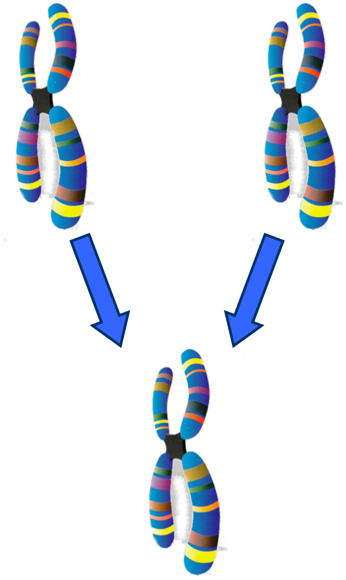
Genetic Algorithms: part I

The basic fundamentals of a simple genetic algorithm



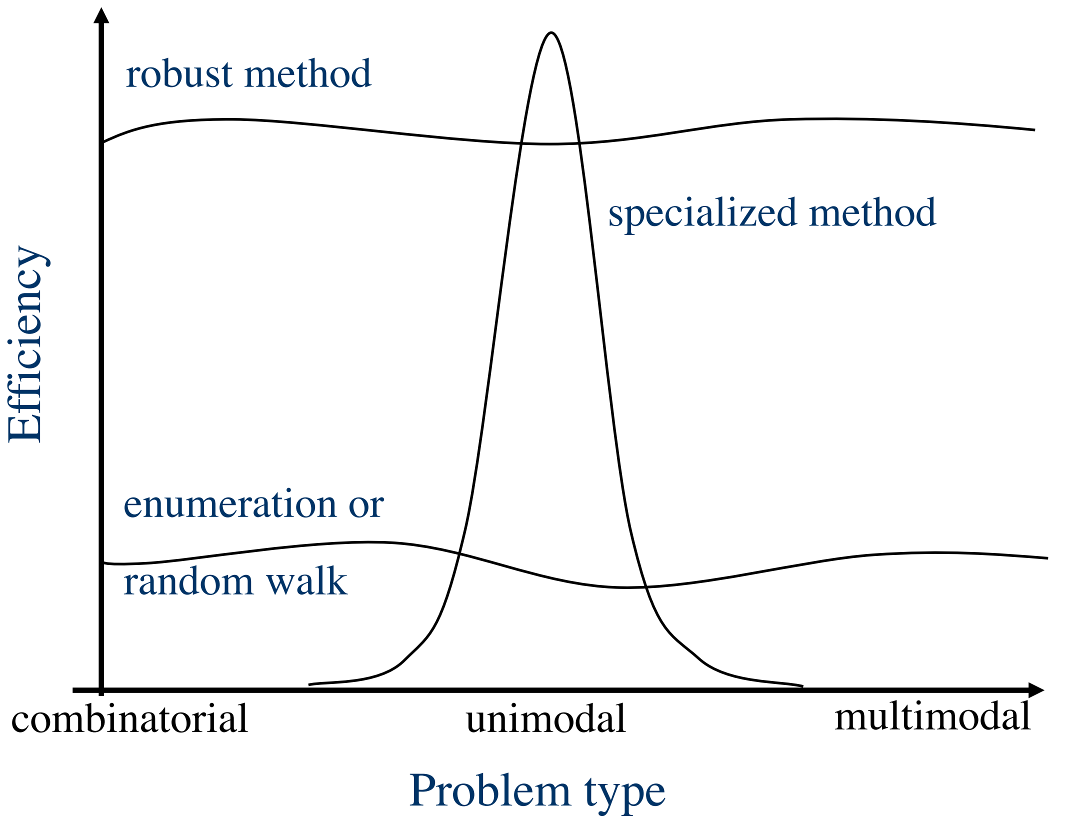
Contents

- GA and other evolutionary techniques
- The basic principle – How does it work?
- Representation of optimization variables
- Selection of parents
- Reproduction – creation of children
- Replacement – insertion of children into the population

Evolutionary Computation

- Genetic Algorithms
- Evolutionary strategies
- Evolutionary programming
- Genetic programming
- Simulated annealing
- Ant colonies
- Particle swarm optimization
- ….

Method efficiency



After Goldberg 1989

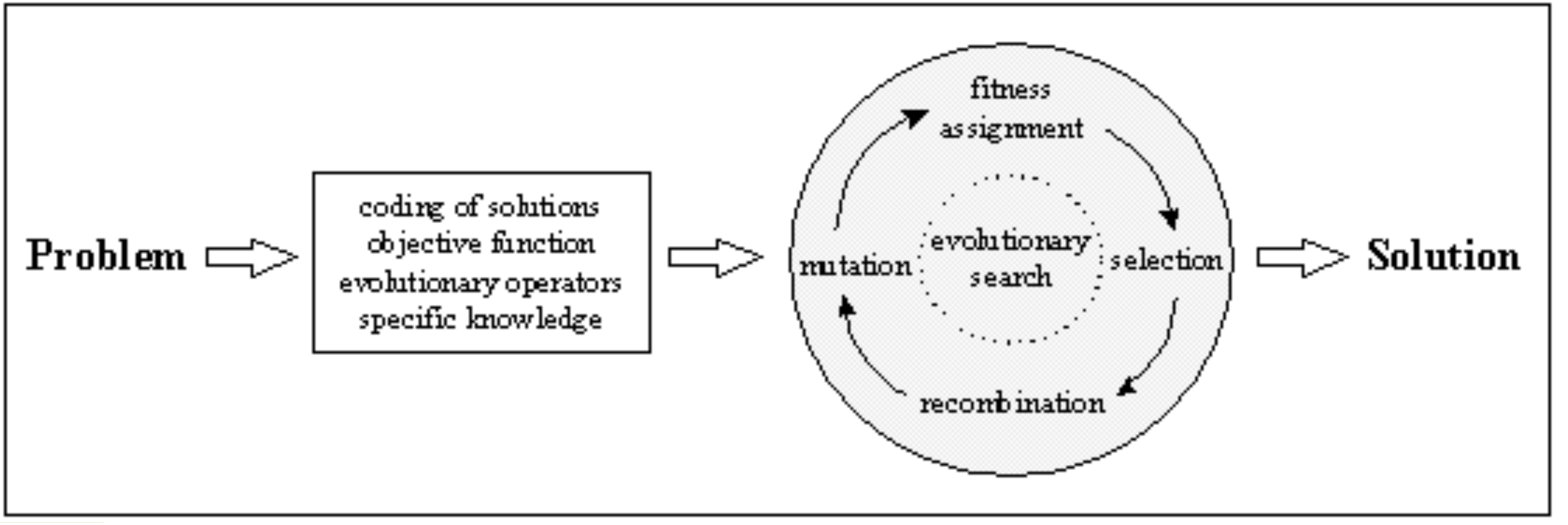
GA – the idea behind

- Mimic the evolution of nature
  - Creative and innovative
  - Complexity
- Code the optimization variables into a chromosome.
- Create a population of individuals and let them evolve over time.
- Survival of the fittest

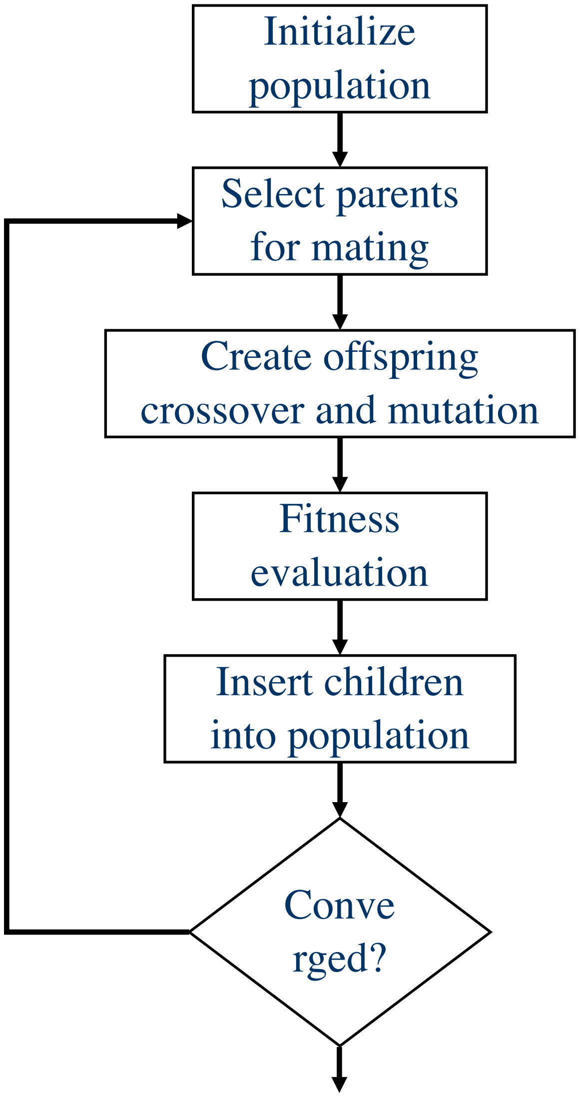
What is a GA

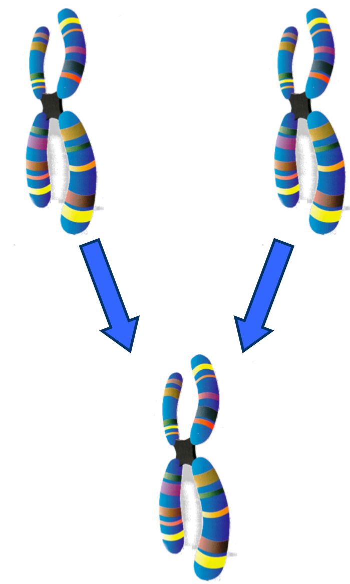
The *genetic algorithm* is a probabilistic search algorithm that iteratively transforms a set (called a *population*) of individuals (i.e. mathematical objects, typically fixed-length binary character strings), each with an associated fitness value, into a new population of offspring objects using the Darwinian principle of natural selection and using operations that are patterned after naturally occurring genetic operations, such as *crossover* (sexual recombination) and *mutation*.

From problem to solution



GA – working principle





Hamburger restaurant problem

- **The price for a meal**

  1 = $ 2.00 / burger

  0 = $20.00 /burger

- **The drinks**

  1 = Coca Cola

  0 = Wine

- **Ambiance**

  1 = Fast sloppy service

  0 = luxurious service with tuxedoed waiter

The search space

1	000
2	001
3	010
4	011
5	100
6	101
7	110
8	111

- Alphabet size K=2, Length L=3
- Size of search space: $K^L=2^L=2^3=8$

Chromosome (genome) for two different ‘designs’

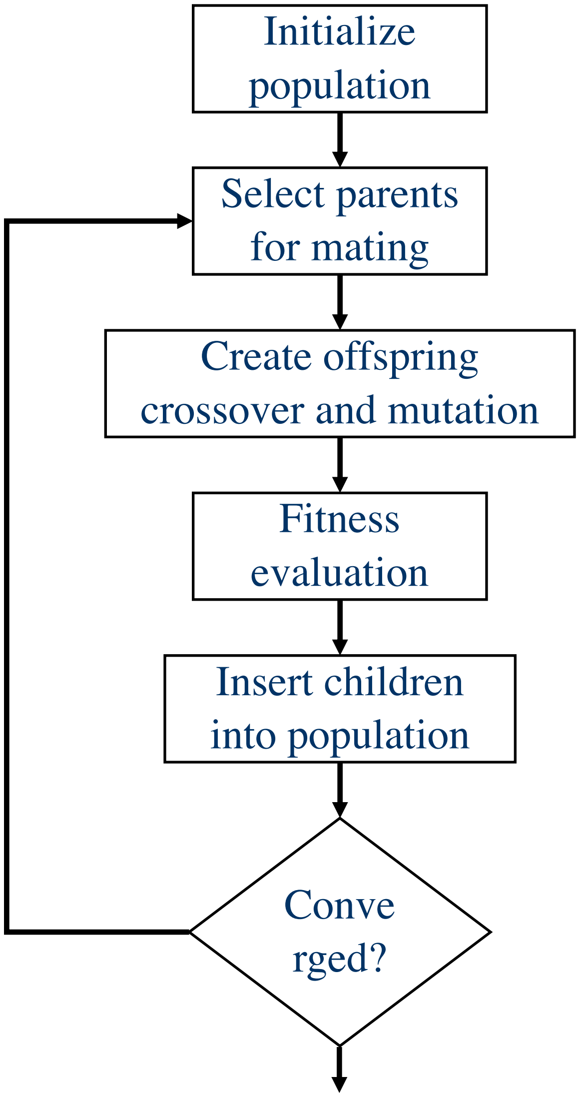
McDONALD's

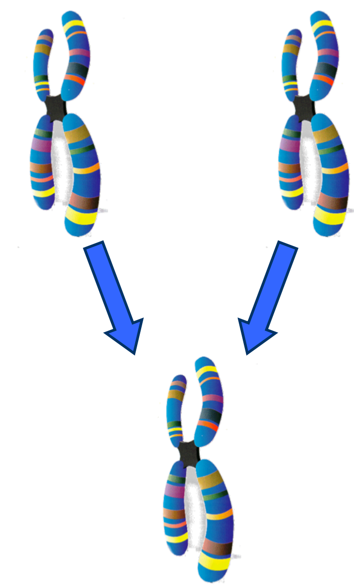
1	**1**	**1**

Frensh resturant

0	**0**	**0**

GA – working principle



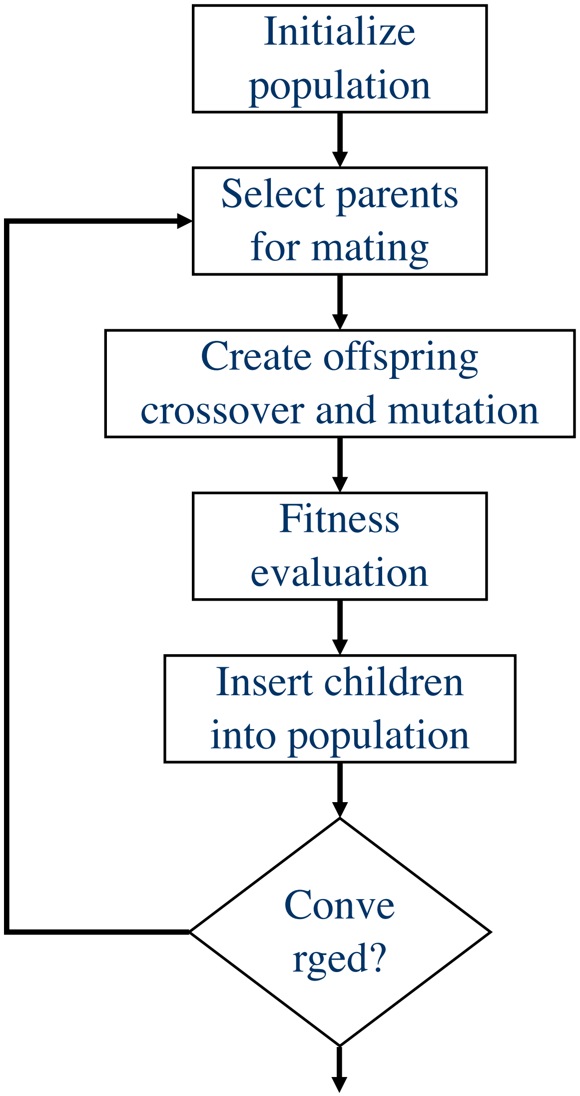


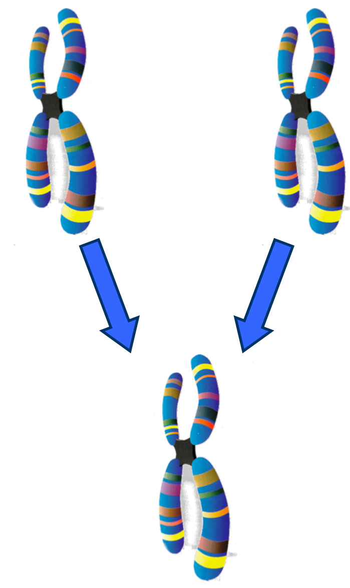
Generation 0

Chromosome **Score**

	Generation 0		
1	011	3	
2	001	1	
3	110	6	
4	010	2	
Total			
Worst			
Average			
Best			

GA – working principle





Generation 0

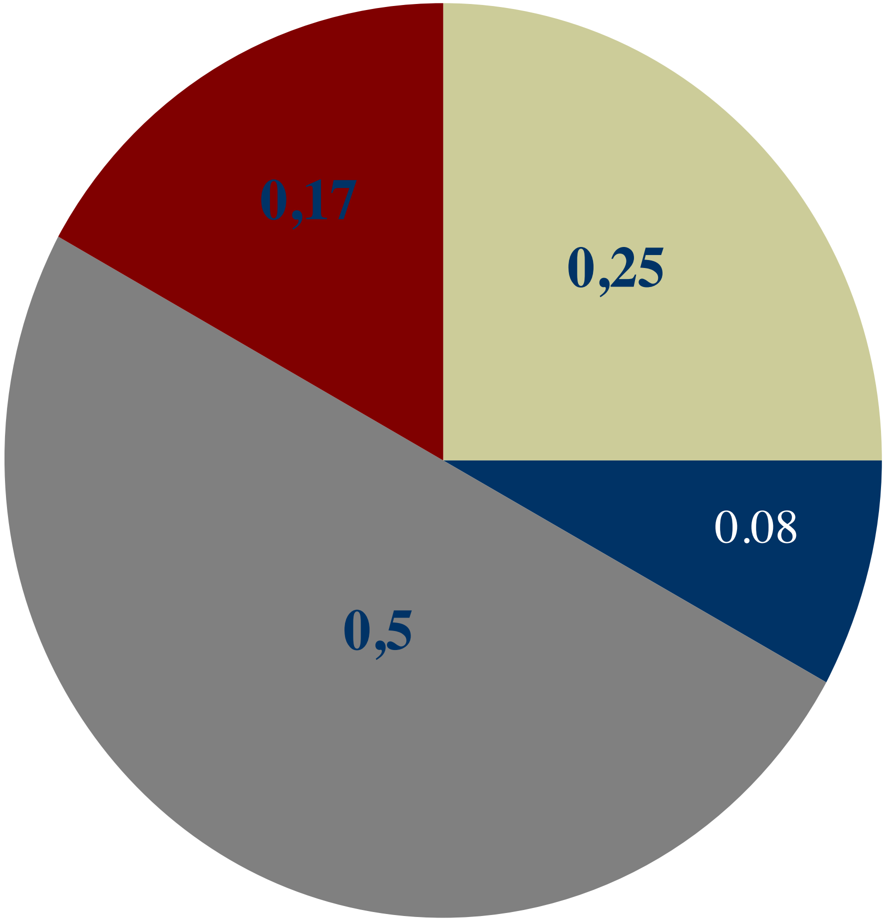
Selection probability

	Generation 0		
1	011	3	.25
2	001	1	.08
3	110	6	.50
4	010	2	.17
Total		12	
Worst		1	
Average		3.00	
Best		6	

Probabilistic selection based on fitness

- Better individuals are preferred
- Best is not always picked
- Worst is not necessarily excluded
- Nothing is guaranteed
- Mixture of greedy exploitation and adventurous exploration

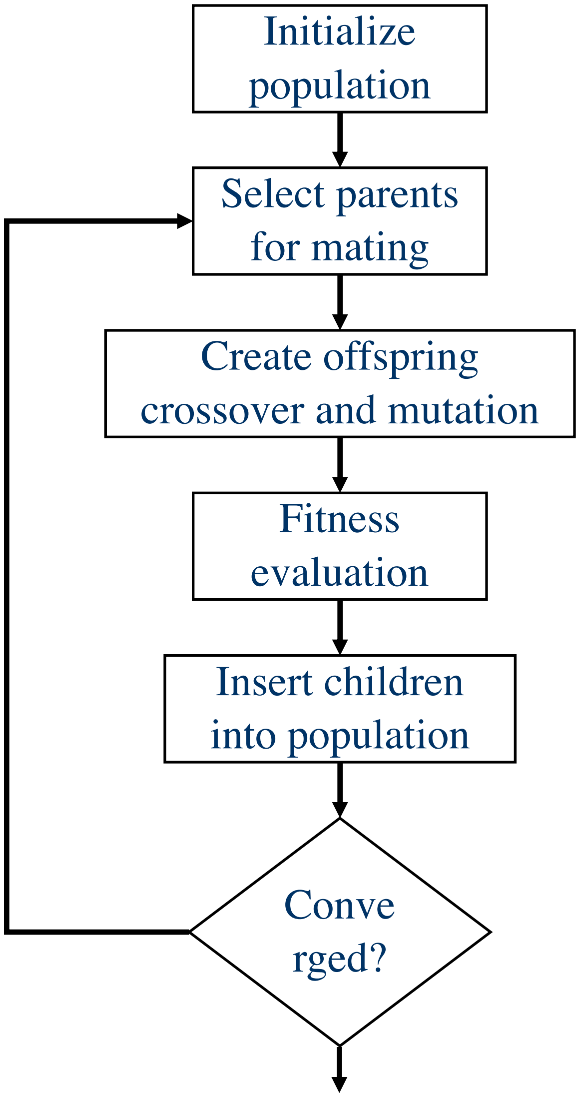
Roulette Wheel Selection

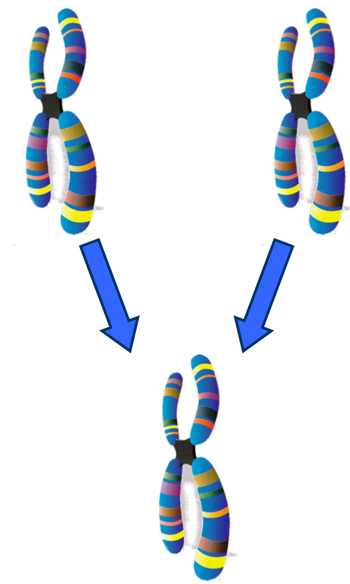


Creation of mating pool

	Generation 0			Mating pool	
1	011	3	.25	011	3
2	001	1	.08	110	6
3	110	6	.50	110	6
4	010	2	.17	010	2
Total		12			17
Worst		1			2
Average		3.00			4.5
Best		6			6

GA – working principle





Crossover operation

2 parents chosen probabilistically

Parent 1	Parent 2
011	110

One point crossover

- crossover point picked at random

Fragment 1	Fragment 2
01-	11-

- 2 remainders

Remainder 1	Remainder 2
- - 1	- - 0

- 2 offspring produced by crossover

Offspring 1	Offspring 2
010	111

After crossover operation

	Generation 0			Mating pool		Generation 1	
1	011	3	.25	011	3	111	7
2	001	1	.08	110	6	010	2
3	110	6	.50	110	6		
4	010	2	.17	010	2		
Total		12			17		
Worst		1			2		
Average		3.00			4.5		
Best		6			6		

Crossover probability might lead to no crossover

	Generation 0			Mating pool		Generation 1	
1	011	3	.25	011	3	111	7
2	001	1	.08	110	6	010	2
3	110	6	.50	110	6	110	6
4	010	2	.17	010	2	010	2
Total		12			17		
Worst		1			2		
Average		3.00			4.5		
Best		6			6		

Mutation operation

- Individual chosen randomly

Individual
010

- Mutation point chosen at random

Individual
--0

- One offspring

Offspring
011

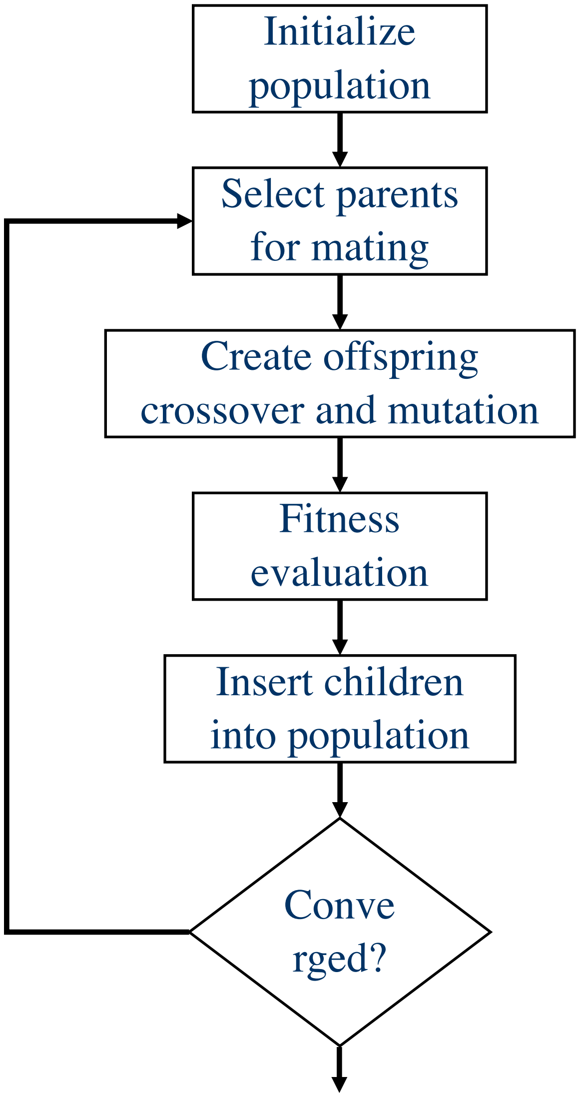
After mutation operation

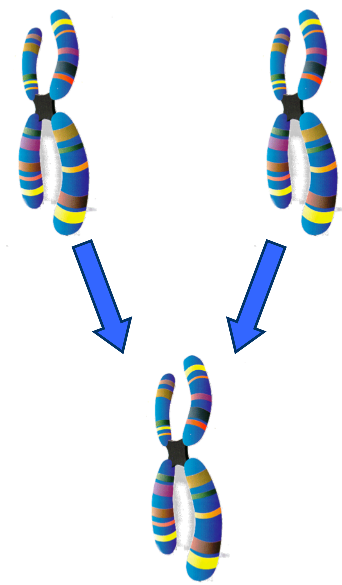
	Generation 0			Mating pool		Generation 1	
1	011	3	.25	011	3	111	7
2	001	1	.08	110	6	010	2
3	110	6	.50	110	6	110	6
4	010	2	.17	010	2	011	3
Total		12			17		
Worst		1			2		
Average		3.00			4.5		
Best		6			6		

Generation 1

	Generation 0			Mating pool		Generation 1	
1	011	3	.25	011	3	111	7
2	001	1	.08	110	6	010	2
3	110	6	.50	110	6	110	6
4	010	2	.17	010	2	011	3
Total		12			17		18
Worst		1			2		2
Average		3.00			4.5		4.5
Best		6			6		7

GA – working principle

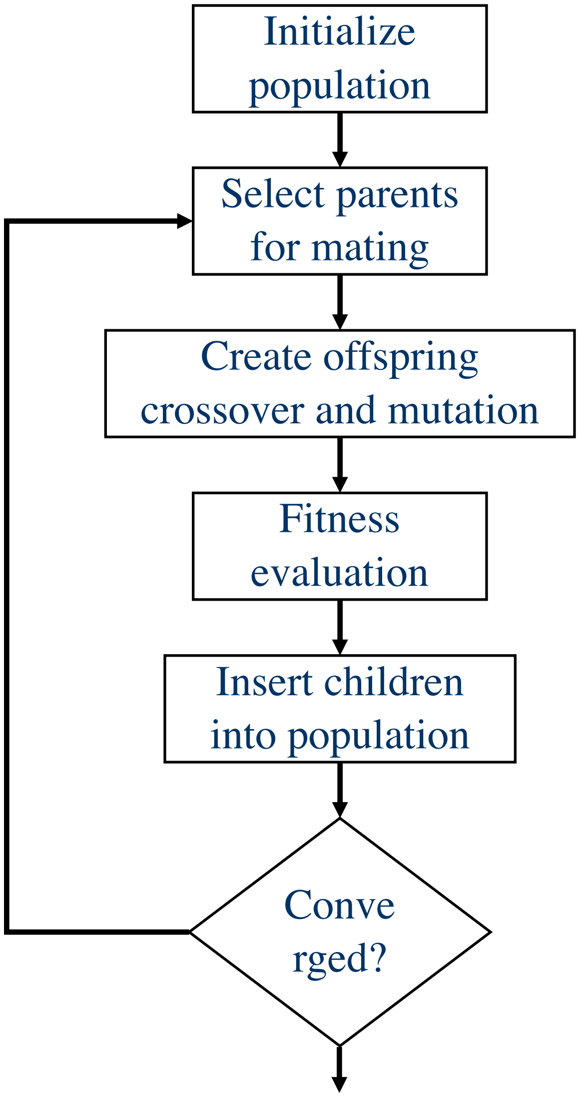


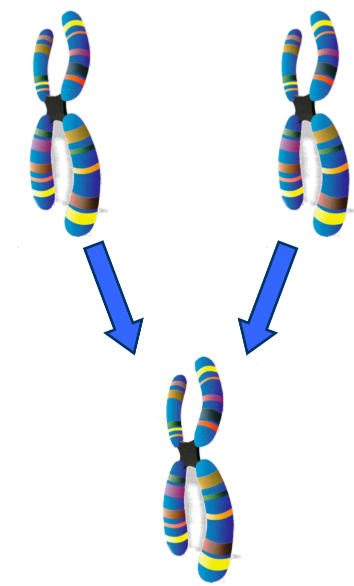


```
clear;
clc;
NIND = 4;           % Number of individuals per subpopulations
MAXGEN = 5;        % maximum Number of generations
% Reset counters
   Best = NaN*ones(MAXGEN,1);    % best in current population
   Aver = NaN*ones(MAXGEN,1);    % average of the current population
   gen = 0;         % generational counter

   % Initialise population
   NBITS = 3;
   Chrom = initbp(NIND,NBITS);    %initbp (InitBinearyPopulation(Size=nuber

   % Evaluate initial population
for i = 1:NIND,
    temp =0;
    for j=1:NBITS,
        temp = temp + 2^(NBITS-j)*Chrom(i,j);
    end
    ObjV(i,:)=-temp;
end
popini = Chrom;
% Display Data for generation 0
 fprintf('** Generation %g.  **\n\n', gen)
Chrom
ObjV

% Track best individual and display convergence
   Aver(gen+1) = mean(ObjV);
   Best(gen+1) = min(ObjV);
   plot((Best),'ro');xlabel('generation'); ylabel('f(x)');
   text(0.5,0.95,['Best = ', num2str(Best(gen+1))],'Units','normalized');
   drawnow;
```

```matlab
    while gen < MAXGEN,
     fprintf('** Generation %g.  **\n\n', gen+1)
     % Calculate fitness values
        FitV=ranking(ObjV);
     % Select individuals for breeding
        SelCh = selection('selrws', Chrom, FitV);
     % Recombine selected individuals (crossover)
        SelCh = recombin('recsp',SelCh,0.6);
     % Perform mutation on offspring

     SelCh = mutate('mutbin',SelCh,0.);                % Binary mutation
     % Evaluate offspring, call objective function
       for i = 1:length(SelCh),
             temp =0;
             for j=1:NBITS,
                 temp = temp + 2^(NBITS-j)*SelCh(i,j);
             end
             ObjVSel(i,:)=-temp;
       end
     % Reinsert offspring into current population
        [Chrom ObjV]=reins(Chrom,SelCh,NaN,NaN,NaN,ObjV,ObjVSel)
     % Increment generational counter
        gen = gen+1;
     % Update display and record current best individual
        Aver(gen+1) = mean(ObjV);
        Best(gen+1) = min(ObjV);
        plot((Best),'ro'); xlabel('generation'); ylabel('f(x)');
        text(0.5,0.95,['Best = ', num2str(Best(gen+1))],'Units','normalized');
        drawnow;
    end
% End of GA
```

GA – working principle



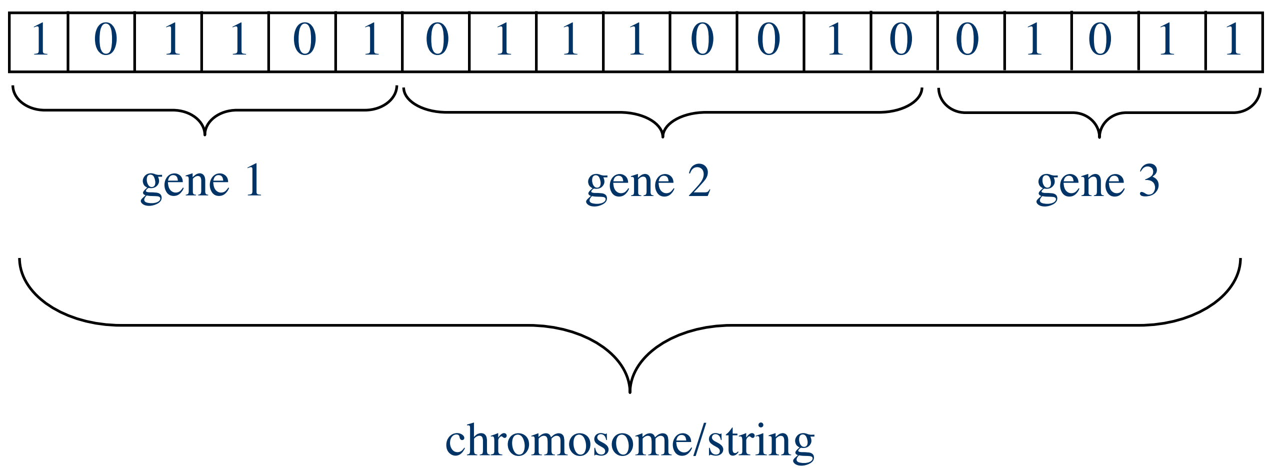


Representation

- How the optimization variables are represented in the GA.
- Each parameter is coded into a gene
- The genes for each parameter forms a chromosome or genome.
- The original GA used a binary encoding.
- We will also look at real coded GA:s

Binary representation

- Three optimization variables:
  - gene 1 coded by 6-bits
  - gene 2 coded by 8-bits
  - gene 2 coded by 5-bits



The chromosome represents one individual solution

Selection

How to select the parents that shall be used to create offspring

- Roulette wheel

  A roulette wheel where the size of the slots are proportional to the fitness value.

- Tournament selection

  Parents compete against each other

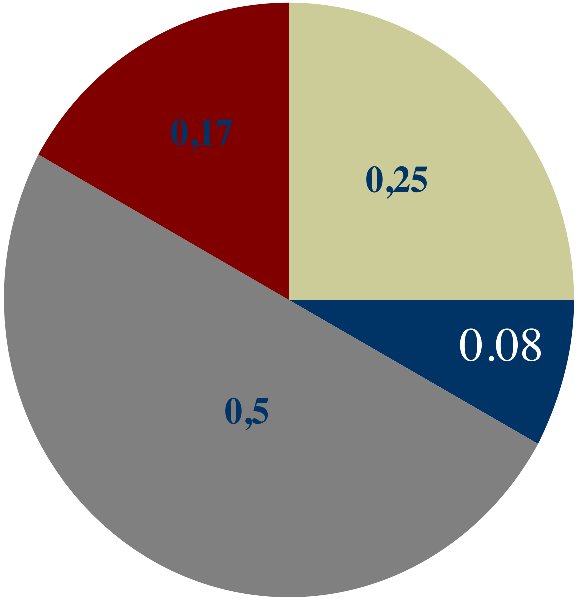
- Uniform selection

  Select parents randomly

Roulette wheel selection

Example population

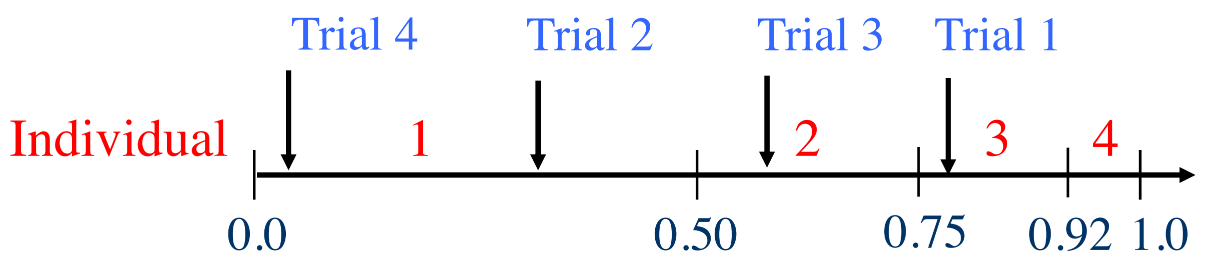
Number of individual	1	2	3	4
fitness value	6	3	2	1
selection probability	0.50	0.25	0.17	0.08



Roulette wheel selection: implementation

<u>Example population</u>

Number of individual	1	2	3	4
fitness value	6	3	2	1
selection probability	0.50	0.25	0.17	0.08



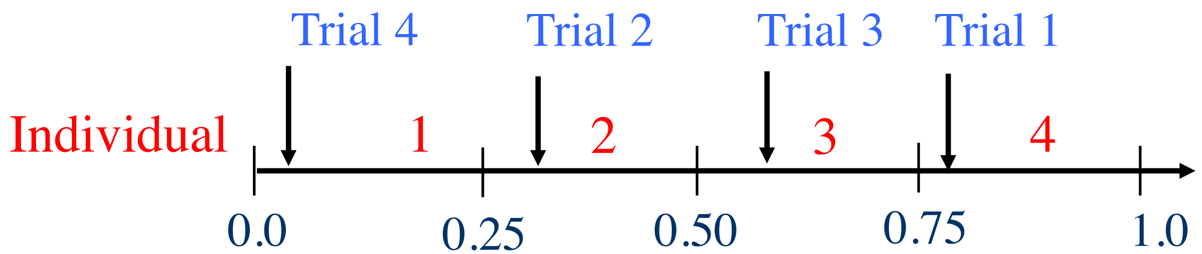
Sample 4 random numbers in order to selects parents:

0.81, 0.32, 0.65, 0.05

Uniform selection

Example population

Number of individual	1	2	3	4
fitness value	6	3	2	1
selection probability	0.25	0.25	0.25	0.25



Sample 4 random numbers in order to selects parents:
0.81, 0.32, 0.65, 0.05

Tournament selection selection

<u>Example population</u>

Number of individual	1	2	3	4
fitness value	6	3	2	1

Select 2 parents randomly from the population, e. g. no 2 and 3
Pick the best of the two as one parent, i.e. 2.

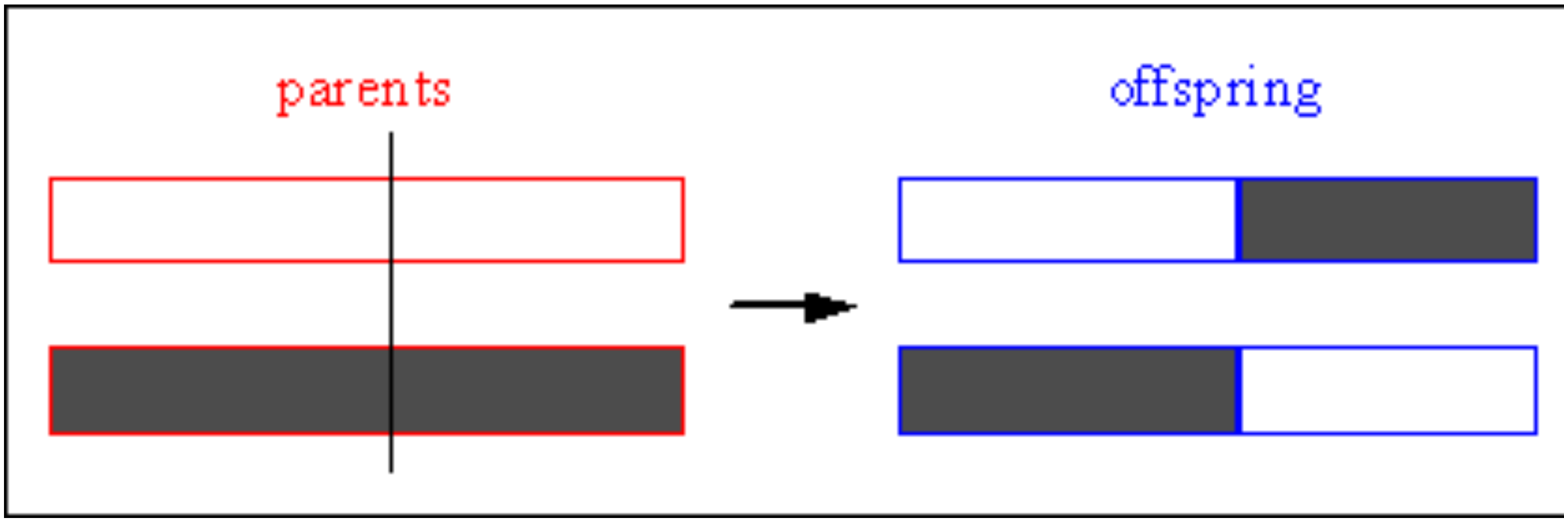
Again Pick 2 parents randomly from the population, e.g. 1 and 3
Select the best of the two as one parent, i.e. no1

Reproduction – crossover

To combine genes from different parents to create a child

- The crossover operator is the main operator for a GA
- There are a countless of ways to combines genes to create new offspring.
- Different representations implies different type crossover.
- Usually there is a probability (*pcross*) for which to cross the parents. Otherwise the children are exact copies of their parents.

One point crossover



One point binary crossover

Dad	1	0	1	1	0	1	0	0	0	1	0	0	1	0	0	1	0	1	1

Crossover site

Mom	0	0	0	0	0	1	0	1	1	1	1	1	1	0	0	1	0	0	0
Child 1	1	0	1	1	0	1	0	1	1	1	1	1	1	0	0	1	0	0	0
Child 2	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	1	1

Reproduction – mutation

- To include an extra amount of randomness
  - encourage exploration
  - avoid premature convergence, and genetic drift
- Mutation is a subordinate operator in a GA
- There are different mutation operators depending on the representation.
- Usually there is a probability (*pmut*) for which to mutate the children.

Binary flip mutation

- Change each bit in the newly created child with a probability equal to *pmut*.
- *pmut* is a small number typically 0.01.

Flip mutation

Dad	1	0	1	1	0	1	0	0	0	1	0	0	1	0	0	1	0	1	1

Crossover site

Mom	0	0	0	0	0	1	0	1	1	1	1	1	1	0	0	1	0	0	0
Child 1	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0	1	0	0	0
Child 2	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	1	0

Replacement strategies

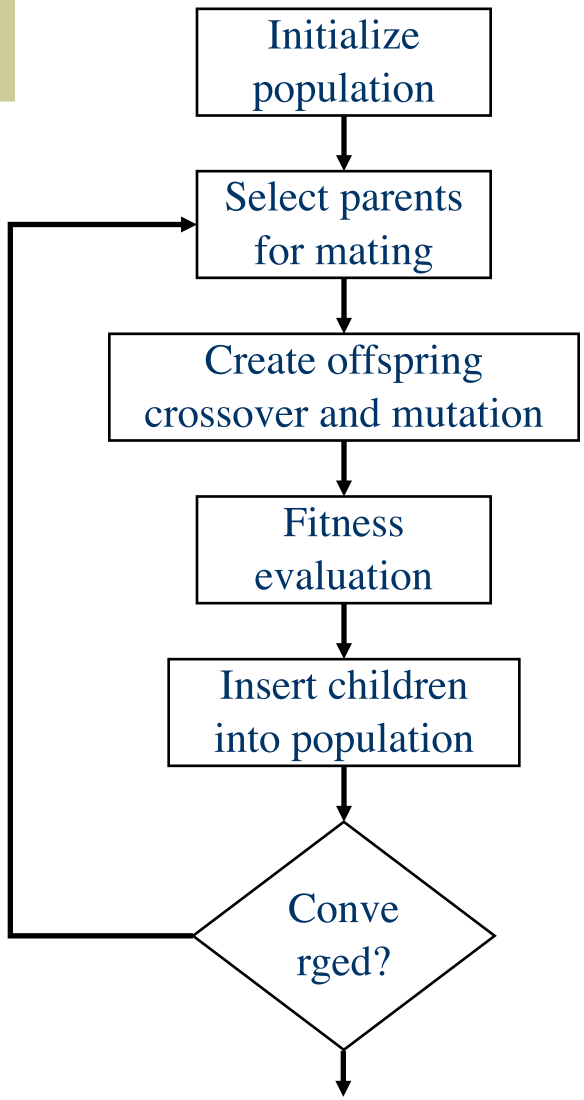
How the children are inserted into the population

- In a simple genetic algorithm all newly created children replace the old population.
- Alternative methods
  - Children replace the worst individuals.
  - Children replace their parents.
  - Children replaces individuals like them selves.

Elitism

- Make sure that the fittest individual always survive to the next generation.
- Sure – why not? Most modern GA’s uses elitism.

GA – the principle revisited



representation, *popsize*

selection method

crossover method, *pcross*

mutation method, *pmut*

fitness assignment

replacements strategy, *gengap*

termination criteria, *nogen*, *convergence*, *stagnation*

Genetic Algorithms available

- There are many publicly available genetic algorithm packages
- We will be using the genetic and evolutionary algorithm toolbox for Matlab: http://www.geatbx.com
- In my research I have used GAlib: http://lancet.mit.edu/ga/
- Other resources: Illinois Genetic Algorithms Laboratory (IlliGAL) http://www-illigal.ge.uiuc.edu/index.php3