

Reference Materials

The Internet. Use a net browser such as Netscape. Do a search on “genetic algorithms.” You will find more information than you care to find. The search may be narrowed by specifying additional search words. Some of the better web sites to start exploring include

<http://www.aic.nrl.navy.mil>

<http://alife.santafe.edu>

Also available from the internet is *The Hitch-Hiker’s Guide to Evolutionary Computation* (Heitkotter and Beasley, 1994). This excellent up-to-date resource lists everything from where to get free software to many frequently asked questions (FAQ) about evolutionary programming.

Articles. Many of the major scientific and engineering journals have published introductory articles on genetic algorithms.

Holland’s article in *Scientific American* (Holland, 1992).

Books. If you are interested in the theory of genetic algorithms, Holland (1975), Goldberg (1989), Mitchell (1996), Whitley (1993), and Schwefel (1995) will satisfy you. The other books [including Goldberg (1989)] are more application oriented.

Journals. Several journals publish articles on genetic algorithms.

Some include: *Evolutionary Computation*

IEEE Expert Intelligent Systems & Their Applications

IEEE Transactions on Systems, Man, and Cybernetics

BioSystems

Complex Systems

Machine Learning

Short Courses. Several institutions offer short courses on genetic algorithms, including Georgia Institute of Technology and UCLA. In addition, many conferences offer short courses and tutorials on various aspects of genetic algorithms.

Software. There is a lot of free software available. For the latest, consult Heitkotter and Beasley (1994). Some examples include:

MATLAB Genetic Algorithm Toolbox

Genetic Search Implementation System (GENESIS)

Genetic Algorithm for Numerical Optimization for Constrained Problems (GENOCOP)

Better to Use Genetic Systems (BUGS)

GENEsYs

Tool Kit for Genetics-Based Applications (TOLKIEN)

Conferences. There are several conferences each year devoted to evolutionary programming, including

Conference on Genetic Programming

IEEE Conference on Evolutionary Computation

International Conference on Genetic Algorithms

Annual Conference on Evolutionary Programming

Foundations of Genetic Algorithms

Parallel Problem Solving from Nature

PSEUDOCODES

Some guidelines for reading the pseudocode:

- A matrix is a variable with all capital letters
- A vector is a variable with the first letter capitalized
- A scalar is all lowercase letters
- A function provided by the user is in boldface
- A function described by pseudocode is italicized
- % indicates a program comment

User-provided subroutines are described below:

- `random(r, c)`—generates an $r \times c$ matrix of uniformly distributed random numbers.
- `round(*)`—rounds numbers to nearest integer.
- `costfunction(CHROMOSOMES)`—returns a column vector with the cost associated with each chromosome or row in the matrix CHROMOSOMES.
- `sort(Cost, CHROMOSOMES)`—sorts the Cost vector and associated chromosomes from lowest cost in row 1 to highest cost in the last row. It also truncates the vector and matrix to the first popsize rows.
- `min`—finds the minimum value of the vector.
- `mean`—finds the average value of the vector.
- `std`—finds the standard deviation of the elements in the vector.
- `roundup`—rounds numbers to next highest integer.

Most math packages provide some form of the above functions. Canned subroutines for programming languages are generally available for most of these functions as well.

The main genetic algorithm and pairing function are the same for the binary and continuous versions. However, the mating and mutation functions are quite different, so two versions are provided.

Pseudocode for a Binary Genetic Algorithm

```
%-----  
% Define variables  
%-----  
maxiterations=?           % maximum number of iterations  
ipopsi= ?                 % population size of generation 0  
popsi=?×ipopsi           % population size of generations 1 through. . .  
keep=?×popsi             % number of chromosomes kept for mating  
bits=?                   % total number of bits in a chromosome  
mutaterate=?            % mutation rate  
  
-----  
%  
% Create the initial population, evaluate costs, and sort  
%-----  
CHROMOSOMES=round(random(ipopsi,bits))  
                % matrix of random 1s & 0s  
  
%  
% Let the generations begin!  
% Cost - vector containing the costs  
% sort - sorts & truncates costs & chromosomes  
  
%-----  
gen#=0                 % initial generation  
quit='no'              % convergence check  
while gen#<maxiterations & quit='no'  
    gen#=gen#+1        % increment the generation number  
    Cost=costfunction(CHROMOSOMES)  
    [Cost,CHROMOSOMES]=sort(Cost,CHROMOSOMES)  
  
%  
% Evaluate cost statistics  
%-----  
Mincost(gen#)=min(Cost)           % minimum cost  
Meancost(gen#)=mean(Cost)        % mean cost  
Stndcost(gen#)=std(Cost)        % standard deviation of cost  
  
-----  
%  
% The chromosomes are paired and offspring are produced  
%-----  
[Mom Dad]=pair(CHROMOSOMES,Cost,keep,popsi)  
CHROMOSOMES=matebin(Mom,Dad,CHROMOSOMES,keep,  
                popsi,bits)  
  
-----
```

```
-----  
%  
% mutate the population with (popsize  $\times$  N  $\times$  mutaterate) mutations  
%-----  
CHROMOSOMES = mutatebin(CHROMOSOMES, mutaterate,  
                        popsize, bits)  
  
-----  
%  
% Check for convergence  
%-----  
if Mincost(gen#) < ? and/or Meancost(gen#) < ?  
    and/or Stndcost(gen#) < ?  $\Rightarrow$  quit = 'yes'  
end
```

Pseudocode for Pairing

```
function [Mom,Dad]=pair(CHROMOSOMES,Cost,keep,popsize)
```

```
%-----
```

```
% Selects one of three options for calculating the probability
```

```
%-----
```

```
replacements=(popsize-keep)/2    % # CHROMOSOMES needing
                                  % replaced
```

$$\text{Prob}(n) = \left\{ \begin{array}{l} \frac{n}{\sum_{r=1}^{\text{replacements}} r} \\ \frac{|\text{cost}(n)|}{\sum_{r=1}^{\text{replacements}} |\text{cost}(r)|} \\ \frac{\text{cost}(n) - \min \{ \text{cost}(\text{replacements} + 1) \}}{\sum_{r=1}^{\text{replacements}} |\text{cost}(r) - \min \{ \text{cost}(\text{replacements} + 1) \}|} \end{array} \right.$$

```
% cumulative probabilities
```

```
Odds=[0, Prob(1), Prob(1)+Prob(2), ..., \sum_{n=1}^{\text{replacements}} Prob(n)]
```

```
%-----
```

```
% Roll dice for parents
```

```
%-----
```

```
Pick1=random(1,replacements)    % vector of random numbers for Mom
```

```
Pick2=random(1,replacements)    % vector of random numbers for Dad
```

```
%-----
```

```
% finds the two mates
```

```
%-----
```

```
ic=1                                % initialize counter
```

```
while ic<replacements                % counter must be less than
                                      % replacement number
```

```
%-----
```

```
%
```

```
% when one of the random Picks falls inside a cumulative probability
% bin, the chromosome associated with that bin is selected as a parent
```

```
%-----
```

```
for id=2:keep+1
```

```
if Pick1(ic)<Odds(id) & Pick1(ic)>Odds(id-1) → Mom(ic)=id-1
```

```
if Pick2(ic)<Odds(id) & Pick2(ic)>Odds(id-1) → Dad(ic)=id-1
```

```
end
```

```
ic=ic+1                                % increment counter
```

```
end
```

Pseudocode for Binary Mating

```
function CHROMOSOMES = matebin(Mom,Dad,CHROMOSOMES,  
                                keep,popsize,bits)
```

```
-----  
%  
% selects a crossover point  
% roundup rounds to next highest integer  
%-----  
Xpoint = roundup{(N-1) × random(1,M)}  
-----  
%  
% row indx contains first offspring  
% row indx + 1 contains second offspring  
% mom - vector containing row numbers of first parent  
% dad - vector containing row numbers of second parent  
%-----  
for ic = 1:popsize  
    indx = 2 × (ic-1) + 1  
    CHROMOSOMES(keep + indx, 1 → popsize) =  
        [CHROMOSOMES(Mom(ic), 1 → Xpoint(ic)),  
         CHROMOSOMES(Dad(ic), Xpoint(ic) + 1 → popsize)]  
    CHROMOSOMES(keep + indx + 1, 1 → popsize) =  
        [CHROMOSOMES(Dad(ic), 1 → Xpoint(ic)),  
         CHROMOSOMES(Mom(ic), Xpoint(ic) + 1 → popsize)]  
end
```

Pseudocode for Binary Mutation

```
function CHROMOSOMES = mutatebin(CHROMOSOMES, mutaterate,  
                                popsize, bits)  
  
-----  
%  
% Inside a loop iterating over the number of mutations, a random  
% bit in the population is selected and changed from a 1 to a 0 or  
% from a 0 to a 1.  
%-----  
#mutations = roundup(popsize × bits × mutaterate) % number of mutations  
  
for ic = 1 → #mutations  
    row = roundup((popsize-2) × random(1,1)) + 1 % random row  
    col = roundup((bits-2) × random(1,1)) + 1 % random column  
    CHROMOSOMES(row,col) = CHROMOSOMES(row,col) - 1  
    % mutation  
end
```

Pseudocode for a Continuous Parameter Genetic Algorithm

```
%-----  
% Define variables  
%-----  
popsize=?×ipopsize    % population size of generations 1 through. . .  
maxiterations=?      % maximum number of iterations  
ipopsize=?           % population size of generation 0  
popsize=?×ipopsize   % population size of generations 1 through. . .  
keep=?×popsize       % number of chromosomes kept for mating  
pars=?               % total number of parameters in a chromosome  
mutaterate=?         % mutation rate  
hi=?                 % maximum parameter value  
lo=?                 % minimum parameter value  
  
-----  
%  
% Create the initial population, evaluate costs, and sort  
%-----  
CHROMOSOMES=(hi-lo)×random(ipopsize,bits)+lo  
                % matrix of random 1s & 0s  
  
-----  
%  
% Let the generations begin!  
%-----  
gen#=0              % initial generation  
quit='no'           % convergence check  
while gen#<maxiterations & quit='no'  
gen#=gen#+1        % increment the generation number  
Cost=costfunction(CHROMOSOMES)  
[Cost,CHROMOSOMES]=sort(Cost,CHROMOSOMES)  
  
-----  
%  
% Evaluate cost statistics  
%-----  
Mincost(gen#)=min(Cost)    % minimum cost  
Meancost(gen#)=mean(Cost)  % mean cost  
Stndcost(gen#)=std(Cost)   % standard deviation of cost  
  
-----  
%  
% The chromosomes are paired and offspring are produced  
%-----
```

```
[Mom Dad]=pair(CHROMOSOMES, Cost, keep, popsize)
CHROMOSOMES=matecon(Mom, Dad, CHROMOSOMES, keep,
                    popsize, pars)
```

```
-----
%
% mutate the population with (popsize×N×mutaterate) mutations
```

```
%-----
CHROMOSOMES=mutatecon(CHROMOSOMES, mutaterate, popsize,
                    pars, hi, lo)
```

```
-----
%
% Check for convergence
```

```
%-----
if Mincost(gen#)<? and/or Meancost(gen#)<? and/or Stndcost(gen#)
    <?⇒quit='yes'
```

```
end
```

Pseudocode for Continuous Parameter Mating

```
function CHROMOSOMES = matecon(Mom,Dad,CHROMOSOMES,  
                                keep,popsize,pars)
```

```
-----  
%  
% row indx contains first offspring  
% row indx + 1 contains second offspring  
% mom - vector containing row numbers of first parent  
% dad - vector containing row numbers of second parent  
%-----  
for ic = 1 → popsize  
  indx = 2 × (ic - 1) + 1  
  alpha = roundup{random × pars}  
  beta = random  
  CHROMOSOMES(keep + indx, alpha)  
    = CHROMOSOMES(Mom(ic), alpha)  
      - beta × [CHROMOSOMES(Mom(ic), alpha)  
                - CHROMOSOMES(Dad(ic), alpha)]  
  CHROMOSOMES(keep + indx + 1, alpha)  
    = CHROMOSOMES(Dad(ic), alpha)  
      + beta × [CHROMOSOMES(Mom(ic), alpha)  
                - CHROMOSOMES(Dad(ic), alpha)]  
  if alpha > 1 & alpha < pars  
    CHROMOSOMES(keep + indx, alpha + 1 → pars)  
      = CHROMOSOMES(keep + indx + 1, alpha + 1 → pars)  
    CHROMOSOMES(keep + indx + 1, alpha + 1 → pars)  
      = CHROMOSOMES(keep + indx, alpha + 1 → pars)  
end  
end
```

Pseudocode for Continuous Parameter Mutation

```
function CHROMOSOMES = mutatecon(CHROMOSOMES,maxval,  
                                mutaterate,popsize,pars,hi,lo)  
-----  
%  
% Inside a loop iterating over the number of mutations, a random  
% parameter in the population is selected and replaced by a new  
% random parameter.  
%-----  
#mutations = roundup(popsize × pars × mutaterate)      % # mutations  
  
for ic = 1 → #mutations  
row = roundup(popsize × random) + 1                    % random row  
col = roundup(pars × random)                          % random column  
CHROMOSOMES(row,col) = (hi - lo) × random + lo      % mutation end
```