Iterated Greedy Graph Coloring and the Difficulty Landscape

by

Joseph C. Culberson

Technical Report TR 92–07
June 1992

DEPARTMENT OF COMPUTING SCIENCE
The University of Alberta
Edmonton, Alberta, Canada

Iterated Greedy Graph Coloring and the Difficulty Landscape

Joseph C. Culberson [*] [†]

June 1992, Revised May 1993

Abstract

Many heuristic algorithms have been proposed for graph coloring. The simplest is perhaps the greedy algorithm. Many variations have been proposed for this algorithm at various levels of sophistication, but it is generally assumed that the coloring will occur in a single attempt. We note that if a new permutation of the vertices is chosen which respects the independent sets of a previous coloring, then applying the greedy algorithm will result in a new coloring in which the number of colors used does not increase, yet may decrease. We introduce several heuristics for generating new permutations that are fast when implemented and effective in reducing the coloring number.

The resulting Iterated Greedy algorithm(IG) can obtain colorings in the range 100 to 103 on graphs in $\mathcal{G}_{1000,\frac{1}{2}}$. More interestingly, it can optimally color k-colorable graphs with k up to 60 and $n = 1000$, exceeding results of anything in the literature for these graphs.

We couple this algorithm with several other coloring algorithms, including a modified Tabu search, and one that tries to find large independent sets using a pruned backtrack. With these combined algorithms we find 86 and 87 colorings for $\mathcal{G}_{1000,\frac{1}{2}}$.

Finally, we explore the areas of difficulty in probabilistic graph space under a natural parameterization. Specifically, we check our system on k-colorable graphs in $\mathcal{G}_{300,p,k}$ for $0.05 \leq p \leq 0.95$ and $2 \leq k \leq 105$. We find a narrow ridge where the algorithms fail to find the specified coloring, but easy success everywhere else.

[*]Supported by Natural Sciences and Engineering Research Council Grant No. OGP8053. Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2H1. email:joe@cs.ualberta.ca

[†]Available ftp ftp.cs.ualberta.ca or gopher gopher.cs.ualberta.ca

Part I
Introduction

Graph coloring is perhaps one of the most notorious of the NP-complete problems. It has many applications in scheduling and timetabling. The best deterministic exact algorithms are hopelessly exponential; the best known polynomial time approximation algorithms cannot guarantee a coloring assignment within less than $O(n^{0.4})$ [2] of the optimal coloring and are very complex. Since it is known that it is NP-hard to guarantee a coloring within a factor of two of the optimal [11], this leaves a large gap in our knowledge of the difficulty of coloring and dismal prospects in the worst case.

Numerous algorithms have been developed for average case coloring. The simple greedy algorithm starts with some permutation of the vertices and as each vertex is considered in turn, it is assigned the minimum color that does not cause a conflict. For random graphs in which each vertex pair is assigned an edge with probability 1/2, it is known that asymptotically this will on average use about twice as many colors as required[14]. In the worst case it will use $\Theta(n)$ times as many colors as needed[17]. Many heuristics have been proposed for ordering the vertices for coloring by the greedy algorithm. But usually the greedy algorithm is seen as a one shot approach to coloring.

In this paper we will examine the possibility of applying the greedy algorithm repeatedly, basing each successive ordering on the preceding coloring, using various heuristics. One feature of this approach is that the successive coloring numbers will be non-increasing.

Our heuristics can be used to generate new permutations starting with any valid coloring. Thus we can use Brélaz's DSATUR [5] algorithm to form an initial coloring, and then proceed with our heuristics. We also explore the use of Tabu search, developing a method to combine this technique with ours by alternating between them. Bollobás and Thomason [4] present an algorithm that recursively selects an approximation to the largest independent set from the uncolored vertices, and colors it. They report the best coloring results for the class $\mathcal{G}_{1000,\frac{1}{2}}$ to date (see section 3 for definitions of graph classes). We show that for various random graph classes improvements can be made by combining these techniques.

Finally we explore the coloring landscape with this system, as we vary a known coloring number, and the density of the graph. We find classes of k-colorable graphs that even our full system cannot color well. On the other hand, most k-colorable graphs are easily colored, even for values of k well outside the ranges suggested in previous analyses [27, 19].

1 Definitions

Given a simple undirected graph $G = (V, E)$, where V is a set of vertices, and E is a set of pairs of vertices called edges, a k-coloring of G is an assignment $C : V \rightarrow \{1 \ldots k\}$ such that if $C(v) = C(w)$ then $(v, w) \notin E$. The set $\{1 \ldots k\}$ is the set of *colors.* The chromatic number $\chi(G)$ is the minimal k for which there exists a k-coloring of G. The coloring problem is given a graph, obtain a χ-coloring of it. Since this is generally impractical, we will relax the problem and attempt to obtain k-colorings, where k is an approximation of χ. Such a coloring is called a *(valid) approximate coloring.*

An *independent set* is $V' \subseteq V$ such that for $v, w \in V', (v, w) \notin E$. A k-coloring thus partitions the vertex set into independent sets (sometimes called the *color classes*), with each color class being the set of vertices of a given color. An assignment of colors to vertices such that for some edges (v, w) we have $C(v) = C(w)$ is an *approximately correct coloring.* In approximately correct colorings, each such edge is referred to as a *conflict*, and we say that vertex v *is in conflict with* vertex w.

A *clique* is a set $V' \subseteq V$ such that for $v, w \in V', (v, w) \in E$. A complement $\bar{G}$ of G is a graph in which each pair v, w is an edge in $\bar{G}$ exactly when it is not an edge in G. Thus, an independent set in G is a clique in $\bar{G}$.

2 Algorithm Classes

Many algorithms have been proposed to obtain approximate colorings in reasonable time. These algorithms generally fall into one of the following classes:

Greedy Find large independent sets and color each of them with one color. One of the simplest implementations is the greedy algorithm, described previously. Intuitively it seems reasonable that the larger the independent sets, the fewer the colors used. However, being too greedy on the first sets may cause problems for later choices, forcing the use of unnecessary colors. Thus, these approaches produce *approximate colorings.* There are many variations, heuristics and analyses [29, 30, 22, 17, 16, 24, 8, 21, 20, 4, 25].

Partition Partition the vertices by some means, and then attempt to remove conflicts by moving vertices from one partition to another. These methods may produce an *approximately correct coloring*, since on termination some conflicts may remain. If the partition number is not the chromatic number, the results may also be an approximation to the coloring number. Example algorithms are Anti-voter[28] and Tabu search[15] and simulated annealing[6]. Tabu search will be described in more detail in section 6.

Clique After choosing the first vertex, choose vertices with a maximal number of constraints on the colors available to them. This is almost the opposite of the first method as now we choose vertices that form large cliques with respect to the vertices already chosen. Examples of these are the No Choice algorithm[27] and Brélaz's DSATUR algorithm[5]. Counter example graphs for DSATUR are provided in [26]. The intuition is that if we choose vertices which are forced to be a certain color, we are less likely to make an error that forces bad colorings later on. We will discuss an implementation of Brélaz's algorithm in a subsequent section.

Zykov Do partial traversals of Zykov's tree[31]. These algorithms use the recursive structure imposed on graphs by the Zykov decompositions, but rather than do a complete search, use heuristics to attempt to find a good if not optimal coloring. We will not be discussing these methods further in this paper; see [10, 9] for further details.

Other A few algorithms use other methods such as eigenvalue decomposition[1].

Most of these algorithms make a single pass over the set of vertices and edges, and produce a coloring result. Tabu and simulated annealing are notable exceptions as they may iterate a basic search loop an arbitrary number of times.

In this paper we will look at an iterated greedy algorithm, which also may be run for an arbitrary amount of time. Unlike partitioning algorithms, it will always produce a correct but possibly non-optimal coloring.

3 Classes of Graphs

There are many classes of graphs that could and should be used to test coloring algorithms. Time permits testing only on a few however.

The most natural class is perhaps the class $\mathcal{G}_{n,p}$, where n is the number of vertices, and for each pair of vertices an edge is assigned with probability p. This class of graphs has been deeply studied with respect to coloring, especially for $p = 1/2$. Asymptotically, for a fixed probability p, the chromatic number is known to almost surely (a.s.) be[3]

$$\left(\frac{1}{2} + o(1)\right) \log\left(\frac{1}{1-p}\right) \frac{n}{\log n}$$

The greedy algorithm is known to assign approximately twice the chromatic number when coloring graphs with $p = 1/2$ [23, 14].

In one sense, the class of random graphs is easy, since good approximations are obtained on average. Also, we do not have a specific target color to shoot for. On the other hand, they are difficult to color optimally or even nearly so. That is, we are unlikely to obtain colorings far from optimal but not likely to obtain very close colorings either. Also, not knowing the chromatic number means we can never be sure just how well or poorly we are doing.

We define the following class of graphs to provide another test bed for our algorithms. $\mathcal{G}_{n,p,k}$ is the set of graphs with n vertices, partitioned into k as nearly equal sized sets as possible. Edges are assigned with probability p provided that the vertices in question are in distinct elements of the partition. No edges are assigned between vertices within any element of the partition. Thus, these graphs are always k-colorable, although the chromatic number may be less than k in some cases.

In practice, the partitioning is done by first generating a random permutation, and then placing all vertices occupying positions congruent modulo k in the permutation into the same set. The permutation is taken to avoid accidental use of the partitioning structure by a coloring algorithm. This class forms much of the basis of our empirical study.

In general partitions may be selected in several alternate ways. These allow variability in the sizes of the elements of the partition. We define a class of graphs $\mathcal{G}_{n,p,k,\epsilon}$, with $0 \leq \epsilon < k$, as follows. If $\epsilon = 0$ then assign to each vertex in turn one of the values $1 \ldots k$ chosen at random. This subclass of graphs is used in the literature [10, 27]. Otherwise, choose γ at random with $0 \leq \gamma \leq \epsilon$. and then assign the value i chosen at random from the range $[\gamma, k]$. As ϵ increases, the variability increases. This allows us to tune for hardness of graphs with respect to our algorithms.

In general, we found for any p and k, the class $\mathcal{G}_{n,p,k}$ is more difficult than $\mathcal{G}_{n,p,k,\epsilon}$ for any ϵ, and that as ϵ increases, the graphs become easier to color. That is, the hardest graphs to color are those in which all sets were of equal size. Thus for all of the results reported here we use either $\mathcal{G}_{n,p}$ or $\mathcal{G}_{n,p,k}$. Note that $\mathcal{G}_{n,p}$ is just the special case $\mathcal{G}_{n,p,n}$.

When we refer to coloring a graph in $\mathcal{G}_{n,p,k}$ or $\mathcal{G}_{n,p,k,\epsilon}$ we say that a k-coloring is a *specified* coloring of the graph. This terminology is used to clearly distinguish the k-coloring from an optimal χ-coloring which in some cases may be less.

Part II

The Iterated Greedy Algorithm

In section 4, we describe the Iterated Greedy algorithm, the heuristics and definitions required to understand it, and some graphs which can cause it to produce arbitrarily bad colorings. In the section 5, we describe the results of experiments using this algorithm with various heuristic mixes on classes of graphs introduced earlier.

4 Algorithm Description

Let $V = \{v_1 \dots v_n\}$. Let π be a permutation of $\{1 \dots n\}$. The simple greedy algorithm is

```
Color $v_{\pi(1)}$ with color 1.
for $i$ from 2 to $n$ do
    Assume $k$ colors have been used.
    Color $v_{\pi(i)}$ with the minimum from $\{1 \dots k+1\}$
    such that no conflict is created.
```

It is easy to see that for any graph, there is some permutation from which the greedy algorithm will produce a χ-coloring. Many heuristics for selecting permutations have been proposed. One of the earliest was that of Welsh and Powell [29] who suggested ordering the vertices by decreasing degree. Variations on that include ordering the vertices by degree in the subgraph which remains after the vertices up to the one in question are deleted.

However, for our purposes we will use the simple greedy algorithm as the core of an iterative process, where the permutations will be determined by previous colorings. The idea is to use information obtained in a previous coloring to produce an improved coloring. The following states that if we take any permutation in which the vertices of each color class are adjacent in the permutation, then applying the greedy algorithm will produce a coloring at least as good.

Lemma 4.1 *Let C be a k-coloring of a graph G, and π a permutation of the vertices such that if $C(v_{\pi(i)}) = C(v_{\pi(m)}) = c$, then $C(v_{\pi(j)}) = c$ for*

$i \leq j \leq m$. Then, applying the greedy algorithm to the permutation π will produce a coloring C' using k or fewer colors.

Proof: The proof is a simple induction showing that the first i color classes listed in the permutation will be colored with i or fewer colors. Clearly, the first color class listed will be colored with color 1. Suppose some element of the ith class requires color $i+1$. This means that it must be adjacent to a vertex of color i. But by induction the vertices in the 1st to the $i-1$th classes used no more than $i-1$ colors. Thus, the conflict has to be with a member of its own color class, but this contradicts the assumption that C is a valid coloring. ■

Clearly, the order of the vertices within the color classes cannot affect the next coloring, because the color classes are independent sets. It is easily seen that if the permutation is such that the k color classes generated by a previous iteration of the greedy algorithm are listed in order of increasing color, then applying the greedy algorithm again will produce an identical coloring. This is the heart of IG: repeatedly generate new permutations satisfying the conditions of lemma 4.1 with respect to the previous coloring, and apply the simple greedy algorithm.

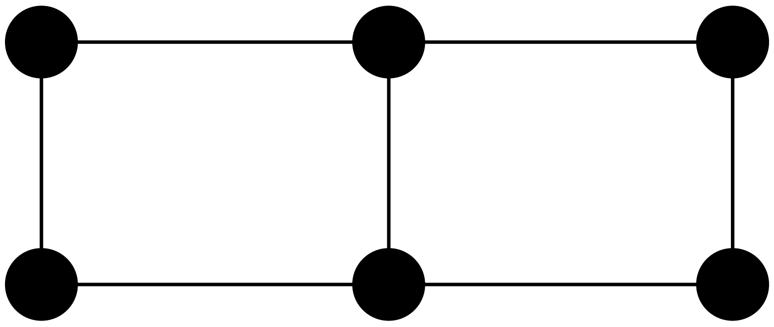

Figure 1: Simple Iterated Greedy Example Graph

The following example shows that rearranging the color classes can improve the coloring number. Applying the greedy algorithm to the permutation “abcdef” of the vertices in figure 1 yields the coloring $a = 1$, $b = 1$, $c = 2$, $d = 2$, $e = 3$, $f = 4$. The permutation “fecdab” reverses the coloring class order, and applying the greedy coloring algorithm to it produces an optimal 2-coloring.

4.1 Reordering Heuristics

To clarify the following discussion on heuristics we refer to figure 2. Here the oval shapes represent the vertices of each of three color classes generated by the previous application of the greedy algorithm. Three different orders are

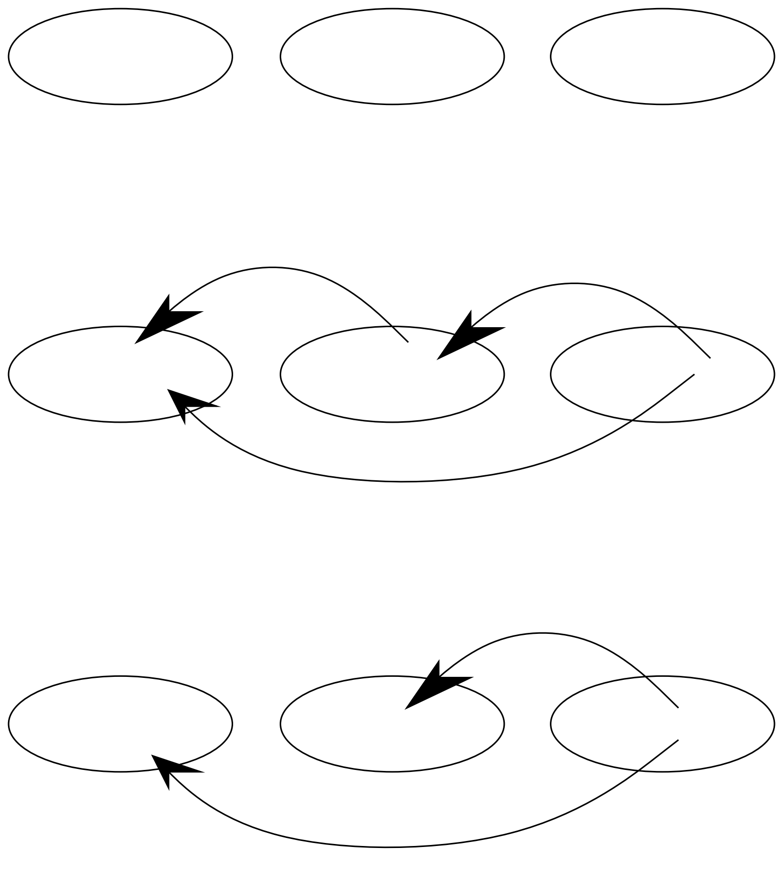

Figure 2: The Effects of Different Orders of the Color Classes

represented. On the next application of Greedy, the vertices in the leftmost set will be colored with color 1, and the rest with colors 2 and 3. The arrows represent the possibilities of certain vertices being assigned a lower color than the other vertices in the same set. We can think of this as moving the vertices from one set to another. Notice that there are more paths for movement in the order $3, 2, 1$ than for the other orders. However, this does not mean it is necessarily the best order. For example, the order $2, 3, 1$ could be better because if the $3, 2, 1$ order is used, moving some vertex from 2 to 3 could block several vertices from moving from 1 to 3. Of course the order $1, 2, 3$ can produce no improvement.

We must restrict ourselves to permutations which rearrange the color classes. Many heuristics could be used. We consider the following where the ordering refers to the color classes generated by a previous iteration of the greedy algorithm:

Reverse order This intuitively should be useful since it gives the maximum opportunity for mixing classes. If $j < k$ and class j precedes class k in a permutation, then no vertex from k can move to class j. This is because every vertex in class k must conflict with some vertex in class j, or else the preceding application of the greedy algorithm would have assigned it the smaller color. However, vertices of class j are not necessarily all in conflict with vertices of class k, and so reversing the two classes could result in some vertices being moved to a different group. This heuristic should maximally enhance the mixing of groups.

Increasing size Intuitively, smaller classes, that is ones with fewer vertices, should have fewer adjacent vertices in total, and so listing them first should provide opportunity for vertices to move into new groupings.

Decreasing size We observed that one of the techniques for coloring is to find large independent sets. Putting the largest independent sets first increases their opportunity for further growth.

Increasing Degree Again we try to enhance mixing, similar to the idea of smallest classes first, placing the color classes with the smallest total degree first.

Decreasing Degree Ordering the classes by decreasing total degree (although similar to the Welsh and Powell suggestion) does not seem to be a good idea, and is listed for reasons of completeness.

Random order This is useful to prevent or break cycles that can arise when other methods are used exclusively.

Although these strategies were tested individually we find that the best strategy is one that mixes the above heuristics. Two types of mixing are used. First, sorting the classes by size is greatly enhanced by reverse ordering the classes of the same size. Whenever we refer to sorting by size (or degree) we assume reverse order on equality unless otherwise stated. Second switching between heuristics on different iterations breaks up cyclic patterns in the coloring process. Clearly it is not possible to have both increasing and decreasing size as heuristics on any one iteration. Decreasing size is generally quicker than increasing size, but has a much stronger tendency to stagnate. This is not unexpected since the largest classes will usually still be the largest after recoloring. However, even using a strategy of pure reversal is less effective than one that occasionally uses a random reordering.

The heuristics based on degree sequence do not seem to be useful. One quite effective combination for the graphs in the classes we are interested in is four or five iterations of decreasing size, followed by one or two of increasing size or reverse order, and occasionally inserting a random reordering step. Another is to allow random selection between various heuristics, with the user specifying the relative frequencies. This last method provides the most flexibility. We found that for $\mathcal{G}_{1000,\frac{1}{2},k}$, very good frequencies were decreasing 70, reverse 50, increasing 10 and random 30.

The program repeats the basic reordering and recolorings step until a specified number of iterations have occurred since the last improvement, or until a target coloring number has been achieved. In the next section we discuss the measurements used to determine when a coloring is an improvement.

One last detail should be mentioned. If, using the measure of progress defined in the next section, the program fails to make an improvement after some number of iterations, then the best result so far is reintroduced. A new sequence of reorderings is attempted from that point. The program allows the user to specify the number of iterations to wait before retrying, and a maximum number of retries to any given point. If the maximum number of retries is exceeded the process continues on indefinitely along the last line of search.

4.2 Measures of Progress

It is easy to see that progress has been made whenever the coloring number is reduced. However, it may be that the program takes hundreds or thousands of iterations on larger graphs between changes in the coloring number. As a more refined measure of progress, we take the sum of assigned colors. This is the sum obtained by adding together all of the colors assigned to the vertices. As the vertices form larger independent sets, one would hope that we are making progress towards reducing this sum. Of course this is not strictly true, since a reduction in coloring number could (and frequently does) correspond to an increase in the sum of colors. Nevertheless, experiments show that this is a reasonable measure of progress, and so it is the measure used to determine when progress has ceased. A fudge factor of current coloring number times the number of vertices is added to the sum. This makes it more probable that a reduction in colors used reduces the measurement.

In the following paragraphs, we show that the smallest sum on a χ-coloring can be arbitrarily larger than the minimal possible sum, and that the coloring number associated with the smallest coloring sum can be arbitrarily larger than the chromatic number.

In figure 3 an example is given to show that the smallest coloring sum does not necessarily correspond to the smallest coloring number. If all of the leaves are given color 1, then three colors are required, and the minimal coloring sum is 11. However, the graph can be 2-colored, but the corresponding sum will be at least 12. By using more leaves, the sum difference can be made arbitrarily large.

Figure 4 shows a 2-colorable graph in which the minimum sum coloring must have 4 colors. To minimize the sum, color the black vertices with color 1, the grey nodes with color 2 and the remaining two nodes with colors 3 and 4. To reduce the coloring number by one, the node colored 4 would have to be colored with color 1 or 2. If 1 then the four adjacent black nodes would have to increase their coloring number, which increases the total. If 2, then either the three adjacent grey nodes would have to increase their coloring, increasing the total, or decrease by one with corresponding increases in the six black nodes at the bottom. Reducing the coloring to a two coloring would increase the sum even further, by a similar argument.

The graphs in figure 3 and figure 4 are the first members of a sequence in which the kth member requires k colors to achieve a minimum sum in a 2-colorable graph. The kth member consists of two identical trees, connected

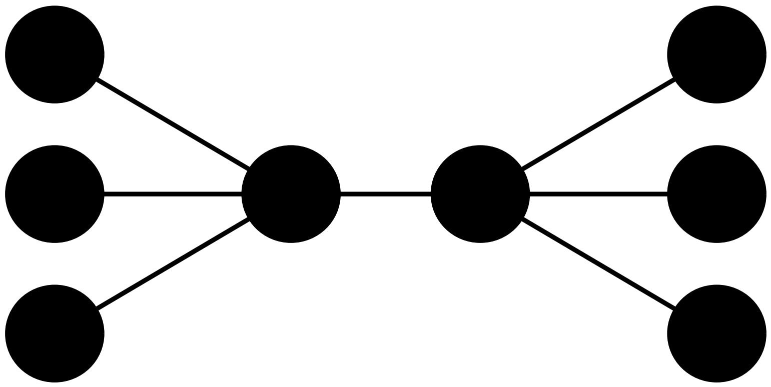

Figure 3: Smallest Coloring Sum Requires 3 Colors

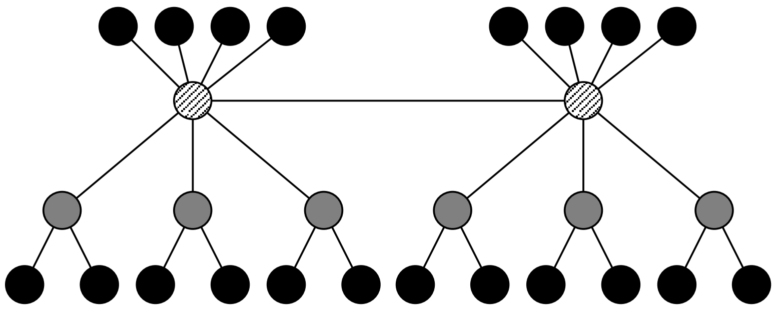

Figure 4: Smallest Coloring Sum Requires 4 Colors

at the root. Each root is connected to k leaves, $k-1$ 2-trees, $k-2$ 3-trees, and so on up to 3 $k-2$-trees. An i-tree consists of a root connected to i leaves, $i-1$ 2-trees and so on. The construction of the 5th member of the sequence is illustrated in figure 5.

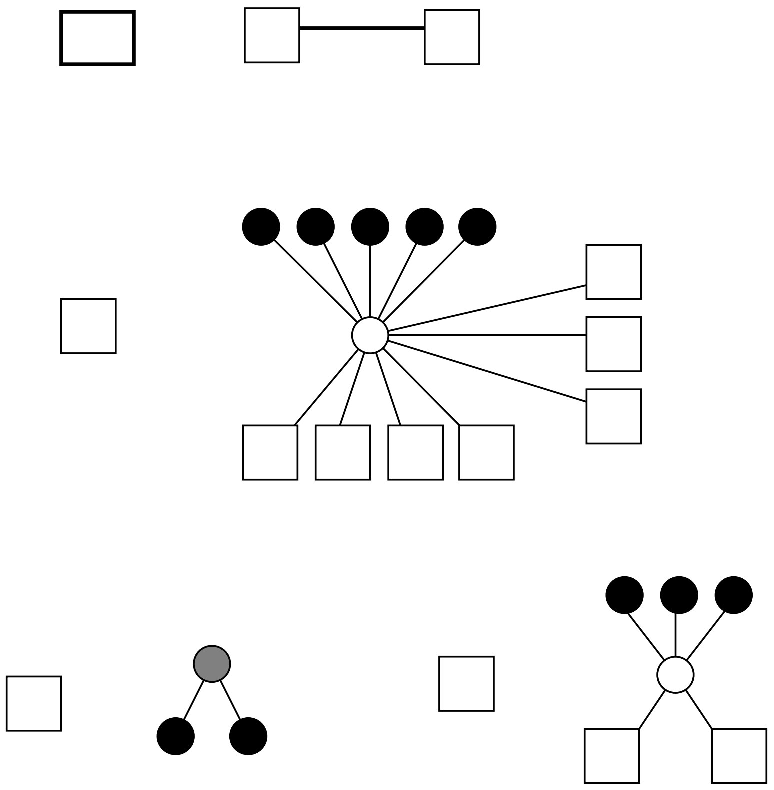

Figure 5: Construction of the $k = 5$ Case

The number of vertices in the i-trees satisfy

$$\begin{aligned} T_1 &= 1 \\ T_2 &= 3 \\ T_i &= 1 + \sum_{j=1}^{i-1}(k+1-j)T_j \\ &= 4T_{i-1} - 2T_{i-2}, i \geq 3 \end{aligned}$$

and thus

$$T_i = \frac{a^i + b^i}{4}, a = 2 + \sqrt{2}, b = 2 - \sqrt{2}$$

The number of vertices in the kth member of the sequence is

$$\begin{aligned} N_k &= 2\left(1 + \sum_{i=1}^{k-2}(k+1-i)T_i\right) \\ &= 2\left(T_k - 2T_{k-1}\right) \\ &= \frac{a^{k-1} - b^{k-1}}{\sqrt{2}}, k \geq 3 \end{aligned}$$

4.3 Anti-IG graphs

IG is not guaranteed to find an optimal coloring. In fact, in the worst case it can be as bad as a single iteration of the greedy algorithm. The almost complete bipartite graph defined in [16] also foils IG. This graph is defined by $G = (V, E)$ where $V = \{a_1 \ldots a_n\} \cup \{b_1 \ldots b_n\}$ and $E = \{(a_i, b_j) | i \neq j\}$. In the worst case the greedy algorithm assigns the same color to a_i and b_i, requiring n colors when 2 would suffice. If this happens then no reordering of the independent sets will lead to an improved coloring.[1]

The probability of producing an n-coloring on this graph is very small for even moderate values of n. Suppose a_i is chosen on the first step. Then a b must be chosen on the second step to produce an n-coloring. If b_i is chosen on the second step with probability of $1/(2n-1)$, then we may choose any vertex for the third step. Otherwise we must choose b_i on the third step with probability of $\frac{(n-1)}{(2n-1)}\frac{1}{2n-2} = \frac{1}{2(2n-1)}$. In either case we have reduced the problem to the second step on $n-1$ pairs, and keeping in mind that on the last vertex there is no choice, we can repeat this to compute the probability

$$\begin{aligned} P = P(n - \text{coloring}) &= \prod_{i=1}^{n-1} \frac{3}{2(2n-2i+1)} \\ &= \left(\frac{3}{2}\right)^{n-1} \prod_{i=1}^{n} \frac{1}{2n-2i+1} \end{aligned}$$

[1]If we start with an application of Brélaz for the initial coloring, then this graph will be two colored as Brélaz guarantees optimal coloring of bipartite graphs.

$$= \quad 2\frac{3^{n-1}n!}{(2n)!}$$

If any other coloring is produced, then the first iteration in which a color class with two or more vertices of one type (a or b) is first in the permutation will color the graph correctly. (If two vertices from one side have the same color, then no vertex from the other side can have that color). We can produce a graph in which the probability of an n-coloring is arbitrarily high using IG by creating enough components of this type, that is by fixing $\delta > 0$ and choosing a repetition number r large enough so that

$$(1 - P)^r < \delta$$

5 Experiments Using IG

With the many combinations of heuristics possible, many tests were done to determine the most advantageous ones. In general mixed strategies seemed to perform best. Even apparently minor changes in control or frequency of use of different heuristics could cause changes in absolute efficiency on particular graphs, but mixes using about forty to fifty percent reverse ordering, with the remainder decreasing size (with reverse ordering on equality) and occasional random reorderings were robust over most graphs.

We first state the results using some of the best mixed heuristics, and then summarize briefly the results obtained using each of the heuristics on its own.

5.1 Mixed Heuristics

In figure 6 we display typical behavior for coloring graphs from the classes $\mathcal{G}_{1000,\frac{1}{2},k}$ for k from 20 to 60 in steps of 5. Each curve represents the drop in coloring number as the number of iterations increases. For each class, ten graphs were generated and colored, and ranked according to the total number of iterations required to reach the specified coloring. The graph of rank five in each class was then chosen to represent the class in the diagram. (For the first few values of k, the plots are nearly coincident.)

Initially, the coloring number drops rapidly (from a range of 120–127 down to about 102 for $k = 60$), then there is a long period of slow decline (to 90) and finally a rapid drop to the specified coloring. The period of slow decline we refer to as the plateau (of difficulty).

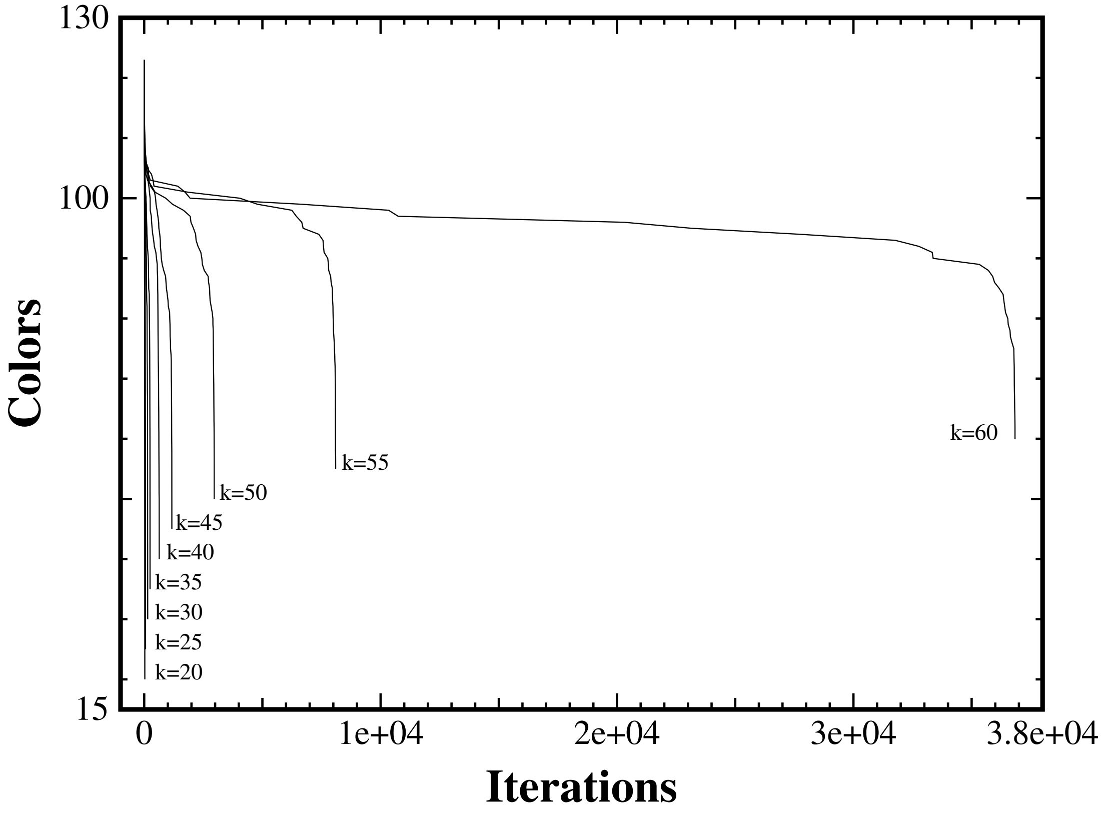

Figure 6: Iterations Required to k-color a graph in $\mathcal{G}_{1000,\frac{1}{2},k}$, $20 \leq k \leq 60$

In table 1, we present the minimum, maximum and average number of iterations to complete the coloring for the ten graphs in each class. Not only does the required number of iterations increase significantly with k, but also the variation increases, the maximum iterations being approximately double the minimum in each case. The cpu time required for the longest run in class $\mathcal{G}_{1000,\frac{1}{2},60}$ was 7262 seconds. This gives an average time of 0.159 seconds per iteration for 1000 vertices. The time per iteration varies slightly, decreasing with decreasing coloring number, but since nearly all of the time is spent in the 100 to 90 range, this average is a good estimate for all of the longer runs. (The time per iteration is greatly reduced for smaller graphs, of course).

Color	Min	Max	Avg
20	20	45	30.5
25	48	89	62.6
30	91	187	140.2
35	191	341	267.2
40	549	945	683.6
45	846	1664	1275.5
50	2188	4549	3160.5
55	5973	13313	8793.5
60	23375	45453	36277.8

Table 1: Iterations to Achieve a Specified Coloring for $\mathcal{G}_{1000,\frac{1}{2},k}$

Table 2 shows the results of applying IG to graphs in the class $\mathcal{G}_{1000,\frac{1}{2}}$. Many more tests were done on this class, with similar results for obtaining colorings down to about 105. These test results are for five graphs using a mixed strategy that seems to be among the best. Only five tests were done due to the time required. The purpose of this test was to explore the limits of IG. The program was set to run for 40,000 iterations after the last improvement, where an improvement is a reduction in the coloring sum as discussed in a previous section.

The table shows the initial coloring number used by greedy, the coloring after one and two iterations, and then the number of iterations to reach colorings in the range from 110 down to 99. Only one graph reached 99 and none did better.

We note that IG is fairly effective down to colorings of about 107 to 105,

	Graph 1	Graph 2	Graph 3	Graph 4	Graph 5
Iterations	Colors used				
It =0	125	128	125	127	128
It =1	123	123	122	122	124
It =2	122	119	119	120	121
Color	Iterations To Reach Color				
110	46	55	44	56	28
109	66	56	68	75	55
108	98	79	108	162	79
107	123	195	365	260	81
106	214	368	620	697	220
105	245	666	708	984	475
104	685	3076	2739	2011	3706
103	1254	4849	5154	2403	5103
102	11054	13072	7040	3442	6027
101	15301	22110	16774	6356	10895
100	19516	24387	37727	24667	56766
99		93053			

Table 2: Iterated Greedy on $\mathcal{G}_{1000,\frac{1}{2}}$

but then slows down significantly. It is not an effective means of reducing the coloring below 100.

We can use another algorithm to generate a first coloring of a graph, and then use the IG approach starting with that coloring. The DSATUR algorithm of Brélaz [5] is one that we have tested.[2] This algorithm colors a vertex, then selects the next vertex to be colored from the set which has the largest number of different colors assigned to its neighbors. The minimum possible color is assigned to the vertices when colored.

For the same graphs in $\mathcal{G}_{1000,\frac{1}{2},55}$ as in table 1, we started the search with one application of our modified Brélaz. While greedy gave initial colorings ranging from 122 to 127, the Brélaz algorithm gave colorings ranging from 113 to 115. (These ranges are also typical of graphs in $\mathcal{G}_{1000,\frac{1}{2}}$) It took from 3 to 8 iterations of IG to reach 115 and from 7 to 14 iterations to reach 113. These latter values represent 1.77 to 2.65 seconds. However, we should keep in mind that Brélaz requires more time than greedy because of the increase in complexity. Perhaps a better comparison is obtained by noticing that both approaches required similar times to reach a 112 coloring. To complete the 55 coloring required from 6198 to 15294 iterations with an average of 9267.2. This average is 5% more than when the coloring process started with greedy. On the other hand, six of the ten graphs were faster using Brélaz. It does not seem possible to draw any firm conclusions on the merits of starting with Brélaz.

Similar comments hold when Brélaz is used to initialize the search on graphs in $\mathcal{G}_{1000,\frac{1}{2}}$.

5.2 Effectiveness of Individual Heuristics

In this section, we report results in which individual heuristics were used alone. These results are not in general nearly as good as the mixed heuristics previously reported, but may have scientific interest.

In a previous section, we demonstrated graphs on which it was possible that an initial coloring could not be improved by IG using any reordering heuristic which preserves coloring classes as adjacent sets. Clearly, there will be more graphs which thwart optimal colorings if the possible reorderings are restricted. When we restrict the heuristics to one of those which are

[2]We should point out that our version is slightly modified from that given in the reference. In particular, for these tests we did not select by decreasing degree. Earlier tests showed that this did not alter the results by more than one or two colors usually. The modifications were made while attempting to use Brélaz in an iterative mode.

deterministic, such as smallest last, or reverse order, this problem can be quite noticeable, even on graphs in $\mathcal{G}_{n,p,k}$ for small k. We did several test on graphs in the class $\mathcal{G}_{1000,\frac{1}{2},40}$ as an illustration. Initial greedy colorings for this class are from 121 to 125 on the graphs tested.

Reverse ordering as stated should yield the maximum opportunity for recoloring vertices. However, in a test on 6 graphs, it never managed to obtain a 40 coloring. The colors obtained were $45, 47, 56, 58, 60, 72$. These colors were obtained quite quickly, ranging from 354 to 616 iterations. Then the process apparently began to cycle, as the coloring sum would become fixed or cycle over one or two values. This occurred after the process penetrated the plateau exhibited in figure 6, and had reached the region where coloring drops rapidly. In fact, this rapid drop was evident in each case right up until the stagnation occurred.

For decreasing size, progress usually terminates after only 4 or 5 iterations, with further iterations repeating one coloring sum. Little improvement is made over the initial coloring, ranging from zero to three colors. If reverse order is used for those sets of equal size, instead of the previous order, then the coloring was reduced further, from 113 to 118 in the three graphs tested, although progress terminated after 10 iterations (in a sample of three graphs).

Using the increasing size heuristic, colorings from 112 to 113 were obtained, but considerably more time was required. In this case the tests were terminated before there was evidence of repetition of the coloring sum. This heuristic just does not seem to progress well. When classes of equal size are placed in reverse order, with smallest coming first, then the coloring was reduced to 108 to 110. Again, the coloring sum did not show evidence of repetition, but no progress was achieved in reasonable time.

Smallest total degree first behaved similar to smallest size first, except that it did better when sets of equal degree were kept in the same order rather than reversing them. The tests only used three graphs (the same three as in the smallest first test).

Largest degree first did not perform as well as smallest degree first, obtaining colorings of from 114 to 116 in 10 to 12 iterations. In this case, reversing the order on sets of equal degree did improve the performance in one case, while worsening it in the other two. More tests need to be performed in these cases, as little experimental work has been done with these two heuristics.

Using random reordering, the specified coloring can usually be achieved, although it takes 2000 or more iterations.

From these limited results one might conclude that the reverse ordering heuristic is the best, but should be mixed with random reorderings occasionally to prevent stagnation. Using a random reordering every 50 iterations resulted in the specified coloring being found on each of 10 graphs, with the number of iterations ranging from 417 to 716 with an average of 552.6. This compares well with the results in table 1.

However, for larger k it is worthwhile applying iterations of largest size first with equal sizes being reverse ordered. A number of different strategies for mixing heuristics have been tried. Surprisingly, one of the best, and the one used to develop most of the 300 vertex landscape in part IV, was to use 5 iterations with largest first, followed by one of smallest first, with every 11th iteration being a random shuffle. On a ten graph sample from $\mathcal{G}_{1000,\frac{1}{2},55}$ we have been unable to improve on the results of this mix even after dozens of attempts. The differences are not large, however, and a lot could depend on the luck of the draw.

It should be pointed out that even with mixed heuristics, on very rare occasions the process will hang up just short of the specified coloring. Rerunning with even a minor change in the mix fixes this. In the next part we introduce mixed algorithms, which also correct this problem.

5.3 Notes on the behavior of IG

Turner [27] proved that for a fixed k, almost all k-colorable graphs are easy to color. The algorithm he used is a refinement of Brélaz's DSATUR algorithm. It operates in two stages, the first trying to grow a clique of size k, and the second selecting vertices which have exactly one of the k colors available to them. He uses this improbable sounding algorithm to prove that as n goes to infinity, almost all k-colorable graphs (for a fixed k) will be optimally colored.

Simulations and analysis both suggest that the point at which the algorithm fails 50% of the time grows logarithmically in n, with the failure at $k = 9$ for $n = 1024$. He finds experimentally that DSATUR has similar capabilities.

On the other hand, Turner shows that the greedy algorithm has low probability of finding a k-coloring, even if k is as small as four. In fact, " the greedy algorithm can be expected to produce colorings that differ from optimal by an arbitrarily large factor."

Kučera [19] improves Turner's result in the asymptotic sense by showing that for $k \leq \sqrt{cn/\log n}$, a k-colorable graph is easy to color using an algorithm he presents. The c he proposes can be as large as 1/196, which for a graph of 1000 nodes gives a k of 1! However, he does not implement or test his algorithm.

In the light of the above negative comments on greedy, it is interesting that iterated greedy performs so well. Here we try to improve our intuitive understanding of the algorithm.

Discussion of IG in previous sections shows *how it could work*, but not *why it should work* on k-colorable graphs. The drift through the plateau takes a very long time. Let us suppose for now that it is just a random search process, except that the coloring number is not allowed to increase. How do we explain the rapid drop below the plateau?

The basic observation is that once a small enough coloring has been discovered, then with high probability there *must* be some sets in the coloring that are subsets of some of the specified sets. For example, suppose we are coloring a graph in $\mathcal{G}_{1000,\frac{1}{2},50}$ and currently have achieved a 60-coloring. We can assume that the largest independent sets in the 60-coloring have at most 20 vertices. If so, these are almost surely the specified sets, because the probability of getting any vertex non-adjacent to all 20 vertices in a specified set is 2^{-20}. [3] In fact, it is well known [20] that the largest independent set in a graph in $\mathcal{G}_{1000,\frac{1}{2}}$ is likely 15, and most sets will be of size 14 or less. For our example, a 60-coloring means there must be some sets of size 17, which will likely be subsets (or very nearly so) of the specified sets. If these sets are placed near the beginning of the order (by the largest first heuristic) then they will almost surely achieve their maximum size of 20. If they are placed near the end (by the reverse order heuristic) then either they retain their size, or other sets of similar size must be formed.

In all cases, the coloring sum will be reduced with high probability, and so the coloring number is forced down. The remaining subgraph, not yet correctly colored, is reduced in size, and so is easier to color. Even when the coloring is large enough to be near the plateau, there is a chance that some set will be a subset of one of the specified sets, and when such a set moves to the front of the ordering, it has a good chance of collecting many of the vertices of the specified set containing it.

Notice how this analysis fails as k goes to the expected coloring number

[3]Given the combinations of sets and vertices available, there is a small probability of a set of size 20 not being a specified one, but for purposes of our intuitive analysis, we ignore this.

of a random graph. In this case, the specified sets have no advantage over the huge number of other maximal independent sets in the graph.

This analysis leaves much to be desired, and is only an initial step to understanding why this algorithm should work. Other considerations have to be taken into account to explain the rapid drop through the plateau when k is small enough. For example, it seems likely that when k is small (say $k = 3$) then out of the set of independent sets on any coloring, there is high probability that some of them will be subsets of one of the large specified sets, or nearly so. Once such a set is the first in the ordering, it will collect most of the remaining vertices in the set. This leaves only two sets in the remaining subgraph to be colored. Notice that if we increase p while keeping k fixed, this argument implies that the coloring becomes still easier. This in fact has been observed while exploring the 300 vertex landscape discussed in Part 4.

Finally, these arguments also yield an intuition as to why increasing the variance in the coloring seems to aid the algorithm. The larger the largest sets are, the easier they are to identify, because the probability will be higher that some independent set is a subset of one of them.

Part III
Combining IG with Other Algorithms

The pure iterated greedy algorithm, although sometimes successful, still does not perform well on $\mathcal{G}_{1000,\frac{1}{2}}$. Hertz and de Werra [15] describe the application of a technique called Tabu search attributed to Glover[12]. Using this technique they were able to color graphs in $\mathcal{G}_{1000,\frac{1}{2}}$ with 93 colors. Using a more sophisticated technique, in which they first found large independent sets using Tabu and then applied Tabu coloring to the remaining vertices, they obtained at least one 87 coloring.

In this section we describe the combination of IG with Tabu, and show where improvements can be made. We discover that in different classes, different mixes are required for effective performance.

6 Tabu Algorithm

The Tabu algorithm in one sense is dual to the greedy algorithm. The greedy algorithm generates independent sets by adding vertices one at a time so that no conflicts are created. Tabu starts with a partition of the vertices, with vertices assigned at random, and then tries to remove conflicts by re-assigning vertices.

To be more precise, suppose at step t we have a partition $s_t = (V_1, \ldots, V_k)$. We call s_t a *point* in the search space. We may generate a new point (referred to as a *neighbor* of s_t) by selecting at random any vertex in conflict, and moving it at random to any other partition element.

Tabu works by generating a set of such neighbors, and then selecting the neighbor with fewest conflicts to be the new partition at step s_{t+1}. To prevent cycling through a small set of good but suboptimal points, a list of the last q moves is kept, and the reverse of these moves are not allowed. As each new move is generated, the oldest move is removed from the Tabu list, and the newest move is added.

Note that the number of conflicts could refer to either the number of edges or the number of vertices, or some function of the two. The choice will affect the nature of the subgraphs of vertices that remain in conflict as the algorithm progresses. Minimizing vertices will generally tend to result in dense subgraphs, while minimizing edges allows for sparser subgraphs. Hertz and de Werra use the number of edges, but do not discuss the trade-offs. In limited tests, this seemed to be as good as any other measure we found.

They do suggest a number of improvements to their algorithm. One is the use of an aspiration function, another is the immediate acceptance of a neighbor if its conflict value is less than the current value. They also use a type of brute force search when the number of conflicts is small. We do not use aspiration, require at least a minimum number of neighbors be generated even if a better solution has been found, and use a type of brute force search when the number of conflicts is small.

Parameters to our Tabu are the number of neighbors to be generated, the minimum number of neighbors and the size of the Tabu list. Following Hertz and de Werra, we use a Tabu list of size 7. For 1000 vertex graphs we usually generate 600 neighbors, while for 300 vertices we generate only 200. The minimum number of neighbors is usually set between 2 and 10. The idea behind this last is that if it is very easy to generate an improved neighbor, then we are likely at a point where there are many improvements. Perhaps one could say we are not believers in lucky breaks!

Using these techniques, we were able to get results similar to those reported in [15] for colorings down to the mid 90's on $\mathcal{G}_{1000,\frac{1}{2}}$. However, we had a further purpose in mind, to be developed in the next section.

7 Combining Tabu and IG

One of the problems inherent in the Tabu design is its all or nothing character, at least if the goal is to find a proper approximate coloring. Tabu starts with a partition of the desired size, and attempts to remove conflicts until there are none, or terminates because it has exceeded some upper limit on its search time. Of course, it will always give an approximately correct solution, which may be useful if the goal is to color vertices with the given number of colors while minimizing the number of conflicts.

We can build a composite algorithm as follows. First, we start with some valid k-coloring of the graph. We then set $k' = k - c$ for some small value c, sort the vertices using the current coloring by decreasing size of independent sets, use the first k' sets to seed the partition, and then distribute the remaining vertices (from the c smallest sets of the current coloring) over these sets. In our implementation, each vertex is tested in turn to see in which independent set it creates the fewest conflicts, and is added to that set. In this way we hope to give Tabu a boost over a random partition as an initial step.

Second, if Tabu fails to find a correct coloring in the prescribed time, then the partition number is incremented by one, and the process continued. This means that the newest partition element is initially empty, but vertices in conflict can be moved into it. An untried improvement might be to move some portion of the conflict vertices into this new element immediately.

Third, after repeating the second step c times, which means the partition is of the same size as the current coloring, we can use IG to rearrange the current best coloring and try again.

Our over all process can now be described:

1. Generate an initial coloring using some algorithm, for example Brélaz or greedy.

2. Perform some number of IG steps.

3. Apply Tabu as described above.

4. Repeat steps 2 and 3 until the coloring criteria are satisfied or time limits are exceeded.

In step 3, we use the best result found by IG (the coloring with the smallest coloring sum found so far) as the initializer. If Tabu fails to find a reduced coloring, then we allow Tabu to attempt to find a new coloring of the same size, and if it does we use that as the new coloring to initialize the next IG step. In this way, we have a chance of getting out of valleys that can trap IG.

In case of failure, the grand loop is terminated by a fixed upper limit on the number of iterations, usually from 2 to 10. IG and Tabu are terminated after some upper limit of iterations have occurred since the last improvement. For Tabu, an improvement is a reduction in the least number of conflicts seen so far. For IG, an improvement is a reduction in the sum of colors.

8 Experimental Results using IG/Tabu

In this section we report on results using the combined IG and Tabu algorithms, and on Tabu alone. In figure 6 and table 1 we showed how IG performed on the class $\mathcal{G}_{1000,\frac{1}{2},k}$ for various k. One of the features we noticed was that after an initial rapid decline, there was an increasingly long plateau where progress was very slow. Then there was a rapid drop again after the coloring dropped through the plateau.

In figure 7 we show the rate at which conflicts are removed using Tabu for graphs in classes $\mathcal{G}_{1000,\frac{1}{2},k}$, for $20 \le k \le 40$.[4] Figure 7 demonstrates that the number of iterations required for Tabu increase in a manner similar to that for IG. In particular, there seems to be a plateau, and after it is penetrated there is a rapid drop in the number of conflicts. As before, we choose a median from each class for inclusion in the figure, where for $k = 40$ it is the median of three runs. Notice that as the value of k increases, the initial number of conflicts decreases as we would expect.

In table 3, we list the minimum, maximum and average iterations required by Tabu to color graphs in $\mathcal{G}_{1000,\frac{1}{2},k}$ for k from 20 to 55. Because the coloring times are so long, we have limited our experiments to 10 graphs each for $20 \le k \le 30$, 3 for $k = 40$ and 2 on $k = 55$. It is obvious, even allowing for inefficiencies in our code, that this is not the right method for coloring graphs in these classes. We note here that our program uses cpu time per iteration very similar to that in [15], but we do not have a good comparison of the processor speeds, and suspect our code could be improved. The longer run on $\mathcal{G}_{1000,\frac{1}{2},55}$ required 22211 seconds on the SUN IPC. Extrapolating from the last line of table 2 in [15], theirs would use 21929 seconds for 702889 iterations.

k	Min	Max	Avg
20	1118	1327	1223.4
25	1359	2738	1777.3
30	2064	3201	2642.2
35	3317	11235	6898.2
40	30606	40481	34086.3
55	634639	702889	668764.0

Table 3: Iterations Required by Tabu on $\mathcal{G}_{1000,\frac{1}{2},k}$

[4]We ran tests on $k = 55$ also, but did not include them in the figure because the runs are so long they distort the graph too much to see the lesser runs.

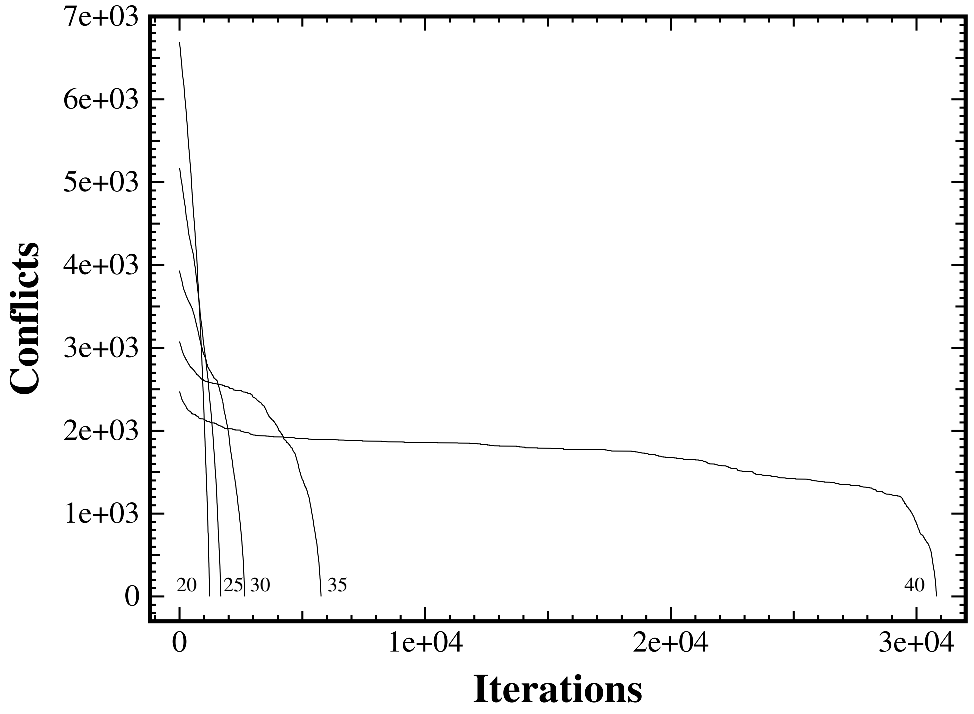

Figure 7: Iterations Used by Tabu by $\mathcal{G}_{1000,\frac{1}{2},k}, 20 \leq k \leq 40$

For graphs in $\mathcal{G}_{1000,\frac{1}{2}}$, we found it best to let IG work for the first few color reductions, down to about 110 colors, then to increasingly rely on Tabu. This means we should vary the number of iterations before switching according to how far the program has progressed. To this end, our program allows the input of a sequence of change over characteristics. Specifically, the user enters a table of control values. Each row of the table consists of four values. The first value is a coloring number, such that the remaining values are in effect when the current coloring is as large as this value, but smaller than the next larger coloring number. The other numbers are: a step number, indicating how many colors to reduce the current coloring by when switching from IG to Tabu; and iteration limits for IG and Tabu, indicating to each how long to continue a search before giving up. These greatly increase the number of parameter combinations available for testing.

Table 4 lists the CPU times used in one experiment on ten graphs in $\mathcal{G}_{1000,\frac{1}{2}}$.[5] A fixed set of parameter settings was used for the entire experiment. These runs were on a SUN 380 which runs slower than the IPC. Dividing these times by 2.75 to get times comparable to the IPC shows that these searches for 93-colorings range from 1407 to 5028 seconds, which compares with the average of 8719 reported by Hertz and De Werra[15]. This seems to reflect favorably on the mixed algorithm, given that an iteration of our implementation of Tabu requires almost the same time as theirs. However, they report only a sample of 2 graphs and as table 4 illustrates, variability is high. Also, we used 500 instead of 600 neighbors in our search step.

In another test on an IPC a 92-coloring was found in 1489 seconds, while yet another reached 93 in 740 seconds but was then unable to reduce the coloring further even after a long time. It appears that luck plays a large part in performance of these algorithms. A more careful series of experiments should perhaps be done, but results in the next section would seem to supersede anything that is likely to be obtained from this approach on graphs in $\mathcal{G}_{1000,\frac{1}{2}}$.

At this point one wonders whether using a mixed approach might improve the coloring times on $\mathcal{G}_{n,p,k}$ for large n and k. Our initial results were promising; the time to reach colorings in the 96 to 99 range on $\mathcal{G}_{1000,\frac{1}{2},55}$ was reduced by a factor of nearly 10. Note these values are in the middle of the plateau in figure 6. However, the total time required to find the 55-coloring

[5]Starred entries indicate a 99-coloring was not produced; the times are for the next smaller coloring.

Graph	99	94	93	92	91
1	259	3880	4527	18487	–
2	251*	2554	6462	–	–
3	209	4993	6967	33725	–
4	336*	2523	3870	24228	–
5	337*	3258	6487	13471	23749
6	588*	3950	11797	25391	–
7	240	3523	4167	10221	13469
8	373	5349	11980	–	–
9	214	7305	13827	34117	–
10	220	6818	10621	19892	–

Table 4: Mixed Algorithm CPU Times on a SUN 380

increased dramatically. One might suspect the long times used by Tabu as indicated in table 3 are at fault, and so reverting to pure IG after reaching 96 would give us the benefits of both approaches; a faster initial reduction to the middle of the plateau, and then the faster approach of IG to finish.

This turns out **not** to be the case. In one typical test of three graphs, IG required from 12435 to 19319 iterations to reduce the coloring to 55 after Tabu had helped it reach the mid-nineties. The total CPU time ranged from 2121 to 3247 seconds. Notice that graphs in this class required from 5973 to 13313 *total* iterations when IG was used alone (table 1). It took more iterations just to get off the plateau after being "helped" by Tabu than the total used by IG working alone!

Other tests seemed to indicate that no amount of Tabu is useful in aiding IG on these graphs, unless k is very close to the expected coloring of a random graph. Various parameter mixes were tried without success. We suspect that applying Tabu to get a smaller coloring leaves the structure of the independent sets in a different form that hinders IG on these graphs. Perhaps Tabu tends to equalize the set sizes, while IG tends to build some larger sets making the smaller ones easier to eliminate. A deeper understanding of this phenomenon could be useful in designing better coloring algorithms.

There is an exception to this negative result. Very rarely, IG will get within a few colors of the specified coloring, and then fail to make any more progress. Apparently IG is locked in a way similar to the Anti-IG graphs of section 4.3. In this case, Tabu will very quickly find a reduction of one or two colors, and since we are far below the plateau of difficulty in these cases, IG can then quickly complete the coloring.

In a subsequent section we do a comprehensive set of experiments on 300 vertices, letting k increase arbitrarily with various edge probabilities. When k is sufficiently large, IG is unable to find the specified coloring in a reasonable time. Here Tabu is helpful in reducing the coloring somewhat, although it too often fails to find the specified coloring. Sometimes when we are very near the ridge on 300 vertex graphs, Tabu is able to reduce the coloring enough to allow IG to complete the coloring.

One wonders if we were to insist on the optimal coloring on random graphs, whether in fact Tabu or IG would be superior, and whether a mix would be an improvement. Unfortunately, time considerations make this a mute question on large graphs.

9 The Maximum Independent Set Approach

Johri and Matula[18] (see also Manvel[20]) and Bollobás and Thomason[4] developed extremely greedy coloring algorithms. These algorithms attempt to recursively find a maximal independent set and assign a color to its vertices. Of course, finding maximum independent sets is also NP-hard, requiring exponential time for the best algorithms known. Bollobás and Thomason used a backtrack approach with heuristic pruning that gave large if not maximal sets. When the uncolored subgraph was reduced to about 200 vertices they switched to a brute force maximum independent set search, and in the range of 35 to 50 vertices they did a brute force coloring. The method of pruning was based on several parameters to limit searches when large independent sets were found. This approach resulted in an average coloring of 86.9 on a set of ten graphs from $\mathcal{G}_{1000,\frac{1}{2}}$.

The approach in [18, 20] is similar and resulted in average colorings of 95.9.

We implemented a similar approach which we call MAXIS, but simplified the pruning somewhat, asking the user to specify a cutoff number for different ranges of remaining vertices. We introduced some other ideas, such as sorting the vertices by degree in the hopes that the number of remaining vertices would be increased, and thus lead to larger independent sets. We do not switch to brute force coloring for small remaining sets, relying instead on IG and Tabu to clean up after MAXIS. We do retain the idea introduced in [4] that when two or more independent sets of maximal size are found, we retain the one with the largest total degree. This reduces the number of edges in the remaining uncolored subgraph, and so often improves the coloring. Heuristics with similar intent are described in [20].

MAXIS is used to obtain an initial coloring of a graph, and then IG and Tabu are used to try to improve the coloring. Many variations have been tested and more details will be given shortly. On $\mathcal{G}_{1000,\frac{1}{2}}$ one test on 11 graphs gave two 86-colorings, eight 87-colorings and one 88-coloring. The coloring times ranged from 1470 seconds to 3128 seconds on a 22 MIPS SUN ELC[6]. All but the one graph received 87 colorings in under 2000 seconds. The program was allowed to continue the IG/Tabu search for about 3200 seconds total before terminating in each case.

However, a bizarre bit of serendipity played an important part in these results. A program bug caused what was intended to be a sort based on degree to generate instead an essentially random permutation. When the bug was removed, the program did not perform as well! This led to further research into heuristics applicable to this approach, which will be presented here.

In [4] it is pointed out that with high probability the coloring number of graphs in this class is at least 80, and with high probability the maximal independent set approach, even if the maximum independent sets could be found, would yield at best an 85 coloring. Thus, these colorings are quite good.

9.1 MAXIS Details

The heart of the MAXIS program is a routine to find a large independent set in an uncolored subgraph. Once a satisfactory set is found, it is colored, and the independent set routine is applied to the remaining subgraph. No attempt is made within MAXIS to reorganize the independent sets that are found.

The independent set routine is based on a simple backtrack approach similar to the one described in [4]. We describe the algorithm recursively in figure 8. As mentioned previously, NewSet is better than MaxSet if it is either larger, or its total degree in the original graph (i.e. the graph in the top level call to IndSet) is larger. The purpose of this is to reduce

[6]Our tests showed that the SUN ELC was almost exactly 1.35 times as fast as the SUN IPC on this program

```
Function IndSet(Graph) returns Set
    Initialize MaxSet to empty
    While (Graph not empty) do begin
        Select a vertex $v$ and Delete it from Graph
        Subgraph = vertices in Graph not adjacent to $v$
        NewSet = IndSet(Subgraph)
        if (NewSet $\cup \{v\}$) is better than MaxSet
            MaxSet = NewSet $\cup \{v\}$
    end
    Return MaxSet
end.
```

Figure 8: The Independent Set Routine

the number of edges in the uncolored subgraph in order to increase the likelyhood of larger independent sets being found later.

The selection of the next vertex can be done at random, or it can be chosen based on some easily computed feature of the graph, such as the degree sequence. In MAXIS the vertices are sorted by the degree sequence of the subgraph, using one of three different sorting criteria. The sort is either by increasing degree, by decreasing degree, or the mean degree is computed and the sort is by the absolute difference from the mean.

The vertices are sorted by degree sequence within each subgraph. We may specify the sorting to be one of the three described above. The effects of different sorting mixes will be described in the next section. Basically, we have tried schemes in which the vertices are always sorted by increasing degree, or the first vertex or two of the independent set may be selected by maximum or mean degree, with the remaining selected from smallest degree. The switch point is specified as a percentage of the initial number of vertices, below which we use increasing degree sort.

Notice that if we do a complete traversal of the backtrack tree, the order will make no difference in either the size or total degree of the independent set found, or in the amount of backtracking that is required. However, in this case sorting by decreasing degree is more efficient, since the size of the subgraphs to be dealt with will on average be smaller. This observation has been confirmed by experiment on small graphs, but if a complete search is to be made it is probably better to avoid sorting and use instead more efficient structures that are not as effective when sorting is used.

Of course, this algorithm is hopelessly exponential if implemented as described. To reduce the search time, we prune the search by truncating the while loop before the graph becomes empty. There are many possible variations and heuristics. We have tried many in MAXIS, including ones similar to those described in [4]. When sorting by increasing degree sequence, we tried truncating when the degree of the next vertex exceeded a certain value. We tried various percentages of the remaining vertices, and even at one point tried setting parameters that were then used to compute a quadratic curve that specified the percentage of the current subgraph to explore. This variation held some promise.

Finally in order to provide more precise control, we simply take as input to the program a sequence of integer pairs, with the first members taken in decreasing order so they specify a set of intervals. MAXIS determines the interval containing the number of vertices in the current subgraph, and then uses the corresponding second number of the pair as the truncation limit. This hackers-delight allows us to fine tune the truncation.

9.2 MAXIS Experimental Results And Analysis

As mentioned previously, we obtained excellent colorings from an early version of this program. Although we intended to use increasing degree sort throughout this experiment, we discovered that an error had caused the initial degree sequence to be in error, thus causing the initial permutation to be essentially random. When the bug was fixed, the program no longer performed as well.

The idea behind using increasing degree is that if we use low degree vertices then the remaining subgraph will be larger. This should increase the likelyhood of finding a larger independent set. In fact, experiments confirm this.

On the other hand, we are not interested in the largest independent sets per se, but rather in the minimal number required to cover the vertices. To this end, when two or more independent sets of the same size are found, the one with the larger total degree (with respect to the initial subgraph) is selected. The idea is that this leaves a sparser uncolored subgraph, decreasing the expected number of colors required for the remaining vertices. This approach has been described by [4].

It is clear that these two ideas conflict with one another, given that we only do partial searches. If we use larger degree vertices we may build smaller sets, but if we do manage to get large sets, then we are more likely to have a greater total degree. The interaction of selection criteria and pruning then requires a balance between these differing objectives.

There is yet another consideration to be made. The coloring obtained by MAXIS is reduced by the action of the IG/Tabu coloring process. Different colorings, even when they require the same number of colors, may respond differently to this latter process. Recall for example the experimental results on k-colorable graphs using IG and Tabu. At this point we are unable to tell what effects there may be, although it seems that increasing the search done by MAXIS generally aids in reducing the final coloring.

To obtain some indication of these trade-offs we report here and in the next section results from a series of experiments.

	First Sort By		
Graph	Mean	Smallest	Largest
1	88/87	91/88	89/87
2	89/87	91/87	89/87
3	90/86	90/87	89/87
4	90/87	90/87	90/87
5	91/87	90/87	89/87
6	89/87	90/88	90/88
7	89/87	90/88	90/88
8	89/87	90/88	88/86
9	90/87	90/88	90/88
10	90/87	91/88	89/87
Total	895/869	903/876	893/872
CPU	1838.5/2268.8	2284.1/2865.8	1509.0/1792.8

Table 5: Colorings Obtained by MAXIS

In table 5 we list the colorings achieved on ten graphs in $\mathcal{G}_{1000,\frac{1}{2}}$. The first vertex of each independent set was selected according to the criteria indicated at the head of each column, while remaining vertices were selected by minimum degree. Using a sort which places values near the mean first, we obtain the best colorings, with an average of 86.9 colors used per graph. In each column a pair of colors is presented. The first member of each pair is the coloring number obtained by MAXIS, while the second is the coloring number after application of IG/Tabu. The first CPU time is the average required for MAXIS (in seconds), while the second is the average time to achieve the minimum coloring. [7] The parameters were chosen so that the algorithm continued for about one hour total time before terminating. Notice that the CPU times decrease as we select larger degree vertices for the first vertex of the independent set. Although it seems clear that the mean-first is better than the smallest-first, the difference between the mean-first and largest-first is not so clear. In particular, the largest-first found an 86-coloring not found by the mean-first.

Mean-first		Smallest-first	
Index	Number	Index	Number
1	4	1	5
2	2	2	1
3	2	3	2
4	1	4	2
-	-	6	1
7	1	7	1
8	1	-	-
9	1	9	1
10	2	-	-
15	1	-	-
total	15		13

Table 6: Number of 15 Sets Found

The role of finding large sets can be put in perspective by examining table 6. Here we list the number of independent sets of size 15 found by IndSet. Index i means the set found was the ith independent set of some graph. For example, the second last line means that the mean-first approach found one independent set of size 15 as the fifteenth independent set generated in a graph. Since all preceding sets were of size 14 except for two of size 15 in that particular graph, this means the 15 set was found in a subgraph of only 802 vertices.

Compared to these results, the largest-first approach only produced two

[7]Times in this section are all from the SUN IPC

15 sets, both from the same graph (with indices 1 and 2). In [20] it is pointed out that 80% of 1000 node random graphs should have a 15 set as the largest independent set, and in [4] the finding of 15 sets is again noted as being of importance. Comparing the results of the three approaches, it is not clear that finding the maximum sets is really the most important aspect of the search.

In each of the three experiments we used the cutoff specifications

$$(500, 8), (200, 6), (10, 3), (0, 10)$$

That is, if there were 500 or more vertices in the subgraph we cutoff after the first 8 vertices have been tried, if 200 to 499 vertices are in the subgraph the cutoff is after six vertices and so on. Many different cutoff patterns have been tried. It generally seems to be better to have a larger branching factor near the root of the search tree, with decreasing branching deeper in the tree. Keeping other factors equal, decreasing the branching factor for larger subgraphs, while increasing it for smaller subgraphs seemed to require more CPU time to achieve similar results.

If we select any vertex v from a χ-colorable graph, then there is some maximal independent set containing v which contains all vertices with the same color as v. This leads to the idea that perhaps it is not necessary to try more than one vertex as the first vertex of an independent set. Notice that this seems to contradict the observations of the preceding paragraph!

This can be implemented in MAXIS by simply not backtracking on the initial graph. The results of this approach are presented in table 7. Here we used only the largest-first approach, but present results from two different cutoff sets.[8]

We see in table 7 that the time required by MAXIS more than doubled for the more thorough search. Interestingly, the colorings achieved by MAXIS did not improve significantly, and actually became worse for graph 5. Comparing the results to table 5 we see support for the claim that it is better to have a higher branching factor near the root of the search tree.

It also seems that putting some time into IG/Tabu is better than relying solely on MAXIS. However, IG/Tabu seems only capable of reducing the coloring by a small amount, even if the initial coloring is quite bad. In fact, IG/Tabu was allowed to run for several minutes after the final colorings were achieved in the first case without further improvement.

[8]Each set actually started with the pair $(500, 8)$, but this is irrelevant given that no backtrack over 500 vertices occurs under the given conditions.

Graph	Cutoff Sets $(150, 10), (0, 3)$	$(120, 13), (0, 3)$
1	89/88	89/87
2	90/88	89/88
3	90/88	89/87
4	90/88	90/87
5	89/87	90/87
6	90/88	90/88
7	90/88	90/87
8	91/88	89/86
9	90/88	89/87
10	91/88	90/88
Total	900/879	895/872
CPU	492.1/1068.3	1390.6/1992.1

Table 7: Limited Backtrack Using MAXIS

Reducing the search time spent by MAXIS even further can still result in good colorings, with considerable savings in time. Using the cutoff set $(200, 6), (10, 3), (0, 10)$, with no backtrack on the first vertex selection, on 5 graphs we obtained a 91 coloring in each case in an average time of 247.8 seconds. This was reduced by IG/Tabu to an 89 or less coloring in an average total time of 377.3 seconds.

1	2	3	4	5	6	7	8	9	10
88/86	89/87	89/87	88/87	89/87	88/87	89/87	89/87	89/87	90/87
Total 888/869					CPU 1278.7/1691.1				

Table 8: Colorings Using Largest First, Mean Second, Smallest for the rest

Finally, we show the best results we have for these ten graphs, in terms of coloring and time in table 8[9]. In this case, we used the largest degree first to select the first vertex of the independent set, the mean degree first for

[9]Using the bounded backtrack ideas described below, which do not change the colorings in any way, preliminary tests indicate these times can be reduced by about 210 seconds to 1070/1480

the next vertex, and the smallest degree first for the remaining vertices. We again achieve an 86.9 average coloring, but this time the 86-colored graph is number one (that makes three of the ten graphs colored with 86 colors). Only two independent sets of size 15 were found. Complete backtrack was done with cutoffs $(500, 8), (150, 6), (0, 3)$. Additional tests were made with the first two or three vertices being chosen using mean degree first etc., but no better mix has yet been found.

A second set of experiments was done on graphs in $\mathcal{G}_{1000,\frac{1}{2},60}$. Recall that IG alone solved these graphs in about one hour. For MAXIS, we found the cutoffs $(500, 8), (200, 6), (10, 3), (0, 10)$ gave us the best results. These cutoffs are similar to the ones we used in the previous experiments, except that now the last pair, $(0, 10)$ helps prevent the formation of several small sets when only a small subgraph remains to be colored.

Using these, we tried the program using largest, mean and smallest-first sorting and present the MAXIS results in table 9. Limiting IG to 200 iterations after the last improvement, it always found the optimal 60-coloring using the *mean-first* approach in an average of 2073.4 seconds; that is an additional 1.8 seconds on average! However, it often failed to find 60-colorings for the other two approaches within this limit. Although below the plateau of difficulty, it nevertheless sometimes requires more than 200 iterations in these cases between improvements.

Earlier, the version of the program with the sorting bug also found similar good colorings for this class of graphs. Using mean-first or largest-first for the second vertex selection generally did not lead to good performance.

The differences in the colorings obtained by MAXIS using different sorting approaches are quite large on these graphs, and beg for an explanation. First, we try to explain why MAXIS should work as well as it does.

The basic feature of k-colorable graphs, when k is small enough, is that the independent sets of the optimal coloring are larger than the (millions of) remaining maximal independent sets. Suppose we have selected our first vertex v, and the maximum independent set containing it is $S(v)$. Then in the subgraph of vertices not adjacent to v will be all vertices in $S(v) \setminus \{v\}$, and about one half of the vertices of the remaining independent sets. A vertex in $S(v) \setminus \{v\}$ will be adjacent to about one half of the vertices in the subgraph not in $S(v)$, or about

$$N_{in} = \frac{(k-1)n}{4k}$$

vertices on average. Any vertex in the subgraph not in $S(v)$ will be adjacent

	First Sort By		
Graph	Mean	Smallest	Largest
1	66	86	85
2	63	84	75
3	66	84	85
4	69	85	78
5	67	87	74
6	73	87	80
7	66	85	82
8	76	71	83
9	65	82	86
10	67	84	85
Total	678	835	813
CPU	2071.6	2652.0	1708.6

Table 9: MAXIS on 60-colorable 1000 node graphs

to about one half of the vertices not in its maximal set, or about

$$N_{out} = \frac{1}{2}\left(\frac{n}{k} - 1 + \frac{(k-2)n}{2k}\right) \approx N_{in} + \frac{n}{2k}$$

For example, for $n = 1000$ and $k = 60$, the average degree of a vertex in $S(v)$ will be about 8 more than one not in $S(v)$. This implies that the vertices most likely to be of minimum degree in the remaining subgraph are those in the maximum set containing v. Thus, choosing a vertex of minimum degree after the first is chosen enhances the probability of selecting a vertex of the same set. This effect increases as further successful selections are made. Notice that the probability of a good selection is also increasing because the good set increases its proportion of the remaining subgraph.

Good approximations may still be made to some maximum set, even if the second selection is not from the same optimal set as the first. The third vertex selected may be from either of the first two sets, and then subsequent selections will be enhanced towards that set in a similar manner to that discussed in the previous paragraph.

If the coloring number increases much beyond 60, then the largest sets are no longer the ones induced by the k-coloring. When these sets are at or near the expected size of the naturally occurring independent sets in a random graph, the bias towards their selection disappears. However, other maximum independent sets of this size will not lead to a k-coloring in general, and so the algorithm fails in this case.

There is an anomaly suggested by this heuristic analysis when compared to the experimental results. It would seem that choosing a vertex of large degree on the first selection would increase the likelyhood of good selections later. The reason is that more vertices of other sets would be removed from the subgraph. However, the experimental results in table 9 provide dramatic evidence to the contrary. This leaves open the question of why this should be so. One possible answer (in part): when we choose the independent sets with the largest total degree, the remaining subgraph will be sparser. But this means the random sets will on average be closer in size to the specified sets, and this will make it harder to find the specified sets. The problem is that we must distinguish the independent sets of the specified coloring from the background.

The analysis above could also be applied to those sets which, when $n = 1000$, are in the range 14–15. This means the average degree difference between a vertex in a specified set and one in a 14 set each containing the initial selection for a 60-colorable graph would be about 1. In random graphs on 1000 nodes, the degree sequence varies from about 450 to 550, and so a bias of 1 is not that large. Apparently it is significant however, since we do manage to color these graphs consistently well.

⋆⋆⋆⋆⋆

This analysis suggested the intriguing possibility that for k-colorable graphs it might be better to select a maximum independent set of minimum total degree. The independent sets that remain might then tend to be more distinguishable from their neighbors. Provided that we find a specified set, choosing one of minimum total degree would leave the remaining subgraph more dense, thus making the natural sets smaller.

Testing this idea, using the maximum degree for the first vertex selected in each independent set, the colorings were in fact worse with a total of 867 colors used compared to 813 using maximum degree selected. These numbers refer to the colors used by MAXIS before applying IG. Using the mean degree for selection of the first vertex, the minimum total degree also proved worse with 711 colors used compared to 678. However, in this case IG was still able to reduce to 60-colorings quite quickly. In sum, the conjecture appears false, and selecting the independent set with the largest total degree seems the right choice.

⋆ ⋆ ⋆ ⋆ ⋆

To use MAXIS on k-colorable graphs for smaller k, we needed to prevent backtrack on small subgraphs when no possibility of finding a better set existed. For example, on $k = 40$, the specified sets are of size 25 ($n = 1000$). When the partially complete independent set has more than 16 vertices, it is likely that it is a subset of one of the specified sets, and that all vertices in the non-adjacent subgraph are in fact also in the same specified set. However, if the last cutoff used is $(0, 3)$, we will search various combinations of this set in vain. The execution times for MAXIS using the same set of cutoffs as above was drastically increased when k was reduced to 40.

To correct this, we used a bounded cutoff, in the manner of branch and bound. The rule used is:

> Bounded Cutoff Rule: If the number of untried vertices in the current non-adjacent subgraph plus the number already in the partial independent set is *less than* the number in the best independent set found so far, then backtrack.

If we replace "less than" in this rule by "less than or equal" we get a tighter bound, but restrict the search because equal sized sets are also tested for total degree.

Applying this rule reduces the search times for 60-colorable graphs significantly, and also reduces the coloring time of random graphs by 15% or so. For example on one 60-colorable graph the time was reduced from 2078.8/2080.5 to 1420.6/1421.8 for MAXIS/IG. As k is reduced in $\mathcal{G}_{1000,\frac{1}{2},k}$, the effect becomes more dramatic.

Using the tighter equality bound on the set of ten graphs in table 9 (with the mean degree selection) resulted in even quicker specified colorings with average times 1338.9/1341.2. For the graph in the preceding paragraph, the time was reduced to 1381.3/1387.5. Note that the average time IG used to complete the coloring increased slightly. This reflects an increase in the total colors used by MAXIS from 678 to 724, for an average increase of 4.6 per graph. Once again, restricting the effectiveness of the maximum total degree heuristic seems to cause a deterioration in the coloring ability of MAXIS, but in this case IG is able to reduce the coloring quickly with an over all time savings.

For $k = 40$, using the cutoffs $(500, 1), (0, 2)$, the MAXIS/IG program gave 40-colorings with an average time of 38.0 seconds, compared to about 108 seconds for IG alone. Using $(0, 1)$ as the cutoff left the coloring above the plateau, so that IG took too long to complete. Looser cutoffs caused MAXIS to take more time.

When we reduce to $k = 30$ it becomes more difficult to compete with IG alone. At this value, IG used only 22 seconds on average. Using the same cutoffs as for $k = 40$ caused MAXIS to take too much time. Using MAXIS with cutoff $(0, 1)$ followed by IG colored these in an average of 18.2 seconds.

With $k = 2$ IG alone used less than 5 seconds. Even with no backtrack, MAXIS required 7.6 to 9.3 seconds, and gave colorings from 24 to 34. IG very quickly reduced these colorings, but IG alone was faster. Of course, DSATUR would color these optimally.

9.3 Some Special Results

In [17] Johnson presents an algorithm that is guaranteed to give a coloring within a factor of $O(n/\log n)$ of the optimal. The algorithm is similar to MAXIS except there is no backtracking. During the independent set finding phase, vertices are recursively selected from the subgraph of vertices not adjacent to any in the current set. Selection is by minimum degree.

This algorithm is easily simulated by MAXIS, since we need only restrict the backtracking (to none). We present here some results from this approach. In table 10 we show the performance of this algorithm, and some interesting variations on graphs in $\mathcal{G}_{1000,\frac{1}{2}}$. A1 is Johnson's algorithm, A2 uses maximum degree for the selection of the first vertex and minimum degree after that, A3 uses maximum degree throughout, A4 uses mean degree throughout and A5 uses mean degree on the first selection with minimum after that.

Not only did A2 give a significant color improvement over the Johnson algorithm, but it also reduced the execution time by about 10% to an average of about 23 seconds.

On the other hand, A3 increased the execution time, likely because of the extra calls to the independent set routine. These numbers are far above those we would expect from the greedy algorithm. They may be a good approximation to the canonical achromatic number[23]. These numbers are a little less than double the best colorings for these graphs. The colorings produced by this algorithm are easily seen to be canonical.

Finally, notice that A4 produces results similar to but slightly better than the greedy algorithm applied to a random permutation.

Graph	A1	A2	A3	A4	A5
1	108	105	153	119	108
2	110	103	150	118	109
3	110	105	150	121	109
4	109	105	151	120	109
5	106	104	151	118	108
6	108	105	150	119	109
7	109	107	153	120	109
8	108	104	151	122	105
9	111	104	151	122	108
10	109	103	151	120	108

Table 10: MAXIS without backtrack

10 Summary of Mixed Algorithms

The results presented in this part of the report indicate that the mixed algorithms can show improvement over any of the algorithms alone when attempting graphs in $\mathcal{G}_{n,p}$. In particular, Tabu is a significant help to IG, and MAXIS yields further improvement, but requires IG/Tabu to reduce the coloring to close to optimal.

Tabu was not seen to be useful, in fact possibly harmful, when attempting to color graphs in $\mathcal{G}_{n,p,k}$. MAXIS made significant improvements in the time required to color these graphs, but IG is still required to reduce the coloring to the specified number. Generally, it would seem to take an inordinate amount of time for MAXIS to find the specified colorings on its own. It is not clear whether MAXIS could be tuned so that it finds colorings a little closer to the plateau, but still below it far enough so that applying IG would lead to further reductions in total time. The balance point between MAXIS and IG is a subject requiring further research.

MAXIS seems at first glance to be the most powerful technique. It should be noted that all of the tests described using MAXIS are on graphs with $p = \frac{1}{2}$. When the density is much lower, it may be difficult to use this approach because the size and number of the independent sets is so large. The time to search grows exponentially with the size and number of the independent sets, and so for low density graphs the pruning must be made much tighter. But then the effectiveness of the search is greatly reduced.

This is particularly true for low density k-colorable graphs in regions where IG is usually successful as illustrated in the next section. Attempting to use MAXIS on these graphs has proved frustrating.

Even when the density is fixed at $p = \frac{1}{2}$, it seems that to be competitive with IG alone the cutoffs must be tuned according to the value of k. Note that designing a reasonable automatic tuning mechanism would be difficult because the total density does not accurately reflect the distribution of edges in these graphs. It seems unlikely that MAXIS can improve the coloring times over IG for small k.

Part IV
The Coloring Landscape

The coloring of graphs in $\mathcal{G}_{n,\frac{1}{2}}$ has been the focus of much research in establishing the performance of coloring algorithms. However, we can achieve colorings near the expected optimal on 1000 nodes, and it is not clear how much further we can go.

In order to establish some goals, it seemed reasonable to further examine the limits of our algorithms by pushing the limits at various densities on k-colorable graphs. Since doing an intensive search on 1000 node graphs is quite time consuming, this mapping is carried out on 300 vertex graphs. Basically, a set of ten graphs was attempted for each combination of k, $2 \leq k \leq 50$, and for densities at every 5% from 5% to 70%. This set of tests was started with an earlier version of the program. The first step was Brélaz's DSATUR algorithm, followed by IG/Tabu. The attack consists of using IG and Tabu in a configuration that is fixed for all classes. This may not be the best approach for all classes, but at least has the advantage of consistency. The setup had to be a trade-off between maximal search time and how good we wished our results to be. Later tests using MAXIS on some of the ranges showed little improvement. Using MAXIS at low densities is difficult, because of the rapid increase in search time given large numbers of large independent sets.

For this set of graphs, we used up to 15 iterations of the main loop. On each loop, the IG cutoff was set at 1000 iterations after the last improvement in the coloring sum, and the Tabu cutoff was set at 5000 iterations after the last improvement. Tabu generated a sample of 200 neighbors on each iteration. Tests which increased the search showed little improvement, although of course if Tabu were to run long enough, with probability one it would find the optimal coloring.

For densities in the range 75% to 95%, a series of tests were run using MAXIS as the first step, followed by IG/Tabu. The main loop iterations were limited to 5 or 6, and IG iterations were reduced to 200, while Tabu was reduced to 2000 iterations.

In all of the graphs, as soon as a coloring was found using no more than the specified number of colors, the search was terminated.

Figure 9 plots the number of graphs not colored by the specified number of colors(i.e. the number of partitions). Recall there are ten graphs at each point. The lines indicate tests that were performed but for which every graph was colored with no more than the specified number of colors. Notice that for densities in the range 10% to 45% there tends to be an immediate drop off from 9 or 10 missed graphs to zero on the next partition number. We refer to the band of graphs forming these peaks as the *ridge of resistance.*

Figure 10 shows the total number of colors used in excess of the specified coloring. Negative excesses are shown on the right for some densities, but to keep the figure readable, the lines have been truncated when the excess is less than -10. The negative regions could no doubt be extended to the left, that is to lower partition numbers, if we were to continue the search after the specified coloring number has been reached. However, that would add enormously to the cost of the experiment. The negative excesses that have been found are due to DSATUR finding colorings less than the specified coloring, or occasionally due to IG reducing the coloring by more than one during some iteration.

Notice that the excess tends to drop gradually to the right, after an initial sharp rise. This is even more significant when we notice that the number of missed graphs is usually lower on the left than the right of the ridge. Figure 11 plots the ratio of excess colors to the number of graphs not colored with the specified number. This clearly shows that on the left side of the ridge, i.e. for low partition numbers, we tend to miss by a large margin, although we do not miss too often. As the partition number increases, we tend to miss more graphs, but reach colorings much nearer the specified coloring. Finally, to the right of the ridge we find colorings as good as the specified coloring number. These are likely not the sets of the specified colorings. For a fixed p, as k increases towards n the expected chromatic number of graphs in $\mathcal{G}_{n,p,k}$ will converge to that for $\mathcal{G}_{n,p}$. We may not find the optimal coloring, but we will find something as good as the specified coloring.

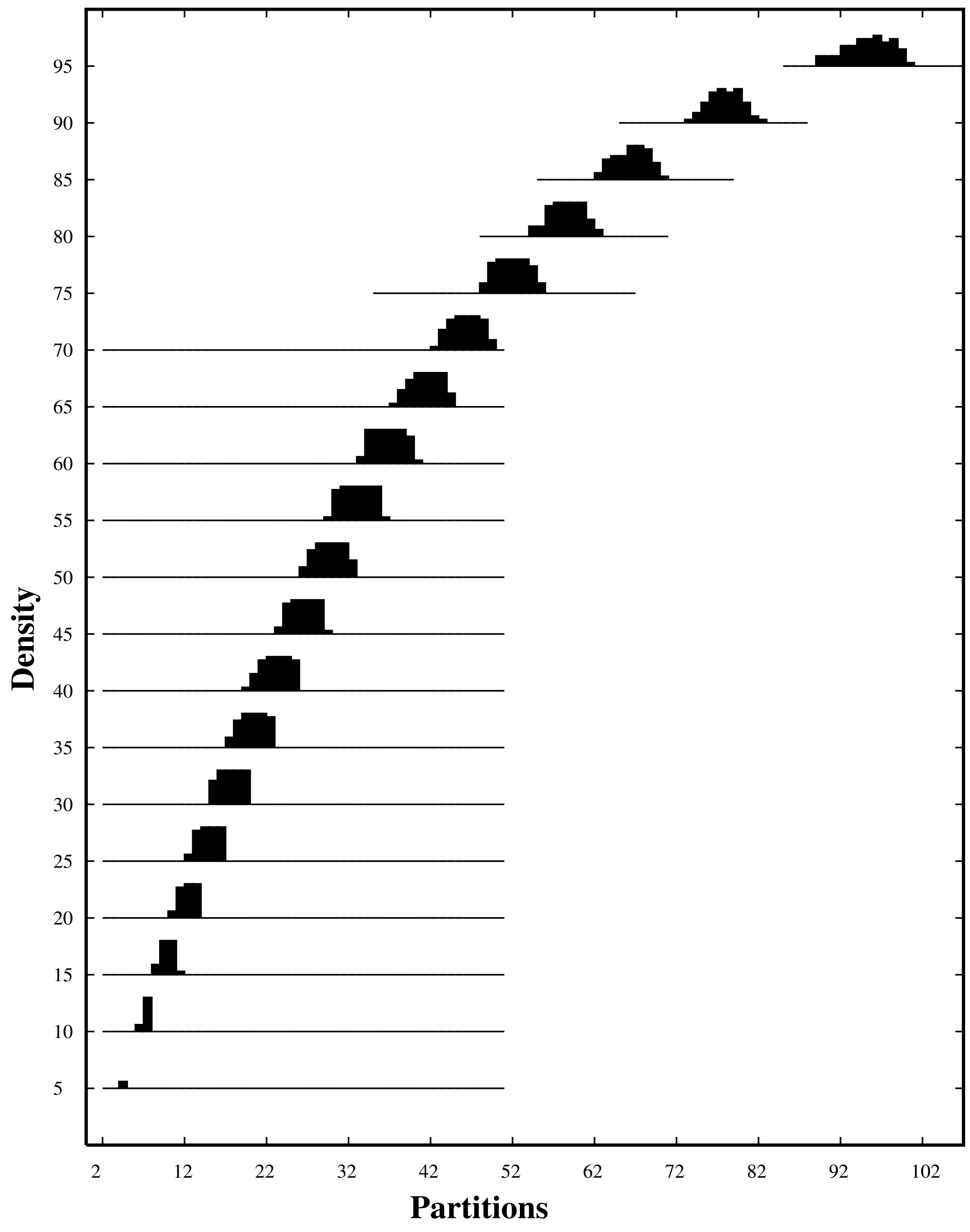

Figure 9: The Number of Missed Graphs

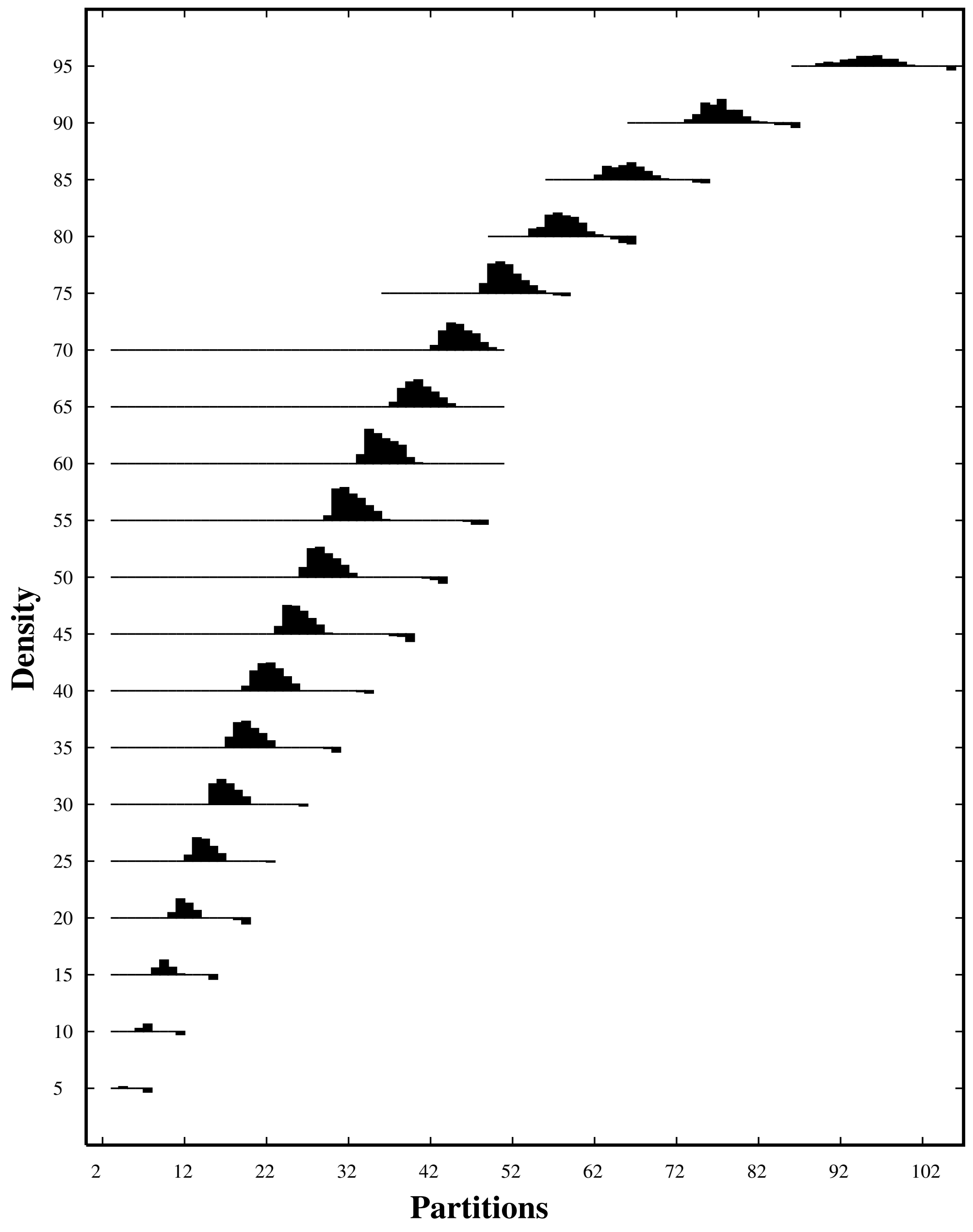

Figure 10: Excess Colors Used in the 300 Vertex Landscape

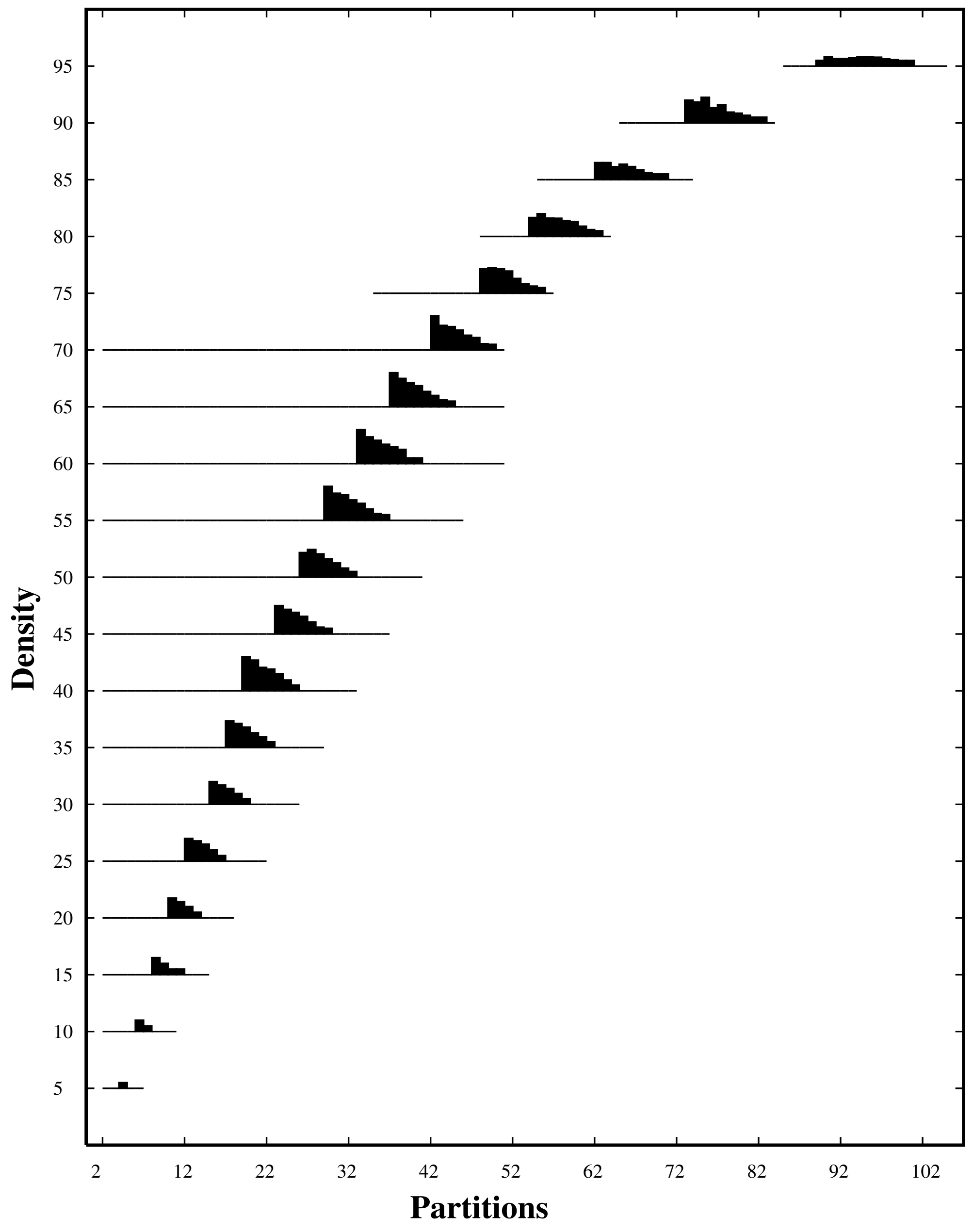

Figure 11: Ratio of Excess Colors to the Number of Missed Graphs

As we increase the density, the ridge gradually becomes lower in terms of the number of colors wasted, but broader. Clearly, for densities of 100% we would color all graphs optimally. Similarly, for partition numbers of two or less, we will obtain optimal colorings using DSATUR.

However, we can extend the ridge down to densities of 2.6% and 3-colorable graphs which are not 3-colorable by our algorithms. (One has to be careful; with probability one, using Tabu for example for an arbitrary length of time, a three coloring would eventually be found. We have to think in terms of reasonable time, and our algorithm generally gives up in twenty minutes to one half hour on these). The fractional percentages are necessary here; the algorithms succeed perfectly and quite quickly for $k = 3$ and density of 2%, and reasonably often at 3%. They fail up to nine out of ten times for intermediate values.

All graphs to the upper left of the ridge are colored with the specified coloring very quickly, often in under a second. We list in table 11 the maximum excess colors used at each density, totaled over ten test graphs for each.

It seems that the reason for the ridge's existence is that it is the point at which the maximum independent sets that occur naturally begin to coincide in size with the sets of the partitioning. When the specified coloring becomes large enough, other colorings are superior, and there are lots of them so one is easily found. When the specified independent sets are slightly smaller than the natural ones, MAXIS for example will select the larger sets. These sets will generally not contain one of the specified sets. But this then forces extra colors later on to color vertices that are left over from the specified sets. For example, with $k = 60$ and $p = 0.80$, MAXIS found independent sets of size 7 in two cases, and many sets of size 6, even though the specified sets were of size 5. In this case MAXIS found colorings ranging from 63 to 66 in the ten graph sample, and although IG reduced each coloring somewhat, no graph in the set of ten received a 60-coloring.

Cheeseman, Kanefsky and Taylor [7] noted a similar threshold effect, stated in terms of constraint satisfaction, for three and four colorable graphs. They related this phenomenon to several different NP-complete problems.

This search only looked at graphs of 300 vertices. It would also be useful to have a map of how the difficulty changes with a variation in the number of vertices. Together these two approaches would yield a three dimensional map of difficulty. In [10] for example, Ellis and Lepolesa mention that their algorithm has difficulty on graphs in $\mathcal{G}_{n<80,\frac{1}{2},10,0}$, although it is effective on

Density %	k	Excess
5	4	2
10	7	10
15	9	20
20	11	26
25	13	32
30	16	34
35	19	36
40	22	38
45	24	39
50	28	41
55	31	45
60	34	47
65	40	37
70	44	37
Altered Search		
75	50	43
80	57	32
85	66	23
90	77	32
95	96	14

Table 11: Maximum Excess Colors Used on 300 Vertex Graphs

very small n. There is undoubtedly a ridge with respect to the number of vertices and k at most given densities.

There is a tendency to think of the ridge as being the difficult part of the landscape to color. However, it should be pointed out that below (and to the right of) the ridge we do not necessarily have the optimal coloring. Thus, although it is easy to reach the specified coloring in this region, it is undoubtedly hard to color these graphs optimally.

We end this section with a bit of speculation. As stated in the introduction of this paper, the best worst case results are those of Blum [2], with colorings in $\tilde{O}(n^{0.4})$ of the optimal. We know from Gary and Johnson's result [11] we are not likely to get closer than a factor of 2 in the worst case in polynomial time. On the other hand, here we see that we need to get very close to the size of the naturally occurring set sizes with our specified sets before they are not distinguishable in reasonable time by our system. Similar results seem to be holding for the 1000 node graphs. The result is that for our test suite, the worst case ratio of coloring found to the specified coloring is 1.246 for $p = 0.30$ and $k = 16$ (see table 11), or 1.3 for three colorable graphs at $p = 0.026$. Similarly, the worst case ratio for 1000 node graphs at $p = 0.50$ will certainly be less than 1.45, since we have easily colored 60-colorable graphs and have consistently obtained 87 or better colorings for graphs in $\mathcal{G}_{1000,\frac{1}{2}}$.

So the question arises, will our system, given polynomial time, give a coloring on average within a constant factor less than 2 of the optimal? Or can it be that choosing a threshold carefully, we can demonstrate that the coloring ratio grows unboundedly? If we use small densities and large n can we obtain arbitrarily large average colorings for $k = 3$?

Part V

Other Results, Conclusions and Questions

11 Other Programs

Iterated greedy seems so successful that we would like to try other iterated approaches. We can do this using the MAXIS iteratively as follows. After an initial coloring, reorder the color sets as in IG, and then using the vertices from the first set in the new ordering, use (pruned) backtracking to try to add as many vertices to this set from the vertices to its right in the new ordering as possible. When the first set is as large as possible, repeat the process on the vertices that remain in the second set, then the third and so on. The idea is that if using the greedy iteratively is successful, then this super greedy iterative approach should do even better.

However, it was not too successful when tried on graphs in $\mathcal{G}_{1000,\frac{1}{2}}$. The smallest set produced by MAXIS has only a few vertices, but the remainder have $m = 8$ or 10 or more vertices. The remaining vertices have probability $\left(\frac{1}{2}\right)^m$ of not conflicting with one of the vertices already in the set, and so the expected number of vertices over which we could backtrack is about 1 to 4. Thus, there is not much chance for backtrack to work. The idea of IG is that it searches through the many combinations of moves, but not many vertices can be moved into any particular set.

We also tried unsuccessfully to create an iterated form of Brélaz's DSATUR algorithm. The idea was that we would use the previous coloring to determine the order when we had more than one vertex with the same maximal number of constraints. Unfortunately, we did not find a way to guarantee that the new coloring would be no worse than the previous one. As a result, the coloring just thrashed around near the 112 to 115 mark, with no improvement discernible.

A super DSATUR algorithm was attempted, based on using MAXIS to find large cliques. These would then be ordered in decreasing size, and this order would be used by DSATUR when more than one vertex had the same number of constraints. This approach showed no significant improvement over DSATUR using a random order.

This entire project grew from an initial trivial attempt to apply the technique of Genetic Algorithms [13] to graph coloring. Several variations were attempted without much success, but in trying to determine what should be preserved we observed that reordering the color sets and applying the greedy algorithm would not produce a worse coloring. This led directly to the IG algorithm.

12 Conclusions and Open questions

We have introduced a new coloring technique we call iterated greedy. This technique seems to be more effective for coloring graphs in the class $\mathcal{G}_{n,p,k}$ than other techniques we have tried. It is also useful in trying to reduce colorings obtained by other algorithms.

We have explored various heuristics that can be used in this algorithm. In addition, we have described a method for combining this technique with Tabu coloring to give better results in certain classes. And we have looked at an algorithm using the maximum independent set approach, and briefly tried that in an iterative form as well.

Finally, we have explored the difficulty of coloring k-colorable graphs in the 300 vertex landscape, identifying a ridge of difficulty, extending down to even 3-colorable graphs. We issue this ridge as a challenge to researchers in testing their coloring algorithms.

References

[1] Bengt Aspvall and John R. Gilbert. Graph coloring using eigenvalue decomposition. *SIAM Journal on Algebraic and Discrete Methods*, 5(4):526–538, 1984.

[2] Avrin Blum. An $\tilde{O}(n^{0.4})$-approximation algorithm for 3-coloring (and improved approximation algorithms for k-coloring). In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 535–542, 1989.

[3] B. Bollobás. The chromatic number of random graphs. *Combinatorica*, 8(1):49–55, 1988.

[4] Béla Bollobás and Andrew Thomason. Random graphs of small order. In *Random Graphs '83*, volume 28 of *Annals of Discrete Mathematics*, pages 47–97. North-Holland Publishing Co., 1985. Section 6: "Colouring large random graphs".

[5] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, April 1979.

[6] M. Chams, A. Hertz, and D. de Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32(2):260–266, 1987.

[7] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the *really* hard problems are. In *International Joint Conference on Artificial Intelligence*, pages 331–337, 1991.

[8] F. D. J. Dunstan. Sequential colourings of graphs. In *Proceedings of Fifth British Combinatorial Conference*, pages 151–158. Utilitas Mathematica, 1976.

[9] R. D. Dutton and R. C. Brigham. A new graph coloring algorithm. *The Computer Journal*, 24(1):85–86, 1981.

[10] J. A. Ellis and P. M. Lepolesa. A Las Vegas coloring algorithm. *The Computer Journal*, 32(5):474–476, 1989.

[11] M. R. Garey and D. S. Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM*, 23:43–49, 1976.

[12] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[13] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc., 1989.

[14] G. R. Grimmett and C. J. H. McDiarmid. On colouring random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 77:313–324, 1975.

[15] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39(4):345–351, 1987.

[16] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[17] D. S. Johnson. Worst-case behavior of graph-coloring algorithms. In *Proceedings of 5th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 513–528, Winnipeg, 1974. Utilitas Mathematica.

[18] A. Johri and D. W. Matula. Probabilistic bounds and heuristic algorithms. Technical Report 82-CSE-06, Southern Methodist University, Department of Computer Science, 1982. Supposed to have appeared?

[19] Luděk Kučera. Graphs with small chromatic numbers are easy to color. *Information Processing Letters*, 30:233–236, 1989.

[20] Bennet Manvel. Extremely greedy coloring algorithms. In *Graphs and applications (Boulder, Colo., 1982)*, Wiley-Intersci. Pub., pages 257–270, New York, New York, 1985. John Wiley & Sons, Inc.

[21] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.

[22] David W. Matula, George Marble, and Joel D. Isaacson. Graph coloring algorithms. In *Graph theory and computing*, pages 109–122. Academic Press, Inc., 1972.

[23] Colin McDiarmid. Colouring random graphs badly. In *Graph theory and combinatorics (Proc. Conf., Open Univ., Milton Keynes, 1978)*, Res. Notes in Math. 34, pages 76–86. Pitman, San Francisco, Calif., 1979.

[24] J. Mitchem. On various algorithms for estimating the chromatic number of a graph. *The Computer Journal*, 19:182, 1976.

[25] S. Sen Sarma and S. K. Bandyopadhyay. Some sequential graph colouring algorithms. *International Journal of Electronics*, 67(2):187–199, 1989.

[26] Jeremy P. Spinrad and Gopalakrishnan Vijayan. Worst case analysis of a graph coloring algorithm. *Discrete Applied Mathematics*, 12(1):89–92, 1985.

[27] Jonathan S. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9:63–82, 1988.

[28] Janez Žerovnik. A randomised heuristical algorithm for estimating the chromatic number of a graph. *Information Processing Letters*, 33:213–219, 1989.

[29] D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its applications to timetabling problems. *The Computer Journal*, 10:85–86, 1967.

[30] D. C. Wood. A technique for coloring a graph applicable to large scale timetabling problems. *The Computer Journal*, 12:317–319, 1969.

[31] A. A. Zykov. On some properties of linear complexes. *Mat. Sb.*, 24:163–188, 1949. English Trans. Amer. Soc. Translation no. 79, 1952.