On The Probable Performance of Graph Coloring Algorithms

Jonathan S. Turner
Computer Science Department, Campus Box 1045
Washington University, St. Louis, MO 63130
jst@wucs.UUCP

ABSTRACT

We define a natural probability distribution over the set of k-colorable graphs on n vertices and study the probable performance of several algorithms on graphs selected from this distribution. The main results are listed below.

1. We describe an algorithm to determine if a given n vertex graph is k-colorable, which runs in time $O(n + m \log k)$, where m is the number of edges. We show that this algorithm can successfully identify almost all random k-colorable graphs for constant or slowly growing values of k.
2. We show that an algorithm proposed by Brélaz, and justified on experimental grounds can successfully k-color almost all random k-colorable graphs for constant or slowly growing values of k. We also describe an efficient implementation, which runs in time $O(m \log n)$.
3. We show that the performance of the well-known greedy algorithm for graph coloring is relatively poor for random k-colorable graphs.

In addition to the analytical results, we present experimental data which provide more detailed information on the performance of these algorithms.

1. Introduction

Let $G = (V, E)$ be a simple undirected graph. A k-coloring of G is a mapping $c : V \rightarrow \{1, 2, \ldots, k\}$; c is a *proper coloring* if $c(u) \neq c(v)$ for all $\{u, v\} \in E$. The *chromatic number* of G, denoted $\chi(G)$, is defined as the smallest positive integer k for which a proper k-coloring exists. The *graph coloring problem* is to determine for a given graph G and an integer k, if $\chi(G) \leq k$.

The graph coloring problem has a long, interesting history and arises in a variety of applications. Karp [7] showed that the problem is NP-complete. Stockmeyer [9,3] strengthened this by showing that it remains NP-complete for any fixed $k \geq 3$. This has led many researchers to seek approximation algorithms capable of producing colorings that don't use too many extra colors. Garey and Johnson [4] proved that unless P = NP, no polynomial time approximation algorithm can guarantee the use of less than $2\chi(G)$ colors. Furthermore, Johnson [6] showed that for many popular heuristics, there are 3-colorable graphs on n vertices for which the heuristics require $\Theta(n)$ colors. Johnson also described a new algorithm using at most $O(n/\log n)$ colors on any 3-colorable graph. This stood as the best worst-case result for graph coloring until recently when Wigderson [11] discovered an algorithm that colors any 3-colorable graph using at most $3\lceil\sqrt{n}\rceil$ colors and any k-colorable graph using at most $2k\left\lceil n^{1-1/(1-k)}\right\rceil$ colors.

The disappointing nature of the worst-case results for graph coloring suggests that probabilistic analysis may provide a more effective way of evaluating candidate algorithms. Grimmet and McDiarmid [5] took the first step in this direction by showing that for almost all graphs on n vertices, $\chi(G) \geq (1-\epsilon)n/(2\log_{1/(1-p)} n)$, where p is a fixed edge probability in the usual random graph model, [1] and ϵ is any positive constant. The also showed that a well-known greedy heuristic uses $\leq (1+\epsilon)n/\log_{1/(1-p)} n$ colors.

[1] In the usual random graph model, edges are generated independently with probability p between each pair of vertices. We say that a property holds for *almost all* random graphs if the probability of the property holding approaches one as $n \rightarrow \infty$.

Grimmet and McDiarmid's results are interesting for what they tell us about random graphs; it's less clear what they tell us about the merits of the greedy heuristic. The naive conclusion one can draw is that the greedy algorithm is a good one for graph coloring. A less obvious, but perhaps more accurate interpretation is that these results cast doubt on the usefulness of a probabilistic analysis based on the usual random graph model for comparing graph coloring algorithms. They suggest that the usual model is too 'easy' a distribution, since it makes even the most simple-minded algorithm look good. In order to obtain meaningful comparative information, we should try to select a more difficult probability distribution, one that poses some challenges for candidate algorithms to overcome. The probability distribution described in the next paragraph and used throughout this paper was motivated in part by these considerations. Another factor motivating this choice is touched on briefly at the end of section 2.

Let n, k be positive integers, $0 < p < 1$ and let $G = (V, E)$ be the graph defined by the following experiment.

- Let $V = \{1, \ldots, n\}$.
- For each $u \in V$ let $c(u)$ be a random integer in $[1, k]$.
- For each pair $u, v \in V$ such that $c(u) \neq c(v)$, include the edge $\{u, v\}$ in E with probability p.

The probability distribution defined by this experiment is denoted $X_n(k, p)$ and the notation $G \in X_n(k, p)$ means that G is a random graph generated in this way.

In section 2, we present a coloring algorithm, which for constant p and k growing slowly with n, finds a k-coloring for almost all graphs $G \in X_n(k, p)$, and we describe an efficient implementation using a novel data structure. In section 3, we discuss an algorithm due to Brélaz [2] and show that under the same conditions it has similar performance. We also describe an implementation of Brélaz's algorithm with running time $O(m \log n)$ for graphs on n vertices and m edges. (Brélaz claimed only an $O(n^2)$ running time.) In section 4, we study the performance of the greedy algorithm for graphs in $X_n(k, p)$; this study reveals performance that is surprisingly poor in light of [8].

2. Recognizing k-Colorable Graphs

Define a *partial coloring* of a graph $G = (V, E)$ to be a mapping $c : V \to [0, n]$. The algorithms we will study start by constructing the partial coloring defined by $c(x) = 0$ for all $x \in V$ and then attempt to convert this to a complete proper coloring. Given a partial coloring c, we can define for each vertex x, a set $avail_c(x) = \{i \mid 1 \leq i \leq n \wedge (\{x, y\} \in E \Rightarrow c(y) \neq i)\}$. If x is currently uncolored ($c(x) = 0$), $avail_c(x)$ is the set of colors that are available for coloring x. We will write $avail(x)$ without the subscript whenever the coloring function is clear from the context.

Our first algorithm attempts to find a k-coloring of a graph $G = (V, E)$, where k is assumed to be an input parameter. We will show that it succeeds with high probability for random k-colorable graphs. The algorithm starts by finding a k-clique, coloring each of its vertices with a distinct color in $[1, k]$, and then repeatedly applies the following rule.

> *Coloring Rule 1.* Select an uncolored vertex x for which $|avail(x) \cap [1, k]| = 1$ and let $c(x) = \min avail(x)$.

We refer to this as the *no choice algorithm* since it succeeds only if it can color all the vertices without making any arbitrary choices. The algorithm can fail to produce a k-coloring if it is unable to find a k-clique or if at some point $|avail(x) \cap [1, k]| \neq 1$ for all uncolored vertices x. We will show that when k is not too large, the no choice algorithm succeeds with high probability. First however, we give a more detailed description of the algorithm.

A program implementing the no choice algorithm is shown in Figure 1. (The algorithmic notation is adapted from Tarjan [10].) Vertices are represented by integers in $[1, n]$ and the graph is represented by an array of vertex sets called *neighbors*. For each vertex x, $neighbors(x)$ is a list containing all vertices adjacent to x in increasing order. Vertices that are ready to be colored are placed in a queue. Each iteration of the algorithm's main loop removes a vertex from the queue, colors it, then examines its neighbors, adding them to the queue if possible. Initially each vertex is assigned a color of -1. When a vertex is added to the queue, its color is changed to 0. The subroutine shown in Figure 2 is used to find

```
function bit nochoice(integer k, n, graph neighbors, modifies array c);
        integer i, nc; vertex x, y; list Q; set X;
        array[1..n] of set avail;
        for x ∈ [1..n] → c(x) ← −1; avail(x) ← {1,...,k}; rof;
        X ← clique(k, n, neighbors);
        if X = → return false fi;
        i ← 1;
        for x ∈ X → c(x) ← i; i ← i + 1; rof;
        Q ← [];
        for x ∈ X →
                for y ∈ neighbors(x) →
                        avail(y) ← avail(y) − c(x);
                        if c(y) = −1 and |avail(y)| = 1 →
                                Q ← Q & [y];
                                c(y) ← 0;
                        fi;
                rof;
        rof;
        nc ← k;
        do Q ≠ [] →
                x ← Q(1); Q ← Q[2..];
                if |avail(x)| ≠ 1 → return false fi;
                let i ∈ avail(x); c(x) ← i; nc ← nc + 1;
                for y ∈ neighbors(x) →
                        avail(y) ← avail(y) − c(x);
                        if c(y) = −1 and |avail(y)| = 1 →
                                Q ← Q & [y];
                                c(y) ← 0;
                        fi;
                rof;
        od;
        return nc = n;
end;
```

Figure 1: Program Implementing the No Choice Algorithm

a clique. The *clique* program runs in linear time, if the set S is implemented as a sorted list of vertices. The key to efficient implementation of the main program is the data structure used to represent the sets $avail(x)$. The simplest approach is to use a bit vector for each set. This leads to an $O(kn+m)$ running time for a graph with n vertices and m edges. We can improve on this by using a special variety of binary search tree described below. (Note that a standard search tree won't help here since initializing n search trees to represent the set $\{1,\ldots,k\}$ takes $\Omega(kn \log k)$ time.)

We define a *shrinking set* to be an abstract data type representing a set of positive integers on which the following operations can be performed.

$makeset(lo, hi)$	Return a new set consisting of the integers in the interval $[lo, hi]$.
$select(s)$	Return an arbitrary element from s.
$selectmin(s)$	Return the smallest element in s.
$delete(x, s)$	Delete the integer x from s.

The operations on shrinking sets are defined in terms of another abstract data structure, which we call an *interval set*. An interval set represents a set of disjoint intervals on the positive integers on which the following operations are defined.

$makeintervalset(i)$	Return a new set consisting of the interval i.

```
set function clique(integer k, n; graph neighbors);
        set S, K;
        S ← {1, ..., n};
        K ← ∅;
        do S ≠ ∅ →
                let x ∈ S;
                K ← K ∪ {x};
                S ← S ∩ neighbors(x);
                if |K| = k → return K fi
        od;
        return ;
end;
```

Figure 2: Subroutine for Finding a Clique

```
i ← member(x, s);
if i ≠ [] →
        deleteinterval(i, s);
        if i.lo < x → insertinterval([i.lo, x − 1], s) fi;
        if i.hi > x → insertinterval([x + 1, i.hi], s) fi;
fi;
```

Figure 3: Program Fragment Implementing the Delete Operation

member(x, s)	Return the interval in s that contains the integer x. If there is no such interval, return [].
select(s)	Return an arbitrary integer contained in some interval in s.
selectmin(s)	Return the smallest integer contained in some interval in s.
insertinterval(i, s)	Insert the interval i in s (i is assumed to be disjoint from intervals already in s).
deleteinterval(x, s)	Delete the interval i from s.

An interval set can be implemented efficiently using any standard balanced search tree structure. Each node of the search tree represents an interval. This yields an $O(\log n)$ running time per operation, where n is the number of intervals in the set. The operation *makeset*(lo, hi) on a shrinking set is implemented simply as *makeintervalset*$([lo, hi])$ on the underlying interval set. The *select* and *selectmin* operations on a shrinking set are implemented as the corresponding interval set operations. Finally, the operation *delete*(x, s) on a shrinking set is implemented by the program fragment in Figure 3. Thus, all the operations on a shrinking set can be implemented to run in $O(\log k)$ time, where k is the size of the set when it is initialized. These observations yield the following theorem.

Theorem 2.1: The no choice algorithm can test a graph for k-colorability in $O(n + m \log k)$ time, where n is the number of vertices and m the number of edges.

We now address the question of effectiveness for the no choice algorithm. First, some definitions.

Define $\lambda_n(c) = -\frac{\ln n}{\ln c}$. Note that $\lambda_n(c) > 0$ when $0 < c < 1$ and $n > 1$, $c^{\lambda_n(c)} = \frac{1}{n}$ and $\lim_{n\to\infty} \lambda_n(c) = \infty$ for fixed $c \in (0, 1)$. We will usually write $\lambda(c)$ instead of $\lambda_n(c)$.

In the remainder of this section $G = (V, E) \in X_n(k, p)$, where $0 < \epsilon < 1$, $0 < p < 1$ are fixed, $n > 0$, $0 < k \leq (1 - \epsilon)\lambda(p)$. We also let $V_1, \ldots, V_k$ be the partition of V used in the generation of G and $c(v) = i$ for each $v \in V_i$. Also, $n_i = |V_i|$ for $1 \leq i \leq k$ and $m = \min_{1\leq i\leq k} n_i$.

We can now state the main theorem.

Theorem 2.2: For almost all G, the no choice algorithm produces a complete k-coloring.

Since the no choice algorithm makes no arbitrary decisions with the exception of coloring the initial clique, the coloring it produces is unique.

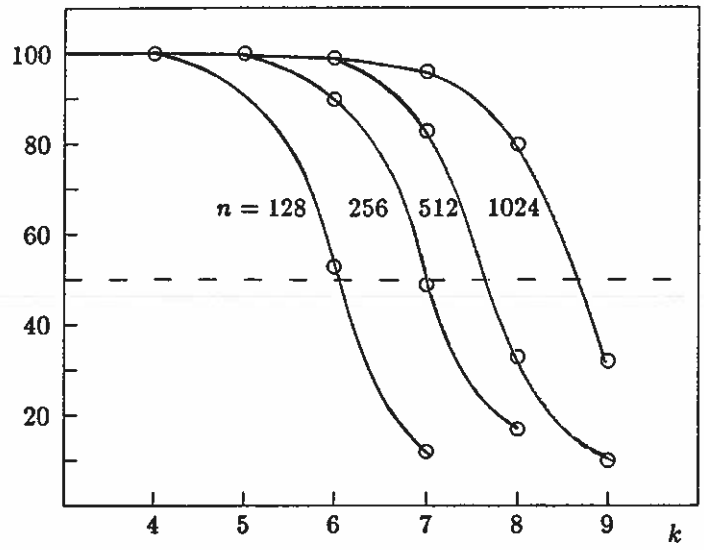

Figure 4: Success Rate of No Choice Algorithm for Graphs in $X_n(k, .5)$

Corollary 2.1: Almost all G are uniquely k-colorable.

Theorem 2.2 is proved in three steps. First, we define a class of graphs called *normal* graphs. Then, we show that the no choice algorithm succeeds for all normal graphs. Finally, we show that almost all G are normal.

We call G *normal* if it satisfies the following five properties and $n \geq \max\{k^2\lambda(1-p)/(1-\epsilon), (2k^2\lambda(1-p)/(1-\epsilon))^{1/\epsilon}\}$.

minimum size property — $m \geq n/2k$.

clique property — the clique finding algorithm of Figure 2 succeeds on G.

star property — for every set $\{x_1, \ldots, x_k\}$ with $x_j \in V_j$, each set V_i contains at least $(1-\epsilon)n^\epsilon/k$ vertices with $k-1$ neighbors in $\{x_1, \ldots, x_k\}$.

subset property — there is an edge joining every pair U_i, U_j $(i \neq j)$, where U_i is any subset of V_i with at least $k\lambda(1-p)$ vertices.

degree property — every vertex x has at least $(1-\epsilon)mp$ neighbors in each set V_i except $V_{c(x)}$.

It is convenient to think of the no choice algorithm as proceeding by a series of discrete steps. We now describe a version of it in which these steps are made explicit.

1. Find a k-clique $x_1, \ldots, x_k$ and let $c(x_i) = i$ for $1 \leq i \leq k$.
2. For $1 \leq i \leq k$, find a set U_i of at least $n^\epsilon/(1-\epsilon)k$ uncolored vertices adjacent to each vertex in $\{x_1, \ldots, x_k\} - \{x_i\}$. Let $c(y) = i$ for all $y \in U_i$.
3. For $1 \leq i \leq k$, find a set W_i of at least $n_i - k\lambda(1-p)$ vertices, such that every vertex $y \in W_i$ is adjacent to some vertex in each U_j for which $j \neq i$. Let $c(y) = i$ for all $y \in W_i$.
4. For each uncolored vertex y, such that $|avail(y) \cap [1,k]| = 1$ let $c(y) = \min avail(x)$.

It's easy to see that if the above algorithm succeeds in finding a k-coloring then the original no choice algorithm does also. While less obvious, it is straightforward to verify that if G is normal, then the algorithm succeeds. The lemmas that follow show that almost all G are normal, which in turn yields the theorem.

The following proposition (Angluin and Valiant [1]) is used in the proofs of several of the lemmas. Let $B(n,p)$ denote the binomial distribution. By definition, if $x \in B(n,p)$ then $P(x=k) = \binom{n}{k}p^k(1-p)^{n-k}$.

Proposition 2.1: If $x \in B(n,p)$ then for any α, $0 < \alpha < 1, P(x \leq (1-\alpha)np) < e^{-\alpha^2 np/2}$ and $P(x \geq (1+\alpha)np) < e^{-\alpha^2 np/3}$.

Lemma 2.1: Almost all G satisfy the minimum size property.

proof. Each n_i is a random variable drawn from $B(n, 1/k)$. By Proposition 2.1, the probabilty that a particular n_i is less than $n/2k$ is $< e^{-n/8k}$ and the probability that any of the n_i is less than $n/2k$ is $< ke^{-n/8k} \to 0$, since $k \leq (1-\epsilon)\lambda(p)$. □

Lemma 2.2: Almost all G satisfy the clique property.

proof. Lemma 2.1, allows us to assume that $m \geq n/2k$. Now consider the operation of the clique finding algorithm. If $0 \leq r \leq k-1$ and K_r is the value of K at the start of iteration r of the loop, the probability that there is no vertex y adjacent to all the vertices in K_r is $\leq (1-p^r)^{m(k-r)} \leq (1-p^r)^{n/k}$. There are at most $\binom{k}{r}n^r$ ways to select K_r, so the probability that *clique* halts without finding a k-clique is

$$\leq \sum_{r=0}^{k-1} \binom{k}{r} n^r (1-p^r)^{n/k} \leq \sum_{r=1}^{k-1} \left[kne^{-p^r n/kr}\right]^r \leq k^2 ne^{-p^k n/k^2} \leq k^2 ne^{-n^\epsilon/k^2} \to 0 \qquad \square$$

Lemma 2.3: Almost all G satisfy the star property.

proof. Lemma 2.1, allows us to assume that $m \geq n/2k$. Let $U = \{x_1, \ldots, x_k\}$ be any vertex set with $x_i \in V_j$ and let s_i be the number of vertices adjacent to every vertex in $U - \{x_i\}$. Clearly, $s_i \in B(n_i, p^{k-1})$. Since $n^\epsilon/k \leq n_i p^{k-1}$, $P(s_i \leq (1-\epsilon)n^\epsilon/k) \leq P(s_i \leq (1-\epsilon)n_i p^{k-1})$, which by Proposition 2.1 is,

$$< e^{-\epsilon^2 n_i p^{k-1}/2} \leq e^{-\epsilon^2 np^k/2k} \leq e^{-\epsilon^2 n^\epsilon/2k}$$

Since there are at most n^k different choices for U, the probability that G does not satisfy the star property is

$$< kn^k e^{-\epsilon^2 n^\epsilon/2k} = kn^{k-(\epsilon^2/2)(n^\epsilon/k \ln n)} \to 0 \qquad \square$$

Lemma 2.4: Almost all G satisfy the subset property.

proof. Lemma 2.1, allows us to assume that $m \geq n/2k$. For $1 \leq i \leq k$, let $U_i \subseteq V_i$ with $|U_i| = r \geq k\lambda(1-p)$. The probability that there is no edge joining a particular pair U_i, U_j is $(1-p)^{r^2}$. Since there are at most $\binom{n}{r}^k$ ways to choose the U_i, the probability that G does not satisfy the subset property is

$$\leq \binom{k}{2}\binom{n}{r}^k (1-p)^{r^2} \leq k^2 \left(\frac{en}{r}\right)^{kr} (1-p)^{r^2} \leq k^2 \left(\frac{en}{r}(1-p)^{r/k}\right)^{kr}$$

$$\leq k^2 \left(\frac{en}{k\lambda(1-p)}(1-p)^{\lambda(1-p)}\right)^{kr} \leq k^2 \left(\frac{e}{k\lambda(1-p)}\right)^{k^2\lambda(1-p)} \to 0 \qquad \square$$

Lemma 2.5: Almost all G satisfy the degree property.

proof. Lemma 2.1, allows us to assume that $m \geq n/2k$. Let $d_i(x)$ be the number of neighbors vertex x has in V_i. Clearly, $d_i(x) \in B(n_i, p)$ for $x \notin V_i$. By Proposition 2.1,

$$P(d_i(x) \leq (1-\epsilon)mp) \leq P(d_i(x) \leq (1-\epsilon)n_i p) < e^{-\epsilon^2 n_i p/2} \leq e^{-\epsilon^2 np/2k}$$

So the probability that G does not satisfy the degree property is $\leq kne^{-\epsilon^2 np/2k} \to 0$. □

This completes the proof of Theorem 2.2.

A series of experiments were run to provide more detailed information on the performance of the no choice algorithm. One hundred random graphs in $X_n(k, .5)$ were generated for each of several values of n and k. The no choice algorithm was then run on each graph. [2] The results are summarized in Figure 4. For each value of n and k the figure shows the number of graphs for which a k-coloring was constructed. The figure shows that the algorithm works well when k is small, but as k gets larger, its performance deteriorates abruptly. This is consistent with the analysis given above. As n increases, the breakdown point also increases. Let $\beta_n(p)$ be the smallest k for which the probability of success on graphs in $X_n(k,p)$ is less than 1/2. We can estimate $\beta_n(p)$ by observing where the curves in Figure 4

[2] The experiments actually involved a slightly different version of the no choice algorithm. This version uses a greedy heuristic to find the initial clique. When selecting the next vertex for inclusion in the clique, this heuristic always selects a vertex of maximum degree from the set of eligible vertices.

```
procedure brelaz(integer k, n, graph neighbors, modifies array c);
        vertex x, y; heap h;
        array[1..n] of set avail;
        array[1..n] of integer deg;
        for x ∈ [1..n] →
                c(x) ← 0;
                avail(x) ← {1, ..., n};
                deg(x) ← |neighbors(x)|;
        rof;
        h ← makeheap({1, ..., n});
        do h ≠ →
                x ← deletemin(h);
                c(x) ← min avail(i);
                for y ∈ neighbors(x) →
                        if c(y) = 0 →
                                avail(y) ← avail(y) − c(x);
                                deg(y) ← deg(y) − 1;
                                siftup(y, h);
                        fi;
                rof;
        od;
end;
```

Figure 5: Program Implementing Brélaz's Algorithm

cross the dashed line. The data suggest that $\beta_{128}(.5) = 6$, $\beta_{256}(.5) = 7$, $\beta_{512}(.5) = 8$, and $\beta_{1024}(.5) = 9$. This is consistent with Theorem 2.2, which suggests that $\beta_n(p)$ grows in proportion to $\lambda_n(p)$.

Let k be any integer greater than 2 and let S_k be the set of all graphs with chromatic number no larger than k. The results of this section suggest that most graphs in S_k are easily identified, leading us to the following conjecture.

Conjecture 2.1: For any fixed $k > 0$, the no choice algorithm successfully colors almost all graphs in S_k.

Note that this is not implied by Theorem 2.2, since the probability distribution $X_n(k,p)$ does not assign equal probability to every k-colorable graph.

3. Brélaz's Algorithm

The no choice algorithm is similar to one proposed by Brélaz [2] and justified on experimental grounds. Brélaz's algorithm can be described as a repeated application of the following rule.

> *Coloring Rule 2.* Select an uncolored vertex x that minimizes $|avail(x)|$ and let $c(x) = \min avail(x)$. If there are several vertices available for selection, select one with maximum degree in the uncolored subgraph.

Consider the behavior of Brélaz's algorithm on a normal graph $G \in X_n(k,p)$. The first vertex colored is one of maximum degree, the second is a neighbor of the first which has maximum degree in the uncolored subgraph, the third is a neighbor of the first two which has maximum degree in the uncolored subgraph, and so forth until k vertices have been colored. In other words, during the coloring of the first k vertices, this algorithm mimics the greedy heuristic for clique finding mentioned in the footnote at the end of the previous section. Once the first k vertices have been colored, the algorithm repeatedly selects vertices for which $|avail(x)| = n - k + 1$. That is, it mimics the no choice algorithm. These observations yield the following theorem.

Theorem 3.1: Let $0 < \epsilon < 1$, $0 < p < 1$ be fixed, $k \leq (1-\epsilon)\lambda(p)$. For almost all $G \in X_n(k,p)$, Brélaz's algorithm produces a k-coloring.

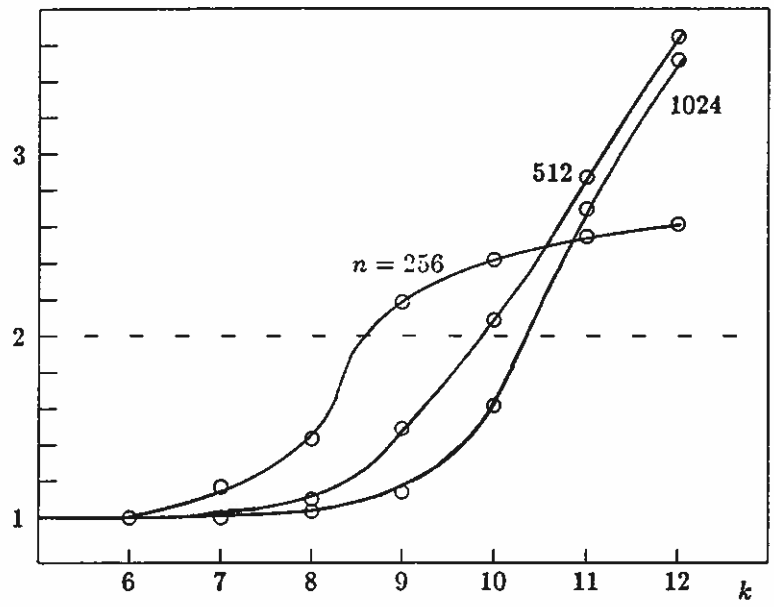

Figure 6: Average Performance Ratio of Brélaz's Algorithm for Graphs in $X_n(k, .5)$

In [2] Brélaz claims an $O(n^2)$ time bound for his algorithm, which is easily proved. In fact, Brélaz's algorithm can be implemented to run in time $O(m \log n)$ for a graph with n vertices and m edges. The program in Figure 5 illustrates this. The heap contains the uncolored vertices. For the purposes of the heap operations, vertex x is smaller than vertex y if $|avail(x)| < |avail(y)|$ or $|avail(x)| = |avail(y)|$ and $deg(x) > deg(y)$. As in the program for the no choice algorithm, the key to an efficient implementation is the data structure used to implement the sets $avail(x)$. If a bit vector is used, the running time is $O(n^2)$. However, using the shrinking set data structure described earlier, each initialization operation can be done constant time, the selection of a minimum can be done in $O(\log n)$ time as can the deletion operation. These observations yield,

Theorem 3.2: Brélaz's algorithm runs in time $O(m \log n)$ on graphs with n vertices and m edges.

Figure 6 shows the results of a series of experiments, which provide more detailed information on the performance of Brélaz's algorithm. One hundred random graphs in $X_n(k, .5)$ were generated for each of several values of n and k, and Brélaz's algorithm was run on each graph. The plot shows the ratio of the average number of colors used to k. As with the no choice algorithm, the performance is quite good for small k, but deteriorates abruptly as k gets large. The point at which the breakdown occurs appears to increase logarithmically with n as one would expect from Theorem 3.2.

4. The Greedy Algorithm

The greedy algorithm for graph coloring is a simple and popular heuristic. It can be described as follows.

For each $x \in [1, n]$, let $c(x) = \min avail(x)$.

Marchetti-Spaccamela and Talamo have shown that for almost all random graphs (in the usual model), the greedy algorithm uses no more than about twice the optimal number of colors. In this section, we study the performance of the greedy algorithm for graphs in $X_n(k, p)$ and conclude that it performs poorly unless k is quite small.

Let $G = (V, E) \in X_n(k, p)$. Let c be the coloring used to generate G and let c' be the coloring computed by the greedy algorithm. We are interested in the probability that c' is a k-coloring. Since almost all G are uniquely k-colorable, this probability is approximately $k!$ times the probability that $c' = c$, for large enough n.

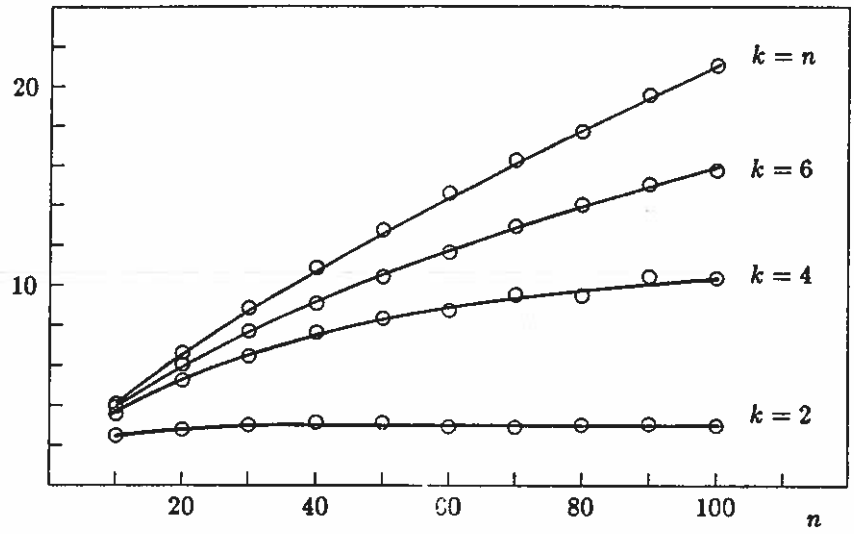

Figure 7: Average Number of Colors Used by Greedy Algorithm for Graphs in $X_n(k, .5)$

Let $S_i(r) = \{1 \le z \le r \mid c(z) = c'(z) = i\}$ for $1 \le i \le k$ and let $P(n_1, n_2, \ldots, n_k)$ be the probability that $|S_i(r)| = n_i$ for all $i \in [1, k]$, where $r = \sum_{i=1}^{k} n_i$. P satisfies the following recurrence.

$$
\begin{aligned}
P(0, \ldots, 0) &= 1 \\
P(n_1, \ldots, n_k) &= 0 \qquad \text{if any } n_i < 0 \\
P(n_1, \ldots, n_k) &= \frac{1}{k} \sum_{h=1}^{k} P(n_1, \ldots, n_{h-1}, n_h - 1, n_{h+1}, \ldots, n_k) \prod_{j=1}^{h-1} \alpha(n_j) \qquad \text{otherwise}
\end{aligned}
$$

where $\alpha(x) = 1 - (1-p)^x$. (We adopt the convention that an empty product is equal to 1.) Now, let $Q(r)$ be the probability that $c(z) = c'(z)$ for $1 \le z < r$ and $c(r) \ne c'(r)$.

$$
Q(r+1) = \sum_{\substack{n_1, \ldots, n_k \ge 0 \\ n_1 + \cdots + n_k = r}} P(n_1, \ldots, n_k) \left[1 - \frac{1}{k} \sum_{h=1}^{k} \prod_{j=1}^{h-1} \alpha(n_j) \right]
$$

Now, the probability that $c' \ne c$ is $\sum_{r=1}^{n} Q(r)$. This yields the following theorem.

Theorem 4.1: Let $0 < p < 1$, $k \ge 1$ be fixed and let $G \in X_n(k, p)$. As $n \to \infty$, the probability that the greedy algorithm produces a k-coloring of G approaches $k! \left(1 - \sum_{r=1}^{n} Q(r)\right)$.

The terms in $\sum_{r=1}^{n} Q(r)$ decline rapidly, so for small k, we can use Theorem 4.1 to estimate the probability that the greedy algorithm produces a k-coloring. We illustrate the procedure for the case, $k = 2$. The general equations reduce to

$$
\begin{aligned}
P(n_1, n_2) &= \frac{1}{2} \left[P(n_1 - 1, n_2) + P(n_1, n_2 - 1) \left(1 - (1-p)^{n_1}\right) \right] \\
Q(r+1) &= \frac{1}{2} \sum_{n_1 = 0}^{r} P(n_1, r - n_1)(1-p)^{n_1}
\end{aligned}
$$

Using these equations and Theorem 4.1 we estimate that for large n, the probability of the greedy algorithm successfully 2-coloring a graph in $X_n(2, .5)$ is approximately .42. In the same way, we estimate that the probability of the greedy algorithm successfully 3-coloring a graph in $X_n(3, .5)$ is approximately .091, and the probability of it successfully 4-coloring a graph in $X_n(4, .5)$ is approximately .044. We

conclude that unless k is quite small, we cannot expect the greedy algorithm to find optimal colorings for random k-colorable graphs.

Of course, the above results don't rule out the possibility of the greedy algorithm producing good but sub-optimal colorings. Experimental methods were used to address this issue. One hundred random graphs in $X_n(k, .5)$ were generated for each of several values of n and k. Figure 7 shows the average number of colors used by the greedy algorithm in these experiments. For any given k, the number of colors used increases with n. The rate of growth is moderate when k is small, but fairly large for $k = 6$. For $k = 6$ and $n = 100$, the greedy algorithm uses almost three times the optimal number of colors. The data indicate that except for very small k, the greedy algorithm can be expected to produce colorings that differ from optimal by an arbitrarily large factor.

For the usual random graph model, the chromatic number grows in proportion to $(n/\log n)$. It's natural to guess a similar growth rate for graphs in $X_n(k, p)$ with the constant of proportionality determined by p and k. However, our data do not support such a conjecture.

5. References

1 Angluin, D., L. G. Valiant. "Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings". In *Journal of Computer and System Sciences 18* , 155-193, 1979.

2 Brelaz, Daniel. "New Methods to Color the Vertices of a Graph". In *Communications of the ACM 22*, 251-256, 4/79.

3 Garey, Michael R., David S. Johnson, L. J. Stockmeyer. "Some Simplified NP-complete Graph Problems". In *Theoretical Computer Science 1*, 237-267, 1976.

4 Garey, Michael R., David S. Johnson. "The Complexity of Near-Optimal Graph Coloring". In *Journal of the ACM 23*, 43-49, 1/76.

5 Grimmet, G. R., C. J. H. McDiarmid. "On Colouring Random Graphs". In *Mathematical Proceedings of the Cambridge Philosophical Society 77*, 313-324, 1975.

6 Johnson, David S. "Worst Case Behavior of Graph Coloring Algorithms". In *Proceedings Southeastern Conference on Combinatorics, Graph Theory and Computing*, 513-527, 1974.

7 Karp, Richard M. "Reducibility Among Combinatorial Problems". In *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher (eds), 1972, Plenum Press.

8 Marchetti-Spaccamela, A., M. Talamo. "Probabilistic Analysis of Graph Colouring Algorithms", Technical report, University of Rome, 1983.

9 Stockmeyer, L. J. "Planar 3-Colorability is NP-complete". In *SIGACT News*, 19-25, 1973.

10 Tarjan, Robert Endre. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, 1983.

11 Wigderson, Avi. "Improving the Performance Guarantee for Approximate Graph Coloring". In *Journal of the ACM*, 729-735, 10/83.