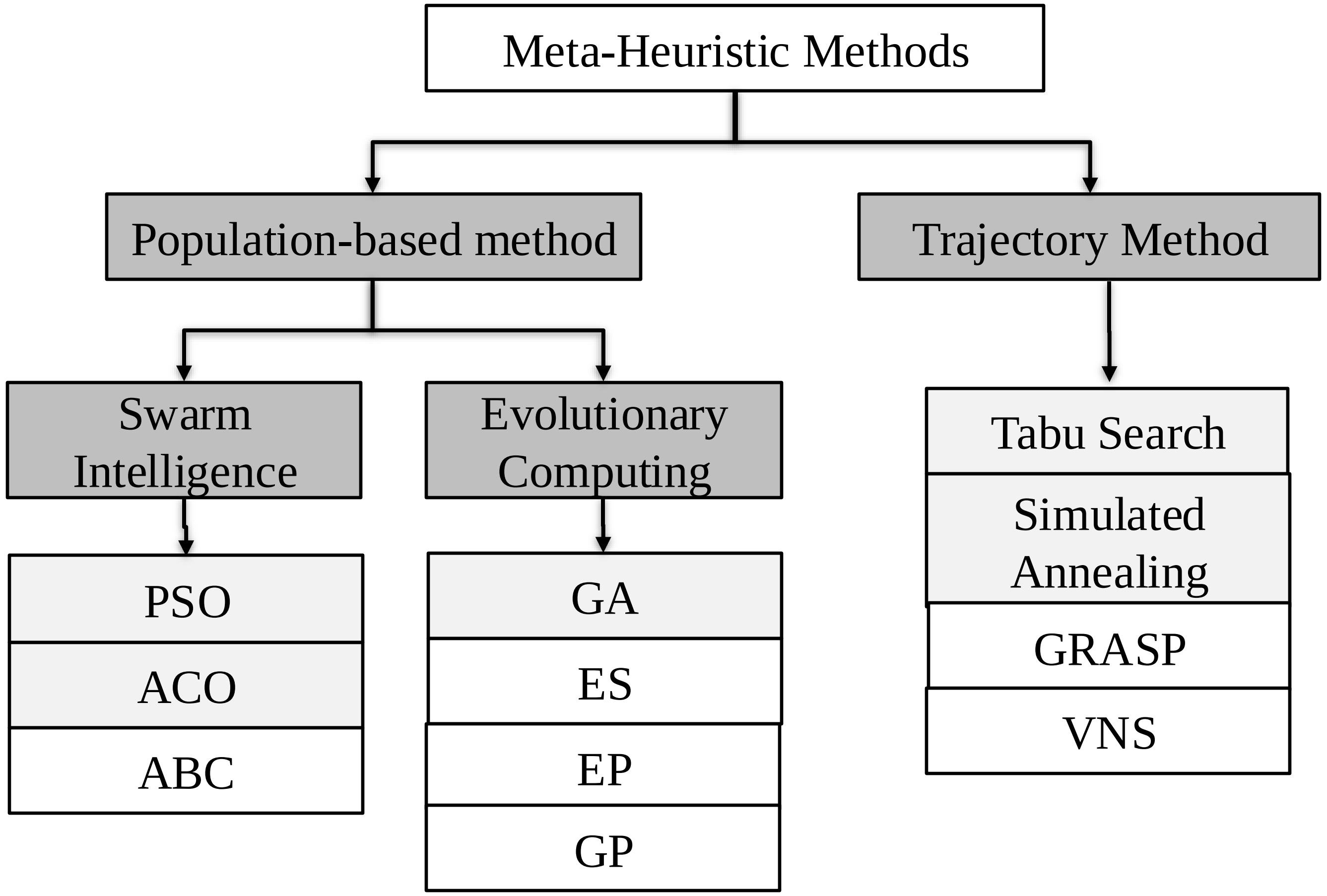
ECE457A Adaptive and Cooperative Algorithms

Population Based Algorithms

META-HEURISTICS



Evolutionary Algorithms

EVOLUTIONARY ANALOGY

A ***population*** of individuals exists in an environment with ***limited resources***

Competition for those resources causes selection of those ***fitter*** individuals that are better adapted to the environment

These individuals act as seeds for the ***generation of new individuals*** through recombination and mutation

The new individuals have their fitness evaluated and compete (possibly also with parents) for survival.

Over time ***Natural selection*** causes a rise in the fitness of the population

BIOLOGICAL INSPIRATION

Evolution:

- Darwin
- from bacteria to sponges, insects, fishes, and mammals
- from simple organs to complex ones
- from randomly spread neurons to highly organized large brains

Foundations:

- Nucleic acids
- DNA
- Chromosomes
- Genes
- RNA, proteins, cells

BIOLOGICAL INSPIRATION

Adaptation by evolution:

- Ecological niche: a set of ecological conditions (e.g., food resources, predators, other environmental risks, threats and opportunities)
- Conquering new ecological niches (e.g., islands)
- Development of new species that are able to use the opportunities provided by a new niche and avoid the related dangers

Adaptation:

- Development of new behaviours and organs;
- New cells and cell behaviours;
- New proteins;
- New genes;

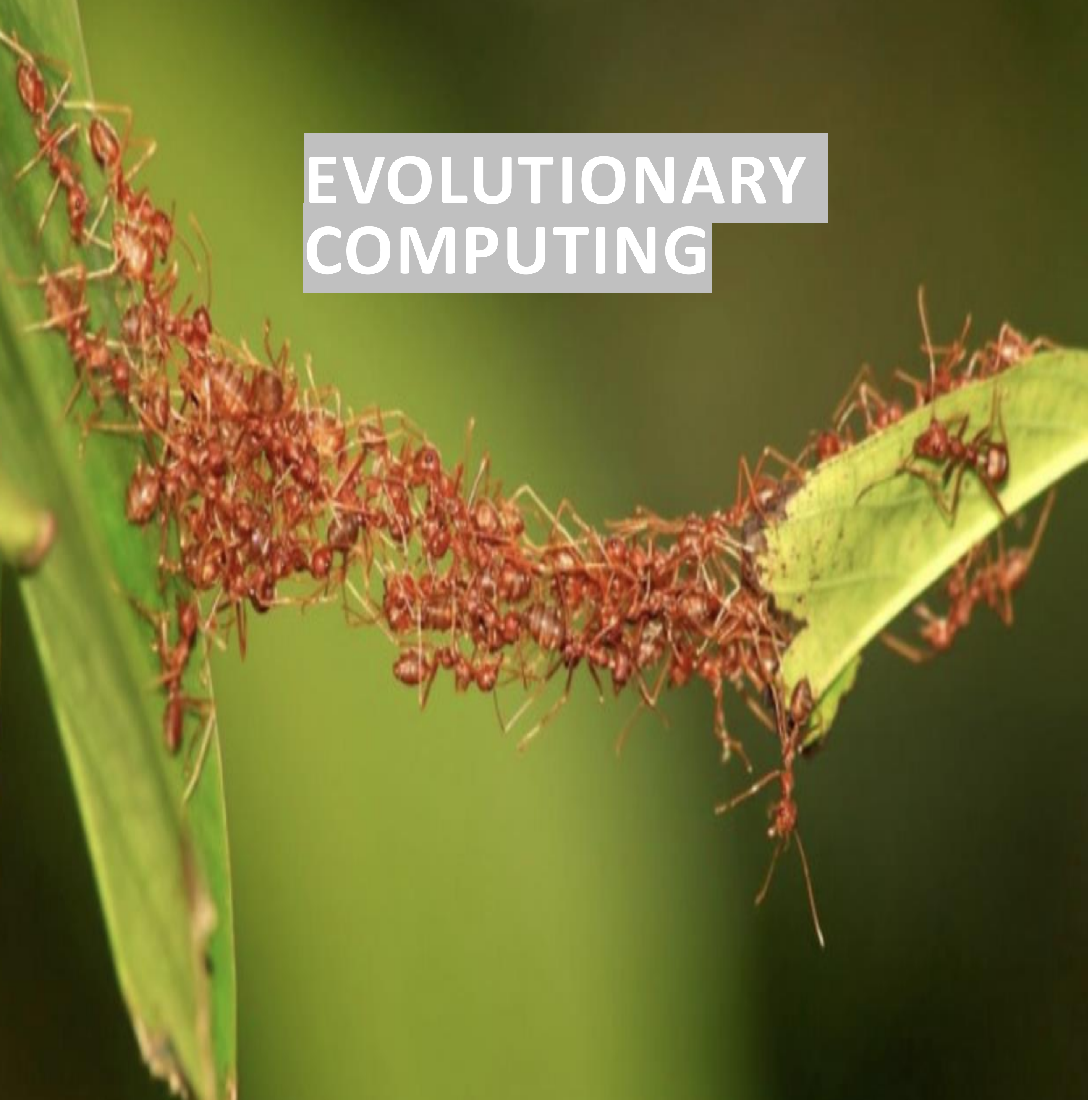
EVOLUTIONARY COMPUTING

Evolutionary Algorithms are stochastic, population-based algorithms

They fall into the category of “generate and test” algorithms

Variation operators (recombination and mutation) create the necessary diversity and thereby facilitate novelty

Selection reduces diversity and acts as a force pushing quality



EVOLUTIONARY COMPUTING

Idea: copying natural evolution by emulating genes and their evolution;

Objective: developing adaptive solutions of some problems;

Learning:

- Learning = adaptation
- Adaptation = optimisation
- Optimisation criteria: fitness in the given environmental conditions;

Active Information

INFORMATION CONSERVATION

Conservation of information theorems indicate that any search algorithm performs on average as well as random search unless it takes advantage of problem−specific information about the search target or the search−space structure.

Combinatorics shows that even a moderately sized search requires problem−specific information to be successful.

Three measures to characterize the information required for successful search:

(1) endogenous information, which measures the difficulty of finding a target using random search;

(2) exogenous information, which measures the difficulty that remains in finding a target once a search takes advantage of problem−specific information; and

(3) active information, which, as the difference between endogenous and exogenous information, measures the contribution of problem−specific information for successfully finding a target.

A methodology is developed based on these information measures to gauge the effectiveness with which problem−specific information facilitates successful search. It then applies this methodology to various search tools widely used in evolutionary search.

INFORMATION TRANSFORMATION

"The [computing] machine does not create any new information, but it performs a very valuable transformation of known information."

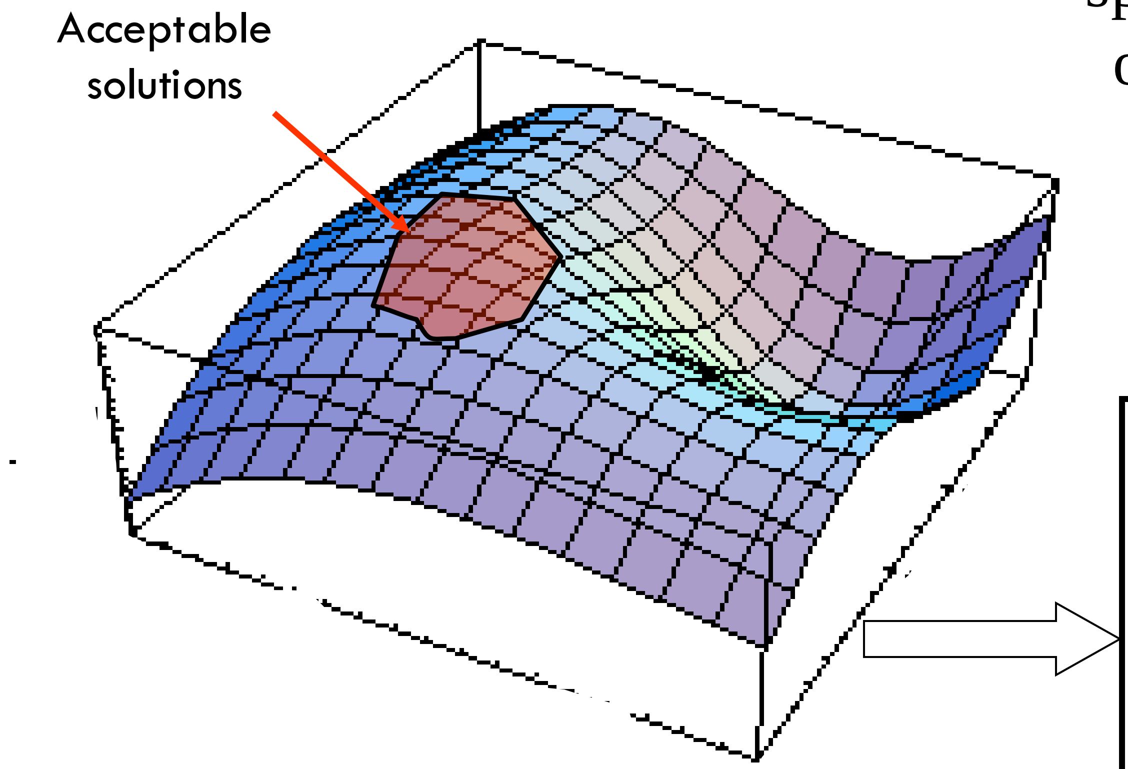
––Leon Brillouin, Science and Information Theory (Academic Press, New York, 1956).

BERNOULLI'S PRINCIPLE OF INSUFFICIENT REASON

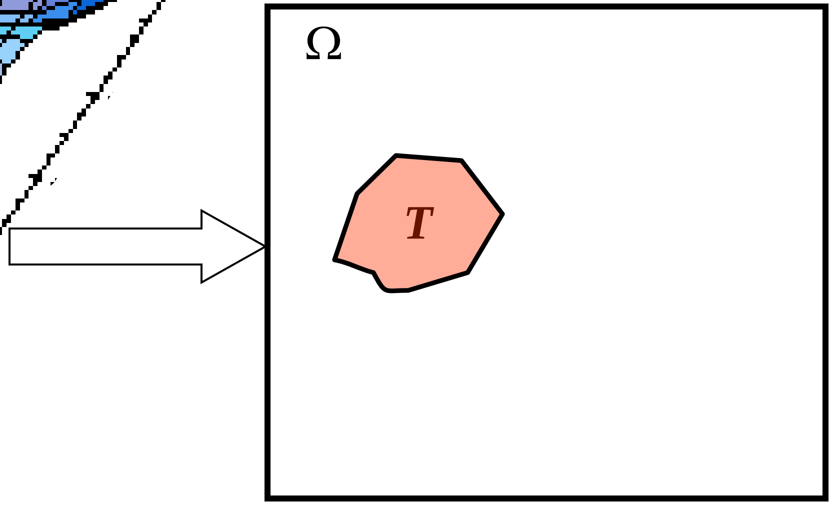
"in the absence of any prior knowledge, we must assume that the events have equal probability

Jakob Bernoulli, ``Ars Conjectandi'' (The Art of Conjecturing), (1713).

THE NOTION OF FITNESS

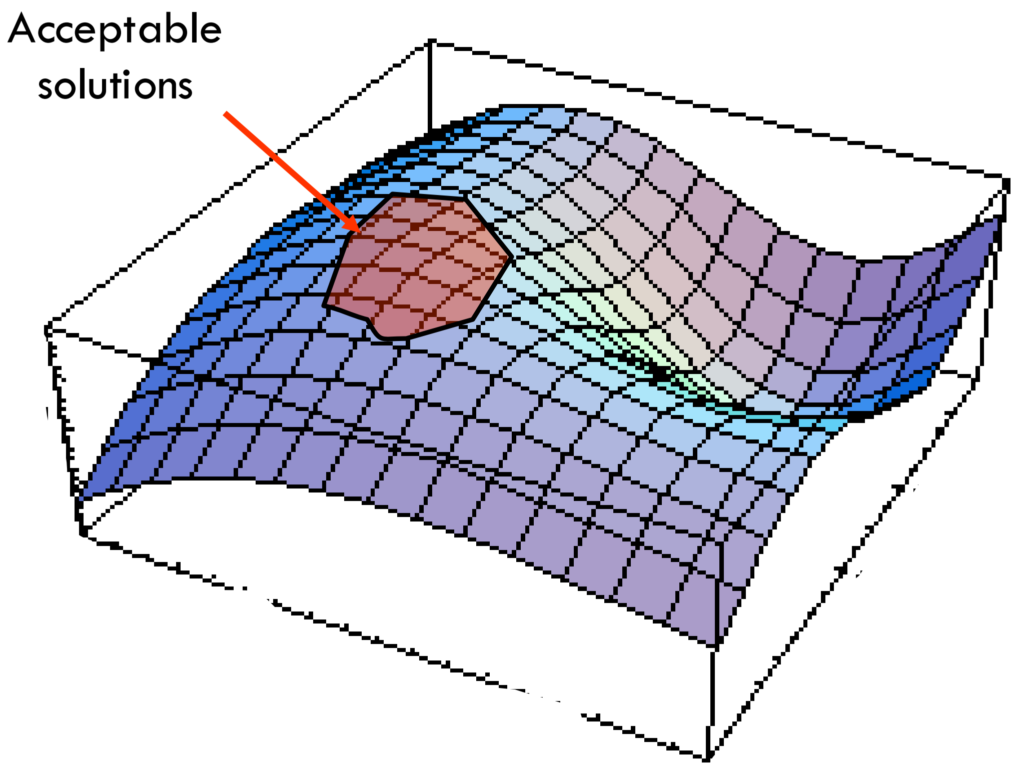


Each point in the parameter space has a fitness. The problem of the search is finding a good enough fitness.



SEARCH ALGORITHMS

Problem: In order to work better than average, each algorithm implicitly assumes something about the search space and/or location of the target.



- Steepest Ascent
- Exhaustive
- Newton-Rapheson
- Levenberg-Marquardt
- Tabu Search
- Simulated Annealing
- Particle Swarm Search
- Evolutionary Approaches

QUOTES ON THE NEED FOR ADDED INFORMATION FOR TARGETED SEARCH …

- “…unless you can make prior assumptions about the ... [problems] you are working on, then no search strategy, no matter how sophisticated, can be expected to perform better than any other” Yu-Chi Ho and D.L. Pepyne, (2001).

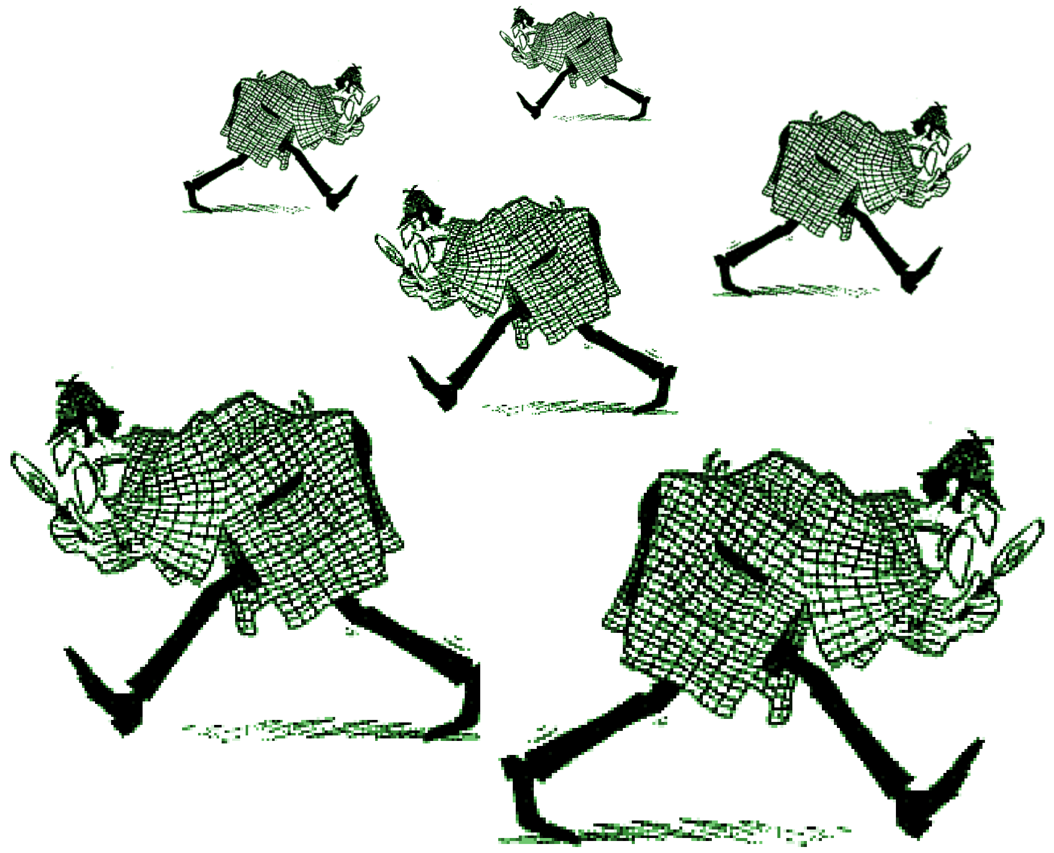
- No free lunch theorems “indicate the importance of incorporating problem-specific knowledge into the behavior of the [optimization or search] algorithm.” David Wolpert & William G. Macready (1997).

1. ``Simple explanation of the No Free Lunch Theorem", Proceedings of the 40th IEEE Conference on Decision and Control, Orlando, Florida,
2. "No free lunch theorems for optimization", IEEE Trans. Evolutionary Computation 1(1): 67-82 (1997).

ACTIVE INFORMATION

Nothing works better, on the average, than random search.

For a search algorithm like evolutionary search to work, we require active information.



ECE 457A ADAPTIVE AND COOPERATIVE ALGORITHMS

Genetic Algorithms

GENETIC ALGORITHMS

Based on the Darwinian theory of evolution,

- The theory offers an explanation of the biological diversity and its underlying mechanisms,

Natural selection plays an important role.

- Natural selection favours those individuals that ***compete*** for the given resource most effectively, those who are ***adapted*** to fit the environmental conditions best,

*This phenomenon is known as **survival of the fittest**.*

GENETIC ALGORITHMS

The fitness is determined by the ***phenotypic*** *traits*

- Behavioural features,
- Physical features.

Selected individuals reproduce, passing their properties to the offspring,

Other individuals die without mating, their properties are thus discarded.

GENETIC ALGORITHMS

- Darwin also recognized that small and random variations, ***mutations***, occur in the phenotypic traits during reproduction,

- Through these variations, new combinations of traits occur and get evaluated,

- The basic operations are hence ***selection***, ***reproduction*** and ***mutation***.

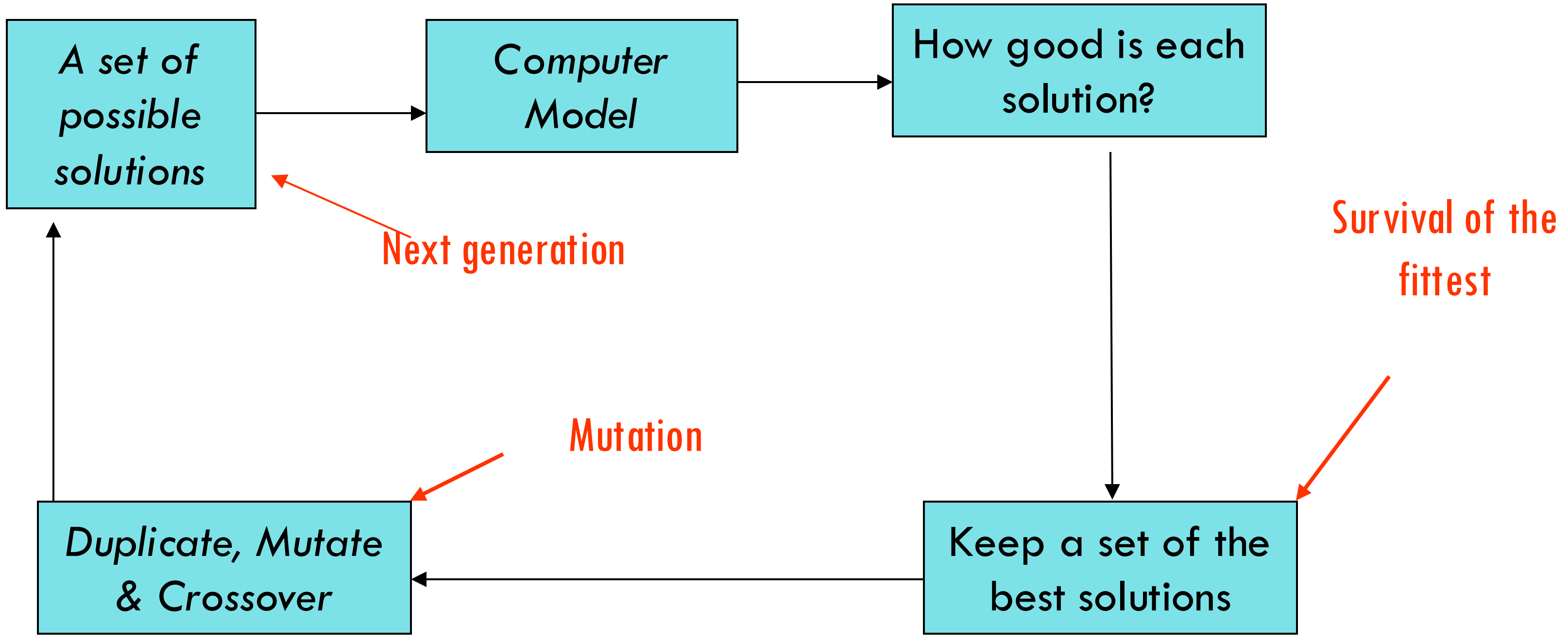
GENETIC ALGORITHMS

- A genetic algorithm maintains a **population of candidate solutions** for the problem at hand, and makes it evolve by ***iteratively applying a set of stochastic operators***

GAs		Problem solving
Environment	⟷	Problem
Individual	⟷	Candidate solution
Fitness	⟷	Quality
Population	⟷	Set of possible solutions, constant

Fitness	chances for survival and reproduction
Quality	chances for seeding new solutions

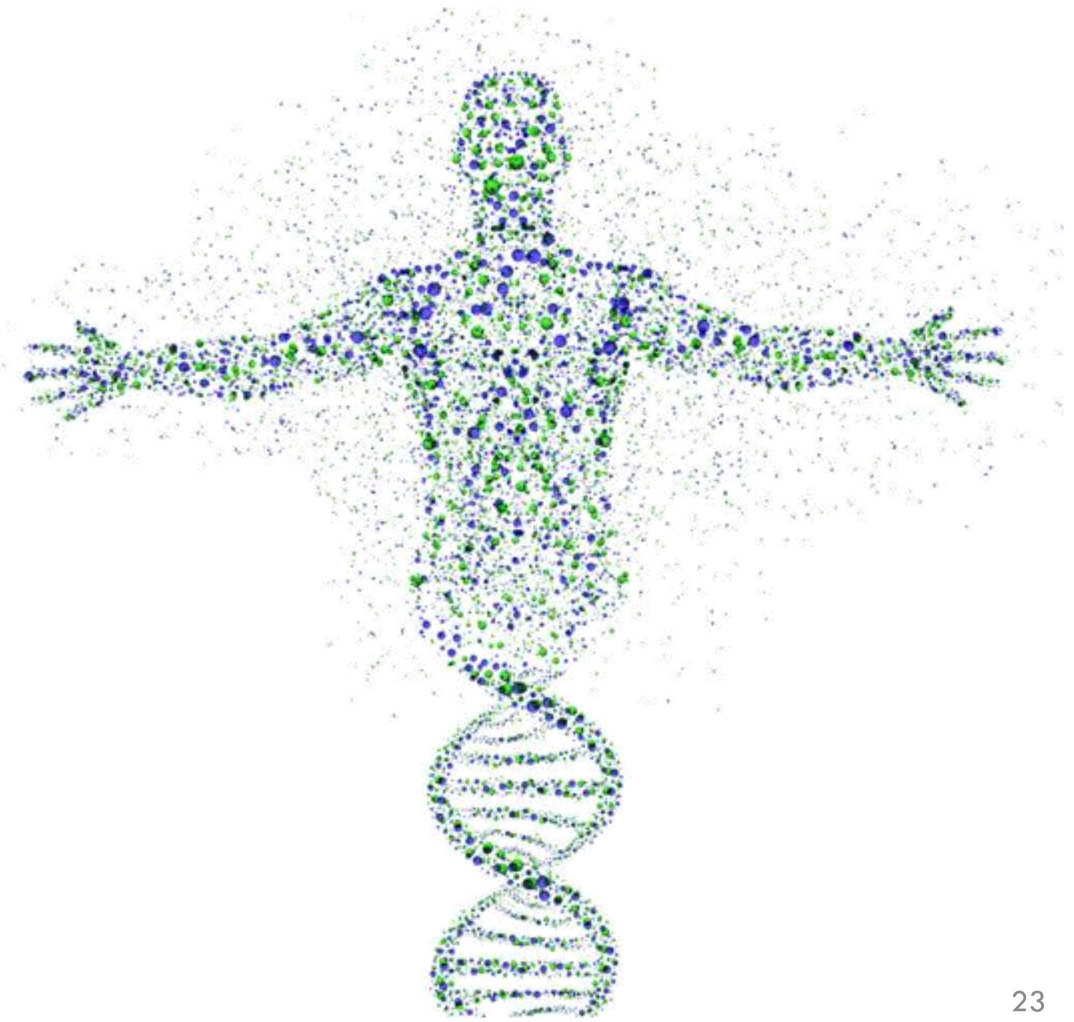
GENETIC ALGORITHMS


A set of possible solutions
Computer Model
How good is each solution?
Survival of the fittest
Keep a set of the best solutions
Mutation
Duplicate, Mutate & Crossover
Next generation

GENETIC ALGORITHMS

First GA was developed by Holland in 1975, referred to as **SGA** (Simple GA),

Sometimes referred to as the *classical* or *canonical* GA.



Simple Genetic Algorithms

Based on material and slides from A. E. Eiben and J. E. Smith. “Introduction to Evolutionary Computing”. Springer, 2003.

SGA - INTRODUCTION

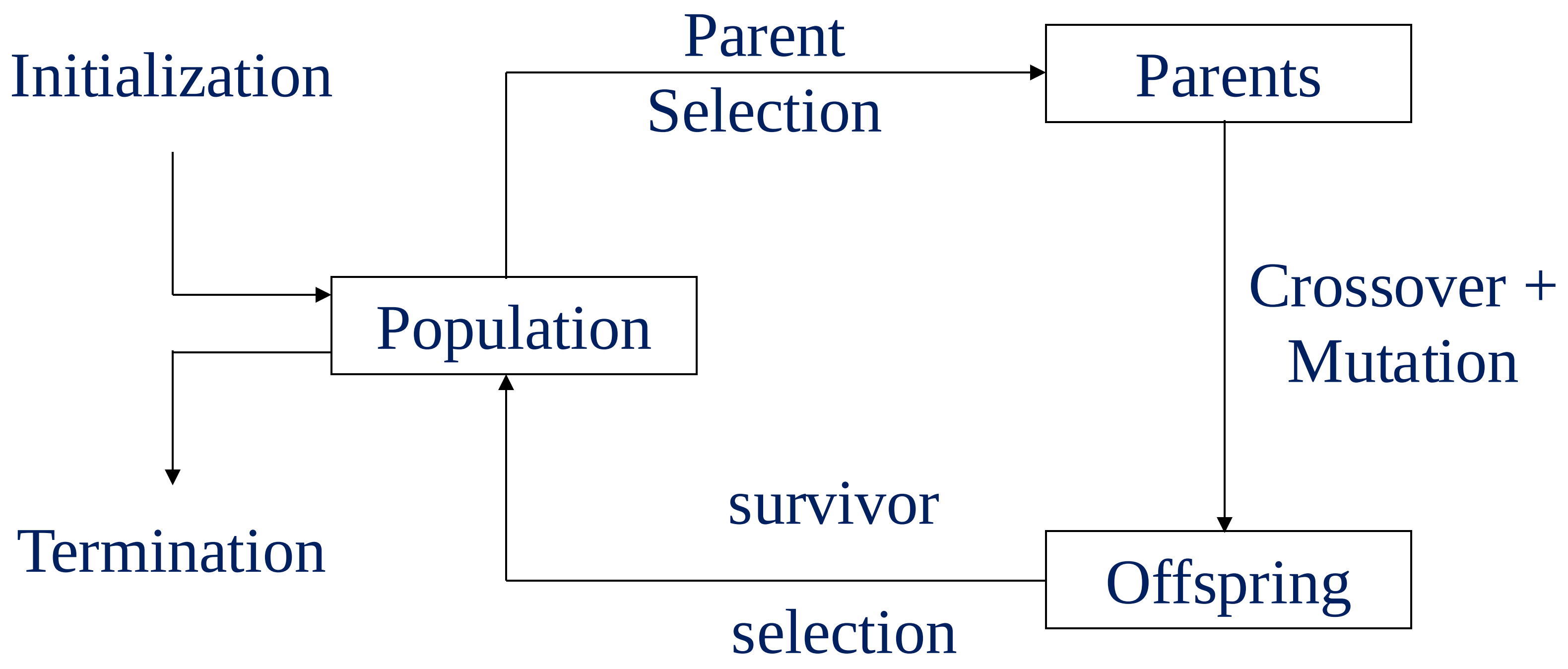
- GAs handle a *population* of *individuals* (solutions),
- The probability of selecting a bad solution is reduced by handling multiple good solutions,
- The population evolves from one iteration to the next according to selection and the ***search operators*** ***(genetic operators),***

Search operators:

- Recombination (Crossover),
- Mutation.

SGA - INTRODUCTION

The general scheme:



SGA INGREDIENTS

Representation	Binary Strings
Recombination	1-point, N-point, or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

SGA – THE ALGORITHM

```
Initialize population with random candidates,
Evaluate all individuals,
While *termination criteria* is not met
    ▪ Select parents,
    ▪ Apply crossover,
    ▪ Mutate offspring,
    ▪ Replace current generation,
end while
```

SGA – THE ALGORITHM

The *termination criteria* could be:

- *A specified number of generations (or fitness evaluations),*
- *A minimum threshold reached,*
- *No improvement in the best individual for a specified number of generations,*
- *Memory/time constraints,*
- *Combinations of the above.*

Simple Genetic Algorithms

Representation

SGA - REPRESENTATION

A mapping is applied from the parameter domain to the binary domain,

Phenotype, a vector in the parameter domain,

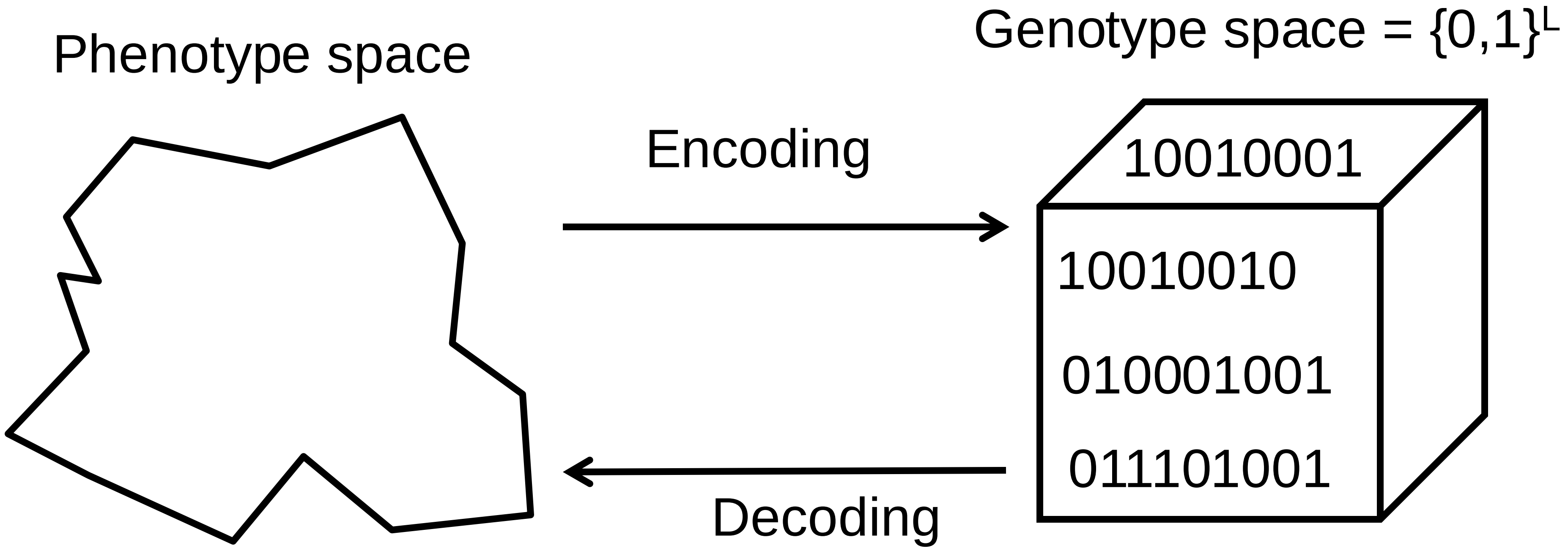
Abstract representation of Phenotype is called ***chromosomes*** or **genotype**

In SGA ***Genotype,*** is a string in the binary domain.

- 0 and 1 for the presence or absence of the features [Chromosomes: 001110101110]
- Order of genes on chromosome can be important

Good coding is probably the most important factor for the performance of a GA

SGA - REPRESENTATION



Simple Genetic Algorithms
Selection

SGA REPRODUCTION CYCLE

Select parents for the mating pool (size of mating pool = population size)

Shuffle the mating pool

Apply crossover for each consecutive pair with probability (p_c) otherwise copy parents

Apply mutation for each offspring (bit-flip with probability (p_m) independently for each bit)

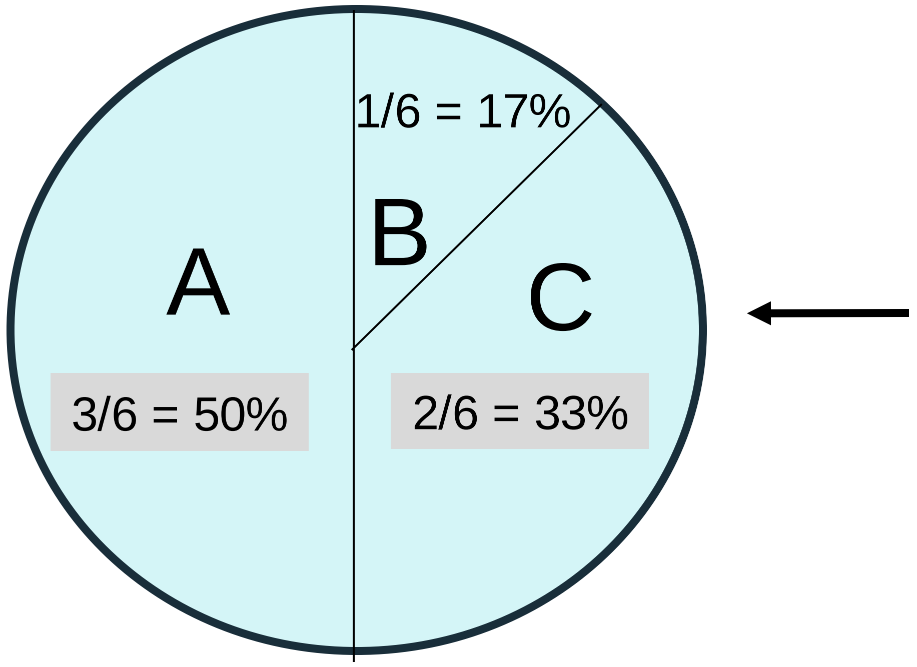
SGA - SELECTION

Parents are selected with a probability proportional to their fitness, ***proportional selection***.

Main idea: better individuals get higher chance:

- *Chances proportional to fitness,*
- *Implementation: roulette wheel technique*
  - *Assign to each individual a part of the roulette wheel,*
  - *Spin the wheel n times to select n individuals.*

SGA - SELECTION



Fitness(A) = 3

Fitness(B) = 1

Fitness(C) = 2

Simple Genetic Algorithms

Crossover

SGA - CROSSOVER

Crossover is applied according to a probability (***P_c***)

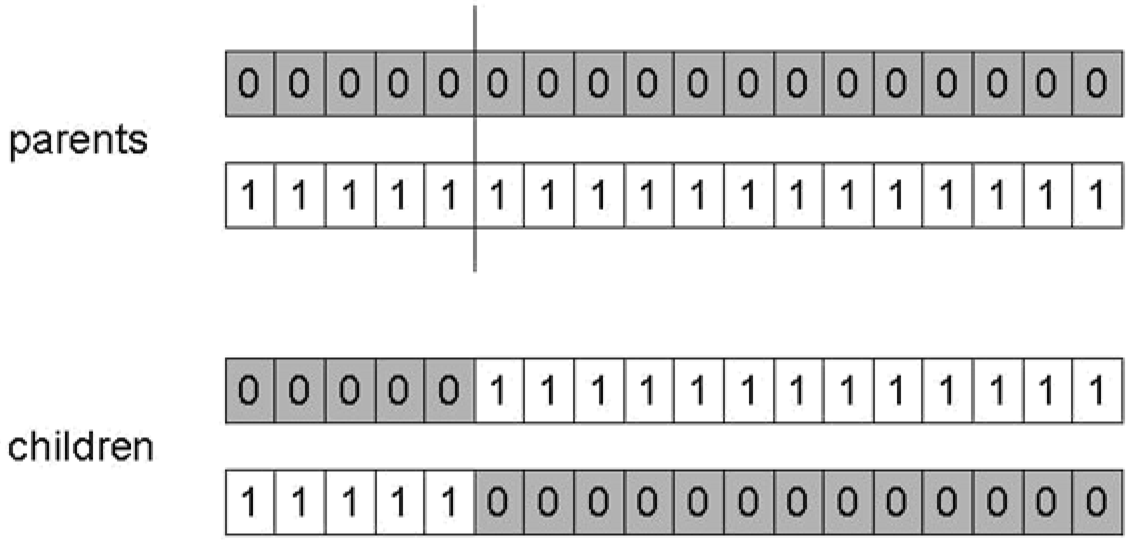
Otherwise, the two parent are copied to the offspring unmodified

P_c typically in range (0.6, 0.9)

1-point crossover

(the binary case)

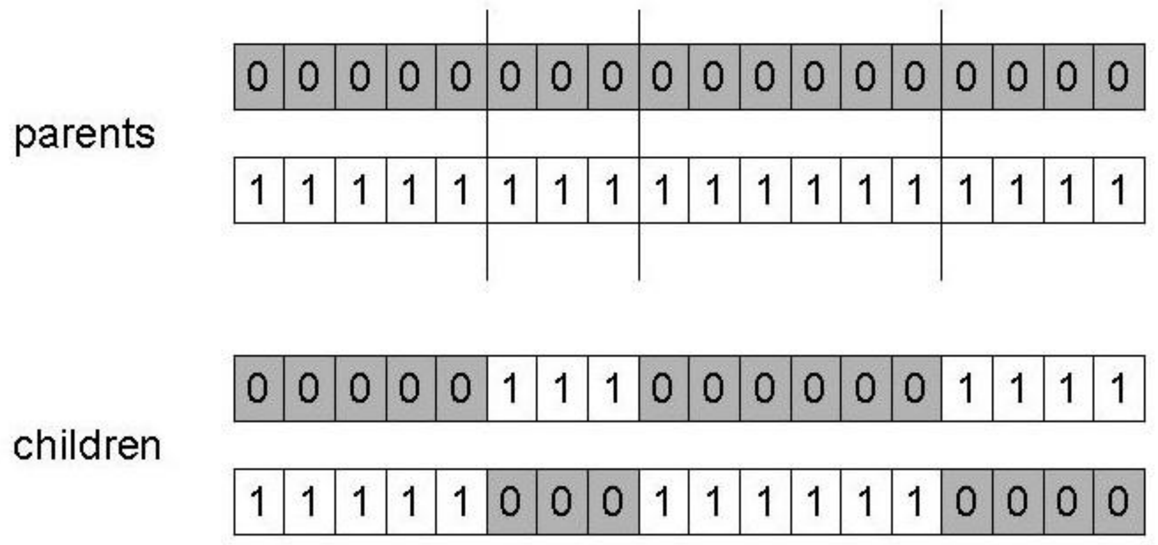
- Choose a random point on the ***two*** parents
- Split parents at this crossover point
- Create ***two*** children by exchanging tails



SGA - CROSSOVER

n-point crossover (the binary case)

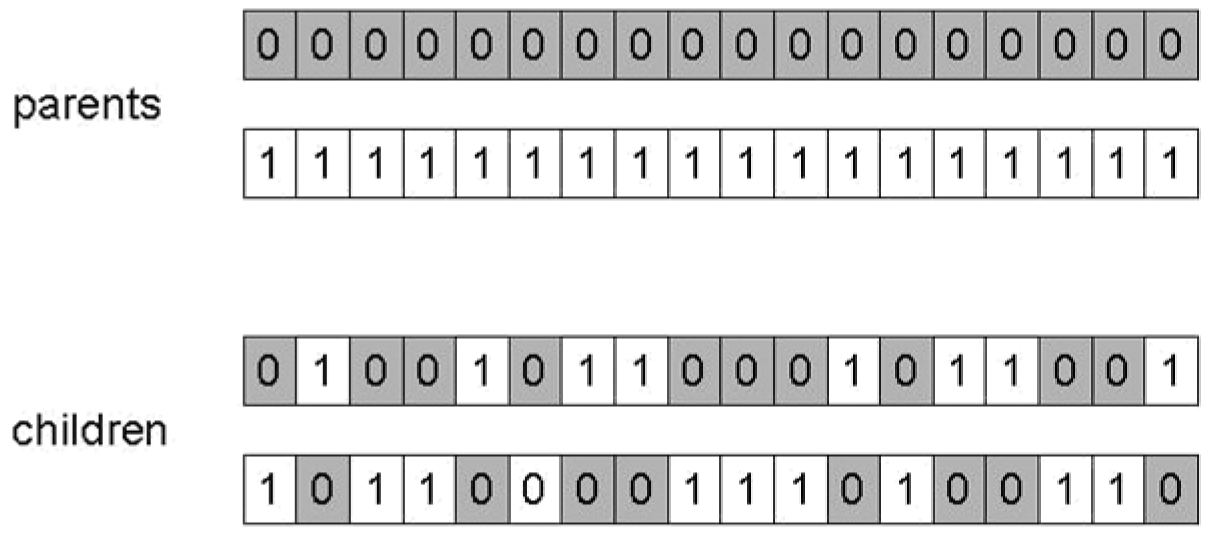
- Choose n random crossover points,
- Split along those points,
- Glue parts, alternating between parents,
- Generalisation of 1 point (still some positional bias).



SGA - CROSSOVER

Uniform crossover (the binary case)

- Treats genes independently
- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position.
- Distributional bias:
  - avoids transmitting co-adapted genes



Simple Genetic Algorithms

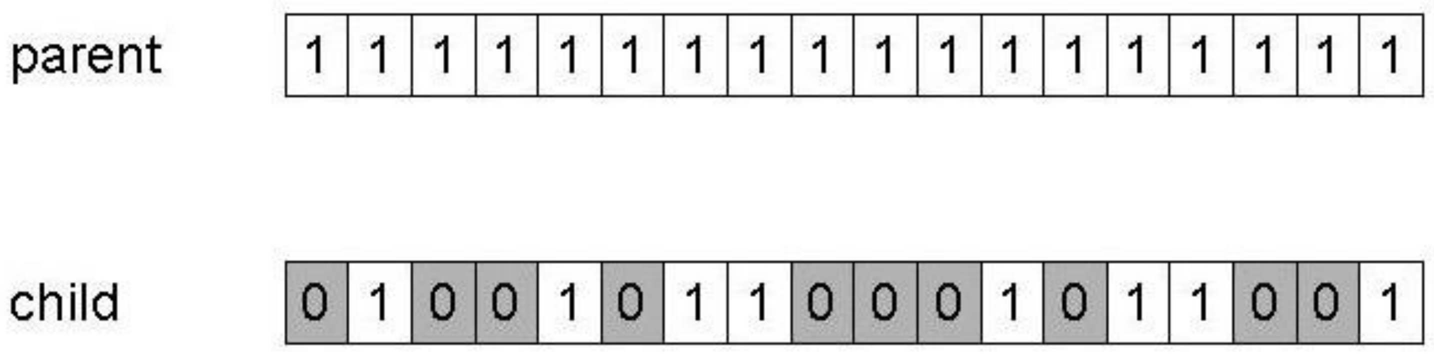
Mutation

SGA - 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



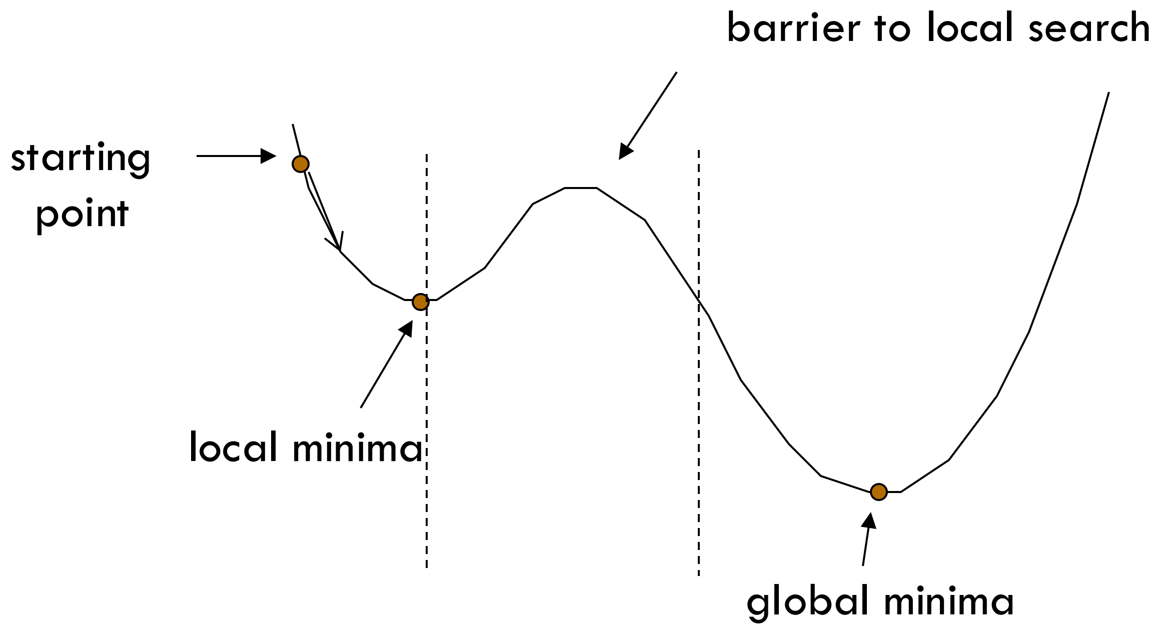
WHY CROSSOVER AND MUTATION?

Exploration:

Discovering promising areas in the search space.

Exploitation:

Optimizing within a promising area.



WHY CROSSOVER AND MUTATION?

Crossover is ***explorative***, it makes a big jump to an area somewhere between the two parent areas,

Mutation is ***exploitative***, it creates random small diversions, thereby staying near (in the area of) the parent.

Only crossover can combine information from two parents,

Only mutation can introduce new information.

Genetic Algorithms

Alternative Approaches

ALTERNATIVE BINARY REPRESENTATION

Gray coding of integers

- ◎ Gray coding is a mapping that means that small changes in the genotype cause small changes in the phenotype (unlike binary coding).
- ◎ "Smoother" genotype-phenotype mapping makes life easier for the GA

Gray Code:

0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

Hamming distance between any consecutive numbers is 1

GAS – OTHER REPRESENTATIONS

Other representations are also possible:

- Integers,
- Real-valued or floating-point numbers,
- Permutations.

The crossover and mutation actual implementation depends on the representation.

POPULATION MODELS

SGA uses a Generational model (GGA):

- Each individual survives for exactly one generation
- The entire set of parents is replaced by the offspring

Steady-State model:

- Only part of the population is replaced by the offspring,
- Usually one member of population is replaced,
- The proportion of the population replaced is called the ***Generational Gap***
    - 1.0 for GGA
    - 1/pop_size for SSGA

PARENTS AND OFFSPRING SELECTION

Selection can occur in two places:

- Selection from current generation to take part in mating (parent selection)
- Selection from parents + offspring to go into next generation (survivor selection)

PARENT SELECTION

Fitness-Proportionate Selection (FPS)

Probability of parent selection

$$p_i = f_i \big/ \sum f_j$$

Expected number of copies of an individual *i*

$$E(n_i) = f_i / \bar{f}$$

f is the fitness and $\bar{f}$ is the average fitness

FPS

Roulette wheel algorithm:

- ◎ Given a probability distribution, spin a 1-armed wheel n times to make n selections
- ◎ No guarantees on actual value of n_i

Baker's Stochastic Universal Sampling algorithm:

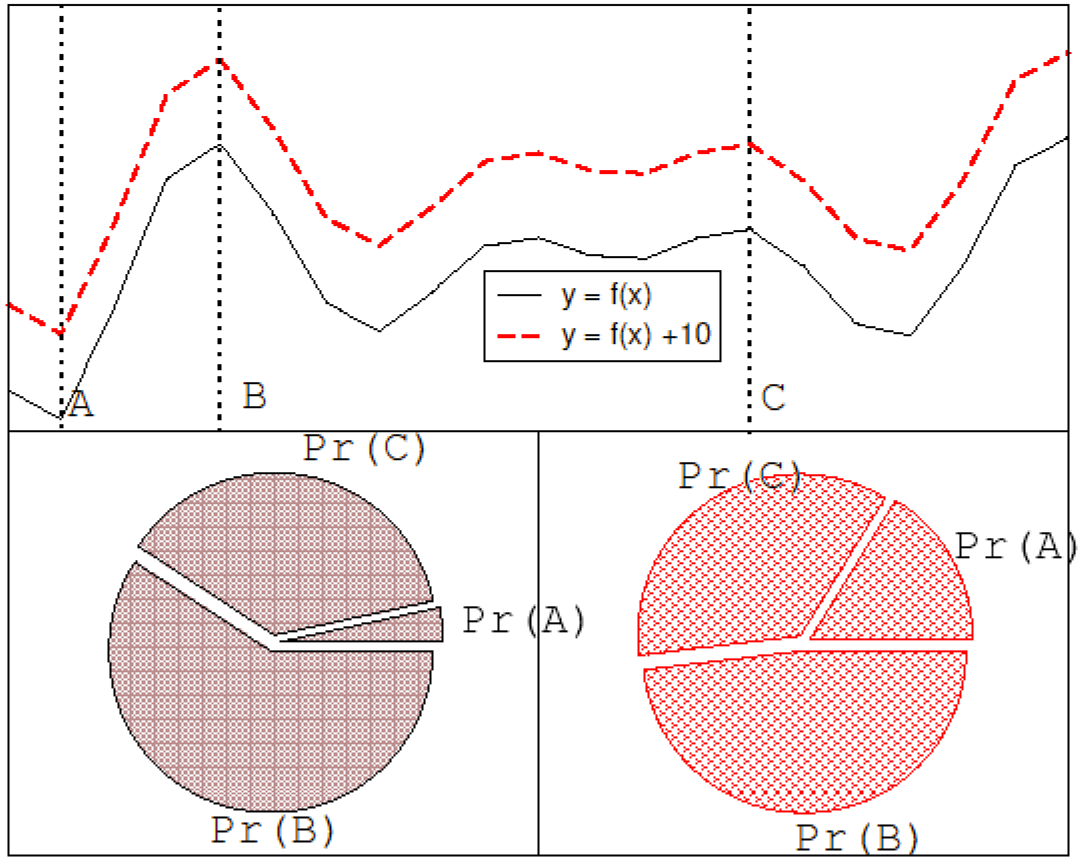
- ◎ n evenly spaced arms on wheel and spin once
- ◎ Guarantees $floor(E(n_i)) \leq n_i \leq ceil(E(n_i))$

FPS

Problems include:

- ***Premature Convergence:*** One highly fit member can rapidly take over if the rest of population is much less fit
- ***No selection pressure:*** At end of runs when fitnesses are similar
- Highly susceptible to ***function transposition***

FUNCTION TRANSPOSITION FOR FPS



Example

A=	2	12
B=	20	30
C=	8	18

T	30	60
A/T	1/15	1/5
B/T	2/3	1/2
C/T	4/15	3/10

RANK–BASED SELECTION

Attempt to remove problems of FPS by basing selection probabilities on ***relative*** rather than ***absolute*** fitness

Rank population according to fitness and then base selection probabilities on rank where fittest has rank μ and worst rank ***1***

This imposes a sorting overhead on the algorithm, but this is usually negligible compared to the fitness evaluation time

TOURNAMENT SELECTION

All methods above rely on ***global population statistics***

- Could be a ***bottleneck*** especially on parallel machines
- Relies on presence of ***external fitness function*** which might not exist: e.g. evolving game players

In tournament selection:

- Pick ***k*** members at random then select the best of these
- Repeat to select more individuals

SURVIVOR SELECTION

Survivor selection can be divided into two approaches:

- ***Age-Based Selection***
  - e.g. GGA
  - In SSGA can implement the selection as “delete-random” or as first-in-first-out (a.k.a. delete-oldest)

- ***Fitness-Based Selection (FPS)***
  - Delete or replace based on inverse of fitness or

FPS

Elitism

- Widely used in both population models (GGA, SSGA)
- Always keep best individuals (at least one copy of the fittest solution so far)

GENITOR: a.k.a. “delete-worst”

- Can lead to rapid improvement and potential premature convergence
- used with large populations or “no duplicates” policy

Simple Genetic Algorithms

A Sample Run

SGA – A SAMPLE RUN

Maximize the value of x^2 over for integers in the range {0,1,..,31}

GA approach:

◎Representation: Binary individuals (5 bits),

◎Population: 4 individuals,

◎Recombination: 1-point crossover,

◎Selection: Roulette wheel selection (fitness proportional) =>

◎Survivor selection: replace all parents by offspring

◎Mutation: uniform random number > threshold

$$p_i = \frac{f_i}{\sum f_i}$$

SGA – A SAMPLE RUN

Initial population, evaluation for parent selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Expected Count= $f_i / \bar{f}$

Actual count = round (Expected count)

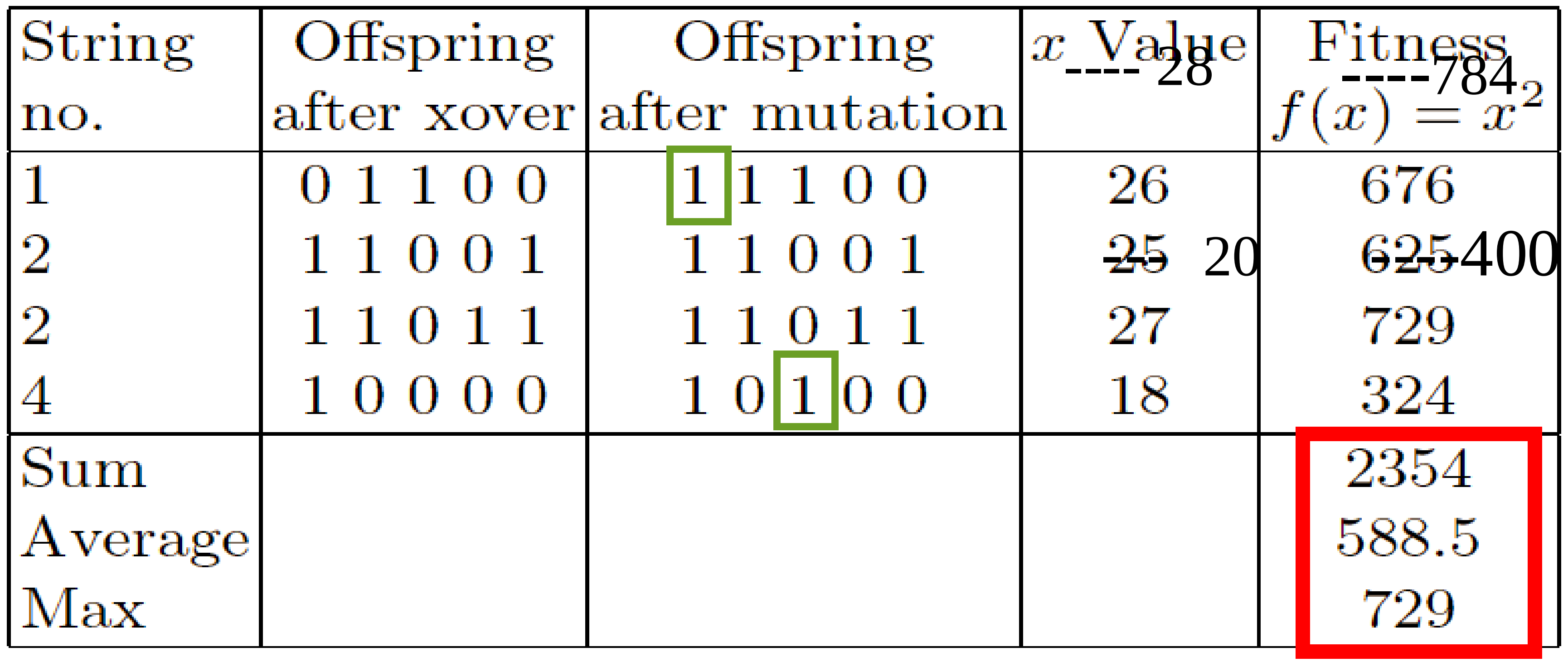
SGA – A SAMPLE RUN

Crossover and offspring evaluation

String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	$0\ 1\ 1\ 0 \mid 1$	4	0 1 1 0 0	12	144
2	$1\ 1\ 0\ 0 \mid 0$	4	1 1 0 0 1	25	625
2	$1\ 1 \mid 0\ 0\ 0$	2	1 1 0 1 1	27	729
4	$1\ 0 \mid 0\ 1\ 1$	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

SGA – A SAMPLE RUN

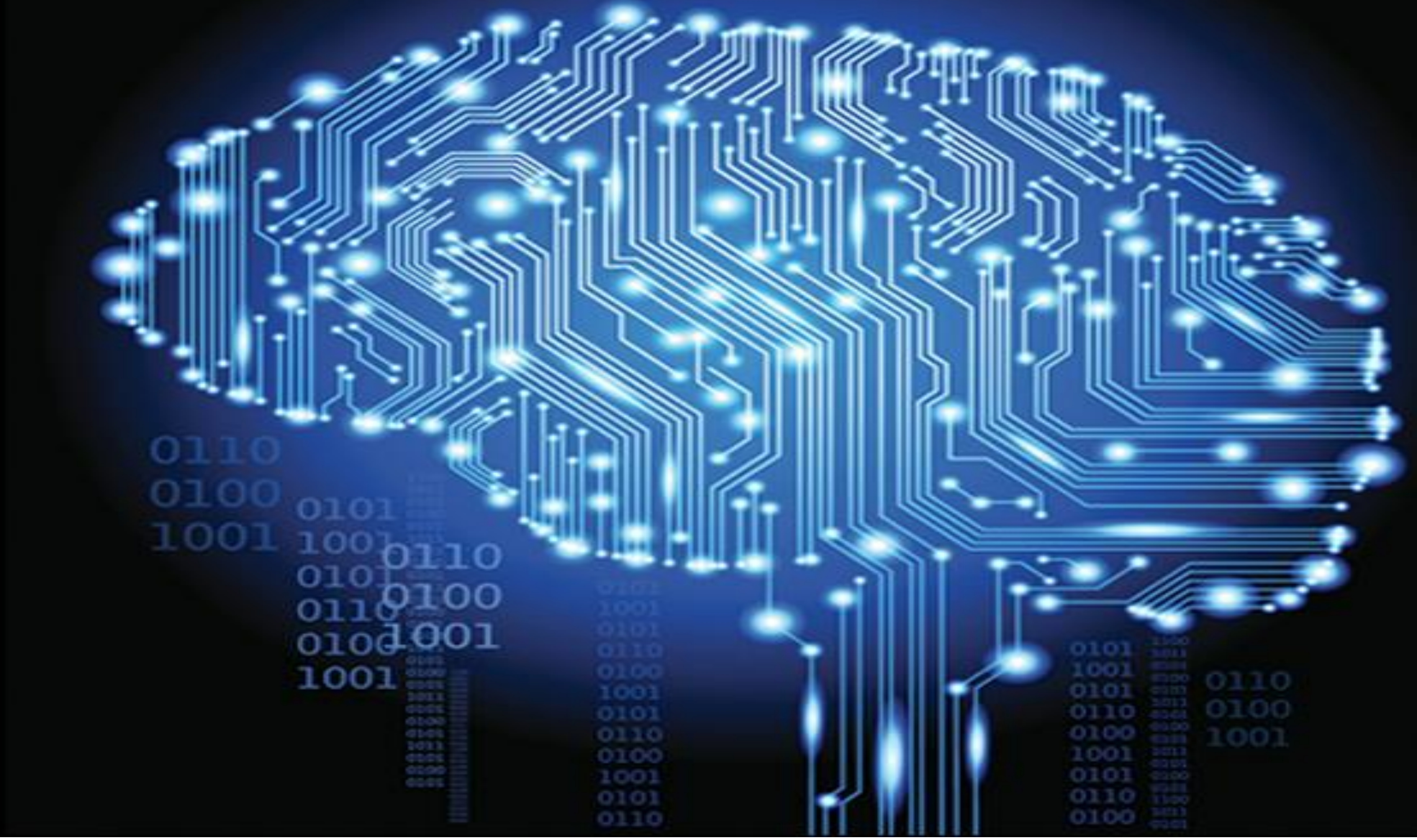
Mutation and offspring evaluation



String no.	Offspring after xover	Offspring after mutation	x Value ----28	Fitness ----784 $f(x) = x^2$
1	0 1 1 0 0	$\boxed{1}$ 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	~~25~~ 20	~~625~~400
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 $\boxed{1}$ 0 0	18	324
Sum				2354
Average				588.5
Max				729

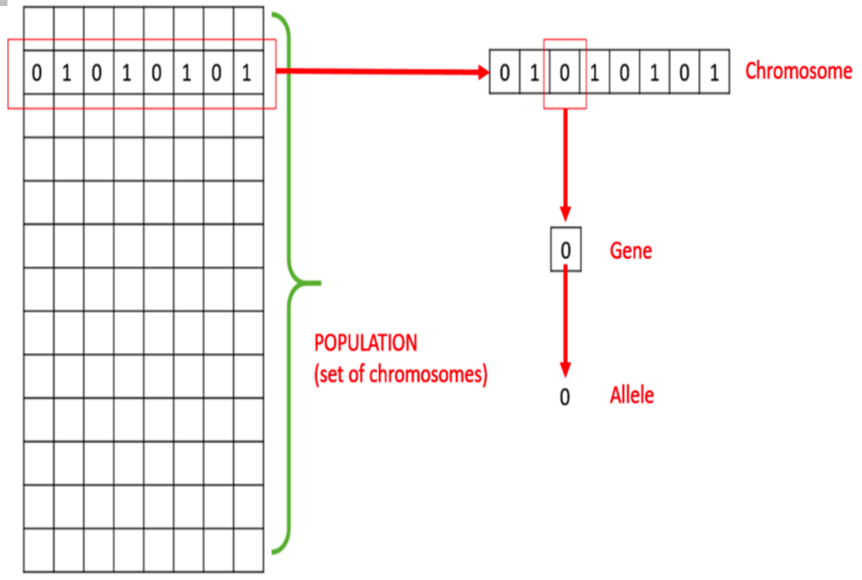
2538
634.5
784

- The algorithm has shown progress:
  - Average fitness grows form 293 to 634.5 and best fitness from 576 to 784.



REAL-VALUED GENETIC ALGORITHMS

GA: STRATEGY AND MECHANICS….. SO FAR



Strategy: explore and exploit existing diversity in a population of solutions to achieve progressive solution enhancement.

Mechanics:

- Multi-candidate Solution creation or initiation- initial population.
- Selection of off-spring next generation producers- parent selection
- Selection criterion: Fitness
- Next generation production: crossover
- Next generation (parents) alteration/enhancement: mutation
- Crossover is ***explorative*** and mutation is ***exploitative***.

◎ Crossover:

- 1-point crossover: Choose a random point on the two parents, split parents at this crossover point, exchanging tails
- N-point crossover: Choose n random crossover points, split along those points, alternating between parents,
- Uniform crossover: Assign 'heads' to one parent, 'tails' to the other, Flip a coin for each gene

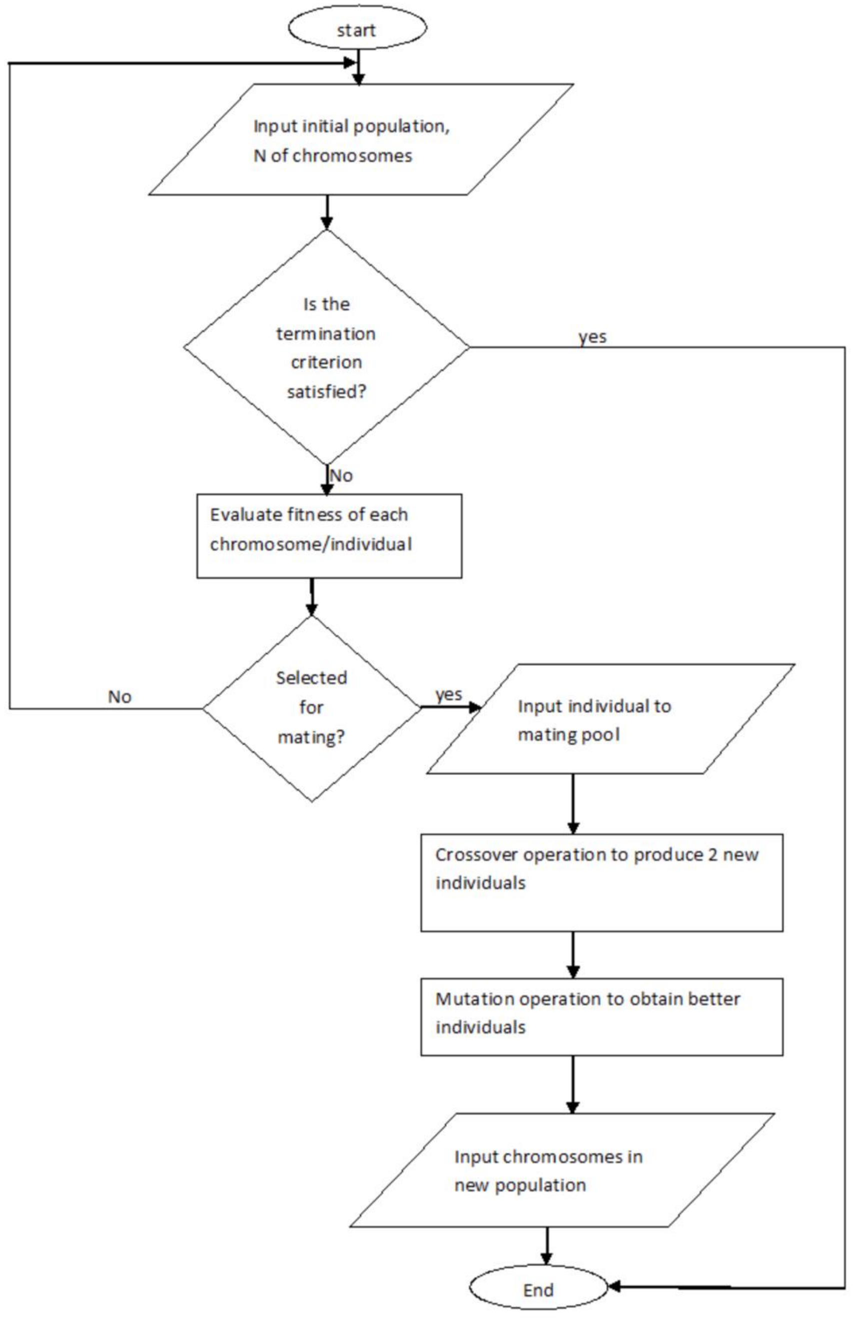
◎ Mutation: A GA operator. It alters each gene independently with a probability

◎ Generation Gap: The proportion of the population replaced in each iteration.

Solution representation (chromosome): bits (genes) of binary values 0/1.

MECHANICS PROCESS

- We start with an initial population (which may be generated at random or seeded by other heuristics),
- Select parents from this population for mating.
- Apply crossover and mutation operators to generate new off-springs.
- Finally these off-springs replace (some or all) of the existing individuals in the population and the process repeats.
- In this way genetic algorithms actually try to mimic the human evolution to some extent.


start
Input initial population, N of chromosomes
Is the termination criterion satisfied?
yes
No
Evaluate fitness of each chromosome/individual
Selected for mating?
No
yes
Input individual to mating pool
Crossover operation to produce 2 new individuals
Mutation operation to obtain better individuals
Input chromosomes in new population
End

ECE 457A ADAPTIVE AND COOPERATIVE ALGORITHMS

REAL-VALUED GA'S

REAL-VALUED GAS

Many problems have real-valued parameters,

If mapped to a binary space, high precision solutions would require very long chromosomes,

How to combine and mutate real-valued solutions?

- Real-valued crossover
- Real-valued mutation

0	0	1	0	1	1	1	0	0	1

Binary Representation

✓

0.5	0.2	0.6	0.8	0.7	0.4	0.3	0.2	0.1	0.9

Real-valued Representation

1	2	3	4	3	2	4	1	2	1

Integer Representation

1	5	9	8	7	4	2	3	6	0

Permutation Representation

REAL-VALUED GAS – CROSSOVER

Two types of real-valued crossover operators:

- ◎ Discrete: use any of the crossover operators identified before for binary representations

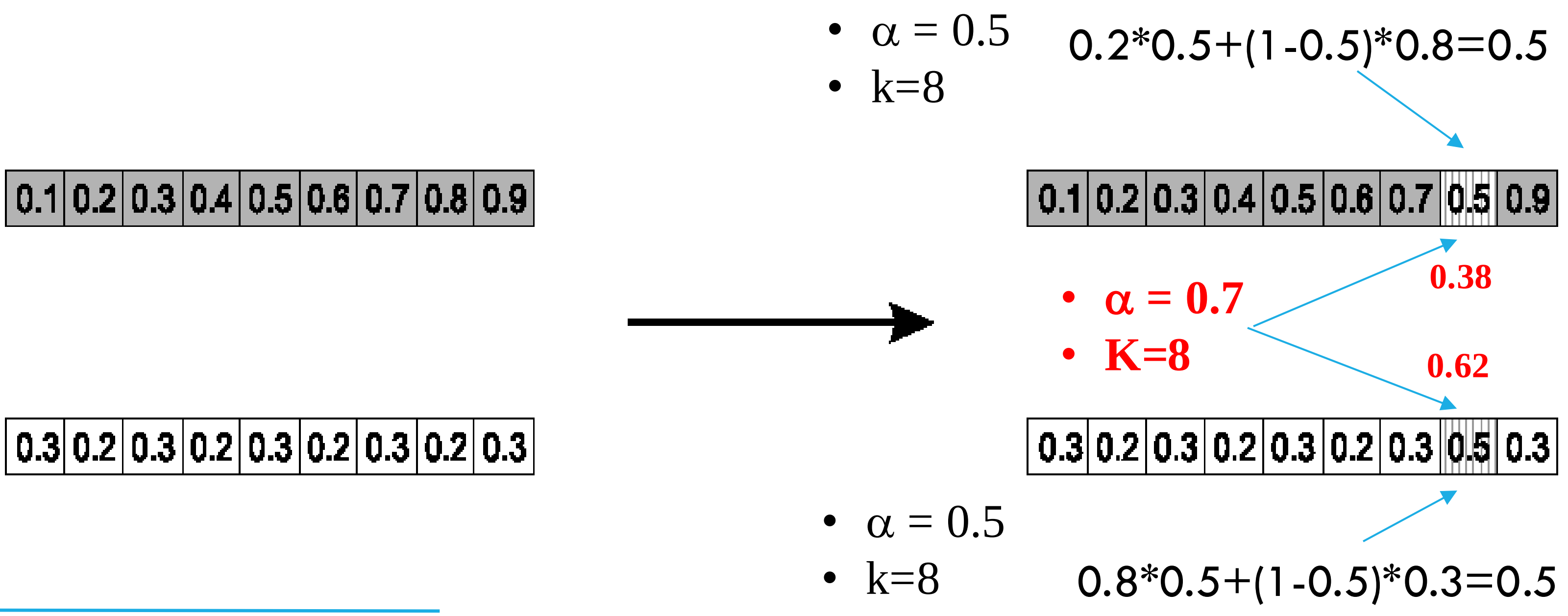
- ◎ Intermediate (arithmetic):
  - Single arithmetic
  - Simple arithmetic
  - Whole arithmetic.

REAL-VALUED GAS – CROSSOVER

Single arithmetic crossover:

- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Pick random gene (k) at random,
- Child 1 is: $$\left\langle x_1,\ \ldots,\ x_{k-1},\ \alpha \cdot y_k + (1-\alpha)\cdot x_k,\ \ldots,\ x_n \right\rangle$$
- Reverse for other child.

Example:



REAL-VALUED GAS – CROSSOVER

Simple arithmetic crossover:

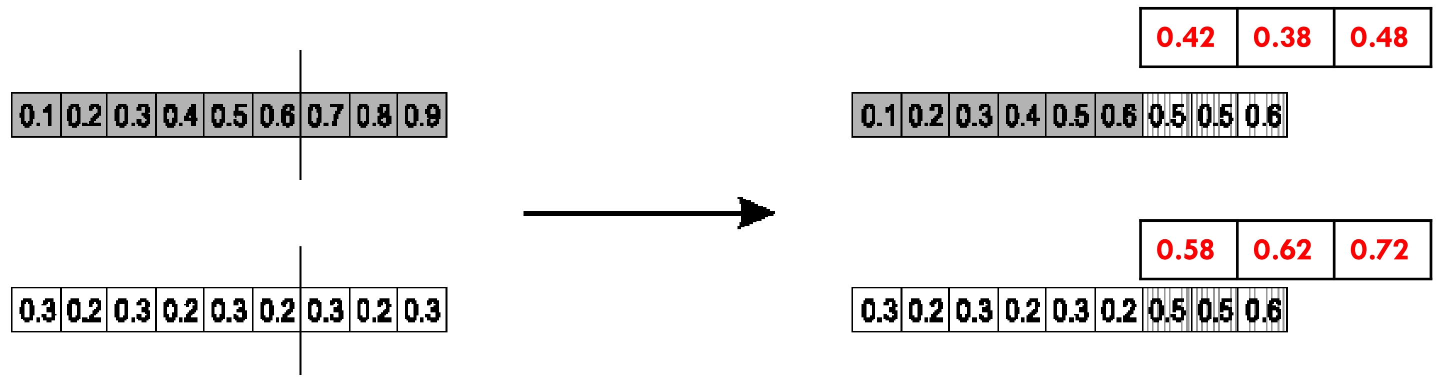
- ◎ Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$,
- ◎ Pick random gene (k), after this point mix the values,
- ◎ Child 1 is:

$$\langle x_1,\ldots,x_k,\alpha\cdot y_{k+1}+(1-\alpha)\cdot x_{k+1},\ldots,\alpha\cdot y_n+(1-\alpha)\cdot x_n\rangle$$

- ◎ Reverse for other child.

Example:

- $\alpha = 0.5$
- k=6
- **$\alpha = 0.7$**

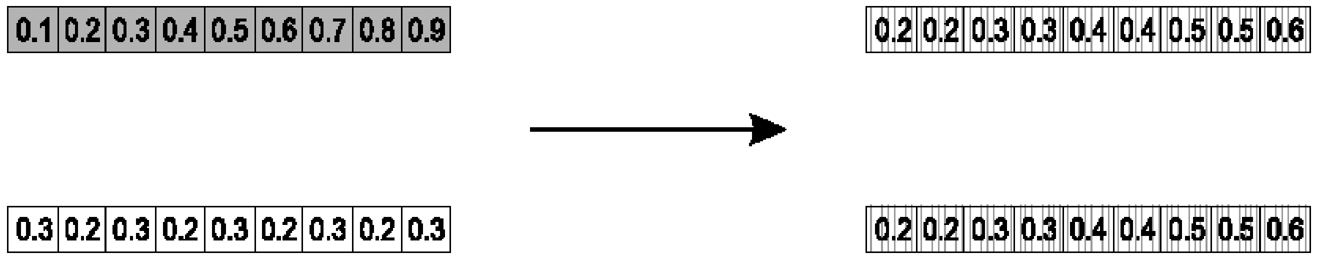


REAL-VALUED GAS – CROSSOVER

Whole arithmetic crossover:

- Most commonly used
- Parents: $\langle x_1,\ldots,x_n \rangle$ and $\langle y_1,\ldots,y_n \rangle$
- Child 1 is: $$a \cdot \bar{x} + (1-a) \cdot \bar{y}$$
- Reverse for other child.

Example:



- $\alpha = 0.5$

REAL-VALUED GAS – MUTATION

The general scheme for mutation (called uniform mutation) is :

$$\bar{x} = \langle x_1, \ \ldots, \ x_l \rangle \rightarrow \bar{x}' = \langle x_1', \ldots, \ x_l' \rangle$$
$$x_i, x_i' \in [LB_i, UB_i]$$

x_i' drawn randomly (uniform) from $[LB_i, UB_i]$

Analogous to bit-flipping mutation.

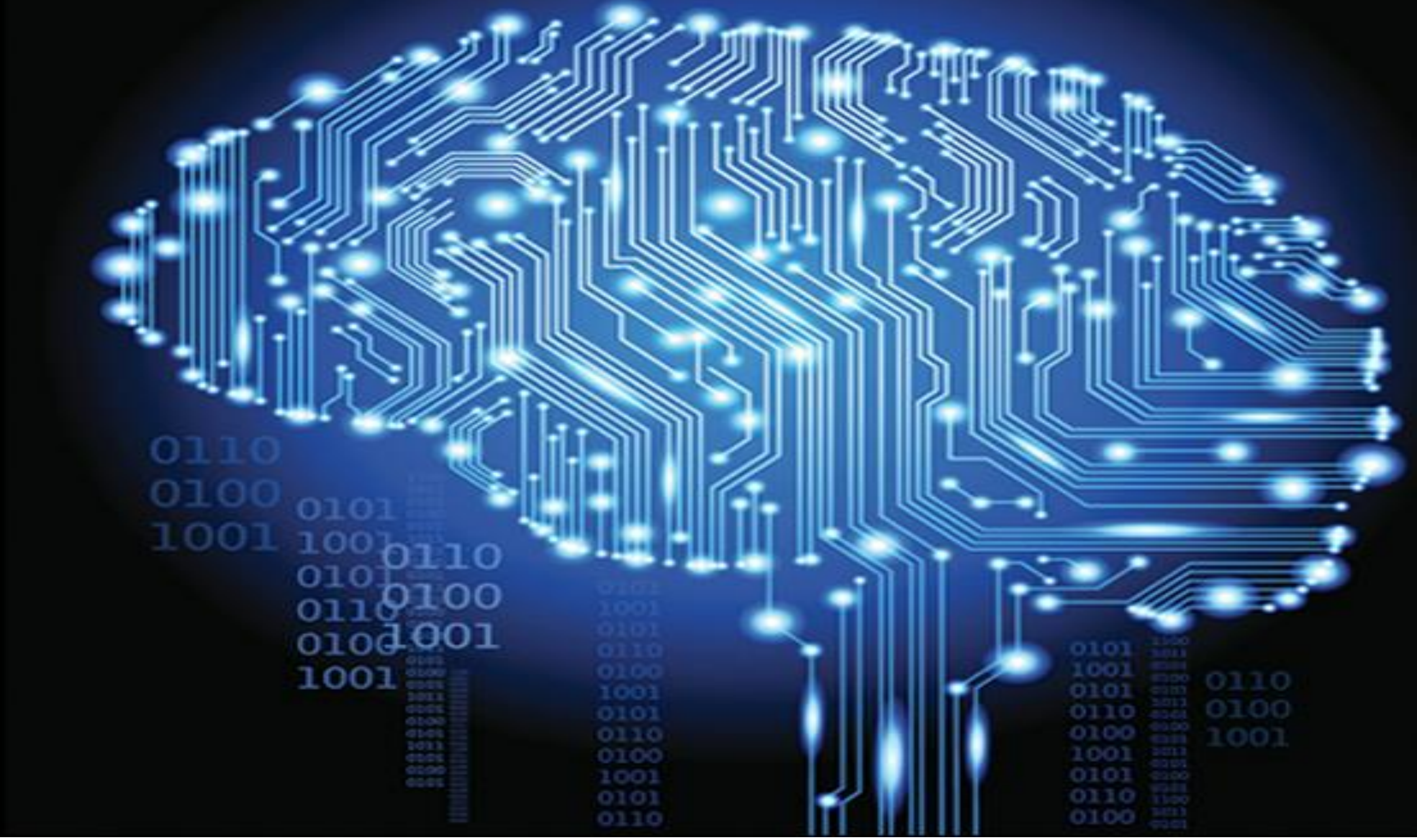
REAL-VALUED GAS – MUTATION

Another mutation scheme is to add random noise, for each gene:

$$x_i' = x_i + N(0,\sigma)$$

where *N(0,σ)* is a random Gaussian number with mean zero and standard deviation *σ*.

This scheme falls into the non-uniform mutation schemes



PERMUTATIONS GENETIC ALGORITHMS

PERMUTATIONS GAS

For problems that take the form of deciding on the order in which a sequence of events should occur

Two types of problems exist:

- ◎ Events use limited resources or time. The ***order*** of events is important. e.g. Job Shop Scheduling,
- ◎ The ***adjacency*** of elements is important. e.g. Travelling Salesman Problem.

If there are *n* variables (e.g. cities) then the representation is a list of *n* integers, each of which occurs ***exactly once***

5 4 3 2 1 6 10 9 8 7

PERMUTATIONS GAS – TSP EXAMPLE

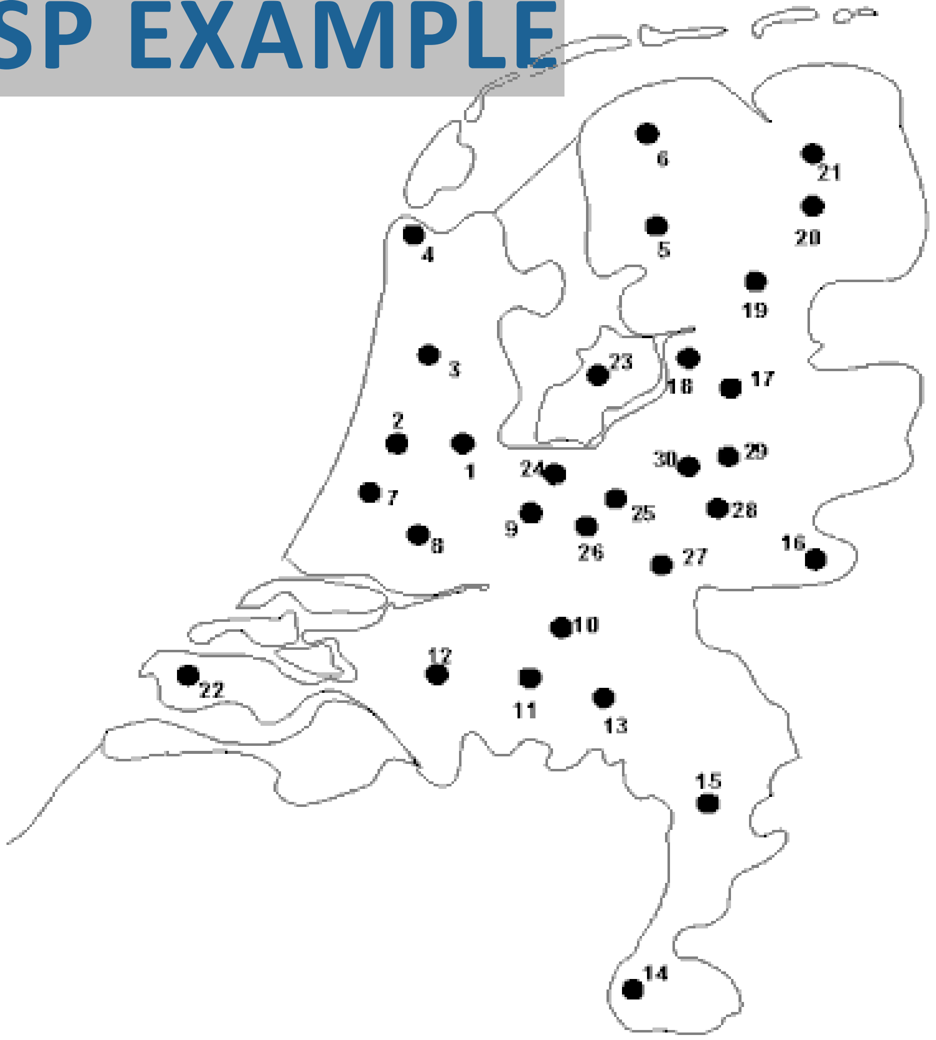
Problem:

- ◎ Given n cities,
- ◎ Find a complete tour with minimal length.

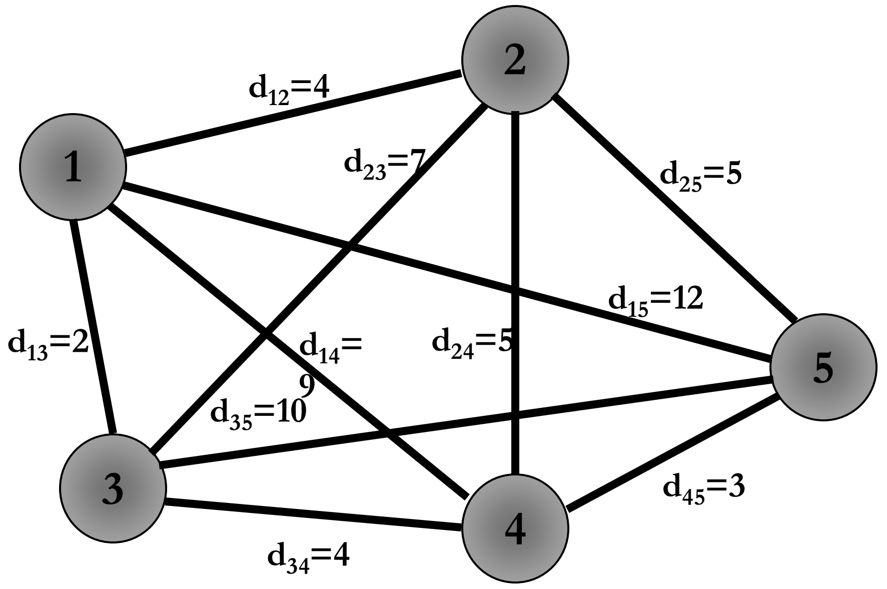
Encoding:

- ◎Label the cities 1,2,…,n
- ◎One complete tour is one permutation
  - [1,2,3,4], [3,4,2,1], …

Search space is **BIG**: for n cities there are (n-1)! possible tours.

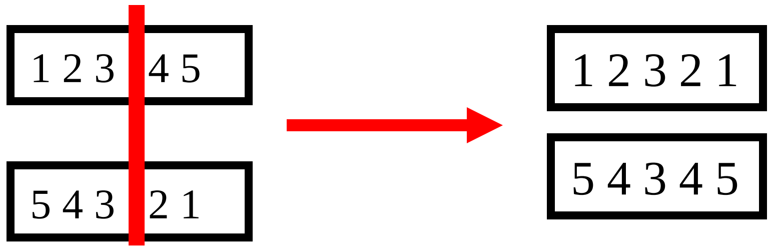


PERMUTATIONS GAS – TSP EXAMPLE



PERMUTATIONS GAS – TSP EXAMPLE

Previously defined crossover operators will often lead to inadmissible solutions,



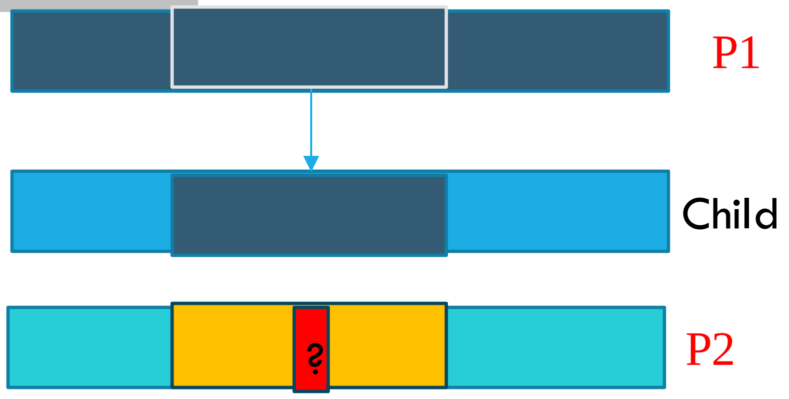
Specialized crossover operators have been devised.

PERMUTATIONS GAS – TSP EXAMPLE

Four types of crossover operators:

- Adjacency-based:
  - **Partially Mapped Crossover (PMX),**
  - Edge crossover.
- Order-based:
  - **Order 1 crossover,**
  - **Cycle crossover.**

PERMUTATIONS GAS – TSP EXAMPLE



PMX: Informal procedure for parents P1 and P2:

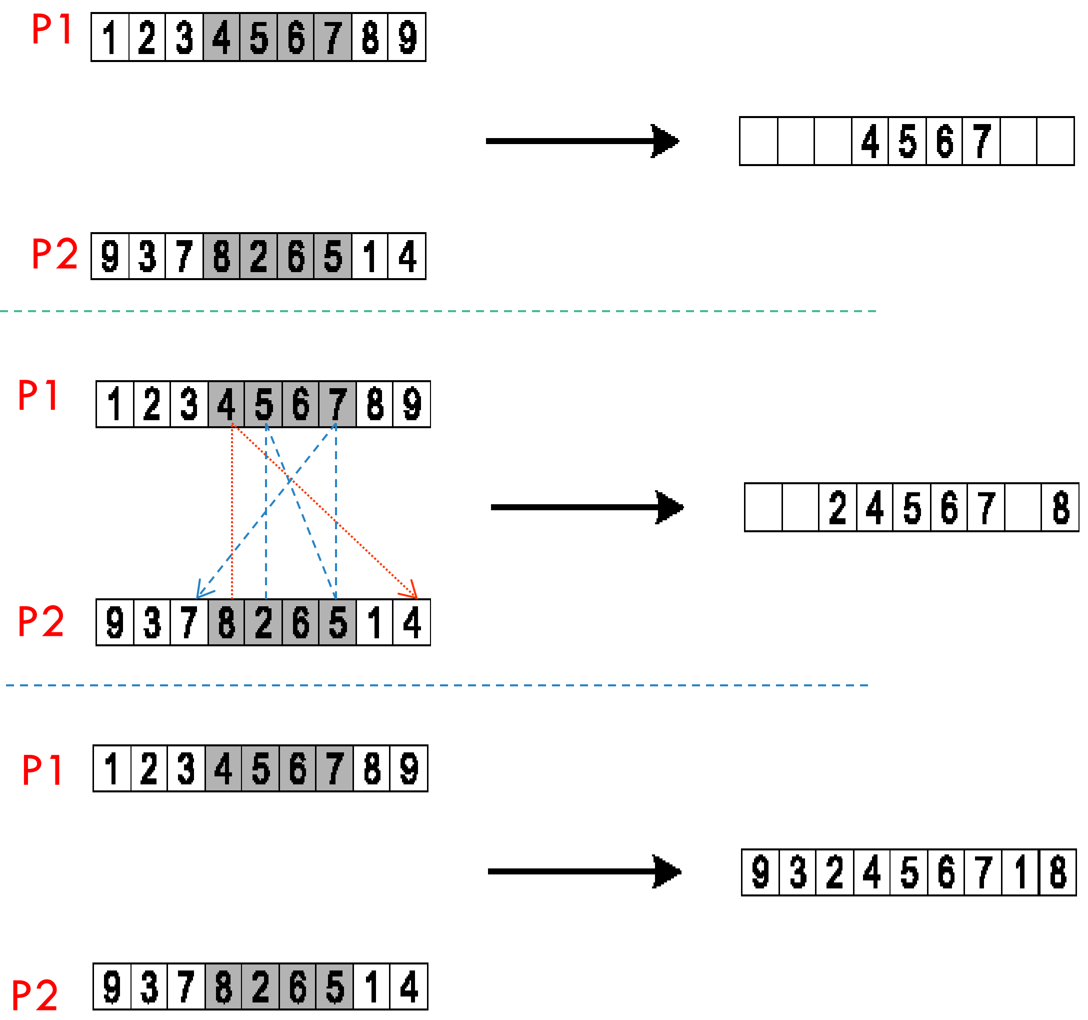
1. Choose random segment in child and copy it from P1,
2. Starting from the first crossover point look for elements in that segment of P2 that have not been copied,
3. For each of these *i*, look in the offspring to see what element *j* has been copied in its place from P1,
4. Place *i* into the position occupied by *j* in P2, since we know that we will not be putting *j* there (as is already in offspring).
5. If the place occupied by *j* in P2 has already been filled in the offspring *k*, put *i* in the position occupied by *k* in P2,
6. Having dealt with the elements from the crossover segment, the rest of the offspring can be filled from P2. Second child is created analogously.

PERMUTATIONS GAS – TSP EXAMPLE

Step 1

Step 2

Step 3



PERMUTATIONS GAS – TSP EXAMPLE

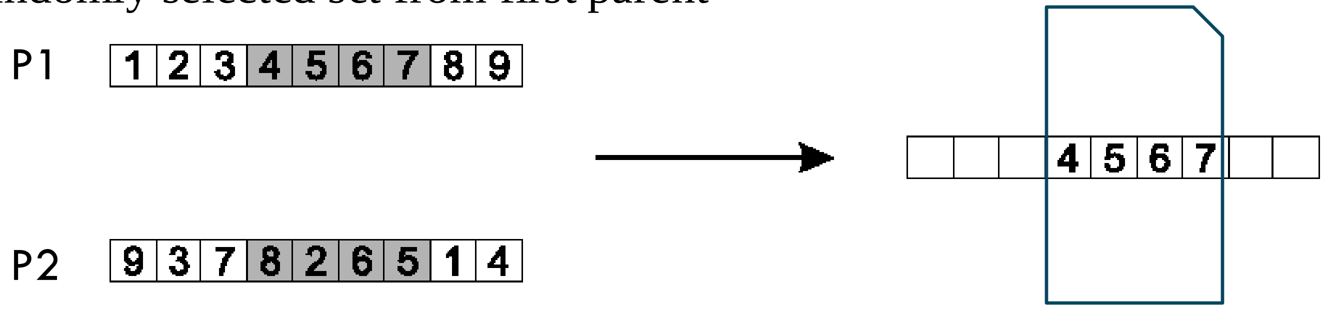
Order 1 crossover:

- The idea is to preserve relative order that elements occur

- Informal procedure:
  1. Choose an arbitrary part from the first parent,
  2. Copy this part to the first child,
  3. Copy the numbers that are not in the first part, to the first child:
     - Start right from cut point of the copied part,
     - Use the **order** of the second parent,
     - Wrap around at the end.
  4. Analogous for the second child, with parent roles reversed.

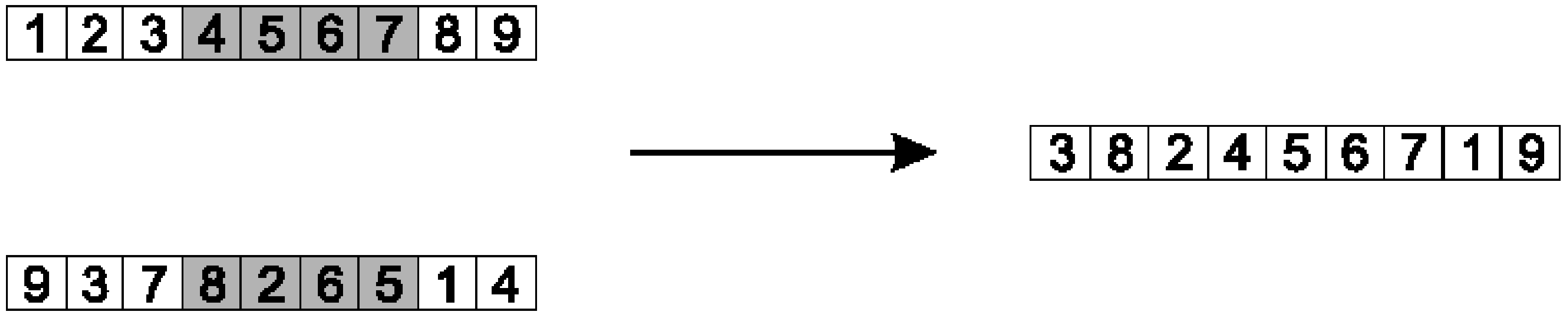
PERMUTATIONS GAS – TSP EXAMPLE

Copy randomly selected set from first parent



Missing to come form P2: 1,2,3,8,9

Copy rest from P2 in order 1,9,3,8,2



PERMUTATION GAS

Cycle Crossover :

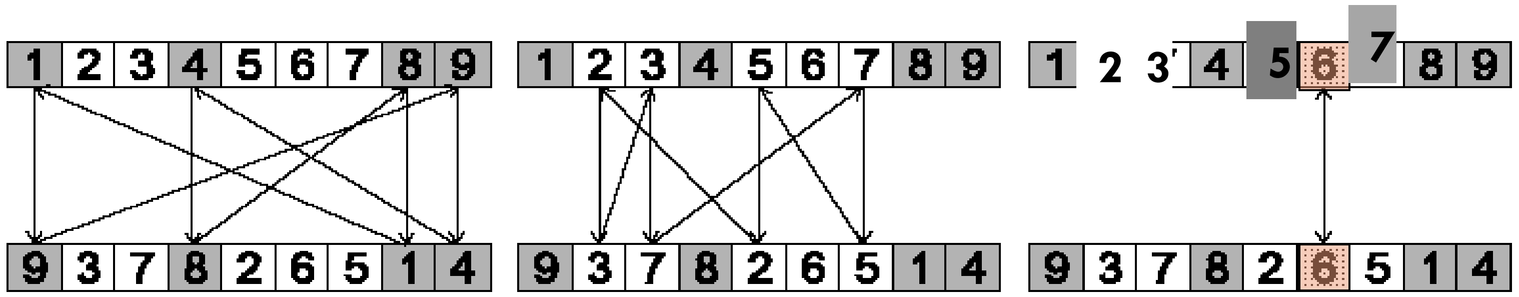
Each allele comes from one parent together with its position.

Informal procedure:

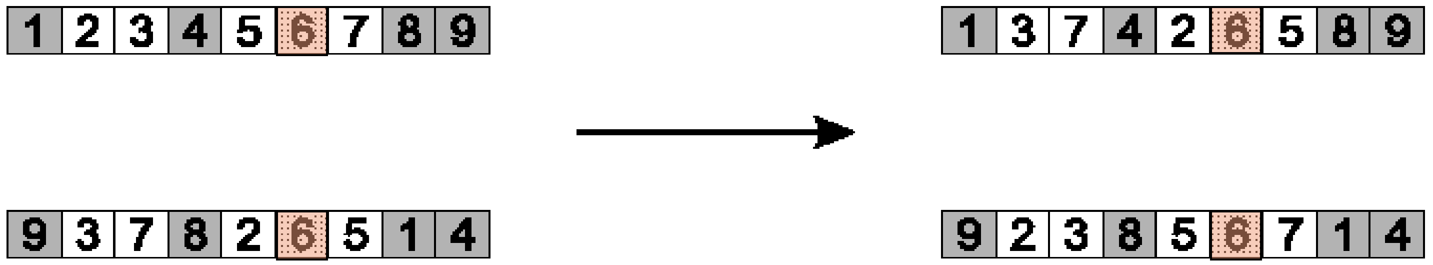
1. Make a cycle of alleles from P1 in the following way.
   (a) Start with the first allele of P1.
   (b) Look at the allele at the same position in P2.
   (c) Go to the position with the same allele in P1.
   (d) Add this allele to the cycle.
   (e) Repeat step b through d until you arrive at the first allele of P1.
2. Put the alleles of the cycle in the first child on the positions they have in the first parent.
3. Take next cycle from second parent

CYCLE CROSSOVER EXAMPLE

Step 1: identify cycles



Step 2: copy alternate cycles into offspring



PERMUTATIONS GAS – TSP EXAMPLE

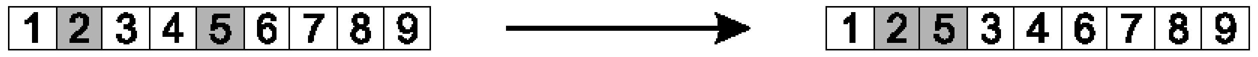
Four types of mutation operators:

- Insert mutation,
- Swap mutation,
- Inversion mutation,
- Scramble mutation.

PERMUTATIONS GAS – TSP EXAMPLE

Insert mutation:

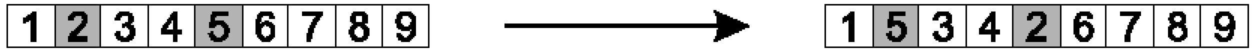
- Pick two genes values at random,
- Move the second to follow the first, shifting the rest along to accommodate
- Preserves most of the order and the adjacency information,



PERMUTATIONS GAS – TSP EXAMPLE

Swap mutation:

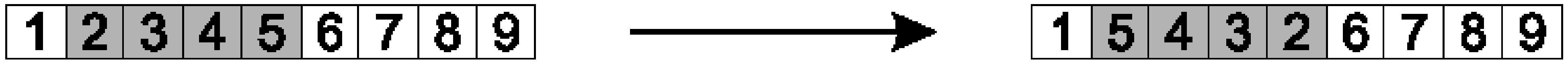
- Pick two genes at random and swap their positions,
- Preserves most of adjacency information (4 links broken), disrupts order more.



PERMUTATIONS GAS – TSP EXAMPLE

Inversion mutation:

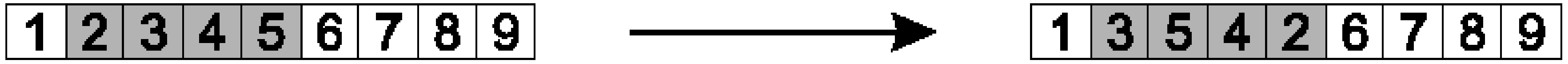
- Pick two genes at random and then invert the substring between them,
- Preserves most adjacency information (only breaks two links) but disruptive of order information.



PERMUTATIONS GAS – TSP EXAMPLE

Scramble mutation:

- Pick gene values at random,
- Randomly rearrange the genes in those positions (note subset does not have to be contiguous)



PERMUTATIONS GAS – TSP EXAMPLE

The GA was applied for the berlin52 TSP instance (52 locations in Berlin) by having:

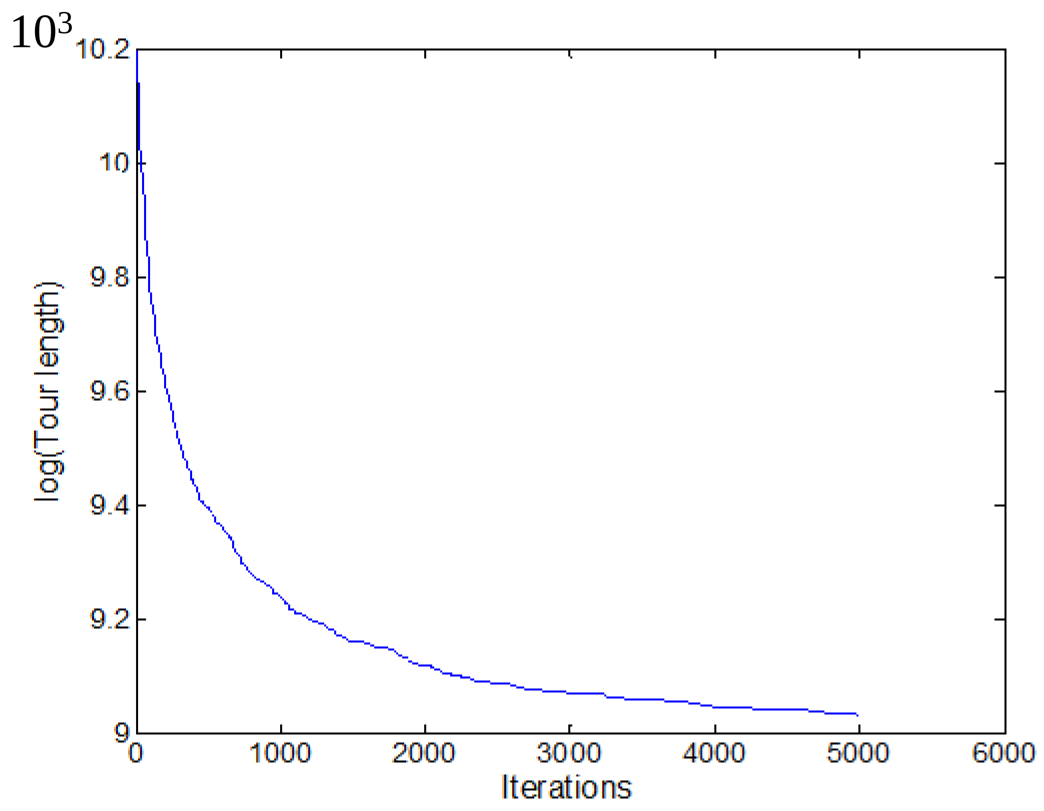
- 20 individuals
- Using order 1 crossover producing only one child
- Crossover probability = 0.7
- Using swap mutation with one swap only
- Mutation probability = 0.05
- Using the *elitism* model while keeping only the best individual from one population to the next,
- Using different number of iterations (500, 1000, 2000 and 5000),

PERMUTATIONS GAS – TSP EXAMPLE

Number of Iterations	Tour length
500	12135
1000	10069
2000	8986.2
5000	**8730.3**

SA obtained a tour of **8861**

PERMUTATIONS GAS – TSP EXAMPLE



GAS - APPLICATIONS

Antenna design

Electronic circuits

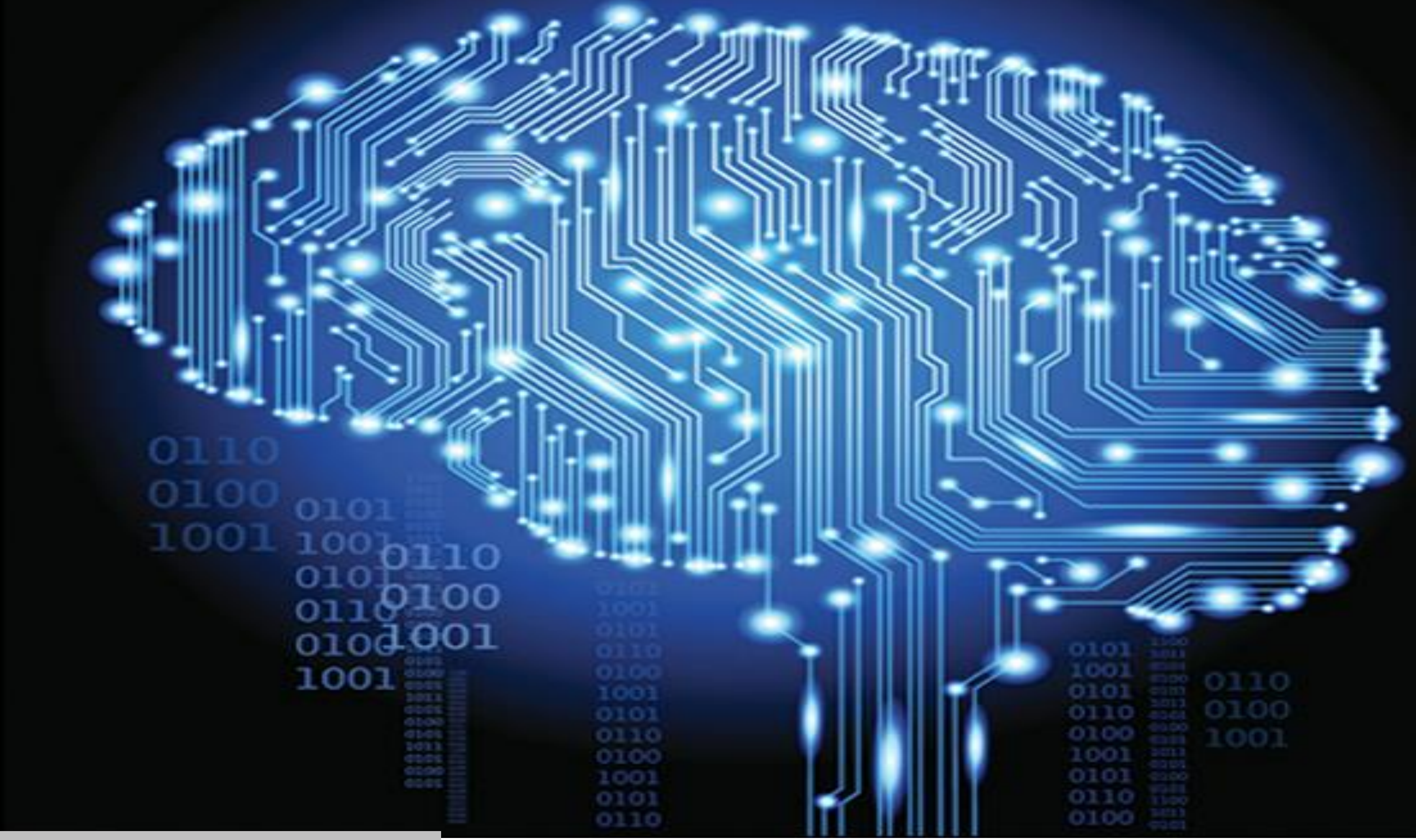
Network design

Protein folding

VLSI partitioning/routing/placement

Data mining

And many more …



2-VARIABLE FUNCTION EXAMPLE

APPLICATIONS

2-VARIABLE FUNCTION

- Consider maximizing the following function:

$$f(x,y) = x - y,$$
$$3 \le x \le 5.5 \text{ and } 0 \le y \le 1$$

We select a binary representation for each variable with enough bits to represent the decimal places with desired precision required.

For an interval L and n bit representation, the precision P is :

$$P = L/(2^n - 1)$$

Then

$$n = \lceil \log_2(L/P + 1) \rceil$$

2-VARIABLE FUNCTION

For a precision of 1 decimal place:

- *x* requires 5 bits and *y* requires 4 bits
- So the chromosome will be concatenation of the two, i.e, a 9 bit long.
- As example, for x=4.1, y=0.7 then we get

01011 0111

  - We subtract from x the lower value of the range 4.1-3=1.1 and for y becomes 0.7. then we convert to binary

2-VARIABLE FUNCTION

- String 1 String 2

  01011 0111

This becomes the chromosome of length 9 bits that we will use for GA.

To apply GA

- Generate a population of a number of chromosomes (say 4). They have to be ***feasible***
  - *i.e* in the ranges of the variables
- Evaluate the fitness of each

2-VARIABLE FUNCTION-INITIAL GENERATION

Suppose the four values are

- For x: 3.5, 4.1, 4.7, 5
- For y: 0.1, 0.3, 0.5, 0.7

Generating	f	f/f'	Actual Count
00101 0001	3.4	0.86	1
01011 0011	3.8	0.97	1
10001 0101	4.2	1.07	1
10100 0111	4.3	1.10	1
Sum	15.7		
Average	3.925		

So they all stay in the parent selection stage

2-VARIABLE FUNCTION - CROSSOVER

- Assuming that probability of crossover is high, we apply 1 point crossover.

Parents	Offspring	x	y	f
00101 0001	00101 0011	3.5	0.3	3.2
01011 0011	01011 0001	4.1	0.1	4.0
10001 0101	10000 0111	4.6	0.7	3.9
10100 0111	10101 0101	5.1	0.5	4.6

2-VARIABLE FUNCTION - MUTATION

- Suppose that mutation probability allows mutation of the 3rd offspring

Offspring	Mutation	x	y	f
001010011		3.5	0.3	3.2
010110001		4.1	0.1	4.0
1**0**0000111	110000111	5.4	0.7	**4.7**
101010101		5.1	0.5	4.6

Then you start from this generation and repeat the process. Notice that solution is progressing

ECE 457A ADAPTIVE AND COOPERATIVE ALGORITHMS

Curve Fitting Example

CURVE FITTING FORMULATION FOR GA

In curve fitting, we have a number of data points (x,y) and we would like to have a mathematical function or a curve that best fits the data.

Need to specify the function or the model in terms of parameters.

Use the data points and a fitting criteria to determine the parameters.

Most common models are line and polynomials with different degrees.

Some more complex models and no-model methods also exist.

CURVE FITTING

Assume we have ***n*** data points of the form (x,y) and our selected model is ***polynomial***

If we want the polynomial to fit all the n points, we select a polynomial of degree ***n-1*** (which will have n coefficients).

$$y = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0$$

If we substitute the data points in this formula, we get n equations in n unknowns.

If the points are distinct, we get a unique solution that determines the n coefficients.

CURVE FITTING

In many cases, we have a ***large*** number of points and using ***high degree polynomials*** will not provide a good fit as they oscillate between the points.

We can use a polynomial of certain degree and get the best fit to the given data.

Best fit can be defined using ***fitting criteria***, most common is ***least squares***.

When we substitute the n data points into the fitting polynomial (***which is of degree k much less than n-1***), we get a ***over-determined system*** of equations which we can solve to minimize the least squares of the errors (actual values – calculated values of y)

CURVE FITTING

There are algebraic methods to solve the least squares problem

Suppose that we want to solve the problem using genetic algorithms.

We can start by initial values for the coefficients $a_{k,} a_{k-1}, \cdots, a_0$, substitute the n x data points in the formula and calculate the y' and then calculate the

$$\sum_n (y - y')^2$$

CURVE FITTING USING GA

- Define chromosome as the vector of coefficients

$$a_{k,} a_{k-1}, \cdots, a_0$$

The fitness is $\sum_n (y - y')^2$ => smaller is better (minimization problem)

- Start with a population of randomly generated initial coefficients vectors as parents
- Evaluate them using the fitness function
- Select parents based fitness
- Apply crossover and mutation to generate offspring

- Repeat till stopping condition is satisfied.

ECE 457A ADAPTIVE AND COOPERATIVE ALGORITHMS

Why are they useful?

GAS – WHEN ARE THEY USEFUL?

Highly multimodal functions

Discrete or discontinuous functions

High-dimensionality functions, including many combinatorial ones

Non-linear dependencies between parameters

DIFFICULTIES THAT MAY OCCUR WITH GA

Premature convergence

Unable to overcome deception

Need more evaluations than time permits

Bad match of representation/mutation/crossover, making operators destructive

Biased or incomplete representation

Problem too hard

◎ (Problem too easy, makes GA *look* bad)