Three efficient algorithms for implementing the preferential attachment mechanism in Yule-Simon Stochastic Process

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 describe how different optimized algorithms may be effective in implementing the preferential attachment mechanism in different cases. We analyze performances with respect to different values of some parameters related to the Yule process associated to the preferential attachment. We examine how performance scales with system size and provide extensive simulations to support our theoretical findings.

Key-Words: - **Complex Systems, Software Process Simulation, Algorithms, Modeling, Stochastic Processes.**

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 num-

ber 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 other papers Concas et al. found that Yule-Simon process can be used precisely to stochastically model the generation of these properties [7], [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 and discuss the algorithms. 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:

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

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 prop-

erty 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:

$$p_i = \frac{v_i + c}{\sum_{i=1}^{n} v_i \ + nc}. \quad (2)$$

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. For readers convenience, we briefly report the main steps.

Let us consider discrete temporal steps. These may be associated to the total number n of entities.

At each time step a new entity is introduced into the system, and m new properties are added according to the preferential attachment mechanism.

Let us indicate $p_{k,n}$ the fraction of entities which have k as porperty value when there are n entities in the system. This provides, in the limit of large numbers, the probability of having n entities with k properties.

On average, the number of such entities having k properties is $np_{k,n}$.

We now look for the probability that the next properties are added exactly to a particular entity having k_i as value of its property. This is proportional to k_i, by the preferential attachment, and after normalization this probability is:

$$p_i = \frac{k_i}{\sum_{i=1}^{n} k_i}.$$

Here the denominator is simply the total number of entities, namely $n(m+1)$.

Now, in going from step n to step $n+1$, m other entities are introduced. Thus the probability of a single entity of increasing by one its property value in this time step is $mk_i/(n(m+1))$.

The total number of entities with k properties getting a new one is:

$$\frac{mk}{n(m+1)} \times np_{k,n} = \frac{m}{m+1} kp_{k,n}.$$

Obviously the number of entities having k properties will decrease by this quantity. On the other side it will also increase, since there are entities having $k-1$ properties which get one more property, thus contributing at the next step to the entities with k properties. This quantity is calculated exactly as above, replacing k with $k-1$.

Thus we obtain a master equation for the number of entities with k properties at time $n+1$:

$$(n+1)\, p_{k,n+1} = np_{k,n} + \frac{m}{m+1}[(k-1)\, p_{k-1,n} - kp_{k,n}]$$

In the case of entities with just one property, this equation becomes:

$$(n+1)\, p_{1,n+1} = np_{1,n} + 1 - \frac{m}{m+1}\, p_{1,n}$$

For large n, when the system goes to equilibrium, we assume the probability $p_{k,n}$ be independent of n. Then the rate equation gives:

$$p_k = \frac{m}{m+1}[(k-1)\, p_{k-1} - kp_k] = \frac{k-1}{k+1+\frac{1}{m}}\, p_{k-1}.$$

Then, introducing the parameter c, this can be generalized to the cases where there is no initial value for the property.

In these cases, 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:

$$q_{k_0} = \frac{k_0 + c + m}{(m+1)(k_0 + c) + m}. \quad (3)$$

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

$$q_k = \frac{B(k+c,\alpha)}{B(k_0+c,\alpha)} q_{k_0}. \quad (4)$$

where α is related to the three parameters of the process according to the following formula:

$$\alpha = 2 + \frac{k_0 + c}{m} \quad (5)$$

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, $B(a,b) \cong a^{-b}$, 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:

$$p_i = \frac{x_i}{\sum_{i=1}^{n} x_i}. \quad (6)$$

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 $m \simeq 0$ 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 $\Sigma_i x_i$, 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 $\Sigma_i x_i$. 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.

3.1 Algorithm 1

The first algorithm we present is very simple and takes advantage of the properties of the power-law distribution. In fact most processes in nature which show a power-law distribution for the rank-frequency representation [1] have exponents in the range 2-3. This means that most of the elements lay in a small percentage of classes. Thus most of the probability of preferential attachment is associated to very few classes. In this algorithm, new cells corresponding 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.

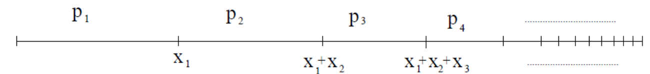

Fig. 1: Scheme of segment representing entities and used to implement the preferential attachment. x_i are the values stored in the array ***x.***

More in detail, the algorithm is:

```
(1) x[1]:= k0 + c; i := 1; s := x[1];
(2) Extract random variable r uniformly dis-
    tributed between 0 and 1;
(3) If r < a then       // new entity
       i:=i+1; x[i]:=k0 +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;
  endif
(5) If i > imax  END;
(6) goto (2)
```

where s is the length of the segment, i_{max} is the maximum number of entities allowed, s' is the sum of consecutive sub-segment lengths during the computation of preferential attachment.
In Fig. 1 s' corresponds to $x_1+x_2+...+x_n$. The values x_i are stored in **x**.

In the case of value of α 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 *while* in the above algorithm.

Thus, this algorithm is very fast when α is low. 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).

For each insertion of a new entity the sum increases by $k_0 + c$, and correspondingly the amounts in the last two cells of the array **u** will differ by $k_0 + c$. In the last cell of the array **v** we instert a new label for the new entity.

When the property of an existing entity increases, the amounts of the last two cells of the array **u** will differ by k_0. In the last cell of the array **v** we insert an already existing label.

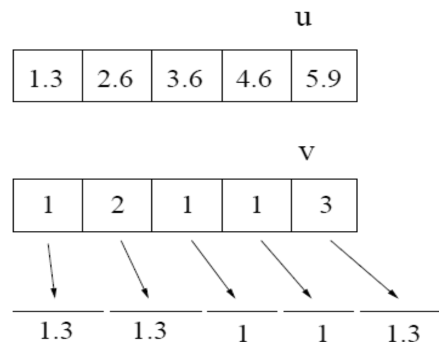

*Fig. 2: Content of arrays **u** and **v** with 3 entities. Entity 1 got two increments. We assume c = 0.3 and k_0 = 1. The line in the bottom represents the associated segments.*

Array **u** can be associated to a one-dimensional segment divided in sub-segments whose length is equal either to k_0 or to $k_0 + c$. 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.

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_0, 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_0, 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_0, 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, accordingly 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 $x_i = j_i k_0 + c$, where j_i is integer, the associated probability for the preferential attachment is:

$$p_i = x_i / (\Sigma x_i) = (j_i k_0 + c) / (K+Nc). \qquad (7)$$

This is proportional to the segment of length c plus all the j_i segments of length k_0 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 *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_0. The quantity *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.

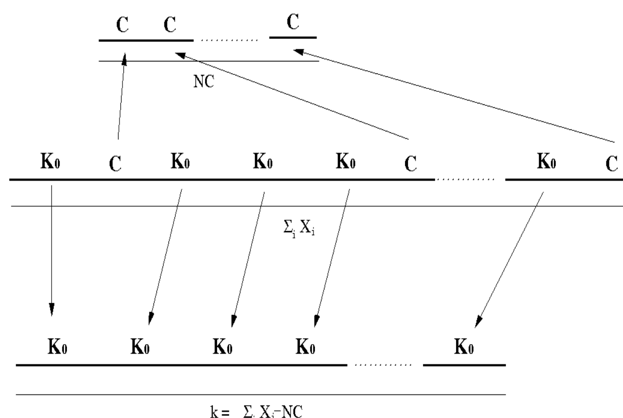

Fig. 3: Scheme of the segment splitting. Segments of length $k_0 + c$ are split in two. The k_0 parts are labeled and their labels are stored into an array of integer. The c parts are cumulated into the floating variable F, summing to Nc.

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. The values for m are 12.8, 6.4, 3.2, 1.6, 0.8, 0.4, 0.2. The values for c are -0.9, -0.6, -0.3, 0.0, +0.3, +0.6, +0.9.

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

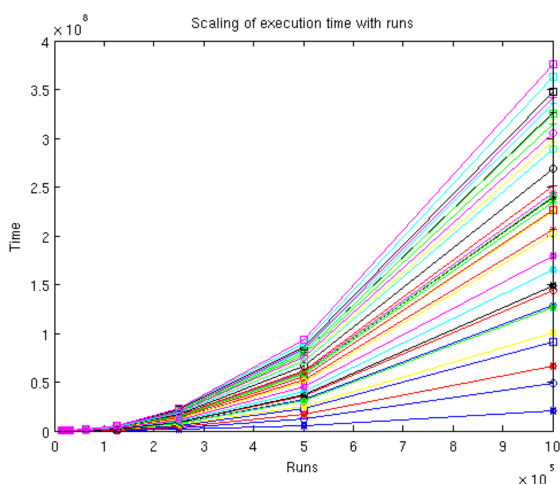

Fig. 4: First algorithm: Scaling of run-time with system size. Different colors and symbols represent different values of parameters m and c. The scaling is quadratic for any choice of these parameters.

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 illustrates the results of our simulations, confirming the quadratic dependence.

It is interesting to analyze also the dependence of execution time on m and on c, shown in Figs. 5 and 6.

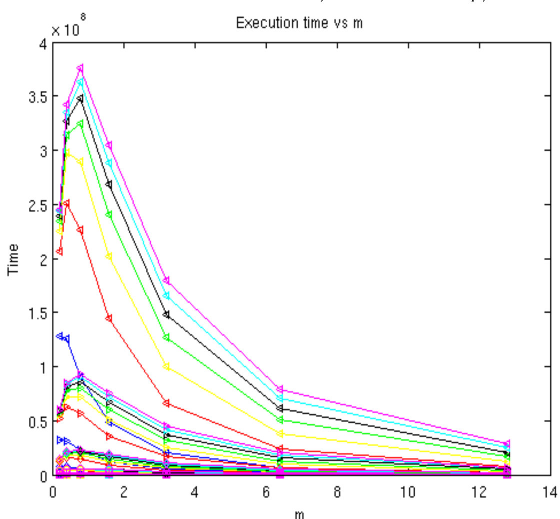

Fig. 5: First algorithm: Scaling of run-time with parameter m for fixed system size. Different colors and sysmbols correspond to different values for system size and parameter c. The behavior is the same regardless to changes in system size and c.

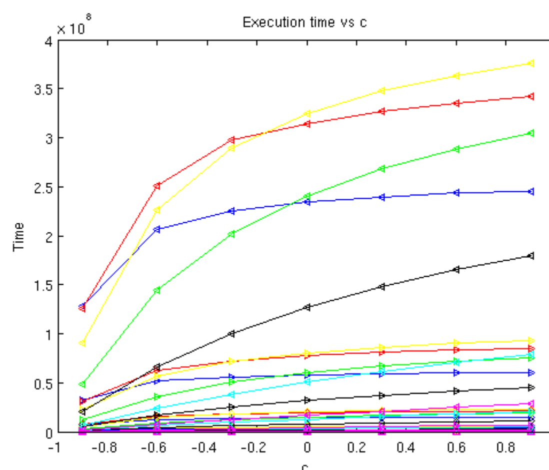

Fig. 6: First algorithm: Scaling of run-time with parameter c. Different colors and symbols represent different values of system size and parameter m. For any choice of m and system size the behavior remains the same.

As predicted, this algorithm may be very effective for power-laws strongly unbalanced, namely for m large.

Fig. 5 shows in fact that large m values have a run time an order of magnitude smaller than small m values. In fact m represents the average number of property for every entity, and it is related to the power law exponent α by the equation (5). For very large m only very few entities contain almost all system's properties. These will be, on average, among the entities created first. According to the algorithm, they will be positioned in the left-most cells of array **x**, and thus the routine for the preferential attachment will require a very short time for very large m. On the other hand, the figure shows also a fast increase for small m, and a decrease after a maximum. The complete algorithm consists actually of two separate steps. First, with probability $a = 1/(1+m)$, a new entity is created. Alternatively, with probability 1-a, an already existing entity will have its property value incremented. Thus the routine of the algorithm used to implement the preferential attachment will be activated only with probability 1-a. Otherwise, with probability a, no call to such routine is needed. In this last case there is no extra time required for searching the entity to add new properties to. For small m values, a tends to one. The preferential attachment will be activated very rarely, and the run time will be

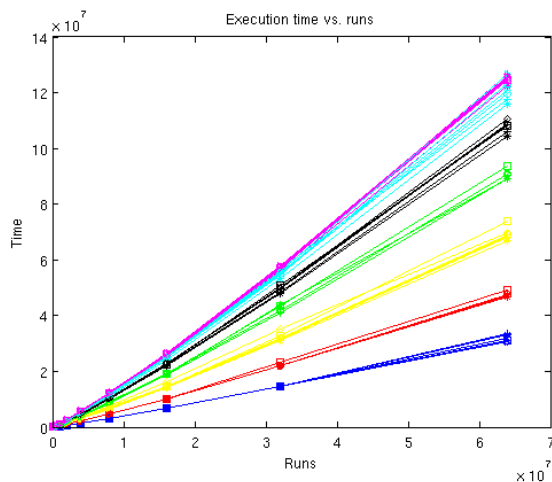

Fig. 7: Second algorithm: Scaling of run-time with system size. The scaling is slightly superlinear, as espected for the O(nlog(n)) case. Different colors and sysmbols correspond to different values for parameters m and c. The scaling remains the same for any choice of these parameters.

shorter. As *m* increases, still keeping low values, the preferential attachment routine will be called more frequently, and the run time will increase. This behavior is valid until *m* reaches a value where the second mechanism, the decrease of searching time for

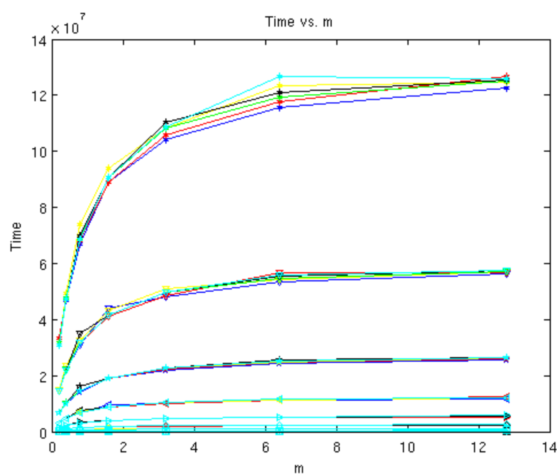

Fig. 8: Second algorithm: Scaling of run-time with parameter m for fixed system size. Different colors and sysmbols correspond to different values for system size and parameter c. For any choice of size and c there is the same initial growth, followed by an asimptotic saturation of run time for large m.

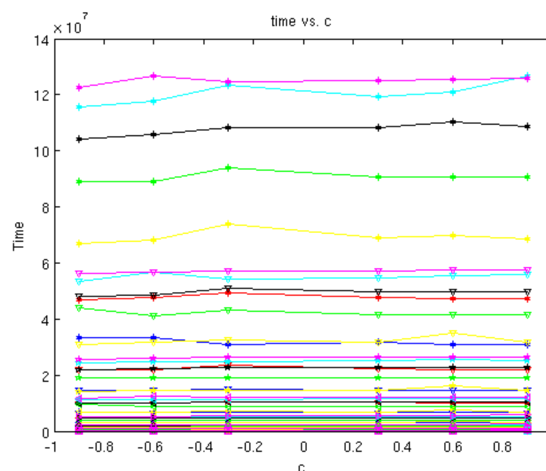

Fig. 9: Second algorithm: Scaling of run-time with parameter c. Different colors and sysmbols correspond to different values for system size and parameter m. Run time is essentially independent on c.

preferential attachment for large *m* values, will dominate. These effects together give rise to the presence of a maximum run time for some *m* value. Before this value the preferential attachment routine is rarely involved, while after this value it is mainly involved and its execution time decreases with *m*.

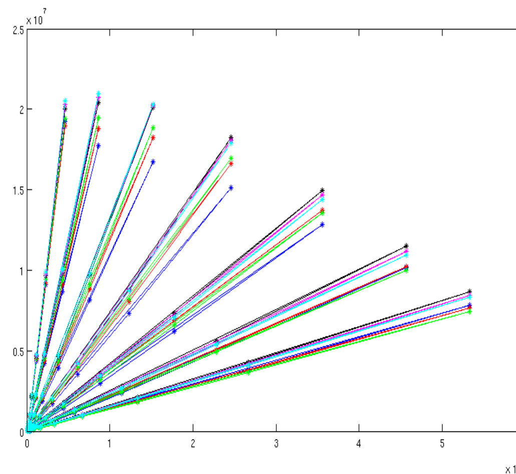

Fig. 10: Third algorithm: Scaling of run-time with system size. Different colors and sysmbols correspond to different values for parameters m and c. The scaling is linear for any choice of m and c, confirming the O(n) relationship. Parameters m and c influence the proportionality coefficient among run time and size, determining the slope of the straight lines.

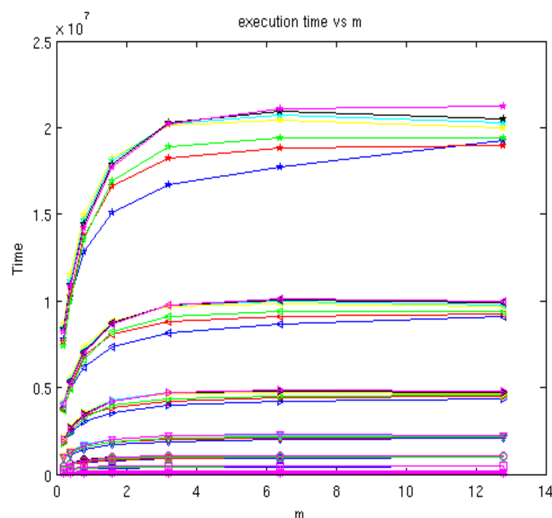

Fig. 11: Third algorithm: Scaling of run-time with parameter m. Different colors and sysmbols correspond to different values for system size and parameter c. The behavior is similar to the second algorithm. An increase of run time for small m values is followed by a saturation at large m. This behavior is the same for any choice of sistem size and c.

Fig. 6 shows that also the value of c may influence the overall run time, even if to a less extent. In general the execution time increases with c. This can be seen from eq. (5). The power-law exponent grows with c, thus c has an effect opposite to m. Since this algorithm's performance strongly relais on the bias in the distribution of properties among entities, any little change in the power-law exponent easily influences the run time.

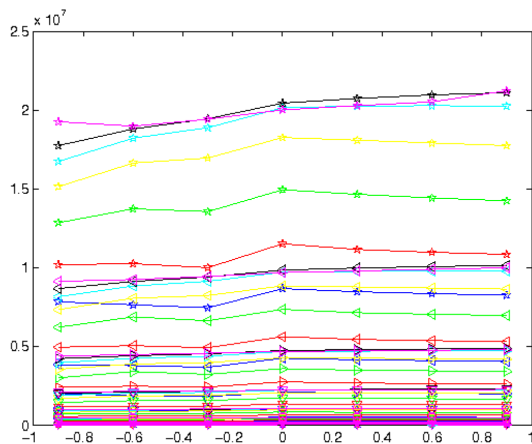

Fig. 12: Third algorithm: Scaling of run-time with parameter c. Different colors and sysmbols correspond to different values for system size and parameter m. The behavior is again similar to the second algorithm, even if a smooth increase of run time with c may be noted, regardeless of any choice for system size and m.

For the second algorithm, the binary search in the sorted array requires 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. 7 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 reaches a stable behavior for larger m (Fig. 8). 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. In this case the parameter c plays almost no role in determining the run time, as illustrated in Fig. 9.

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 10. 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. 11). Finally, the run time tends again to increase in general with c, even if the effect is clearly reduced with respect to the first algorithm (Fig. 12).

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:

[1] M. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics* **46** (2005) 323-351.
[2] G. U. Yule, A mathematical theory of evolution based on the conclusions of Dr. J. C. Willis. *Philos. Trans. R. Soc. London B* **213**, 21-87 (1925)
[3] H. A. Simon, On a class of skew distribution functions. *Biometrika* **42**, 425-440 (1955).
[4] A. Barabasi and R. Albert. Emergence of scaling in random networks. *Science* **286** (1999) 509-512.
[5] S. Valverde, R. Ferrer-Cancho, and R. Sole'. Scale-free networks from optimal design. *Europhysics Letters* **60** (2002) 512-517.
[6] G. Concas, M. Marchesi, S. Pinna, and N. Serra, Power-Laws in a Large Object-Oriented Software System, *IEEE Transactions on Software Engineering*, vol. 33, No. 10, 2007, pp. 687-708.
[7] Giulio Concas, Michele Marchesi, Sandro Pinna, Roberto Tonelli, Ivana Turnu: A Dynamic Model of Software Product Generative Process . APSEC 2008: 43-50
[8] P. Louridas, D. Spinellis and V. Vlachos, Power Laws in Software. *ACM Trans. Software Eng. and Method.*, Vol. 18, No. 1, 2008.
[9] G. Concas, M. Marchesi, S. Pinna, and N. Serra. On the suitability of yule process to stochastically model some properties of object-oriented systems. *Physica A* 370 (2006) 817-831