5-2015

A More Robust Corpus of Büchi Automata

Corey S. Fisher
Trinity University, cfisher@trinity.edu

A More Robust Corpus of Büchi Automata

Corey Fisher

A departmental senior thesis submitted to the
Department of Computer Science at Trinity University
in partial fulfillment of the requirements for graduation
with departmental honors.

April 16th, 2015

Thesis Advisor

Department Chair

Sheryl Tynes, AVPAA

Student Agreement

I grant Trinity University (“Institution”), my academic department (“Department”), and the Texas Digital Library ("TDL") the non-exclusive rights to copy, display, perform, distribute and publish the content I submit to this repository (hereafter called "Work") and to make the Work available in any format in perpetuity as part of a TDL, Institution or Department repository communication or distribution effort.

I understand that once the Work is submitted, a bibliographic citation to the Work can remain visible in perpetuity, even if the Work is updated or removed.

I understand that the Work's copyright owner(s) will continue to own copyright outside these non-exclusive granted rights.

I warrant that:

1) I am the copyright owner of the Work, or
2) I am one of the copyright owners and have permission from the other owners to submit the Work, or
3) My Institution or Department is the copyright owner and I have permission to submit the Work, or
4) Another party is the copyright owner and I have permission to submit the Work.

Based on this, I further warrant to my knowledge:

1) The Work does not infringe any copyright, patent, or trade secrets of any third party,
2) The Work does not contain any libelous matter, nor invade the privacy of any person or third party, and
3) That no right in the Work has been sold, mortgaged, or otherwise disposed of, and is free from all claims.

I agree to hold TDL, Institution, Department, and their agents harmless for any liability arising from any breach of the above warranties or any claim of intellectual property infringement arising from the exercise of these non-exclusive granted rights.

I choose the following option for sharing my thesis (required):

[X] Open Access (full-text discoverable via search engines)
[ ] Restricted to campus viewing only (allow access only on the Trinity University campus via digitalcommons.trinity.edu)

I choose to append the following Creative Commons license (optional):

A More Robust Corpus of Büchi Automata

Corey Fisher

Abstract

Formal verification is a process that proves computational systems adhere to a specification. The automata-theoretic model uses automata, in our case Büchi automata, to model programs and specify how the programs should behave. A core part of formal verification is to check that the program is contained in its specification. Checking the containment of Büchi automata is PSPACE-complete: in the worst case taking time exponential in the size of the automata.

Such terrible worst cases are not always present in practice, so researchers wish to test algorithms on more practical cases. To date, the field has used the Tabakov-Vardi random model of automata as a corpus for experiments. However, the Tabakov-Vardi model produces unstructured automata that thus do not strongly resemble real-world problems.

We define 7 new models of random automata, most of which are based on structured graph models from other disciplines. Additionally, we define two properties that a random model should possess to be useful for experiments: texture and stability. We empirically examine the models. Six of the models are textured, and we found promising stability in two models.

Acknowledgments

Thanks to Dr. Seth Fogarty, my thesis advisor, and Drs. Paul Myers and Matt Hibbs, my thesis committee, for teaching me and giving me feedback.
Thanks to my grandparents for providing laundry service and a place to retreat to for good food and conversation.
Thanks to Dr. Moshe Vardi and Rice University for allowing me to use the DAVinCI cluster for my research.

This work was funded in part by the Data Analysis and Visualization Cyberinfrastructure funded by NSF under grant OCI-0959097, as well as a Murchison Summer Undergraduate Research Fellowship and a Mach Fellowship.

Contents

1 Introduction

Today, programs and other computational systems operate everything from banks, to traffic lights, to the government itself. These automated processes depend heavily on continued functioning of the systems that run them. If those programs fail the results are generally unpleasant and often catastrophic. It is therefore quite important to prevent failures if at all possible - usually by fixing bugs.

Ideally, we want the programs running important automated systems to be completely error-free. However, finding and fixing all the bugs in a program is a difficult task. The standard debugging process is far from perfect. It can never say that a program is bug-free - it can only say that, as more and more testing is done, bugs are less and less likely to still exist. Therefore, we want to find an alternative approach.

We use a technique called formal verification to mathematically prove that a program is bug-free. Formal verification is a process of comparing a model representing a program to a specification of that program. If the behavior of the program conforms to the specification, then the program is considered bug-free.

The automata-theoretic approach to formal verification models programs and their specifications as state diagrams called automata. In this approach, a possible execution of the program is represented as a sequence of characters - a word. An automaton accepts a language: a set of words. The program automaton accepts any word representing a valid execution of the program, intentional or not. The specification automaton only accepts those executions which satisfy the properties we desire in the program. We concern ourselves with Büchi automata, used to verify fairness, liveness, and safety properties of a system.

This approach reduces the question of program correctness to the question of automata, and therefore language, containment. More specifically, to determine if the program satisfies its specification, check if the language of the program automaton is contained in the language of the specification automaton. This process proceeds in three steps. First, complement

the specification automaton. The complemented automaton accepts exactly those words that would be rejected by the specification. Therefore, it accepts every possible counterexample to the program satisfying the specification. As complementation is PSPACE-complete[14], this is the most computationally difficult step. Second, take the intersection of the complemented specification automaton and the program automaton - a polynomial operation. The intersection automaton will only accept words that would be rejected by the specification, but are accepted by the program automaton. Finally, check if the language of the intersection automaton is empty. If the language of the intersection automaton is empty, then the program satisfies the specification. If not, then we have found a counterexample that may correspond to a bug.

The ability to verify programs is extremely useful, and so we want to use it in practice. Many different constructions for containment checking have been proposed for this. However, a practical approach to verification via containment checking must not just work, but work quickly. Thus we need to compare the algorithms to determine which is most effective. The usual method of comparing the efficiency of algorithms is to compare their worst-case running time. Unfortunately, the worst case for this problem is intractable: complementation is PSPACE-complete. However, for many problems with intractable worst cases, real-world cases do not exhibit this pathological behavior. If this holds true here, then the usual methods tell us little. Thus, we opt to step away from the theoretical analysis and compare approaches empirically. We implement the algorithms and test them on a corpus of example automata, allowing us to compare the efficiency of algorithms in the real-world.

To perform an empirical analysis, we require a corpus of containment problems on which to run these experiments. Unfortunately, real-world examples are still uncommon, and often trivially simple. Instead, almost all experiments to date have been performed on corpora of automata derived from the Tabakov-Vardi (T-V) random model of automata[16, 17, 5, 1]. The T-V model has three parameters - size, transition density, and accepting state density. The T-V model generates automata with these parameters by

selecting transitions and accepting states uniformly at random.

Tests using this model have focused on the most difficult step of containment: complementation. Instead of checking if one randomly generated automaton accepts the language of another, they check whether a random automaton accepts every string. Such an automaton is called *universal*. Universality checking avoids the intersection step of containment checking. However, since it still requires complementation and emptiness checking, it provides useful data on the efficiency of the algorithm.

The Tabakov-Vardi model can produce automata with a wide variety of chances of universality, using different configurations of transition and acceptance density. This property is called *textured*. Having a variety of options in the parameter space makes a model more useful for experiments, since it allows us to find problems that are easier or harder for a given approach.

Unfortunately, the uniformly random T-V model corresponds to the real world in a very specific way - it doesn't. Unlike Tabakov-Vardi automata, real world problems generally have structure. This prevents us from being confident in results derived from experiments run exclusively on Tabakov-Vardi automata. To increase confidence in experimental results, we create in this thesis more structured random models - models possessing properties mimicking real-world networks.

The Tabakov-Vardi model results from lifting Karp's random digraph model to a random automata model[9]. Karp's model, however, is far from the only one. Many other models of random graphs have been proposed. Many of these models have a structure that reflects the properties of certain real-world networks, as well as parameters to control the resulting graphs. Thus, to create more structured random models of automata, we scoured through papers in a variety of disciplines, from biology to sociology to electrical engineering. We have located existing random graph models, and then lifted six of them into models of structured random automata. Additionally, we define one new model which cannot exist as a random graph model.

Having created new models possessing properties similar to real-world networks, we check if they are textured. To do so, we implement generators

for the models in Haskell and create corpora of examples from them. These corpora contain automata made with a variety of parameters for each model, allowing us to measure variance in the chance of universality. By varying the parameters, we test that the models can generate a range of universal and non-universal automata.

We also introduce a new concept, stability, for evaluating random models of automata. A *stable* model tends to have the same universality probability at a given configuration while n increases. Because we want to see how an approach to containment checking scales as problems get larger, we increase the size of the automata during comparative testing. A stable model keeps the conditions of the testing constant as size changes.

2 Background

This section introduces the concepts and definitions required in this thesis.

2.1 Büchi Automata

We consider Büchi automata in this thesis. A Büchi automaton is similar to a finite state automaton, save that it takes an infinite word as input instead of a finite one. Whereas automata over finite words accept a word if the run ends in an accepting state, Büchi automata are more complex. Instead, a Büchi automaton accepts an infinite word if an accepting state of the automaton is visited an infinite number of times.

Formally, a *Büchi automaton* is a five-tuple $A = (\Sigma, Q, Q_0, \delta, F)$, where Σ is an *alphabet* of input characters, Q is a finite set of *states*, $Q_0 \subseteq Q$ is a set of *starting states*, $\delta \subseteq S \times \Sigma \times S$ is a *transition relation*, and $F \subseteq Q$ is a set of *final states*. For states $q, r \in Q$ and a character $\sigma \in \Sigma$, the tuple (q, σ, r) is in the transition relation δ exactly when the automaton can transition from q to r on σ.

A Büchi automaton takes an infinite *word*, or sequence of characters from Σ, as input. A *run* of an automaton A on a word $w = a_0, a_1, \ldots \in \Sigma^\omega$ is an infinite sequence of states $q_0, q_1, \ldots \in Q^\omega$ such that for every $i \geq 0$,

the tuple $(q_i, s_i, q_{i+1}) \in \delta$. As the transition relation is nondeterministic, the automaton may have a number of possible runs on w. A run is said to be *accepting* if there exists a state $q_f \in F$ that occurs in the run infinitely often. We say that A accepts w if there is an accepting run of A on w. The *language* of A, written $L(A)$, is the set of all words accepted by A. If A accepts all words in Σ^ω then it is said to be *universal*. We occasionally want to speak about finite paths path through Büchi automata. A *path* from a state q_0 *to* a state q_i is a sequence of states $q_0, q_1, \ldots, q_i$ where for every k, $0 \leq k < i$, there exists a character $\sigma \in \Sigma$ such that $(q_k, \sigma, q_{k+1}) \in \delta$.

Büchi automata gain expressive power from nondeterminism. This is in contrast to automata on finite words, where there is a deterministic equivalent to every nondeterministic automaton.

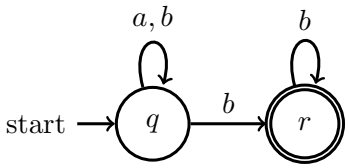


Figure 1: A simple Büchi automaton accepting the language $(a \mid b) * b^\omega$. The initial state in q, marked with the incoming arrow. The only accepting state is r, denoted by the double circle. The automaton can stay in q indefinitely. In order to accept a word, the automaton must move to r on a b. However, once in r the run can only continue if the word contains only b's.

Example 2.1. Figure 1 displays an example of a Büchi automaton that has no deterministic equivalent. This automaton accepts the language of any infinite sequence of a's and b's with only a finite number of a's. An accepting run of the automaton remains in the starting state q until it has seen the last a, then transitions to state r on a b. However, the automaton cannot know what characters it will see in the future. Therefore, each time it sees a b, a run guesses that no more a's remain. If the guess was wrong, and there still is an a, the run crashes - there is nowhere it can go in state r on an a. The automaton continues with another run that did not make the failed

guess. If the guess is right, and only b's remain, this run continues in the accepting state and accepts the word. This language cannot be represented by a deterministic Büchi automaton[18].

2.2 Containment Checking and Universality

In automata-theoretic verification, we prove that a program satisfies a specification by modeling the program as a Büchi automaton A and the specification as a Büchi automaton B. Each state of A corresponds to some state in the program it models. This might be a function and the parameter values, or a line number and values of all global variables in a program. Each word is a trace of the program. This means that $L(A)$ is the language of all possible traces of the program, and $L(B)$ is the language of all traces allowed by the specification. The program satisfies the specification when the language of A is a subset of the language of B. That is, all possible program traces are also allowed traces. When this is true, we say that A is contained in B.

Checking if A is contained in B, proceeds in three steps. First, we complement B, creating a new automaton $\bar{B}$ automaton whose language is $\Sigma^\omega \setminus L(B)$. The complemented automaton $\bar{B}$ accepts all traces which violate the specification. Once we have $\bar{B}$ we construct an intersection automaton that accepts all words in $L(\bar{B} \cap A)$. We then check the language of the intersection automaton for emptiness - if the language is empty, then $L(A)$ is contained in $L(B)$. If the intersection is non-empty then $L(A) \not\subseteq L(B)$. Every word in $L(\bar{B} \cap A)$ is a potential counterexample to the correctness of the original program: a possible trace that violates the specification. However, the automaton model is necessarily a conservative approximation of the real program. This means that the automaton may accept traces that the program cannot actually generate. In this case, a word in $L(\bar{B} \cap A)$ may not correspond to an actual bug.

The most computationally complex step of this algorithm is complementation. Unlike complementing automata on finite words, complementation of Büchi automata is not straightforward. For automata on finite words,

it suffices to determinize the automata and swap the accepting and non-accepting states. Since Büchi automata are not closed under determinization, other approaches are needed. Complementation of Büchi automata is PSPACE-complete: in the worst case, the complementary automaton will have exponentially more states than the source automaton. The intersection automaton is only polynomial in the size of A and $\bar{B}$. Checking for emptiness can be done in linear time, although frequently quadratic algorithms prove more efficient in practice

Because of the complexity of complementation, there have been numerous constructions for creating complemented automata. Further, it is often more efficient to construct the complemented automaton on the fly during emptiness checking, rather than performing complementation as a separate step. This has resulted in a plethora of containment checking algorithms. Experiments on these algorithms have focused on universality: a simpler form of containment checking that focuses on complementation. An automaton is universal if it accepts all possible inputs. Therefore, to check if an automaton is universal, complement it and check if the language of the complemented automaton is empty.

2.3 Random Choice

With a variety of universality solvers available, it is useful to know which ones are viable in practice for solving containment problems in reasonable amounts of time. However, because practical cases are so significantly better than worst cases, we need to test them on data. We don't have large amounts of useful real-world data, so instead we build a corpus of random automata for testing.

We define here different kinds of random choice used in the rest of this thesis to generate such a corpus. When making a *uniformly random choice* from a set S, each element $e \in S$ has an equal chance of being chosen. When making a *weighted random choice* from S, each element $e \in S$ has a *weight* w_e. The chance of e being selected is equal to $w_e / \sum_{s \in S} w_s$. Finally, when given two options, a coin flip simply assigns both equal probability.

2.4 The Tabakov-Vardi Model of Random Büchi Automata

The Tabakov-Vardi (T-V) model is an established random model of Büchi automata. The T-V model lifts Karp random graphs [9] into a model of automata. The model has three parameters - an integral *size* n, a positive real *transition density* r, and a real *accepting state density* f between 0 and 1. The transition density is the average out-degree of each node per character. The accepting state density is the proportion of states that are accepting states.

Formally, the (n, r, f) T-V model is defined as follows. Each random automaton $A = (\Sigma, Q, Q_0, \delta, F)$ has the alphabet $\Sigma = \{1, 0\}$ and set of states $Q = \{0, \ldots, n-1\}$. The set Q_0 of initial states is $\{0\}$. For each $\sigma \in \Sigma$, the model generates a Karp directed graph D_σ over the nodes $\{0, \ldots, n-1\}$ with $n * r$ edges chosen uniformly at random. The transition relation δ is then defined as $\{(q, \sigma, r) \mid (q, r) \in D_\sigma\}$. The final states F comprise $\lfloor n * f \rfloor$ states selected uniformly at random from Q.

The T-V model is used to compare universality checkers. The reason why the T-V model works for these is because varying the parameters can change the properties of the resulting automaton. A *configuration* of the T-V model is a specific transition and acceptance density. The *universality probability* of a configuration is the chance of an automaton generated by that configuration is universal. Figure 2 demonstrates the effects of varying the configurations on universality for automata of size $n = 20$. In the T-V model, the universality probability increases with both r and f. However, r creates much larger effects, and f smaller, finer ones. Because the probability of universality varies from 0 to 1 between different configurations, the T-V model is considered interesting.

When comparing checkers, we first run a terrain experiment to find difficult configurations for each of the approaches. Since universality probability does not directly correlate with difficulty, this terrain experiment measures the running time of the approach on the generated automata instead. Once difficult configurations have been identified, a small number of configurations are chosen, preferably ones that are difficult for both models. We can

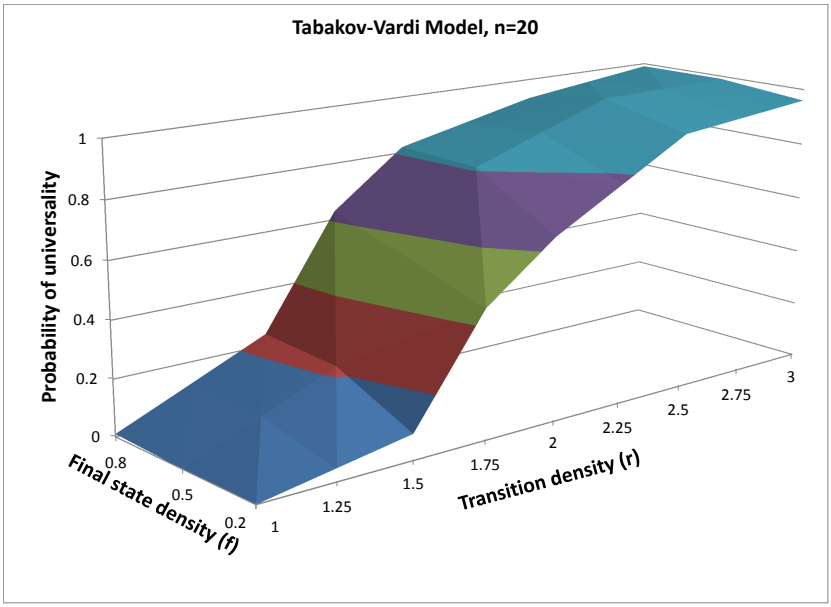


Figure 2: A terrain graph showing the universality probabilities of automata generated by different configurations of the Tabakov-Vardi model.

then run scaling experiments on them. To run a scaling experiment, we fix a configuration and then steadily increase n, and measure the running times of the approaches on these automata. Since size is an extremely important component of difficulty, this means we are testing the ability of the approaches to scale up to steadily more difficult problems.

As might be expected, configurations that do not overwhelmingly produce one of either universal or non-universal automata more than the other tend to be the most difficult for universality solvers. Since these tend to be cases that are close calls, they are less likely to terminate early. These configurations are also useful for comparison between checkers because some approaches terminate early on universal automata, and some terminate early on non-universal automata - so these configurations tend to more equally

distribute early termination. These effects of the universality probability of a configuration are precisely why the ability of configurations to affect universality probability in T-V make it useful for experiments.

3 New Random Models of Automata

By scouring through papers from various disciplines, we have found a number of random graph models which were amenable to adaption into random automata models. Each model can be configured by a number of parameters to generate variations in the resulting graphs. The model composes these graphs into an automaton. For all models we use the two-character alphabet $\Sigma = \{0, 1\}$.

In most cases we employ the technique of Tabakov and Vardi described in Section 2.4 to lift the model of random graphs to a model of random automata. The new automata model has all of the parameters of the graph model, plus final state density f. For a fixed set of parameters, each model defines the set of nodes as integers $N = \{0, \ldots, n\}$ The set of states Q is the set N of nodes. The set Q_0 of initial states is $\{0\}$. For each character $\sigma \in \Sigma$ create a directed graph D_σ from the random graph model. Each graph is made with the same parameters. The transition relation δ is $\{(q, a, r) \mid (q, r) \in D_\sigma\}$. Lastly, the set F of final states is $\lfloor n * f \rfloor$ elements of N chosen uniformly at random.

The first five models, small-world, scale-free, scale-free B, vertex-copying, and Kronecker, employ precisely this lifting. Because the states in the Frank-Strauss model are pairs of integers, it requires a straightforward variant of this approach. The accessible models are more complex as they employ notions of connectivity that are relative to all transitions, not just the transitions on a single character. Thus the graphs cannot be defined independently.

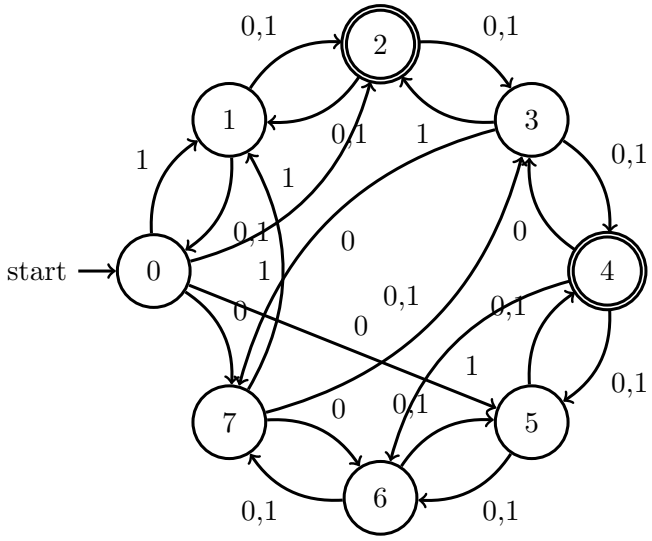


Figure 3: A small-world automaton with size $n = 8$, connectivity $k = 1$, rewire probability $p = 0.3$, and final state density $f = 0.3$.

3.1 Small-World Automata

The small-world graph model, described by Watts and Strogatz[19], defines a family of random graphs notable for two traits - clustering and low path length. Graphs posses high clustering when the neighbors of a given node tend to be neighbors of each other, creating the eponymous clusters. Graphs possess low path length when, on average, few edges are required to travel from one node to another. Small-world graphs have low path length because their clusters are connected at random by edges that cut across the graph, combining quick traversal from one cluster to another with the ease of moving from node to node in a cluster.

Small-world graphs can model many different things, such as collaborations of actors or scientists and friendship networks. They also may model modular systems[13], including programs. Most function calls in a modular program will be clustered with other, similar functions, with a few exceptions that reference other modules.

A random (n, k, p) small-world graph is parameterized by a *size* n, an integral *connectivity* k, and a *rewire probability* p. First, the model creates

an (n, k) ring lattice: a graph with the nodes $\{0, \ldots, n-1\}$ where every node is connected to its k nearest neighbors on each side, where $n-1$ and 0 are considered adjacent. Second, it chooses $n * p$ edges $(u, v) \in E$ uniformly at random to rewire[1], selecting a new destination j uniformly at random and replacing (u, v) with (u, j). Using the standard lifting, we extend this model into the random (n, k, p, f) small-world automata model.

Example 3.1. Figure 3 presents an example small-world automaton where $n = 8$, $k = 1$, and p and $f = 0.3$. Since $k = 1$, each state is connected to its nearest neighbor on either side. Similarly, slightly under a third of the transitions have been rewired to more distant states because $p = 0.3$. For example, the transition from 4 to 3 on input 1 has been reassigned to transition from 4 to 6. Note that each transition in the small-world graph starts out as a transition on both characters, but rewiring only reassigns one of the characters.

3.2 Scale-Free Automata

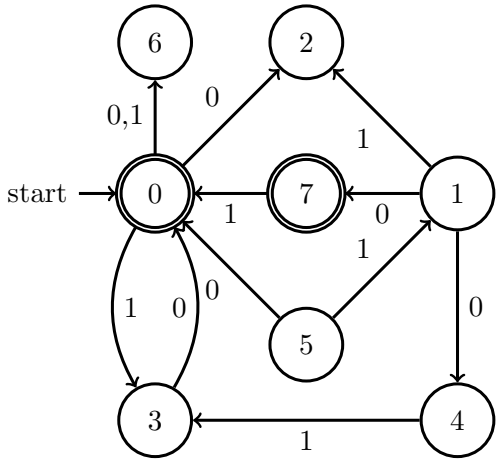


Figure 4: A scale-free automaton. It has 8 states ($n = 8$), but started with 2 (m=2). Transition density $k = 1$ and final state density $f = 0.3$.

[1]The original Watts-Strogatz model instead rewires each edge with probability p. We have chosen this formulation so the model is more stable at small sizes.

The random scale-free graph model, described by Barabasi *et. al.*[2], grows graphs in a series of timesteps. As nodes and edges are added, the model demonstrates a "rich-get-richer" property: nodes of high degree are more likely to get more edges. This creates a heavy-tailed degree distribution: a distribution which has a tail, like an exponential distribution, but even heavier. In this model the degree distribution of the nodes is consistent, regardless of the size ("scale") of the graph, and so it is considered "scale-free".

When a small number of nodes collect an large number of edges, these nodes become into hubs from which one can reach almost the entire graph. Among other things, scale-free graphs are commonly used to represent graphs of links between web pages. To link to a web page, you must already know about the page. However, the more links a page has going to it, the more likely it is to be known by people. So, pages that are already popular will naturally tend to accumulate more links than less-popular pages.

The scale-free model also represents structures like the New York stock exchange[7] and citations of scientific papers. Scale-free networks are quite common in biology[8, 3]. Programs following an actor model with a master-coordinator actor may exhibit scale-free structure. Operations between actors will tend to end up back at the master, creating hubs in the flow of the program.

Formally, a random (n, m, k) scale-free graph takes as parameters a *final size* n, a *starting size* m, and an integral *connectivity* k such that $k \leq m$. The model constructs the graph in a series of timesteps. At $t = 0$, the set V of nodes is $\{0, \ldots, m-1\}$ and $E = \emptyset$. At every timestep between $t = 1$ and $t = n - m$, it adds a node u to V. The model then selects k vertices according to the weighted probability in which each vertex $v \in V$ has weight $\deg(v) + 1$. For each w selected, either (u, w) or (w, u) is added to the set of edges, based on a coin flip. The model then proceeds to the next timestep. The scale-free model lifts to an automata model in the standard fashion.

Example 3.2. Figure 4 presents a scale-free automaton with $n = 8$, $m = 2$, $k = 1$, and $f = 0.3$. Note that there are no transitions between state 0 and

state 1 - this is always true of elements of the initial state set in scale-free automata. 0 and 1 are also the hubs, which is usually true of the initial states. In fact, every other state is connected to at least one of the hubs, with half of them connected to two. Since k=1, only one transition appeared with each new state per character, giving us an average of 2 transitions per state.

3.3 Scale-Free B Automata

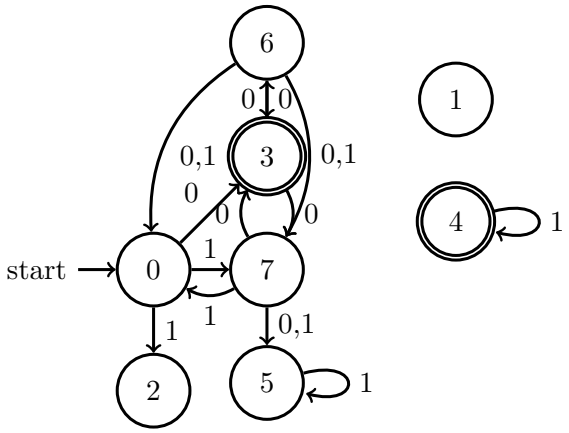


Figure 5: A scale-free B automaton. It has 8 states ($n = 8$), and one transition per state per character ($k = 1$). $f = 0.3$.

The random scale-free B graph model is a variant of the random scale-free graph model, also described by Barabasi *et. al.*[2]. While still demonstrating the "rich get richer" properties of scale-free graphs, scale-free B graphs do not grow in size, only number of transitions. Because of this, while they exhibit a heavy-tailed distribution when only partially connected, they will shift away from being scale-free as full connectivity is approached. The scale-free B model represents very similar things to its parent.

A random (n, k) scale-free B graph takes as parameters the size n, and the connectivity k. The model constructs a graph in a series of timesteps, much like the scale-free model. At $t = 0$, $V = 0, \ldots, n-1$ and $E = \emptyset$. Edges are added to E over $n * k$ timesteps. At each timestep, it chooses a node

$u \in V$ uniformly at random, and another node, v', is chosen from a random weighted distribution where each $v \in V$ has weight $\deg(v) + 1$. It then adds either (u, v') or (v', u) to E, chosen by coinflip. Like its predecessor, it is extended to an automata model in the standard fashion.

Example 3.3. Figure 5 presents a scale-free B automaton with $n = 8$, $k = 1$, and $f = 0.3$. Notice that states 1, 2, and 4 all have very few transitions, either 0 or 1. Meanwhile, state 0 has 6 transitions - nearly half of the total! States 3, 6, and 7 are not very far behind.

3.4 Vertex-Copying Automata

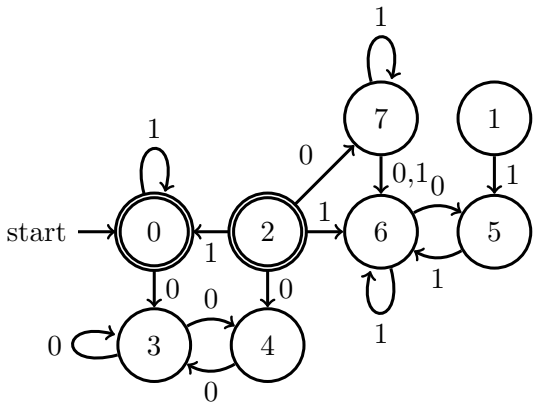


Figure 6: A vertex copying automaton of size 8 ($n = 8$). A certain proportion of transitions in the graph are copied - in this case, likely about a third ($b = 0.3$). It possesses in total one transition per state per character ($r = 1.0$) and $f = 0.3$

The random vertex-copying graph model presented here is a simplification of the model defined by Kleinberg *et al*[10]. The vertex-copying model, like the scale-free models, adds edges over time, and creates a heavy-tailed distribution. However, it does not produce this effect through preferential attachment, but by occasionally copying edges from one node to another.

This copying is intended to model hyperlinks on the Web - links are often created when someone discovers a link to a site they're interested in

on another site, then adds a link to it on their own website, thus "copying" the link from one site to another. It may also model code reuse - when a code block is reused, then calls to functions are duplicated.

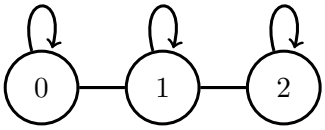
An (n, b, r) vertex-copying random graph takes as parameters the *size* n, the *probability of copying* b, and the *transition density* r. The vertices are $\{0, \ldots, n-1\}$. The model begins with edges and adds edges (u, v) to the graph one at a time. However, each time it does so, it has a probability b of copying an edge from one node to another, and a probability $1 - b$ of simply generating each end uniformly at random. If it copies, then it chooses a $(u, v) \in E$ and a node $u' \in V \setminus u$ uniformly at random. It then adds (u', v) to E. If it generates the edge at random, it acts identically to Tabakov-Vardi. This graph model extends to automata in the standard fashion.

Example 3.4. Figure 6 presents a vertex-copying automaton with $n = 8$, $r = 1$, $b = 0.3$, and $f = 0.3$. Notice that state 3 and state 6 have a large number of incoming transitions on 0 and 1, respectively. This is likely the result of a transition to that state on that character being copied to a number of other states.

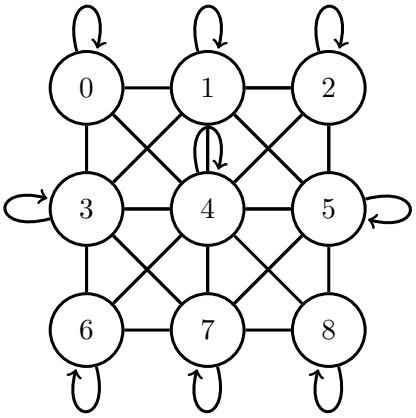
3.5 Kronecker Graphs

The Kronecker graph model, described by Leskovec et al. [11], uses an initial graph G as a seed and performs Kronecker exponentiation on G's adjacency matrix to create a larger, more complex graph. As Kronecker exponentiation is deterministic process, randomness emerges from giving each entry in the adjacency matrix a probability, rather than 1 or 0. The Kronecker model was designed to mimic properties displayed by real networks, such as email communications or gene regulatory networks, that evolve over time. These properties include heavy-tailed distributions for in- and out-degree, as well as a small diameter and densification.

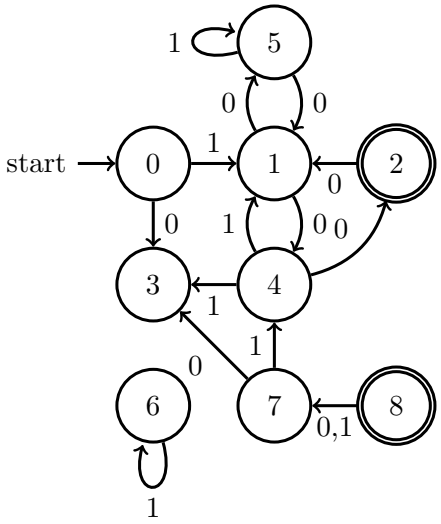
Kronecker graphs are built using Kronecker multiplication. Intuitively, the Kronecker multiplication of two matrices A and B, written $A \otimes B$, replaces each element of A with a copy of B multiplied by that element.



(a) The chain graph of length 3 (C3)



(b) The C3 graph to the second Kronecker power



(c) A Kronecker-C3 automaton based off the graph in 7b where $a = 0.35$, $b = 0.05$, and $k = 2$.

Figure 7: Example deterministic Kronecker graphs, and a randomized Kronecker automaton based on them.

However, instead of creating a 4D matrix of matrices, each copy of B is a "block" of multiple entries in the 2D representation of the result matrix C. More formally, given a matrix A of size $n \times m$ and a matrix B of size $n' \times m'$, the Kronecker product $C = A \otimes B$ is defined as the matrix C of size $n * n' \times m * m'$ where $C_{ij,kl} = A_{i,k} * B_{j,l}$. The Kronecker exponentiation of a $n \times n$ matrix A to the power k, written $A^{[k]}$, is the Kronecker product of k copies of a, or $a_1 \otimes ... \otimes a_k$. The matrix $A^{[k]}$ will be $n^k \times n^k$ in size. This iterates Kronecker multiplication in the same fashion that numerical exponentiation iterates numerical multiplication.

The random (a, b, G, k) Kronecker graph model takes as parameters the *seed graph* G of n nodes, two probabilities a and b used to replace 1 and 0, respectively, in the randomized adjacency graph, and an integral Kronecker power k. Let M be the $n \times n$ matrix where $M_{i,j}$ equals a if G has an edge from i to j, and b otherwise. $M^{[k]}$, or M raised to the kth Kronecker power, defines new $n^k \times n^k$ matrix. A random graph G' over the nodes $\{0, \ldots, n^k - 1\}$ is generated from this matrix. Each entry in $M^{[k]}$ represents the probability of an edge between the corresponding nodes. For each u and v, the edge (u, v) occurs with probability $M^{[k]}_{u,v}$. Like the small-world graph, the Kronecker graph can be extended to an automaton model using the standard lifting.

Example 3.5. An example Kronecker automaton is shown in Figure 7c with $k = 2$, $a = 0.35$, $b = 0.05$, and $f = 0.3$. G was C3, which can be seen in Figure 7a, and $G^{[k]}$ can be seen in Figure 7b. Notice that about a third of the edges from the original automaton are present, while no edges which were not present there are in the new graph, since b is so low.

3.6 Frank-Strauss Automata

The Frank-Strauss random graph model, based off an idea briefly presented by Frank & Strauss[2][6] that limits the space of possible edges. Instead of the vertices being integers, vertices are unordered pair of integers. The Frank-

[2]Referred to in their paper as a "Markov graph" - we call it the Frank-Strauss model to avoid unnecessary overloading of the term "Markov".

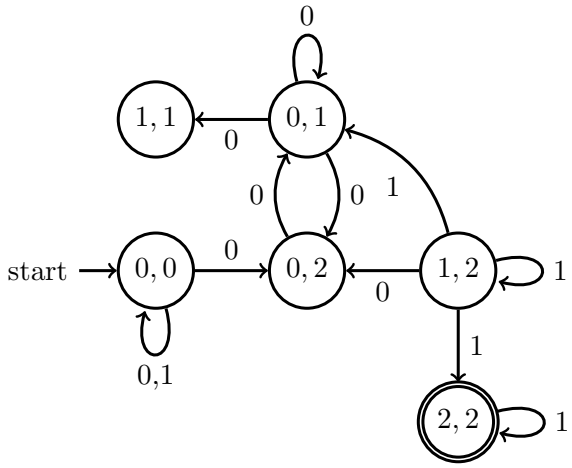


Figure 8: A Frank-Strauss automaton with $l = 2$, $r = 1.0$, and $f = 0.3$. No edge exists between two states that do not share a label.

Strauss model only permits edges between vertices that share an element – the vertex $(0, 1)$ can connect to $(0, 3)$ and $(1, 3)$, but not to $(2, 3)$. Within this space, edges are generated uniformly at random.

The Frank-Strauss model can be used to represent systems that require some sort of relationship between actors. For example, it can be used to represent binary relationships between individuals in a social setting. Alternatively, we may have a program such that if one module calls another, then they must be related somehow. However, it may be that they are dealing with the same data, or they could be performing the same kinds of operations. At least one of these has to hold.

An (l, r) Frank-Strauss random graph is parameterized by a *label size* l and a *transition density* r. The model defines the set V of vertices to be $\{(i, j) \mid i, j \in 0, \ldots, l-1\}$ of unordered pairs of elements. There are $\binom{l+1}{2} = \frac{l(l+1)}{2}$ such vertices. The model selects $\lfloor |V| * r \rfloor$ edges. To generate each edge, first a vertex (u_1, u_2) is chosen uniformly at random to be the source, and then a vertex $(v_1, v_2) \in \{u_1, u_2\} \times \{0, \ldots, l\}$ is chosen uniformly at random for the destination. To lift this model into a model of random automata, we modify the standard lifting only to define $Q_0 = \{(0, 0)\}$.

Example 3.6. Figure 8 shows an example Frank-Strauss automaton with $l = 2$ ($n = 6$), $r = 1.0$, and $f = 0.3$. Note that the only final state is $(2, 2)$. This means that the starting state $(0, 0)$ cannot connect to it directly - it has to travel through at least one intermediate state, since they do not share a label. Unfortunately, the appropriate connections do not exist - the language of this automaton is empty.

3.7 Accessible Automata

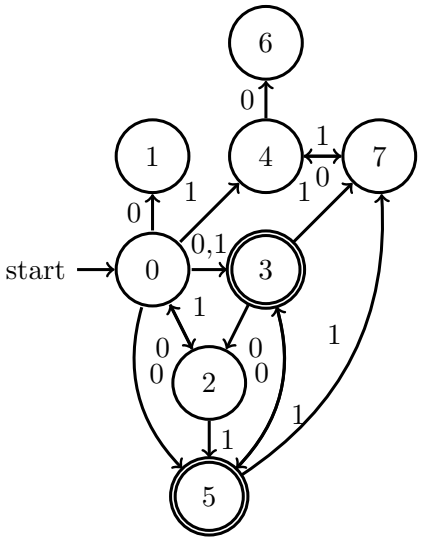


Figure 9: An accessible automaton with $n = 8$, $r = 1.0$, and $f = 0.3$. All states can be reached from the starting state. Notably, however, not all states allow you to leave once you have reached them. State 1 and state 6 both have no outbound transitions.

The accessible family of random automata models are not derived from random graphs. Loosely based on Leslie's generation of connected automata,[12] this family contains three different automata models - the accessible model, the co-accessible model, and the bi-accessible model. These models are named after a property that they guarantee resulting automata will possess. A state $q \in Q$ is *accessible* if there is a path *from* a starting state *to* q. A state $q \in Q$ is *co-accessible* if there is a path *from* q *to* an accepting

state. An automaton is accessible if every state is accessible, co-accessible if every state is co-accessible, and bi-accessible if it is both accessible and co-accessible.

Automata possessing these properties correspond to useful program properties - for example, a co-accessible automaton may specify that program that can recover and perform its intended function from any state. Similarly, an accessible automaton might be useful for verifying that a program can reach all of its intended states.

All models in the accessible family are parameterized by a *size* n, a *transition density* r, and an *accepting state density* f. Unlike other models, the accessible models do not define the transition relation based on a set of graphs. The accessible models start with an an empty transition relation δ and a set Q of states $\{0 \dots n-1\}$ They then construct a randomized spanning tree of transitions over the states. Lastly, the models add transitions to ensure that for each character in Σ, there are $\lfloor n * r \rfloor$ transitions in δ. As with most models, $Q_0 = \{0\}$ and F as $\lfloor n * r \rfloor$ random states from Q.

The accessible model adds transitions one at a time. For each edge, it selects an accessible state q, a non-accessible state r, and a $\sigma \in \Sigma$ uniformly at random. The transition (q, σ, r) is added to δ. The process is repeated until all states are accessible. The co-accessible models similarly adds transitions one at a time. In this, transitions are built from a co-accessible state r, a non-co-accessible state q, and a $\sigma \in \Sigma$ chosen uniformly at random. This is repeated until all states are co-accessible. The bi-accessible model first adds transitions to ensure every state is accessible. It then examines which states are still not co-accessible, and adds transitions to ensure all states are co-accessible.

Once the spanning tree has been constructed, the model must fill in the rest of the transition relation. Each character $\sigma \in \Sigma$ must be associated with exactly $\lfloor n * r \rfloor$ edges. If some σ has more than $\lfloor n * r \rfloor$ transitions, random transitions on σ are replaced by transitions on another character. The model then generates new edges uniformly at random for each character with fewer than $\lfloor n * r \rfloor$ transitions.

Example 3.7. Figure 9 shows an example accessible automaton with $n = 8$, $r = 1.0$, and $f = 0.3$. All states can be reached from the starting state. In fact, the starting state has a direct connection to every state except 6 and 7, and a path of length 2 to each of those. Note that it is not guaranteed that states allow you to leave once you have reached them - states 1 and 6 have no outbound transitions.

4 Experimental Results

The new random models we have developed are structured, in that they possess properties designed to mimic real-world problems. This fixes one of the issues preventing confidence in the T-V model. However, the utility of the T-V model is that it allows an empirical comparison of approaches to containment checking. We need to ensure that our models can be used for comparison. If they can't, then the models are not actually useful. The production of structured automata by itself is insufficient.

We identify two qualities that make a model well-suited to comparing approaches to universality. First, it should be *textured*: by varying configurations we can produce a range of universality probabilities in the resulting automata. Certain approaches to universality checking terminate early on universal automata, and other approaches terminate early on non-universal automata. Thus, a range of universality probabilities produced by different configurations is desirable. Additionally, it is important to have multiple configurations where the automata generated are neither overwhelmingly universal nor non-universal. These configurations tend to be extremely difficult for universality checkers, and are reasonably equal when comparing approaches that terminate early on universal automata with those that terminate early on non-universal automata. Second, it should be *stable*: a given configuration should produce a consistent universality probability for different sizes of automata. A stable model will allow us to compare universality checkers on increasingly large automata with confidence that increasing size with our chosen configurations actually results in problems that are merely larger. If the universality probability changed, it would actually be a differ-

ent problem instead of a scaled-up problem.

To test if a model is textured, we use a terrain experiment. In a terrain experiment, we fix the size of the automata and vary other parameters. We measure the probability of an automata being universal for each configuration. In an interesting model, we will see a range of probabilities. For Tabakov-Vardi, there are only two other parameters: transition and acceptance density. For some of these models, there are more. In this case, we fix the least interesting parameters to focus our attention on the two that seem most likely to produce textured behavior.

To perform a simple check if a model is stable, we compare terrain experiments on multiple sizes n. This allows us to compare the shape of the terrain and notice immediate differences over a small variation in n. If we wanted further assurance that the model was stable, we would need to run scaling experiments. In a scaling experiment, we fix the configuration at one point, and vary n, as opposed to fixing n and varying the configuration in a terrain experiment. We check if the universality probability of that configuration is consistent as we increase n. If, for a small number of configurations, universality probability stays relatively constant as the size of the automata increases, we can be confident that the model is stable. However, because of time limitations, we were not able to run scaling experiments, and intuit stability from the results of multiple terrains.

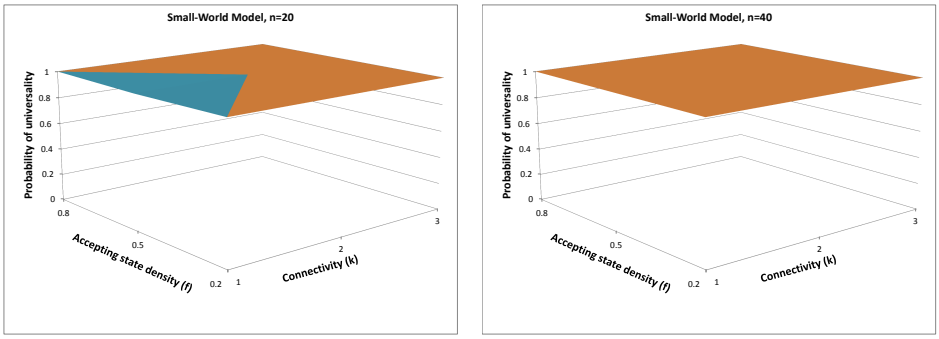
For each experiment, we generated a corpus of automata. We generated 100 automata for each configuration and size. The corpora used in the experiments were generated using a Haskell program and Haskell's standard pseudo-random number generator. We tested the resulting automata for universality using the Rank tool[4] on the DAVinCI cluster[3] at Rice University. We gave the rank tool one day to determine if each automaton was universal. If the rank tool timed out, we removed it from consideration. The percentage of automata that are universal at a given configuration is the number of automata that the tool called universal divided by the number of automata finished, whether universal or not, in the allotted time.

[3] http://www.rcsg.rice.edu/sharecore/davinci/

4.1 Small-World Automata

Small-world automata are highly clustered automata with low path length. Small-world automata are parameterized by an integral size n, an integral connectivity k that controls how many transitions are present, a rewire probability p that governs how many connections there are between clusters, and a final state density f.

We generated a corpus of (n, k, p, f) small-world automata with k drawn from $\{1, 2, 3\}$, p from $\{0.2, 0.5, 0.8\}$, and f was set to 0.2. We performed two terrain experiments, one with $n = 20$ and one with $n = 40$. We generated 100 examples of each configuration and tested them for universality.



(a) Small-world model, $n = 20$. **(b)** Small-world model, $n = 40$.

Figure 10: Small-world terrain experiments displaying the universality probability for $n = 20$ and $n = 40$. The connectivity k ranged from 1 to 3 and the rewiring chance p from 0.2 to 0.8. The final-state density f is 0.3. Almost every automaton was universal.

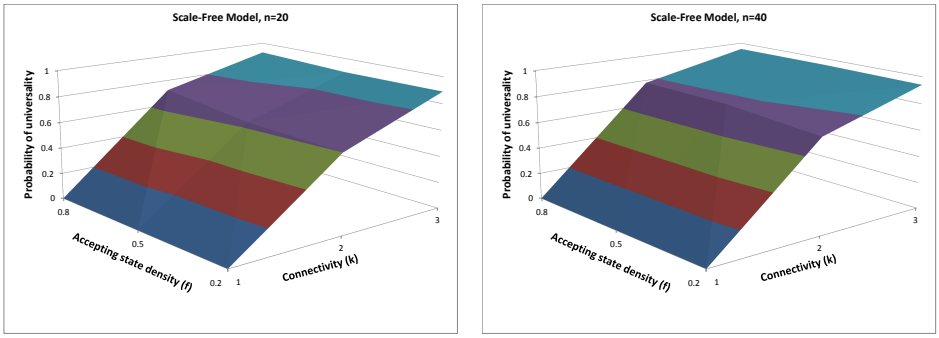
As seen in Figure 10, the small-world model unequivocally failed to be interesting: across both experiments, only one automaton was non-universal. While these results are disappointing, they are consistent with [15]. The minimum connectivity k for a small-world graph is 1, corresponding to a T-V transition density r of 2.0. At that transition density, the T-V model is already highly universal. It may be possible to create an interesting variant of the small-world model that allows for smaller transition densities. For

instance, we could use a directional ring lattice that connects each node to it's nearest neighbors on only one side, or we could select only some of the edges to nearest neighbors. However, these compromises may affect the clustering and low path length properties of small-world graphs.

4.2 Scale-Free Automata

Scale-free automata have a power-law degree distribution, so that a few states tend to have a very large portion of the transitions, turning them into "hubs". Scale-free automata are parameterized by an integral final size n, an integral starting size m, an integral connectivity k, and a final state density f.

For the terrain corpus of (n, m, k, f) scale-free automata, k is drawn from $\{1, 2, 3\}$, f from $\{0.2, 0.5, 0.8\}$, and m was set to 1. We performed two terrain experiments, one with $n = 20$ and one with $n = 40$ We created 100 examples of each parameter combination and tested them for universality.



(a) Scale-free model, $n = 20$. **(b)** Scale-free model, $n = 40$.

Figure 11: Scale-free terrain experiments displaying the universality probability for $n = 20$ and $n = 40$. The connectivity k ranged from 1 to 3 and the final state density f ranged from 0.2 to 0.8. The starting size $m = k$ in all configurations. Both graphs show that the scale-free model is textured.

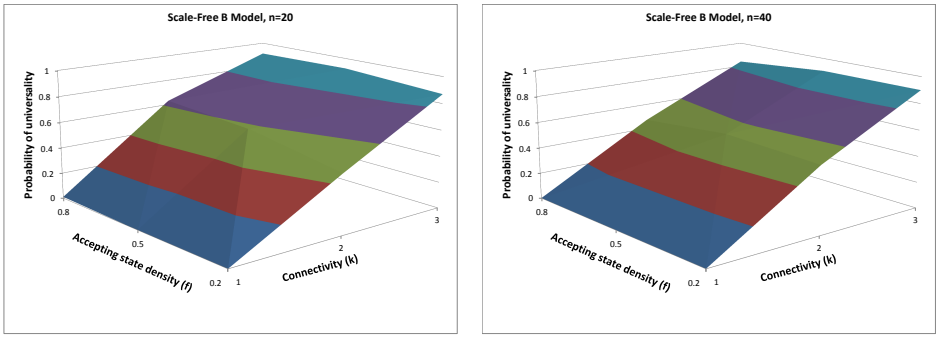
Note in figure 11 the range of probabilities from 0 to 1, that forms an upward slope across the graph. This indicates that the chance of universality

is changing as the parameters change - precisely telling us that the model is textured. The primarily unidirectional slope indicates that universality depends on k much more than on f. This correlates well with results from Tabakov-Vardi, since k in scale-free corresponds to r in T-V. In fact, there are no universal automata whatsoever at $k = 0$ in either graph.

Unfortunately, the model may not be stable - there is a small but noticeable increase in universality in the $n = 40$ graph as compared to the $n = 20$ one. It may be that at large sizes, the scale-free model is unable to generate automata with a variety of universality probabilities, and be instead limited to creating automata that have high universality probabilities. Proper scaling experiments need to be done on the scale-free model to verify this.

4.3 Scale-Free B Automata

Like scale-free automata, scale-free B automata have a power-law distribution of transitions. However, their size remains constant instead of growing over time. They are parameterized by an integral size n, an integral connectivity k, and a final state density f.



(a) Scale-free B model, $n = 20$. **(b)** Scale-free B model, $n = 40$.

Figure 12: Scale-free B terrain experiments displaying the universality probability for $n = 20$ and $n = 40$. The connectivity k ranged from 1 to 3, and the final state density f from 0.2 to 0.8.

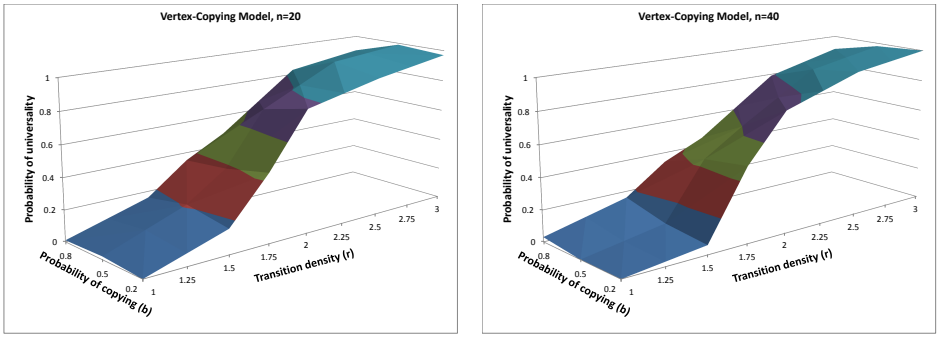
For the terrain experiments on (n, k, f) scale-free B automata, we use

essentially identical parameters to scale-free: k was drawn from $\{1, 2, 3\}$, and f from $\{0.2, 0.5, 0.8\}$. We performed two terrain experiments, one with $n = 20$ and one with $n = 40$. We created 100 examples of each parameter combination and tested them for universality.

The results are presented in Figure 12. Again like scale-free, the scale-free B model has a distinct slope upwards as k increases, so the model is nicely textured. Notably, the $n = 40$ case actually has a more level slope from $k = 1$ to $k = 3$ than $n = 20$, while still reaching similar heights. This may indicate that the model is very stable, which is supported by preliminary data from scaling experiments.

4.4 Vertex-Copying Automata

Vertex-copying automata achieve power-law scaling in a notably different way than the scale-free models, by copying edges from one vertex to another. Vertex-copying automata are parameterized by an integral size n, a copy probability b, a transition density r, and a final state density f.



(a) Vertex-copying model, $n = 20$ **(b)** Vertex-copying model, $n = 40$

Figure 13: Vertex-copying terrain experiments displaying the universality probability for $n = 20$ and $n = 40$. We ranged transition density r from 1 to 3, and copying probability b from 0.2 to 0.8. Notably, increasing b does not always increase universality.

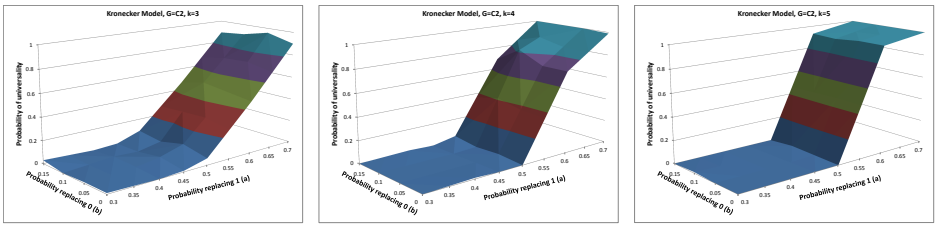
For the terrain corpus of (n, b, r, f) vertex-copying automata, we draw r from the set $\{1, 1.5, 1.75, 2, 2.5, 3\}$ and b from $\{0.2, 0.5, 0.8\}$. f is set to 0.3. We ran 2 experiments, with $n = 20$ and $n = 40$. We generated 100 examples for each parameter combination and tested them for universality.

Based on Figure 13, the vertex-copying model is definitely interesting, generating a range of terrain. The terrain sadly has a narrow set of useful configurations, however, ranging across a space of only .5 r. Notably, increasing b does not necessarily increase universality - after a certain tipping point, it actually reduces universality chance. This is presumably because nearly all edges are ending at the small number of nodes that were chosen as destinations for the first edges.

4.5 Kronecker Automata

Kronecker automata create a wide variety of properties occurring in real-world graphs through exponentiation of a seed graph. They are parameterized by a seed graph G, an integral Kronecker exponent k, edge probabilities a and b, and a final state density f.

For the terrain corpus of (G, k, a, b, f) Kronecker automata, we used a drawn from $\{0.3, 0.4, 0.45, 0.5, 0.6, 0.7\}$, b from $\{0, 0.05, 0.1, 0.15\}$, and set f to 0.3. The types of seed graphs used were chains, rings, stars, "Y"s which have a long tail and two short branches, and "Y-with-ring"s that had the two branches connect into a small ring. We denote the chain graph of size 2 as C2, the ring of size 3 as R3, and so on. $|G|$ was between 2 and 5. The size n of the automata is the size of G to the kth power. Since n is derived from two features that strongly affect the properties of the automaton, the notion of a configuration does not straightforwardly extend to Kronecker automata, and determining how to test the stability of Kronecker was not straightforward. To explore what good approaches there might be to stability, we performed a large number of different terrain experiments, using all combinations of G and k such that the resulting derived n was between 9 and 27. We produced 100 automata for each parameter combination and tested them for universality.

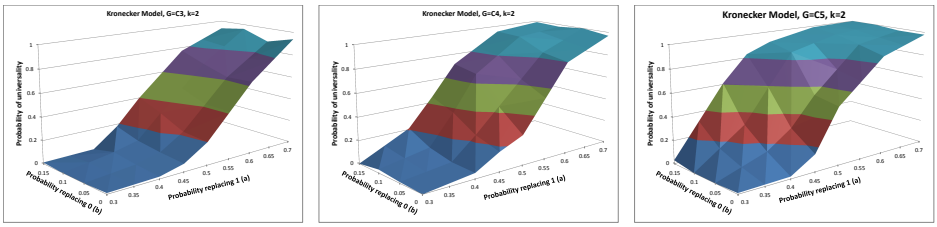


(a) Kronecker model, $G = C2, k = 3, n = 8$

(b) Kronecker model, $G = C2, k = 4, n = 16$

(c) Kronecker model, $G = C2, k = 5, n = 32$

Figure 14: Graphs from experiments on Kronecker automata with $G = C2$ and steadily increasing k. k ranges from 3 to 5, a from 0.3 to 0.7, and b from 0 to 0.15. Note that the slope becomes dramatically sharper, and because of this the range of usable configurations decreases dramatically. The range also shifts slightly, contributing to the lack of stability.



(a) Kronecker model, $G = C3, k = 2, n = 9$

(b) Kronecker model, $G = C4, k = 2, n = 16$

(c) Kronecker model, $G = C5, k = 2, n = 25$

Figure 15: Graphs from experiments on Kronecker automata with $k = 2$ and steadily increasing the size of G. G ranges from C3 to C5, a from 0.3 to 0.7, and b from 0 to 0.15. Note the fact that as $|G|$ increases, b becomes a more important determinant of universality, making the model increasingly unstable as b increases. However, at $b = 0$, it is much closer to being a stable model.

Originally, we anticipated that we would see the most stable results as size increased by incrementing k while holding G constant. However, on examination of the results[4], we discovered that this was not necessarily true. As can be seen in Figure 14, when we increase k, the textured qualities and useful range of the model decrease quickly. The sharp increase in slope means there are very few useful configurations to choose from, and likely even fewer as we continue to increase the size. The alternatives for increasing n are not much better. If we merely look at combinations of G and k which produce similar sizes, we find an utter lack of consistency - for example, compare 14b and 15b, both of which have 16 states, but produce completely different graphs. If we use progressively greater G, as seen in Figure 15, to scale the size of the graph, it produces a massive "hump" on the left side of the graph as b becomes more important to determining universality. However, the model is somewhat stable at $b = 0$ in the case of steadily increasing G to scale the size of the graph. While limiting b to 0 is rather restrictive and limits the utility of the model, this may be the best solution.

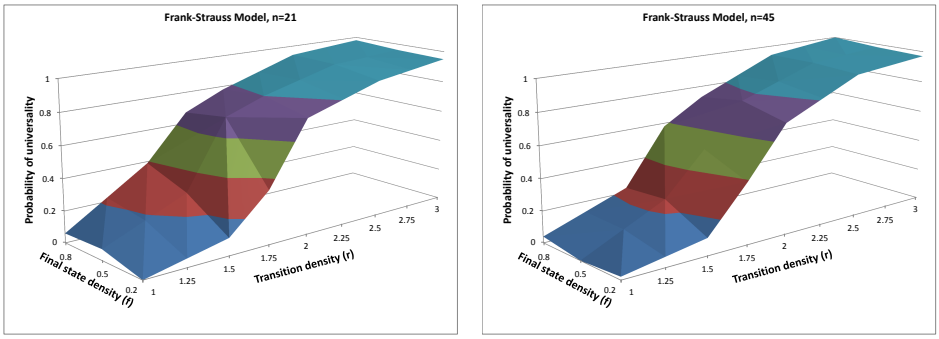
4.6 Frank-Strauss Automata

Frank-Strauss automata select transitions uniformly at random, much like the Tabakov-Vardi model. However, only certain transitions are valid, limiting the set of possible transitions to generate. The automata are labeled with a pair of integers, instead of one integer, and transitions are only valid between states that share at leach one label from the pair. They are parameterized by a label size l, a transition density r, and a final state density f.

For the terrain corpus of (l, r, f) Frank-Strauss automata, r was drawn from the set $\{1, 1.5, 1.75, 2, 2.5, 3\}$, and f from $\{0.2, 0.5, 0.8\}$. We ran 2 experiments, with $l = 7$ and $l = 10$. When an n is derived, these correspond to $n = 21$ and 45. We generated 100 automata and tested them for universality.

There is a definite presence of texture in the model. In Figure 16, it goes reasonably smoothly from 0 to 1 universality probability. The model seems

[4]This includes the results for other configurations of Kronecker. The other Kronecker graphs can be found in Appendix A.



(a) Frank-Strauss model, $l = 7, n = 21$ **(b)** Frank-Strauss model, $l = 10, n = 45$

Figure 16: Frank-Strauss terrain experiments displaying the universality probability for $n = 20$ and $n = 40$. We ranged r from 1 to 3 and f from 0.2 to 0.8. While the model seems stable at $f = 0.2$, the smaller case is affected to an abnormal degree by the f value, which seems to fade in the larger case. Thus, it may be best to run experiments on the Frank-Strauss model at $f = 0.2$.

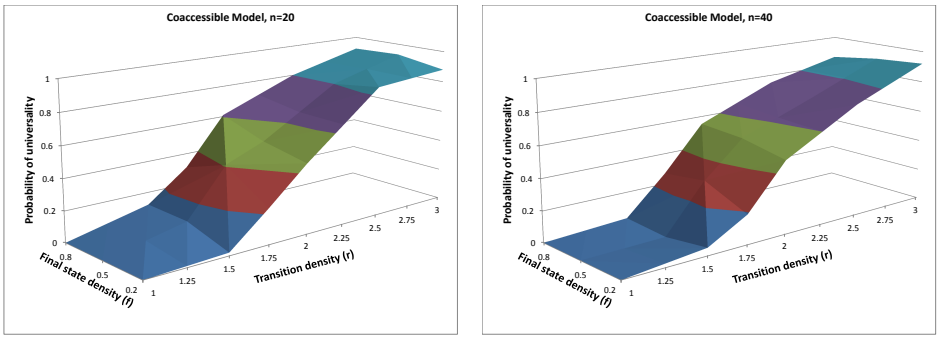
quite stable along r, with only minor changes in the $f = 0.2$ case between the two graphs. However, f has an abnormally high impact on universality probability in the $n = 21$ case, which seems to have been greatly reduced in the $n = 45$ experiment. It may be best to conduct scaling experiments on configurations where $f = 0.2$.

4.7 Accessible Automata

Accessible automata are a family of automata models that guarantee certain states can be reached from other states. Accessible automata guarantee that every state can be accessed by some path from a starting state, co-accessible automata guarantee that from every state there is a path that can access a final state, and bi-accessible automata guarantee both. They are parameterized by a size n, a transition density r, and a final state density f.

For all three models in the family of (n, r, f) accessible automata, r

was drawn from $\{1, 1.5, 1.75, 2, 2.5, 3\}$, and f from $\{0.2, 0.5, 0.8\}$. We ran 2 terrain experiments for each model, one at $n = 20$ and one at $n = 40$. For each configuration, we generated 100 automata and tested them for universality.



(a) $n = 20$ **(b)** $n = 40$

Figure 17: Co-accessible terrain experiments displaying the universality probabilities for $n = 20$ and $n = 40$, r ranges from 1 to 3, and f ranges from 0.2 to 0.8. Notice that, in both experiments, the slope was much shallower than in previous models. This gives us an extremely wide range of useful configurations for testing.

While all of the accessible automata types succeeded at being textured, the co-accessible results in Figure 17 were particularly notable.[5] Because the slope of the graph is relatively low, it provides much finer control over the chance of universality than the other types of accessible automata. In fact, the co-accessible model had the lowest slope of any model tested. Since the co-accessible graph is also stable, this is an extremely promising model.

5 Conclusion

We adapted structured random graph models into six new random automata models using the graph-lifting technique pioneered by Tabakov and Vardi.

[5] While only the co-accessible graphs are shown here, the rest of the graphs are included in Appendix B.

While the technique was straightforward in most cases, the Frank-Strauss model required a variant state labeling. We further formalized a family of accessible automata models, which were not based on graphs.

To see if the models were useful, we tested them for texture and stability. We implemented generators for each model in Haskell and created corpora containing automata of various configurations. We determined the chance of universality for each configuration by checking the generated automata using the rank-based universality solver. By examining the terrain of the models we were able to judge the texture and complexity of the model for the purpose of universality experiments, and by comparing the terrains at different sizes we acquired a rough idea of the stability of models.

Six of the seven new models proved textured. The small-world model is not textured, and therefore not useful. The co-accessible model and scale-free B model have a particularly wide range of useful configurations. The vertex-copying and Kronecker models also have notably interesting terrain. The universality of those models relies more heavily on both parameters than other models, which depend almost exclusively on connectivity or transition density.

Of the six models that were textured, all look to have at least some stability. The Kronecker and Frank-Strauss models are stable, but only at certain points on the graph. This essentially limits the Kronecker and Frank-Strauss to changing only one parameter. Most of the other models exhibit only minor variation. The scale-free B model may be more promising than the others for stability - it seems to get increasingly stable as n increases.

There are four paths of future work. First, further tests on the stability of the models are needed through using scaling experiments on universality probability. Second, it may be possible to modify the small-world model to be more textured by allowing lower or fractional connectivity. Third, a complementary avenue of research would analyze programs to determine what properties they possess. Do modular programs exhibit small-world properties? Can the use of libraries be described as scale-free? Finally, we have yet to establish that these structured models provide more insight into the behavior of universality checkers than Tabakov-Vardi. To test this, we

need to compare universality checkers on corpora derived from our models and Tabakov-Vardi. Different models may exaggerate the differences in performance between approaches. A plausible and interesting outcome may be that different approaches are better suited to different models.

References

[1] Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr, and Tomáš Vojnar. Simulation subsumption in ramsey-based büchi automata universality and inclusion testing. In *Computer Aided Verification*, pages 132–147. Springer, 2010.

[2] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[3] Nizar N Batada, Laurence D Hurst, and Mike Tyers. Evolutionary and physiological importance of hub proteins. *PLoS Computational Biology*, 2(7):e88, 2006.

[4] Laurent Doyen and Jean-François Raskin. Improved algorithms for the automata-based approach to model-checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 451–465. Springer, 2007.

[5] Seth Fogarty and Moshe Y Vardi. Efficient büchi universality checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 205–220. Springer, 2010.

[6] Ove Frank and David Strauss. Markov graphs. *Journal of the American Statistical Association*, 81(395):832–842, 1986.

[7] Xavier Gabaix, Parameswaran Gopikrishnan, Vasiliki Plerou, and H Eugene Stanley. A theory of power-law distributions in financial market fluctuations. *Nature*, 423(6937):267–270, 2003.

[8] Luke Hakes, John W Pinney, David L Robertson, and Simon C Lovell. Protein-protein interaction networks and biologywhat's the connection? *Nature biotechnology*, 26(1):69–72, 2008.

[9] Richard M Karp. The transitive closure of a random digraph. *Random Structures & Algorithms*, 1(1):73–93, 1990.

[10] Jon M Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S Tomkins. The web as a graph: Measurements, models, and methods. In *Computing and combinatorics*, pages 1–17. Springer, 1999.

[11] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *The Journal of Machine Learning Research*, 11:985–1042, 2010.

[12] Ted Leslie. Efficient approaches to subset construction. Technical report, University of Waterloo, Canada, 1995.

[13] Raj Kumar Pan and Sitabhra Sinha. Modular networks with hierarchical organization: The dynamical implications of complex structure. *Pramana*, 71(2):331–340, 2008.

[14] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic. 1985.

[15] Deian Tabakov and Moshe Y. Vardi. Experimental evaluation of classical automata constructions. *LPAR*, pages 396–411, 2005.

[16] Deian Tabakov and Moshe Y. Vardi. Model checking Büchi specifications. 2007.

[17] Ming-Hsien Tsai, Seth Fogarty, Moshe Y Vardi, and Yih-Kuen Tsay. State of büchi complementation. In *Implementation and Application of Automata*, pages 261–271. Springer, 2011.

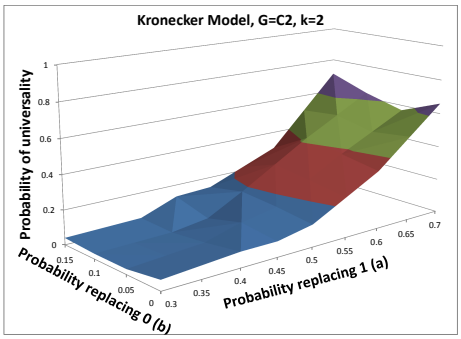
[18] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata, volume 1043 of Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.

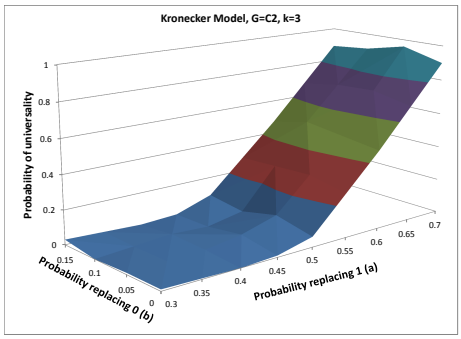
[19] Duncan J Watts and Steven H Strogatz. Collective dynamics of small-world networks. *nature*, 393(6684):440–442, 1998.

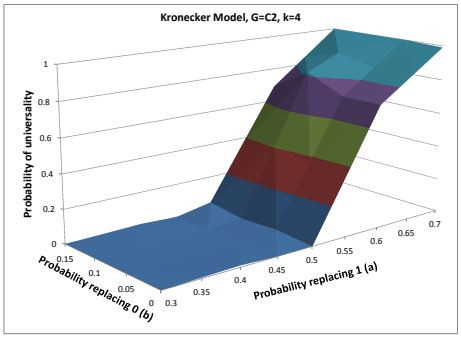
A Appendix: Kronecker Model

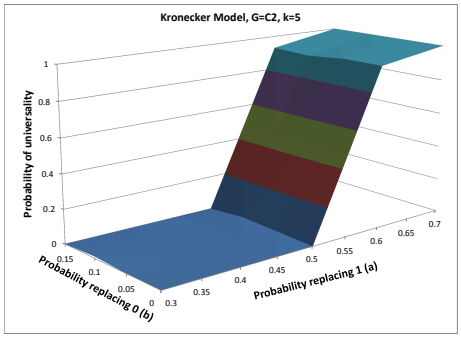
This appendix contains experimental results for further configurations of Kronecker automata.

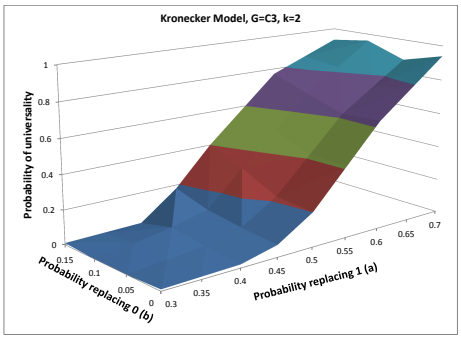
A.1 Automata using chain graphs as seeds

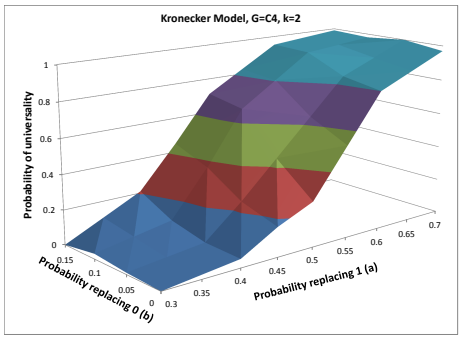


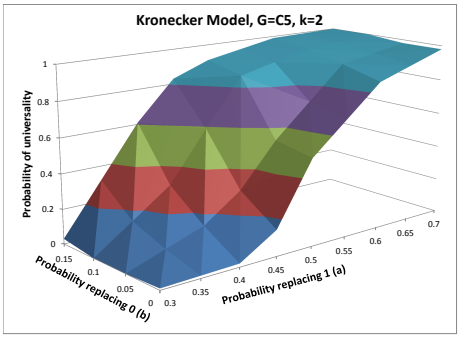




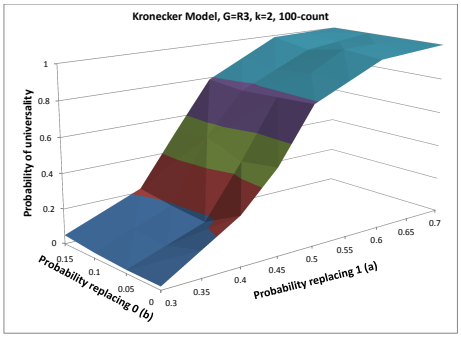


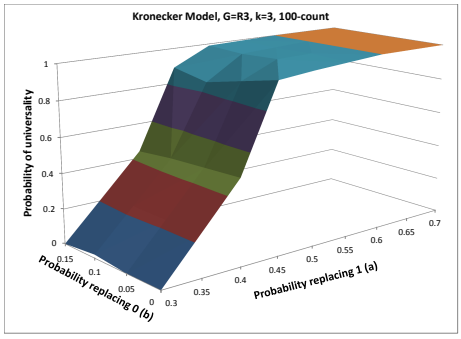

Kronecker Model, G=C3, k=2
Probability of universality
1
0.8
0.6
0.4
0.2
0
Probability replacing 0 (b)
0.15
0.1
0.05
0
Probability replacing 1 (a)
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7

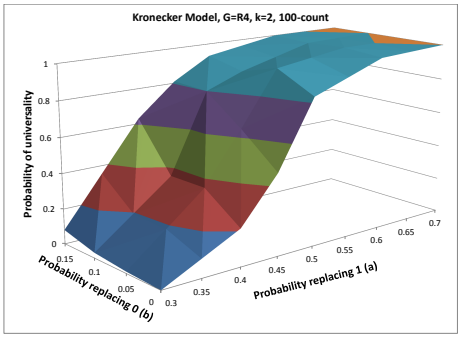

Kronecker Model, G=C4, k=2
Probability of universality
1
0.8
0.6
0.4
0.2
0
Probability replacing 0 (b)
0.15
0.1
0.05
0
Probability replacing 1 (a)
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7

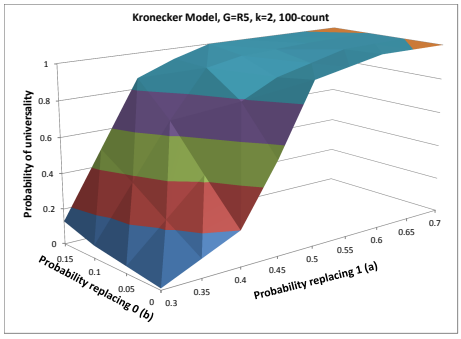

Kronecker Model, G=C5, k=2
Probability of universality
1
0.8
0.6
0.4
0.2
0
Probability replacing 0 (b)
0.15
0.1
0.05
0
Probability replacing 1 (a)
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7

A.2 Automata using ring graphs as seeds

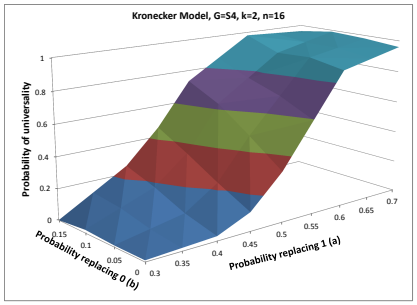


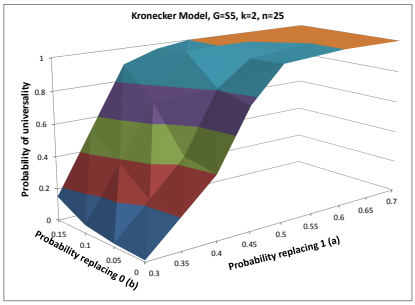


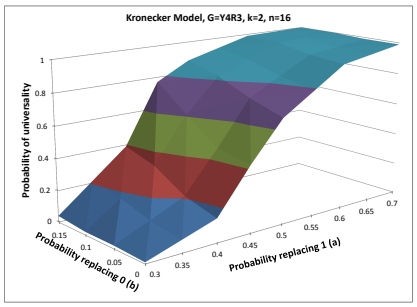


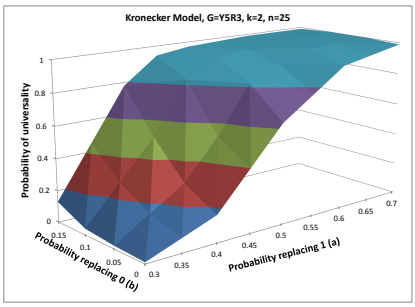


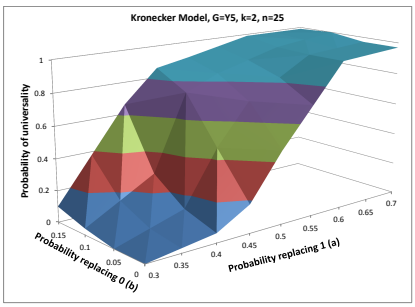
A.3 Automata using star or y-shaped graphs as seeds.











B Appendix: Accessible Models

This appendix contains experimental results for accessible and biaccessible automata from the accessible family of models.