**Distribution Agreement**

In presenting this thesis or dissertation as a partial fulfillment of the requirements for an advanced degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis or dissertation in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis or dissertation. I retain all ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Signature:

| | |
|---|---|
| Andrés Celis | Date |

**APPROVAL SHEET**

A Stochastic Model for Networks that Arise from Conference Scheduling Problems

By

Andrés Celis

Master of Science

Computer Science

_____

Michelangelo Grigni, PhD

Advisor

_____

James Lu, PhD

Committee Member

_____

Vicki Hertzberg, PhD

Committee Member

Accepted:

Lisa A. Tedesco, Ph.D.

Dean of the James T. Laney School of Graduate Studies

_____ Date

# A Stochastic Model for Networks that Arise from Conference Scheduling Problems

By

Andrés Celis
M.S., Emory University, 2015

Advisor: Michelangelo Grigni, Ph.D.

An abstract of
A thesis submitted to the Faculty of the Graduate School
of Emory University in partial fulfillment
of the requirements for the degree of
Master of Science
in Mathematics and Computer Science
2015

**Abstract**

A Stochastic Model for Networks that Arise from Conference Scheduling Problems
By Andrés Celis

A conference scheduling problem may be viewed as an undirected graph whose vertices correspond to the events of the conference, and whose edges correspond to constraints that prohibit two events from being scheduled at the same time. In this thesis we propose and analyze a new random graph model inspired by a series of experimental observations on datasets from the industry, as well as conversations with a conference scheduler.

Our models differs from existing random graph models in the following:

1. We find that graph models with independently chosen edges do not result in degree distributions found in conference scheduling problems. Thus our model's edges are statistically dependent on each other.

2. Our model introduces new vertices into the graph as time evolves. The existing models that do this have small, bounded clique and chromatic numbers and are trivial to color, which is not an accurate representation of scheduling problems. We show that the expected clique number of our model has a lower bound of $\Omega(T^{1/4}/(\log T)^{3/4})$. We also argue that the expected clique and chromatic numbers of our model are upper bounded by $O(T^{1/4})$ and $O(T^{2/5})$, respectively.

# A Stochastic Model for Networks that Arise from Conference Scheduling Problems

By

Andrés Celis
M.S., Emory University, 2015

Advisor: Michelangelo Grigni, Ph.D.

A thesis submitted to the Faculty of the Graduate School
of Emory University in partial fulfillment
of the requirements for the degree of
Master of Science
in Mathematics and Computer Science
2015

**Acknowledgments**

I wish to express my sincere gratitude to my advisor, Michelangelo Grigni, whose guidance was invaluable. I am grateful to my family for their endless love and support, this thesis would not have been possible without them.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Charles Haws of the Society of Biblical Literature schedules conferences. For a given conference, he has several hundred talks to schedule in a few time slots, but he must honor conflict constraints; in particular, pairs of talks cannot be scheduled together because of a common speaker. There are multiple speakers per talk and many speakers are involved in multiple talks. In the worst case, scheduling problems are computationally difficult, but we observed that Haws' instances were relatively easy to solve. We propose an explanation for this in terms of a stochastic social network model and go on to analyze its properties.

There were several characteristics of Haws' datasets that stood out to us:

1. The high degree of collaboration between the speakers of these conferences: it is likely they are presenting a talk together because they coauthored a paper or report.

2. The graphs that arose from Haws' datasets tended to have a low average degree, with only a small, high degree "core" of the graph that was difficult to schedule.

3. The bipartite nature of this problem: a set of talks that have conflict constraints imposed on them by a set of speakers.

We believe that these characteristics are essential to getting an accurate representation of conference scheduling problems. Reviewing the literature, we were unable to find a stochastic model that captured these features, so we propose a stochastic model which we designed with these characteristics in mind.

# Chapter 2

# Graph Coloring and Existing Graph Models

## 2.1 Graph Coloring

In the Graph Coloring Problem (GCP), one is given an undirected graph $G = (V, E)$, with $V$ being the set of $|V| = n$ vertices and $E$ being the set of edges. A *k-coloring* of $G$ is a mapping $\phi : V \to \Gamma$, where $\Gamma = 1, 2, ..., k$ is the set of $|\Gamma| = k$ integers, each one representing a color. We say that a coloring is *valid* if for each $(u, v) \in E$ we have $\phi(u) \neq \phi(v)$; otherwise it is *invalid*. The *chromatic number* of a graph, $\chi(G)$, is the smallest $k$ for which a valid k-coloring can be found. If the vertices of a graph represent the events of a conference, and there is an edge between two vertices if the corresponding events cannot be scheduled at the same time, then finding a valid coloring for the graph is equivalent to creating a schedule for the conference, since each color can be thought of as a timeslot.

Many random graph models have been proposed in order to study and better understand networks that appear in the real world such as genetic networks, interbank payment networks, airline networks, radio frequency networks, and the World Wide Web. In this section, we overview some of these

existing models and discuss why they do not accurately represent conference scheduling problems.

## 2.2 Erdős-Rényi Model

An Erdős-Rényi (ER) graph[8], $G(n,p) = (V,E)$, is constructed with $|V| = n$ vertices, where each of the $\binom{n}{2}$ possible edges have probability $p$ of being added to the edge set $E$. The expected number of edges in the graph, $G(n,p)$, is $\binom{n}{2}p$, and the ER model has a binomial degree distribution. A clique is a subset $K \subset V$, such that for all $k_i, k_j \in K$, there is an edge $(k_i, k_j) \in E$. The size of the largest clique in a graph is called the *clique number*, $\omega(G)$. By definition, $\omega(G) \leq \chi(G)$. The ER model is well studied and it has been shown that, $E[\omega(G)] \sim \frac{2\ln(n)}{\ln(1/p)}$ [3], and $E[\chi(G)] \sim \frac{n}{2\ln(n)}$[2].

We believe this model is an inappropriate representation of a typical conference scheduling problem for several reasons. First, there is no natural way to introduce scheduling constraints in the ER model, as the only parameter that influences the structure of the graph is $p$. Second, it does not capture the bipartite nature of conference scheduling. Additionally, the average degree in the ER model is proportional to the size of the graph, whereas in conference scheduling we observe a constant average degree. Although the model does have a small, high degree "core", as evidenced by $\omega(G)$ being $\theta(\log(n))$, the high average degree of the graph masks the location of the "core, which makes the graph much harder to schedule. This can be seen by the large gap between $\omega(G)$ and $\chi(G)$, as well as the slower performance of coloring algorithms on the ER model. This contrasts with networks that arise from conference scheduling problems, where the difficult "core" is easily identifiable among the low degree vertices.

## 2.3 Barabási-Albert Model

In a Barabási-Albert (BA) graph[1], $G(n, m, m_0) = (V, E)$, the construction process begins with an initial network of $m_0 \leq n$ vertices, usually connected as a cycle. A new vertex, $v_t$, is then added at each timestep $t = m_0 + 1, m_0 + 2, ..., n$, so at the end of the construction $n = |V|$. Each vertex $v_i$ is connected to $m \leq m_0$ existing vertices on timestep $i$, with a probability that is proportional to the degree of the existing vertices. The probability that a new vertex is connected to vertex $v_j$ is $d_j / \sum_i d_i$, where $d_i$ is the degree of vertex $i$.

The BA model has a constant average degree, which is desirable for representing conference scheduling problems. It also has a power law degree distribution, which asserts that the number of vertices with degree $k$ is proportional to $k^{-\beta}$ for some exponent $\beta \geq 1$[10]. This implies that the model has many low degree vertices, and a small high degree "core".

The reason we find the BA model inapropiate is that it has a small, bounded $\chi(G)$, and therefore $\omega(G)$, and is trivial to color. This is due to the construction process of the model. Imagine that you are allowed to "observe" the construction process, and that the intial graph of size $m_0$ has been colored. We know $\chi(G) \leq m_0$, and at each timestep a vertex is added to the graph with fixed degree $m$. Coloring this newly added vertex, with a simple greedy coloring rule will require no more than $m + 1$ colors, so we can conclude that $\chi(G)$ stays constant as the graph grows, and no large clique emerges. This is an unrealistic setting when it comes to conference scheduling problems.

## 2.4 Evolving Copying Model

The evolving copying (EC) model[9], has similar characteristics to the BA model, such as a power law degree distribution and a constant average degree.

An EC graph, $G(n, p, d) = (V, E)$, also begins with a small initial graph of size $\geq d$. Then, at each timestep $t = d+1, d+2, ..., n$ a vertex is added to the graph. Each vertex added selects a "prototype" vertex uniformly at random from the existing graph. The newly introduced vertex then randomly chooses $d$ edges to existing vertices, with probability $p$ of choosing a vertex uniformly at random, and $p-1$ of copying an edge from its prototype. The EC model is inappropriate for reasons similar to the BA model.

## 2.5 Coloring Algorithms

The GCP has many real world applications such as air traffic flow management, register allocation, frequency assignment, timetabling and scheduling. As such, there has been much work put into finding efficient algorithms for the GCP. In this section, we overview some of the algorithms presented and analyzed in Chirandini et al.[6] We go on to measure their performance on Haws' datasets and our model.

### 2.5.1 Greedy

Given a graph $G = (V, E)$, an ordering of $V$, $(v_1, v_2, ..., v_n)$, and a set of colors $\Gamma = 1, 2, ..., k$, the greedy algorithm has $n$ coloring steps, where at each step $i$, it assigns $v_i$ the smallest color in $\Gamma$ that does not cause a conflict with the already colored graph. The performance of the greedy algorithm depends heavily on the ordering of the the vertices. For example, when coloring a complete bipartite graph with the edges of a perfect matching removed, if all of the vertices in one of the sets appear before all the vertices of the other set, the greedy approach will find the optimal coloring of 2 colors. However, if the ordering is an alternation between vertices of the two sets, the greedy approach will use up to $n/2$ colors.

### 2.5.2 DSATUR

Similarly to the greedy approach, the DSATUR heuristic uses an ordering of vertices, however, the ordering of the remaining vertices is recomputed at the end of each coloring step. The initial ordering of the vertices is found by sorting the vertices by degree. Afterwards, the order is decided in part by the partial coloring in the graph. At the end of each coloring step, the remaining vertices are ordered by the number of distinct colors that their neighbors have been assigned. Ties are broken by degree in the uncolored portion of the graph and further ties are broken arbitrarily. The idea behind this approach is to try to color the most difficult vertices first, and a good indication of a vertex being "difficult" to color is its degree and the amount of distinct colors its neighbors are using.

### 2.5.3 RLF

Given a graph $G = (V, E)$, the Recursive Largest First algorithm recursively builds the largest independent set in a graph that it can find, given some heuristic or stochastic local search approach. RLF then assigns the independent set a color, removes it from the graph, and repeats the process. An independent set of a graph is a subset of the vertices in the graph, $I \subset V$, where no two vertices in $I$ are adjacent. This definition is complementary to that of a clique, where all pairs of vertices in a subset have an edge between them. A common method for building an independent set is to create a set $P$ of potential vertices to add to the indenpendent set $I$, and a set $U$ of vertices which are unavailable to the independent set. $I$ and $U$ begin as the empty set and $P$ begins as a copy of $V$. Then the vertex with highest degree to vertices in $U$ is removed from $P$ and added to $I$, and all its neighbors are moved into $U$. The initial vertex added to $I$ can be a random choice or it can be motivated by some heuristic. For a more detailed description we refer

the reader to [5].

### 2.5.4 Ex-DSATUR

We use the above algorithms to be able to quickly find upper bounds on the chromatic number of a graph. However, in order to find $\chi(G)$ exactly, we use Michael Trick's implementation of Ex-DSATUR. This algorithm uses the DSATUR heuristic, but explores many vertex orderings. To avoid exploring unnecessary orderings, it uses the notion of *tight colorings* to prune the search tree. For a more technical description, we refer the reader to [11] and [14].

A graph for which $\omega(G) = \chi(G)$ is called *perfect*, and if $\omega(G)$ is close to $\chi(G)$ the graph is called *quasi-perfect*. For graphs that are perfect or near perfect, Ex-DSATUR can effectively prune the search tree and performs well[6].

# Chapter 3

# Haws Model

In this section, we introduce the "Haws" model, which we believe is representative of typical conference scheduling instances. It is also based on a plausible social model for how communities with a high degree of collaboration develop. We model Haws' conference scheduling problem as a bipartite graph $B = (U, V, E)$ where $U$ is the set of speakers, $V$ is the set of talks, and $E$ is the set of edges corresponding to the constraints on the talks by the speakers.

## 3.1 General Description

We construct a Haws graph in $T$ steps. At each timestep of the construction $t$, from $1, 2, ...T$, where $T$ is the number of talks, we introduce a vertex $u_t$ to the set $U$, and a vertex $v_t$ to the set $V$, and an edge $(u_t, v_t)$ to the edge set $E$. The intuition behind this is that there is a lead speaker for each talk. At timestep 1, vertex $u_1$ is added to $U$, $v_1$ to $V$, and $(u_1, v_1)$ to $E$. At timestep 2, $u_2$ is added to $U$, $v_2$ to $V$, and $(u_2, v_2), (u_2, v_1)$ to $E$. At timestep $j$, $u_j$ and $v_j$ are introduced with an edge between them, and $u_j$ gets $minimum(j, d)$ additional edges to previously introduced vertices in the set $V$, where d is a parameter to the graph. The methods used to choose these additional edges is discussed in section 4.2. If $u_i$ and $u_j$ both have an edge to some $v_k$, then the two speakers $(u_i, u_j)$ are involved in the same talk $(v_k)$.

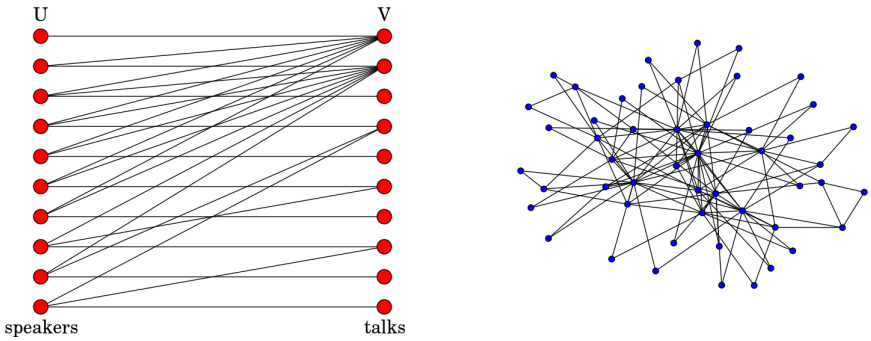


Figure 3.1: Example constructions. Above left: bipartite graph when T = 10 and d = 2. Above right: square of bipartite graph on the vertex set $V$, when T = 50 and d = 2.

While we believe this construction leads to an accurate representation of Haws' datasets, this form of the graph does not lend itself to coloring or schedule-finding algorithms since it includes the speaker set $U$. However, by taking the square of $B$ and then restricting it to the talk set $V$, the resulting graph $G = (B^2|_V, B[E]^{(2)})$ has an edge $(v_i, v_j) \in B[E]^{(2)}$ if and only if talks $v_i$ and $v_j$ have a common speaker (i.e. they cannot be scheduled in the same timeslot). In other words, the vertex set of the square graph, $B^2|_V$, is simply the set $V$ from the bipartite graph, and the edge set of the square graph, $B[E]^{(2)}$, is comprised of edges between vertices $v_i$ and $v_j$ when there is a path of length two between them in the bipartite graph. Note that $G = (B^2|_U, B[E]^{(2)})$ would have much larger cliques since some vertices in $V$ have very high degree.

Alternatively, we can define the bipartite graph as an adjacency matrix $A$, of size $|U| \times |V|$, where $A_{i,j} = 1$ if $u_i$ has an edge to $v_j$, and $A_{i,j} = 0$ otherwise, for $i = 1, 2, ..., |U|$, $j = 1, 2, ..., |V|$. The adjacency matrix of the square of the graph $= S$, restricted to the talk set, can then be found by the boolean matrix multiplication $S = AA^T$.

Thus, if we can find a valid k-coloring for $G$, where k is the number of

timeslots available, we have created a schedule for the conference.

## 3.2 Construction Process and Choosing Edges

In this section we go into more detail of how edges are chosen when introducing a new vertex $u_t$ with it's corresponding $v_t$, and Algorithm 1 describes the construction process of the graph based on parameters.

Choosing a vertex "by degree" means that an edge from the graph will be chosen uniformly at random, and the appropriate endpoint will be selected (thus the probability of a vertex being selected is proportional to its degree). We refer to this method of picking the $d$ random edges as the "standard" model.

This method of choosing edges draws inspiration from the Barabasi Albert model, but our full generator also produces variants of the Haws model described above. One of the variants contains elements of the EC model, where on the introduction of each vertex $v_t$ for $t = 1, 2, ..., T$ a "parent" vertex is picked. The parent could be chosen either uniformly at random, or with probability proportional to its degree. The vertex $v_t$ would then have some probability $p$ of copying edges from its parent, and probability $1 - p$ of choosing an edge from the graph. Whether choosing from the graph is done uniformly or by degree is another variant to consider. We call this the "copy" model.

Additionally, our generator can also produce graphs for the more realistic case where $|U| > |V|$ or $|U| >> |V|$, that is, when there are more speakers than talks in a conference. We have some probability $q$ of taking a "speaker" step. That is, of introducing a speaker $u_t$ without a corresponding talk $v_t$ at that step. We call this the "speaker" model. The speaker can then be related to pre-existing talks by one of the methods mentioned above. The full graph generator for both the standard model and its variants is available

in the appendix.

Like the graph models mentioned earlier, BA and EC, our model also has a power law degree distribution. However, an important difference between our model and the existing ones is that large cliques can emerge through the construction process of our model. As discussed above, the other two power law degree distribution models have a small, bounded $\chi(G)$, and are trivial to color if the generation order of the vertices is known.

We argue that this does not hold true in the Haws model. Suppose we have an initial graph that is colored. At each timestep we add a new vertex with fixed degree, but we also add an edge between existing vertices in the graph. This can cause conflicts, since two vertices that had the same color could find themselves connected by an edge later in the construction process, forcing one of them to be recolored. We conjecture that $\omega(G)$, and therefore $\chi(G)$, is unbounded as the graph grows. We go on to experimentally and analytically determine how fast they grow.

**Algorithm 1** Graph Generator

Parameters:
T - the number of talk vertices, $|V| = |U|$.
d - the number of random edges each speaker vertex in $U$ gets to $V$.
// Define a bipartite graph, $B = (U_B, V_B, E_B)$, speaker set $U_B$, talk set $V_B$. $E_B$
// defines the edges of the graph, it is a vector of adjacency lists where each
// entry is indexed by a speaker vertex.
$U_B \leftarrow \emptyset$
$V_B \leftarrow \emptyset$
$E_B \leftarrow$ *vector of size n where every entry is an adjacency list*
// For every timestep, from 1 to T
**for** $t < T$ **do**
    // Add $u_t$, to $U_B$, $v_t$ to $V_B$.
    $U_B.add(u_t)$
    $V_B.add(v_t)$
    // Add edge $v_t$ to the adjacency list corresponding to $u_t$.
    // This defines the edge $(u_t, v_t)$.
    $E_B[u_t].add(v_t)$
    // Add minimum(d,t-1) distinct, random edges from $u_t$ to $V_B$.
    $i \leftarrow 0$
    **while** $|E_B[u_t]| < minimum(d, t-1) + 1$ **do**
        // Choose a vertex $v_j$ from $j = 0, 1, ..., t-1$ by degree.
        $v_j$ = getVertexByDegree()
        // Ensure $v_j$ is not already adjacent to $u_t$
        **if** $v_j \notin E_B[u_t]$ **then**
            // Add edge. Since we insist on distinct edges,
            // we can add the edge to the graph now, without
            // affecting the relative probabilities of the
            // vertices being chosen at later steps.
            $E_B[u_t].add(v_j)$
// Now we take the square of the bipartite graph, restricting it to $V_B$. Define
// a squared graph, $S = (V_S, E_S)$. $V_S$ is the vertex set, $E_S$ is the new edge set.
$V_S \leftarrow V_B$.
$E_S \leftarrow \emptyset$.
**for each** $u \in U_B$ **do**
    **for each** pair $(v_i, v_j) \in E_B[u]$ **do**
        // Here we use that set operations (automatically throws out duplicates).
        $E_s.add((v_i, v_j))$

# Chapter 4

# Analysis of Haws Model and Future Work

## 4.1 Experimental Results

The standard model can be created from the graph generator in the appendix with the following configuration of parameters: p = 1.0, s = 0.0, uniformParent = false, and uniformChild = false. The following results, unless stated otherwise, are conducted on the standard model with d = 2. We call this variant the "triangle" model, as it adds a 3-clique to the graph at every timestep in the construction.

### 4.1.1 Degree Distribution Results

The degree distribution of the vertices in $V$, after taking the square of the bipartite graph, is, at least experimentally, a power law distribution. This is unsurprising given that the probability of a vertex $v_i \in V$ being selected as an endpoint of an edge in timestep $t > i$ is proportional to its degree. This gives preference to high degree vertices and is often called "the rich get richer." The power law asserts that the number of vertices with degree $k$ is proportional to $k^{-\beta}$ for some exponent $\beta \geq 1$.

A power law degree distribution implies that the graph is mostly sparse, with only a small set of vertices having very high degree. We believe this is one reason why Haws' instances are relatively easy to color.

### Haws Dataset Distributions

In this section we discuss the degree distributions of the datasets we received from Haws. We believe our model is a good representation of a realistic conference scheduling problem for the following reasons:

- The degree sequence roughly approximates that of Haws' datasets.
- Our generator is based on preferential attachment schemes, which simulates the social nature of collaborating on papers or reports.
- Through discussions with with Haws and experiments, we noted that there tended to be many low degree vertices and a small, high degree "core" of the graph that was difficult to schedule. Both of these attributes come naturally with the power law.

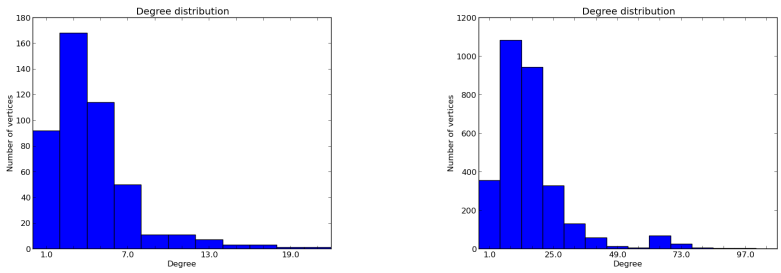


Figure 4.1: Degree distributions of two of Haws' datasets, standard axes.

We show a degree distribution as a histogram, that is, we split the range of degrees in a graph into a series of evenly sized intervals, and we count the number of vertices that appear in each interval. These histograms show that

Haws' datasets have many low degree vertices, which is consistent with the sparsity of graphs with a power law degree distribution.

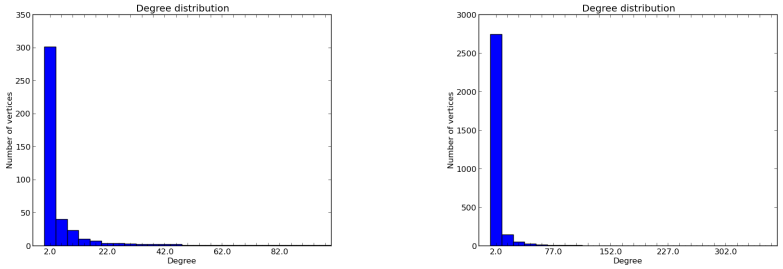


Figure 4.2: Degree distributions from our "triangle" model for graphs of similar size to Haws' datasets.

### Stochastic Model Distribution

Here we show the degree distribution as a loglog plot. We choose this over a histogram because a consequence of the power law distribution is that it looks like a straight line when plotted on a loglog scale. This was helpful in experimentally verifying that our "triangle" model was an example of the power law.

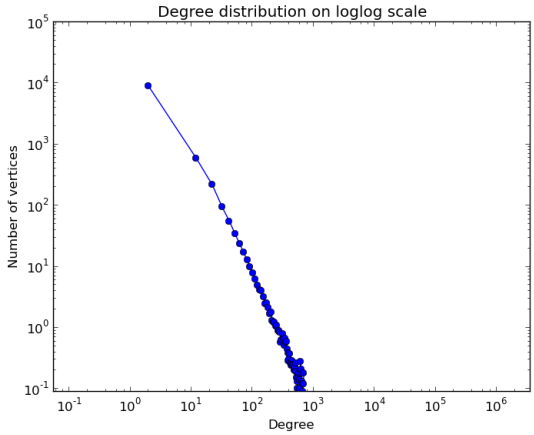


Figure 4.3: Degree distribution averaged over 100 "triangle" model graphs.

### Estimated Beta

In a power law distribution, the number of vertices with degree $k$ is proportional to $k^{-\beta}$ for some exponent $\beta \geq 1$. To help show that Haws graphs have power law degree sequences, we estimated the exponent $\beta$ on some of the variants for our model.

| q | d=2 | d=3 | d=4 | d=5 |
|---|---|---|---|---|
| 0.0 | 1.45 | 1.50 | 1.54 | 1.53 |
| 0.1 | 1.46 | 1.49 | 1.53 | 1.58 |
| 0.2 | 1.47 | 1.52 | 1.54 | 1.59 |
| 0.3 | 1.48 | 1.52 | 1.56 | 1.59 |
| 0.4 | 1.49 | 1.54 | 1.58 | 1.61 |
| 0.5 | 1.49 | 1.55 | 1.59 | 1.65 |
| 0.6 | 1.51 | 1.58 | 1.62 | 1.68 |
| 0.7 | 1.53 | 1.59 | 1.69 | 1.75 |
| 0.8 | 1.57 | 1.66 | 1.71 | 2.04 |
| 0.9 | 1.65 | 1.70 | 2.00 | 2.03 |

Table 4.1: $\beta$ estimates for Haws graphs of size 10,000 averaged over 200 trials. $d$ defines the number of edges we add at each timestep of the graph generation and $q$ is the probability of taking a "speaker" step. Note that the top left estimation is for what we call the "triangle" model.

In order to find the estimates for $\beta$, we use PLFIT, the goodness-of-fit based method described in Clauset, Shalizi, Newman[12][7]. The fitting procedure works as follows:

1. Given a vector $x$, for each possible choice of the minimum value $x_{min}$, they estimate $\beta$ via the method of maximum likelihood, and calculate the Kolmogorov-Smirnov goodness-of-fit statistic $D$.

2. The $x_{min}$ value that gives the minimum value of $D$ over all values of $x_{min}$ is then selected as the estimate of the true $x_{min}$, and the corresponding $\beta$ is reported.

### 4.1.2 Clique and Chromatic Number Results

The chromatic number of a graph, $\chi(G)$, is important to study in our model since it defines the minimum number of timeslots a conference needs in order

to create a schedule with no conflicts. We use several algorithms to give us an idea of the difficulty of scheduling instances from our model, and to help determine the growth rate of $\chi(G)$. In addition to using an exact solver for $\chi(G)$, we also use a clique finding algorithm to find $\omega(G)$, which gives a lower bound on $\chi(G)$. To find upper bounds on $\chi(G)$, we use approximation and stochastic local search algorithms. We find that the growth rate of $\omega(G)$ is logarithmic, and that there is little separation between the values of $\omega(G)$ and $\chi(G)$, at least within the range of realistic conference sizes.

When running the exact solver, Ex-DSATUR, on the datasets we received from Haws, the chromatic number was found in under a second. We believe that this is due in part to the low average degree of the graph and to $\omega(G)$ being close to $\chi(G)$, since Ex-DSATUR has been shown to perform well in such cases.

We also relate our model to the Erdős-Rényi model by defining a subset of vertices which has a constant fraction of the possible edges in the set. This makes the subset, $S$, similar to an ER graph $G(|S|, p)$. Relating our model to the ER model helps us give analytical bounds on $\omega(G)$ in section 6.

### Clique Number Growth

In addition to providing a lower bound on the growth rate of $\chi(G)$, we want to study $\omega(G)$ to confirm that our model is providing a realistic representation of typical conference scheduling problems. That is, we wish to have a model that has a small, high degree core that is more difficult to color. In the plot below, since the x-axis is on a log scale, we see that a straight line indicates a logarithmic growth rate of $\omega(G)$. We measured $\omega(G)$ with the clique finding program "cliquer"[13].

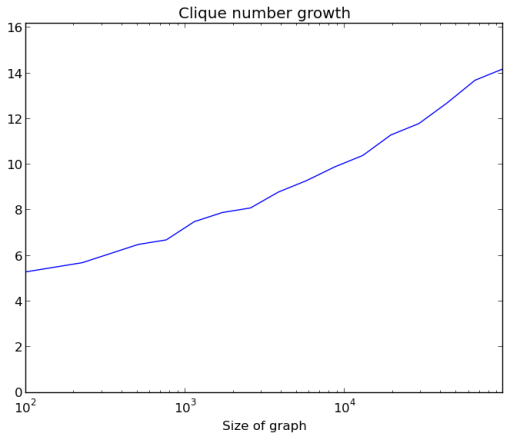


Figure 4.4: The averaged max clique size as T grows from 5,000 to 100,000, with the x-axis in log scale.

This convex curve is consistent with clique size $T^{1/4}$. The growth rate indicates that there is a high degree core of the graph, which is unsurprising. We know the average degree of the graphs generated by our model is at most 6 when $d = 2$ because we add at least 2 edges and at most 3 edges every time we add a vertex to the graph. However, due to the preferential attachment scheme of our generator, we know there is a small set of vertices that received high degree early in the graph generation, and the "rich get richer" process allowed them to maintain their high degree.

### Celebrity Vertices

In this section, we introduce the notion of "celebrity" vertices, those vertices which have very high degree compared to the rest of the network. We name them celebrity vertices since at many timesteps, they were "chosen" by other vertices, and so they are "popular."

It becomes important to study these vertices when analyzing the growth rate of $\omega(G)$ and $\chi(G)$ because they are the most likely vertices to be involved

in the part of the graph that is difficult to color. Thinking about it in terms of trying to color the graph as it is being generated, these vertices are the most likely to be "chosen" by a later vertex and gain an edge that could cause a conflict in the current partial coloring and force a recoloring among the vertices.

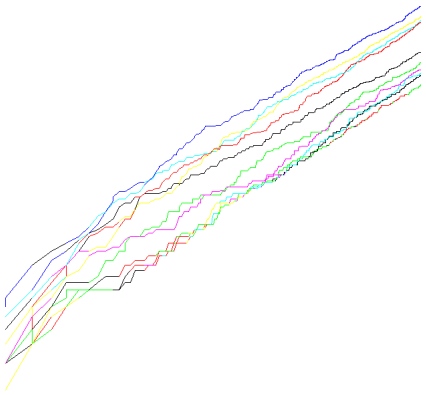

Figure 4.5: A view of how the degrees of the top 10 celebrities evolve over time on a loglog scale.

The figure above shows the growth rate of the top ten celebrity vertices on a loglog scale. The straight lines indicate that the corresponding vertices have polynomial degree growth with respect to the size of the graph, and the fact that the slopes are parallel indicates that the vertices have the same exponent in their polynomial growth. We define a set $S$, which holds all the celebrity vertices of a graph. We would like to know how large the set of celebrities tends to be.

**Size of Celebrity Set** An alternative way of thinking about the set of celebrity vertices is as a set that has a constant fraction of the possible edges within that set. Only very high degree vertices have a high likelihood of being

connected, since the only way for two existing vertices to get connected in our model is for both to be chosen by a vertex being introduced on a later timestep, and the likelihood of being chosen is proportional to their degree. Since a constant fraction of edges are present in $S$, we can relate it to the ER model.

We find the size of the celebrity set in two different steps. First, we add vertices to the celebrity set $S$ in generation order, as those have the best chance of having high degree. We continue adding the vertices to $S$ as long as more than 50% of the edges are present in $S$. The top plot of the three below shows the average size of $S$ with this construction as a function of the size of the graph. We plot it on a loglog scale in order to determine whether the growth rate is polynomial, and if so, to get a estimate on how large the exponent of the polynomial is.

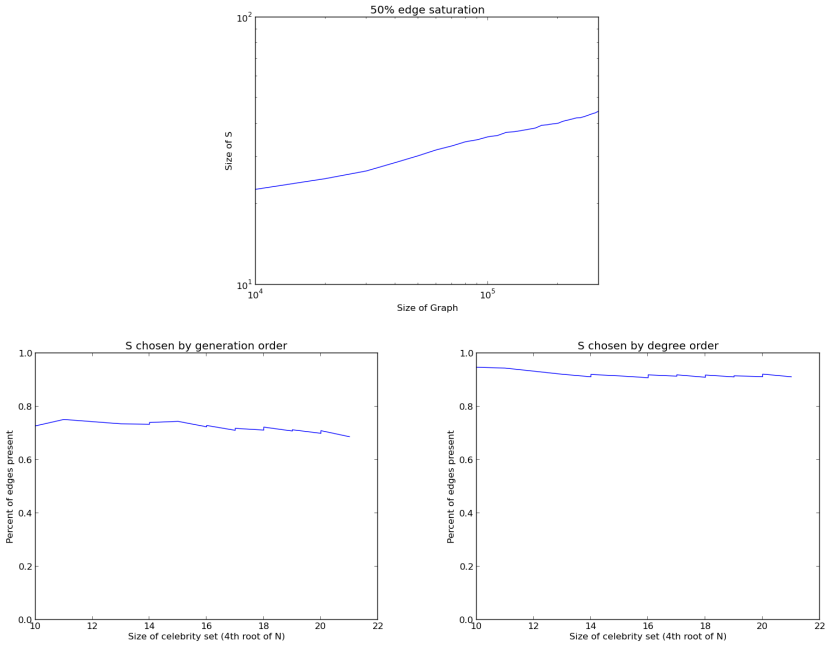


Figure 4.6: Estimations of the size of the celebrity set.

The bottom two figures represent a way of verifying the size of $S$. We choose a size for $S$ and see what fraction of the possible edges are present as a function of the size of the graph. We chose to look at the top $\sqrt[4]{T}$ vertices, where $T$ is the size of the graph, because this number was suggested by the slope of the first plot. In the bottom left plot, we choose the first $\sqrt[4]{T}$ introduced by the graph, i.e., in generation order. In the bottom right figure we choose the same number of vertices but ordered by their degree, so they are guaranteed to be the highest degree vertices in the graph. As such, there is a jump in the fraction of possible edges present.

**Arrival Rate of Edges to Celebrity Set** In order to be able to estimate at what rate edges are added to $S$, we partially generate a graph, and identify the highest degree vertices and add them to $S$. After building $S$ we no longer add any more vertices to it as the graph continues to grow. Instead, we keep track of how many new edges are arriving to $S$ as the graph grows. We found that edges arrive at a high rate to $S$ initially, and then there is a sharp decline in incoming edges. Since we do not allow duplicate edges, this is unsurprising because as the number of edges in $S$ increases, there is a higher chance of an edge being "rejected", i.e., the endpoints of an already existing edge being chosen by the newly introduced vertex.

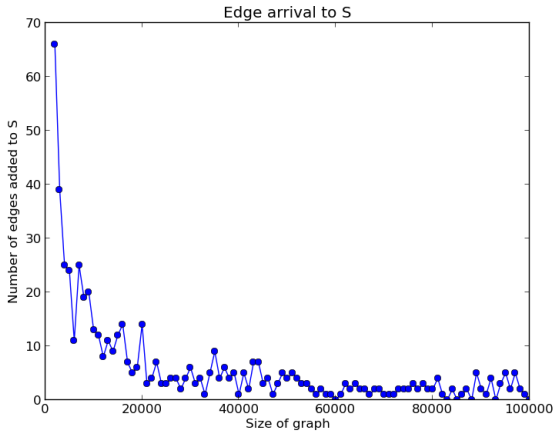


Figure 4.7: The number of edges added to the celebrity set, per every 1,000 timesteps as the size of the graph increases from 1,000 to 100,000.

In the figure above we see that $S$ becomes saturated early in the graph generation process and afterwards few new edges are added simply because there is a low percentage of edges that do not already exist in the set.

**Left Degree** This experiment attempts to measure how large the average left degree is in the celebrity set $S$. When the vertices are named by the timestep they were generated $(1, 2, ..., T)$, the "left degree" of vertex $i$ is the number of edges that vertex $i$ has to vertices in the set $\{1, 2, ..., i-1\}$. The left degree of a vertex $v$ does not include the two edges that are introduced upon the arrival of $v$.

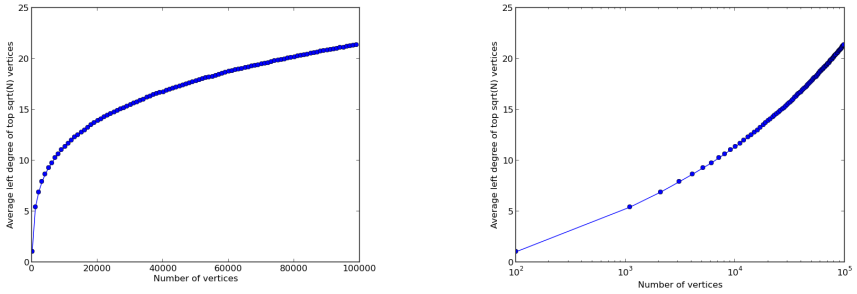


Figure 4.8: Average left degree in $S$ on normal and loglog scales.

The above figure on the left is plotted on a normal scale, but since it was difficult to distinguish whether the the growth was logarithmic or a small factional power of $T$, we additionally plotted the the x-axis on a log scale. Since the curve is clearly concave up we can see that the growth is greater than logarithmic. This supports our argument that $S$ becomes saturated as the size of the graph increases.

### Chromatic Number Growth

It is important to note that the chromatic number is experimentally close to the clique number for the Haws model, i.e. $\chi(G) \approx \omega(G)$. Due to this, Ex-DSATUR performs well since it can effectively prune the search tree.

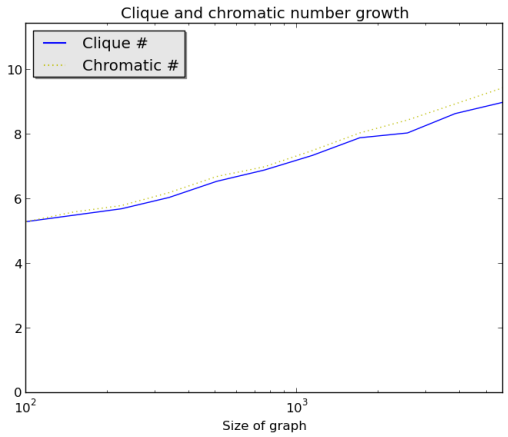


Figure 4.9: Plot of $\chi(G)$ and $\omega(G)$, x-axis in log scale.

The clique and chromatic number seem to grow at the same rate, with only a small gap between them. The gap does widen for larger sizes of the graph, but the average difference between the clique and chromatic numbers remains less than 1 for sizes up to $10,000$. This indicates that $\omega(G) \approx \chi(G)$ for realistic conference sizes.

### Timing and Accuracy Results for Heuristics

We also believe that the instances generated by the Haws model are easy to color because stochastic local search algorithms such as DSATUR and RLF, detailed in chapter 2, also perform well on these instances. They find good approximations of $\chi(G)$, and run in a fraction of the time of Ex-DSATUR. The times for the datasets given to us by Haws are solved in a similar amount of time as instances of the model of the same size.

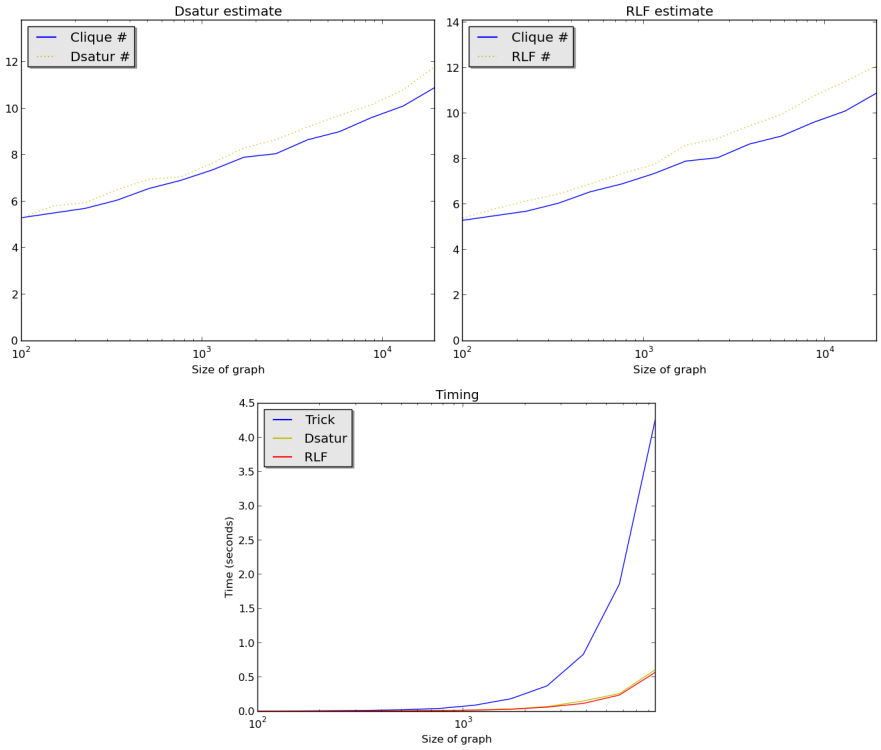


Figure 4.10: Accuracy and time performance of coloring algorithms. The top two figures have the x-axis on a log scale

Even the greedy coloring approach, when coloring the vertices by their generation order, or coloring them in descending degree order, performed very well. This is due to the small core of the graph that is difficult to color. When the vertices are ordered by degree or in generation order, the core of the graph is colored first, and afterwards the rest of the graph is trivial to color. This can be seen in the below figures, where we plot the approximate chromatic number found by the greedy approach, and the coloring timestep that maximum color was needed.

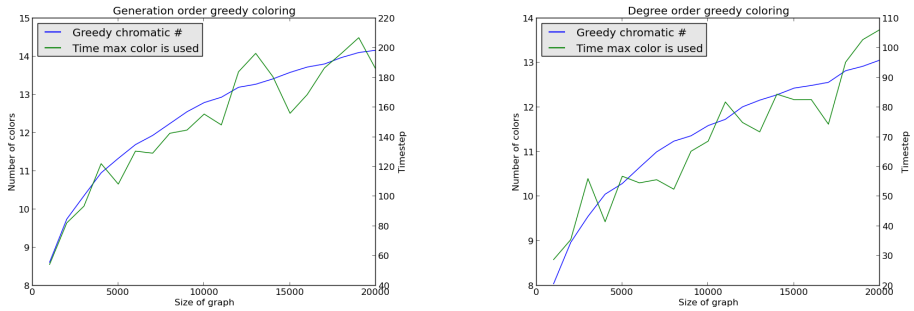


Figure 4.11: Colors needed by the greedy coloring approach given vertex ordered by generation and by degree. Overlaid with the timestep that the maximum color was used

Note that in the greedy coloring figures, the timestep that the maximum color was needed often drops dramatically before growing again. The intuition for this is that as the graph develops, more edges are added to the high degree portion of the graph, which tends to consist of vertices that arrived early in the generation process. This forces the maximum color to be used earlier in coloring process, causing the sudden drops. Eventually, however, enough edges are added to the graph that additional colors are needed, and the new maximum color is used at a later timestep.

Since this range of sophisticated to simple coloring algorithms ran quickly and effectively for all instances of reasonable size (conferences do not have tens of thousands of events), we felt it was unnecessary to formulate the problem for other solvers such as CPLEX or SAT solvers.

### Preprocessing Heuristic

In addition to our analysis of existing coloring algorithms, we wanted to evaluate the effectiveness of a preprocessing heuristic for graph coloring. This experiment uses the preprocessing technique of recursively removing all vertices from a graph whose degree is less than $d$. We know all the vertices

removed can be colored with at most $d$ colors if we add them back to the graph in reverse order.

We find the point where the number remaining vertices becomes less than the value of $d$. This point, which we call $d^*$, is of interest because it provides an upper bound on the chromatic number. Additionally, this preprocessing technique should quickly reveal the hard core of the graph.

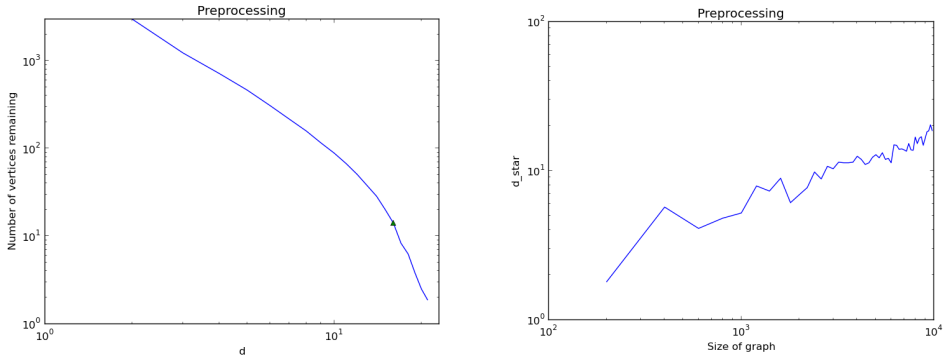


Figure 4.12: Preprocessing figures on loglog scale.

In the preprocessing figure on the left, we plot the number of vertices that remain in the graph for different values of $d$. The first value of $d$ is 2. Since we know all vertices have degree $\geq 2$, the first point shows the original size of the graph. Clearly the graph reduces in size dramatically as soon as $d \geq 3$ . We mark the point where $d$ is larger than the size of the remaining graph with a green triangle. The preprocessing figure on the right shows $d^*$, the point at which $d$ is larger than the size of the remaining graph, as a function of the size of the graph. Note that $d^*$ gives an upper bound on $\chi(G)$, and seems to grow as a fractional power of the size of the graph.

## 4.2 Analysis

Our goal here is to establish simple lower and upper bounds on the expected clique number and chromatic number in the Haws model. In particular our analysis shows they are both polynomial (rather than logarithmic).

In Section 4.2.1 we introduce the "triangle model" $G_T$, a simple version of the Haws model. In Section 4.2.2 we estimate the expected degree of the $k$th vertex in $G_T$. In Section 4.2.3 we argue a lower bound, that the clique number (and hence the chromatic number) is at least $\Omega(T^{1/4}/(\log T)^{3/4})$. In Section 4.2.4 we argue upper bounds, showing $\omega(G_T) = O(T^{1/4})$ and $\chi(G_T) = O(T^{2/5})$.

### 4.2.1 The Triangle Model

Here we specify a version of the Haws-network model that we will analyze, the "triangle model". The edges in our network are undirected, but we do allow self-loops and parallel edges. (Such edges do not increase the clique or chromatic number.) For $T \geq 3$, we present a $T$-step process producing a random network $G_T$ on vertices $v_1, v_2, v_3, \ldots, v_T$.

The first three steps are fixed: in step 1, vertex $v_1$ appears; in step 2, vertex $v_2$ appears with an edge to $v_1$, and in step 3, vertex $v_3$ appears with edges to both $v_1$ and $v_2$. So $G_3$ is a triangle, where each vertex has degree 2. After that, for each time step $t \geq 4$, we add one new vertex $v_t$ and three new edges to $G_{t-1}$, producing $G_t$. The three edges are a triangle chosen as follows:

- Among the previous vertices $v_1, \ldots, v_{t-1}$, we pick a vertex $v_i$ with probability proportional to its degree in $G_{t-1}$. Or in other words: we uniformly pick an edge of $G_{t-1}$, and then we uniformly pick one of its two endpoints as $v_i$.

- We pick a second vertex $v_j$ from the identical distribution (so possibly

we have $v_j = v_i$).

- We add the new vertex $v_t$, and the three edges $\{v_i, v_t\}$, $\{v_j, v_t\}$, $\{v_i, v_j\}$. Note that if $v_i = v_j$, we are adding a self-loop and two parallel edges. If $\{v_i, v_j\}$ is already an edge, then we add it again as a parallel edge. In all cases the total degree increases by six: $+2$ at $v_i$, $+2$ at $v_j$, and 2 at the new vertex $v_t$.

Let $G_T$ denote this random graph after $T$ steps. We make the following preliminary observations about $G_T$:

- The network $G_T$ has $T$ vertices, $3(T-2)$ edges, and total degree $6(T-2)$.

- In each triangle, the edge $\{v_i, v_j\}$ is distinguished because it connects two existing vertices of $G_{t-1}$, we call it the *hypotenuse* edge of the triangle. If we omitted these hypotenuse edges from our construction, then our model would be a version of the Barabási-Albert model. Let $G'_T$ denote $G_T$ with all of its hypotenuse edges removed. Then $G'_T$ is 3-colorable: as each $v_t$ is added, give it a color not already taken by $v_i$ and $v_j$.

- We did not allow $v_i = v_j$ in our standard Haws network generator, but it is more convenient for analysis to allow such events. Based on our degree estimates below, we see that "$v_i = v_j$" only occurs $\Theta(T^{1/3})$ times during the construction of $G_T$. Such events do not add to the clique or coloring numbers, since they contribute a trivial hypotenuse edge.

- For $1 \leq k \leq T$, let $D_{k,T}$ be the degree of vertex $v_k$ in $G_T$ (a random variable), and let $D_T$ be the degree vector $(D_{1,T}, \ldots, D_{T,T})$. If we want to predict the vector $D_{T+1}$, then it is sufficient to condition on $D_T$, there is no further useful information in the graph $G_T$.

### 4.2.2 Degree Estimates

As above, let $D_{k,T}$ denote the degree of vertex $v_k$ in $G_T$. Initially we have $D_{k,k} = 2$. In step $T+1$, the probability that $v_k$ is chosen as $v_i$ is $D_{k,T}/(6(T-2))$. That is also the probability that it is chosen as $v_j$. Either of these independent events would increase the degree of $v_k$ by 2. Let $d_{k,T}$ be its expectation $E[D_{k,T}]$. Then we have this recurrence:

$$d_{k,T+1} = d_{k,T} \cdot \left(1 + \frac{4}{6(T-2)}\right)$$

Using $(1+\epsilon) \sim e^{\epsilon}$, we derive this estimate:

$$\begin{aligned} d_{k,T} &= 2 \cdot \prod_{t=k}^{T-1} \left(1 + \frac{4}{6(t-2)}\right) \\ &\sim 2 \cdot \exp\left(\sum_{t=k}^{T-1} \frac{4}{6(t-2)}\right) \\ &\sim 2 \cdot \exp\left(\frac{2}{3}(\ln(T-2) - \ln(k-2))\right) \\ &\sim 2 \cdot (T/k)^{2/3}. \end{aligned}$$

The leading constant is accurate for large enough $k$. We remark that this degree distribution is consistent with the power law exponent $\beta = 2.5$. By Azuma's inequality (applied to a Doob martingale for $D_{k,T}$) we can also derive a concentration bound ([4], Theorem 1): $Pr\left[|D_{k,T} - d_{k,T}| \geq \alpha 4\sqrt{T}\right] \leq \exp(-\alpha^2/2)$.

### 4.2.3 Lower Bounds

Fix $T$. In this section, we argue that the expected clique number of $G_T$ is $\Omega(T^{1/4}/(\log T)^{3/4})$. This is also a lower bound on its chromatic number.

Consider the graph $G_T$. We let $n = T^{1/4}/(100(\ln T)^{3/4})$, and we say that

vertex $v_k$ is a "celebrity" in $G_T$ if $1 \leq k \leq n$. These celebrity vertices have expected degree $d_{k,T} \geq 2(T/n)^{2/3} \geq 40\sqrt{T \ln T}$. Say a celebrity is "good" if $D_{k,T} \geq \frac{1}{2} d_{k,T}$. By the concentration bound above, a celebrity fails to be good with very small probability (at most $e^{-10\sqrt{lnT}}$). Let $n'$ be the number of good celebrities; we expect $n'$ is at least $n/2$, if not (which is unlikely) we give up and start over.

Suppose we generate $G_T$, and pick good celebrities $v_i$ and $v_j$: $v_i$ has degree at least $(T/i)^{2/3}$ in $G_T$, and $v_j$ has degree at least $(T/j)^{2/3}$, and these are both at least $20\sqrt{T \ln T}$. Now continue evolving the graph for $T$ more steps, arriving at $G_{2T}$. We want to lower-bound the probability that $\{v_i, v_j\}$ was added as a hypotenuse edge, during these last $T$ steps. We note that on each step, the probability that we do pick both $v_i$ and $v_j$ is at least $(T/i)^{2/3}/(12T) \cdot (T/j)^{2/3}/(12T) \geq (20\sqrt{T \ln T})^2/(144T^2) \geq 2.5(\ln T)/T$. Repeating the experiment $T$ times, we see that the probability that the edge was never added is at most $(1 - 2.5(\ln T)/T)^T \leq e^{-2.5 \ln T} = T^{-2.5}$. Therefore, of the $\binom{n'}{2}$ potential edges between good celerities, the expected number that were not added is only $\binom{n'}{2} \cdot T^{-2.5} \leq T^{-2}$, so with high probability we have a clique of size $n'$.

### 4.2.4 Upper Bounds

In this section we consider upper bounds in $G_T$. However, these arguments are only "plausible" arguments, they are less rigorous than our lower bounds in the previous section. We argue that the expected chromatic number is $O(T^{2/5})$, and the expected clique number is $O(T^{1/4})$.

First we consider the clique number. Let $n = T^{1/4}$, and partition the vertices of $G_T$ into two groups: the first $n$ vertices ($V_0$), and the rest ($V_1$). We suppose that many of the vertices of $V_0$ may belong to a clique. However, we argue (in the next paragraph) that edges between vertices in $V_1$ occur with

probability at most 1/2. Furthermore, these edges are largely independent of each other, therefore the graph on $V_1$ is dominated by an Erdos-Renyi graph with $p = 1/2$. This implies that the largest clique in $V_1$ has size $O(\log T)$. Together with a potential clique on all of $V_0$, the maximum clique size in $G_T$ is at most $n + \log T = O(T^{1/4})$.

Now to bound the probability that $\{i, j\}$ is chosen as an edge, we consider step $t$ (where $j \leq t \leq T$), and suppose that $i$ and $j$ have degrees near their expectation. Then the probability that we add $\{i, j\}$ on step $t$ is at most $(2(t/i)^{2/3} \cdot 2(t/i)^{2/3})/(6t)^2 \leq 1/(9t^{2/3}n^{4/3})$. Summing this for $j \leq t \leq T$, we see that the expected number of times that $\{i, j\}$ was added is at most 1/3.

Second, we consider the chromatic number in $G_T$. Set $m = T^{2/5}$. By our degree estimate, the expected degree of vertex $m$ is $2(T/m)^{2/3} = 2m$. So for some constant $c$, we expect $G_T$ has at most $cm$ vertices with degree more than $cm$. Now we can color $G_T$ with $cm$ colors: first assign distinct colors to all the vertices with degree at least $cm$, and then for each lower degree vertex (in whatever order), assign it a color that is not being used by one of its neighbors. Note that this approach is similar to the upper bound argument of the preprocessing heuristic, i.e., we expect $d^*$ to be $O(T^{2/5})$.

## 4.3 Summary and Future Work

We analyzed conference scheduling problems from the industry and, seeing there was no existing stochastic model that represented the problem accurately, we proposed our own random graph model. We based our model on insights gathered from conversations with a conference scheduler and from analysis of his datasets. Our model is also based on a sensible social model of how conference topics develop. Through experiments, we have argued that our model has a similar degree distribution as typical conference scheduling problems, and we have measured the performance of several approaches,

heuristics, and algorithms for coloring and clique finding, as they have applications in scheduling.

Analytically we have argued that the degree distribution of our model is a power law, that $E[\omega(G)]$ is $\Omega(T^{1/4}/(\log T)^{3/4})$ and $O(T^{1/4})$. We also argue that $E[\chi(G)]$ is upper bounded by $O(T^{2/5})$.

There are several directions that require future work. Firstly, we would like to extend the model to include conflict constraints other than solely speaker constraints. Ideal conference schedules also consider side constraints such as avoiding topical overlaps, or the availability of speakers and rooms. Considering these side constraint moves the problem from the realm of scheduling to a timetabling problem. Creating a timetabling algorithm that takes advantage of the structural properties of this model is another ambition we wish to pursue, as well as formulating the timetabling problem for simplex method optimizers (such as CPLEX) or for SAT solvers. On the analytical side, we would like to show stronger bounds on $\omega(G)$ and $\chi(G)$, as we suspect that they can be improved. In particular, we conjecture that both quantities are $\theta(T^{1/4})$, with only a small gap between them. Additionally, we would like to provide a careful analysis the other variants of our model.

# Appendix

## 5.1 Haws Model Graph Generator

```
"""
This module defines haws(), which generates random graphs
according to several variants of the Haws model.  The
graph is returned as an array of adjacency lists.  The
arrays and vertex ids are all 0-based. It can generate
a graph when invoked from the command line.
"""
import sys
from random import Random

def haws(N=10000,          # number of vertices, ids 0 to N-1
    D=2,                   # each clique[v] has size min(v,D+1)
    C=1.0,                 # "copy factor", from 0.0 to 1.0
    seed=None,             # controls random number generator (int)
    uniformParent=False,   # Pick prototype uniformly (not by degree)
    uniformChild=False,    # Pick children randomly (not by degree)
    verbosity=0,           # amount of verbose output for debugging
    listener=None,         # called after each vertex added
    S = 0.0                # "speaker factor", from 0.0 to 1.0
    ):
  # Create our random number generator.
  gen = Random(seed)
  # We will return the graph G, once it is populated.
  # For each vertex v, from 0 to N-1, G[v] is its adjacency list.
  # Each edge is represented twice (u in G[v] and v in G[u]).
  # Initially, we create an empty graph.
  G = [[] for u in range(N)]
  # For each vertex v, clique[v] will be the other members of the
  # clique formed when we add v.  It is a list of length min(v,D).
  clique = [[] for u in range(N)]
  # edges is a list of all graph edges: pairs (u,v) with u<v.
  # We also maintain edgeset, a set of the same edges, to be
```

```
# sure we add each edge only once to the edges list.
edges = []
edgeset = set()
# vertex is a list of which numbers between 0 and N-1 are nodes
# in the graph. If vertex[u] == -1, we don't add a vertex
# v to the squared graph, instead we just impose edges implied
# by the integers in clique[u]. Otherwise, vertex[u] == the id
# of the talk vertex that accompanies that speaker.
vertex = [-1]*N
vertex[0] = 0
# Note u==0 is easy: we simply leave clique[0]==[].
# v denotes the number of "talk" vertices
v = 1
# u denotes the number of "speaker" vertices
for u in range(1,N):
  # S chance of having a "speaker step"
  if gen.random() >= S or u <= D:
    vertex[u] = v
    v = v + 1
  # Pick a "parent" or "prototype" vertex p.
  if uniformParent or u==1:
    # pick uniformly among previous (0 to u-1)
    p = gen.randrange(0, u)
  else:
    # pick by degree (random edge endpoint)
    p = gen.choice(gen.choice(edges))
  if verbosity > 1:
    print "%d copying from prototype %d" % (u, p)
  # Build clique[vu, initially just a copy of clique[p].
  clique[u] = [w for w in clique[p]]
  # For each entry in clique[u] (inherited from clique[p]), with
  # probability C we will erase it (replace it with value None).
  # These None values are later replaced with random vertices.
  for i in range(len(clique[u])):
    if gen.random() < C:
      clique[u][i] = None
  # Also pad clique[u] to the right length, with None values.
  while len(clique[u]) < min(u, D):
    clique[u].append(None)
  # Now replace each None in clique[u] with a random vertex v,
  # distinct from the other vertices already in the clique,
  # and vertex[v] != -1 to make sure there is a talk.
  if verbosity > 2:
    print "pre clique[%d] ==" % u, clique[u]
```

```
for i in range(len(clique[u])):
  while clique[u][i] == None:
    w = 0
    # Like "parent" cases (uniform or by degree)
    if uniformChild or u==1:
      w = gen.randrange(0, u)
    else:
      w = gen.choice(gen.choice(edges))
    if w not in clique[u] and vertex[w] != -1:
      # w is good, add it (ending while)
      clique[u][i] = w
if verbosity > 1:
  print "adding clique[%d] ==" % u, clique[u]

# At this point, we could sort clique[u].  But better
# to preserve the correspondence with clique[p] entries.
# Now add all the new edges to G, implied by clique[u].
# The edges to u must be new, but the other edges between
# members of clique[u] might not be new. Note that G
# deals solely with "talk" vertex ids. We keep the edges
# and edgeset with the original ids to not affect the
# random choices
for w1 in clique[u]:
  if vertex[u] != -1:
    # Not a speaker step, add vertex u.
    wu = (w1, u)
    # assert vu not in edgeset
    edgeset.add(wu)
    edges.append(wu)
    G[vertex[w1]].append(vertex[u])
    G[vertex[u]].append(vertex[w1])
  # Impose edges between vertices in clique[u].
  for w2 in clique[u]:
    if w1 < w2:
      ww = (w1, w2)
      if ww not in edgeset:
        edgeset.add(ww)
        edges.append(ww)
        G[vertex[w1]].append(vertex[w2])
        G[vertex[w2]].append(vertex[w1])
  if verbosity > 0:
    print "after clique[%d]: G has %d edges" % (u, len(edges))
  if listener:
    listener(G=G, u=u, v=v, clique=clique[u], edges=edges,
```

```
        edgeset=edgeset, speaker=vertex[u])
  # end outer for
  # All done!  Return the graph.
  return G

# If invoked as the top-level program, generate a Haws graph
# according to command line arguments, and then output the
# graph in DIMACS format.
if  __name__  == '__main__':
  import getopt
  # These are default values for the command-line arguments.
  # All except 'outFile' correspond to arguments of haws().
  N = 10000      # -N 1000
  C = 1.0      # -C 0.0
  D = 2      # -D 4
  seed = None    # -s 17
  uniformParent = False  # -p (to toggle)
  uniformChild = False  # -c (to toggle)
  verbosity = 0    # -v (to increment)
  outFile = "-"    # filename, or "-" for stdout
  S = 0.0      # -S 1.0
  # Parse the command line:
  (opts, rest) = getopt.getopt(sys.argv[1:], "pcvN:C:D:s:S:o:")
  if len(rest)>0:
    print "unexpected arguments:", " ".join(rest)
    sys.exit(1)
  for (opt, val) in opts:
    if opt=="-p": uniformParent = not uniformParent
    if opt=="-c": uniformChild = not uniformChild
    if opt=="-v": verbosity += 1
    if opt=="-N": N=int(val)
    if opt=="-C": C=float(val)
    if opt=="-D": D=int(val)
    if opt=="-s": seed=int(val)
    if opt=="-o": outFile=val
    if opt=="-S": S=float(val)
  # Ok, run haws with all the arguments:
  G = haws(N=N, C=C, D=D, seed=seed,
     uniformParent=uniformParent,
     uniformChild=uniformChild,
     verbosity=verbosity,S=S)
  # Convert G to a DIMACS format string.
        # We prepend a comment describing the graph parameters.
        dimacs = "c haws(N=%d, C=%g, S=%g, D=%d, seed=%s, \
```

```
                uP=%s, uC=%s)\n" % \
               (N, C, S, D, str(seed), uniformParent, \
                uniformChild)
        dimacs += toDimacs(G)
  # Finally output the string appropriately.
        if outFile=="-":
                # file = sys.stdout
                print dimacs
        else:
                file = open(outFile, "w")
                file.write(dimacs)
                file.close()
                print "wrote graph to", outFile
        # all done
```

## 5.2 Usage

The graph generator outputs the graph in the ".col" format, which is the standard input used for coloring algorithms. When a line begins with "c", that means it is a comment. We use a solitary comment line to indicate which parameters the graph was generated with. Exactly one line should begin with "p edge", and it indicates the number of vertices and edges in the graph. A line starts with "e" to indicate a specific undirected edge. The graph generator will output to stdout unless asked to write to a file with the "-o" option.

Running the graph generator from the terminal:

```
$ python --version
Python 2.7.3


$ python HawsGenerator.py -N 10
c haws(N=10, C=1, S=0, D=2, seed=None, uP=False, uC=False)
p edge 10 18
e 1 2
e 1 3
```

```
e 1 4
e 1 5
e 1 6
e 1 7
e 1 9
e 2 3
e 2 5
e 2 6
e 2 7
e 3 4
e 3 8
e 3 7
e 3 10
e 7 8
e 7 9
e 8 10

$ python HawsGenerator.py -p -c -N 10 -C 0.2 -D 4 -s 4789345 -S 0.1
c haws(N=10, C=0.2, S=0.1, D=4, seed=4789345, uP=True, uC=True)
p edge 10 24
e 1 2
e 1 3
e 1 4
e 1 5
e 1 6
e 1 7
e 2 3
e 2 4
e 2 5
```

```
e 2 6
e 2 7
e 2 8
e 3 4
e 3 5
e 3 6
e 3 7
e 3 8
e 4 5
e 4 6
e 4 8
e 5 6
e 5 7
e 6 7
e 6 8

$ python HawsGenerator.py -p -c -N 10 -D 4 -s 4789345 -o hm.col
wrote graph to hm.col
```

# Bibliography

[1] Albert-lászló Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 1999.

[2] B. Bollobás. The chromatic number of random graphs. *AMS Combinatorica*, 1987.

[3] B. Bollobás and P. Erdős. Cliques in random graphs. *Math. Proc. Camb. Phil. Soc.*, 1976.

[4] Spencer J. Bollobás B., Riordan O. and Tusnády G. The degree sequence of a scale-free random graph process. *Random Structures and Algorithms*, 2001.

[5] Marco Chiarandini, Giulia Galbiati, and Stefano Gualandi. *Efficiency issues in the RLF heuristic for graph coloring*, pages 461–469. 2011.

[6] Marco Chiarandini and Thomas Stützle. An analysis of heuristics for vertex colouring. In Paola Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 326–337. Springer Berlin Heidelberg, 2010.

[7] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, pages 661–703, 2009.

[8] P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 1960.

[9] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, pages 57–, Washington, DC, USA, 2000. IEEE Computer Society.

[10] Linyuan Lu and Fan Chung. *Complex Graphs and Networks*. Cbms Regional Conference Series in Mathematics. American Mathematical Society, August 2006.

[11] Anuj Mehrotra and Michael A. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8:344–354, 1995.

[12] M. E. J. Newman. Random graphs as models of networks. *Santa Fe Institute Working Paper*, 2002.

[13] Sampo Niskanen and Patric R. J. Östergärd. Cliquer user's guide, version 1.0, 2003.

[14] Pablo San Segundo. A new dsatur-based algorithm for exact vertex coloring. *Computers and Operations Research*, 39(7):1724–1733, 2012.