Genetic Algorithm

1 Background

There are many optimization problems that, quite simply, cannot be solved in a reasonable amount of time – even on the fastest computers in the foreseeable future[1]. When faced with such problems engineers (and computer scientists) have abandoned[2] their search for the optimum and look for a pretty good solution instead. The field of research studying such "pretty good solution" algorithms is called "metaheuristics", or more simply heuristic algorithms[3].

My favorite algorithm from the class of metaheuristics is called the *genetic algorithm*. This algorithm emulates biological natural selection[4] to find "pretty good solutions". The algorithm is easy to code, widely applicable, and remarkably successful across many problems in civil and geo engineering.

2 Genetics redacted

We formulate our problem so that all possible solutions take the form of a large set of binary switches. Borrowing vocabulary from genetics[5], we call each of these binary switches an *allele*, and we call a large set of alleles deigning a possible solution a *chromosome*. That is, a *chromosome* as a $(m \times 1)$ vector of 1's and 0's.

3 Outline of a basic genetic algorithm

The following outline is adapted from:

`http://www.obitko.com/tutorials/genetic-algorithms`.

1. (Initialize) Generate a random starting population of chromosomes (each chromosome is a suitable candidate solution for the problem). While the selection of this initial population can include prior knowledge, it is important to incorporate a high degree of randomness at the start. I have found that a generation size of between 20 and 100 to work well. This initial population defines our current generation.

2. (Loop) Repeatedly evaluation and generate new generations until an end condition is reached.

   (a) (Fitness) Evaluate the fitness, $f(x)$, of each chromosome, x, in the current generation. This is a slight misnomer, in that the smaller values are more fit than the larger values – i.e. we want to minimize $f(x)$.

[1]See, for example, the list of NP-complete problems at `http://en.wikipedia.org/wiki/List_of_NP-complete_problems`.

[2]This brings to mind, Ashley Brilliant's quote: "I have abandoned my search for truth and am now looking for a good fantasy."

[3]See, for example, `http://en.wikipedia.org/wiki/Metaheuristic`.

[4]See, for example, `http://en.wikipedia.org/wiki/Natural_selection`.

[5]For the real definition of an allele see, for example, `http://en.wikipedia.org/wiki/Allele`. Similarly, for the real definition of a *chromosome* see, for example, `http://en.wikipedia.org/wiki/Chromosome`.

   (b) (Next generation) Create a new generation by repeating following steps until the new generation is complete.
      i. (Elitism) The best of one generation are replicated in the next generation. This guarantees that the solution never gets worse from one generation to the next. I typically replicate the best one or two of each generation.
      ii. (Selection) Select two parent chromosomes from the current population according to their fitness (the better fitness, the bigger chance to be selected). The mapping of the fitness score to the probability of selection is defined by the *selection option* and the *fitness scaling.* I have found the "roulette wheel" approach[6] to be a robust approach.
      iii. (Crossover) With a crossover probability cross over the parents to form a new offspring (i.e. member of the next generation). I have found that a 50/50 simple coin flip to determine which parent contributes an allele, on an allele-by-allele basis, to be simple to implement and effective.
      iv. (Mutation) With a mutation probability mutate new offspring at each locus (position in chromosome). This probability should be relatively small. I typically use a value between one and three percent chance.
   (c) (Replace) Throw out the old generation and replace it with the new generation.

3. (Loop) If we are not done, go back to step 2.

4. (Return the best so far)

4 Coding a basic genetic algorithm

The fingerprint for the function is going to look something like the following.

```
function [best, score, flag] = genetic( fitness, M )
```

where "fitness" is a function handle, and "M" is the number of binary alleles in a chromosome.

1. (Start)

```
% Initialize the "roulette wheel" selection with a weight of 1/N.
div = 1 ./ [1:GENERATION_SIZE];
div = cumsum(div)/sum(div);
```

```
% Initialize the population -- each row is a chromosome.
oldpop = rand( GENERATION_SIZE, M ) < 0.5;
```

2. (Loop)

```
% Loop through the specified number of generations.
for generation = 1:MAXIMUM_NUMBER_OF_GENERATIONS
```

[6]See, for example, http://en.wikipedia.org/wiki/Fitness_proportionate_selection.

(a) (Fitness)

```
% Evaluate and sort the population.
[scores,I] = sort( fitness( oldpop ) );
oldpop = oldpop(I,:);
```

(b) (Next generation)

i. (Elitism)

```
% Elitism
newpop = oldpop;
```

ii. (Selection)

```
% Sexual reproduction with mutation.
for i = NUMBER_OF_ELITE_MEMBERS+1:GENERATION_SIZE
    mom = find( div >= rand(1), 1, 'first' );
    dad = find( div >= rand(1), 1, 'first' );
```

iii. (Crossover)

```
    selection = rand(1,M) < 0.5;
    newpop(i,selection)      = oldpop(mom,    selection);
    newpop(i,not(selection)) = oldpop(dad,not(selection));
```

iv. (Mutation)

```
    mutate = rand(1,M) < PROBABILITY_OF_MUTATION;
    newpop(i,mutate) = not( newpop(i,mutate) );
```

```
end
```

(c) (Replace)

```
% Replace the old generation with the new
oldpop = newpop;
```

3. (Loop)

```
end
```

4. (Return the best so far)

```
% Return the best so far.
[scores,I] = sort( fitness( oldpop ) );
oldpop = oldpop(I,:);

best  = oldpop(1,:);
score = scores(1);
flag  = 1;
```