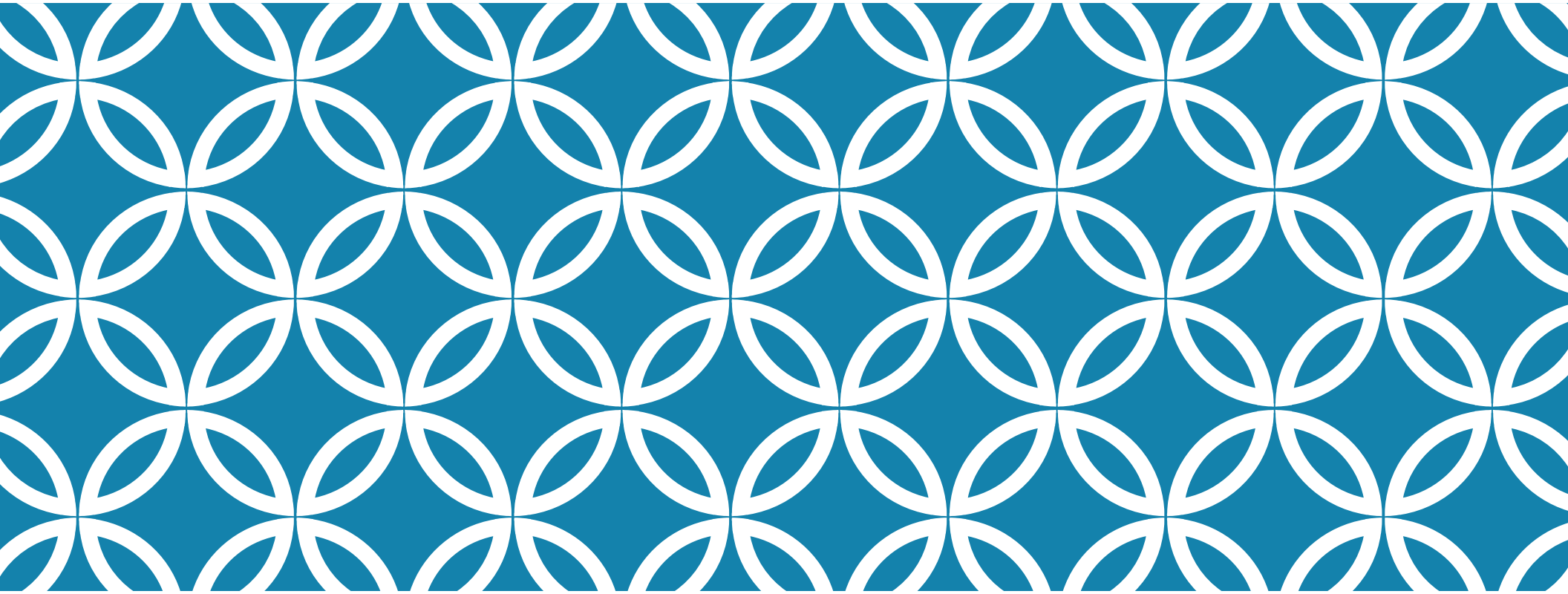


ARTIFICIAL INTELLIGENCE

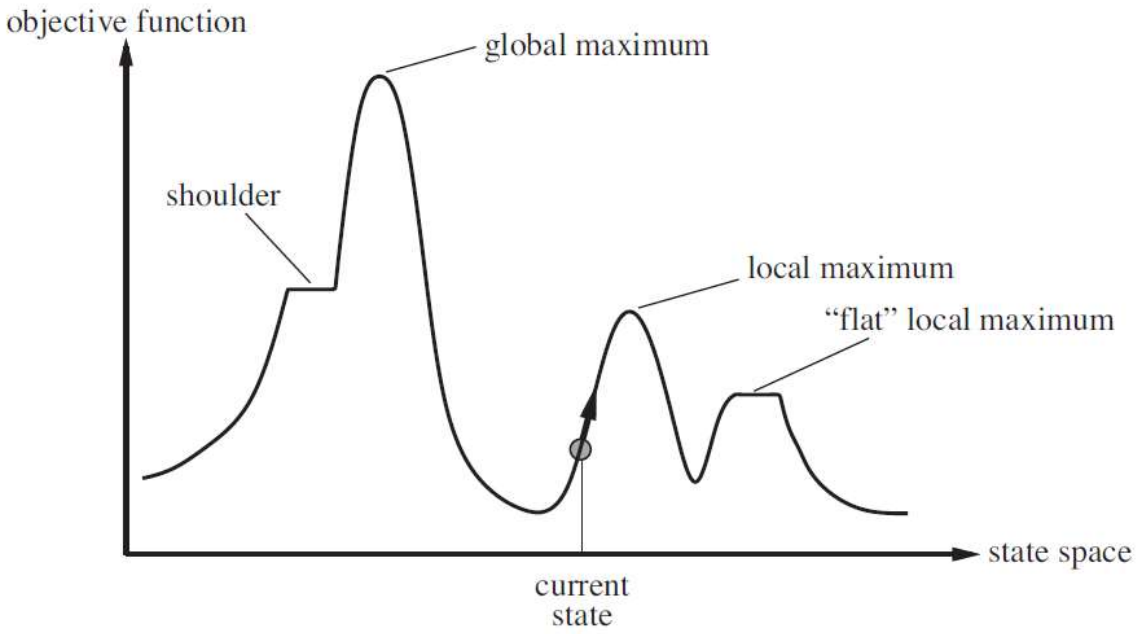
Genetic Algorithm

RECAP

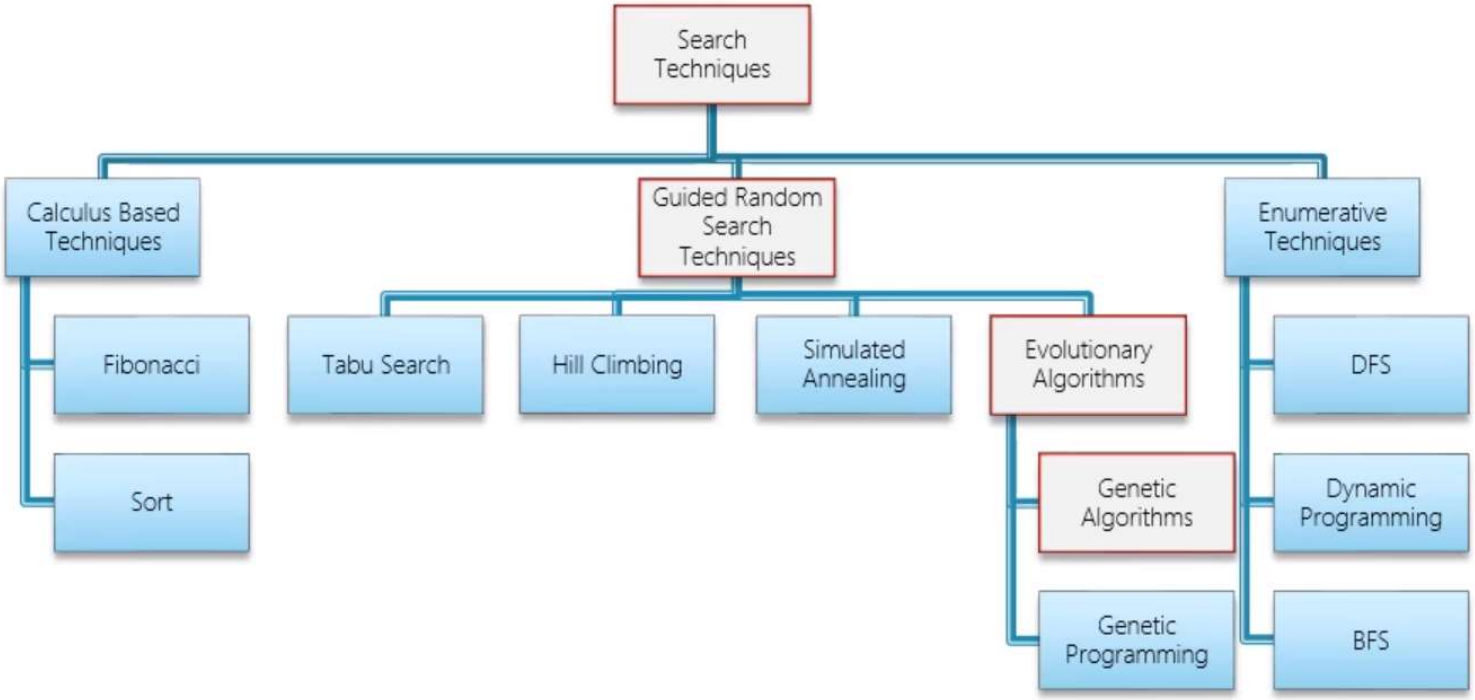
Local Search

Hill Climbing

- Random Restart Hill Climbing
- Local Beam Search
- Genetic Algorithms
- Simulated Annealing



PROBLEM SOLVING



"Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime."

- Salvatore Mangano
Computer Design, May 1995

Genetic Algorithms

FIRST – A BIOLOGY LESSON

A *gene* is a unit of heredity in a living organism

Genes are connected together into long strings called *chromosomes*

A gene represents a specific *trait* of the organism, like eye colour or hair colour, and has several different settings.

- For example, the settings for a hair colour gene may be blonde, black or brown etc.

These genes and their settings are usually referred to as an organism's *genotype*.

The physical expression of the genotype – the organism itself - is called the *phenotype*.

FIRST – A BIOLOGY LESSON

Offsprings inherit traits from parents

An offsporint may end up having half the genes from one parent and half from the other - *recombination*

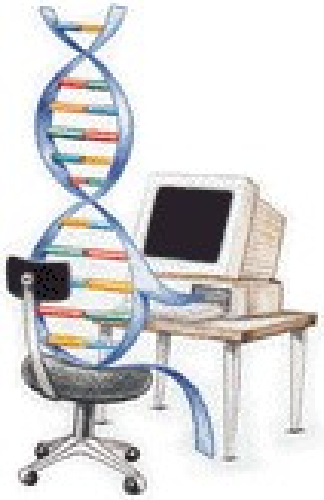
Very occasionally a gene may be *mutated* – Expressed in an organism as a completely new trait

- For example: A child may have green eyes while none of the parents had

GENETIC ALGORITHM IS

… Computer algorithm

That resides on principles of genetics and evolution



GENETIC ALGORITHMS

Search algorithms based on the mechanics of biological evolution

Developed by John Holland, University of Michigan (1970's)

Provide efficient, effective techniques for optimization and machine learning applications

Widely-used today in business, scientific and engineering circles

GENETIC ALGORITHM

Genetic algorithm (GA) introduces the principle of evolution and genetics into search among possible solutions to given problem

The idea is to simulate the process in natural systems

This is done by the creation within a machine a population of individuals

GENETIC ALGORITHM

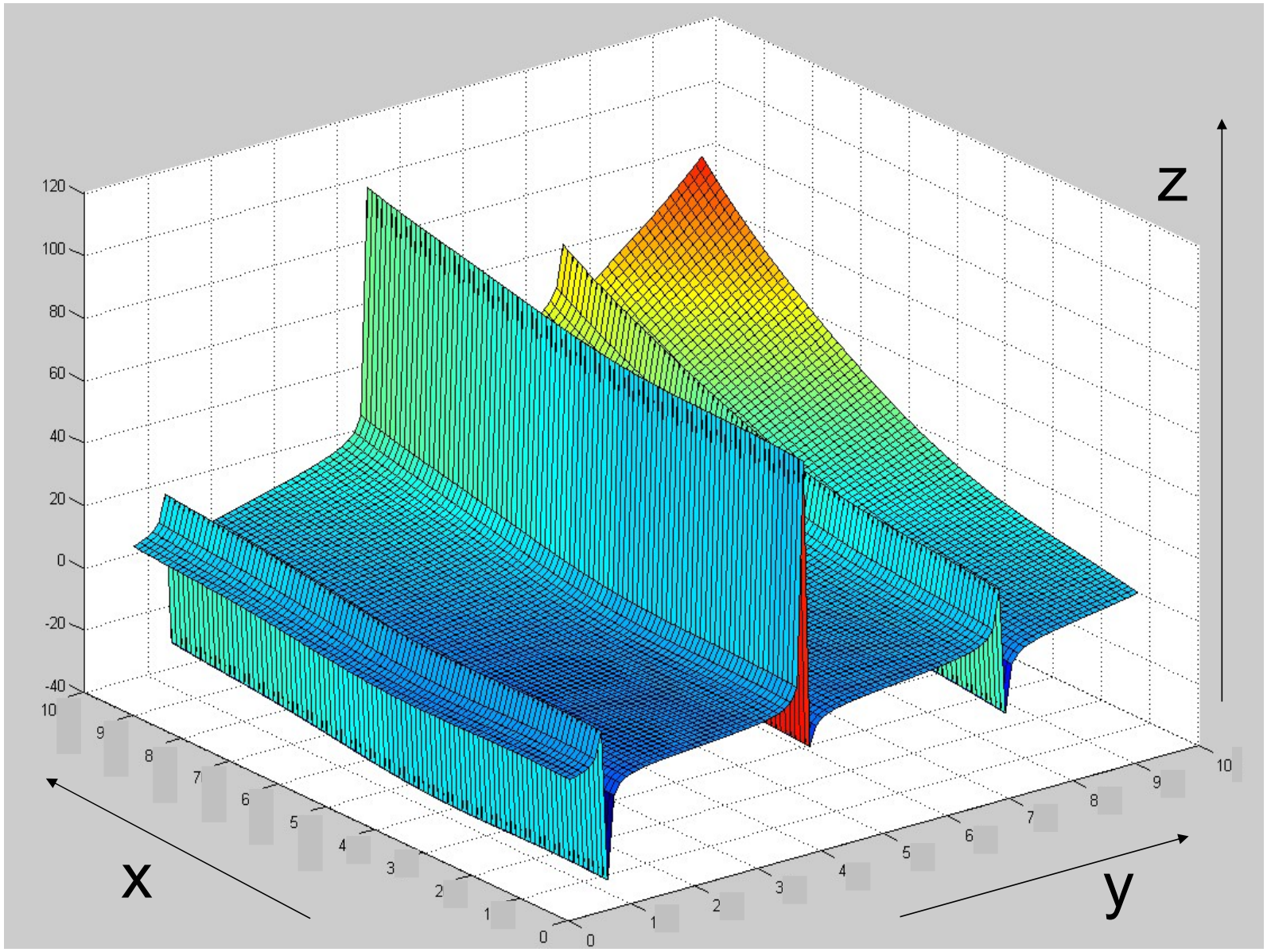
- Survival of the fittest
  - The main principle of evolution used in GA is "*survival of the fittest*".
  - The good solution survive, while bad ones die.

APPLICATIONS: OPTIMIZATION

Assume an individual is going to give you *z* dollars, after you tell them the value of *x* and *y*

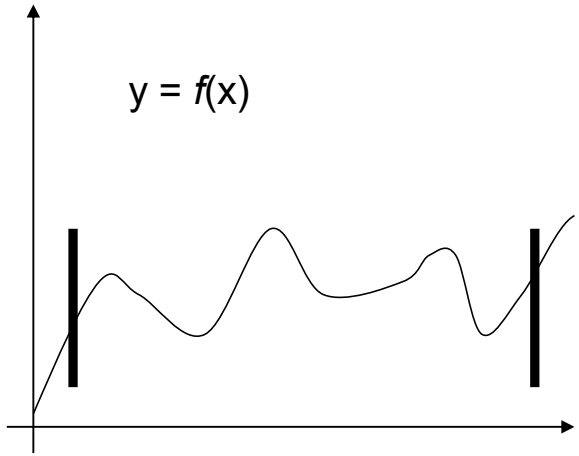
$$z = \sin(x) + \tan(y) + 1.25^{(x+y)}$$

x and *y* in the range of 0 to 10

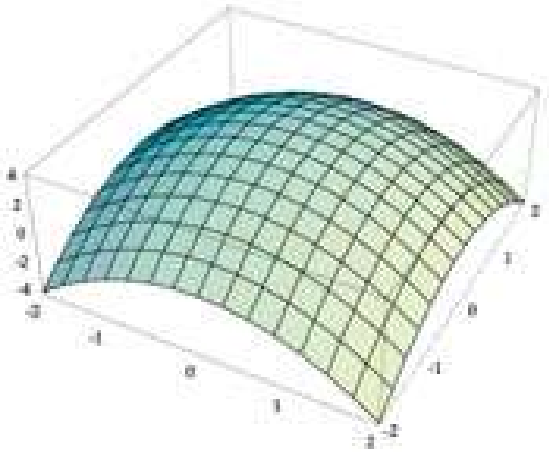

z
120
100
80
60
40
20
0
-20
-40
x
10
9
8
7
6
5
4
3
2
1
0
y
0
1
2
3
4
5
6
7
8
9
10

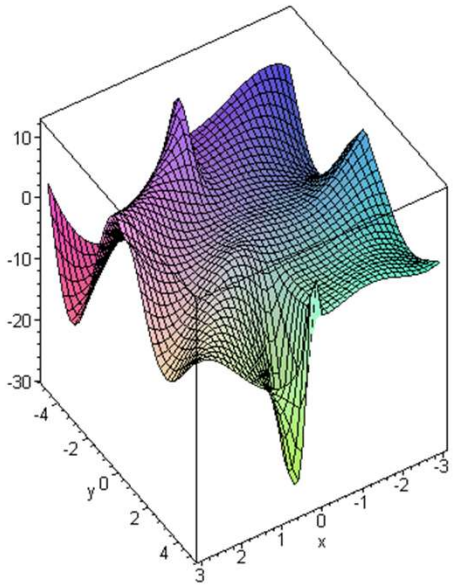
EXAMPLE PROBLEM I

(CONTINUOUS)



Finding the maximum (minimum) of some function (within a defined range).





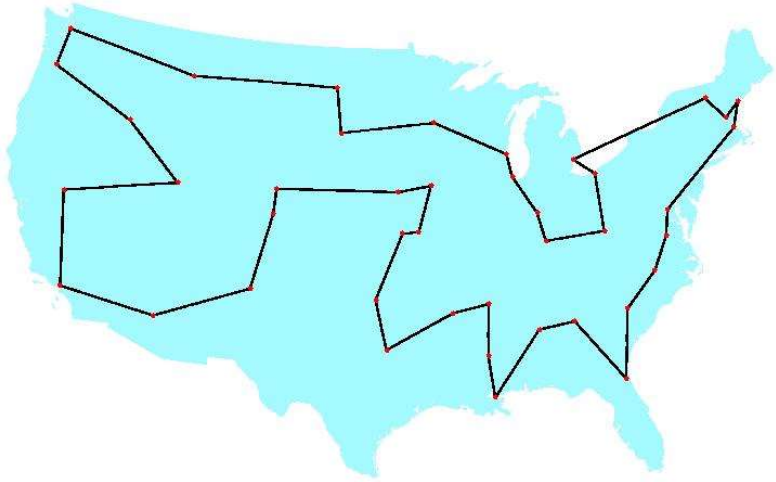
EXAMPLE PROBLEM II

(DISCRETE)

The Traveling Salesman Problem (TSP)

A salesman spends his time visiting *n* cities. In one tour he visits each city just once, and finishes up where he started. In what order should he visit them to minimize the distance traveled?

There are (n-1)!/2 possible tours.

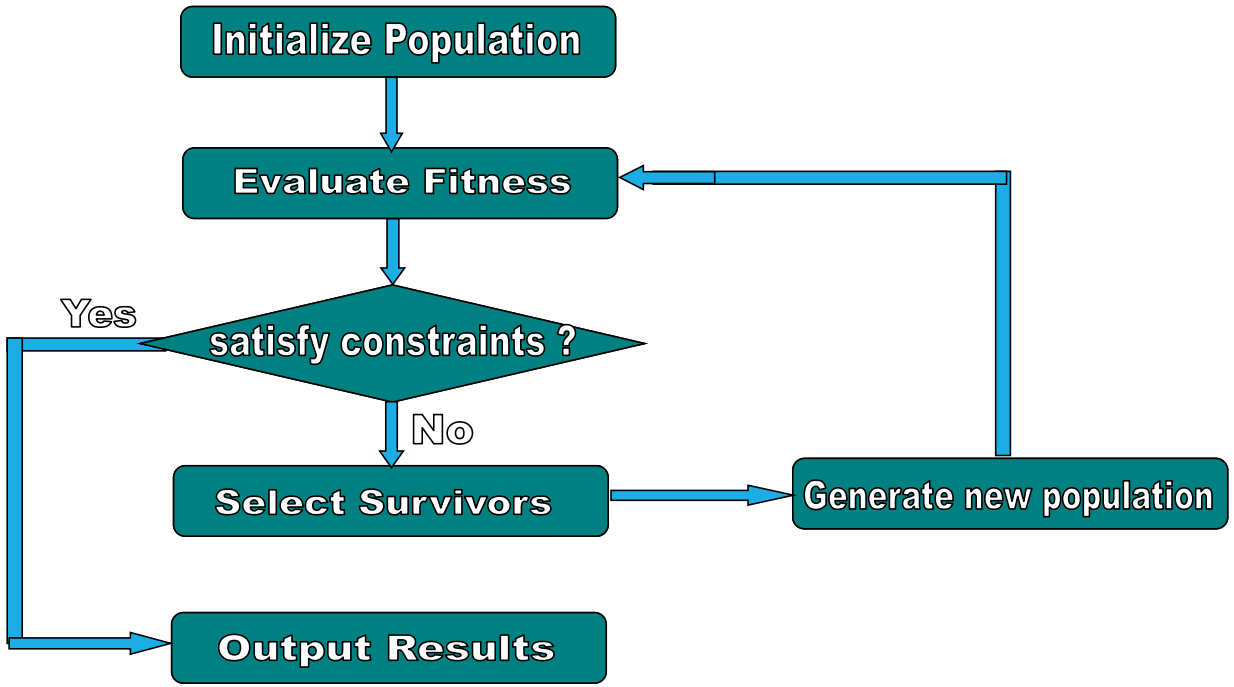


	A	B	C	
A	0	12	34	...
B	12	0	76	...
C	34	76	0	...
...	...	...	...	...

GENETIC ALGORITHM

- Inspired by natural evolution
- Population of individuals
  - *Individual is feasible solution to problem*
- Each individual is characterized by a Fitness function
  - *Higher fitness is better solution*
- Based on their fitness, parents are selected to produce offspring for a new generation
  - *Fitter individuals have more chance to reproduce*
  - *New generation has same size as old generation; old generation dies*
- Offspring has combination of properties of two parents
- If well designed, population will converge to optimal solution

GENETIC ALGORITHM



GENETIC ALGORITHM

Coding or Representation

- Possible solutions to problem

Fitness function

- Parent selection

Reproduction

- Crossover
- Mutation

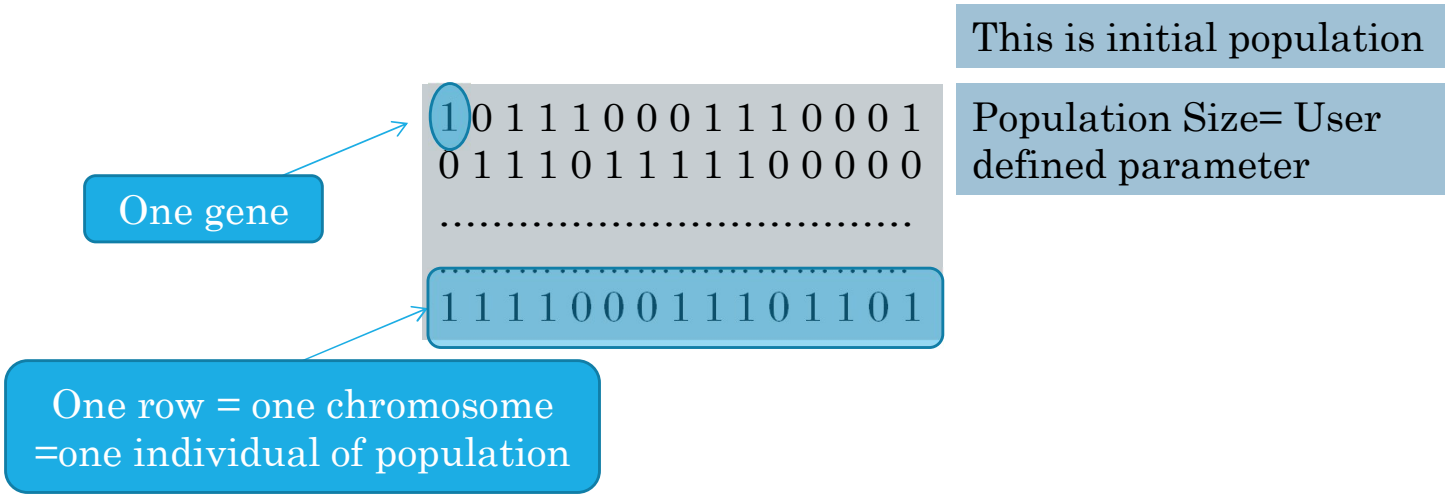
Convergence

- When to stop

CODING – EXAMPLE: FEATURE SELECTION

Assume we have 15 features f1 to f15

Generate binary strings of 15 bits as initial population



1 means the feature is used – 0 means the feature is not used

CODING: EXAMPLE

The Traveling Salesman Problem:

Find a tour of a given set of cities so that

- Each city is visited only once
- The total distance traveled is minimized

Representation is an ordered list of city numbers

1) London 3) Dunedin 5) Beijing 7) Tokyo

2) Venice 4) Singapore 6) Phoenix 8) Victoria

CityList1 (3 5 7 2 1 6 4 8)

CityList2 (2 5 7 6 8 1 3 4)

FITNESS FUNCTION/PARENT SELECTION

- Fitness function evaluates *how good* an individual is in solving the problem
- Fitness is computed for each individual
- Fitness function is application depended
- For classification – we may use the classification rate as the fitness function
- Find the fitness value of each individual in the population

FITNESS FUNCTION/PARENT SELECTION

Parent/Survivor Selection

- RouletteWheel Selection
- Tournament Selection
- Rank Selection
- Elitist Selection

ROULETTE WHEEL SELECTION

Main idea: better individuals get higher chance

Individuals are assigned a probability of being selected based on their fitness.

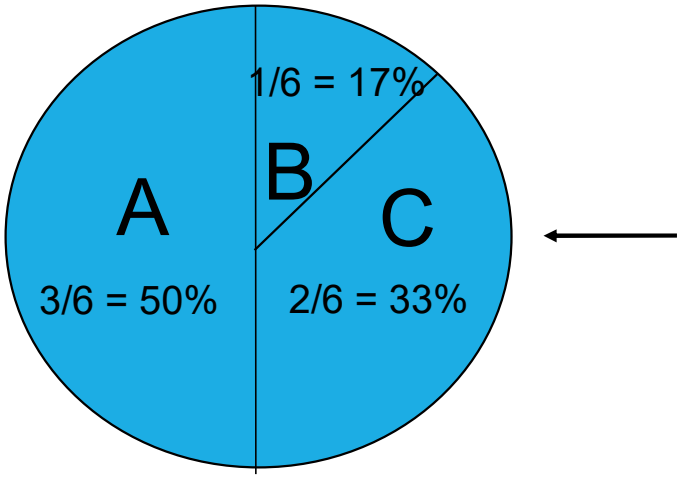
$$p_i = f_i / \Sigma f_j$$

- Where p_i is the probability that individual i will be selected,
- f_i is the fitness of individual i, and
- Σf_j represents the sum of all the fitnesses of the individuals with the population.

ROULETTE WHEEL SELECTION

Assign to each individual a part of the roulette wheel

Spin the wheel n times to select n individuals



fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

TOURNAMENT SELECTION

Binary tournament

- Two individuals are randomly chosen; the fitter of the two is selected as a parent

Larger tournaments

- *n* individuals are randomly chosen; the fittest one is selected as a parent

OTHER METHODS

Rank Selection

- Each individual in the population is assigned a numerical rank based on fitness, and selection is based on this ranking.

Elitism

- Reserve *k* slots in the next generation for the highest scoring/fittest chormosomes of the current generation

REPRODUCTION

Reproduction operators

- Crossover
- Mutation

Crossover is usually the primary operator with mutation serving only as a mechanism to introduce diversity in the population

REPRODUCTION

Crossover

- Two parents produce two offspring
- There is a chance that the chromosomes of the two parents are copied unmodified as offspring
- There is a chance that the chromosomes of the two parents are randomly recombined (crossover) to form offspring
- Generally the chance of crossover is between 0.6 and 1.0

Mutation

- There is a chance that a gene of a child is changed randomly
- Generally the chance of mutation is low (e.g. 0.001)

CROSSOVER

Generating offspring from two selected parents

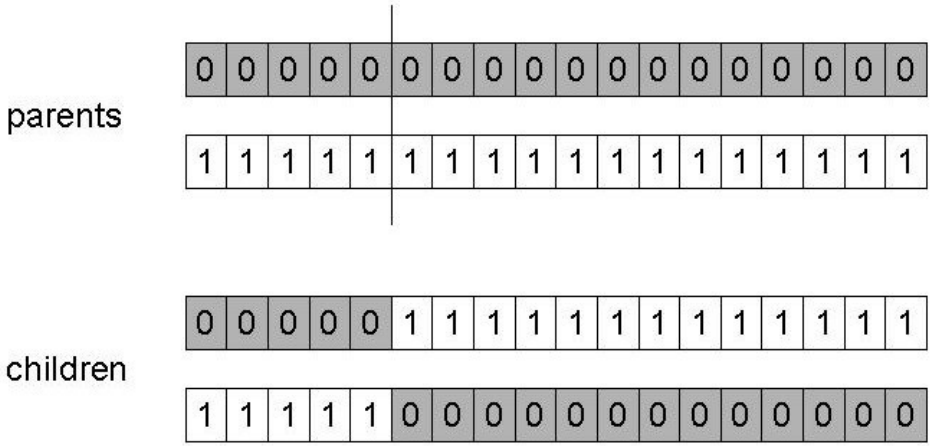
- Single point crossover
- Two point crossover (Multi point crossover)
- Uniform crossover

ONE PONT CROSSOVER

Choose a random point on the two parents

Split parents at this crossover point

Create children by exchanging tails



ONE PONT CROSSOVER

Choose a random point on the two parents

Split parents at this crossover point

Create children by exchanging tails

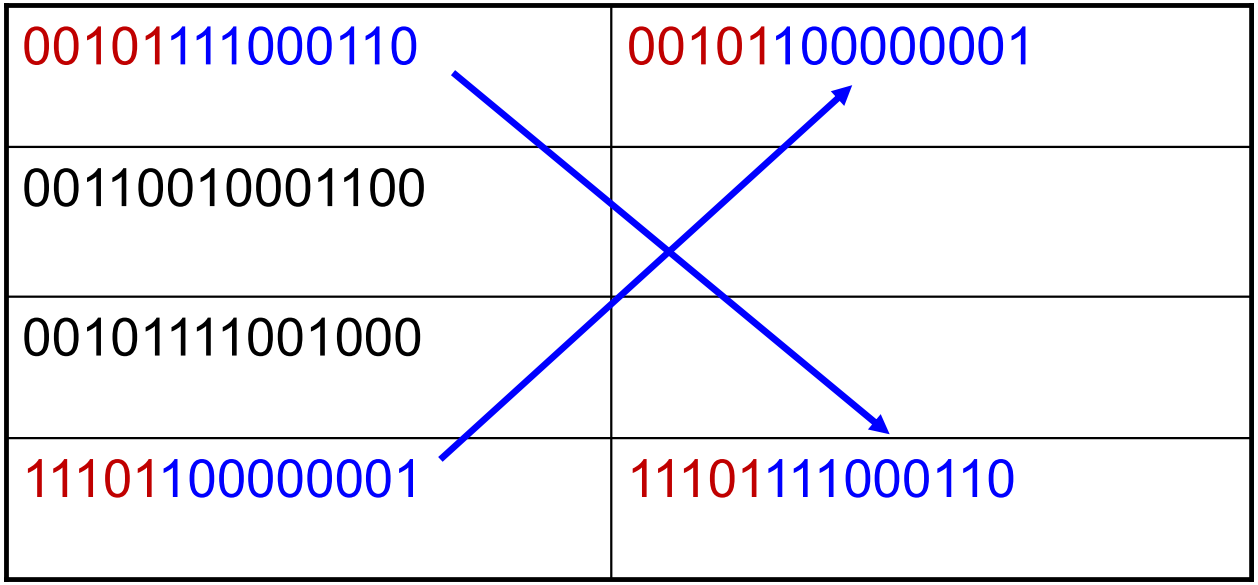
Parent 1:	**X X** | X X X X X
Parent 2:	Y Y | **Y Y Y Y Y**
Offspring 1:	**X X Y Y Y Y Y**
Offspring 2:	Y Y X X X X X

CROSSOVER

00101111000110	←
00110010001100	
00101111001000	
11101100000001	←

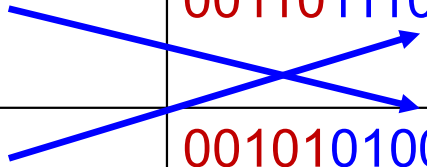
Crossover point

CROSSOVER


00101111000110
00101100000001
00110010001100
00101111001000
11101100000001
11101111000110

CROSSOVER

00101111000110	00101100000001
00110010001100	00110111001000
00101111001000	00101010001100
11101100000001	11101111000110



TWO POINT CORSSOVER

Two-Point crossover is very similar to single-point crossover except that two cut-points are generated instead of one.

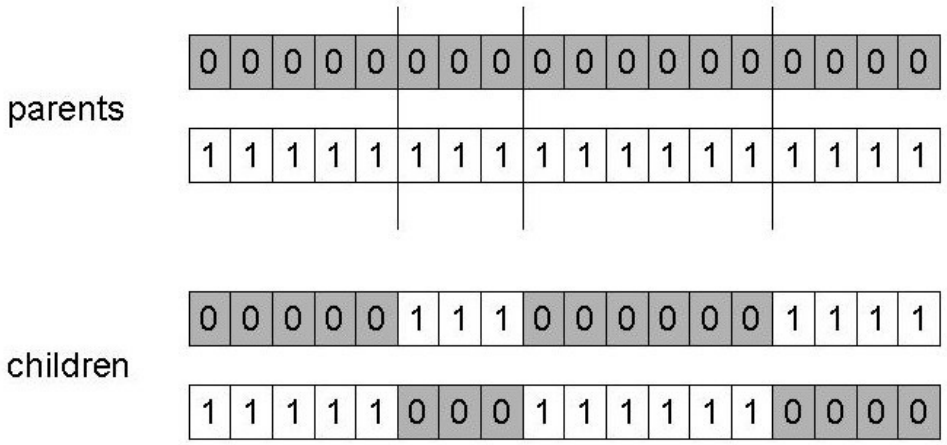
Parent 1: **X X** | X X X | **X X**
Parent 2: Y Y | **Y Y Y** | Y Y
Offspring 1: **X X Y Y Y X X**
Offspring 2: Y Y X X X Y Y

N POINT CROSSOVER

Choose n random crossover points

Split along those points

Glue parts, alternating between parents



UNIFORM CORSSOVER

- A random mask is generated
- The mask determines which bits are copied from one parent and which from the other parent
- Bit density in mask determines how much material is taken from the other parent

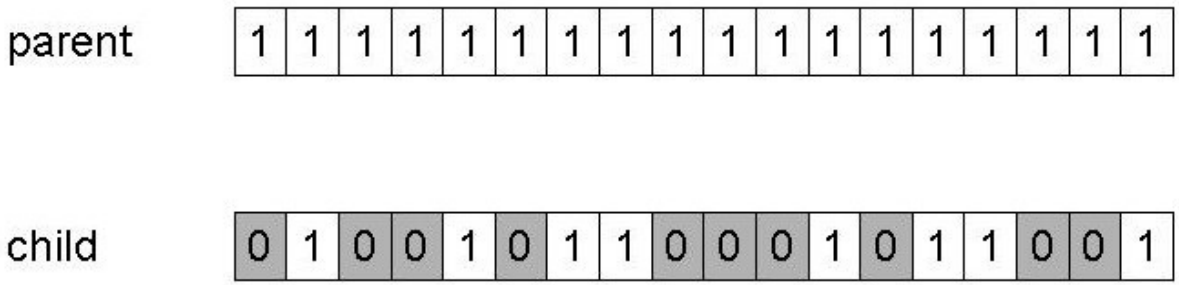
```
Mask:        0110011000      (Randomly generated)
Parents:     1010001110  0011010010

Offspring:   0011001010  1010010110
```

MUTATION

Alter each gene independently with a probability p_m

p_m is called the mutation rate

- Typically between 1/pop_size and 1/ chromosome_length



SUMMARY – REPRODUCTION CYCLE

Select parents for producing the next generation

For each consecutive pair apply crossover with probability p_c , otherwise copy parents

For each offspring apply mutation (bit-flip with probability p_m)

Replace the population with the resulting population of offsprings

CONVERGENCE

Stop Criterion

- Number of generations
- Fitness value
  - How fit is the fittest individual

GA FOR FEATURE SELECTION

The initial population is randomly generated

Each chromosome is evaluated using the fitness function

The fitness values of the current population are used to find the off springs of the next generation

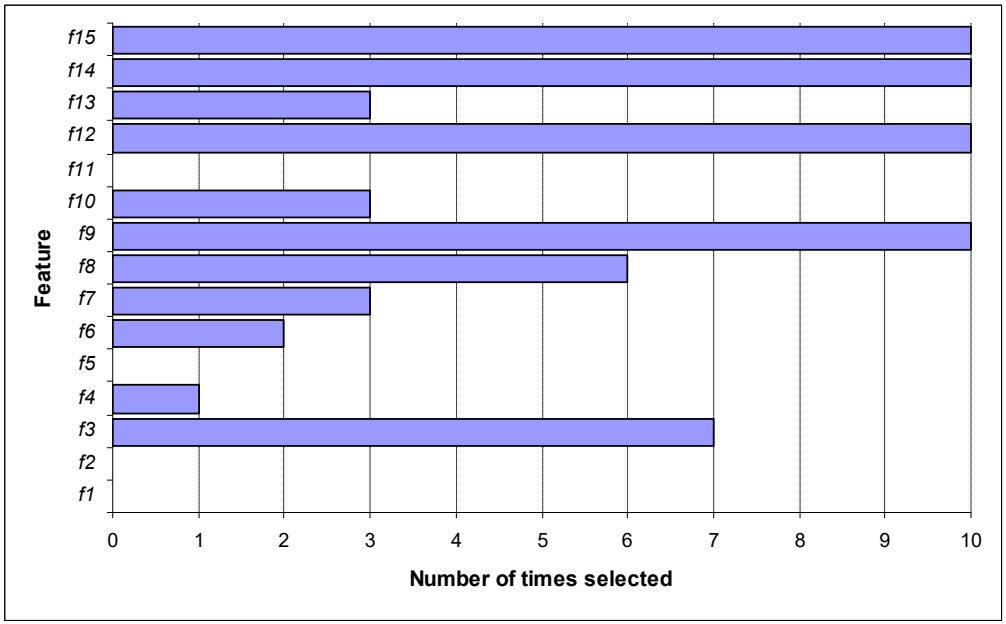
The generational process ends when the termination criterion is satisfied

The selected features correspond to the best individual in the last generation

GA FOR FEATURE SELECTION

GA can be executed multiple times

Example: 15 features, GA executed 10 times



GA FOR FEATURE SELECTION

Feature categories based on frequency of selection

Indispensable:

- Feature selected in each selected feature subset.

Irrelevant:

- Feature not selected in any of the selected subsets.

Partially Relevant:

- Feature selected in some of the subsets.

N QUEEN PROBLEM
STATE REPRESENTATION

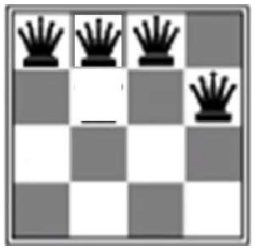
State / chromosome / individual / survivor / string

Representation also known as encoding i.e,. State representation = chromosome encoding

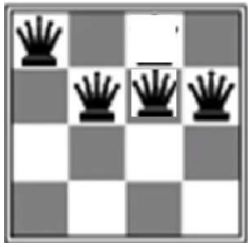
4 3 2 3 → Row position of each queen

State = "4323" → String of digits, can also be a binary string in which case its called bitstring.

h =
2+3+2+3=10
f=10/91

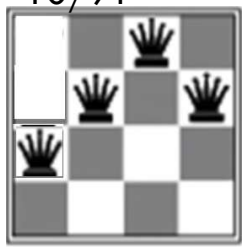


h = 2+2+3+1=8
f=8/91



h = 1+3+2+2=8
f=8/91

h = 3+2+3+2=10
f=10/91



h = 2+3+3+2=10
f=10/91

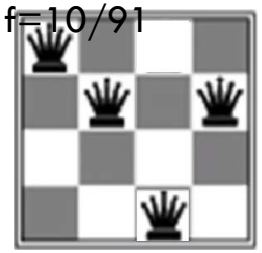
h = 1+0+2+1=4
f=4/91

h =
2+3+3+2=10
f=10/91

h = 2+2+2+0=6
f=6/91

h = 0+2+2+2=6
f=6/91

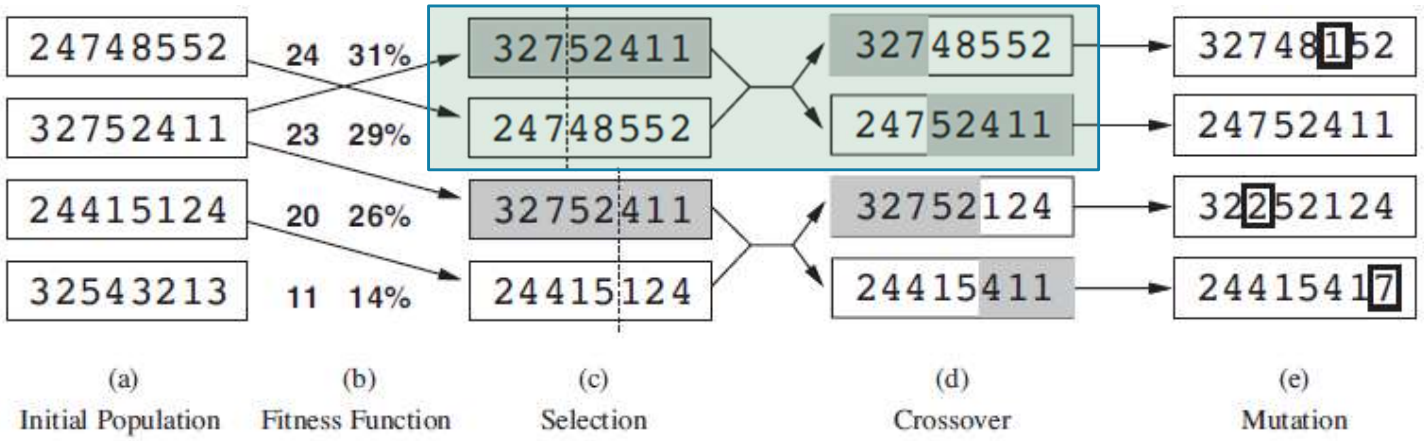
h = 1+1+2+1=5
f = 5/91

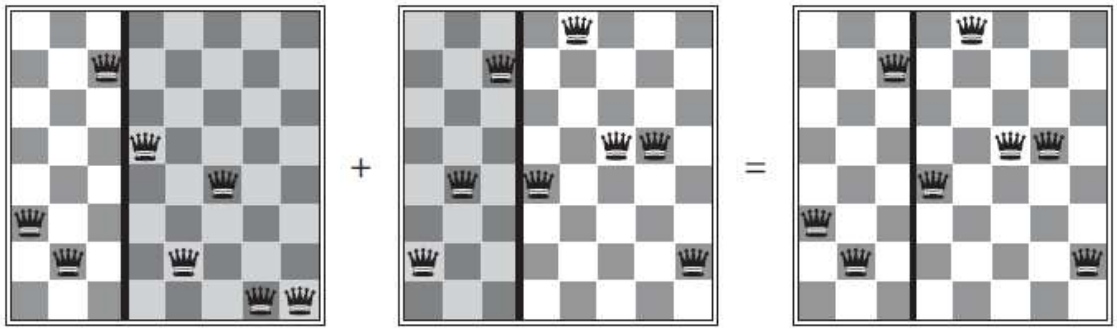


h = 1+2+0+1=4
f=4/91

h = 3+3+2+2=10
f=10/91

8 QUEENS PROBLEM: GA SOLUTION





GA WORKED EXAMPLE

- Suppose that we have a rotary system (which could be mechanical - like an internal combustion engine or gas turbine, or electrical - like an induction motor).

- The system has five parameters associated with it - *a, b, c, d and e*. *These parameters can take any* integer value between 0 and 10.

- When we adjust these parameters, the system responds by speeding up or slowing down.

- Our aim is to obtain the highest speed possible in revolutions per minute from the system.

Step 1

GA WORKED EXAMPLE

Generate a population of random strings (we'll use ten as an example):

a	*b*	*c*	*d*	*e*
1	*7*	*5*	*2*	*9*
3	8	5	8	4
9	4	8	10	1
2	6	4	8	5
3	6	8	6	9
6	8	5	9	4
7	8	6	4	5
9	10	8	7	5
1	3	5	7	8
9	8	9	6	5

Step 2

GA WORKED EXAMPLE

- Feed each of these strings into the machine, in turn, and measure the speed in revolutions per minute of the machine. This value is the fitness because the higher the speed, the better the machine:

a	*b*	*c*	*d*	*e*	**Fitness**
1	7	5	2	9	**500**
3	8	5	8	4	**2000**
9	4	8	10	1	**100**
2	6	4	8	5	**2200**
3	6	8	6	9	**1000**
6	8	5	9	4	**1100**
7	8	6	4	5	**750**
9	10	8	7	5	**330**
1	3	5	7	8	**800**
9	8	9	6	5	**1500**

Step 3

GA WORKED EXAMPLE

To select the breeding population, we'll go the easy route and sort the strings then delete the worst ones. First sorting:

a	*b*	*c*	*d*	*e*	**Fitness**
2	6	4	8	5	**2200**
3	8	5	8	4	**2000**
9	8	9	6	5	**1500**
6	8	5	9	4	**1100**
3	6	8	6	9	**1000**
1	3	5	7	8	**800**
7	8	6	4	5	**750**
1	*7*	*5*	*2*	*9*	**500**
9	10	8	7	5	**330**
9	4	8	10	1	**100**

Step 3

GA WORKED EXAMPLE

Having sorted the strings into order we delete the worst half

a	*b*	*c*	*d*	*e*
2	6	4	8	5
3	8	5	8	4
9	8	9	6	5
6	8	5	9	4
3	6	8	6	9

GA WORKED EXAMPLE

a	*b*	*c*	Step 4	
2	6	4	8	5
3	8	5	8	4
9	8	9	6	5
6	8	5	9	4
3	6	8	6	9

We can now crossover the strings by pairing them up randomly. Since there's an odd number, we'll use the best string twice. The pairs are shown below:

2	6	4	8	5	} **Pair 1**
6	8	5	9	4	
3	6	8	6	9	} **Pair 2**
3	8	5	8	4	
9	8	9	6	5	} **Pair 3**
2	6	4	8	5	

Step 4

GA WORKED EXAMPLE

The crossover points are selected randomly and are shown by the vertical lines. After crossover the strings look like this:

2	8	5	9	4
6	6	4	8	5
3	6	8	8	4
3	8	5	6	9
9	8	4	8	5
2	6	9	6	5

These can now join their parents in the next generation

Step 4

GA WORKED EXAMPLE

New Generation

a	*b*	*c*	*d*	*e*
2	6	4	8	5
3	8	5	8	4
9	8	9	6	5
6	8	5	9	4
3	6	8	6	9
2	8	5	9	4
6	6	4	8	5
3	6	8	8	4
3	8	5	6	9
9	8	4	8	5
2	6	9	6	5

Step 4

GA WORKED EXAMPLE

We have one extra string (which we picked up by using an odd number of strings in the mating pool) that we can delete after fitness testing.

a	*b*	*c*	*d*	*e*
2	6	4	8	5
3	8	5	8	4
9	8	9	6	5
6	8	5	9	4
3	6	8	6	9
2	8	5	9	4
6	6	4	8	5
3	6	8	8	4
3	8	5	6	9
9	8	4	8	5
2	6	9	6	5

Step 5

GA WORKED EXAMPLE

Finally, we have mutation, in which a small number of numbers are changed

a	*b*	*c*	*d*	*e*
2	6	4	8	5
3	8	5	8	4
9	8	9	3	5
6	8	5	9	4
3	6	8	6	9
2	8	5	9	4
6	6	4	8	5
3	6	8	8	4
3	8	5	6	9
9	8	4	8	5
2	6	9	6	5

GA WORKED EXAMPLE

After this, we repeat the algorithm from stage 2, with this new population as the starting point.

Keep repeating until convergence

GA WORKED EXAMPLE

Roulette Wheel Selection

- The alternative (roulette) method of selection would make up a breeding population by giving each of the old strings a chance of ending up in the breeding population which is proportional to its fitness

- Making the fitness for each string the addition of its own fitness with all of those before it

GA WORKED EXAMPLE

Roulette Wheel Selection

a	*b*	*c*	*d*	*e*	Fitness	Cumulative Fitness
2	6	4	8	5	**2200**	**10280**
3	8	5	8	4	**2000**	**8080**
9	8	9	6	5	**1500**	**6080**
6	8	5	9	4	**1100**	**4580**
3	6	8	6	9	**1000**	**3480**
1	3	5	7	8	**800**	**2480**
7	8	6	4	5	**750**	**1680**
1	*7*	*5*	*2*	*9*	**500**	**930**
9	10	8	7	5	**330**	**430**
9	4	8	10	1	**100**	**100**

1. If we now generate a random number between 0 and 10280 we can use this to select strings.
2. If the random number turns out to be between 0 and 100 then we choose the last string.
3. If it's between 8080 and 10280 we choose the first string.
4. If it's between 2480 and 3480 we choose the string 3 6 8 6 9, etc. You don't have to sort the strings into order to use this method.