Almost all k-Colorable Graphs are Easy to Color

Jonathan S. Turner*
Computer Science Department
Washington University
St. Louis, Missouri 63130

ABSTRACT

We describe a simple and efficient heuristic algorithm for the graph coloring problem and show that for all $k \geq 1$, it finds an optimal coloring for almost all k-colorable graphs. We also show that an algorithm proposed by Brélaz and justified on experimental grounds optimally colors almost all k-colorable graphs. Efficient implementations of both algorithms are given. The first one runs in $O(n+m\log k)$ time where n is the number of vertices and m the number of edges. The new implementation of Brélaz's algorithm runs in $O(m\log n)$ time. We observe that the popular greedy heuristic works poorly on k-colorable graphs.

[0]* This research supported in part by the National Science Foundation, grant number DCR-8409435.

1. Introduction

Let $G = (V, E)$ be a simple undirected graph. A k-coloring of G is a mapping $c : V \to \{1, 2, \ldots, k\}$; 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 [10] showed that the problem is NP-complete. Stockmeyer [12],[5] 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 [6] proved that unless P = NP, no polynomial time approximation algorithm can guarantee the use of fewer than $2\chi(G)$ colors. Furthermore, Johnson [9] 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 Wigderson [14] 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 [7] 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, and ϵ is any positive constant. (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 $n \to \infty$.) They also showed that a well-known greedy heuristic uses $\leq (1+\epsilon)n/\log_{1/(1-p)} n$ colors.

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 analysis of a backtrack search algorithm given in [15] and [2] reinforces this interpretation. These authors show that the expected size of the backtrack search tree explored by their algorithm is $O(1)$, when graphs are selected using the usual random graph model, suggesting once again that the usual model is too easy.

The set of k-colorable graphs on n vertices is the set of all n vertex graphs that can be colored with k or fewer colors. Let Q be a predicate defined on graphs and G_n^k be selected at random from the set of k-colorable graphs on n vertices. We say that Q holds for almost all k-colorable graphs if $\lim_{n\to\infty} \Pr(Q(G_k^n)) = 1$.

In section 2, we introduce a simple heuristic coloring algorithm and define a natural probability distribution over the set of k-colorable graphs. We then show that for graphs selected from this distribution, the algorithm finds an optimal coloring with high probability. In section 3, we show that this algorithm finds an optimal coloring for almost all k-colorable graphs. In section 4, we give similar results for Brélaz's algorithm. Section 5 gives efficient implementations of both algorithms. Section 6 gives experimental results that provide more detailed information on the performance of the two algorithms. Section 7 gives evidence that the popular greedy heuristic performs poorly on k-colorable graphs and section 8 contains closing remarks.

2. The No-Choice Algorithm

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.

The probability distribution $X_n(k, p)$ assigns non-zero probability to every k-colorable graph on n vertices, and zero probability to to every graph requiring more than k colors. However, the k-colorable graphs are not assigned equal probability; even for $p = 1/2$, some graphs are more likely than others (roughly speaking, graphs possessing many k-colorings are most likely). This distribution is however, closely related to the uniform probability distribution, which assigns equal probability to each k-colorable graph on n vertices. That relationship will be made clear in the next section. For the moment, we focus on the distribution $X_n(k, p)$.

Let $0 < p < 1$ be fixed and let $k = k(n)$ be an integer function that satisfies $2 \leq k \leq n$. Let Q be a predicate defined on graphs and let $\Pr(Q(G))$ be the probability that $Q(G)$ is true for $G \in X_n(k, p)$. We say that Q holds for *almost all* $G \in X_n(k, p)$ if $\lim_{n\to\infty} \Pr(Q(G)) = 1$.

In this section we present a heuristic coloring algorithm, which for constant p and k growing slowly with n finds a k-coloring for almost all $G \in X_n(k, p)$. In the next section, we will use this result to show that the algorithm successfully colors almost all 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 algorithm attempts to find a k-coloring of a graph $G = (V, E)$, where k is assumed to be an input parameter. The algorithm has two phases. In the first phase it attempts to find a k-clique by repeating the following step k times.

Clique Finding Step. Select a vertex x adjacent to all previously selected vertices.

If it finds a clique, it colors each of the vertices in the clique with a distinct color in $[1, k]$ and starts the second phase which consists of repeated applications of 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 (after coloring the initial k-clique). 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 for $G \in X_n(k,p)$.

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 $\lambda_n(c)$.

Theorem 2.1. *Let $0 < \epsilon < 1$, $0 < p < 1$ be fixed, $n \to \infty$ and $2 \leq k \leq (1-\epsilon)\lambda(p)$. For almost all $G \in X_n(k,p)$, the no-choice algorithm finds a k-coloring.*

We say that a graph is *uniquely k-colorable* if all proper k-colorings induce the same partition on the vertex set. Since the no-choice algorithm makes no arbitrary decisions with the exception of coloring the initial clique, the graphs it colors successfully are uniquely k-colorable.

Corollary 2.1. *Let $0 < \epsilon < 1$, $0 < p < 1$ be fixed, $n \to \infty$ and $2 \leq k \leq (1-\epsilon)\lambda(p)$. Almost all $G \in X_n(k,p)$ are uniquely k-colorable.*

To prove Theorem 2.1, we first define a class of graphs which we call *easily colorable graphs* and observe that the no-choice algorithm succeeds for all easily colorable graphs. We then present a series of lemmas which together imply that almost all $G \in X_n(k,p)$ are easily colorable.

We say that a k-colorable graph G satisfies the *clique property* if for every $r < k$, all cliques on r vertices can be extended to $r+1$ vertices.

Let G be a k-colorable graph containing at least one k-clique and let $\{x_1, \ldots, x_k\}$ be any k-clique in G. We define

$$\begin{aligned} A_i(x_1,\ldots,x_k) &= \{y \in V \mid \{y, x_j\} \in E \text{ for all } j \neq i\} \\ B_i(x_1,\ldots,x_k) &= \{y \in V \mid y \text{ is adjacent to some } z_j \in A_j(x_1,\ldots,x_k) \text{ for all } j \neq i\} \\ C_i(x_1,\ldots,x_k) &= \{y \in V \mid y \text{ is adjacent to some } z_j \in B_j(x_1,\ldots,x_k) \text{ for all } j \neq i\} \end{aligned}$$

Note that $x_i \in A_i(x_1,\ldots,x_k) \subseteq B_i(x_1,\ldots,x_k) \subseteq C_i(x_1,\ldots,x_k)$ for all i.

We say that a k-colorable graph G is *easily colorable* if the clique property holds and for *all* cliques $\{x_1,\ldots,x_k\}$,

$$\cup_{i=1}^{k} C_i(x_1,\ldots,x_k) = V.$$

It's easy to see that the no-choice algorithm will succeed for any easily colorable graph. It remains to show that almost all $G \in X_n(k,p)$ are easily colorable.

The following proposition (Angluin and Valiant [1]) is used in the proofs of several of the lemmas which follow. 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 all α, $0 < \alpha < 1$, $P(x \leq (1-\alpha)np) < e^{-\alpha^2 np/2}$ and $P(x \geq (1+\alpha)np) < e^{-\alpha^2 np/3}$.*

For $G \in X_n(k,p)$ we define c to be the randomly selected k-coloring used to generate G and we let

$$V_i = \{u \in V \mid c(u) = i\} \qquad n_i = |V_i| \qquad m = \min_{1 \leq i \leq k} n_i$$

Our first lemma puts a lower bound on $|V_i|$.

Lemma 2.1. *Let $0 < \epsilon < 1$, $0 < p < 1$ be fixed, $n \to \infty$ and $2 \leq k \leq (1-\epsilon)\lambda(p)$. For almost all $G \in X_n(k,p)$, $n_i = |V_i| \geq n/2k$ for all i.*

proof. Each n_i is a random variable drawn from $B(n, 1/k)$. By Proposition 2.1, the probability 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 $k = O(\log n)$. □

Lemma 2.2. *Let $0 < \epsilon < 1$, $0 < p < 1$ be fixed, $n \to \infty$ and $2 \leq k \leq (1-\epsilon)\lambda(p)$. For almost all $G \in X_n(k,p)$, the clique property holds.*

proof. By Lemma 2.1, the probability that $m < n/2k$ vanishes for large n. Assume then that $m \geq n/2k$. Let K_r be any clique of size $r < k$. The probability that there is no vertex y adjacent to all the vertices in K_r is $\leq (1-p^r)^{m(k-r)}$. There are at most n^r ways to select K_r, so the probability that there is an r-clique which cannot be extended is

$$\leq \sum_{r=1}^{k-1} n^r (1-p^r)^{m(k-r)} \leq kn^k(1-p^k)^{n/2k} \leq kn^k e^{-np^k/2k} \leq \exp[\ln k + k \ln n - n^{\epsilon}/2k] \to 0$$

since $k = O(\log n)$. □

LEMMA 2.3. *Let $0 < \epsilon < 1$, $0 < p < 1$ be fixed, $n \to \infty$ and $2 \le k \le (1-\epsilon)\lambda(p)$. For almost all $G \in X_n(k,p)$, if $x_1, \ldots, x_k$ is any k-clique with $x_i \in V_i$ $(1 \le i \le k)$ then $|A_i(x_1, \ldots, x_k)| \ge (1-\epsilon)n^\epsilon/2k$ for all i.*

proof. By Lemma 2.1, the probability that $m < n/2k$ vanishes for large n. Assume then that $m \ge n/2k$. From this and the bound on k, we obtain $n_i p^{k-1} \ge n^\epsilon/2k$ for all i. Let $s_i = |A_i(x_1, \ldots, x_k)|$ for a particular choice of $x_1, \ldots, x_k$. Using Proposition 2.1, we obtain

$$P(s_i \le (1-\epsilon)n^\epsilon/2k) \le P(s_i \le (1-\epsilon)n_i p^{k-1}) < e^{-\epsilon^2 n_i p^{k-1}/2} \le e^{-\epsilon^2 n^\epsilon/4k}$$

Since $x_1, \ldots, x_k$ can be chosen in at most n^k ways, the probability that there is any choice of $x_1, \ldots, x_k$ for which some s_i is smaller than $(1-\epsilon)n^\epsilon/2k$ is

$$< kn^k e^{-\epsilon^2 n^\epsilon/4k} = \exp[\ln k + k \ln n - \epsilon^2 n^\epsilon/4k] \to 0$$

since $k = O(\log n)$. □

LEMMA 2.4. *Let $0 < \epsilon < 1$, $0 < p < 1$ be fixed, $n \to \infty$ and $2 \le k \le (1-\epsilon)\lambda(p)$. For almost all $G \in X_n(k,p)$, if $x_1, \ldots, x_k$ is any k-clique with $x_i \in V_i$ $(1 \le i \le k)$ then $|B_i(x_1, \ldots, x_k)| \ge n_i - k\lambda(1-p)$ for all i.*

proof. Suppose G satisfies the following property

(*) For every choice of $U_1, \ldots, U_k$ where $U_i \subseteq V_i$ and $|U_i| = r \ge k\lambda(1-p)$, each U_i contains a vertex y_i having a neighbor in each U_j $(j \ne i)$.

If in addition, $|A_i(x_1, \ldots, x_k)| \ge k\lambda(1-p)$ for all i, then it follows that $|B_i(x_1, \ldots, x_k)| \ge n_i - k\lambda(1-p)$ for all i. By Lemma 2.3, almost all G satisfy $|A_i(x_1, \ldots, x_k)| \ge k\lambda(1-p)$ for all i, so it suffices to show that (*) holds for almost all G.

Consider a particular choice of $U_1, \ldots, U_k$ and let $y \in U_1$. The probability that there is a $j \in [2,k]$ such that U_j has no neighbor of y is $\le k(1-p)^r$. The probability that U_1 contains no vertex with neighbors in each of $U_2, \ldots, U_k$ is $\le (k(1-p)^r)^r$. Since $U_i \subseteq V_i$ and the V_i partition V, each choice of $U_1, \ldots, U_k$ corresponds to a distinct choice of kr elements from V. Hence, the probability that G does not satisfy (*) is

$$\le k\binom{n}{kr}(k(1-p)^r)^r \le k^{r+1}\left(\frac{en}{kr}\right)^{kr}(1-p)^{r^2} \le \left(\frac{en}{r}(1-p)^{r/k}\right)^{kr} \le (e/r)^{kr} \to 0$$

since $r = \Omega(\log n)$. □

LEMMA 2.5. *Let $0 < \epsilon < 1$, $0 < p < 1$ be fixed, $n \to \infty$ and $2 \le k \le (1-\epsilon)\lambda(p)$. For almost all $G \in X_n(k,p)$, if $x_1, \ldots, x_k$ is any k-clique with $x_i \in V_i$ $(1 \le i \le k)$ then $\cup_{i=1}^k C_i(x_1, \ldots, x_k) = V$.*

proof. Suppose G satisfies the following property.

(**) $u \in V_i \wedge j \ne i \Rightarrow u$ has at least $(1-\epsilon)np/2k$ neighbors in V_j.

If in addition, $|B_i(x_1,\ldots,x_k)| > n_i-(1-\epsilon)np/2k$ for all i, then it follows that $C_i(x_1,\ldots,x_k) = V_i$ for all i. By Lemma 2.4, almost all G satisfy $|B_i(x_1,\ldots,x_k)| > n_i-(1-\epsilon)np/2k$ for all i, so it suffices to show that (**) holds for almost all G.

By Lemma 2.1, the probability that $m < n/2k$ vanishes for large n. Assume then 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,

$$P(d_i(x) \leq (1-\epsilon)np/2k) \leq P(d_i(x) \leq (1-\epsilon)n_ip) < e^{-\epsilon^2 n_ip/2} \leq e^{-\epsilon^2 np/4k}$$

So the probability that G does not satisfy (**) is $\leq kne^{-\epsilon^2 np/4k} \to 0$. □

This completes the proof of Theorem 2.1.

3. Most k-Colorable Graphs are Easily Colorable

In this section we show that almost all k-colorable graphs are easily colorable. The proof is indirect and depends on a careful examination of the process by which graphs $G \in X_n(k,1/2)$ are generated. We view this process as a random walk in a certain graph which we now define.

Let Θ_n^k be the set of all k-colorings for n vertex graphs. Let Φ_n^k be the set of all k-colorable graphs on n vertices. We define $\Upsilon_n^k = (W,F)$ to be a directed graph in which

$$\begin{aligned} W &= \{u\} \cup \Theta_n^k \cup \Phi_n^k \\ F &= \{[u,c] \mid c \in \Theta_n^k\} \cup \{[c,G] \mid c \in \Theta_n^k \wedge G \in \Phi_n^k \wedge c \text{ is a proper } k\text{-coloring for } G\} \end{aligned}$$

The structure of Υ_n^k is illustrated in Figure 1. The process by which graphs in $X_n(k,1/2)$ are generated can be viewed as a two step random walk in Υ_n^k starting at vertex u. We first select a coloring, giving each one equal probability of selection. We then select a graph for which the selected coloring is proper, giving equal probability to each such graph.

Let $\theta \subseteq \Theta_n^k$ contain all colorings that assign each color to at least $n/2k$ vertices. We refer to these as *balanced colorings.* Let $\phi \subseteq \Phi_n^k$ contain all easily colorable graphs that can be colored using a balanced coloring. Also, let $\overline{\theta} = \Theta_n^k - \theta$ and $\overline{\phi} = \Phi_n^k - \phi$. With these definitions we can give a more detailed picture of Υ_n^k as shown in Figure 2. Note that all edges leaving $\overline{\theta}$ terminate in $\overline{\phi}$.

We claim that $|\overline{\phi}|/|\phi| \to 0$ as $n \to \infty$. By Corollary 2.1 each $G \in \phi$ has exactly $k!$ incoming edges. Also, note that each $G \in \overline{\phi}$ has at least $k!$ incoming edges. Hence, we can prove our claim by showing that the ratio of the number edges entering $\overline{\phi}$ to the number of edges entering ϕ vanishes.

Let $d(\theta\phi)$ be the number of edges joining θ and ϕ. Define $d(\theta\overline{\phi})$ and $d(\overline{\theta}\,\overline{\phi})$ similarly. We will prove our claim by showing that

$$\frac{d(\theta\overline{\phi}) + d(\overline{\theta}\,\overline{\phi})}{d(\theta\phi)} \to 0$$

First, note that Lemmas 2.1–2.5 together imply that given any balanced coloring, almost all graphs for which that coloring is proper are easily colorable. Hence, $d(\theta\overline{\phi})/d(\theta\phi)$ vanishes. It remains to show that $d(\overline{\theta}\,\overline{\phi})/d(\theta\phi)$ vanishes. We can do this by establishing the following sub-claims.

- $|\overline{\theta}|/|\theta| \to 0$
- The expected out-degree of a randomly selected vertex in $\overline{\theta}$ is less than the expected out-degree of a randomly selected vertex from θ.

The first sub-claim follows immediately from Lemma 2.1. We now prove the second. For any coloring k-coloring c on n vertices, let n_i be the number of vertices assigned color i $(1 \leq i \leq k)$ and define

$$\sigma(c) = \sum_{i=1}^{k} \binom{n_i}{2} \qquad \text{and} \qquad \delta(c) = 2^{\binom{n}{2}-\sigma(c)}$$

Note that $\delta(c)$ is exactly the number of graphs for which c is proper. Let c_a be a randomly selected coloring in $\theta \cup \overline{\theta}$, c_b be a randomly selected coloring in θ and c_u be a randomly selected coloring in $\overline{\theta}$. We wish to show that $\mathrm{E}(\delta(c_u)) \leq \mathrm{E}(\delta(c_b))$, which we do by showing that $\mathrm{E}(\delta(c_u)) \leq \mathrm{E}(\delta(c_a))$

Aside. If x is any random variable defined on a discrete sample space A and $B \subseteq A$ then $\mathrm{E}(x|B) \leq \mathrm{E}(x) \Leftrightarrow \mathrm{E}(x|B) \leq \mathrm{E}(x|A-B)$.

Note first, that

$$2^{\binom{n}{2}-\mathrm{E}(\sigma(c_a))} \leq \mathrm{E}(\delta(c_a))$$

by Jensen's inequality [4, 8]. Next, note that if α is a lower bound for $\sigma(c_u)$, then

$$\mathrm{E}(\delta(c_u)) \leq 2^{\binom{n}{2}-\alpha}$$

So it suffices to find an appropriate bound α and show that $\alpha \geq \mathrm{E}(\sigma(c_a))$.

We start by calculating $\mathrm{E}(\sigma(c_a))$.

$$\mathrm{E}(\sigma(c_a)) = \frac{1}{2}\sum_{i=1}^{k}(\mathrm{E}(n_i^2) - \mathrm{E}(n_i))$$

Since in this case, $\mathrm{E}(n_i) = n/k$ and $\mathrm{E}(n_i^2) = (n/k) + n(n-1)/k^2$, it follows that

$$\mathrm{E}(S) = \frac{1}{k}\binom{n}{2}$$

Before we determine α, we note that for any convex function f,

$$f((x_1 + \cdots + x_k)/k) \leq (f(x_1) + \cdots + f(x_k))/k$$

(see [8, page 72]). For example,

$$\sum_{i=1}^{k} \binom{n_i}{2} \geq k\binom{n/k}{2} \qquad \text{so} \qquad \sigma(c_a) \geq k\binom{n/k}{2}$$

Since in any unbalanced coloring, at least one of the n_i is no larger that $n/2k$, $\sigma(c_u) \geq \min_{0 \leq x \leq n/2k} g(x)$ where

$$g(x) = \binom{x}{2} + (k-1)\binom{(n-x)/(k-1)}{2}$$

The function $g(x)$ has a single global minimum at $x = n/k$. Its minimum value in the interval $[0, n/2k]$ occurs at $n/2k$ Therefore, $\sigma(c_u) \geq \alpha = g(n/2k)$. A straightforward calculation yields

$$\alpha = \frac{1}{k}\binom{n}{2} + \frac{n}{2k}\left[\frac{n}{4(k-1)} - (k-1)\right]$$

The expression in brackets is non-negative when $n \geq 4(k-1)^2$. Consequently, $\mathrm{E}(\delta(c_u)) \leq \mathrm{E}(\delta(c_b))$, when $n \geq 4(k-1)^2$ and we have the following theorem.

Theorem 3.1. *Let $0 < \epsilon < 1$ be fixed, $n \to \infty$ and $2 \leq k \leq (1-\epsilon)\log_2 n$. Almost all k-colorable graphs are easily colorable.*

Corollary 3.1. *Let $0 < \epsilon < 1$ be fixed, $n \to \infty$ and $2 \leq k \leq (1-\epsilon)\log_2 n$. For almost k-colorable graphs the no-choice algorithm produces a k-coloring.*

4. Brélaz's Algorithm

The no-choice algorithm is similar to one proposed by Brélaz [3] 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 k-colorable graph G on n vertices that is easily colorable. Because G satisfies the clique property, the first k vertices colored will form a k-clique. Once the first k vertices have been colored, the algorithm repeatedly selects a vertex x for which $|avail(x)| = n - k + 1$; that is, it mimics the no-choice algorithm. These observations yield the following theorem.

Theorem 4.1. *Let $0 < \epsilon < 1$, $0 < p < 1$ be fixed, $n \to \infty$ and $k \leq (1-\epsilon)\lambda(p)$. For almost all $G \in X_n(k,p)$, Brélaz's algorithm produces a k-coloring.*

Corollary 4.1. *Let $0 < \epsilon < 1$ be fixed, $n \to \infty$ and $2 \leq k \leq (1-\epsilon)\log_2 n$. For almost k-colorable graphs Brélaz's algorithm produces a k-coloring.*

5. Efficient Implementations of Coloring Algorithms

A program implementing the no-choice algorithm is shown in Figure 3. (The algorithmic notation is adapted from Tarjan [13].) 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 4 is used to find a clique. The *clique* program can be implemented to run in linear time, if the set S is represented as a bit vector and a supplementary list of vertices ordered by degree is used to determine x on each iteration. (This supplementary list can be sorted in linear time using a radix sort.) 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.
$member(x, s)$	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 must 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 5. 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 5.1. *The no-choice algorithm can be implemented to run in $O(n + m \log k)$ time on graphs with n vertices and m edges.*

Note that this is superior to the obvious implementation only when the graph is quite sparse. However, the same technique also yields a substantial improvement to Brélaz's algorithm for all but the densest graphs.

In [3] 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 6 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 $deg(x) > deg(y)$. The *siftup* operation restores the order of items in the heap after the changes to $avail(y)$ and $deg(y)$. See [13] for details. 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 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 5.2. *Brélaz's algorithm can be implemented to run in $O(m \log n)$ time on graphs with n vertices and m edges.*

6. Experimental Results

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. The results are summarized in Figure 7. 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 in section 2. 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 7 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 2.1, which suggests that $\beta_n(p)$ grows in proportion to $\log n$.

Figure 8 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 4.1.

7. 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 $c(x) = \min avail(x)$.

Grimmet and McDiarmid [7] 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.

Let $S_i(r) = \{1 \leq z \leq r \mid c(z) = c'(z) = i\}$ for $1 \leq i \leq k$ 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 $1 \leq z < r$ and $c(r) \neq c'(r)$.

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

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

THEOREM 7.1. *Let $0 < p < 1$, $k \geq 1$ 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 $k!\,(1 - \sum_{r=1}^{n} Q(r))$.*

The terms in $\sum_{r=1}^{n} Q(r)$ decline rapidly, so for small k, we can use Theorem 7.1 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

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

Using these equations and Theorem 7.1 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 9 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.

8. Closing Remarks

In this paper, we have shown that when k is not too large relative to n, almost all k-colorable graphs are easily colorable. For larger values of k, all the algorithms discussed here perform poorly. One open problem is to find algorithms that work well when k is as large as say, $n^{1/2}$.

Theorem 3.1 implies that the complexity of recognizing k-colorable graphs is caused by a relatively small number of "pathological cases." Similar results may hold for other NP-complete problems. Indeed it may be possible to classify NP-complete sets as hard or easy based on whether or not they contain large subsets whose members can be efficiently identified. The current work represents a first step in such a classification.

Acknowledgements. I want to thank Professor Herbert Wilf for his tireless (and sometimes tiresome) criticisms of early drafts of this paper. His efforts have improved it in both substance and form. I especially want to thank him for doubting a conjecture that appeared in an earlier version. His skepticism goaded me into renewing the attack, this time successfully. The former conjecture appears here as Theorem 3.1. I would also like to thank two anonymous referees for their careful reading and helpful comments.

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] Bender, Edward A and Herbert S. Wilf. "A Theoretical Analysis of Backtracking in the Graph Coloring Problem," *Journal of Algorithms* 6, 275–282, 1985.

[3] Brelaz, Daniel. “New Methods to Color the Vertices of a Graph.” In *Communications of the ACM 22*, 251–256, 4/79.

[4] Feller, William. *An Introduction to Probability Theory and Its Applications*, John Wiley & Sons, 1971.

[5] Garey, Michael R., David S. Johnson, L. J. Stockmeyer. “Some Simplified NP-complete Graph Problems.” In *Theoretical Computer Science 1*, 237–267, 1976.

[6] Garey, Michael R., David S. Johnson. “The Complexity of Near-Optimal Graph Coloring.” In *Journal of the ACM 23*, 43–49, 1/76.

[7] Grimmet, G. R., C. J. H. McDiarmid. “On Colouring Random Graphs.” In *Mathematical Proceedings of the Cambridge Philosophical Society 77*, 313–324, 1975.

[8] Hardy, G. H., J. E. Littlewood, G. Pólya. *Inequalities*, Cambridge University Press, 1934.

[9] Johnson, David S. “Worst Case Behavior of Graph Coloring Algorithms.” In *Proceedings Southeastern Conference on Combinatorics, Graph Theory and Computing*, 513–527, 1974.

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

[11] Marchetti-Spaccamela, A., M. Talamo. “Probabilistic Analysis of Graph Colouring Algorithms,” Technical report, University of Rome, 1983.

[12] Stockmeyer, L. J. “Planar 3-Colorability is NP-complete.” In *SIGACT News*, 19–25, 1973.

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

[14] Wigderson, Avi. “Improving the Performance Guarantee for Approximate Graph Coloring.” In *Journal of the ACM*, 729–735, 10/83.

[15] Herbert S. Wilf. “Backtrack: an $O(1)$ Expected Time Algorithm for the Graph Coloring Problem.” In *Information Processing Letters* 18, 119–121, 1984.

Figure 1: Structure of Υ_n^k

Figure 2: Detailed Structure of Υ_n^k

```
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 $x \in [1\,..\,n] \rightarrow c(x) \leftarrow -1$; $avail(x) \leftarrow \{1, \ldots, k\}$; rof;
    $X \leftarrow$ clique$(k, n, neighbors)$;
    if $|X| \neq k \rightarrow$ return false fi;
    $i \leftarrow 1$; for $x \in X \rightarrow c(x) \leftarrow i$; $i \leftarrow i + 1$; rof;
    $Q \leftarrow [\,]$;
    for $x \in X \rightarrow$
        for $y \in neighbors(x) \rightarrow$
            $avail(y) \leftarrow avail(y) - c(x)$;
            if $c(y) = -1$ and $|avail(y)| = 1 \rightarrow$
                $Q \leftarrow Q \,\&\, [y]$; $c(y) \leftarrow 0$;
            fi;
        rof;
    rof;
    $nc \leftarrow k$;
    do $Q \neq [\,] \rightarrow$
        $x \leftarrow Q[1]$; $Q \leftarrow Q[2\,..]$;
        if $|avail(x)| \neq 1 \rightarrow$ return false fi;
        $c(x) \leftarrow \min avail(x)$; $nc \leftarrow nc + 1$;
        for $y \in neighbors(x) \rightarrow$
            $avail(y) \leftarrow avail(y) - c(x)$;
            if $c(y) = -1$ and $|avail(y)| = 1 \rightarrow$
                $Q \leftarrow Q \,\&\, [y]$; $c(y) \leftarrow 0$;
            fi;
        rof;
    od;
    return $nc = n$;
end;
```

Figure 3: Program Implementing the No Choice Algorithm

```
**set function** clique(**integer** $k, n$; **graph** *neighbors*);
    **set** $S, K$;
    $S \leftarrow \{1, \ldots, n\}$;
    $K \leftarrow \emptyset$;
    **do** $S \neq \emptyset \rightarrow$
        Select $x \in S$ of maximum degree.
        $K \leftarrow K \cup \{x\}$;
        $S \leftarrow S \cap neighbors(x)$;
    **od**;
    **return** $K$;
**end**;
```

Figure 4: Subroutine for Finding a Clique

```
$i \leftarrow$ member$(x, s)$;
**if** $i \neq [\,] \rightarrow$
    deleteinterval$(i, s)$;
    **if** $i.lo < x \rightarrow$ insertinterval$([i.lo, x-1], s)$ **fi**;
    **if** $i.hi > x \rightarrow$ insertinterval$([x+1, i.hi], s)$ **fi**;
**fi**;
```

Figure 5: Program Fragment Implementing the Delete Operation

```
**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** $x \in [1\,..\,n] \rightarrow$
        $c(x) \leftarrow 0$;
        $avail(x) \leftarrow \{1, \ldots, n\}$;
        $deg(x) \leftarrow |neighbors(x)|$;
    **rof**;
    $h \leftarrow$ makeheap($\{1, \ldots, n\}$);
    **do** $h \neq \rightarrow$
        $x \leftarrow$ deletemin($h$);
        $c(x) \leftarrow \min avail(i)$;
        **for** $y \in neighbors(x) \rightarrow$
            **if** $c(y) = 0 \rightarrow$
                $avail(y) \leftarrow avail(y) - c(x)$;
                $deg(y) \leftarrow deg(y) - 1$;
                siftup($y, h$);
            **fi**;
        **rof**;
    **od**;
**end**;
```

Figure 6: Program Implementing Brélaz's Algorithm

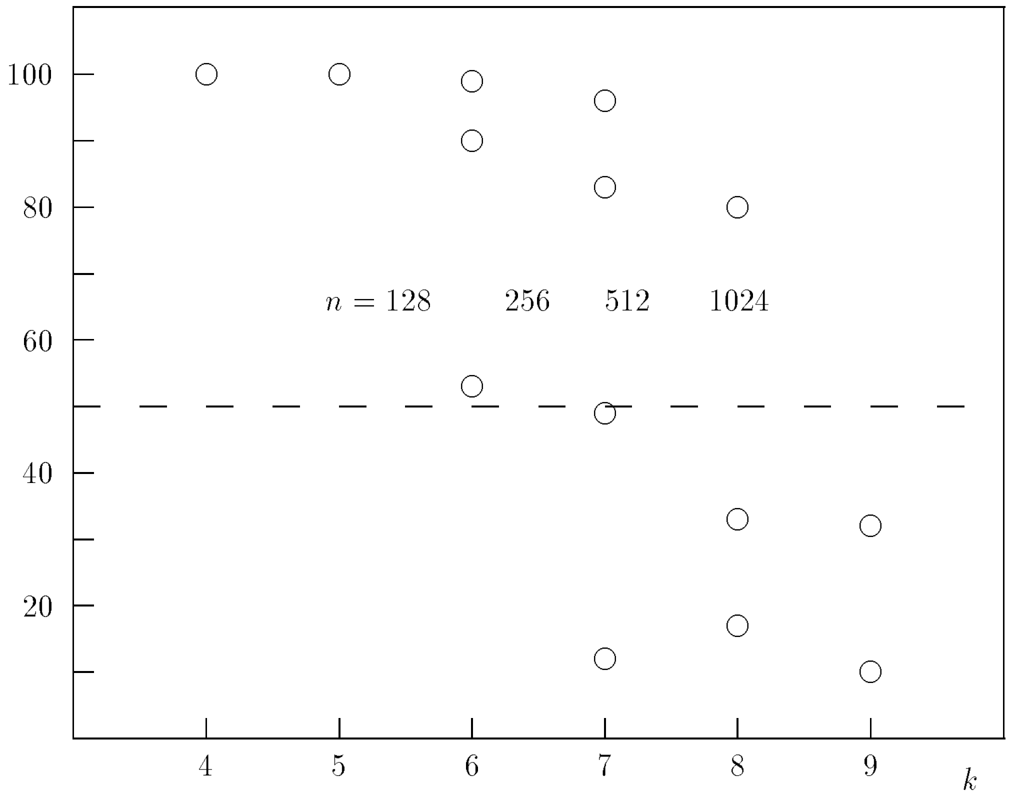

Figure 7: Success Rate of No Choice Algorithm for Graphs in $X_n(k, .5)$

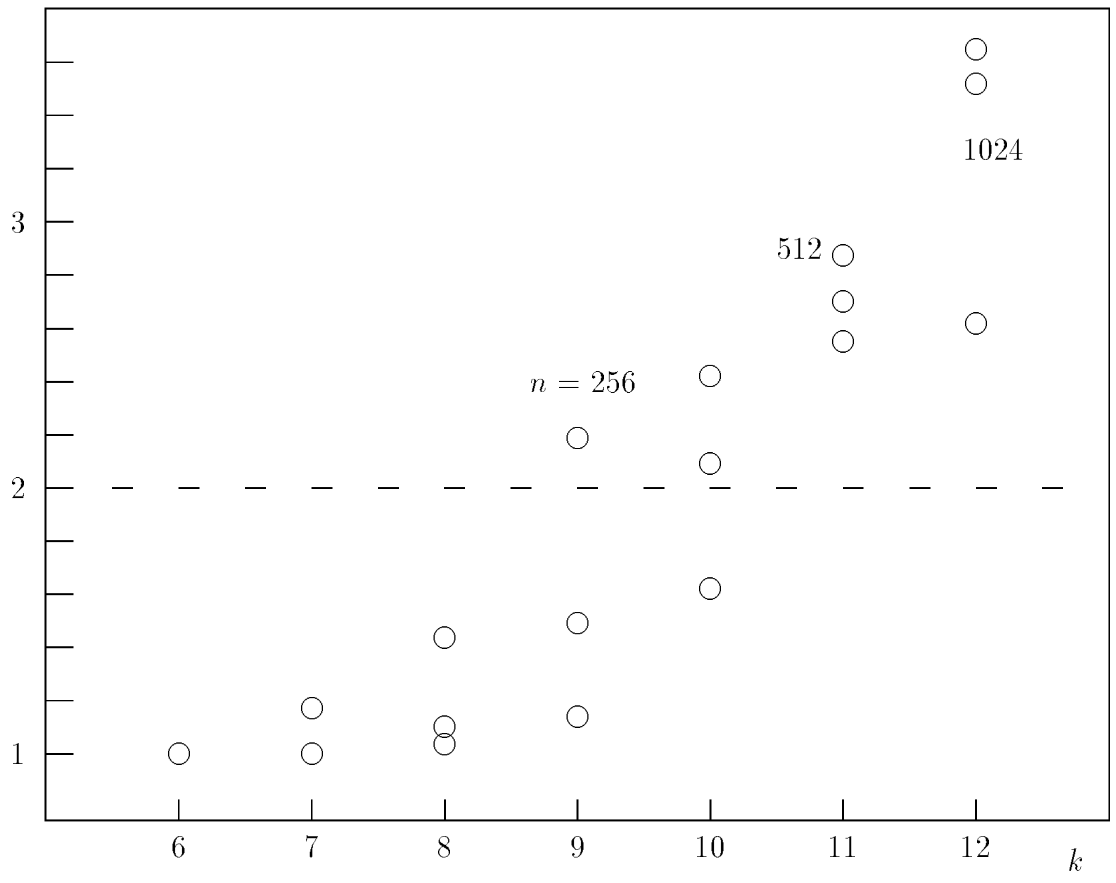

Figure 8: Average Performance Ratio of Brélaz's Algorithm for Graphs in $X_n(k, .5)$

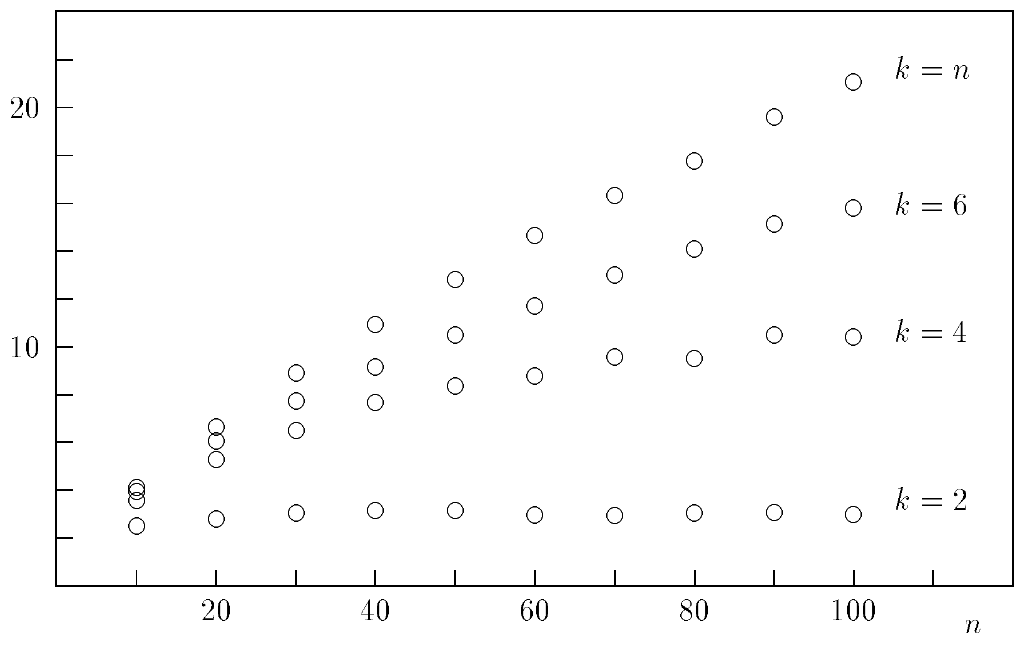

Figure 9: Average Number of Colors Used by Greedy Algorithm for Graphs in $X_n(k, .5)$

List of Captions

Figure 1: Structure of Υ_n^k

Figure 2: Detailed Structure of Υ_n^k

Figure 3: Program Implementing the No Choice Algorithm

Figure 4: Subroutine for Finding a Clique

Figure 5: Program Fragment Implementing the Delete Operation

Figure 6: Success Rate of No Choice Algorithm for Graphs in $X_n(k, .5)$

Figure 7: Program Implementing Brélaz's Algorithm

Figure 8: Average Performance Ratio of Brélaz's Algorithm for Graphs in $X_n(k, .5)$

Figure 9: Average Number of Colors Used by Greedy Algorithm for Graphs in $X_n(k, .5)$