Rochester Institute of Technology
RIT Scholar Works

Presentations and other scholarship

4-10-2000

Ordered greed

Peter Anderson

William Gustafson

Follow this and additional works at: http://scholarworks.rit.edu/other

Recommended Citation
Anderson, Peter and Gustafson, William, "Ordered greed" (2000). Accessed from
http://scholarworks.rit.edu/other/199

This Conference Proceeding is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Presentations and other scholarship by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.

Ordered Greed

Peter G. Anderson and William T. Gustafson
Rochester Institute of Technology
Rochester, NY 14623-0887
anderson@cs.rit.edu

April 10, 2000

Abstract

Scheduling problems are among the most challenging and realistic problems application of problem solving heuristics, such as genetic algorithms (GAs). The naive greedy algorithm for scheduling simply assigns, in turn, each item to be scheduled the best yet untaken position for that item. We investigate using a genetic algorithm to search the space of orderings for this greedy algorithm. That is, the GA individuals are permutations that *determine* the permutations that are the schedules, rather than the GA individuals directly *being* the schedules.

We have experimented with the classical N Queens problem and a realistic soccer tournament scheduling problem, comparing the GA individual as the assignment with our greedy hybrid algorithm ("ordered greed").

Warnsdorff's heuristic is introduced to modify blind greed with excellent results.

We also introduce the use of *signatures* in our GAs to represent permutations. Signatures are easy to create and manipulate in crossover and mutation operations.

1 Genetic Algorithms and Scheduling

There are a wide variety of problems without particularly good algorithms for solving (section 8 lists several) that do have, however, very simple methods

for evaluating proposed solutions. It is, for example, very difficult to assign teams to fields and times for a large, weekend (Friday evening through Sunday afternoon) soccer tournament. The scheduling committee must try to satisfy all of the following criteria:

- A team can only play one match at a time.
- A field can only be used for a single match at a time.
- The older players' teams must play on the larger fields.
- Fields used in the evening must have lights.
- Teams must be scheduled at times that do not conflict with other commitments.
- Teams should rest two game periods between matches.
- Teams playing late on Friday should not play early on Saturday.
- Divisions (i.e., groups of 4–6 teams mad according to age and sex) should have as few game times as possible.
- If a division's teams cannot be run simultaneously, they should be held as close together as possible, temporally and physically.
- Games for a team should be evenly distributed over the playing days.
- The games should finish as early as possible on Sunday.
- Teams should play on at most two different field areas.
- Each team should play at least once in the main filed area.

Each of these constraints can be assigned a numeric value, with high costs associated with violating *hard constraints*, such as the first five listed, and successively smaller costs for violating the less significant constraints. The total cost gives the *fitness* of the proposed schedule. The goal is to determine an optimum schedule, if possible one with zero cost, within a reasonable period of time. More reasonably, we would seek an acceptably small cost (a playable schedule) in an acceptable length of time.

The scheduling committee must have some algorithm to create the schedule. One approach (recommended only in the spirit of brainstorming) is to

generate a huge variety of schedules, evaluate them, and choose the best. Certainly, this naive approach will only be suitable for tournaments with a very small number of teams. Alternatively, the committee could have the individual teams select their own playing schedule, inviting them in some order (team standing or lot). Teams or matches choosing early would have the largest choice of playing fields and times; those choosing later would have to make the best out of whatever is left. This second approach is an example of a greedy algorithm (discussed further in section 2), and it may or may not create a workable or repairable schedule (schedules can be *mutated* by making small changes; perfection *may* be thus achieved).

Genetic algorithms (GAs) [2] apply to such situations: problems without algorithms but with evaluation methods. A GA works with a *population* of problem solutions (such as the output of a random schedule generator), and that population is manipulated by a process inspired by *selective breeding*—the relatively more fit schedules 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* schedules from the pieces of the chosen *parent* schedules. 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 such that $f(x)$ is maximized (for a problem where calculus is not applicable), then we could let the individuals be represented by fixed-length strings of binary or decimal digits. Fitness evaluation is straightforward (assuming 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.

Other issues that the GA designer must address are:

- population size,
- mutation rate,
- breeding algorithm,
- population size management (that is, how are individuals 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 Greedy Algorithms

"The greedy algorithm does what is locally best without regard to future consequences [5]."

Some version of a greedy algorithm (it will, of course depend upon the particular problem this term actually means) may work perfectly for certain problems, and it may, by chance, work for some others. Problems for which it guarantees the right answer are

- *Minimum spanning trees*: given a connected graph with costs associated with each edge, find the minimum cost connected subgraph. Prim's and Kruskal's algorithms for solving this problem are greedy; each builds its solution by adding the least expensive edge it can.

- *Change making*: given an amount of money to represent and a currency system to represent it with, represent that value with the smallest number of coins. The greedy algorithm iteratively uses the largest possible coin to reduce the value down to zero. This succeeds with a variety of currency systems, such as any system for which every denomination is a multiple of the next smallest, down to 1. This principle extends to the representation of numbers in some radix (two, ten, etc.).

  For US currency without the nickel, the greedy algorithm fails; it give us $30 = 25+1+1+1+1+1$ rather than the optimum $30 = 10+10+10$.

  The greedy algorithm also works for currency systems based on the Fibonacci sequence [3].

A huge variety of problems could be solved be solve by a greedy algorithm—but only by extreme luck: the order of building the solution must be fortuitous. Optimum solutions for the problems discussed in section 8 conceivably come from a greedy algorithm, if only we start with the right permutation.

3 Ordered Greed—A Natural Hybrid

The ordered greed algorithm (OG) uses a population of permutations (actually, we prefer to work with *signatures* of populations; see section 7), and uses each permutation to drive a greedy algorithm towards a trial solution for the given problem.

For many of the problems we have considered (in particular, the N Queens puzzle and soccer tournament scheduling) the order-based GA for which the individual permutations were the solutions (the list of Queens' column positions for each row or the soccer field-and-time assignments for each match) performed miserably. We found solutions for these two problems only after an unreasonably long time or for unrealistically small problem instances. It appeared that selective breeding had no value, and that a sequence of randomly chosen individuals would just as quickly solve the problem. OG performed dramatically better.

4 N Queens

The N Queens puzzle is an old chestnut (known to Gauss) which captures the essence of a lot of the difficulty of scheduling and assignment problems. A variety of simple solutions have been discovered for easily placing N non-attacking Queens on an $N \times N$ chess board, for any $N \geq 4$. It stretches the point to call N Queens an NP-complete problem, but it is an instance of the maximum independent set problem, which is an NP-complete problem. Variations are, of course, easy to come by: chess board locations can have associated values, or some locations could be forbidden.

Although there are many known ways to solve this problem for any $N \geq 4$, it remains a popular goal for any new heuristic problem solving method. Because it can be cast as an instance of MIS, the Maximum Independent Set problem, N Queens appears to be closely related to the NP-complete problems.

We have experimented with various approaches to solving the N Queens problem, and the GA approach using a population of permutations (that is, an individual permutation, $\langle a_0, a_1, a_2, \cdots, a_{N-1} \rangle$ of $\langle 0, 1, 2, \cdots, N-1 \rangle$ with the meaning that the Queen in row k is on column a_k) appears to be only marginally better than unadulterated random search. The fitness function for this GA would be the number of Queens not attacked by a Queen on a

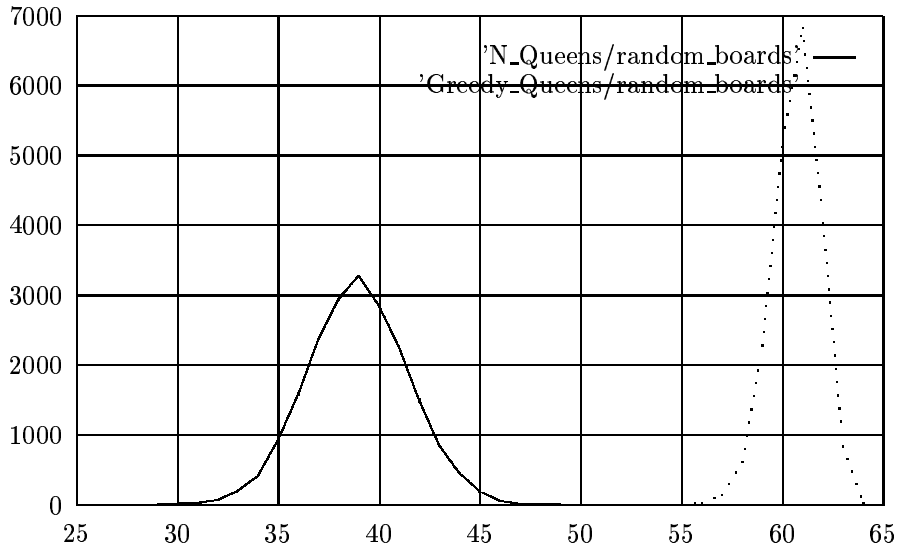

Figure 1: Ordered greed placement vs. random placement for 64 Queens.

higher row.

The ordered greed approach to this problem uses a permutation, $\langle a_0, a_1, a_2, \cdots, a_{N-1} \rangle$, as follows. The Queens are placed, in order, into columns $a_0, a_1, a_2, \cdots, a_{N-1}$, with each Queen placed as close to the top (i.e., smallest row number) so that it does not attack any Queen previously placed. If previous Queens attack all positions for a given column, then no Queen can be placed in that column. The fitness of the permutation is the number of Queens successfully placed on the $N \times N$ board.

Any permutation which is, itself an N Queens solution will also be the solution developed by ordered greed using that same permutation. Thus ordered greed applies in the described manner to N Queens.

To demonstrate the relative efficacy of ordered greed, we used both approaches to generate several thousand placements for 64 Queens; fitnesses are shown in Fig. 1.

We ran the unadorned GA and the ordered greed GA each with 100 different random seeds, and allowed them to perform 2,050 fitness evaluations—the initial population followed by 1,000 creations of a pair of children. We used a steady state algorithm, choosing parents with tournaments of size two, uniform crossover, and a mutation rate of 0.001. Ordered greed discovered

solutions in 57 of the trials, in an average of 563.5614 fitness evaluations; the 43 cases that did not achieve a fitness of 256 all achieved a fitness of 255. The unadorned GA was only able to discover Queens' placements with a maximum fitness of 187 (15 cases) and an average best fitness of 182.11.

An additional heuristic that often applies to ordered greed is inspired by Warnsdorff's heuristic for the Knight's tour problem. Warnsdorff's heuristic is presented in Ball's classic recreational mathematics book [1] (Ball cites Warnsdorff as [7]) in terms of a method for improving a greedy algorithm to find a Knight's tour on a chess board. The trick is to keep track of how many possible Knight moves there are from each board position, and to move the traveling Knight to the square with the fewest possibilities. (In the present author's experience, it does not matter how ties are broken nor where the tour begins; a Knight's tour is always found.)

For the N Queens problem, this translates to the requirement to place Queens on the columns that have the fewest number of unattacked positions. When more than a single column has the minimum, use the given permutation to decide which column to choose.

The ordered greed GA augmented by Warnsdorff's heuristic discovered perfect solution to the 256 Queens problem for all 100 attempts, but this approach found the solutions in the initial population. The average number of evaluations was 14.65.

5 Soccer Tournament Scheduling

Every June, the town of Chili, New York, hosts a two-day soccer tournament. The 1997 tournament consisted of 131 teams in 10 divisions (according to age group and sex). Each team played three or four matches (depending on the division size), for a total of 209 matches. They had 14 playing fields of five different sizes (not all of which were lighted for evening games) and 17 playing times, Friday evening to Sunday afternoon (leaving time for playoffs).

A small committee spent 50–100 person-hours designing a tournament schedule that satisfies the requirements we enumerated in section 1. The goal of Bill Gustafson's Masters project [4] was to automate the manual scheduling. In addition to completing his MS, several benefits would accrue from this automation:

- The constraints would be verified accurately. Special constraints (e.g., a team that cannot attend on Friday evening) can be easily kept track

of.

- The addition or deletion of teams at the last minute would be less painful.
- A computerized schedule could include automatic production of printed tournament schedules for the audience, with specialized schedules for each team and field.

In the course of this work, the constraint list became, for the first time, fully described, as past members of the scheduling committee criticized the computer produced schedules.

The algorithm for this project went through several versions before it became usable. In all cases, as in the manual scheduling process, the list of matches was predetermined. Divisions with four or five teams would have all six or ten possible matches, and divisions with six teams would have nine matches. The first version of the algorithm worked with a population of 600 permutations of the 209 matches. Parents were selected via a two-element tournament, and the mutation rate was 0.01. Fitness was determined by placing the permutation on the field-and-time grid and evaluating the constraints. These experiments were terminated after 10–12 hours; they all showed no hope of generating a schedule even meeting the hard constraints—no schedule was even playable.

A second attempt worked with the age group subsets of the tournament and the appropriate size fields for those age groups. These problems were approximately one fifth the size of the whole problem, (Artificially small problems were successful for the first approach.) A version of ordered greed came into play in this attempt: instead of simply putting each match on the field-and-time grid in order, it was put on the first open field and time that both teams could play, given the previous assignments. Thus most hard constraints were dealt with early. This attempt worked much better, and it developed some playable schedules for the smaller age groups, but it failed to find acceptable schedules which gave the teams the required two-period rest time between matches.

The third attempt finally produced tournaments at least as good as those produced manually. In this attempt, the ordered greed step searched for the *best* time and field to play each match in order, considering all the constraints. This algorithm is quite time-consuming, but breaking the problem into the five age groups and treating them separately speeds the process up, and is the

appropriate granularity in case the schedule has to be re-derived for teams that drop out at the last minute.

6 Warnsdorff's Heuristic Tempers Greed

For the problems we have considered, this means keeping track of the possible moves or assignments for each of the unassigned steps and processing the hardest-to-assign step next, breaking the ties with the GA individual's permutation. From our problem list, this entails the following:

- *The N Queens puzzle.* Place the Queen in the row with the most positions under attack.
- *Matching.* Match the item with the fewest remaining possible pairings. This could be extended to intelligently choosing *which* of those pairing to choose.
- *Graph coloring.* Color the country or vertex that has the fewest remaining legal colorings (or the largest numbers of differently colored neighbors).
- *Ramsey theory coloring.* Color the edge with the smallest number of valid remaining colorings.
- *Soccer tournament scheduling.* Assign the match that has the smallest number of valid remaining assignments.

7 Permutation-Preserving Crossovers

As we have seen, a wide variety of applications require a population of permutations. Our purpose in this section is to discuss how to generate and deal with permutations—generating them is straightforward misunderstood, and breeding must be done with some care because simpleminded crossovers destroy the permutation property. (Goldberg [2] discusses three such algorithms.)

Here is a common but terrible way to generate a permutation of the numbers $\{0, 1, 2, \ldots, N-1\}$ (to protect the guilty, no citations will be given):

```
for( i = 0; i < N; i++ ) {
    do {
        m = a random number in 0,1,2,...,N-1
    } while( m is not yet in List )
    List[i] = m;
}
```

The problem is possible interminable searching for the last few numbers that have not yet been generated.

A superior algorithm is:

```
for( i = 0; i < N; i++ ) { List[i] = i; }
for( i = 0; i < N; i++ ) {
    m = a random number in 0,1,2,...,N-1-i
    interchange List[i] with List[i+m]
}
```

The second algorithm suggests using the notion of the *signature* of a permutation rather than the permutation itself. A list $S = \langle s_0, s_1, s_2, \ldots, s_{N-1} \rangle$ is a signature of a permutation in case $0 \leq s_i \leq i$ for all i. If `Sig` is a signature, we may modify the permutation generation algorithm as follows:

```
for( i = 0; i < N; i++ ) { List[i] = i; }
for( i = 0; i < N; i++ ) {
    interchange List[i] with List[i+Sig[i]]
}
```

A second approach to decoding permutations from signatures uses the concept of *places*. In this interpretation, we regard a signature $S = \langle s_0, s_1, s_2, \ldots, s_{N-1} \rangle$ as dictating that 0 is in position s_0, then 1 is in position s_1 of the remaining $N-1$ positions, then 2 is in position s_2 of the remaining $N-2$ positions, and so on. For example, with $N = 8$, the signature, $S = \langle 5, 4, 5, 3, 2, 2, 0, 0 \rangle$ gives us the permutation $S = \langle 6, 7, 4, 3, 1, 0, 5, 2 \rangle$ The most straightforward signature to permutation decoding algorithm simply steps through the open position in a permutation to place each value; this algorithm has time complexity $\mathcal{O}(N^2)$. A more efficient conversion process arises as follows. Consider the problem as one of converting a list of N label-position pairs from the form that gives the position for each value:

labels	0	1	2	3	4	5	6	7
position	5	4	5	3	2	2	0	0

to the form with all zero positions—i.e., the permutation can be read off directly:

labels	6	7	4	3	1	0	5	2
position	0	0	0	0	0	0	0	0

To manipulate the first list into the second, we use the following rule: whenever an adjacent pair

a	b
p	q

such that $p > q$, we may interchange them

a	b
q	p-1

For example, if we interchange two pairs, say those with labels 0 and 1, we get

labels	1	0	2	3	4	5	6	7
position	4	4	5	3	2	2	0	0

The position, 5, which was associated with label 0, reduced to position 4. This observation naturally suggests an operation based on insertion sort (also of complexity $\mathcal{O}(N^2)$) to reduce all the positions to zero; this immediately leads to an operation based on merge sort, with complexity $\mathcal{O}(N \log N)$.

7.1 Advantages of Using Signatures

We propose that an order-based GA can very easily work with a population of signatures rather than a population of permutations:

- Signatures are easy to generate (they may be regarded as the first step for generating permutations).
- Signatures are amenable to mutation—simply regenerate one of the entries.
- Signatures may be bred using two-point crossover—the signature property of the parents guarantees that the children will be valid signatures.
- There are straightforward algorithms to convert permutations to signatures; this enables permutation-based or greedy mutations (Lamarkian evolution?).

8 An OG Applications Sampler

Ordered greed is an appropriate approach to a wide variety of combinatorial optimizations 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 identified. Many of these problems—but not all—are NP-complete.

- **The job assignment problem.** If there are N jobs to be done and N people to be assigned each to one job, then this problem would be *the N Rooks puzzle*, and any permutation could be used to place N non-attacking rooks on an $N \times N$ chess board would represent a solution (rows correspond to jobs, columns to people, and Rooks to assignments). The plot thickens in case the individuals have different skills or training costs associated with each possible job assignment. Update the chess board model by associating a payoff value with each square or assignment. Now we want to place N non-attacking Rooks on squares to give the largest possible total payoff. (See [5] for a variety of related problems.)

- **Matching.** Given a graph $G = (V, E)$, a perfect matching is a spanning subset $M \subset E$ which cover each vertex exactly once; i.e., for each $v \in V$ there is exactly one $w \in V$ such that $(v, w) \in M$. The edges of G could represent compatible roommates, matching power amplifier tubes, or chemical bonds; them M would be roommate assignments, set of tube pairs, or a Kekulé structure.

- **Graph coloring.** Given a graph $G = (V, E)$, a coloring of its vertices is an assignment $c : V \to K$, where K is a finite set of colors, and for every edge $(v, w) \in E$, $c(v) \neq c(w)$. If G represents a map of countries (vertices are the individual countries and edges indicate that two countries share a border segment) c provides a coloring using $||K||$ colors. Map makers who wish to minimize $||K||$ (using only four colors for maps on the Earth or seven for maps on a donut) must solve this graph coloring puzzle.

- **Ramsey theory coloring.** Ramsey theory tells us that we must have a monochromatic triangle in any coloring of the edges of the complete

graph K_N in case:

$N > 5$ and we have only two colors,

$N > 16$ and we have only three colors,

$N > 65$ and we have only four colors.

The first two results are known to best possible; there are triangle-free two-colorings of the edges of the edges of K_5 and three-colorings of the edges of the edges of K_{16}, although the latter is a challenge to find through trial and error (but easily found by the method of ordered greed). The third result is not known to be best possible; the largest triangle-free four-colored K_N thus found is K_{50} [6]; this is clearly a challenge for ordered greed.

- **Bin packing.** Given a set of objects, each with a size, pack them into a given set of containers, each with a given capacity.

  A greedy algorithm would take each of the items, in turn, and place it into the first container with enough remaining capacity for it. If the given problem has a solution, the greedy algorithm will find it, given the right permutation.

- **Satisfiability.** Given a Boolean expression in conjunctive normal form, (that is, the logical product of many clauses, each clause of which is the logical sum of variables and their negations), locate a truth-value assignment to the variables that satisfies the expression.

  The greedy algorithm would use each variable, in turn, to satisfy the first clause in which it appears, and then either satisfying or simplifying following clauses in which it appears.

  Warnsdorff's heuristic would apply to choose variables that appear in clauses with fewer (remaining) variables—i.e., the clauses that would be the most difficult to satisfy.

- **Pentominoes.**

- **Multiprocessor scheduling.**

- **Faculty teaching assignments.**

- **Airline crew assignments — set partition.**

References

[1] W. W. Rouse Ball and H. S. M. Coxeter. *Mathematical Recreations & Essays.* University of Toronto Press, 1974.

[2] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[3] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics.* Addison-Wesley, Reading, MA, 1989.

[4] William Gustafson. Building a soccer tournament schedule using a genetic algorithm. Master's project, Rochester Institute of Technology, 1998.

[5] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley & Sons, 1988.

[6] Stanislaw P. Radiszowski. Samll ramsey numbers. *The Electronic Journal of Combinatorics,* `http://www.combinatorics.org/`, 1998.

[7] H. C. Warnsdorf. *Des Rösselsprunges einfachste und allgemeinste Lösung.* Schamlkalden, 1823.