Balanced Greedy Colorings of Sparse Random Graphs

Fredrik Manne* Erik Boman†

Abstract

We investigate the computation of a coloring of the vertices of a graph so that each color class is close to equal in size. For sparse random graphs we show that there is a fairly sharp transition phase on the number of available colors such that if one uses fewer colors than the transition phase it is highly unlikely that a greedy algorithm will succeed in producing a well balanced coloring, while if one uses more colors the probability of success is close to 1. A formula is given for predicting where this transition phase will occur.

When the number of available colors is less than that of the aforementioned transition phase, we show that it is possible to fairly accurately predict how balanced one can make a coloring using the First Fit algorithm.

A number of computer experiments validate the accuracy of our results.

1 Introduction

There are several settings where it might be useful to compute a coloring of the vertices of a graph so that each color class is of roughly equal size. In a number of papers dealing with parallel graph coloring it is shown that computing a balanced coloring will reduce the overall running time of the algorithm [2, 3, 5]. Another application where balanced colorings have been used is in channel assignment for various types of communication networks [1, 10].

In the current paper we investigate how *greedy* algorithms can be used to produce various types of balanced colorings on sparse random graphs. Greedy algorithms have the advantage that a coloring can be computed fast and as is often needed in the frequency assignment problem, using an online algorithm. Although random graphs are idealized compared to real world problems they still give an indication of the trends one can expect in real applications.

We first show that there exists a fairly sharp transition phase on the number of available colors such that if one uses less colors than the transition phase it is very unlikely that a greedy algorithm will succeed in producing a well balanced coloring,

*University of Bergen, Bergen, Norway Fredrik.Manne@ii.uib.no
On sabbatical leave at the Computer Science Research Institute of the Sandia National Laboratories
†Sandia National Laboratories, NM, USA egboman@sandia.gov
Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

while if one uses more the probability of success is close to 1. We also show how one can predict where this transition phase will occur.

The most common objective in graph coloring is to color the graph using as few colors as possible. This reflects the fact that the colors in some sense represent resources. Thus we next consider how to compute a balanced coloring when the number of available colors is less than that of the aforementioned transition phase.

One of the most studied algorithms for coloring random graphs is the greedy algorithm employing the *First Fit* strategy (*FF*) [7], i.e. for each vertex select the smallest available color. We consider two problems related to computing a balanced coloring using the *FF* algorithm. The first is to predict the number of colors and the color distribution when there is a limit on the number of vertices in any one color class. Secondly, we consider the problem of predicting how well one can balance the color classes when using the *FF* algorithm and there is a limit on the maximum number of colors one can use.

In both cases we show that it is possible to fairly accurately predict how the algorithm will behave and how many vertices there will be in each color class. In doing so we also develop a new formula for predicting the number of color classes used by the unrestricted *FF* algorithm on sparse random graphs. Through examples we show that this is more accurate than previously suggested formulas.

The remaining paper is organized as follows. In Section 2 we present the notation and algorithms that will be used throughout the paper. In Section 3 we investigate greedy algorithms specifically designed for producing balanced colorings, in Section 4 we consider how well one can use the First Fit strategy for producing balanced colorings, and finally in Section 5 we conclude.

2 Preliminaries

Here we review terminology and the algorithms that will be used throughout this paper.

Let $G(V, p)$ be a random unordered graph with $|V| = n$ vertices and edge probability p, $0 < p \leq 1$. That is, for every unordered pair $v_i, v_j \in V$ the probability that the edge (v_i, v_j) exists in G is p. When dealing with a sparse graph we will write $G(n, \frac{c}{n})$ where c is a (small) constant. We denote the vertex degree of vertex v_i by δ_i and the maximum vertex degree by Δ. The *open* neighborhood of a vertex v is is denoted by $N(v)$. Note that the average vertex degree of $G(n, \frac{c}{n})$ is $\frac{c(n-1)}{n} \approx c$ and the expected number of edges is $\frac{c(n-1)}{2}$.

A graph coloring of the vertices of a graph G is an assignment of numbers (colors) from a set $\{1, 2, \ldots, k\}$ to the vertices such that two adjacent vertices are assigned different colors. The largest color gives the cardinality of the coloring.

A *greedy* algorithm for the graph coloring problem proceeds by coloring the vertices of the graph according to some predetermined order, and for each vertex v choosing a color different from the already colored vertices in $N(v)$. How the actual color is chosen and in what order the vertices are processed gives rise to different greedy coloring algorithms. The outline of the general algorithm is as shown in Algorithm 1.

The most common use of this algorithm is to achieve a coloring using the fewest possible colors. In this case the color of v would typically be chosen as the lowest possible color that does not conflict with any already colored vertices in $N(v)$ (that is, the neighbors of v). This is what is known as the *First Fit* algorithm.

Algorithm 1 The Greedy Coloring Algorithm

```
1: procedure GREEDY(G = (V, E))
2:     for each v ∈ V do
3:         color(v) = k ∈ {l ∈ Z^+, ∀w ∈ N(v), color(w) ≠ l}
4:     end for
5: end procedure
```

We will also study two other greedy coloring algorithms. These are the *Least Used* (LU) and the *Random* algorithm. In the LU algorithm the current color k is chosen as a least used legal color in the interval $[1, \gamma]$ where γ is the largest color used so far. Only when there is no legal color available in $[1, \gamma]$ can color $\gamma + 1$ be chosen. The only difference between the *Random* algorithm and the LU algorithm is that instead of choosing the least used legal color one chooses a random color among the legal ones. Note that it is possible to set the initial value of γ to a value larger than 1. The algorithms would then start with the colors $[1, \gamma]$ being equally likely to be used.

There has been extensive work in trying out different kinds of orderings of the vertices. Such orderings include among others, ordering the vertices in a random manner, by decreasing vertex degree, and by the number of unique color classes a vertex is incident to. In the following we will only study random orderings. This is of interest since it is the fastest method with the least amount of overhead. Moreover, a random ordering also models the *online* coloring problem where the algorithm only knows about the part of the graph colored so far.

To generate random graphs for the experiments presented in the paper we have used the package described in [6]. This takes a number of vertices and edges as input and outputs a corresponding graph with each edge equally likely to appear. Thus to generate an instantiation of a particular random graph we have used the number of vertices and the expected number of edges as input parameters. For each experiment presented in the paper we have generated 10 random graphs. In addition each graph has been colored 10 times using different random orderings. The presented numbers are then the average over these.

3 Completely Balanced Colorings

In a *completely balanced* coloring we want each resulting color class to be of almost equal size, with only a relatively small variance between the size of each color class. Of the greedy algorithms considered in Section 2 only the LU and *Random* algorithms are expected to be able to produce such colorings.

In the following we show how to predict the minimum initial value of γ such that either the LU or the *Random* algorithm is likely to compute a completely balanced coloring using exactly γ colors. As it turns out this number is very close to the number of colors used by either algorithm if the algorithm starts with $\gamma = 1$.

Knowing the value of γ in advance is of importance in parallel graph coloring where one would like to have a large initial spectrum of color available in order to reduce the possibility that two vertices colored on different processors receive the same color [2]. For other applications being able to predict the number of colors needed might be used to predict the amount of resources the application will require. Moreover, one is also likely to obtain a more even color distribution when

using the LU or the *Random* algorithm if all the colors are available at the start of the algorithm.

We let γ_r, $0 \leq r \leq 1$, denote the least value such that both the LU and the *Random* algorithms are expected to produce a completely balanced coloring with probability at least r. As input we use the random graph $G(n, p)$.

Predicting γ_r

We now proceed to construct a formula that can be used for predicting γ_r for the LU and the *Random* algorithms. In the following we will not distinguish between these two algorithms but merely refer to a generic "algorithm" and assume that it tries to balance the distribution of the first γ color classes, where γ is a predetermined number of colors available.

If the algorithm requires more than γ colors we say that the algorithm *fails*. We will assume that if the algorithm does not fail then the resulting coloring is completely balanced. This will be shown later to be true.

Let F be the actual number of colors used by the algorithm. Then for a given value γ we first wish to bound the probability that $F \leq \gamma$.

Let f_i be the color of vertex i. It then follows that

$$P(F \leq \gamma) = \Pi_{i \in V} P(f_i \leq \gamma) \tag{1}$$

where $P(f_i \leq \gamma)$ is the probability that vertex i will receive a color $\leq \gamma$ *given* that vertices 1 through $i-1$ has also received a color $\leq \gamma$.

For increasing values of i the value of $P(f_i \leq \gamma)$ is a non-increasing function. This follows since as i increases the number of vertices in each of the γ first color classes will not decrease. Thus the probability that vertex $i+1$ has an edge to at least one vertex in each of the γ first color classes is no less than that of vertex i. We therefore have $P(f_{i+1} \leq \gamma) \leq P(f_i \leq \gamma)$.

We use this observation to bound $P(F \leq \gamma)$ as follows. Starting with vertex $2\gamma+1$ we select every γ vertex $v_{\gamma \times j+1}$, $j \geq 2$. Then $P(f_{\gamma \times j+1} \leq \gamma) \leq P(f_{\gamma \times j+1-l} \leq \gamma)$ for $l \geq 0$. Taking the product of the γ equations obtained by varying l from 1 to γ we obtain $P(f_{\gamma \times j+1} \leq \gamma)^\gamma \leq \Pi_{l=1}^{\gamma} P(f_{\gamma \times j+1-l} \leq \gamma)$. Assuming that γ divides n and adding in a dummy $n+1$st vertex we see in the upper part of Figure 1 how a particular colored vertex is compared against each of the γ previous vertices. Note that $P(f_i \leq \gamma) = 1$ for $i \leq \gamma$.

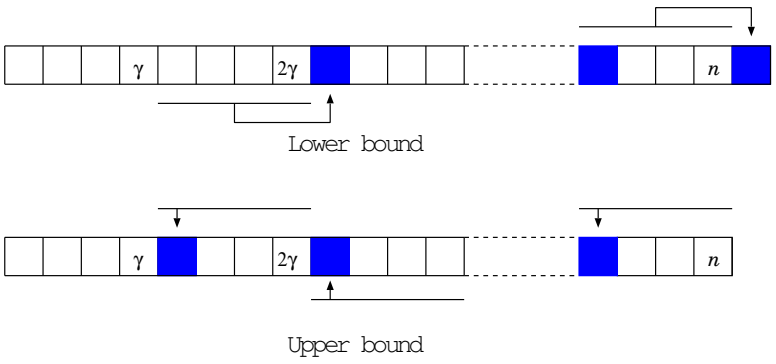

Figure 1: Vertices used for computing lower and upper bounds

Taking the product over all values of j we get the following lower bound on $P(F \leq \gamma)$

$$\Pi_{j=2}^{\frac{n}{\gamma}} P(f_{j\times\gamma+1} \leq \gamma)^\gamma \leq \Pi_{i\in V} P(f_i \leq \gamma) \tag{2}$$

In a similar fashion we can obtain an upper bound on $P(F \leq \gamma)$ by noting that $\Pi_{l=1}^{\gamma} P(f_{\gamma\times j+l} \leq \gamma) \leq P(f_{\gamma\times j+1} \leq \gamma)^\gamma$ for $l \geq 1$. This is as shown in the lower part of Figure 1. Again by taking the product over all values of j, $j \geq 1$, we get

$$\Pi_{i\in V} P(f_i \leq \gamma) \leq \Pi_{j=1}^{\frac{n}{\gamma}-1} P(f_{j\times\gamma+1} \leq \gamma)^\gamma \tag{3}$$

We now continue to expand $P(f_{j\times\gamma+1} \leq \gamma)$. To do so we look at the expected color distribution at the time when vertex $v_{j\times\gamma+1}$ is being colored. As stated initially we assume that our algorithm strives to balance the γ first color classes in each step. To see that it is reasonable to assume that this is possible note that as soon as one color class starts to get fewer vertices than the others, the probability that the next vertex can use this color will be higher than for the other color classes. Thus over time we expect each color class to receive roughly the same amount of vertices. Based on this we make the assumption that when we are coloring vertex $v_{j\times\gamma+1}$ there are j vertices in each of the γ first color classes. Thus the probability that $v_{j\times\gamma+1}$ can or cannot use one of the first γ color classes is the same for each color class. From this we can deduce

$$\begin{aligned} P(f_{j\times\gamma+1} \leq \gamma) &= 1 - P(f_{j\times\gamma+1} > \gamma) \\ &= 1 - \Pi_{l=1}^{\gamma} P(f_{j\times\gamma+1} \neq l) \\ &= 1 - P(f_{j\times\gamma+1} \neq 1)^\gamma \end{aligned} \tag{4}$$

Since we expect there to be j vertices in color class 1 it follows that $P(f_{j\times\gamma+1} \neq 1)$ is equal to the probability that vertex $v_{j\times\gamma+1}$ has an edge to at least one of the j vertices in color class 1. This again is equal to one minus the probability that the vertex has *no* edges to any vertex in color class 1. Thus we get

$$P(f_{j\times\gamma+1} \neq 1) = 1 - (1-p)^j \tag{5}$$

Setting $q = (1-p)$ in Eq. 5 and combining Eq. 1 through Eq. 5 we now get

Lemma 3.1

$$\Pi_{j=2}^{\frac{n}{\gamma}} (1-(1-q^j)^\gamma)^\gamma \leq P(F \leq \gamma) \leq \Pi_{j=1}^{\frac{n}{\gamma}-1} (1-(1-q^j)^\gamma)^\gamma \tag{6}$$

If γ does not divide n the formulas for the lower and upper bound in Eq. 6 should run to $\lceil \frac{n}{\gamma} \rceil$ and $\lceil \frac{n}{\gamma} \rceil - 1$ respectively. The outermost power of γ for the last term of both bounds should also be replaced by $n \, mod \, \gamma$. Thus the only difference between the lower and upper bound is in the first term and at most the last two terms.

Ideally, in order to determine γ_r from Eq. 6 for a given graph $G(V,p)$ and value of r, one would replace $P(F \leq \gamma)$ with r and then solve both inequalities for γ. Since we do not know how to do this we instead have to resort to some kind of search method to find the value of γ_r. This is further elaborated on in the next section.

Experiments

To test if the bounds given by Lemma 3.1 are accurate and useful for determining γ_r we have performed several experiments where we compare the predictions given by Eq. 6 with the *LU* and *Random* algorithms.

Figure 2 displays the values given by Eq. 6 for $G(1000, p)$ with $p = 0.01, 0.1$, and 0.5 as γ is varied. These values are plotted together with the fraction of the number of times that the *LU* and *Random* algorithms were successful in coloring the given graph using γ colors.

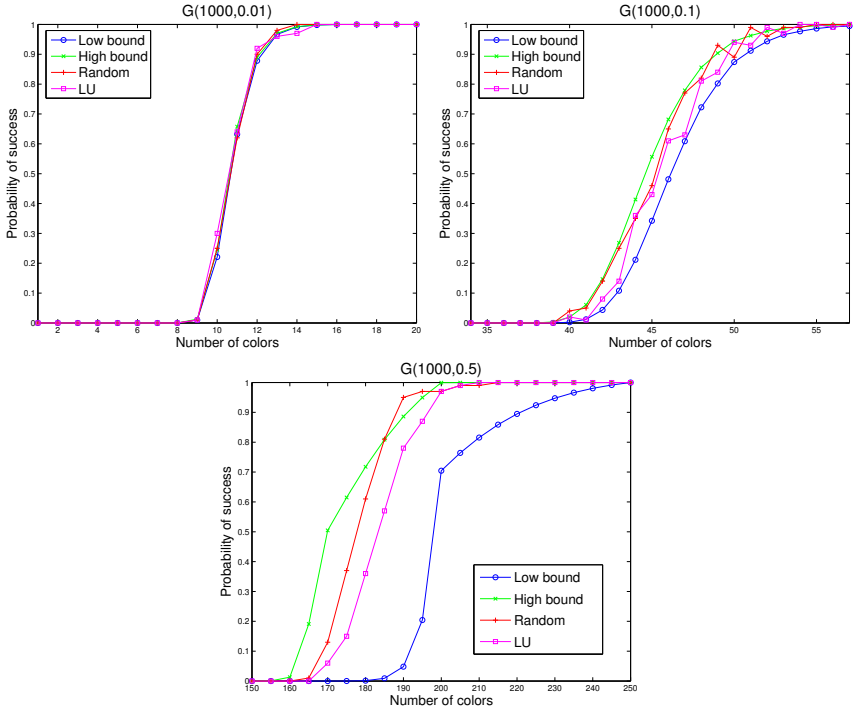

Figure 2: Probability of achieving a balanced coloring for $G(1000, p)$ for $p = 0.01, 0.1$, and 0.5

It is interesting to note that there is a fairly limited transition phase where the probability of success changes from close to 0 to close to 1, and it is only in this phase that there is some discrepancy between the predicted and the observed values. As can be seen from the figures the empirical data almost always stays within the predicted bounds and for small values of p the bounds accurately predicts the behavior of the algorithms. However, the upper and lower limits separate as p is increased. This is as expected since as p is increased the lowest value of γ for which one can achieve a balanced coloring will also increase, and thereby make the bounds coarser.

The *Random* algorithm has a slightly higher success rate than the LU algorithm, however this comes at the expense of a less smooth color distribution. We have computed the variance in the color distribution for the first value of γ when each of the algorithms succeeded in at least half of their attempts and then only using data from the successful attempts. This showed that the variance remained below 0.1 in all experiments except for the *Random* algorithm with $p = 0.01$ when the variance was 2.6.

As stated in Section 3 to determine the actual value of γ_r requires some kind of search process such as binary search. We note that this can be accelerated by

using the *secant* method as follows. For fixed values of p, r, and γ let n' be the maximum number of vertices such that there exists a completely balanced coloring of $G(n', p)$ with probability at least r. This can be calculated by applying the lower bound of Eq. 6 until just before the probability of success drops below r. Note that this might result in $n < n'$. Starting from two such points (γ_1, n_1) and (γ_2, n_2) we use the straight line between these to estimate γ_3 so that we will be able to accommodate the actual number of vertices. Again using the lower bound of Eq. 6 we then calculate the actual maximum number of vertices we expect to be able to color using γ_3 colors. This process is then repeated with the two last data points until convergence. As an example if we set $r = 0.5$ for $G(1000, 0.1)$ and start with the values 666 and 333 for γ_1 and γ_2 this will generate the following subsequent values for γ: $92, 58, 47, 47$.

We have also performed experiments using values of $\gamma < \gamma_r$. These showed that one always needs close to γ_r colors. As an example, setting $\gamma = 1$ for $G(1000, 0.01)$ and $G(1000, 0.1)$ resulted in the LU algorithm using 10.6 and 40.5 colors while the *Random* algorithm used 11.8 and 44.1 colors. As γ was increased the number of colors increased only very slowly. For all of these colorings the last couple of color classes contained significantly less vertices than the average.

4 The *First Fit* algorithm

In the following we consider using the *First Fit* (*FF*) algorithm to obtain a balanced coloring of a random sparse graph. As discussed in Section 1 this is a viable approach if the number of available colors is less than what is needed for an even color distribution as described in Section 3. In particular we consider the following two problems; (1) What is the expected number of colors that *FF* will use if no color class can contain more than l vertices and (2) What is the lowest limit one can have on each color class while still ensuring that *FF* does not use more than γ colors.

However, before we address these questions we first need to consider the expected behavior of *FF* on sparse random graphs when there are no restrictions on the coloring.

The Unrestricted *FF* Algorithm

Let $\chi_{FF(G)}$ denote the expected number of colors used by the *FF* algorithm on graph G. Grimmet and McDiarmid [4] showed that the number of colors used by the *FF* algorithm when applied to $G(n, p)$ is given by

Lemma 4.1 *Almost surely*

$$\frac{(1-\epsilon)n}{log_b n} \leq \chi_{FF(G(n,p))} \leq \frac{(1+\epsilon)n}{log_b n}$$

for any fixed ϵ*, where* $b = 1/(1-p)$.

Although this is an asymptotically tight bound, it is not a very accurate approximation for sparse graphs and Pittel and Weishaar subsequently gave the following result showing that the expected number of colors used by the greedy algorithm on $G(n, \frac{c}{n})$ is concentrated on at most two values [9].

Lemma 4.2 *Let* $j(c) = \min\{j : c_j \leq \frac{1}{2}\}$ *where* $c_1 = c$ *and* $c_j = c_{j-1} - \log(c_{j-1} + 1)$. *Further let* $f_1(c) = \log_2(\log \frac{2}{c^*} + c^*)$ *and* $f_2(c) = \log_2 \log \frac{2}{c^*} - 1$, *where* $c^* = c_{j(c)}$. *Then almost surely*

$$\log_2 \log n + j(c) - f_1(c) < \chi_{FF(G(n,\frac{c}{n}))} < \log_2 \log n + j(c) - f_2(c)$$

Finally, $\sup(f_1(c) - f_2(c)) \leq 1.5$, *so that the number of colors used is concentrated on at most two values almost surely.*

This result was based on the following earlier result also due to Pittel [8] giving the expected number of vertices in an independent set when using a greedy algorithm on $G(n, \frac{c}{n})$.

Lemma 4.3 *The expected number of vertices in an independent set when using the greedy algorithm on* $G(n, \frac{c}{n})$ *is about* $n\frac{\log(c+1)}{c}$.

Based on this result we now give a new lemma for predicting the number of vertices in each color class when using the *FF* algorithm.

Lemma 4.4 *When using the FF algorithm on* $G(n, \frac{c}{n})$ *the expected number of vertices in color class* i *is* $\frac{n}{c}\log(\alpha_i)$ *where* $\alpha_1 = c + 1$ *and* $\alpha_i = \alpha_{i-1} - \log(\alpha_{i-1})$.

Proof. Note first that for any subset $V' \subseteq V$ the graph induced by V' is $G(V', \frac{c}{n})$. This follows since the edge probability between vertices of V' is unaffected by vertices in $V - V'$. Thus if n_i is the number of vertices remaining after removing the vertices and adjacent edges of the $i-1$ first color classes, we expect that the i'th color class of $G(n, \frac{c}{n})$ will contain as many vertices as the first color class of $G(n_i, \frac{c}{n})$.

We now prove by induction on the number of color classes that the expected value of n_i is $\frac{n}{c}(\alpha_i - 1)$. For $i = 1$ we have the original graph with n vertices. Since $\frac{n}{c}(\alpha_1 - 1) = n$ this proves that the induction hypothesis holds for $i = 1$.

For the induction step assume that $n_k = \frac{n}{c}(\alpha_k - 1)$, $k \geq 1$. Then the expected number of vertices in the k'th color class is equal to the expected number of vertices in the first color class of $G(n_k, \frac{c}{n})$. By the induction hypothesis $n_k = \frac{n}{c}(\alpha_k - 1)$ and thus $n = \frac{n_k c}{\alpha_k - 1}$. Substituting this expression for n we get $G(n_k, \frac{c}{n}) = G(n_k, \frac{\alpha_k - 1}{n_k})$. Using Lemma 4.3 it follows that the expected number of vertices in color class k is $n_k \frac{\log(\alpha_k)}{\alpha_k - 1}$. Again, by applying the induction hypothesis on n_k this simplifies to $\frac{n}{c}\log(\alpha_k)$.

Thus the expected value of n_{k+1} is

$$\begin{aligned} n_{k+1} &= n_k - \frac{n}{c}\log(\alpha_k) \\ &= \frac{n}{c}(\alpha_k - 1) - \frac{n}{c}\log(\alpha_k) \\ &= \frac{n}{c}(\alpha_k - \log(\alpha_k) - 1) \\ &= \frac{n}{c}(\alpha_{k+1} - 1) \end{aligned}$$

which proves the induction hypothesis.

Since the induction hypothesis is true the above argument on the expected number of vertices in color class k is true which again proves the lemma. ∎

We can use Lemma 4.4 to generate a simple formula for $\chi_{FF(G(n,\frac{c}{n}))}$.

Corollary 4.5 *Let* $\lambda = \min\{j : \frac{n}{c}\log(\alpha_j) \geq 0.5$. *Then*

$$\lambda \leq \chi_{FF(G(n,\frac{c}{n}))} \leq \lambda + 1 \tag{7}$$

Proof. We wish to show that the cumulative number of vertices in color classes larger than λ is small. First let $f(x) = x - \log(x)$. Then from the Taylor expansion of $f(x)$ at the point $a = 1$ it follows that $f(1+t) < 1 + \frac{1}{2}t^2$. Thus setting $\alpha_{\lambda+1} = 1 + t$ we get $\alpha_{\lambda+2} = f(t+1) < 1 + \frac{1}{2}t^2$. Since $\log(1+t) \leq t$ it follows that $\log(\alpha_{\lambda+2}) < \log(1 + \frac{1}{2}t^2) < \frac{1}{2}t^2$ and in general that $\log(\alpha_{\lambda+1+i}) < \frac{t^{2^i}}{2^{(2^i-1)}}$, $i \geq 0$. Thus the cumulative number of vertices in all color classes larger than λ is bounded by $\frac{n}{c}\Sigma_{i=0}^{\infty}\frac{t^{2^i}}{2^{(2^i-1)}}$.

Now consider the first term $\frac{n}{c}t$ in this sum. This bounds the number of vertices in color class $\lambda + 1$. Since $\frac{n}{c}\log(1+t) < 0.5$ it follows that $t < e^{\frac{c}{2n}} - 1$ which shows that $\frac{n}{c}t < \frac{n}{c}(e^{\frac{c}{2n}} - 1) < e^{\frac{1}{2}} - 1 < 0.65$. It follows that $\frac{n}{c}\Sigma_{i=0}^{\infty}\frac{t^{2^i}}{2^{(2^i-1)}} < 0.65\Sigma_{i=0}^{\infty}(\frac{t}{2})^{2^i-1}$. Since $\frac{n}{c}t < 0.65$ and $\frac{n}{c} \leq 1$ we have $t < 0.65$. Using this it is easy to show that the total number of vertices in color classes larger than λ is less than 1.0. ∎

Note that in the proof of Corollary 4.5 we made the pessimistic assumption that $\frac{n}{c} = 1$. For graphs where $c \ll n$ the cumulative number of vertices in color classes larger than λ will be much smaller than what is projected in the proof. Thus if $x = \frac{n}{c}\log(\alpha_{\lambda+1})$ we expect that $\lambda.x$ will be a good estimate of the expected number of color classes used by the *FF* algorithm.

In the two topmost graphs of Figure 3 we compare the presented equations for predicting the behavior of the *FF* algorithm with empirical values from coloring graphs with 1000 vertices and with increasing edge probability. The lines marked *FF* give the empirical data, the lines marked *Cor. 1* gives the expected value derived from Corollary 4.5, *GM* gives the expected value from Lemma 4.1, while *PW low* and *PW high* show the bounds given by Lemma 4.2.

As can be seen from the graphs the expected value given by Corollary 4.5 is almost identical with the empirical data as long as the graph is sufficiently sparse, but diverges as the edge probability increases. The bounds given by Lemma 4.2 also behave asymptotically similar to the expected value from Corollary 4.5 although these bounds seem to be consistently high compared with the *FF* algorithm on sparse graphs. Moreover, this difference seems to increase as the graph becomes sparser. As expected, the value given by Lemma 4.1 does not become accurate until the graph gets sufficiently dense.

In the bottom graph of Figure 3 we show the predicted color distribution for $G(1000, 0.1)$ as given by Lemma 4.4 plotted together with that given by the *FF* algorithm.

To verify that the *FF* algorithm uses less colors than the *LU* or *Random* algorithm we note that these algorithm required more than 40 colors to color $G(1000, 0.1)$ while *FF* only needs about 31 colors.

Restricted *FF*

We now use the results from Section 4 to show how one can predict the number of colors used by the *FF* algorithm when there is an upper limit l on how many vertices any color class can contain.

What happens in this setting is that color classes starting with the smallest numbered ones will start to fill up and become unavailable for the algorithm. If k

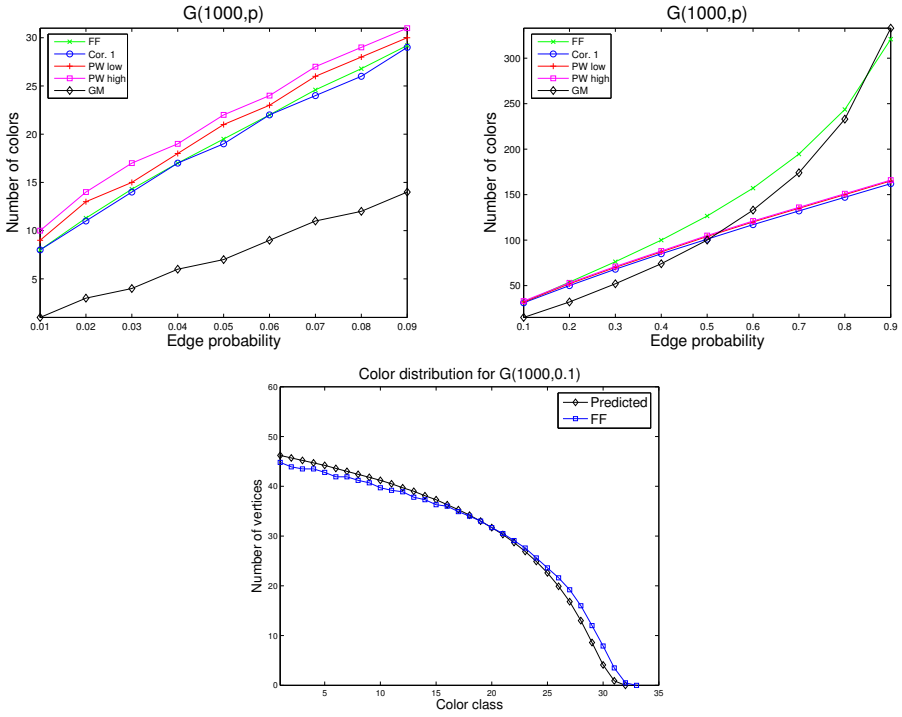

Figure 3: $G(1000, p)$: Predicting the behavior of the *FF* algorithm

classes fill up completely then the remaining graph will consist of $n_1 = n - k \times l$ vertices. Since the edge probability between these vertices is unaffected by the filled color classes they will be colored as if one was applying the regular *FF* algorithm. In the following we show how to predict k.

We now have the graph $G(n_1, \frac{c}{n}) = G(n_1, \frac{cn_1/n}{n_1}) = G(n_1, \frac{c_1}{n_1})$. The first color class of this graph will receive $\frac{n_1}{c_1}\log(c_1 + 1)$ vertices. Since this color class is the first color class smaller than l we see that k must be the smallest number such that $\frac{n_1}{c_1} log(c_1 + 1) < l$.

Substituting $c_1 = \frac{cn_1/n}{n_1}$ and $n_1 = n - k \times l$ and then solving for k we obtain $k > \frac{n}{l}(1 - \frac{1}{c}(e^{l\frac{c}{n}} - 1))$. Since k is an integer we get:

Lemma 4.6 *The restricted FF algorithm where no color class can have more than l vertices will have the first $k = \lceil \frac{n}{l}(1 - \frac{1}{c}(e^{l\frac{c}{n}} - 1))\rceil$ color classes completely filled, while the remaining $n_1 = n - k \times l$ vertices will be distributed like in the regular FF algorithm.*

Note that we allow equality in Lemma 4.6. In this case the first color class of the n_1 remaining vertices will contain exactly l vertices.

To show that Lemma 4.6 gives an accurate estimate we present Figure 4. The left graph shows the predicted number of colors used versus the actual number when using the restricted *FF* algorithm on the graph $G(1000, 0.01)$ for varying maximum size of any color class. The graph on the right side shows both the predicted and actual color distribution in detail for the case when the maximum color size is set to 150. The two curves overlap almost perfectly.

Finally we briefly discuss how to solve the problem where one is given a maximum number of colors γ and want to know what the minimum limit one can impose on the size of any color class without the *FF* algorithm using more than γ colors.

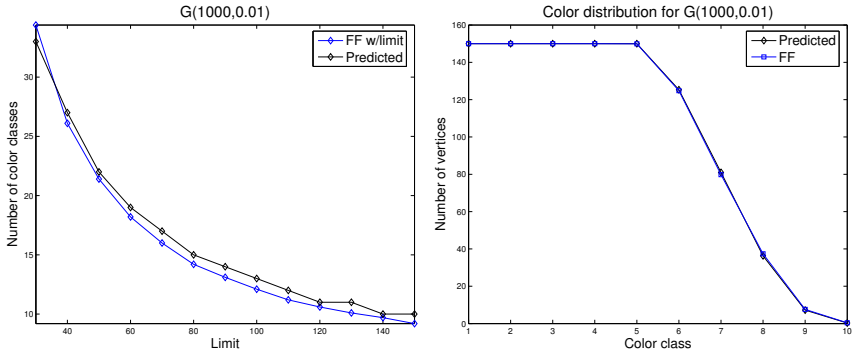

Figure 4: Number of colors used versus limit (left) and color distribution for a limit of 150 (right)

One immediate way of solving this problem is to use binary search on the limit and then applying Corollary 4.5 and Lemma 4.6 to predict the number of color classes needed.

A more analytic approach is obtained if one processes the color classes in reverse order starting with color class γ and asks for the maximum number of vertices n_γ one can have and still only use one color class. This is then continued recursively as follows. Assume that one knows the maximum number of vertices n_k one can have without the *FF* algorithm using more than $\gamma - k + 1$ color classes. From this number one can find the maximum number of vertices n_{k-1} that can be colored using at most $\gamma - k$ color classes as follows. Let $f(n_{k-1}) = \frac{n}{c}\log(c\frac{n_{k-1}}{n} + 1)$ be the number of vertices that *FF* would assign to the first color class given n_{k-1} vertices. The value of n_{k-1} is then the maximal value such that $n_{k-1} - f(n_{k-1}) = n_k$. This process is then repeated until $(n - n_k)/k \leq f(n_k)$ at which point there are $n - n_k$ vertices remaining that can be evenly distributed among the remaining $k - 1$ color classes. The exact minimum limit is found by solving $(n - n_k)/k = f(n_k)$ for n_k.

Note that this way of solving the problem requires that one repeatedly solves the non-linear equation $n_{k-1} - \frac{n}{c}\log(c\frac{n_{k-1}}{n} + 1) = n_k$ for n_{k-1}. However, this should be fairly simple using a standard iterate method such as a Newton iteration.

As one example how this can be used we note that while the regular *FF* algorithm on $G(1000, 0.01)$ will assign approximately 240 vertices to the first color class while using 8 color classes, it is possible to achieve a restricted *FF* coloring where no color class contains more than 187 vertices while still being more likely to use 8 color classes than 9.

5 Conclusion

As a continuation of this work we are currently working on showing how the presented results can be used to predict the expected behavior of different parallel graph coloring algorithms.

Finally we mention some open problems:

- Both the bound given by Pittel and Weishar and the one given in the current paper for $\chi_{FF(G(n,\frac{c}{n}))}$ relies on a recursion of the form $\alpha_{i+1} = \alpha_i - \log(\alpha_i)$. It would be interesting to know if one can predict for a given value of α_0 and constant c, what is the smallest value of i such that $\alpha_i \leq c$.

- The given formula for predicting the number of colors needed for a balanced coloring requires some computation before an estimate is obtained. Is it possible to obtain a closed formula for this?

- It would be of interest to study to what extent real world graphs also behave in a manner similar to the one described here.

References

[1] K. I. Aardal, S. P. V. Hoesel, A. M. Koster, C. Mannino, and A. Sassano, *Models and solution techniques for the frequency assignment problem*, tech. report, Konrad-Zuse-Zentrum fur Informationstechnik, Berlin, 2001.

[2] I. Finocchi, A. Panconesi, and R. Silvestri, *Experimental analysis of simple, distributed vertex coloring algorithms*, in Proc. 13th ACM-SIAM symposium on Discrete Algorithms (SODA 02), 2002.

[3] A. Gebremedhin, F. Manne, and T. Woods, *Speeding up parallel graph coloring*, in Para'04 Workshop on state-of-the art in scientific computing, Lecture Notes in Computer Science, Springer, 2004.

[4] G. Grimmett and C. McDiarmid, *On coloring random graphs*, Math. Proc. Cam. Phil, Soc., 77 (1975), pp. 313–324.

[5] Ö. Johansson, *Simple distributed $\delta + 1$-coloring of graphs*, Inf. Proc. Letters, 70 (1999), pp. 229–232.

[6] R. Johnsonbaugh and M. Kalin, *A graph generation software package*, ACM SIGCSE Bulletin, 23 (1991), pp. 151 – 154.

[7] M. Krivelevich, *Coloring random graphs - an algorithmic perspective*, in Proc. MathInfo'2002, the 2nd Colloquium on Mathematics and Computer Science, Birkhauser, 2002, pp. 175–195.

[8] B. Pittel, *On the probable behavior of some algorithms for finding the stability number of a graph*, Math. Proc. Cambridge Philos. Soc., 92 (1982), pp. 511–526.

[9] B. Pittel and R. S. Weishaar, *On-line coloring of sparse random graphs and random trees*, J. Alg., 23 (1997), pp. 195–205.

[10] H. Zang, J. P. Jue, and B. Mukherjee, *A review of routing and wavelength assignment approaches for wavelength-routed optical wdm networks*, SPIE Optical Networks Magazine, 1 (2000), pp. 47–60.