left.

### 3.2 Algorithm 2

The second algorithm uses a standard approach for efficiently search in a list of \( p \) items, the binary search, slightly modified in order to incorporate the preferential attachment mechanism. Let consider two arrays, \( u \) and \( v \). In \( u \) we store, in the \( p \)-th cell, the total number of elements in the system up to the \( p \)-th step, namely the quantity \( \sum_{i=1}^{p} x_i \). Thus each time we add a new element in some entity or add a new entity with initial content \( k_0 + c \), we add a new cell at the end of array \( u \), whose content will be, respectively, the amount of it former last cell increased by \( k_0 \) or by \( k_0 + c \). In \( v \), at the same time, in correspondence to each cell of \( u \), we store the integer number which identifies the selected entity (that may be an already existing entity or a newly created one). Array \( u \) can be associated to a one-dimensional segment divided in sub-segments whose length is equal either to \( k_0 \) or by \( k_0 + c \). The values in the cells in \( u \) mark the points in this segment implementing the above mentioned partition. A label is associated to each piece by means of \( v \), keeping track of the related entity. The situation is illustrated in Fig 2.

The extraction of a random number, \( r \), between zero and \( \sum_{i=1}^{p} x_i \), selects one of the sub-segments with probability proportional to its size. If an entity is associated to many sub-segments, its cumulative probability to be selected is proportional to the sum of the lengths of all its sub-segments. The label attached allows to identify the entity selected through preferential attachment. A binary search is applied to the array \( u \), which is sorted. After extracting the random number \( r \), we compare it with the value of the cell in the middle of \( u \). If it is larger (smaller) than the value in the cell, we perform again the same comparison with the value in the cell in the middle of the half right (left) part of \( u \). We repeat this procedure until the same cell is found two consecutive times. This binary search requires at most \( \log_2(p) \) steps, which thus is the time associated to the execution of the preferential attachment mechanism.

![Fig. 2: Content of arrays u and v with 3 entities. Entity 1 got two increments. We assume c = 0.3 and k_0 = 1. The line in the bottom represents the associated segment.](image-url)

In the case of value of \( \alpha \) quite low, say in the range 2-3, most elements lay in a small percentage of entities, located on average in the cells with lower index \( i \). This means that most of the probability of preferential attachment is associated to very few entities, and the run through the segment to find an entity with probability \( p_i \) will in most cases end after a few steps of the \textit{while} in the above algorithm.

Thus, this algorithm is very fast when \( \alpha \) is low.
3.3 Algorithm 3

The third algorithm splits segments associated to probabilities. Any new inserted entity has starting property value \( k_0 + c \). A label, identifying the entity, is associated to the integer part \( k_p \), and is stored into an array of integers \( u \). The part \( c \) is fractionary, and stored into a real variable \( F \), containing the sum of all the \( c \)'s relative to all the entities. Since there is only one \( c \) for each entity, this sum will amount to \( Nc \), where \( N \) are the entities. When a property of an entity is incremented by \( k_p \), a new cell is allocated in the array of integers \( u \), and filled with the label of the entity. At a generic time step this array will contain, for each entity, as many labels as the amount of its property value expressed in \( k_0 \) units. The real variable \( F \) instead will contain as many \( c \)'s as the total number of entities. Let us indicate by \( K \) the sum of all the values into the array's cells. The total amount of properties into all the entities will be \( K+Nc \). Again we use segments to extract probabilities. Associated to the array there is a segment of length \( K \), made of subsegments of length \( k_p \), each labeled correspondingly to the entity it represents. Associated to the real variable there is another segment, of length \( Nc \), made of subsegment of length \( c \), ordered from the first to the last, according to the entities as they were inserted into the system. The total segments length is \( K+Nc \), namely the total amount of properties. Thus, if the entity \( i \) has property value \( x_i = j_k \) \( k_0 + c \), where \( j \) is integer, the associated probability for the preferential attachment is:

\[
p_i = x_i / (\Sigma x_i) = (j_k k_0 + c) / (K+Nc).
\]

This is proportional to the segment of length \( c \) plus all the \( j \) segments of length \( k_i \) labeled by \( i \). The probability associated to each entity is recovered as follows. We extract a random variable \( r \), uniformly distributed among zero and \( K+Nc \). If \( r \) is larger than \( K \), it identifies a point in the second segment, lying into a particular subsegment of length \( c \). The quantity \( \text{round}(r-K)/c + 1 \) then provides immediately the label of the associated entity. If \( r \) is smaller than \( K \), it identifies a point in the first segment, lying into a particular subsegment of length \( k_p \). The quantity \( \text{round}(r/k_0)+1 \) then provides the number of the cell into the array of integers \( u \), containing the label of the associated entity, which may be immediately recovered. Thus the execution time for the preferential attachment mechanism is of order \( O(1) \). The correspondence among segments lengths and entities probabilities is illustrated in Fig. 3.

![Fig. 3: Scheme of the segment splitting. Segments of length \( k_0 + c \) are split in two. The \( k_0 \) parts are labeled and their labels are stored into an array of integer. The \( c \) parts are cumulated into the floating variable \( F \), summing to \( Nc \).](image)

4 Discussion

We performed numerical simulations in order to analyze the effectiveness of the three algorithms. All the simulations were performed averaging the run times over ten independent trials, and with different choices for the parameters \( m \) and \( c \), which influence the execution time.

In the first algorithm the overall execution time grows quadratically with \( N \) but, if the power-law is strongly unbalanced, on average it will take only few steps for each search. The execution time in fact depends on the preferential attachment algorithm, which requires \( O(N) \) iterations on average, and on the number of iterations selected, which is directly proportional to the final number of entities in the system. This linear dependence is determined by the value of parameter \( m \). The combination of the two provides a run time of the order \( O(N^2) \). Fig. 4-a illustrates the results of our simulations, confirming the quadratic dependence. It is interesting to analyze also the dependence of execution time on \( m \), shown in Fig. 4-b. As predicted, this algorithm may be very effective for power-laws strongly unbalanced, namely for \( m \) large.

For the second algorithm, the binary search in the sorted array implies an execution time of the order \( O(log_2(N)) \). This, combined with the \( O(N) \) time
linearly related to the $N$ entities, provides a total run time of $O(N \log_2(N))$. In Fig. 5-a the plot of time versus the number of runs is slightly super-linear, as it should be for an $N \log_2(N)$ process. In this case the time dependence on $m$ shows a fast increase for lower $m$, while get stable for larger $m$ (Fig. 5-b). Larger $m$ determines a larger probability for adding a new element in the existing classes, thus a more frequent call to the routine implementing the preferential attachment process which is the more time consuming part.

For the third algorithm, there is still a linear dependence of the execution time on the number of iterations, which is directly proportional to the final number of entities in the system, but there is no additional time dependence on $N$, which is due to the routine implementing the preferential attachment mechanism. Thus, the total execution time grows as $O(N)$ for any choice of $m$ and $c$, as illustrated in Fig. 6-a. At the same time, the execution time increases with $m$, because the probability of adding new elements to existing classes, which is also the probability of making a call to the preferential attachment procedure, increases too (Fig. 6-b).

5 Conclusions

We described three different algorithms implementing the Yule process, and supported them by numerical simulations. We discussed main advantages and disadvantages in relation to the values assumed by the model parameters. In particular, the first algorithm is convenient when the power-law is strongly unbalanced, while the third is outperforming for large system sizes and may be easily extended in order to incorporate entities deletions and non-linearity of the preferential attachment. Since the Yule model is one of the most commonly used to produce power-laws populations in a wide variety of research fields, we hope that this work be of help in providing efficient algorithms to all interested researchers.
References: