

Ordered Greed, II: Graph Coloring

Peter G. Anderson
Rochester Institute of Technology
Rochester, NY
anderson@cs.rit.edu

Abstract

A popular application of genetic algorithms (GAs) is to attempt to generate good, rapid, approximate solutions to NP-complete or NP-hard problems.

Previously, in [1], and [4], we introduced a hybrid algorithm combining a GA with simple greedy algorithms applied to the N -Queens problem and to sports tournament scheduling. The greedy algorithm makes locally optimal assignments (to Queens or matches) in some order. We treat that *ordering* as the sought after goal, and thus work with a population whose individuals are permutations.

The subject of the present paper is the problem of graph coloring. We focus the present paper on the single benchmark problem of coloring a three-colorable graph that was constructed as a subgraph of the complete 3-partite graph $K_{p,q,r}$ in which each edge exists with probability 0.1. (We have applied our method successfully to several other categories of graphs, but present space limitations dictate presenting the results for this special case.)

1 Plan of the Paper

Section 2 reviews genetic algorithms and presents the details of the GAs used for the present study.

Section 3 presents the necessary graph theory notation and the simple greedy graph coloring algorithm which we hybridize with a genetic algorithm.

Section 4 gives the details of our experiments and the outcomes.

Finally, section 5 gives a sampler of other applications of the ordered greed algorithm.

2 Genetic Algorithms and Graph Coloring

In this section we review the basic genetic algorithm and introduce our principal application, coloring a 3-partite graph.

There are a wide variety of problems—the NP-complete problems—without particularly efficient algorithms for solving them (Section 5 lists several). How-

ever, these problems often have very simple methods for evaluating or scoring proposed solutions.

Genetic algorithms (GAs) [3] often apply to such situations: problems without efficient algorithms but with very simple evaluation methods. A GA works with a *population* of problem solutions, generally initialized with the help of a random schedule generator. The population is manipulated by a process inspired by *selective breeding*—individuals are evaluated, and that evaluation is termed their *fitness*. The relatively more fit individuals are *selected* and are subject to operations of *mutation* and *breeding*. *Mutation* generally refers to small random modifications, and *breeding* refers to the creation of new *children* individuals from the pieces of the chosen *parent* individuals. The qualities that made the parents more likely to be chosen for breeding may be combined in a child whose fitness exceeds that of the parents.

A significant issue for a GA designer is how the solution for the particular problem (the *phenotype*) is represented as an element of the population (the *genotype* or *chromosome*) in a manner that make mutation, breeding, and fitness evaluation straightforward. If the problem were of the form: locate a number x between 0 and 1 to maximize $f(x)$ (suppose calculus does not easily apply to f), then we could let the GA's individuals be fixed-length strings of binary or decimal digits, representing x in usual radix arithmetic. Fitness evaluation is straightforward (supposing f is easily computed), mutation can be a simple modification of a bit or digit, and breeding can be *two-point crossover*, which is simply the interchange of substrings. For our present application, the genotype is a permutation of N objects, so mutation and crossover have to be handled more delicately. This issue is addressed as permutation *signatures* in [1].

Other issues that the GA designer must address are:

- population size,
- mutation rate,
- breeding algorithm,
- population size management (that is: how are individuals chosen to be removed from the population),

- parent selection strategy.

These affect how rapidly the GA will solve the problem, which can be the difference between seconds and eons. GA practitioners develop an insight for good values for these parameters, and systematic experimentation with a specific problem can yield reasonable values—this is especially important for repetitious problems (e.g., scheduling many tournaments or rescheduling a tournament after some teams drop out).

2.1 GA Details

The genetic algorithm used in these experiments has the following form.

```

Initialize the population to POP_SIZE random individuals.

Evaluate the fitnesses of the members of the initial population.

WHILE the termination condition is not met {
    Run two tournaments to select two parents and two individuals to leave the population.

    Cross-over the parents to form two children.

    Mutate the children.

    Evaluate the children's fitnesses.
}

```

The WHILE loop terminates after a specified number of fitness evaluations have been executed or the desired fitness has been achieved.

Parent are selected by tournaments. The most fit of `tournament_size` *unique* individuals is chosen to be a parent; the least fit is chosen to be replaced by a new child. The participants in the two tournaments are all unique, and the tournament losers are never equal to the winners (as could occur when fitness values are the same). When `tournament_size=2`, this method is equivalent to parent selection proportional to rank ordering; when `tournament_size= k`, this method is equivalent to parent selection proportional to the $k - 1$ power of the rank. Hence, for lower values of `tournament_size`, the search is more exploratory; for higher values of k it is more exploitive. Our present experiments have shown that the search is quite insensitive to `tournament_size`, and we have preferred to use tournament sizes of 5. (We report an experiment with a large range of tournament sizes in Section 4.

We use *uniform crossover* of permutation signatures. (Signatures are simple representations of permutations; see [1].)

We typically use a mutation rate of 0.04; i.e., every element of each new child is randomly recomputed with this probability. In other words, we expect to recompute 4% of each new child by mutation.

3 Graph Coloring Problems

A graph $G = (V_G, E_G)$ consists of a finite set V_G of *vertices* and a set E_G of *edges*. An edge is a set of two distinct vertices. The edge (v, w) is said to *connect* vertices v and w , and those two vertices are said to be *adjacent* in G .

A *coloring* of a graph G is a function

$$c : V_G \rightarrow K$$

where K is the set of colors (represented in our application by positive integers), such that

$$(v, w) \in E_G \Rightarrow c(v) \neq c(w)$$

G is *k-colorable* in case there exists a coloring for which the number of elements of K is k . The smallest k for which G is k -colorable is called the *chromatic number* of G .

The most famous instance of a graph coloring problem is that of coloring a graph associated with a map of countries. In this case the vertices of the graph are the individual countries, and the edges connect vertices corresponding to countries that share a non-zero length of border. Touching countries should be colored differently. Graph coloring also provides a model for examination scheduling with the exams providing vertices, conflicts providing edges, and colors the examination time slots; conflicting exams should be given at different times. A similar example is process scheduling: processes that share resources must be scheduled at different times, and this can be modeled by the graph coloring problem. For all of these problems, we desire the minimum number of “colors” or, at least, a very small number of colors in a coloring that can be determined in a reasonable amount of time. To determine whether a given graph is k -colorable is known to be NP-complete, so there is no known efficient algorithm—fast approximation algorithms are desirable.

In the present report, we restrict our study to random k -partite graphs with edge density $p = 0.1$ (the reason for this choice is detailed in Section 3.1). Graphs in this family are constructed so they can clearly be k -colored, although some instances are quite difficult. We focus on $k = 3$. See [2],

3.1 A Graph Coloring Algorithm

The following algorithm, called *the sequential algorithm* in [5], is the simplest method to color a graph assigning different colors to adjacent vertices. The “colors” are represented by positive integers. (See also

[6].)

```
Given: a permutation of  $V_G$ :  $v_1, v_2, \dots, v_N$ 

for (  $k = 1$ ;  $k \leq N$ ;  $k++$  ) {

    Color  $v_k$  using the smallest color number
    not assigned to a vertex adjacent to  $v_k$ .

}
```

Different permutations of the set V_G can give different color assignments. An assignment using the minimum number of colors is clearly achievable using this algorithm: simply color all the vertices that should have color 1 first, followed by those that should have color 2, and so on. A wide variety of heuristics, along the lines of Warnsdorff’s knight’s tour [7], can often improve the performance of the sequential algorithm. For example, first color the vertices whose adjacent vertices use the largest number of colors, and break the ties among these by the given vertex permutation.

The efficacy of this algorithm is demonstrated by the fitness histograms shown in Figures 1 and 2. The “fitness” of a coloring is the number of vertices colored by a specified number of colors; this is the fitness we use in the GAs. The experiments illustrated use graphs G with N vertices and edge set constructed according to the following algorithm.

```
Build a graph with  $V_G = \{1, 2, 3, \dots, N\}$ .

for (  $v = 1$ ;  $v \leq N$ ;  $v++$  ) {

    for (  $w = v + 1$ ;  $w \leq N$ ;  $w++$  ) {

        if  $v \not\equiv w \pmod{3}$ 
        then  $(v, w) \in E_G$  with probability  $p$ .

    }

}
```

Figure 1 refers to a graph G_{100} with 100 vertices, and Figure 2 refers to a graph G_{3000} with 3,000 vertices. Both graphs were constructed with edge density $p = 0.1$. Fitness is the number of vertices colored using colors 1, 2, and 3. These graphs are clearly three-colorable, but the sequential algorithm is most unlikely to stumble onto a valid three-coloring. The reason for our selection of edge density $p = 0.1$ is illustrated in Figure 3. We performed 10,000 coloring attempts with graphs of 100 vertices and edge densities of 0.01–0.40. Graphs with very small edge densities probably have many very small connected components, thus easy to color. Those with high edge densities have an easy coloring strategy: locate and color the vertices in a tri-

angle, then color the vertices in triangles that have an edge in common with a colored triangle. Graphs with large edge density have many triangles. Presumably, the sequential algorithm was easily able to exploit this coloring principle. We chose to concentrate our efforts at edge density 0.1, because that is near the hardest for $N = 100$ and for its mnemonic value.

3.2 Ordered Greed—A Natural Hybrid

The ordered greed algorithm (OG) uses a population of permutations. The fitness of a permutation is the fitness of the resulting phenotype created using the greedy algorithm driven by that particular permutation. In the present case, the fitness is the number of vertices successfully colored with a specified number of colors using the sequential algorithm.

In our previous paper, [1], we placed Queens on an $N \times N$ chess board according to an ordering of the rows of the board. Each Queen was placed as far left as possible avoiding attacking any previously placed Queens. Similarly, we scheduled an invitational soccer tournament by breeding permutations of the matches that needed to be played, and placing each match on the schedule greedily, in turn given by the permutation.

4 Experiments and Results

Our first experiment, to get a feel for the problem and the behavior of our algorithms (greedy coloring and ordered greed), used the three-colorable graph G_{100} with 100 vertices and edge density $p = 0.1$. The three vertex subsets of mutually non-adjacent vertices are of size 33, 33, and 34. The fitnesses of 1,000,000 attempts to three-color G_{100} using the sequential algorithm are shown in Figure 1. The ordered greed algorithm was then run 990 times with the population size running from 10 to 999. Figure 4 shows the number of fitness evaluations needed for each of these population sizes to achieve the 3-coloring of the entire G_{100} . For populations under 130, tens of thousands of fitness evaluations were often needed; apparently the population diversity was too poor to succeed quickly. For populations over 130, the number of required fitness evaluations was 10–20 times the population size. The other parameters for these runs were fixed at a tournament size of five and a mutation rate of 0.04.

Tournament size appeared not to make a significant difference on the speed of three-coloring G_{100} . Figure 5 shows 98 experiments with the tournament size varying from 2 to 99. The average number of fitness evaluations was 2,674. We used a population size of 200, as was suggested by the previous experiment.

Figure 6 shows the rapid progression of the ordered greed algorithm towards a three coloring solution for a graph with 3,000 vertices. As illustrated in Figure 2, randomly searching the space of permutations is a

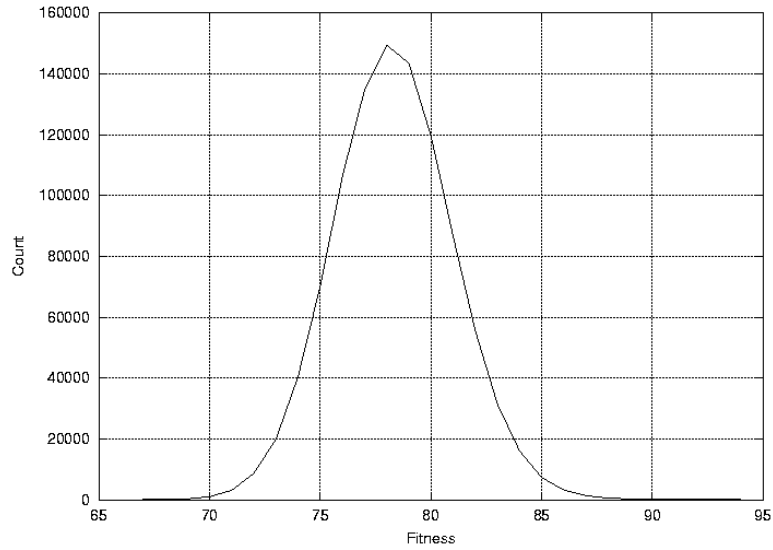


Figure 1: Fitness histogram for 1,000,000 attempts to color the 100-vertex, three-colorable graph, G_{100} , using the sequential algorithm with 1,000,000 randomly chosen permutations of $(1 \cdots 100)$. The fitness is the number of vertices successfully colored with three colors.

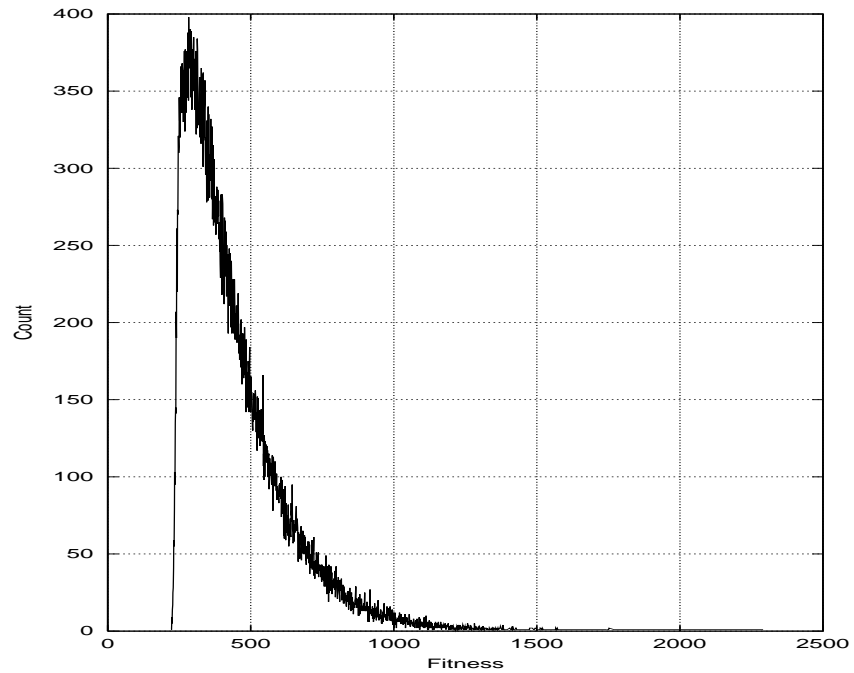


Figure 2: Fitness histogram for 100,000 fitness evaluations. The fitness was the number of vertices successfully colored with three colors in a three-colorable graph G_{3000} with 3,000 vertices.

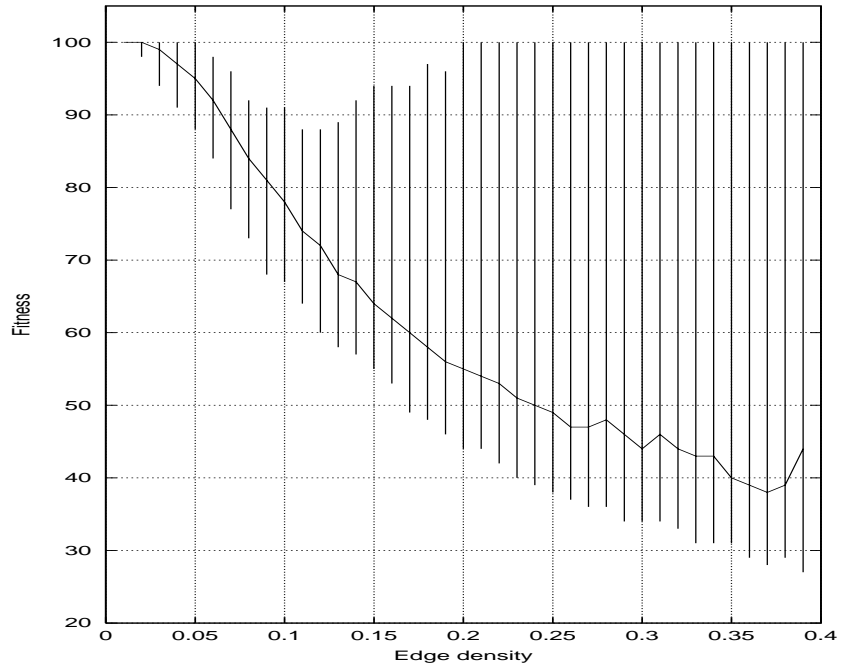


Figure 3: The minimum, mode, and maximum fitness for the graph coloring algorithm to attempt to three-color a three-colorable graph with 100 vertices and edge density 0.01–0.40. For each density, there were 10,000 fitness evaluations.

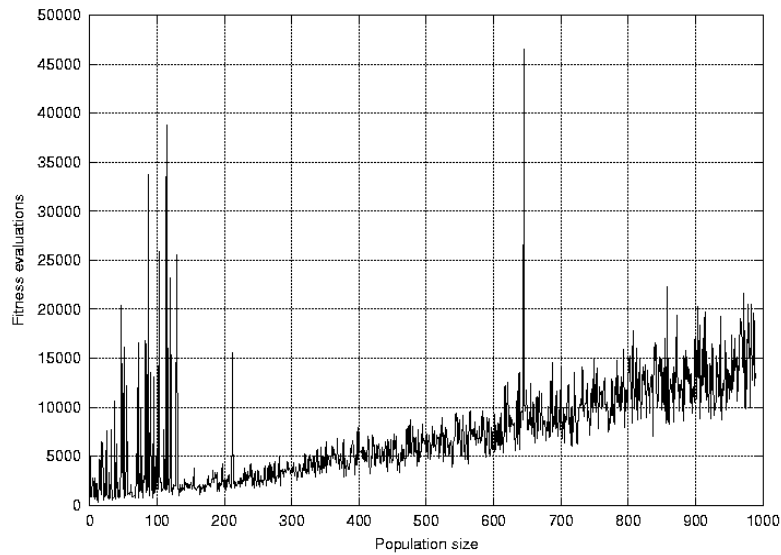


Figure 4: The number of fitness evaluations to three-color G_{100} using ordered greed with population size varying from 10 to 999.

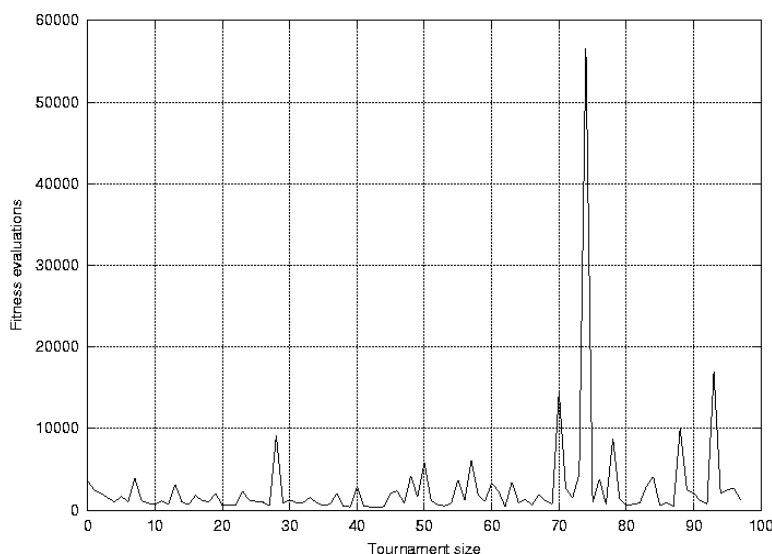


Figure 5: The number of fitness evaluations needed to three color G_{100} using tournament sizes from 2 to 99.

terrible search technique for this problem, but ordered greed solved it readily.

5 An OG Applications Sampler

Ordered greed is a natural and appropriate approach for a wide variety of difficult problems. It is only necessary that a problem's optimal solution can be found using a greedy algorithm given the right permutation. Then, there will generally be a huge number of permutations that produce the same, or equivalent, answer.

Here is a list of some of these problems that we have investigated. (Space constraints preclude more than just a brief mention.)

- The N Queens problem.
- Sports tournament scheduling.
- The job assignment problem.
- Matching.
- Ramsey theory coloring.
- The Traveling Salesman Problem.
- Bin packing.
- Pentominoes.
- Multiprocessor scheduling.
- Faculty teaching assignments.
- Airline crew assignments — set partition.

References

- [1] Peter G. Anderson and William T. Gustafson. Ordered greed. In *Proceedings of the ICSC Conference on Soft Computing, Genoa, Italy (SOCO'99)*. International Computer Science Conventions, 1999.
- [2] Béla Bollobás. *Random Graphs*. Academic Press, London, 1985.
- [3] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [4] William Gustafson. Building a soccer tournament schedule using a genetic algorithm. Master's project, Rochester Institute of Technology, 1998.
- [5] Marek Kubale. *Introduction to computational complexity and algorithmic graph theory*. Gdańskie Towarzystwo Naukowe, Gdańsk, 1998.
- [6] James A. McHugh. *Algorithmic Graph Theory*. Prentice Hall, Englewood Cliffs, NJ, 1990.
- [7] H. C. Warnsdorf. *Des Rösselsprunges einfachste und allgemeinste Lösung*. Schamlkalden, 1823.

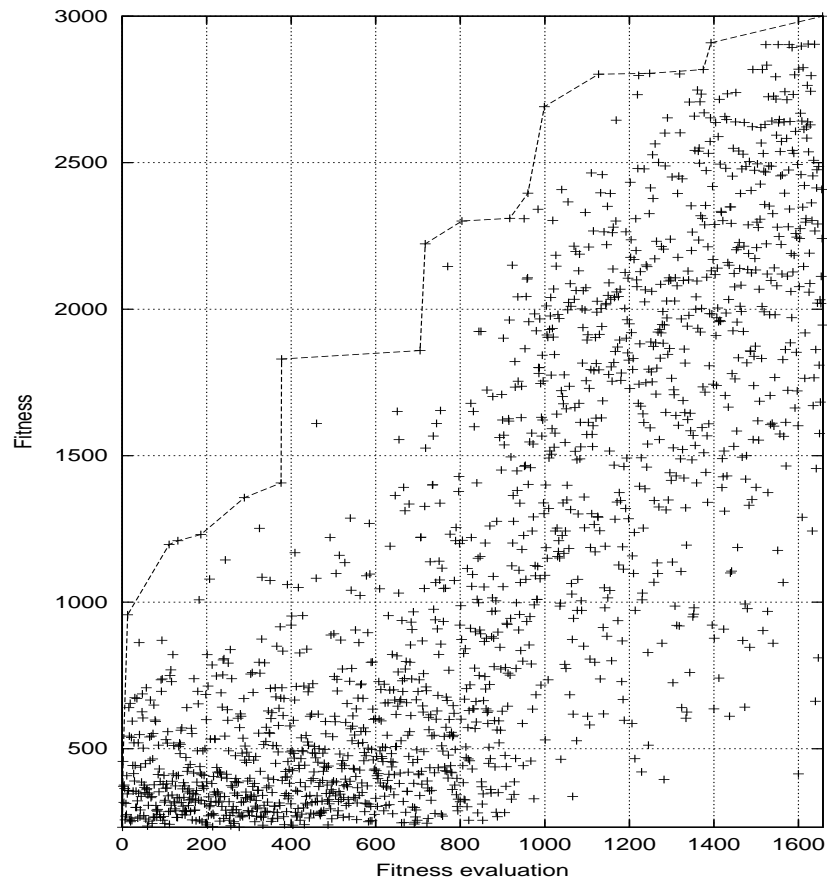


Figure 6: The 1,658 fitness evaluations to 3-color G_{3000} , the 3,000 vertex, 3-colorable graph with edge density 0.1. The ordered greed algorithm used a population of 200, 5-element tournament selection, and uniform crossover of permutation signatures.