

University of Babylon

College of Information Technology



Software Department

Artificial Intelligence Application (Genetic Algorithms)

By

Ass. Prof. Dr. Asaad Sabah Hadi

2016-2017

EVOLUTIONARY ALGORITHM

- The main idea about for Evolutionary Algorithm is : Given a population of individuals then using the idea of “survival of the fittest” to make natural selection and this causes arise in the fitness of the population.
- Given the function to be maximized, we can randomly create a set of candidate solutions, i.e. , elements of the function’s domain , and apply the quality function and the higher is the better.
- Based on this fitness some of the better candidates are chosen to seen in the next generation by applying recombination and/or mutation to them.
- Recombination is an operator applied to two or more selected candidates(parents) and results one or more new candidates(Children).
- Mutation is applied to one candidate and results in one new candidate.
- Executing recombination and mutation leads to a set of new candidates (the offspring) that compete –based on their fitness with the old ones for a place in the next generation.
- This process can be iterated until a candidate with a sufficient quality (a solution) is found or a previously set computational limit is reached.

EVOLUTIONARY ALGORITHM

BEGIN

INITIALISE population with random candidate solutions;

EVALUATE each candidate;

REPEAT UNTIL (*TERMINATION CONDITION* is satisfied) DO

1 *SELECT* parents;

2 *RECOMBINE* pairs of parents;

3 *MUTATE* the resulting offspring;

4 *EVALUATE* new candidates;

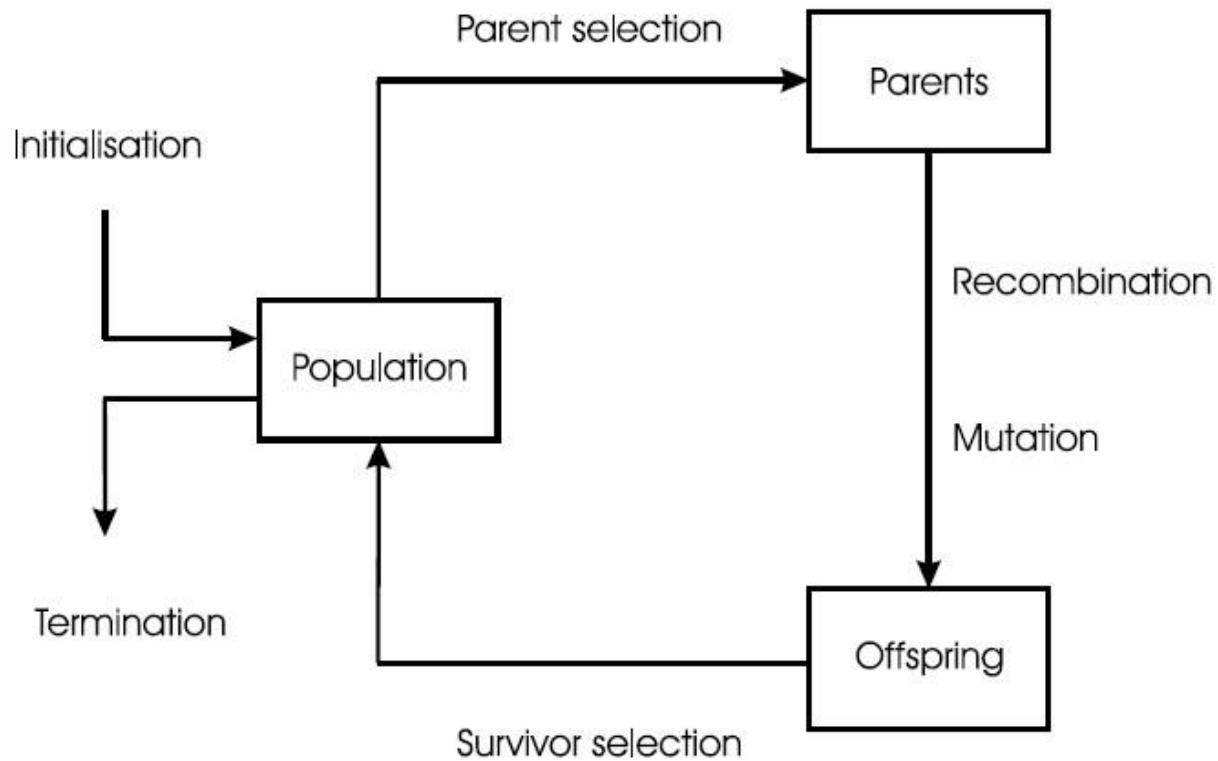
5 *SELECT* individuals for the next generation;

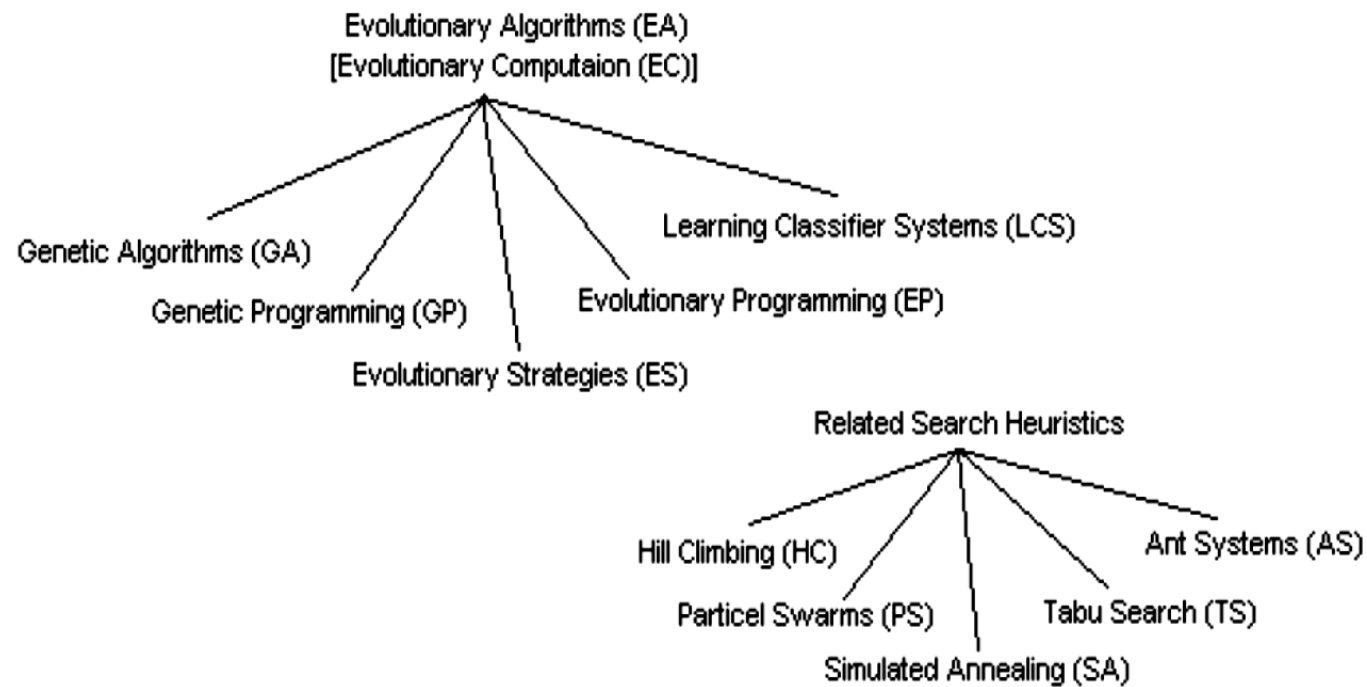
OD

END

- It is easy to see that this algorithm falls in the category of generate-and-test algorithm.
- The evaluation (fitness) function represent a heuristic estimation of solution operators.
- Evolutionary Algorithm posses a number of features :
 1. EAs are population based, i.e., they process a whole collection of candidate solutions simultaneously.
 2. EAs mostly use recombination to mix information of more candidate solutions into a new one.
 3. EAs are Stochastic.

EA FLOWCHART



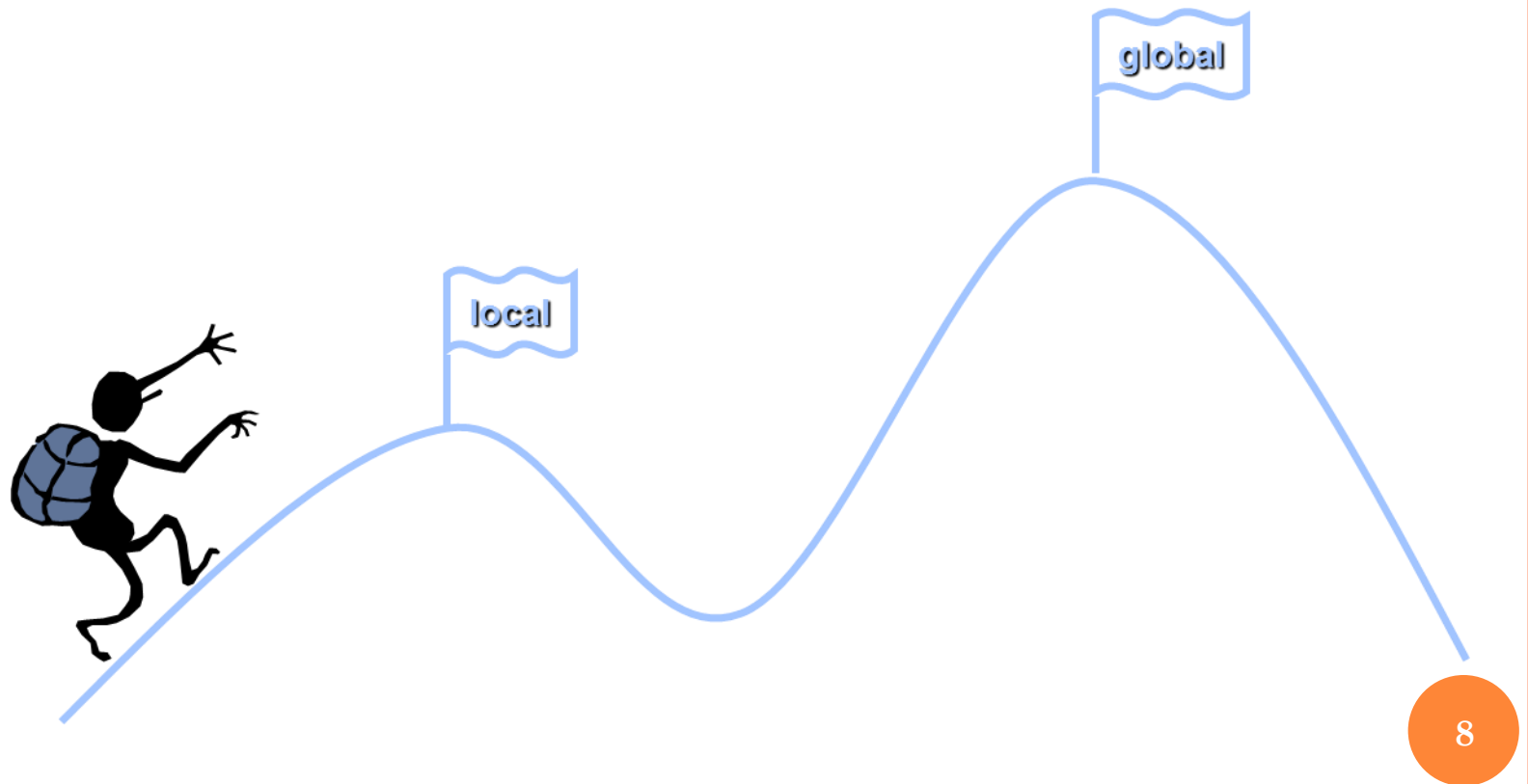


PART I: GA THEORY

- What are genetic algorithms?
- How to design a genetic algorithm?
- Genetic algorithms are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply recombination operators to these structures in order to preserve critical information.
- Genetic Algorithm often viewed as function optimizers.

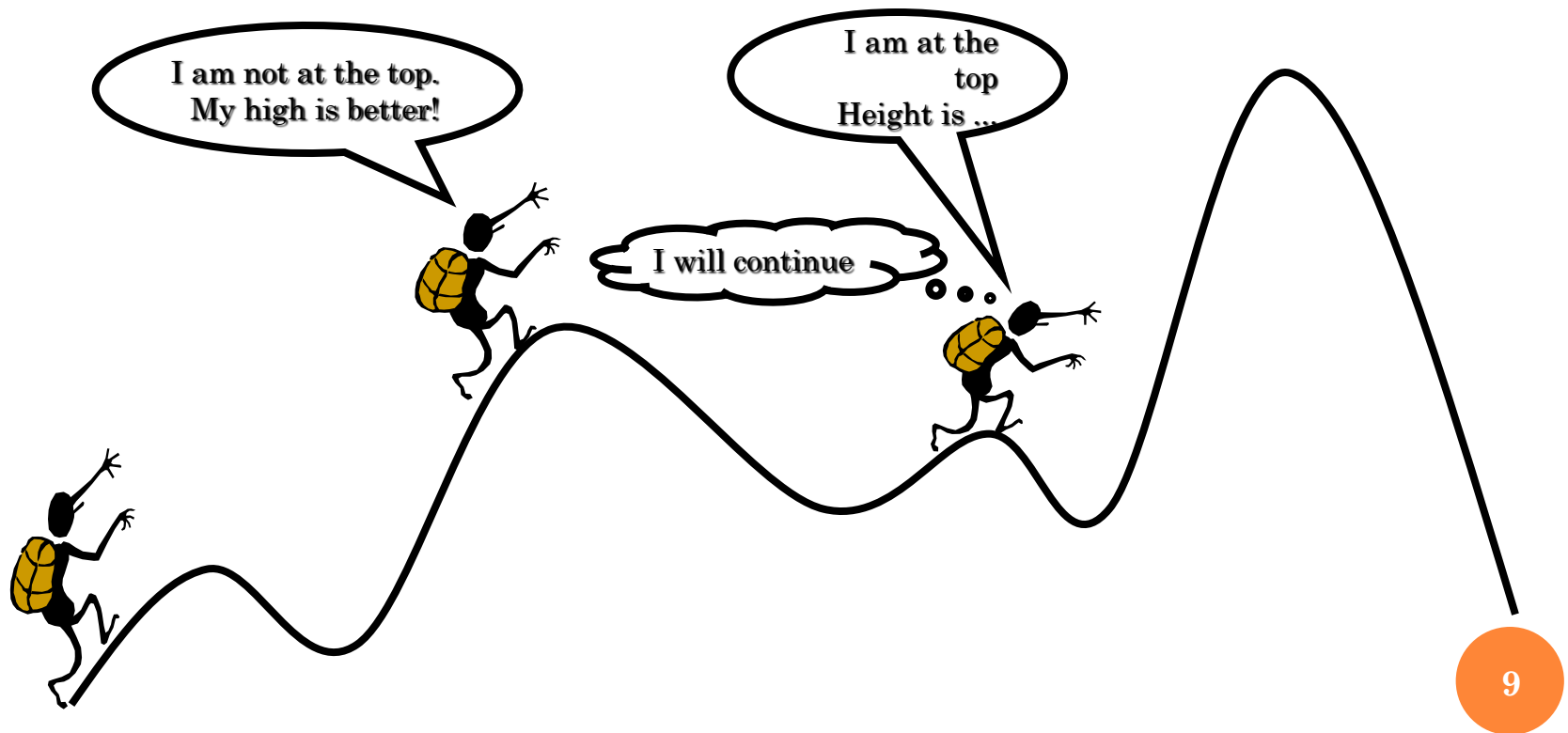
INSTEAD OF INTRODUCTION...

- Hill climbing

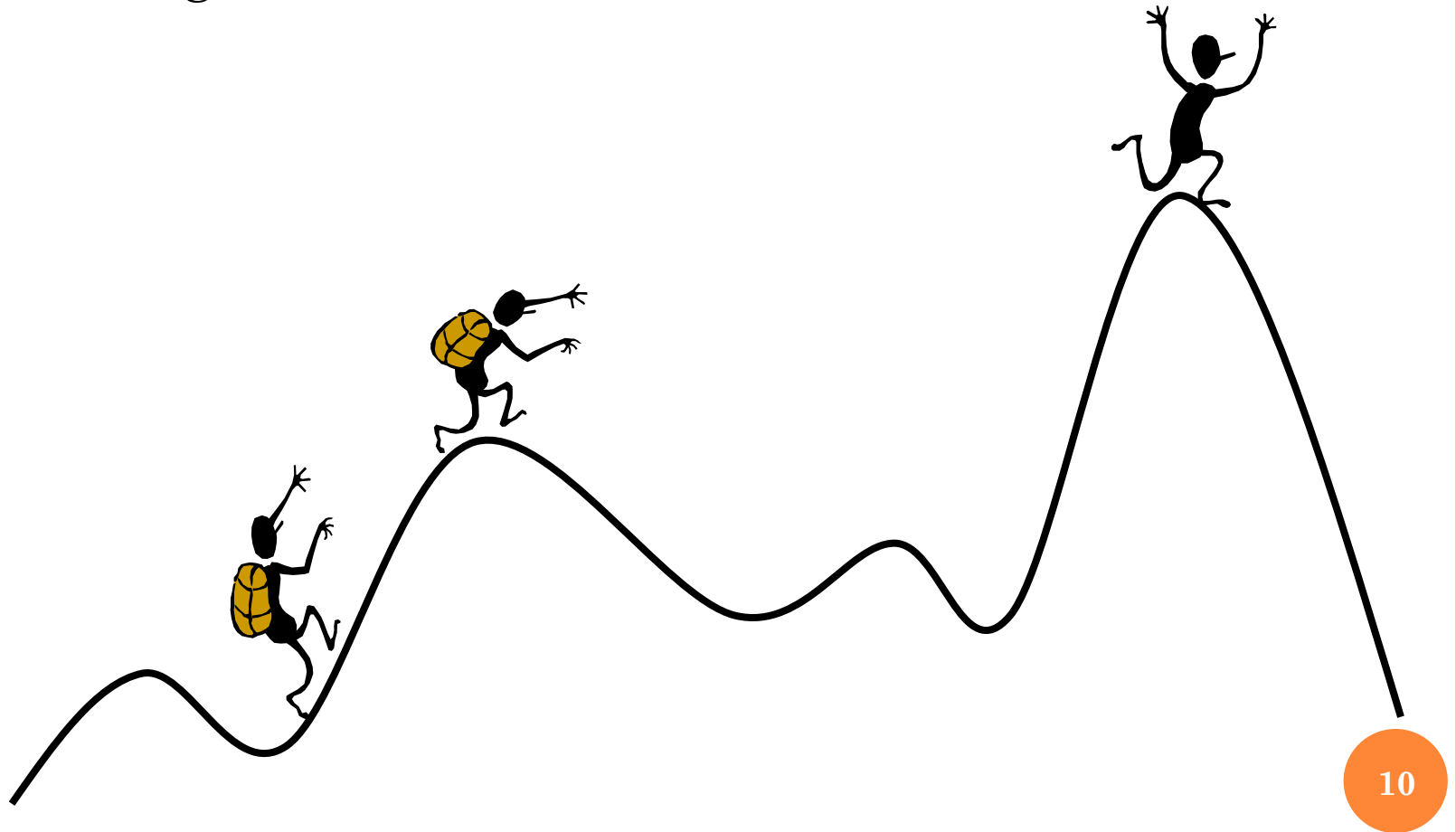


INSTEAD OF INTRODUCTION...(3)

- Genetic algorithm



- Genetic algorithm - few microseconds after



GA CONCEPT

- 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 of a population of individuals represented by chromosomes, in essence a set of character strings, that are analogous to the DNA, that we have in our own chromosomes.

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.



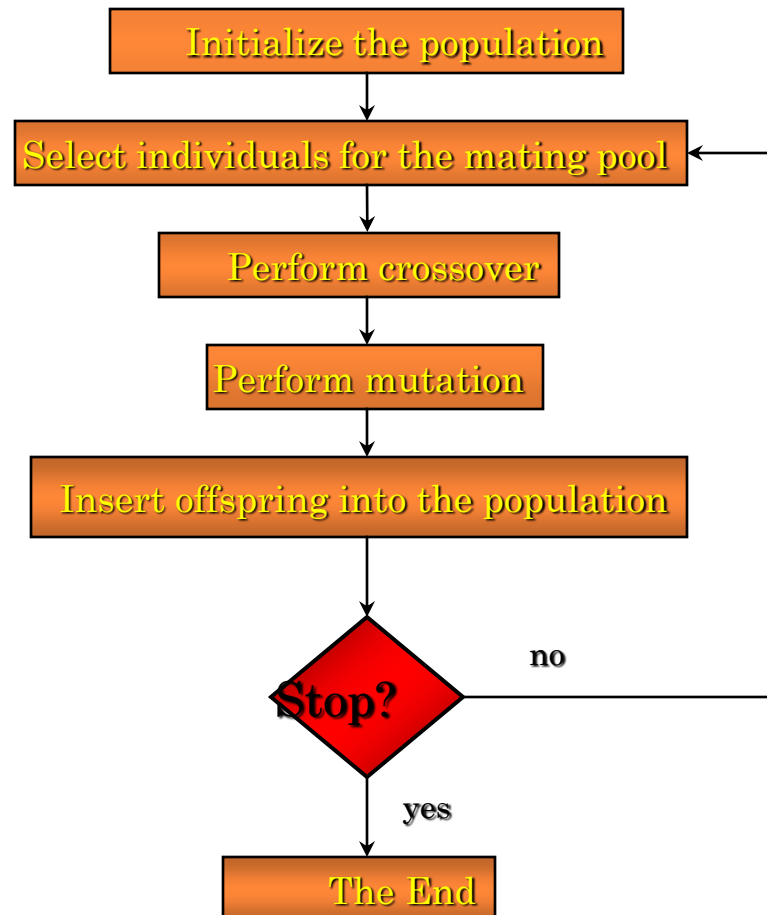
NATURE AND GA...

<i>Nature</i>		<i>Genetic algorithm</i>	
Chromosome	→	String	
Gene	→	Character	
Locus	→	String position	
Genotype	→	Population	
Phenotype	→	Decoded structure	

THE HISTORY OF GA

- Cellular automata
 - John Holland, university of Michigan, 1975.
- Until the early 80s, the concept was studied theoretically.
- In 80s, the first “real world” GAs were designed.

ALGORITHMIC PHASES



SGA_pseudo-code

- {
- Choose an initial population of individuals: $P(0)$;
- Evaluate the fitness of all individuals of $P(0)$;
- Choose a maximum number of generations: t_{\max} ;
- While (not satisfied and $t < t_{\max}$) do {
 - $t \leftarrow t+1$;
 - Select parents for offspring production;
 - Apply reproduction and mutation operators;
 - Create a new population of survivors: $P(t)$;
 - Evaluate $P(t)$;
- }
- Return the best individual of $P(t)$;
- }

DESIGNING GA...

- How to represent genomes?
- How to define the crossover operator?
- How to define the mutation operator?
- How to define fitness function?
- How to generate next generation?
- How to define stopping criteria?

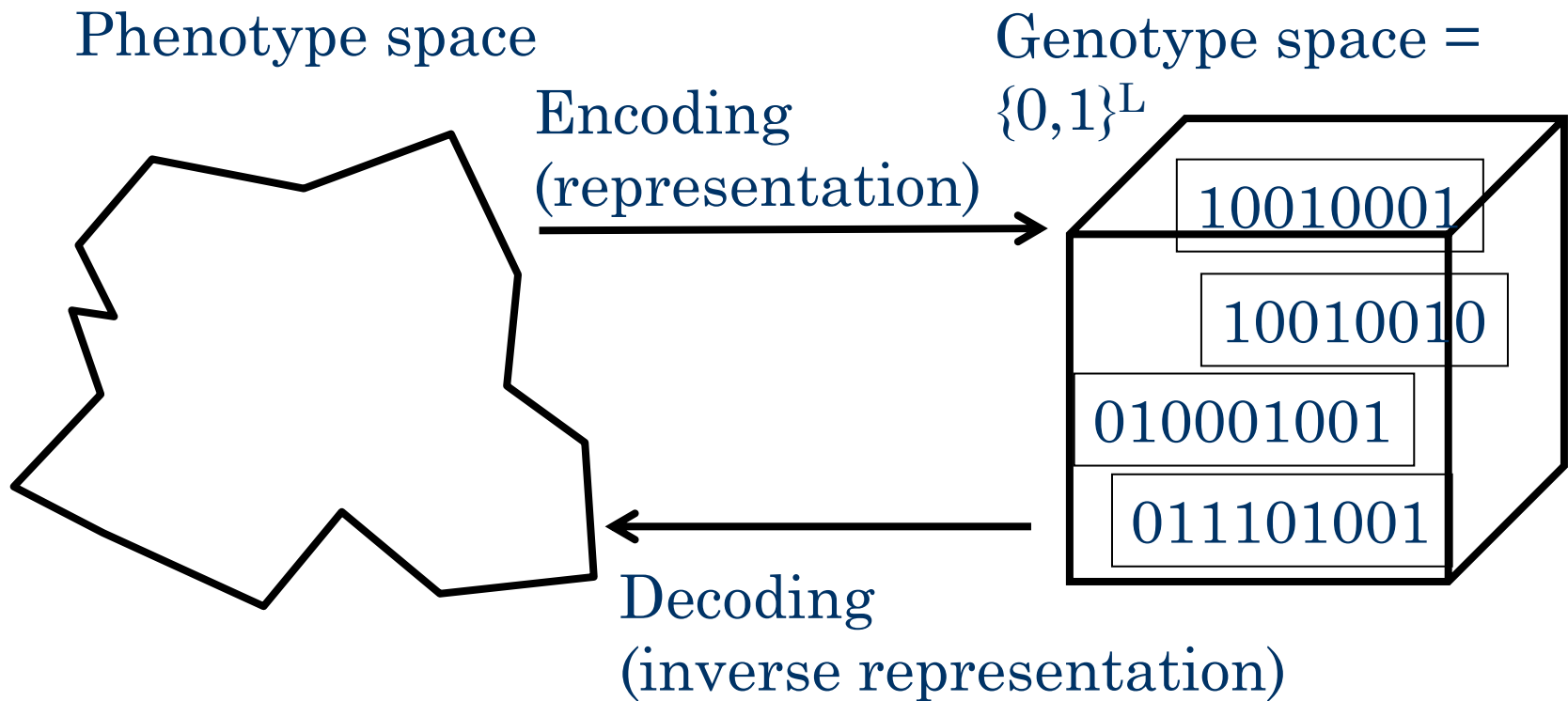
REPRESENTING GENOMES...

<i>Representation</i>	<i>Example</i>							
string	<table border="1"><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table> 1	1	0	1	1	1	0	0
1	0	1	1	1	0	0		
array of strings	<table border="1"><tr><td>http</td><td>avala</td><td>yubc</td><td>net</td><td>~apopovic</td></tr></table>	http	avala	yubc	net	~apopovic		
http	avala	yubc	net	~apopovic				
tree - genetic programming	<pre>graph TD; or((or)) --- gt((>)); or --- c((c)); gt --- xor((xor)); gt --- b1((b)); xor --- a((a)); xor --- b2((b));</pre>							

CROSSOVER

- Crossover is concept from genetics.
- Crossover combines genetic material from two parents,
in order to produce superior offspring.
- Few types of crossover:
 - One-point
 - Multiple point.

REPRESENTATION

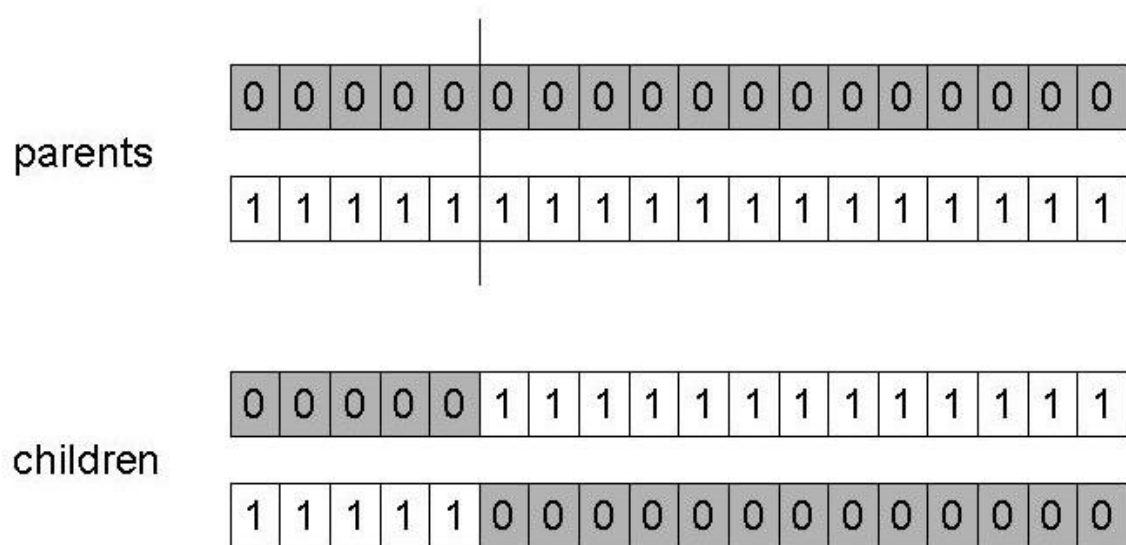


SGA REPRODUCTION CYCLE

1. Select parents for the mating pool
(size of mating pool = population size)
2. Shuffle the mating pool
3. For each consecutive pair apply crossover with probability p_c , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with the resulting offspring

SGA OPERATORS: 1-POINT Crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails
- P_c typically in range (0.6, 0.9)



SGA OPERATORS: MUTATION

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
 - Typically between $1/\text{pop_size}$ and $1/\text{chromosome_length}$

parent

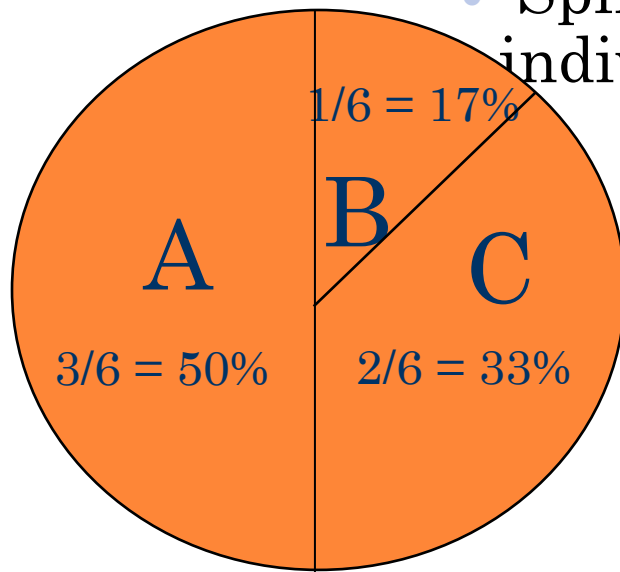
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

child

0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

SGA OPERATORS: 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



$\text{fitness}(\text{A}) = 3$

$\text{fitness}(\text{B}) = 1$

$\text{fitness}(\text{C}) = 2$

AN EXAMPLE

- Simple problem: $\max x^2$ over $\{0,1,\dots,31\}$
- GA approach:
 - Representation: binary code, e.g. $01101 \leftrightarrow 13$
 - Population size: 4
 - 1-point xover, bitwise mutation
 - Roulette wheel selection
 - Random initialisation
- We show one generational cycle done by hand

X² EXAMPLE: 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

X² EXAMPLE: CROSSOVER

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

X² EXAMPLE: MUTATION

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

THE SIMPLE GA

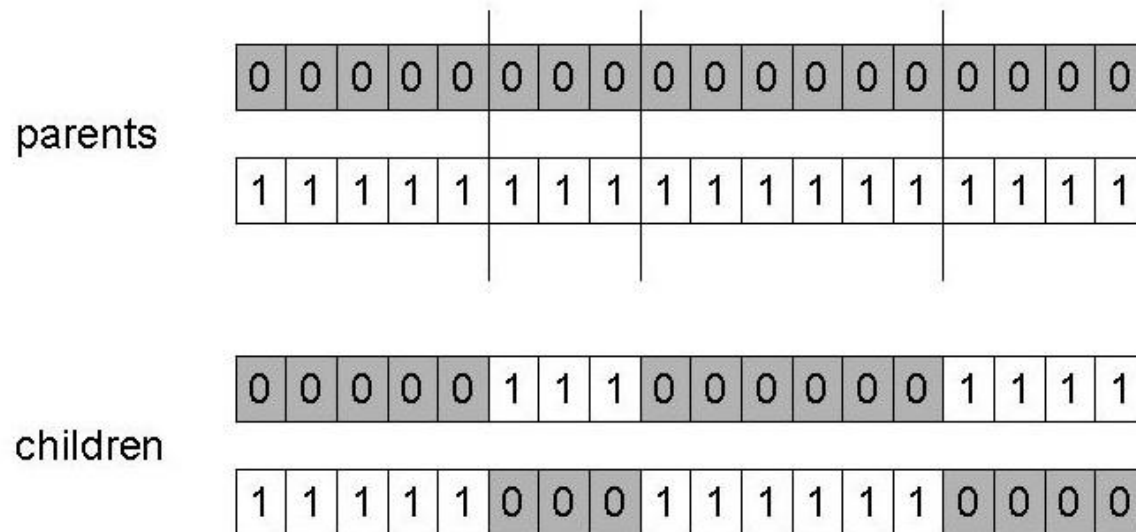
- Has been subject of many (early) studies
 - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
 - Representation is too restrictive
 - Mutation & crossovers only applicable for bit-string & integer representations
 - Selection mechanism sensitive for converging populations with close fitness values
 - Generational population model (step 5 in SGA repr. cycle) can be improved with explicit survivor selection

ALTERNATIVE CROSSOVER OPERATORS

- Performance with 1 Point Crossover depends on the order that variables occur in the representation
 - more likely to keep together genes that are near each other
 - Can never keep together genes from opposite ends of string
 - This is known as *Positional Bias*
 - Can be exploited if we know about the structure of our problem, but this is not usually the case

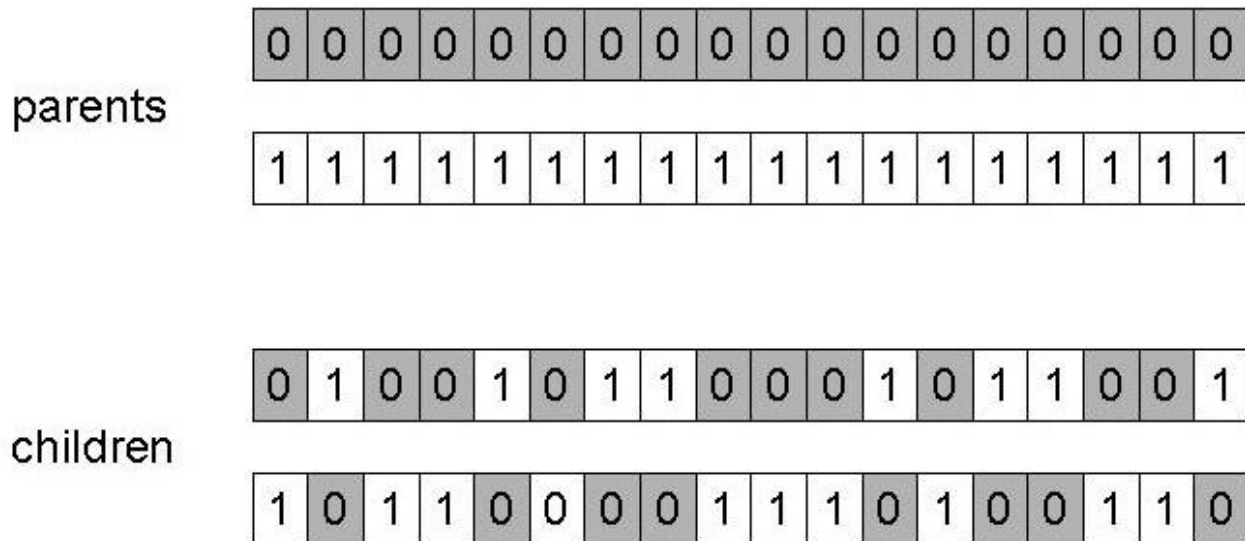
N-POINT CROSSOVER

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



UNIFORM CROSSOVER

- 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



CROSSOVER OR MUTATION?

- Decade long debate: which one is better / necessary / main-background
- Answer (at least, rather wide agreement):
 - it depends on the problem, but
 - in general, it is good to have both
 - both have another role
 - mutation-only-EA is possible, crossover-only-EA would not work

CROSSOVER OR MUTATION? (CONT'D)

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

There is co-operation AND competition between them

- Crossover is explorative, it makes a *big* jump to an area somewhere “in between” two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent

CROSSOVER OR MUTATION? (CONT'D)

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, ?% after performing n crossovers)
- To hit the optimum you often need a 'lucky' mutation

OTHER REPRESENTATIONS

- Gray coding of integers (still binary chromosomes)
 - 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

Nowadays it is generally accepted that it is better to encode numerical variables directly as

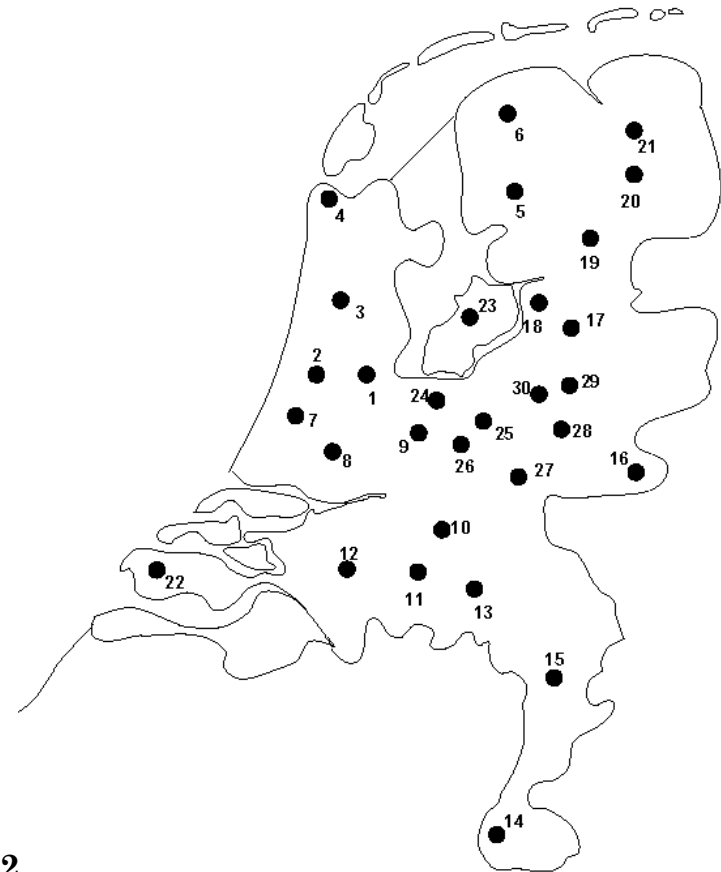
- Integers
- Floating point variables

INTEGER REPRESENTATIONS

- Some problems naturally have integer variables, e.g. image processing parameters
- Others take *categorical* values from a fixed set e.g. {blue, green, yellow, pink}
- N-point / uniform crossover operators work
- Extend bit-flipping mutation to make
 - “creep” i.e. more likely to move to similar value
 - Random choice (esp. categorical variables)
 - For ordinal problems, it is hard to know correct range for creep, so often use two mutation operators in tandem

PERMUTATION REPRESENTATION: TSP EXAMPLE

- Problem:
 - Given n cities
 - Find a complete tour with minimal length
- Encoding:
 - Label the cities $1, 2, \dots, n$
 - One complete tour is one permutation (e.g. for $n=4$ $[1,2,3,4]$, $[3,4,2,1]$ are OK)
- Search space is BIG:
for 30 cities there are $30! \approx 10^{32}$ possible tours

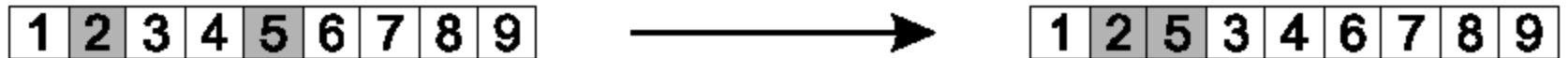


MUTATION OPERATORS FOR PERMUTATIONS

- Normal mutation operators lead to inadmissible solutions
 - e.g. bit-wise mutation : let gene i have value j
 - changing to some other value k would mean that k occurred twice and j no longer occurred
- Therefore must change at least two values
- Mutation parameter now reflects the probability that some operator is applied once to the whole string, rather than individually in each position

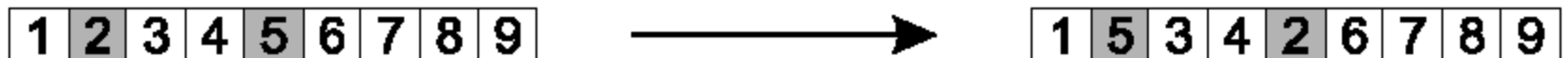
INSERT MUTATION FOR PERMUTATIONS

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate
- Note that this preserves most of the order and the adjacency information



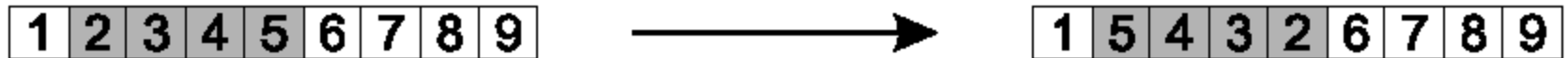
SWAP MUTATION FOR PERMUTATIONS

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



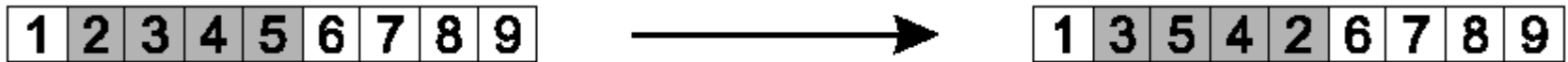
INVERSION MUTATION FOR PERMUTATIONS

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



SCRAMBLE MUTATION FOR PERMUTATIONS

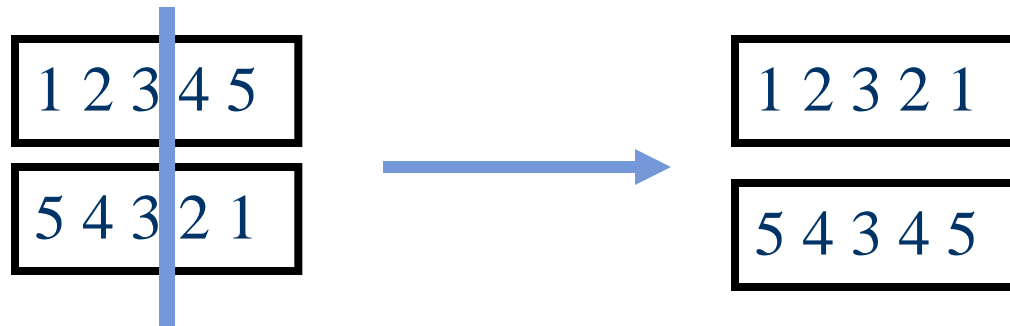
- Pick a subset of genes at random
- Randomly rearrange the alleles in those positions



(note subset does not have to be contiguous)

CROSSOVER OPERATORS FOR PERMUTATIONS

- “Normal” crossover operators will often lead to inadmissible solutions



- Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

ORDER 1 CROSSOVER

- 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:
 - starting right from cut point of the copied part,
 - using the **order** of the second parent
 - and wrapping around at the end
 4. Analogous for the second child, with parent roles reversed

ORDER 1 CROSSOVER EXAMPLE

- Copy randomly selected set from first parent
- Copy rest from second parent in order 1,9,3,8,2

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



			4	5	6	7		
--	--	--	---	---	---	---	--	--

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



3	8	2	4	5	6	7	1	9
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

PARTIALLY MAPPED CROSSOVER (PMX)

Informal procedure for parents P1 and P2:

1. Choose random segment 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 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

PMX EXAMPLE

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



			4	5	6	7		
--	--	--	---	---	---	---	--	--

Step 1

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



		2	4	5	6	7		8
--	--	---	---	---	---	---	--	---

Step 2

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



9	3	2	4	5	6	7	1	8
---	---	---	---	---	---	---	---	---

Step 3

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

CYCLE CROSSOVER

Basic idea:

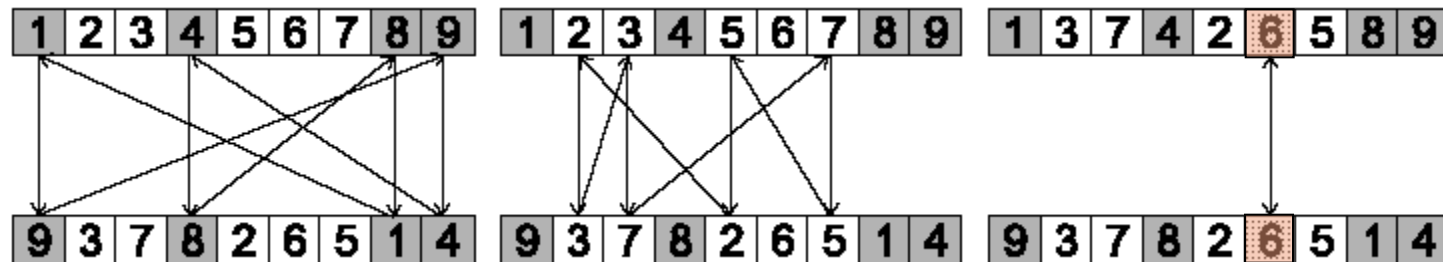
Each allele comes from one parent *together with its position*.

Informal procedure:

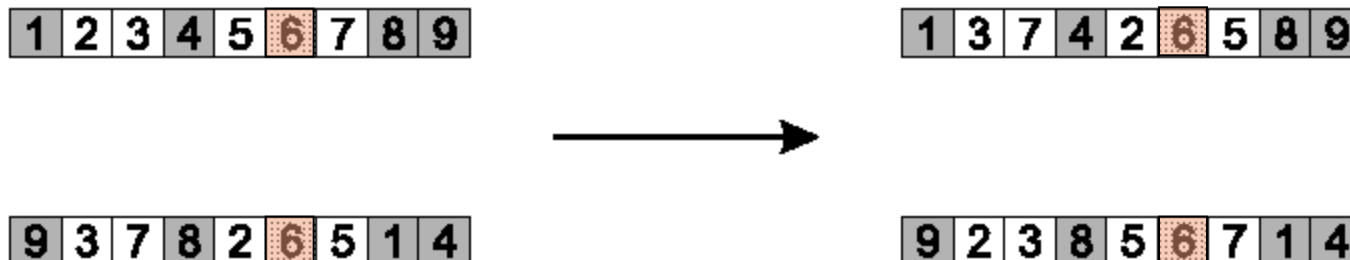
1. Make a cycle of alleles from P1 in the following way.
 - (a) Start with the first allele of P1.
 - (b) Look at the allele at the *same position* in P2.
 - (c) Go to the position with the *same allele* in P1.
 - (d) Add this allele to the cycle.
 - (e) Repeat step b through d until you arrive at the 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



MUTATION

- Mutation introduces randomness into the population.
- Mutation is asexual reproduction.
- The idea of mutation is to reintroduce divergence into a converging population.
- Mutation is performed on small part of population, in order to avoid entering unstable state.

ABOUT PROBABILITIES...



- Average probability for individual to crossover is, in most cases, about 80%.
- Average probability for individual to mutate is about 1-2%.
- Probability of genetic operators follow the probability in natural systems.
- The better solutions reproduce more often.

FITNESS FUNCTION

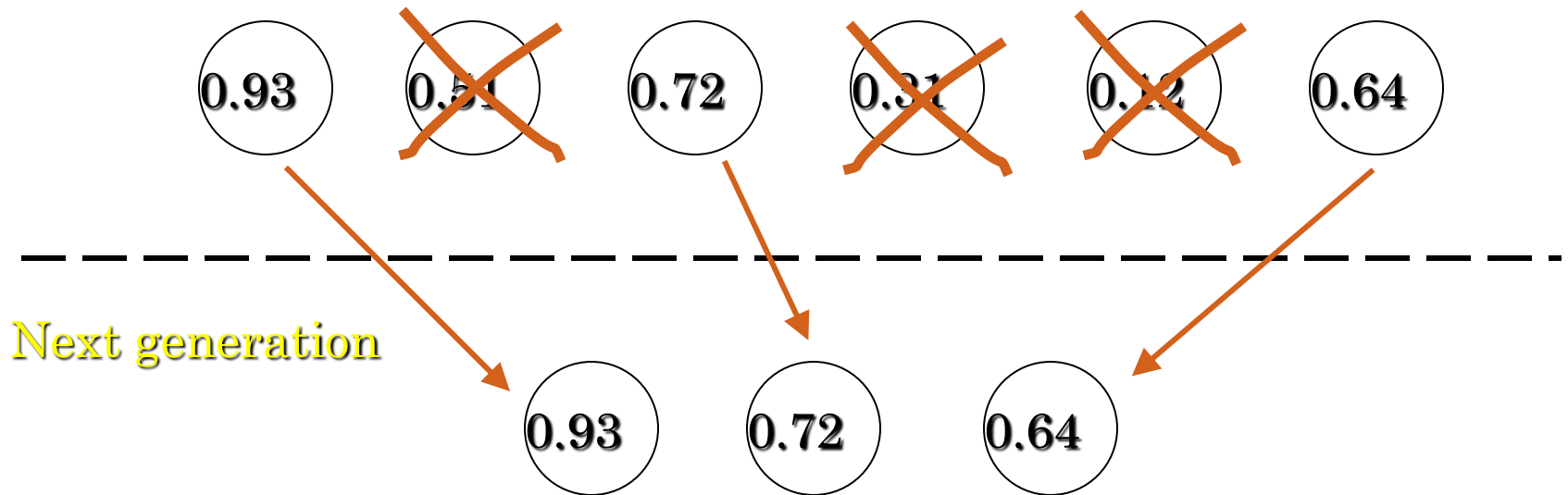
- Fitness function is evaluation function, that determines what solutions are better than others.
- Fitness is computed for each individual.
- Fitness function is application depended.

SELECTION

- The selection operation copies a single individual, probabilistically selected based on fitness, into the next generation of the population.
- There are few possible ways to implement selection:
 - “Only the strongest survive”
 - Choose the individuals with the highest fitness for next generation
 - “Some weak solutions survive”
 - Assign a probability that a particular individual will be selected for the next generation
 - More diversity
 - Some bad solutions might have good parts!

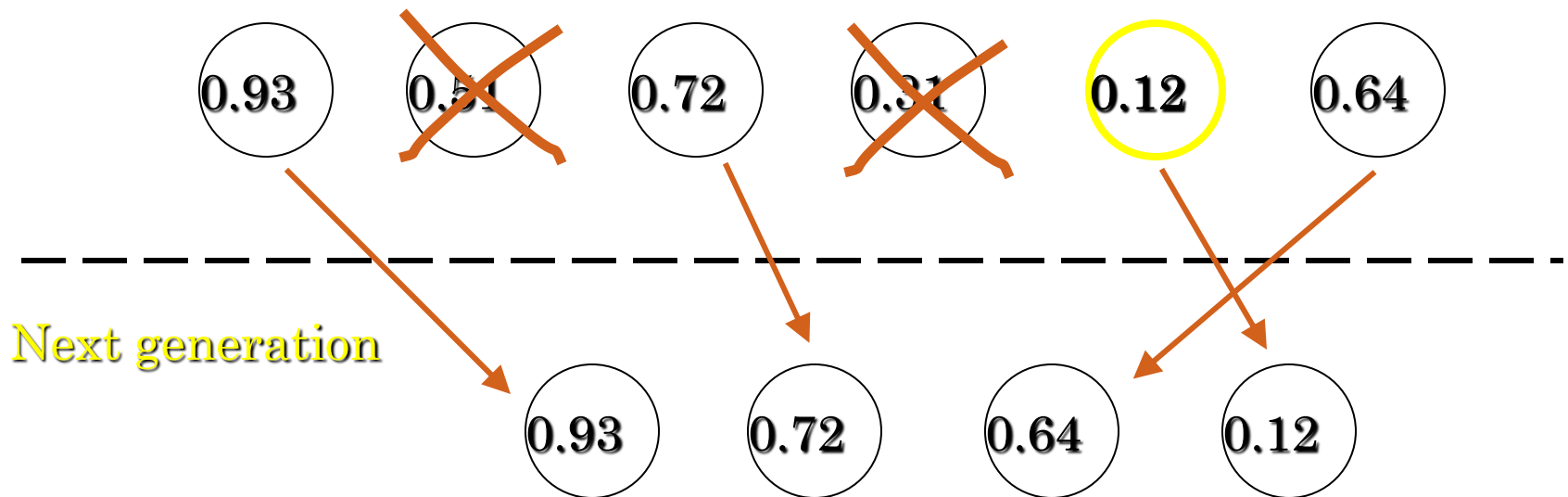
SELECTION - SURVIVAL OF THE STRONGEST

Previous generation



SELECTION - SOME WEAK SOLUTIONS SURVIVE

Previous generation



OTHER TYPE OF SELECTION

❖ Rank Selection

- Rank selection is an alternative method where the individuals in the population are ranked according to fitness, and the expected value of each individual depends on its rank rather than on its absolute fitness.
- The linear ranking method proposed by Baker is as follows: Each individual in the population is ranked in increasing order of fitness, from 1 to N.
- The user chooses the expected value Max of the individual with rank N, with $Max \geq 0$. The expected value of each individual in the population at time t is given by:

$$ExpVal(i, t) = Min + (Max - Min) \frac{rank(i, t) - 1}{N - 1},$$

❖ Tournament Selection

- Tournament selection is similar to rank selection in terms of selection pressure, but it is computationally more efficient and more amenable to parallel implementation.

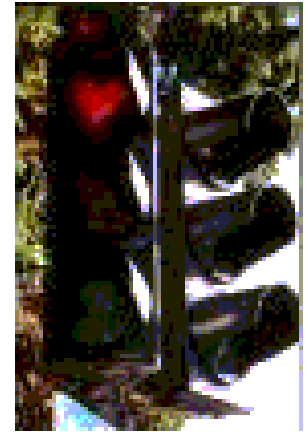
There are two types of tournament selection :

1- **Binary Tournament Selection**

In this method, Two individuals are chosen at random from the population. A random number r is then chosen between 0 and 1. If $r < k$ (where k is a parameter, for example 0.75), the fitter of the two individuals is selected to be a parent; otherwise the less fit individual is selected. The two are then returned to the original population and can be selected again .

2- **Triple Tournament Selection**





The main difference in this method is that we choose three individuals at random from the population and choose one of them.



STOPPING CRITERIA

- Final problem is to decide when to stop execution of algorithm.
- There are two possible solutions to this problem:
 - First approach:
 - Stop after production of definite number of generations
 - Second approach:
 - Stop when the improvement in average fitness over two generations is below a threshold

GA VS. AD-HOC ALGORITHMS

	Genetic Algorithm	Ad-hoc Algorithms
 Speed	Slow *	Generally fast
 Human work	Minimal	Long and exhaustive
 Applicability	General	There are problems that cannot be solved analytically
 Performance	Excellent	Depends

ADVANTAGES OF GAS

- Concept is easy to understand.
- Minimum human involvement.
- Computer is not learned how to use existing solution, but to find new solution!
- Modular, separate from application
- Supports multi-objective optimization
- Always an answer; answer gets better with time !!!
- Inherently parallel; easily distributed
- Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained
- Easy to exploit previous or alternate solutions

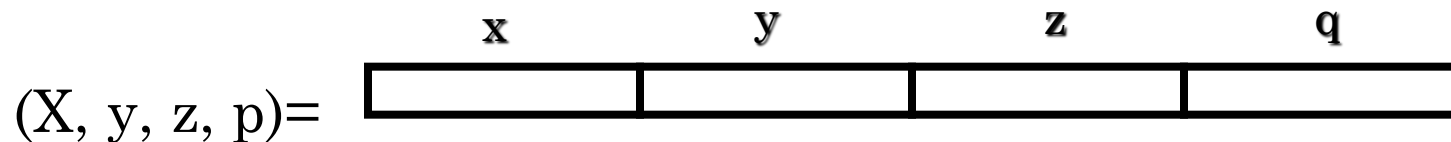
GA: AN EXAMPLE - DIOPHANTINE EQUATIONS

- Diophantine equation (n=4):

$$A*x + b*y + c*z + d*q = s$$

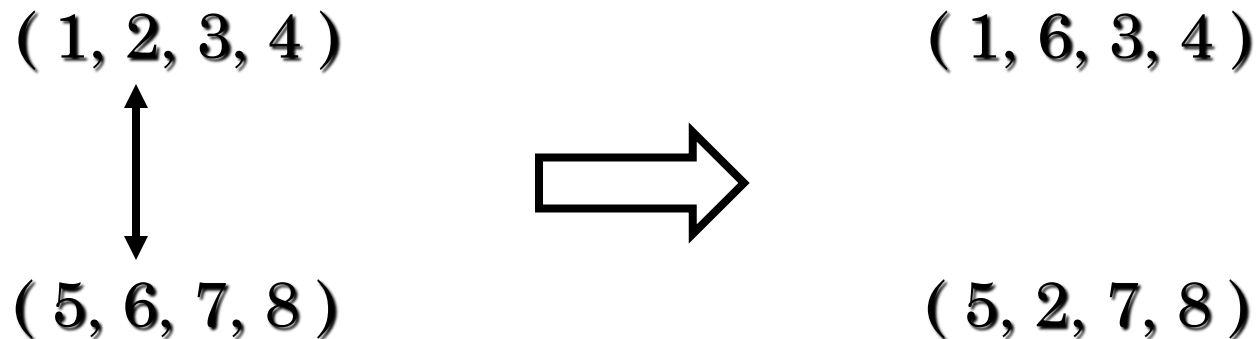
- For given a, b, c, d, and s - find x, y, z, q

- Genome:

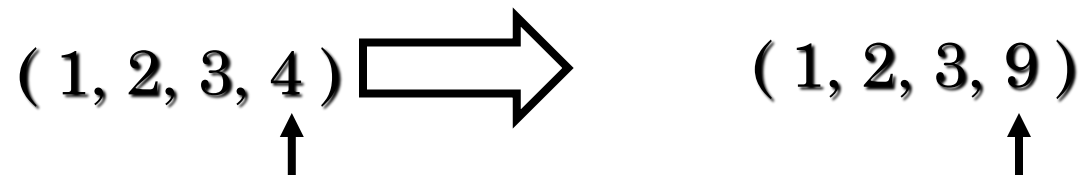


GA:AN EXAMPLE - DIOPHANTINE EQUATIONS(2)

- Crossover



- Mutation



GA:AN EXAMPLE - DIOPHANTINE EQUATIONS(3)

- First generation is randomly generated of numbers lower than sum (s).
- Fitness is defined as absolute value of difference between total and given sum:

$$\text{Fitness} = \text{abs} (\text{total} - \text{sum}) ,$$

- Algorithm enters a loop in which operators are performed on genomes: crossover, mutation, selection.
- After number of generation a solution is reached.



PART II: MATHEMATICS BEHIND GA-S

Two methods for analyzing genetics algorithms:

Schema analyses

Mathematical modeling

SCHEMA ANALYSES

Weaknesses:

- In determining some characteristics of the population
- Schema analyses makes some approximations that weaken it

Advantages:

- A simple way to view the standard GA
- They have made possible proofs of some interesting theorems
- They provide a nice introduction to algorithmic analyses

SCHEMA ANALYSES...

Schema – a template made up of a string of 1s, 0s, and *s, where * is used as a wild card that can be either 1 or 0

For example, $H = 1 ** 0 * 0$ is a schema.

- It has eight **instances** (one of which is 101010)
- **Order**, $o(H)$, the number of non-*, or defined, bits (in example 3)
- **Defining length**, $d(H)$, greatest distance between two defined bits (in example H has a defining length of 3)

Let S be the set of all strings of length l .

- There is 3^l possible schemas on S , but 2^{2^l} different subsets of S
- Schema cannot be used to represent every possible population within S , but forms a representative subset of the set of all subsets of S

SCHEMA ANALYSES...

The end-of-iterations conditions:

expected number of instances of schema H as we iterate the GA

- $M(H, t)$ – the number of instances of H at time t
- $f(x)$ – fitness of chromosome x
- $\bar{f}(t)$ - average fitness at time t :

$$\bar{f}(t) = \frac{\sum_{x \in S} f(x)}{n}, \quad n = |S|$$

- $\hat{u}(H, t)$ - average fitness of instances of H at time t :

$$\hat{u}(H, t) = \frac{\sum_{x \in H} f(x)}{m(H, t)}$$

SCHEMA ANALYSES...

- If we completely ignore the effects of crossover and mutation,
we get the expected value:

$$E(m(H, t + 1)) = n \frac{\sum_{x \in H} f(x)}{\sum_{x \in S} f(x)} = \frac{\sum_{x \in H} f(x)}{\bar{f}(t)} = \frac{\hat{u}(H, t) m(H, t)}{\bar{f}(t)}$$

- Now we consider only the effects of crossover and mutation, which lower the number of instances of H in the population.

Then we will get a good lower bound on $E(m(H, t + 1))$

- $s_c(H)$ - probability that a random crossover bit is between the defining bits of H
- p_c - probability of crossover occurring

$$s_c(H) = 1 - p_c \left(\frac{d(H)}{l - 1} \right)$$

SCHEMA ANALYSES...

- $S_m(H)$ - probability of an instance of H remaining the same after mutation; it is dependent on the order of H
- p_m - probability of mutation

$$S_m(H) = (1 - p_m)^{o(H)}$$

- With the above notation , we have:

Schema Theorem, provided by John Holland

$$E(m(H, t + 1)) \geq \frac{\hat{u}(H, t)m(H, t)}{\bar{f}(t)} (1 - p_c \frac{d(H)}{l - 1}) ((1 - p_m)^{o(H)})$$

SCHEMA ANALYSES...

- ➡ The Schema Theorem only shows how schemas dynamically change, and how short, low-order schemas whose fitness remain above the average mean receive exponentially growing increases in the number of samples.
- ➡ It cannot make more direct predictions about the population composition, distribution of fitness and other statistics more directly related to the GA itself.

REFERENCES

- David E. Goldberg, “Genetic Algorithms in search, Optimization& machine learning, 2006.
- Sivanandam, S.N. and Deepa ,S.N, “ An introduction to genetic Algorithms”, 2008.