The Missing Models: A Data-Driven Approach for Learning How Networks Grow

Rob Patro*
Department of Computer Science
University of Maryland
College Park, MD 20742
rob@cs.umd.edu

Geet Duggal*
Department of Computer Science
University of Maryland
College Park, MD 20742
geet@cs.umd.edu

Emre Sefer
Department of Computer Science
University of Maryland
College Park, MD 20742
esefer@cs.umd.edu

Hao Wang
Department of Electrical and Computer Engineering
University of Maryland
College Park, MD 20742
hwang825@umd.edu

Darya Filippova
Department of Computer Science
University of Maryland
College Park, MD 20742
dfilippo@cs.umd.edu

Carl Kingsford†
Department of Computer Science
University of Maryland
College Park, MD 20742
carlk@cs.umd.edu

ABSTRACT

Probabilistic models of network growth have been extensively studied as idealized representations of network evolution. Models, such as the Kronecker model, duplication-based models, and preferential attachment models, have been used for tasks such as representing null models, detecting anomalies, algorithm testing, and developing an understanding of various mechanistic growth processes. However, developing a new growth model to fit observed properties of a network is a difficult task, and as new networks are studied, new models must constantly be developed. Here, we present a framework, called GrowCode, for the automatic discovery of novel growth models that match user-specified topological features in undirected graphs. GrowCode introduces a set of basic commands that are general enough to encode several previously developed models. Coupling this formal representation with an optimization approach, we show that GrowCode is able to discover models for protein interaction networks, autonomous systems networks, and scientific collaboration networks that closely match properties such as the degree distribution, the clustering coefficient, and assortativity that are observed in real networks of these classes. Additional tests on simulated networks show that the models learned by GrowCode generate distributions of graphs with similar variance as existing models for these classes.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

*Authors contributed equally.

†To whom correspondence should be addressed.

KDD'12, August 12–16, 2012, Beijing, China.

Keywords

network growth models, algorithms, graph mining

1. INTRODUCTION

The study of the processes by which biological, social, and technological networks have evolved over time has become increasingly central to gaining insight into how these networks function. Of particular importance is understanding the emergence of topological characteristics such as shrinking diameter, assortativity or disassortativity, and modularity as networks grow. One successful approach to understanding how these properties arise is the creation of idealized network models such as the forest fire model [17], the Kronecker model [16], duplication/mutation models [3, 12, 24, 26, 27], preferential attachment models [e.g. 2], and others [6, 9, 14, 15, 23]. These provide a probabilistic and mechanistic way to describe growth of particular classes of networks, generally in terms of combinations of simple operations such as node and edge creation and deletion, node duplication, node expansion (replacing a node by a subgraph), or influence propagation.

In addition to providing an idealized simulation of real-world growth, network models have a number of other uses—some of which do not even require that the programs themselves are interpretable. For example, they can serve as null models for the detection of statistically surprising topological features in a graph, can be used for large-scale performance testing for time-consuming graph algorithms, can aid in reconstructing ancient networks [21], and can help with anonymization [16].

Early theoretical work on network models began in 1960 with the Erdős-Rényi model [10]. Subsequent work identified a scale-free degree distribution [2] and small-world property [28] as common features of real-world networks and produced models that generated them. Later models incorporated other network properties as objectives in various domains, such as shrinking diameter of a growing social network [17] (the forest fire model) and clustering coefficient for biological protein interaction networks [27] (the duplication, mutation, with complementarity or DMC model). Subsequent efforts [e.g. 1, 22] have attempted to manually design models that fit various additional features simultaneously in order to produce more realistic networks. Recently, there has also been

work on models that attempt to match not only the topology of real networks, but also richer features such as node attributes [25, 13]. The creation of a parsimonious, plausible, and well-fitting growth model is typically a challenging task, and as more varied, large-scale networks are studied, new important properties will be identified, requiring new models to be developed. However, hand-crafted models can match desired topological properties only as well as the creativity and persistence of the model designer allow.

Here, we introduce a formal representation that can encode many of the most commonly studied growth models, as well as many other models yet to be discovered. We also present an optimization framework that allows for the automatic discovery of new models fitting desired properties within this formal representation. These learned models can be used to generate large classes of exemplar networks that match input features well. They are also interpretable (with some amount of effort). Often, because the framework can generally find many distinct models that match the desired properties, the set of models itself can be minded for motifs that are effective at generating a particular property. Additionally, the ease with which good-fitting models can be found can be used as a measure of the ubiquity of that feature among graph growth mechanisms. Finally, in many cases, the computationally optimized models match real-world properties better than hand-crafted models.

Very little previous work has addressed the challenge of automatic design of network models. Some previous frameworks are capable of adapting known models to new data by re-estimating parameters that govern the network growth process. For example, the Kronecker graph model [16] can be combined with a Markov Chain Monte Carlo method (KronFit [16]) to estimate its parameters in order to fit some properties for very large networks. Similar parameter estimation has been done for other recursive network models [1]. These approaches, however, are limited to parameter estimation only and cannot generate truly novel network growth mechanisms. Middendorf et al. [19, 20] address the model-selection problem of choosing from among a small number of existing models, for example, they found that protein-protein interaction networks were best fit by the DMC [27] model. However, their procedure neither generates new models nor fits parameters for existing models.

The framework we propose, GrowCode, addresses these deficiencies by representing basic random graph operations and other natural building blocks of models as instructions that operate in a register-based virtual machine. The intuitive motivation behind our framework is to provide a general and effective set of atomic operations or building blocks of network growth. A sequence of such operations defines an iteration of the network growth process, and repeated iterations of this sequence of operations evolve a network over time. We show that a small set of instructions — only 4 of which have parameters — are sufficient to describe preferential attachment, a forest-fire-like model, and a duplication model (and intuitively many other models as well). Because, additionally, the instructions operate on a simple machine with only 3 registers, this formal representation limits the search space of possible programs, allowing a genetic algorithm to search the space effectively.

We show that it is possible to quickly and automatically learn network growth models that satisfy key properties of social, technological, and biological networks using the GrowCode framework. The fit of these models to the basic topological properties of degree distribution, assortativity, and clustering coefficient is often superior to hand-crafted models. In particular, we learn a model for yeast protein interaction networks [30] that generates graphs with far better agreement to the observed values of the clustering coefficient and degree distribution than the popular duplication/mutation with complementary (DMC) model often used to simulate these networks. For a recent scientific co-authorship network [5], we are able to better match assortativity and degree distribution than graphs produced by the Kronecker model with optimized parameters [17]. Finally, for an autonomous systems internet graph, we are able to find a model that is simultaneously a much better match than a Kronecker model for clustering coefficient, assortativity, and degree distribution. The models we learn in all these settings produce graphs that are at least as diverse as those produced by the competing hand-crafted models, indicating that we are producing truly random graph models.

Although the framework we present here applies to undirected and unattributed graphs, the GrowCode approach is general, and can be easily extended to other classes of graphs as well. GrowCode also points the way to new techniques for more systematic and automatic study of network growth models themselves.

2. THE GROWCODE FRAMEWORK

We describe a novel framework, GrowCode, in which growth models may be expressed concisely and programmatically. We define a simple register machine and a set of 15 instructions that execute on it. Every sequence of instructions is a syntactically correct program that encodes some network growth model in a compact form. The instruction set contains instructions that represent specific, basic operations affecting the graph topology. There are also few instructions to manage registers and control the program flow. The instructions were selected because they are natural building blocks of growth models capable of representing a variety of extant and unknown models.

As the GrowCode machine executes a program, it modifies the topology of a growing graph. Each pass through a GrowCode program defines a single step of the growth procedure. To grow a network for t steps using the GrowCode program, the program is executed from start to finish t times. Thus, the model described by every GrowCode program is implicitly parameterized on t, which is related to the desired size of the output graph. Between subsequent growth steps (i.e. between subsequent invocations of a program), the registers of the GrowCode machine are populated randomly with nodes from the current graph. In addition to several randomized instructions, this helps GrowCode programs encode non-deterministic growth models, and different runs of the same program nearly always produce different graphs.

2.1 A register machine

GrowCode runs on a virtual machine with three registers `r0`, `r1`, `r2` that can store positive integers. The positive integer values in the registers usually correspond to vertex IDs in the graph, although they sometimes hold implicit parameters used by some instructions. The special value `NIL` in a register means that the register is empty.

The machine maintains a program counter, `PC`, indicating the currently executing instruction. After an instruction is executed, `PC` is incremented so that the program is executed sequentially unless one of the instructions responsible for control flow manipulates the `PC`. Programs can be self-modifying in a very limited way to support looping (see the REWIND instruction below). Program execution terminates once the location of `PC` has exceeds the length of the program.

Additionally, the machine has a limited memory $\mathtt{L} : V \to V$ that can store a single vertex ID associated with each vertex in V, the set of vertices in the growing graph. The value $\mathtt{L}(v)$ on node v need not be the vertex ID of v, but rather can be the ID of some other node in V. This allows programs to mark nodes with IDs of arbitrary other nodes in the graph, which is how programs can spread the influence

Table 1: Complete GrowCode instruction set

Name	Description
NEW NODE	create new node
CREATE EDGE	create new edge
RANDOM NODE	pick random node
RANDOM EDGE	pick random edge
INFLUENCE NEIGHBORS(p)	label neighbors with u
ATTACH TO INFLUENCED	add edges to neighbors labeled u
DETACH FROM INFLUENCED	remove edges to neighbors labeled u
CLEAR INFLUENCED	clear all labels from L
REWIND(r, i)	jump back r positions i times
SKIP INSTRUCTION(p)	skip next instruction
SET(i)	copy node ID to r2
SAVE	copy r0 to r2
LOAD	copy r2 to r0
SWAP	swap r0 and r1
CLEAR r2	set r2 to NIL

of a node in the graph (see section 2.2). If $L(v) = \texttt{NIL}$, then v is considered to have no label. Other, more sophisticated memories or influence instructions are possible, but the experiments below indicate that this minimal scheme is all that is required to achieve good agreement in several settings.

2.2 An instruction set

Instruction set design.

Design of instruction sets for physical and virtual processors is a difficult, long-standing problem. Operations in the GrowCode instruction set were selected so that they are representative of the basic network operations. Individual instructions are easily interpretable and are similar to those used by the hand-created growth models. Intuitively, combinations of these instructions can produce growth models that can generate networks with the desired properties. The combination of instructions used here is but one example among many possible instruction sets. This set of instructions can be extended to include other instructions to accommodate new growth processes. A good set of instructions makes it much easier to optimize a difficult objective [4]; however, it is out of the scope of this paper to completely resolve the problem of instruction set design. Rather we provide evidence that one instruction set (Table 1) works well for several common classes of graphs.

GrowCode instructions.

The GrowCode instructions (Table 1) can be subdivided into 4 categories: (1) graph operations, (2) influence operations, (3) control flow operations, and (4) register manipulation operations. The first two sets of operations deal with modifying the topology of the growing graph while the 3rd and 4th sets of operations deal with managing the control flow of the GrowCode program and the state of the GrowCode machine. See Table 1 for a complete list of instructions. Below, we describe how each of these affects the state of the GrowCode machine and the graph being generated. In section 3, we show how several network growth models can be expressed with these instructions.

Graph operations. The graph operations perform basic modifications of graph topology. The NEW NODE operation creates a new node in the growing graph with a unique ID that is placed in register r0. The CREATE EDGE operation is used to introduce a single new edge in the graph. The machine first fetches the nodes from r0 (u) and r1 (v) and then creates a new edge $\{u, v\}$ in the graph. The state of the registers after a CREATE EDGE operation remains unchanged, so if such an edge already exists, the operation has no effect. The RANDOM NODE operation selects a node uniformly at random from the current graph and places this node into r0. Finally, the RANDOM EDGE operation selects an edge $\{u, v\}$ uniformly at random from the current graph, and places u in r0 and v in r1.

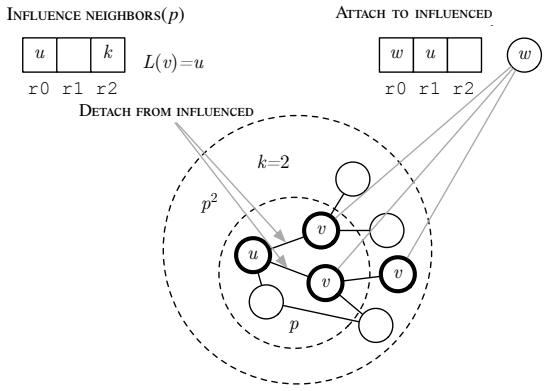

Figure 1: Schematic of the three influence operations. First node u influences its neighbors with probability p, then the influenced neighbors v influence their neighbors with probability p^2. If u were to detach from its influenced neighbors the two edges indicated by the gray arrows would be removed from the graph. Finally, w can attach to all nodes influenced by u.

Influence operations. The influence operations allow nodes to exert an influence on other nodes in the graph. We say that a node v is influenced by u if $L(v) = u$. This concept of influence is important to produce graphs with properties such as homophily where, to varying degrees, connected nodes share topological neighborhoods. The core influence operation, INFLUENCE NEIGHBORS(p), allows a node to influence a subset of its neighborhood. To execute the INFLUENCE NEIGHBORS(p) operation, the machine fetches the node ID from r0; this node, u, becomes the central, or *influential* node. It then assigns the mark u to every neighbor v of u by setting $L(v) = u$ independently with probability p. Each newly marked node, v, in turn marks its neighbors with the value u with probability $p^{d(u,v)}$, where $d(u, v)$ is the shortest path distance between u and v. If r2 is not NIL, only nodes v such that $d(u, v) < \texttt{r2}$ can potentially be affected by the influence operation. If $\texttt{r2} = \texttt{NIL}$, the influence operation continues until the probabilistic process dies out and no more nodes are marked.

The INFLUENCE NEIGHBORS(p) operation works in conjunction with three other influence operations. ATTACH TO INFLUENCED creates edges between the node in r0, w, and all nodes in the graph marked with the value in $\texttt{r1} = u$. That is, it creates edges $\{w, v\}$ for all v such that $L(v) = u$. This provides a general mechanism to make the neighborhoods of two nodes more similar to each other. The DETACH FROM INFLUENCED operation fetches a node u from r0 and removes all edges $\{u, v\}$ from the graph where $L(v) = u$. Finally, the CLEAR INFLUENCED operation erases the contents of L so that future operations can work with a clear memory. Figure 1 illustrates these influence operations.

Control flow operations. The control flow operations alter the order of execution of GrowCode instructions. The REWIND(r, i) instruction allows for loop-like structures in GrowCode programs. The first argument r is a natural number specifying the number of times PC should be decremented when the instruction is executed

Algorithm 1 Barabási-Albert

1: NEW NODE	▷ Create a new node, u
2: SAVE	
3: RANDOM EDGE	▷ Choose a random edge, e
4: SKIP INSTRUCTION(0.5)	▷ Choose random endpoint v of e
5: SWAP	
6: LOAD	
7: CREATE EDGE	▷ Create an edge between v and u
8: REWIND(5, i)	▷ Attach it to i random nodes

(i.e. how far the PC should jump backwards). The second argument i specifies the number of times the instruction should be executed. Each time the instruction is executed, the instruction is modified by decrementing the value of i by 1. When $i = 0$, the instruction will no longer be executed and the value of PC will not be rewound. The parameters of the rewind instruction are reset between consecutive program executions. The other instruction in this group, SKIP INSTRUCTION(p), advances the PC by a value of 2 with probability p. This allows the next instruction to be conditionally executed with probability $1 - p$.

Register operations. Finally, the register operations allow one to manipulate the 3 registers directly. The SET(i) instruction assigns integer i to `r2`. The SAVE operation copies the contents of `r0` into `r2`. Conversely, LOAD places the contents of `r2` into `r0`. The SWAP operation swaps the contents of `r0` and `r1`. Finally, the CLEAR `r2` sets the contents of `r2` to `NIL`.

3. REPRESENTING EXISTING MODELS

To demonstrate the generality of the GrowCode instruction set, we show how it can be used to express three existing network growth models, Barabási-Albert (B-A) [2], duplication and mutation with complementarity (DMC) [27], and forest fire (FF) [17] by writing hand-coded GrowCode programs that match the properties of these models. These growth models match different classes of real-world networks, and they exhibit different topological qualities. For example, the DMC model (with the appropriate parameters) produces graphs with a range of clustering coefficients that match those observed in protein-protein interaction networks, while the FF model produces graphs that exhibit shrinking diameter and a densification power law property as they grow. Despite significant differences in the mechanisms they model and the graph properties they produce, there are fairly simple GrowCode programs capable of representing each of these models while using the same set of primitive instructions. The instructions are re-used in different models which indicates their overall utility in expressing different network growth behavior.

3.1 Barabási-Albert

In the B-A model, new nodes added to the growing network are more likely to connect to existing high-degree nodes [2]. This process reproduces the scale-free degree distribution often found in real-world networks, where there are relatively few nodes having a very high degree and a long tail of low-degree nodes.

The GrowCode program in Algorithm 1 closely simulates the B-A model. While there are subtle differences between the program and the model, the graphs generated by the program match those produced by B-A closely. The essence of the B-A model is encoded in lines 3–5. The RANDOM EDGE instruction (line 3) picks an edge that is likely to have a high-degree node as one of its endpoints. The probability that a randomly chosen edge e contains the node u is directly proportional to the degree of u: $d(u)/|E| = 2d(u)/\sum_{v \in V} d(v)$. Once e is selected, instructions 4 and 5 choose an endpoint for this edge at random to ensure that there is no bias when selecting a node within the edge. This process selects nodes proportional to their degree as desired with the minor difference that the degrees $d(u)$ are changed as the program executes, in contrast to B-A . The newly added node is then connected to u (line 7) completing the preferential attachment of the new node. The rewind on line 8 iterates this procedure so that we connect the new node to i existing nodes.

Algorithm 2 DMC

1: RANDOM NODE	▷ Put a random node v in `r0`
2: SET(1)	▷ Set `r2` (k-hop for influence) to 1
3: INFLUENCE NEIGHBORS(1.0)	▷ Influence v's neighbors
4: SWAP	▷ Swap `r0` and `r1`
5: NEW NODE	▷ Create a new node u and put it in `r0`
6: ATTACH TO INFLUENCED	▷ Connect u to influenced nodes
7: CLEAR INFLUENCED	
8: INFLUENCE NEIGHBORS($q_{\text{MOD}}/2$)	▷ Influence u's neighbors
9: SWAP	
10: INFLUENCE NEIGHBORS($q_{\text{MOD}}/2$)	▷ Influence v's neighbors
11: DETACH FROM INFLUENCED	▷ Actually delete edges from v
12: SWAP	
13: DETACH FROM INFLUENCED	▷ Do the same for u
14: CLEAR INFLUENCED	
15: SKIP INSTRUCTION($1.0 - q_{\text{CON}}$)	▷ Skip adding $\{u, v\}$
16: CREATE EDGE	▷ with probability q_{CON}

3.2 Duplication and Divergence

The duplication and mutation with complementarity model [27] (abbreviated DMC) aims to reproduce the topological characteristics of protein interaction networks. Under the DMC model, the driving mechanism of network growth is the duplication of existing nodes. The model has two parameters, q_{MOD} and q_{CON}, that govern the process as follows. Each new node u chooses an anchor v and attaches to all of v's neighbors. Then, for each node w now adjacent to both u and v, an edge is randomly chosen that connects w either to u or v, and the edge is removed with probability q_{MOD}. Finally, the edge $\{u, v\}$ is added with probability q_{CON}. This mechanism of growth is motivated by the common occurrence of gene duplication, wherein genes, the precursors of proteins, are commonly copied within the genome. Initially, the copied genes are exact duplicates, and therefore the resultant proteins maintain the same set of interactions as the original protein. However, after duplication, evolutionary pressure on genes to maintain the interactions is reduced, and the interaction patterns between the original and copied genes start to diverge. Algorithm 2 gives a close approximation to the DMC model in GrowCode.

The DMC model presented in Algorithm 2 differs slightly from that introduced by Vazquez et al. [27] in that we cannot precisely mimic the procedure of selecting shared neighbors of u and v with probability q_{CON} and then deleting the edge to one or the other. However, to achieve a similar effect, we can influence the shared neighbors of each node with probability $q_{\text{MOD}}/2$ (lines 8 and 10) after we have copied v's neighborhood to u. If the set of influenced neighbors is disjoint, then the influence instruction has exactly the same effect as the traditional DMC operation. It is possible that a neighboring node will be influenced by both u and v. Complementarity (the fact that only the edge to u or v is removed, and not both) is maintained in this case as well since the mark on the shared node will be overwritten, ensuring that only one of $\{u, w\}$ and $\{v, w\}$ can be deleted. This procedure can result in values of q_{MOD} having a

Algorithm 3 FF

```
1: RANDOM NODE                          ▷ Put a random node in r0
2: CLEAR r2                             ▷ Clear r2 to allow full graph influence
3: INFLUENCE NEIGHBORS(b)               ▷ Breadth-first recursive influence
4: SWAP                                 ▷ Move the random node into r1
5: NEW NODE                             ▷ Create a new node, u
6: CREATE EDGE
7: ATTACH TO INFLUENCED                 ▷ Connect u to influenced nodes
```

slightly different effect in the GrowCode program as compared to the original DMC model. However, we have verified that graphs generated by GrowCode DMC and the original DMC have similar Zipf plots (through visual inspection) and clustering coefficients (section 5.3), which are the two features on which the authors of the DMC model focused. Further, Algorithm 2 produces graphs with similar Zipf plots and clustering coefficients as those observed in the yeast protein interaction network. Thus, despite the subtle differences, the GrowCode algorithm 2 maintains the essential characteristics of the original growth model.

3.3 Forest fire

The forest fire (FF) model [17] was first introduced to model scale-free degree distributions (of both in-degree and out-degree) as well as shrinking diameter and densification over time (under certain parameter regimes). The FF model is very intuitive and easy to explain from the perspective of network growth. We present a model that has been slightly altered to apply to undirected networks. When a new node u enters the network, it chooses an existing node v uniformly at random to act as an *ambassador*, and the edge $\{u, v\}$ is added to the network. Next, a number n is drawn from a geometric distribution with probability b of success, and n neighbors of v are chosen to be *burned*. An edge is added from u to each of these burned nodes, and the process of selecting a set of neighbors and burning them is repeated recursively.

Algorithm 3 shows a GrowCode program that encodes the forest fire model. We observe that it produces networks with the same essential characteristics as those produced via the forest fire model. In particular, the networks produced by algorithm 3 exhibit (for certain parameter ranges of b) shrinking diameter and densification power law during network growth.

4. LEARNING GROWCODE MODELS

One of the benefits of expressing growth models as a set of instructions is that we can now formalize the problem of learning a growth model as an optimization problem over the space of GrowCode programs. Given a set of graph features, we use genetic programming techniques to learn a GrowCode program that produces graphs closely approximating these features. These graph properties are encoded into the fitness function of an individual GrowCode program. The goal of our learning procedure is not to recover previously proposed growth models, but rather to learn programs that grow graphs that are representative of a particular class of graphs as measured under specific similarity measures.

4.1 Constructing a fitness function

We define a feature collection $\mathbf{x} = [x_1, x_2, \ldots, x_m]$ to be a m-long vector of features where each property x_i may be a scalar (such as clustering coefficient) or a vector (such as a sampling of the graph's effective diameter during its growth process). The goal of the feature collection is to represent the essential graph characteristics that we want our growth model to match. We define a (possibly weighted) similarity measure between any two feature collections $s(\mathbf{x}^i, \mathbf{x}^j)$, which are of the same size, as:

$$s(\mathbf{x}^i, \mathbf{x}^j) = \sum_{\ell=1}^{m} w_\ell s_\ell(x_\ell^i, x_\ell^j), \tag{1}$$

where $s_\ell(\cdot, \cdot)$ is a user-defined measure of similarity between the ℓ^{th} features of the collections. This measure of similarity can be as simple as the inverse of the difference between the two features (e.g. for scalar features), or it could be as complex as a measure of the similarity of distributions (for more complex features). The only requirement on $s_\ell(x_\ell^i, x_\ell^j)$ is that it should be a monotonically non-decreasing function of similarity between the two features, and it should achieve its maximum value when $x_\ell^i = x_\ell^j$. The w_ℓ allow one to weight each feature differently, forcing the optimization procedure to prefer some features over the others. In the experiments reported here, $w_\ell = 1$ for all ℓ.

We define the fitness of a GrowCode program based on eq. (1). Let $\mathbf{x}^{\mathrm{T}}$ be a target feature collection and let $\mathbf{x}^{\mathrm{P}}$ be a random variable representing the feature collection for the graph generated by the randomized program P. Then we can define our problem as the search for P* such that:

$$\mathtt{P}^* = \arg\max_{\mathtt{P}} \mathbb{E}[s(\mathbf{x}^{\mathrm{P}}, \mathbf{x}^{\mathrm{T}})], \tag{2}$$

where the expectation is taken over various runs of P. That is, we seek the program P* such that the graph generated by P* are expected to have features most similar to those given by $\mathbf{x}^{\mathrm{T}}$ as measured by the similarity function $s(\cdot, \cdot)$. This optimization problem is difficult given the size of the space of potential programs. To tackle this problem effectively, we adopt genetic programming techniques which have proven effective in similarly difficult optimization scenarios.

4.2 Optimization with genetic algorithms

We use the ECJ package [18] to perform the optimization, and we use its abilities to evaluate individuals within a generation in parallel, customize the selection methods and breeding architecture for multiple sub-populations, perform NSGA-II multi-objective optimization [8], and handle arbitrary representations of fixed and variable length genomes.

Each individual encodes a program. At each generation, we evaluate the fitness for all individuals in the fixed-size population. Each program's fitness is calculated by running it for k iterations, and comparing its feature vector $\mathbf{x}^{\mathrm{P}}$ against the target set of features. This is repeated some number M times, and the results are averaged, so that the fitness of program P is:

$$\mathcal{F}(\mathtt{P}) = \text{avg } s(\mathbf{x}^{\mathrm{P}}, \mathbf{x}^{\mathrm{T}}). \tag{3}$$

Alternatively, we can average each $s_\ell(x_\ell^i, x_\ell^j)$ in eq. (1) as a separate objective, and employ a multi-objective optimization strategy (e.g. NSGA-II) [8].

When breeding individual programs, a two-point crossover operation allows the programs to mix with each other thus varying their contents and length. At the end of each generation, individuals compete in a tournament of successive comparisons of two randomly chosen individuals, where winners are chosen deterministically based on the higher fitness value. Winners of the tournament become members of the subsequent population. Individuals are drawn with replacement and can thus be replicated in the subsequent population. More fit individuals are more likely to win tournaments, making "elite" members more likely to survive into the next generation.

5. APPLICATIONS TO SYNTHETIC AND REAL NETWORKS

We demonstrate the use of our framework to learn GrowCode programs that produce networks matching specified properties of both synthetic and real networks. Unless specified otherwise, we use the following parameters in our optimization procedures. All programs in the first generation of an optimization are initialized randomly with 10 instructions. Each generation consists of 100 programs that are evaluated on the basis of a single-objective (section 5.1) or multi-objective (sections 5.2 to 5.4) fitness function. Individuals are chosen to advance to successive generations using tournament-selection. At the start of each generation, the population of individuals is bred from the selected individuals from the previous generation using two-point list crossover breeding. This produces new individuals, potentially with programs of different length, which are then subject to mutation (we use a mutation rate of 0.1). The optimization procedure is carried out for 15 generations, and we select the most fit individual from the final generation as the representative GrowCode program against which we compare other models.

5.1 Learning scale-free graphs

Scale-free distributions are the key network property that motivated the B-A growth model, and we show that GrowCode can learn models that produce scale-free distributions. Given the large space of models defined by the instruction set and their parameters, it is unclear at first if effectively exploring this space is even possible.

We measure the similarity of two degree distributions via a shape function. Although one straightforward option is to choose the goodness of fit to a scale-free distribution as one of our features, this approach is specific to scale-free distributions, and in general, we would like to generate graphs that match the shape of any specified degree distribution, not only scale-free distributions. We define the shape ψ_{shape} of a graph to be the cumulative distribution of node degrees where the support of the distribution (the degrees of the nodes) is normalized between 0 and 1. This normalization allows for the comparison of the degree distribution of graphs of different sizes. We define the similarity metric for the shape feature to be:

$$s_{\text{shape}}(\psi^i_{\text{shape}}, \psi^j_{\text{shape}}) = \frac{1}{\|\psi^i_{\text{shape}} - \psi^j_{\text{shape}}\|_1 + \epsilon}, \qquad (4)$$

where ϵ is a small positive constant to assure that the fitness is defined (as a large value) when the shapes coincide exactly.

The B-A model is parameterized on i, the number of vertices to which a new vertex connects. To get the target degree distribution shape for various i, we generate graphs for $i = 3, 4, 5, 6$ and obtain maximum-likelihood estimates of scale-free exponents of $\alpha = 2.61, 2.71, 2.73, 2.92$, respectively. We then use s_{shape} in eq. (3) to define the fitness of a program as the difference in degree-distribution shape between the graphs produced by the program and the estimated target shape.

With this fitness, GrowCode learns many non-identical programs that are scale free. Algorithm 4 shows one of the effective scale free GrowCode programs learned during the optimization process. To test for the plausibility of a scale-free distribution, we use statistical tests specific to that distribution as described in [7]. We find that even though the α parameter was not directly used in the fitness function, the networks instantiated from the learned models that pass the scale-free test have an average α value of 2.69, which is reasonably close to that of the target graphs.

In fact, we posit that it is quite easy for our optimization procedure to discover a GrowCode program that produces networks with a scale-free degree distribution. Figure 2 shows a trace of one of the optimization runs when attempting to fit a degree-distribution generated from the B-A model with $i = 4$. Scale-free models are discovered in the first generation of the optimization procedure, even before fitness selection has had an opportunity to affect the population. The total fitness of the scale-free individuals grows quickly, and by generation 6, is already substantially greater than the total fitness of the non-scale-free individuals (fig. 2). These observations have two implications. First, the shape function seems to correlate well with the scale-free plausibility of the graph. Second, discovering a scale-free model is not difficult.

Algorithm 4 Example Learned Scale-Free Model

```
1: NEW NODE
2: RANDOM NODE
3: ATTACH TO INFLUENCED
4: CLEAR r2
5: SET(1)
6: RANDOM EDGE
7: DETACH FROM INFLUENCED
8: RANDOM NODE
9: CREATE EDGE
10: INFLUENCE NEIGHBORS(0.692)
```

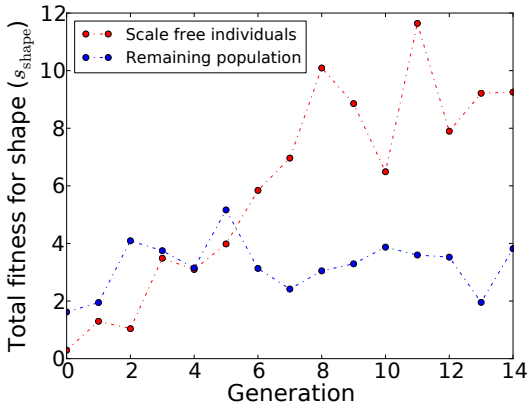

Figure 2: Total fitness for shape (s_{shape}) at each generation. The total fitness for those individuals that pass the scale-free plausibility test is drawn in red, while the total fitness for the rest of the individuals in the generation is drawn in blue. After just the fifth generation, the total fitness of scale-free individuals is approximately twice as large as the remaining population.

5.2 Performance on a social network

We apply GrowCode to a recent network of co-authorship of genome-wide association studies (GWAS) [5]. Specifically, we consider the social network of "repeated co-authorship" where pairs of scientists are linked if they published together more than once. This network is associated with a high assortativity (0.19), which implies that scientists who collaborate prolifically tend to connect with scientists who also collaborate with many other researchers. We simultaneously optimize for the shape distribution feature commonly studied in social networks [7] and assortativity using the multi-objective scheme of section 4.2.

We compare graphs generated by GrowCode programs to graphs

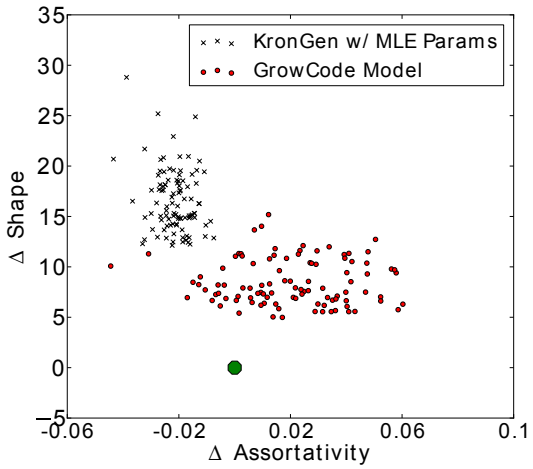

Figure 3: GWAS target network. Each point represents a single generated graph from a model. The x axis represents the difference between the assortativity of a graph and the target co-authorship graph. The y axis represents the difference between the shape of a learned graph and the co-authorship graph. The green dot represents a perfect match to the co-authorship graph.

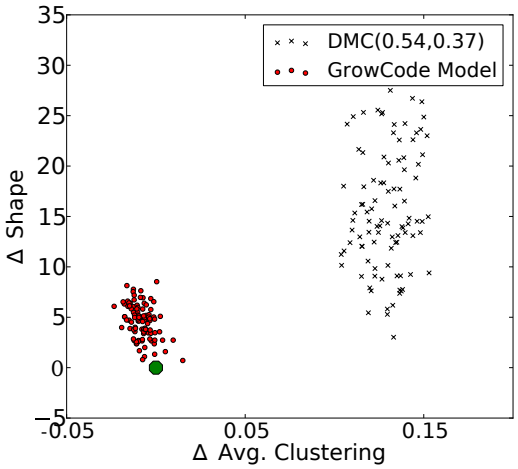

Figure 4: PPI target network. Each point represents a single generated graph. The x axis gives the difference between the average clustering coefficients of a graph and the protein interaction graph. The y axis gives the difference between the shape of a learned graph and the protein interaction graph. The green dot represents the origin and the protein interaction graph.

generated by the Kronecker model [16], a fast, recursive graph generation model that has been shown to reproduce many characteristics of real-world networks. The Kronecker model requires parameters to generate random graphs with the desired properties, and we use the KronFit maximum-likelihood algorithm on the GWAS graph to estimate these parameters. We compare the features of 100 graphs generated by each model to the real GWAS network.

The learned GrowCode model better matches the assortativity of the original graph as well as the shape of the degree distribution (Figure 3) than the best-fit Kronecker model. The average shape difference of the GrowCode model (mean 8.47, std. dev 2.24) is closer to the shape of the co-authorship network than that for the Kronecker model (mean 16.6, std. dev 3.09). The mean assortativity for the GrowCode graphs is 0.206 while the mean for the Kronecker graphs is 0.165. However, in this case, different graphs produced by the GrowCode model are associated with a wider range of assortativity values (std. dev 0.0208) than the Kronecker graphs (std. dev 0.00629).

5.3 Performance on a biological network

We demonstrate the ability of GrowCode to learn a program that generates graphs similar to a high-quality and recent yeast protein interaction network compiled by Gibson et al. [11]. We optimize for both the shape distribution and the clustering coefficient, both shown to be biologically relevant in protein interaction networks [27]. We follow a similar procedure as in section 5.2, but instead of the Kronecker model, we use the DMC model as the baseline comparison. We determined the best parameters for DMC ($q_{\text{mod}} = 0.55$ and $q_{\text{con}} = 0.37$) via a grid search over the parameter space, and we selected the pair of parameters for which the graphs produced by the model closely match the number of edges, clustering coefficient, and diameter of the input PPI network.

The networks generated by the learned GrowCode program match the target characteristics of the real network substantially better than the networks produced by the DMC model (Figure 4). The mean average clustering coefficient produced by the GrowCode program is 0.091 (std. dev 0.006), which matches the average clustering coefficient (0.099) of the protein interaction network very well. Conversely, the mean average clustering coefficient produced by DMC is 0.227 (std. dev 0.013) which is quite far from the true value. The results for the shape distribution yield a similar conclusion (fig. 4). Over the random networks generated by the GrowCode program, the average shape distribution distance is 4.58 (std. dev 1.69) while the average distance among the DMC-generated networks is 15.48 (std. dev 6.29). The GrowCode program not only produces graphs that match the target better but also that exhibit greater parametric stability (i.e. less variance) with regard to these metrics.

5.4 Performance on a technological network

Above, we showed that GrowCode performs well when learning models for pairs of network properties. In each case, the particular pair of properties were chosen because they had been studied in the context of the corresponding class of network. The GrowCode framework is not restricted to only two target features, and here, we provide evidence that it is possible to extend the learning process to more attributes. Using the Autonomous Systems (AS) Route View graph discussed in [16] as a target, we learn a GrowCode program by simultaneously optimizing for all three of the previously considered features (shape, assortativity, and average clustering coefficient). In this case, 150 individuals were evolved for 25 generations. As in Section 5.2, we compare graphs generated by the learned GrowCode program to those generated by the Kronecker model.

The three plots in Figure 5 show that for all pairs of properties, graphs generated by GrowCode are close to the real world network and, in fact, match the AS graph better than those generated by the Kronecker model. Note that the plots show only two dimensions at a time, but a single learned GrowCode program was optimized for

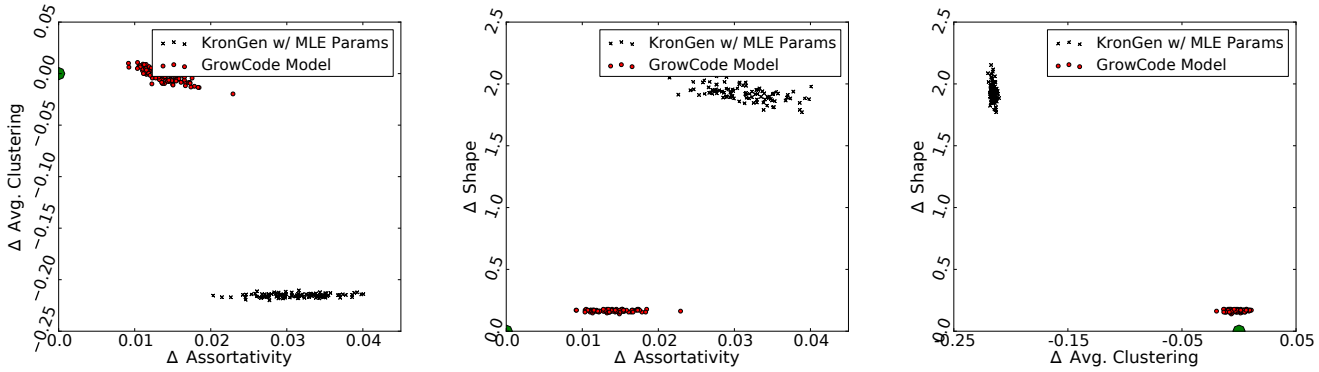

Figure 5: AS target network. Each point represents a single generated graph. The difference between the coefficients of a graph and the AS graph are plotted for three pairs of network properties: average clustering vs. assortativity, shape vs. assortativity, and shape vs. average clustering. The green dot represents the origin and the AS graph.

Table 2: Mean and standard deviation $(\mu \pm \sigma)$ of spectral distance between all graphs generated by the B-A model and by GrowCode programs.

i	3	4	5	6
B-A	0.0086 ± 0.0058	0.0034 ± 0.0007	0.0029 ± 0.0006	0.0026 ± 0.0005
GC	0.0141 ± 0.0115	0.0252 ± 0.0206	0.0288 ± 0.0228	0.0182 ± 0.0152

all three features at once. As with the biological network in Section 5.3, the GrowCode program produces graphs that have low variance with respect to the target topological properties.

5.5 GrowCode generates random models

To ensure that the diversity of the graphs produced by GrowCode matches that of those produced by hand-coded models, we computed the mean and standard deviation of the spectral distance between 100 graphs generated by both the traditional B-A model and the GrowCode program that best fit a scale-free graph. The spectral distance is a reasonable, efficiently-computable measure of graph similarity that correlates well with graph edit distance [29]. In order to estimate spectral distances between graphs, we used a discretized histogram of the normalized Laplacian eigenvalue distribution (100 bins). The spectral distance is then the Euclidean distance between such histograms.

As Table 2 shows, GrowCode does not produce deterministic models, and in fact the models learned by GrowCode generate an ensemble of graphs that has higher diversity than the ensemble produced by the B-A model, despite matching the target properties better. Similar diversity is also observed when this experiment is repeated to compare graphs generated by the optimal model learned by GrowCode to fit the yeast PPI network and the graphs generated by DMC (data not shown).

6. CONCLUSIONS AND FUTURE WORK

We have introduced a novel framework, GrowCode, for representing network growth models as programs composed from a short and descriptive set of instructions. We demonstrate that this representation is sufficiently general to reproduce close approximations to several existing growth models. Additionally, this formal encoding allows for an effective search procedure to find models with desired properties. In representative social, biological, and technological networks, a fairly fast optimization procedure (no run took more than 30 minutes for two objectives and no more than 4 hours for three objectives) is able to produce GrowCode programs that are competitive with recent network growth models designed to match properties of graphs in these domains.

We are also able to match scale-free graphs with several different attachment parameters i, and we are able to learn GrowCode programs that pass rigorous statistical tests for scale-free plausibility. Indeed, our optimization procedure comes across scale-free GrowCode programs quickly, and by the end yields a large number of non-identical programs that produce graphs passing the scale-free plausibility test. Additionally, we show that the graphs produced by these GrowCode programs are at least as diverse as graphs generated by the B-A model. This indicates that the scale-free property is quite ubiquitous among possible growth models.

The framework presented here applies to both undirected and unattributed networks, yet we believe it can be extended to handle directed networks and networks with node and edge attributes. Certain generalizations may be achieved by extending the instruction set, for example, by incorporating instructions that create and modify directed edges, allow node attributes to spread throughout the network, or that encode a more complex and general influence mechanism. Others may require enhancements to the machine. For example, handling edge attributes may require the addition of an edge memory akin to the label memory of the current machine. Such modifications, however, are not conceptually difficult, though their careful design is important.

Finally, although the instructions used in GrowCode programs are individually interpretable, the optimization procedure may produce programs whose overall growth mechanisms are sometimes, though not always, opaque. In the future, we plan to explore how ensembles of learned programs can be analyzed to extract from them interpretable mechanisms of growth by finding commonly occurring instruction patterns (motifs). For example, on further analysis of programs like Algorithm 4, we have noticed certain repeated patterns of instructions that are often used to match scale-free networks and to create edges to existing nodes proportional to their degrees. These patterns show up with NEW NODE and CREATE EDGE instructions as well as the influence operations. Mining GrowCode programs for repeated patterns could reveal sets of instructions that are interpretable as a unit.

Acknowledgements

This work was partially supported by the National Science Foundation [CCF-1053918, EF-0849899, and IIS-0812111], the National Institutes of Health [1R21AI085376], and a University of Maryland Institute for Advanced Studies New Frontiers Award. C.K. received support as an Alfred P. Sloan Research Fellow.

References

[1] L. Akoglu and C. Faloutsos. RTG: a recursive realistic graph generator using random typing. *Data Mining and Knowledge Discovery*, 19(2):194–209, 2009.

[2] A. Barabási. Emergence of Scaling in Random Networks. *Science*, 286(5439):509–512, 1999.

[3] A. Bhan, D. Galas, and T. Dewey. A duplication growth model of gene expression networks. *Bioinformatics*, 18:1486–1493, 2002.

[4] M. Brameier. *On linear genetic programming*. PhD thesis, University of Hamburg, 2004.

[5] B. K. Bulik-Sullivan and P. F. Sullivan. The authorship network of genome-wide association studies. *Nature Genetics*, 44(2):113–113, 2012.

[6] D. Callaway, J. E. Hopcroft, J. M. Kleinberg, M. E. Newman, and S. H. Strogatz. Are randomly grown graphs really random? *Phys. Rev. E*, 64:041902–041908, 2001.

[7] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-Law Distributions in Empirical Data. *SIAM Review*, 51(4):661, 2009.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[9] S. N. Dorogovtsev, J. F. Mendes, and A. N. Samukhin. Structure of growing networks with preferential linking. *Phys. Rev. Lett.*, 85(21), 2000.

[10] P. Erdős and A. Rényi. On the evolution of random graphs. *Evolution*, 5(1):17–61, 1960.

[11] T. A. Gibson and D. S. Goldberg. Improving evolutionary models of protein interaction networks. *Bioinformatics*, 27(3):376–382, 2011.

[12] I. Ispolatov, P. L. Krapivsky, and A. Yuryev. Duplication-divergence model of protein interaction network. *Phys. Rev. E*, 71(6 Pt 1):061911, 2005.

[13] M. Kim and J. Leskovec. Multiplicative attribute graph model of real-world networks. *Internet Mathematics*, 8(1-2):113–160, 2012.

[14] W. K. Kim and E. M. Marcotte. Age-dependent evolution of the yeast protein interaction network suggests a limited role of gene duplication and divergence. *PLoS Comput. Biol.*, 4(11):e1000232, 2008.

[15] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 611–617, 2006.

[16] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker Graphs: An Approach to Modeling Networks. *The Journal of Machine Learning Research*, 11:985–1042, 2010.

[17] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time. In *Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 177–187, New York, USA, 2005.

[18] S. Luke. *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat*. PhD thesis, University of Maryland, College Park, 2000.

[19] M. Middendorf, E. Ziv, C. Adams, J. Hom, R. Koytcheff, C. Levovitz, G. Woods, L. Chen, and C. Wiggins. Discriminative topological features reveal biological network mechanisms. *BMC Bioinformatics*, 5:181, 2004.

[20] M. Middendorf, E. Ziv, and C. Wiggins. Inferring network mechanisms: The *Drosophila melanogaster* protein interaction network. *Proc. Natl. Acad. Sci. USA*, 102:3192–3197, 2005.

[21] S. Navlakha and C. Kingsford. Network archaeology: Uncovering ancient networks from present-day interactions. *PLoS Comput. Biol.*, 7(4):e1001119, 2011.

[22] G. Palla, L. Lovász, and T. Vicsek. Multifractal network generator. *Proc. Natl. Acad. Sci. USA*, 107:7640–7645, 2010.

[23] N. Przulj, O. Kuchaiev, A. Stevanovic, and W. Hayes. Geometric evolutionary dynamics of protein interaction networks. *Pac. Symp. Biocomput.*, 15:178–189, 2010.

[24] R. Solé, R. Pastor-Satorras, E. Smith, and T. B. Kepler. A model of large-scale proteome evolution. *Adv. Complex Syst.*, 5(1):43–54, 2002.

[25] Y. Sun, Y. Yu, and J. Han. Ranking-based clustering of heterogeneous information networks with star network schema. In *Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 797–806, New York, NY, USA, 2009. ACM.

[26] S. A. Teichmann and M. M. Babu. Gene regulatory network growth by duplication. *Nat. Genet.*, 36:492–496, 2004.

[27] A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani. Modeling of protein interaction networks. *Complexus*, 1(38):9, 2001.

[28] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 363:202–204, 1998.

[29] R. C. Wilson and P. Zhu. A study of graph spectra for comparing graphs and trees. *Pattern Recognition*, 41:2833–2841, 2008.

[30] H. Yu, P. Braun, M. A. Yildirim, I. Lemmens, K. Venkatesan, J. Sahalie, T. Hirozane-Kishikawa, F. Gebreab, N. Li, N. Simonis, T. Hao, J.-F. Rual, A. Dricot, A. Vazquez, R. R. Murray, C. Simon, L. Tardivo, S. Tam, N. Svrzikapa, C. Fan, A.-S. De Smet, A. Motyl, M. E. Hudson, J. Park, X. Xin, M. E. Cusick, T. Moore, C. Boone, M. Snyder, F. P. Roth, A.-L. Barabási, J. Tavernier, D. E. Hill, and M. Vidal. High-quality binary protein interaction map of the yeast interactome network. *Science*, 322(5898):104–110, 2008.