Atwood *et al. Computational Social Networks* (2015) 2:7
DOI 10.1186/s40649-015-0012-9

Computational Social Networks
a SpringerOpen Journal

RESEARCH Open Access

Efficient network generation under general preferential attachment

James Atwood[1]*, Bruno Ribeiro[2] and Don Towsley[1]

*Correspondence:
jatwood@cs.umass.edu
[1] School of Computer Science, University of Massachusetts Amherst, 01003, Amherst, MA, USA
Full list of author information is available at the end of the article

Abstract

Preferential attachment (PA) models of network structure are widely used due to their explanatory power and conceptual simplicity. PA models are able to account for the scale-free degree distributions observed in many real-world large networks by sequentially introducing nodes that attach preferentially to existing nodes with high degree. The ability to efficiently generate instances from PA models is a key asset in understanding both the models themselves and the real networks that they represent. Surprisingly, little attention has been paid to the problem of efficient instance generation. In this paper, we show that the complexity of generating network instances from a PA model depends on the preference function of the model, provides efficient data structures that work under any preference function, and presents empirical results from an implementation based on these data structures. We demonstrate that, by indexing growing networks with a simple augmented heap, we can implement a network generator which scales many orders of magnitude beyond existing capabilities (10^6 to 10^8 nodes). We show the utility of an efficient and general PA network generator by investigating the consequences of varying the preference functions of an existing model. We also provide 'quicknet,' a freely available open-source implementation of the methods described in this work.

Keywords: Preferential; Attachment; Network; Science

Introduction

There is a clear need for scalable network generators, as the ability to efficiently generate instances from models of network structure is central to understanding both the models and the real networks that they represent. Ideally, researchers of communication and social networks should be able to generate networks on the same scale as the real networks they study, and many interesting networks, such as the World Wide Web and Facebook, have millions to billions of nodes. Furthermore, network generation is the primary tool both for empirically validating the theoretical behavior of models of network structure and for investigating behaviors that are not captured by theoretical results. The generation of very large networks is of particular importance for these tasks because theoretically derived behavior is often asymptotic.

However, the generation of large networks is difficult because of its high complexity. In the case of preferential attachment (PA), arguably the most widely used generative model of networks, a nonlocal distribution over node degrees must be both sampled from and updated at each time-step. If we naively index this distribution, we will need to update every node at every time-step, which implies that generating a network will have complexity of at least $O\left(|V|^2\right)$.

PA models are of particular interest because they account for the scale-free distribution of degree observed in many large networks [1]. For instance, scale-free degree distributions have been observed in the World Wide Web [2-4], the Internet [5-7], telephone call graphs [8,9], bibliographic networks [10], and social networks [11].

Preferential attachment models generate networks by sequentially introducing nodes that prefer to attach to nodes with high degree. While many extensions to this model class exist, all members share the same basic form: At each time-step, a node is sampled from the network with probability proportional to its degree, a new node is introduced to the network, and an edge is added from the new node to the sampled node. This behavior has important implications for implementation. First, PA is inherently sequential, because the next action taken depends on the state of the network, and the state of the network changes at each time-step. This implies that the algorithm is not easily parallelized. Second, network nodes must be indexed such that they can be efficiently sampled by degree, and, because we are introducing a new node at each time-step, the index must also support efficient insertion. Third, the relevant distribution over nodes is nonlocal, in that the introduction of a new node and edge affects the probability of every node in the network through the normalization factor.

Much of the work in modeling network structure has focused on the asymptotic regime. A model is defined, and a limiting degree distribution (as $|V|$ approaches infinity) is obtained analytically. Less effort has focused on generating finite networks. In the following sections, we provide a robust framework for generating networks via PA. This framework easily scales to millions of nodes on commodity hardware. We also provide 'quicknet,' a freely available open-source C implementation of the framework [12].

The remainder of the paper is structured as follows. In the 'Complexity' section, we analyze the complexity of generating networks from PA models. The 'Implementation' section describes candidate methods for efficiently implementing preferential attachment generators and presents results from a simulator which implements them. The 'Applications' section describes several applications of a PA network generator which scales to many millions of nodes. We describe related work in the 'Related work' section and present conclusions and future work in the 'Conclusion' and 'Future work' sections, respectively.

Complexity

In this section, we then provide a formal definition of a PA model, then describe two existing PA models as examples. This is followed by an analysis of the complexity of generating networks from PA models.

Definitions

In this section, we provide a framework for representing general preferential attachment models. Note that the idea of a general PA model is not new to this work and that the formulation presented here is only used to facilitate algorithmic analysis. For a detailed treatment of general preferential attachment, please see 'The Organization of Random Growing Networks' by Krapivsky and Redner [13].

Let $G_t = (V_t, E_t)$ be the network that results from t iterations of a PA simulation. V_t is the set vertices (or nodes) within the network, and E_t is the set of edges between elements of V_t. Let $T(G_t)$ be the worst case time complexity of generating G_t; that is, the worst case time complexity of a preferential attachment simulation of t iterations.

Recall that the number of iterations required to generate a network with $|V|$ nodes via PA is $\Theta(|V|)$. Accordingly, we will omit t and frame our discussion of complexity $T(G)$ in terms of $|V|$.

Let $A = \{a_1, a_2, \ldots, a_{|A|}\}$ be a set of attributes that can be defined on a network node. Let $X_v = \{x_{va_1}, x_{va_2}, \ldots, x_{va_{|A|}}\} \in \mathbb{R}^{|A|}$ be a setting of A for node $v \in V$, and let $\lambda_{va_i} \in \mathbb{R}$ be the fitness of node v for attribute a_i. Let:

$$f = \left\{ f_{a_i}(x_{va_i}, \lambda_{va_i}) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+ \mid a_i \in A \right\}$$

be a set of functions, where $f_{a_i} \in f$ maps $x_{va_i} \in \mathbb{R}$ and $\lambda_{va_i} \in \mathbb{R}$ to a preference mass $\mu_{va_i} \in \mathbb{R}^+$. The 'preference mass' μ_{va_i} is a nonnegative real value that is proportional to the probability of selecting v by a_i under the PA model. We will refer to the elements of f as the 'preference functions' of the PA model. Note that, in this work, we restrict our attention to the set of degree-related attributes D (i.e., in-degree, out-degree, and total degree) with settings $x_{vd} \in \mathbb{N} \, \forall \, d \in D$. This implies that the elements of f are defined over the natural numbers:

$$f = \left\{ f_d(x_{vd}, \lambda_{vd}) : \mathbb{N} \times \mathbb{R} \to \mathbb{R}^+ \mid d \in D \right\}$$

The restriction is purely elective; any attribute with real-valued settings could be specified.

A PA model has one or more preference functions. Price's model, for example, has a single linear preference function. Krapivsky's model has two: one for in-degree and another for out-degree. A 'linear preferential attachment model' only admits linear preference functions of the form $g(x, \lambda) = c_1 x + \lambda$, a 'quadratic preferential attachment model' only admits quadratic preference functions of the form $g(x, \lambda) = c_2 x^2 + c_1 x + \lambda$ and so on.

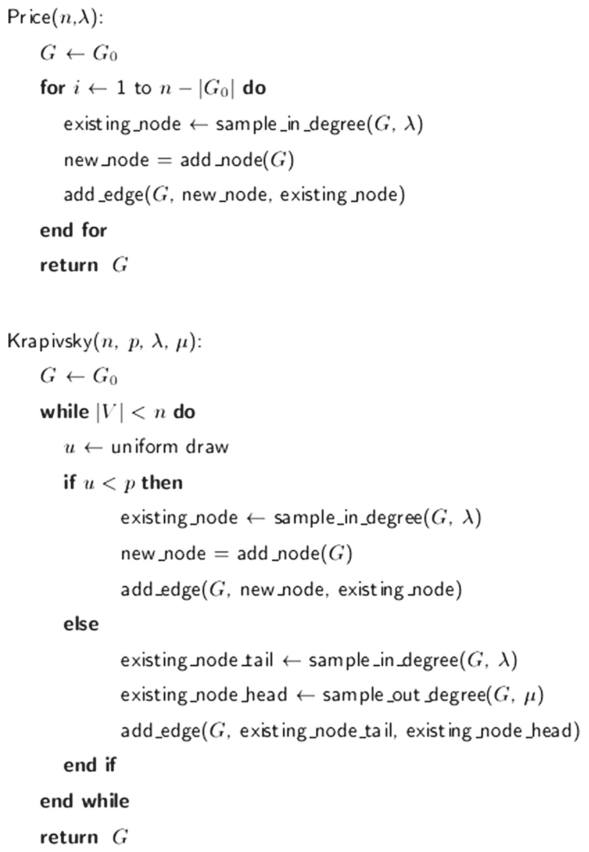
Description of considered models

Price's model

Figure 1 describes Price's algorithm. Briefly, at each time-step, a node is sampled from the network with probability proportional to its in-degree, a new node is introduced to the network, and a directed edge is added from the new node to the sampled node. Notice that a node is added at each time-step, so that the generation of a network with $|V|$ nodes takes $|V|$ steps.

Krapivsky's model

Figure 1 also describes the algorithm of Krapivsky et al. At each step, the algorithm of Price's model is followed with probability p, and a 'preferential edge step' is taken with probability $1 - p$. During a preferential edge step, two nodes, n_o and n_i, are sampled from the network by out- and in-degree, respectively, and an edge is added from n_o to n_i. Note that a node is no longer added at every step; rather, a node is added at a given step with probability p. This implies that the number of iterations required to generate a network with $|V|$ nodes is a random variable with expected value $|V|/p$. $|V|/p$ is $\Theta(|V|)$



```
Price($n$,$\lambda$):
    $G \leftarrow G_0$
    for $i \leftarrow 1$ to $n - |G_0|$ do
        existing_node $\leftarrow$ sample_in_degree($G$, $\lambda$)
        new_node = add_node($G$)
        add_edge($G$, new_node, existing_node)
    end for
    return $G$

Krapivsky($n$, $p$, $\lambda$, $\mu$):
    $G \leftarrow G_0$
    while $|V| < n$ do
        $u \leftarrow$ uniform draw
        if $u < p$ then
                existing_node $\leftarrow$ sample_in_degree($G$, $\lambda$)
                new_node = add_node($G$)
                add_edge($G$, new_node, existing_node)
        else
                existing_node_tail $\leftarrow$ sample_in_degree($G$, $\lambda$)
                existing_node_head $\leftarrow$ sample_out_degree($G$, $\mu$)
                add_edge($G$, existing_node_tail, existing_node_head)
        end if
    end while
    return $G$
```

Figure 1 Generating a network with *n* nodes under Price and Krapivsky's models. G_0 is some small seed network. λ and μ are scalers which give the fitness of nodes for incoming and outgoing edges, respectively.

$\forall p$, so asymptotically, this is no different than Price's model. More generally, the number of iterations required to generate a network with $|V|$ nodes via a PA model is $\Theta(|V|)$.

Generation complexity

We obtain a trivial lower bound on $T(G)$ by noting that, in order to generate G, we must at the very least output $|V|$ nodes, so $T(G) = \Omega(|V|)$.

A discussion of the upper bound follows. Recall that the salient problem in generating networks from a PA model is indexing the network's nodes in such a way that sampling, insertion, and incrementation can be accomplished efficiently. Tonelli et al. [14] provide a clever method for accomplishing all three tasks in constant time, provided that the preference function is linear and the fitness is both uniform across all nodes and constant. Given constant insertion and sampling times, the generation of a network with $|V|$ nodes takes $\mathrm{O}(|V|)$ time. Considering that the lower bound is $\Omega(|V|)$, we have the asymptotically tight bound of $T(G) = \Theta(|V|)$.

However, this method does not extend to nonlinear preferential attachment (see 'Related work' section for details). We can improve performance by shifting to data structures which provide $O(\log|V|)$ insertion, sampling, and incrementation, giving an overall complexity of $T(G) = O(|V|\log|V|)$.

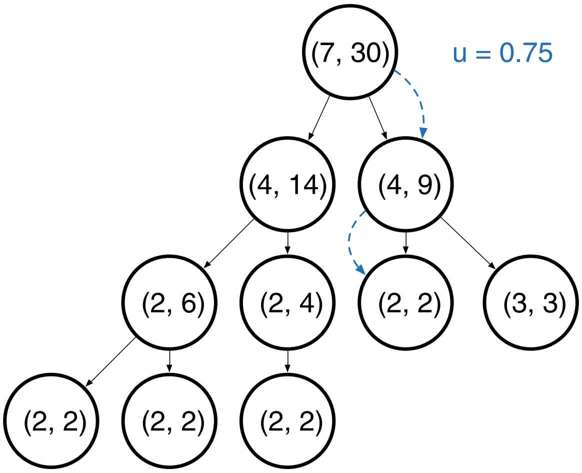


Figure 2 An example of an augmented tree structure. Each node is annotated with (μ_n, μ_s), where μ_n the node mass and μ_s is the subtree mass. The preference function associated with this tree is $f(d) = d + 2.0$. The sample path through the tree structure is illustrated for $u = 0.75$.

We accomplish this with a set of augmented tree structures. Each tree supports a preference function of the model by indexing the preference mass assigned to each node in the network by that preference function. Each item in the tree indexes a node in the network. The tree items are annotated with the preference mass of the network node under the preference function and the subtree mass, which is the total preference mass of the subtree that has the item as root; see Figure 2. Note that we refer to 'items' in the tree rather than the more typical 'nodes'; this is to avoid confusion between elements of the tree and elements of the network. We can sample from such a structure by recursively comparing the properly normalized subtree mass of a given item and its children to a uniform random draw; see Figures 2 and 3.

Note that, at each iteration of a standard PA simulation, we must sample a node, update that node's mass, and insert a new node. In what follows, we show that each of these steps can be accomplished in asymptotically logarithmic time.

Implementation

The tree structure that we described in the previous section can be implemented in a number of different ways that each have a generation time of $O(|V|\log|V|)$. They differ in their computational time for finite $|V|$. In this section, we empirically evaluate a set of realizations of the annotated tree structure. Specifically, we investigate a simple binary max-heap where priority is defined by node mass and a set of binary treaps with various sort and priority keys.

Note that, in the discussion of the heap-based and treap-based implementations of the tree structure, we will often refer to a 'sort invariant' and a 'heap invariant'. The sort invariant states that, for any three nodes $Y \leftarrow X \rightarrow Z$ where Y and Z are the left and right children of parent X, respectively, and a 'sort key' k that is associated with each item, $Y.k \leq X.k \leq Z.k$. The heap invariant states that for any three nodes $Y \leftarrow X \rightarrow Z$ (defined in the same fashion) and some 'priority key' p associated with each item, $X.p \geq Y.p$ and $X.p \geq Z.p$.

```
Sample(tree):
    sampled_node ← NULL
    $u$ ← uniform sample
    if tree.root != NULL then
        sampled_node ← SampleItem(tree, tree.root, 0., $u$)
    end if
    return sampled_node

SampleItem(item, $\eta$, $u$):
    if item.left != NULL then
        if $u$ < ($\eta$ + item.left.subtree_mass) / tree.total_mass then
                return SampleItem(tree, item.left, $\eta$, $u$)
        end if
        $\eta \leftarrow \eta$ + item.left.subtree_mass
    end if
    $\eta \leftarrow \eta$ + item.node_mass
    if $u$ < observed_mass / tree.total_mass then
        return item.node
    end if
    if item.right != NULL then
        return SampleItem(tree, item.right, $\eta$, $u$)
    end if
```

Figure 3 General algorithm to sample from the augmented tree structure. η is the mass observed thus far, and *u* is a sample from the standard uniform distribution.

We first describe the binary maximum heap. We annotate each item in the heap with a node mass, which is defined by the preference function, and a subtree mass, which is initialized to the node mass. When inserting a new item *i*, *i*'s node mass is added to all traversed items, so that the subtree mass remains accurate upon insertion. Sampling is accomplished via the algorithm of Figure 3. Node mass may only increase, so we implement an augmented version of increase-key which maintains subtree mass under exchanges; see Figure 4 for a diagram of the exchange operation. The increase-key operation supports the Increment operation, which is described below. We set priorities to be equivalent to node masses so that the most probable nodes can be accessed more quickly.

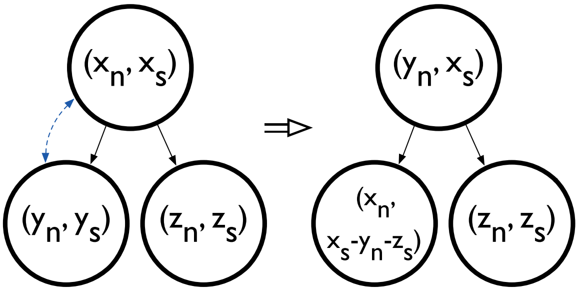


Figure 4 A diagram of the heap exchange process in the augmented heap. Each node is annotated with (μ_n, μ_s), where μ_n the node mass and μ_s is the subtree mass. Note that exchanges maintain the subtree mass invariant.

The PA process is supported by the binary maximum heap via the operations Sample, Increment, and Insert. As previously mentioned, sampling is performed via the algorithm of Figure 3. Increment increases an item's mass and then performs heap exchanges to account for any violation of the heap invariant; it is described in Figure 5. Insert adds an item to the index, appropriately updating the subtree masses of any parent items; see Figure 6.

Note that, if we were to annotate each item with a node's *probability* mass rather than *preference* mass, insertion would be a linear time operation. When a new node is introduced, the probability of every existing node decreases because the normalization factor increases. Thus, upon insertion, every item's probability mass would need to be updated. There are $|V|$ items, so insertion becomes a $\Theta(|V|)$ operation in this situation. Conversely, the preference mass of each node is unaffected by the introduction of a new node. Insertion in this scenario is a $O(\log|V|)$ operation; see Figures 6 and 7.

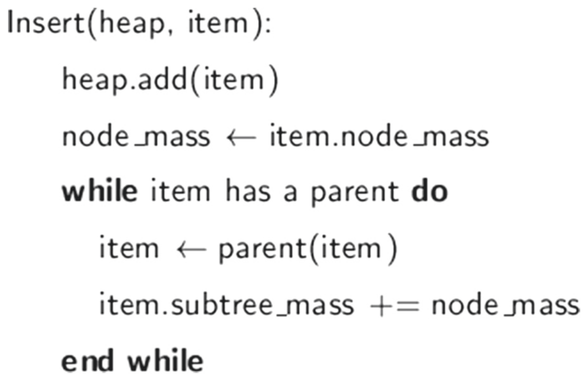
We use Price's model as an illustrative example. Recall that, in Price's model, a new node is introduced at each time-step and an edge from the new node to an existing node is added preferentially. We first identify an existing node via Sample. We then create a new node and add an edge from the new node to the existing node. Increment is called on the existing node to reflect the change in preference mass due to the new incoming edge. Finally, the new node is added to the index via Insert. Sample, Increment, and Insert are $O(\log|V|)$ operations, which implies that a single iteration is $O(\log|V|)$ and that a simulation with $|V|$ iterations is $O(|V|\log|V|)$. Generating a network with $|V|$ nodes takes $\Theta(|V|)$ iterations, so $T(G) = O(|V|\log|V|)$.

The augmented heap is implemented via a dynamic array that provides amortized constant insertion time at the cost of some wasted space. We sought to avoid this wastage by instead using some sort of binary tree, where insertion can be defined according to some ordinal value rather than an index into an underlying array. Binary treaps are an extension of binary trees that maintain a heap invariant over a random priority assigned to each item, guaranteeing that the tree is balanced in expectation [15].

The treap-based tree structure supports two operations: Insert and Sample-Destructive. Sample-Destructive is built on the Sample procedure that is given in Figure 3.

```
Increment(heap,item,new_mass):
    additional_mass ← new_mass - item.mass
    item.mass ← new_mass
    item.subtree_mass += additional_mass
    while item != heap.root && parent(item).mass ¡ item.mass do
        parent(item).subtree_mass += additional_mass
        heap_exchange(heap, item, parent(item))
        item ← parent(item);
    end while
    while item != heap.root do
        parent(item).subtree_mass += additional_mass
        item ← parent(item);
    end while
```

Figure 5 The Increment operation of the heap-based tree structure. The constant-time operation heap_exchange is demonstrated in Figure 4. The two while loops collectively over an item's $O(\log|V|)$ ancestors, so Increment is a $O(\log|V|)$ operation.



```
Insert(heap, item):
    heap.add(item)
    node_mass ← item.node_mass
    while item has a parent do
        item ← parent(item)
        item.subtree_mass += node_mass
    end while
```

Figure 6 The Insert operation of the heap-based tree structure. Note that each item has $O(\log|V|)$ ancestors, so Insert is a $O(\log|V|)$ operation.

It alters the procedure so that the sampling operation is destructive; that is, the sampled item is removed from the treap. The Insert operation inserts an item so that the sort invariant is maintained, much like one would insert an item into a binary tree. After the item is inserted, the heap invariant may have been violated, so tree rotations are performed until the heap invariant has been restored.

Figure 8 shows the empirical runtime of each of these structures as a function of generated network size. All networks were generated from Krapivsky's model. The binary heap consistently took significantly less time than the treap-based methods to generate networks of several different sizes.

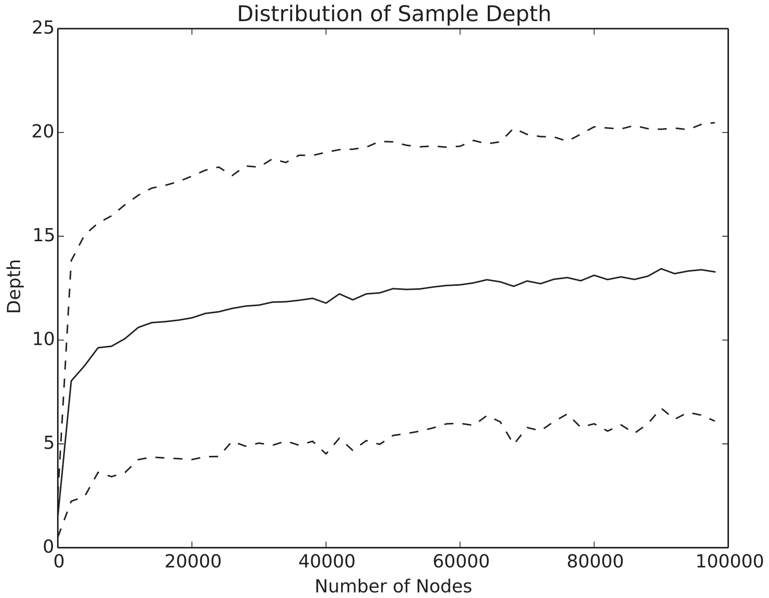


Figure 7 The empirical distribution of the depth of a sampled node within the heap as a function of network size. The solid line indicates the expected depth, and the dashed lines provide the 95% confidence interval. Note that the expected depth grows logarithmically with size of the network, which is consistent with the theoretical bounds presented in the 'Implementation' section.

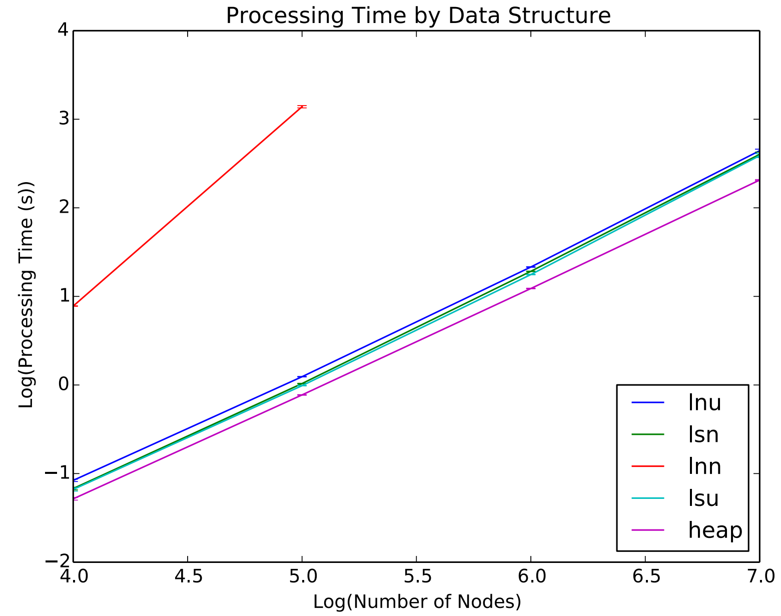


Figure 8 The empirical runtime of the simulator using different index types. The three-letter acronyms in the legend indicate the preference function type ('l' for linear), the sort key ('s' for subtree mass or 'n' for node mass), and the definition of priority ('n' for node mass or 'u' for uniform) for different variants of the treap structure. Twenty networks were generated for each configuration and size. Error bars, barely visible, indicate the 95% confidence interval.

Applications

We validate our generation model by generating sets of networks from the Krapivsky model and comparing the marginal degree distributions inferred from the generated networks with the asymptotic value predicted by the model. We then use the generator to explore some interesting questions. Specifically, we analyze the effect of changing the fitnesses of the Krapivsky model from a constant value to a random variable with various distributions. We also analyze the robustness of Krapivsky's model to superlinear preference functions.

Validating the network generator

We validate our framework by comparing the inferred exponents of the marginal distributions of generated networks with the known (theoretical, asymptotic) values for the exponents. We generated 10 networks with 10^7 nodes each. Figure 9 shows a plot of the base-10 logarithm of both degree an complementary cumulative distribution. The exponents of the marginal degree distributions were inferred via linear regression. We find, as expected, that they both exhibit power-law behavior (evident in the linearity) and that the inferred exponents of the distributions are in relatively good agreement with asymptotic theoretical values. Note that, while networks with 10^7 nodes are very large, they are still finite; we believe that this accounts for the small discrepancy between the inferred exponents and the theoretic values.

Exploring extensions to Krapivsky's model

Pareto fitness

We use our network generator to investigate the effects of altering the Krapivsky model. Specifically, we generated networks from a variant where the fitnesses assigned to each

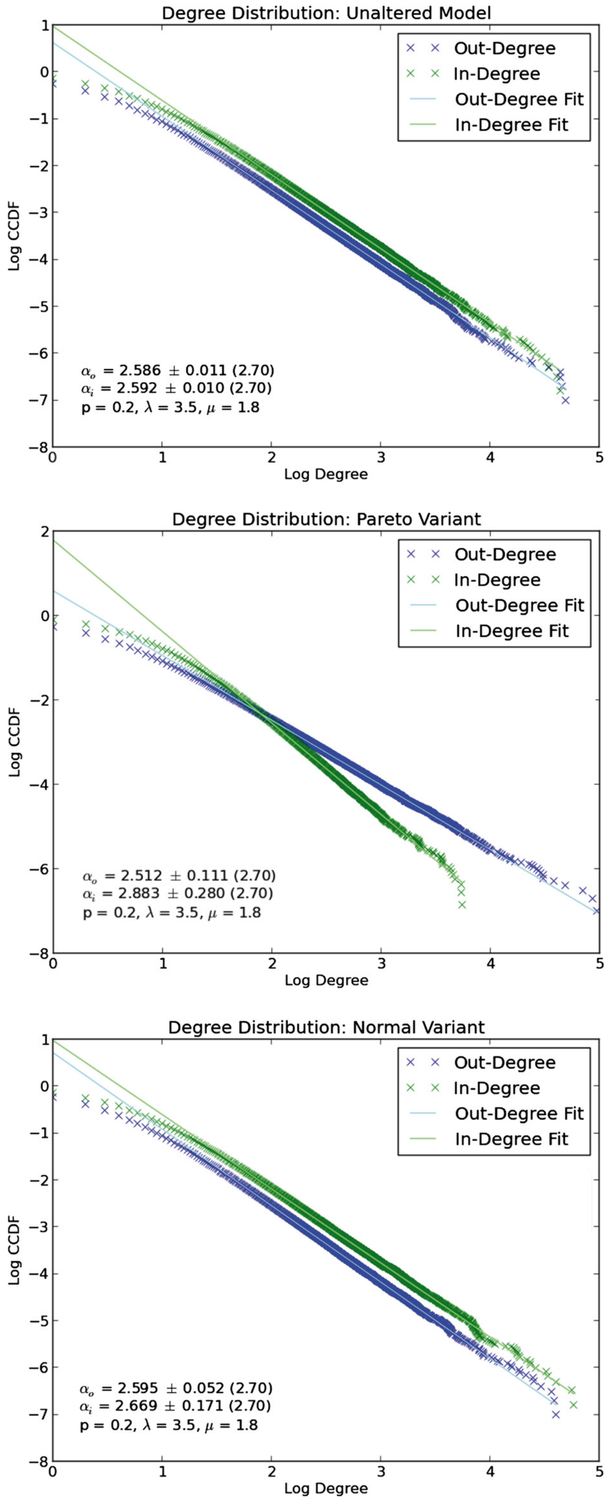


Figure 9 A comparison of the generated and theoretical marginal distributions of in- and out-degree under Krapivsky's model. The model was parameterized with $\lambda = 3.5$ and $\mu = 1.8$, and 10 networks with 10^7 nodes were generated. We plot the degree distribution of a single example network. Three variants are investigated: The unaltered model (top), a model with Pareto-distributed fitnesses (center), and a model with normally distributed fitnesses (bottom). $\alpha_O = \mu_O \pm 2\sigma_{SE}(\alpha_O^*)$ specifies the inferred exponent of the marginal out-degree distribution, where μ_O is the observed mean of the exponent, σ_{SE} is the standard error, and α_O^* is the predicted exponent for the unaltered model. The same form holds for α_i.

node were sampled from a Pareto distribution, rather than assigning the same constant value to each node. Results can be seen in Figure 9. The distribution of in-degree fitness is $\frac{\lambda d_m^{\lambda}}{d^{\lambda+1}}$ and has expected value $\frac{\lambda d_m}{\lambda-1}$. The parameter d_m is set to $(\lambda - 1)$ so that the expected value of the distribution simplifies to λ. The same form was used for the out-degree fitness. Note that this variant still exhibits scale-free behavior, that the inferred exponents are in better agreement with the predicted values than the exponents inferred from the simulation of the unaltered model, and that the variance of the inferred exponents is higher.

Normal fitness

We also simulated a variant of the Krapivsky model where fitnesses were sampled from a truncated normal distribution. Results can be seen in Figure 9. In-degree fitnesses were sampled from $\mathrm{N}\big(\lambda, (\lambda/4)^2\big)$ and out-degree fitnesses from $\mathrm{N}\big(\lambda, (\lambda/4)^2\big)$. The variances were chosen such that the probability of sampling a negative fitness is very small (less than 10^{-4}); the distributions were truncated so that any negative samples were replaced with zero. Note that scale-free behavior is still observed and that the inferred exponents of the marginal distributions of in- and out-degree are in very close agreement with the simulation of the original model.

Robustness to superlinear preference functions

Superlinear preference functions increase the strength of the ‘rich-get-richer’ effect. This can lead to situations where one node quickly overtakes all others and is thus a component of most of the edges in the network. In the extreme case, a star will form; all edges will be connected to the outlier node. We investigate the robustness of Krapivsky’s model to superlinear preference functions by plotting the ratio $\frac{d_{\max}}{|E|}$, were $d_{\max}$ is the maximum degree, as a function of the preference function exponent α; see Figure 10. $\frac{d_{\max}}{|E|}$ will approach one as the network approaches a star formation.

There is an interesting side effect to the transition from scale-free to star-structured networks. As the network becomes more star-like, the probability of selecting the most probable node tends to increase. The most probable node always sits at the top of the heap, so it can be accessed in constant time. So, the closer a network’s structure is to a star formation, the larger the probability that an iteration of a PA algorithm will be constant time. For a star structured network in the limit, every iteration will be constant time and the generation of a network with $|V|$ nodes will be $\Theta(|V|)$. This behavior is apparent for finite $|V|$; we have observed that the runtime of the generator tends to decrease as α increases.

Related work

Summary

This work is concerned with the problem of efficiently generating networks from PA models. Some examples of PA models include the models of Price (directed networks with scale-free in-degrees) [16], Barabási and Albert (undirected networks with scale-free degrees) [17], Krapivsky et al. (directed networks with nonindependent in and out-degrees which exhibit marginally scale-free behavior) [18], and Capocci et al. (like Krapivsky’s model, but with reciprocation) [19].

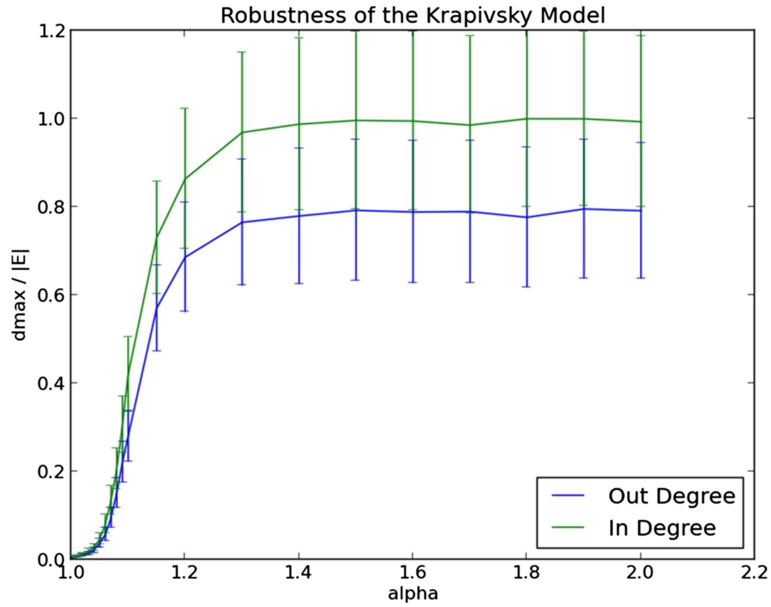


Figure 10 The robustness of the Krapivsky model to superlinear preference functions. α indicates the exponent of a preference function of the form $f(d) = d^{\alpha} + c$. $d_{\max}$ is the maximum degree of the generated network; $\frac{d_{\max}}{|E|}$ gives the proportion of edges that involve the maximum degree node. $\frac{d_{\max}}{|E|} = 1$ indicates a star formation. Note that there is a phase transition from scale-free to star-structured networks between $\alpha = 1.0$ and $\alpha = 1.2$. One-hundred networks with 10^6 nodes each was generated for each value of α. Error bars indicate the 95% confidence interval.

There has been some prior work in efficiently generating networks from PA models. Ren and Li [20] describe the simulation of a particular linear PA model, RX, but do not address the general problem of simulating networks from models with general preference functions. Hruz et al. [21] and D'Angelo and Ferreti [22] provide methods for parallelizing the simulation of linear PA but do not treat the nonlinear case. Machta and Machta [23] analyze the general case for the PRAM shared-memory parallel architecture. To the best of our knowledge, our work is the first to address the problem of efficient generation from PA models under possibly nonlinear preference functions using a sequential model of computation.

Tonelli et al. [14] provide a method for computing an iteration of the linear Yule-Simon cumulative advantage process in constant time. This method can naturally be extended to network generation through linear PA. However, the extension to nonlinear PA is very inefficient in both time and space, as shown in the next section.

Extension to nonlinear PA

Let u be an array of integers and F a real number. Consider a preferential attachment model with a linear preference function $f(d) = ad + b$, where a is the coefficient of the preferential attachment model, d is a node's in-degree, and b is a fitness value which is the same for all nodes. Assume that, like Price's model, each new node is introduced with an outgoing edge which attaches preferentially to an existing node. When a node n is inserted into the network, n's fitness is added to F, and a label identifying the node that n attaches to is appended to u. It is easy to see that the probability of selecting node i after the nth insertion is proportional to $a|d_i^n| + F$. The real number, F, can be thought of as indexing the probability mass due only to the fitness of each node in the network, whereas the array of integers, u, indexes the mass due to the degrees of nodes. Tonelli et al. provide a constant-time algorithm for sampling from this structure in their paper.

The array, u, stores a collection of integers which map to node labels. Each time a node is attached to, that node's label is appended to u. The real number, F, stores the sum of all of the individual fitnesses b. We sample from this structure as follows. Let K be the length of u, a be the linear coefficient of the process, and r be a random variable uniformly distributed on the interval $[0, K + nb]$. If $r > K$, then the quantity round $\left(\frac{r-K}{b}\right) + 1$ provides the label of the node. Otherwise, the quantity round $\left(\frac{r}{a}\right) + 1$ specifies an index into u which in turn specifies a node label. Both calculations take constant time, so sampling does as well.

Notice that this generation algorithm relies on two assumptions: the preferential attachment scheme must be linear, and the fitnesses must be the same for all nodes. To understand the first assumption, consider a model with a quadratic preference function $f(d) = ad^2 + b$. In order to index a node's transition from degree d to degree $d + 1$, we must append $(d + 1)^2 - d^2 = 2d + 1$ entries to u. Indexing a node of degree d is thus an $O(d)$ operation, and the array u requires $\sum_{v \in V} d_v^2$ entries. More generally, under a preference function of degree α, indexing a node's transition from d to $d + 1$ requires $O\left(d^{\alpha-1}\right)$ operations, and the array u will have $\sum_{v \in V} d_v^{\alpha}$ entries.

The second assumption is necessary for the real number F to map directly to a node index. The generation algorithm also assumes that the fitnesses of each node are the same. Imagine if, instead of adding each node's fitness to a real number F, we had an array z with one entry for each node, and that each entry in z contained an identifying label. We could sample a label from z in constant time simply by uniformly choosing an index into z and returning the label in z at that index. Now consider the situation where fitnesses are real-valued and not the same. We can no longer sample from z simply by sampling an index of the array, because different indices now imply different fitnesses, and thus different probability masses. We could account for this by annotating each item in z with the node's fitness value, but then, sampling would entail a search and be an $O(\log|V|)$ operation.

So, when extended to nonlinear preferential attachment, the algorithm of Tonelli et al. becomes very inefficient. Note also that the extension only holds when the preference function is a polynomial.

Conclusion

We provide an efficient framework for simulating preferential attachment under general preference functions which scales to millions of nodes. We validate this framework empirically and show applications in the generation and comparison of large networks.

Future work

We have shown that, for nonlinear preferential attachment, the complexity of generating a network with $|V|$ nodes is both $\Omega(|V|)$ and $O(|V|\log|V|)$. Future work could provide asymptotically tighter bounds. Furthermore, generation methods that focus on creating disk-resident or distributed networks could potentially scale to much larger sizes than the in-memory approach proposed in this paper.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

JA wrote the implementation and the initial manuscript under the supervision of BR and DT. JA, BR, and DT all contributed to the final manuscript. All authors read and approved the final manuscript.

Acknowledgements
This work was supported by the MURI ARO grant 66220-9902 and NSF grant CNS-1065133.

Author details
[1]School of Computer Science, University of Massachusetts Amherst, 01003, Amherst, MA, USA. [2]School of Computer Science, Carnegie Mellon University, 15213, Pittsburgh, PA, USA.

Received: 31 October 2014 Accepted: 29 January 2015
Published online: 15 April 2015

References

1. Newman, MEJ: The structure and function of complex networks. SIAM Rev. **45**(2), 167–256 (2003)
2. Barabási, A-L, Albert, R, Jeong, H: Internet: diameter of the World-Wide Web. Nature. **401**(6749), 130–131 (1999)
3. Barabási, A-L, Albert, R, Jeong, H: Scale-free characteristics of random networks: the topology of the world-wide web. Physica A: Stat. Mech. Appl. **281**(1–4), 69–77 (2000)
4. Broder, A, Kumar, R, Maghoul, F, Raghavan, P, Rajagopalan, S, Stata, R, Tomkins, A, Wiener, J: Graph structure in the web. Comput networks. **33**(1), 309–320 (2000)
5. Chen, Q, Chang, H, Govindan, R, Jamin, S: The origin of power laws in Internet topologies revisited. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, pp. 608–617. IEEE, (2002)
6. Faloutsos, M, Faloutsos, P, Faloutsos, C: On power-law relationships of the internet topology. ACM SIGCOMM Comput Commun Rev. **29**(4), 251–262 (1999)
7. Vázquez, A, Pastor-Satorras, R, Vespignani, A: Large-scale topological and dynamical properties of the Internet. Phys. Rev. E. **65**(6), 066130 (2002)
8. Aiello, W, Chung, F, Lü, L: A random graph model for massive graphs. In: the Thirty-second Annual ACM Symposium, pp. 171–180. ACM, New York, (2000)
9. Aiello, W, Chung, F, Lu, L: Random evolution in massive graphs. Found Comput Sci (2001)
10. de Solla Price, DJ: Networks of scientific papers. Science. **169**, 510–515 (1965)
11. Ribeiro, B, Gauvin, W, Liu, B, Towsley, D: On MySpace account spans and double Pareto-like distribution of friends. In: INFOCOM. NFOCOM IEEE Conference on Computer Communications Workshops, New York, USA, (2010)
12. Quicknet Implementation. https://github.com/hackscience/quicknet
13. Krapivsky, P, Redner, S: Organization of growing random networks. Phys. Rev. E. **63**(6), 066123 (2001)
14. Tonelli, R, Concas, G, Locci, M: Three efficient algorithms for implementing the preferential attachment mechanism in Yule-Simon stochastic process. WSEAS Trans. Inf. Sci. App. **7**(2), 176–185 (2010)
15. Aragon, CR, Seidel, RG: Randomized search trees. In: Foundations of Computer Science, 1989, 30th Annual Symposium on, pp. 540–545. IEEE, (1989)
16. Price, D. d. S: A general theory of bibliometric and other cumulative advantage processes. J. Am. Soc. Inform. Sci. **27**(5), 292–306 (1976)
17. Barabási, AL, Albert, R: Emergence of scaling in random networks. Science, 509–512 (1999)
18. Krapivsky, P, Rodgers, G, Redner, S: Degree distributions of growing networks. Phys. Rev. Lett. **86**(23), 5401–5404 (2001)
19. Capocci, A, Servedio, V, Colaiori, F, Buriol, L. S, al, e: Preferential attachment in the growth of social networks: the Internet encyclopedia Wikipedia. Phys. Rev. E. **74**(3), 036116 (2006)
20. Ren, W, Li, J: A fast algorithm for simulating scale-free networks. ICCTA, 264–268 (2009)
21. Hruz, T, Geisseler, S, Schöngens, M: Parallelism in simulation and modeling of scale-free complex networks. Parallel Comput. **36**(8), 469–485 (2010)
22. D'Angelo, G, Ferretti, S: Simulation of scale-free networks. In: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, p. 20. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), (2009)
23. Machta, B, Machta, J: Parallel dynamics and computational complexity of network growth models. Phys. Rev. E. **71**(2), 026704 (2005)