CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

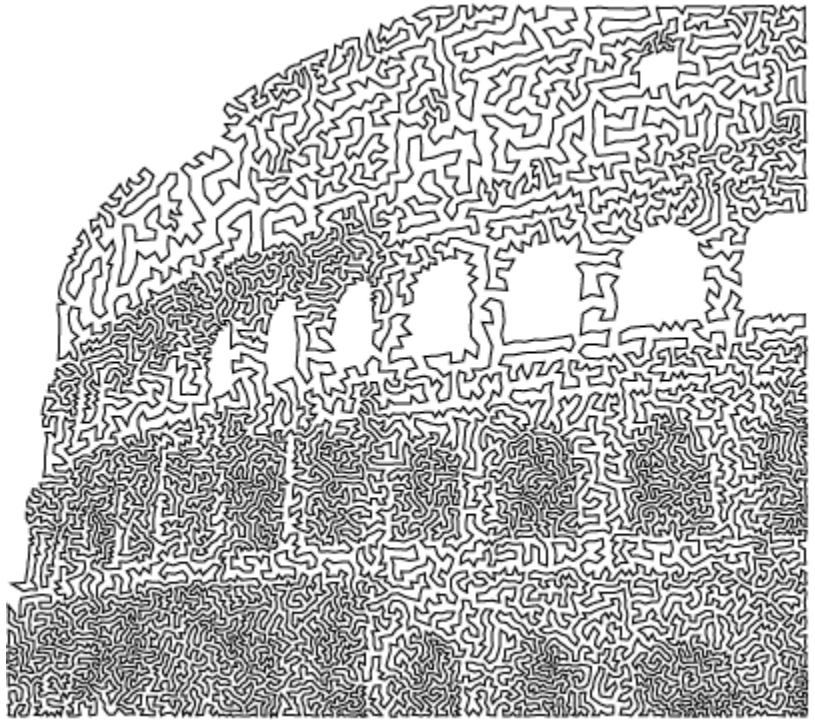
Computing@CIT

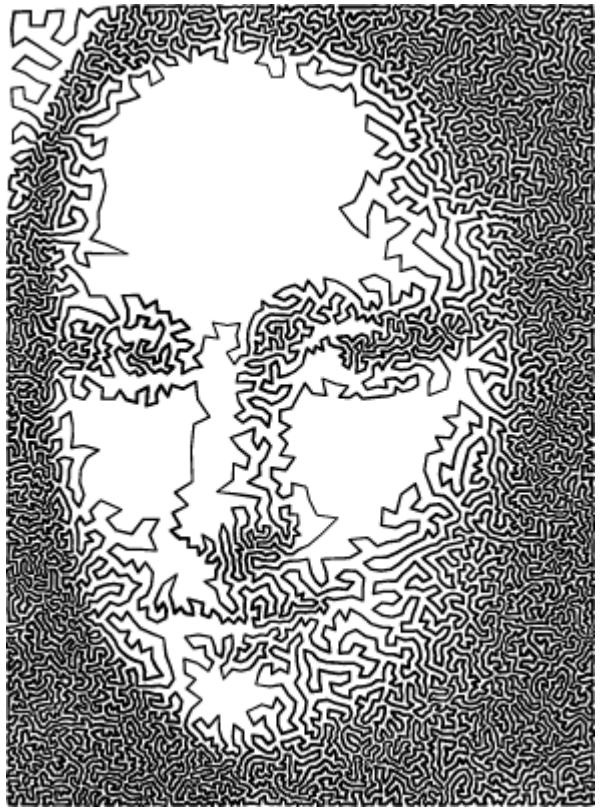
Metaheuristic Optimization

Genetic Algorithms

Dr. Diarmuid Grimes

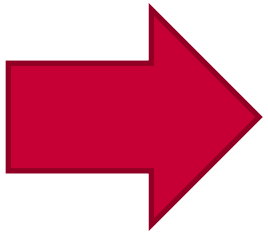
TSP Art

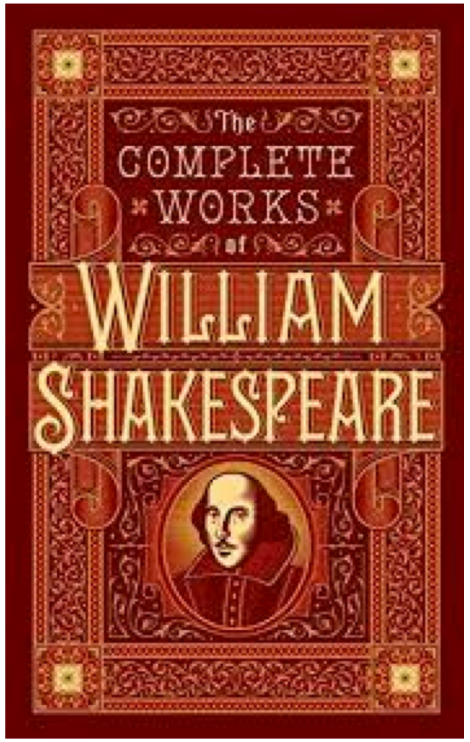




http://www.cgl.uwaterloo.ca/csk/projects/tsp/

Infinite Monkey Theorem





Given enough time, a hypothetical chimpanzee typing at random would, as part of its output, almost surely produce all of Shakespeare's plays.

Practical Theorem

- The probability of a monkey exactly typing a complete work such as Shakespeare's Hamlet is so tiny that the chance of it occurring during a period of time of the order of the age of the universe is minuscule, but not zero
- Example: typing 'banana'
  - Assume only 50 keys
  - Prob. Of each letter to be typed right is 1/50
  - Prob. Of that 'banana' is typed right is (1/50)^6 < 1 in 15 billion
  - Expected number of trials to write 'banana' = 15 billion

Monkeys, typewriters and Evolution

- Evolution being a randomized generate-and-test process present some similarities with the typewriter process

- However, natural selection can produce unlikely results. Could a monkey accidentally type the Hamlet line “To be or Not To Be”? The chances are virtually zero.

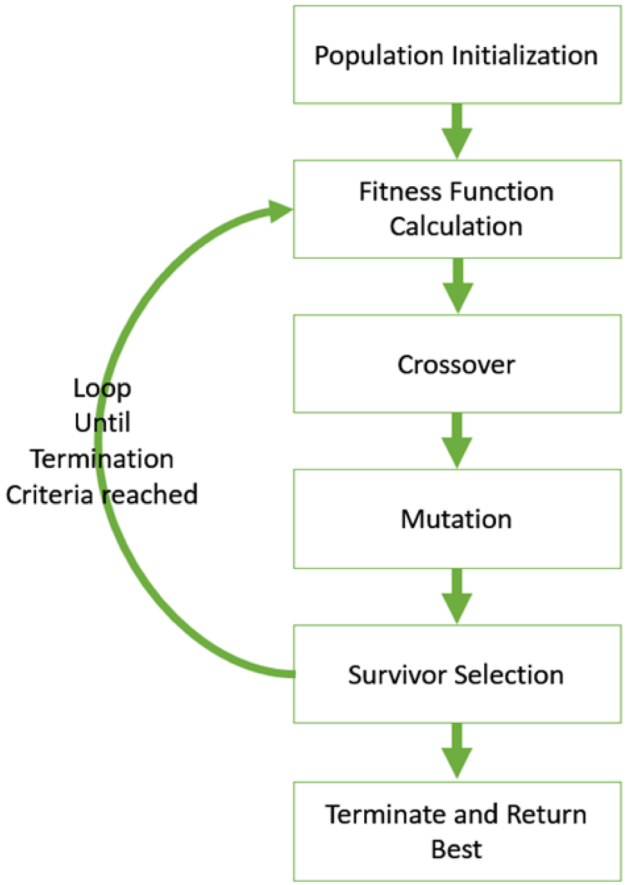
- How does an evolutionary algorithm work?

Evolutionary Computation

- There's a variety of classes of Evolutionary Algorithms

- Today, we will only look at the so-called **Simple Genetic Algorithm**

Evolutionary computation

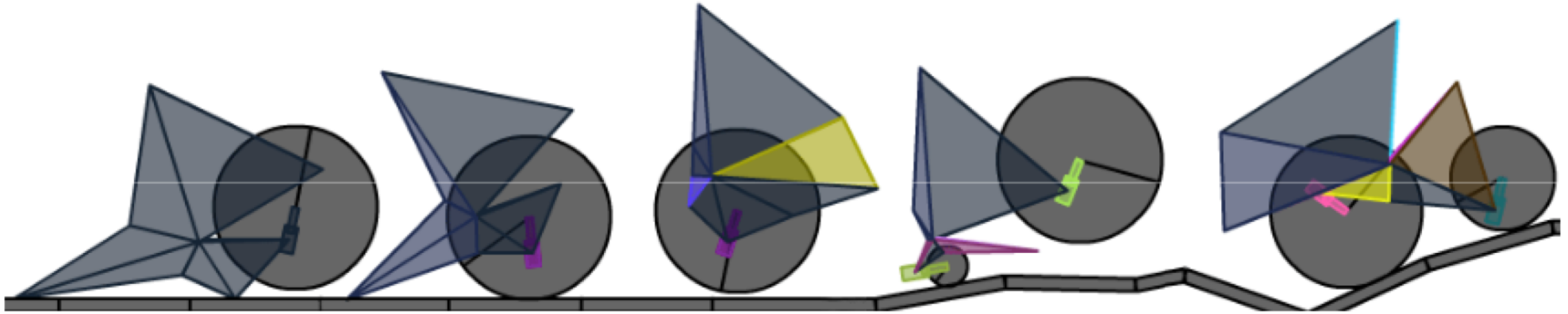


Characteristics of GAs

Flexible:	Applicable to different problems
Robust:	Can deal with noise and uncertainty
Adaptive:	Can deal with dynamic environments
Autonomous:	Without human intervention
Decentralized:	Without a central authority

Structure Design: Car Evolution

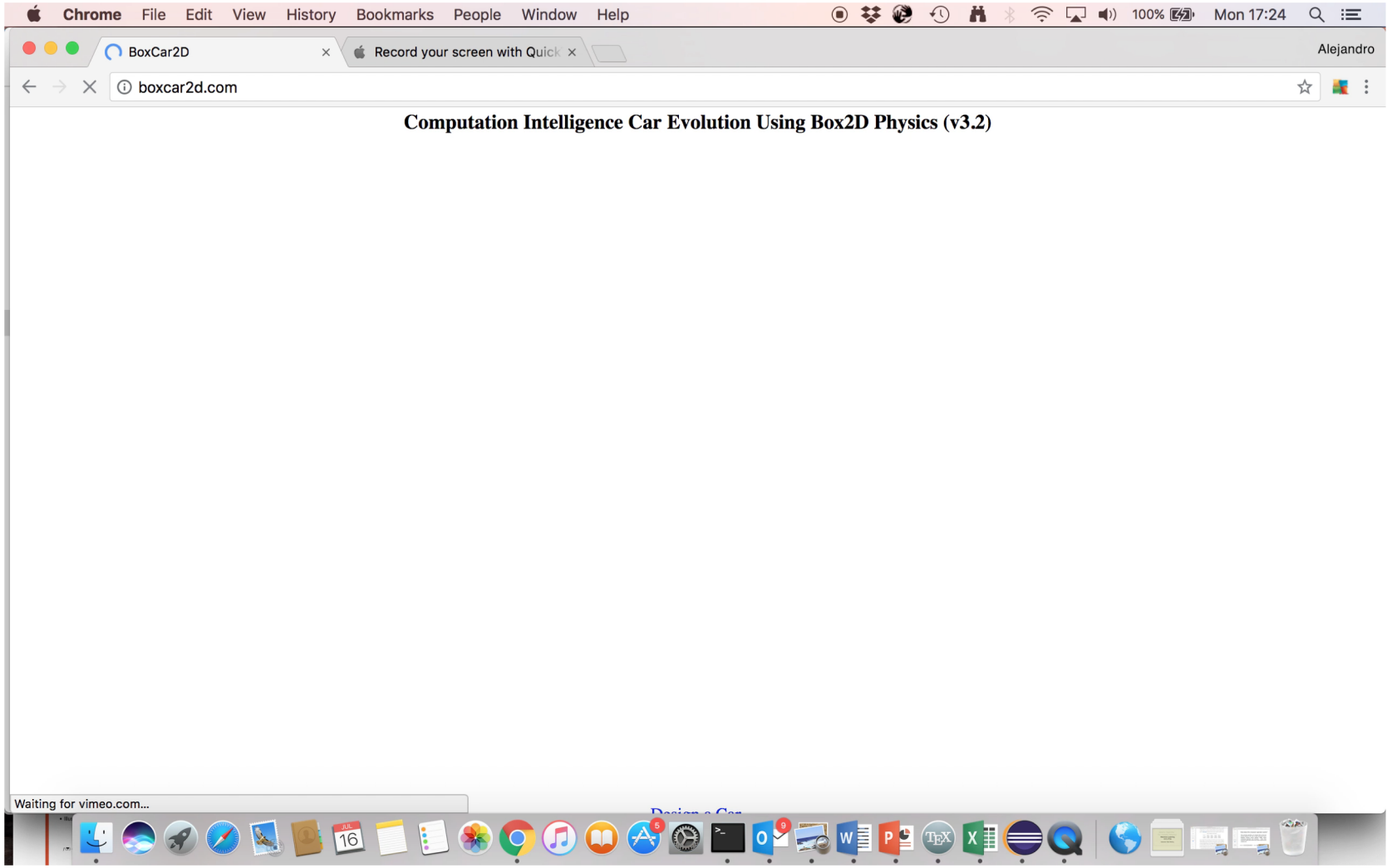
Problem: Design a car using polygons and wheels able to run on a terrain



Structure Design: Car Evolution

- A **candidate solution** is a set of polygons connected in a central point, and wheels attached to them.
- **Representation**: for each polygon there is a real vector (a “gene”) describing the shape of the polygon. For each wheel there is a value specifying its radius, location, and its centre
- **Fitness**: how far the car goes on the “terrain” when run
- **Mutation** and **Crossover**: variations on mutation and crossover for real vectors

Car Evolution Using Box2D



http://boxcar2d.com

Designing a car using Genetic Algorithms

Evolutionary Art: Mona Lisa Evolution

Problem: paint a replica of the Mona Lisa using only 50 semi transparent polygons

https://www.youtube.com/watch?v=S1ZPSbImvFE

Evolutionary Art: Mona Lisa Evolution

- A candidate solution is a set of 50 transparent polygons of various colors on the canvas
- Representation: for each polygon there is a real vector representing the shape, the location and the colour of the polygon
- Fitness (to minimize): sum of the differences in colour components (RGB) on each pixel between the phenotype and the target image
- Standard crossover and mutation on real vectors

https://www.youtube.com/watch?v=S1ZPSbImvFE

Application Areas

- Planning
  - Routing, Scheduling, Packing
- Design
  - Electronic Circuits, Neural Networks, Structure Design
- Simulation
  - Model economic interactions of competing firms in a market
- Identification
  - Fit a function to medical data to predict future value
- Control
  - Design a controller for gas turbine engine, design control system for mobile robots
- Classification
  - Game Playing, Diagnosis of heart disease, Detecting SPAM

What is a Genetic Algorithm (GA)?

- It is a search based on the mechanics of natural selection and genetics
- Often used to solve difficult optimization problems

Evolution

- Evolution can be seen as an optimization process where living creatures constantly adapt to their environment
  - The stronger survive and propagate their traits to future generations
  - The weaker die and their traits tend to disappear

- Darwin’s survival of the fittest

How do GAs differ from other techniques?

- Work with a population of solutions, rather than a single solution

- Use probabilistic rather than deterministic mechanisms

Two essential components

- Selection
- Variation
  - Recombination (or crossover)
  - Mutation

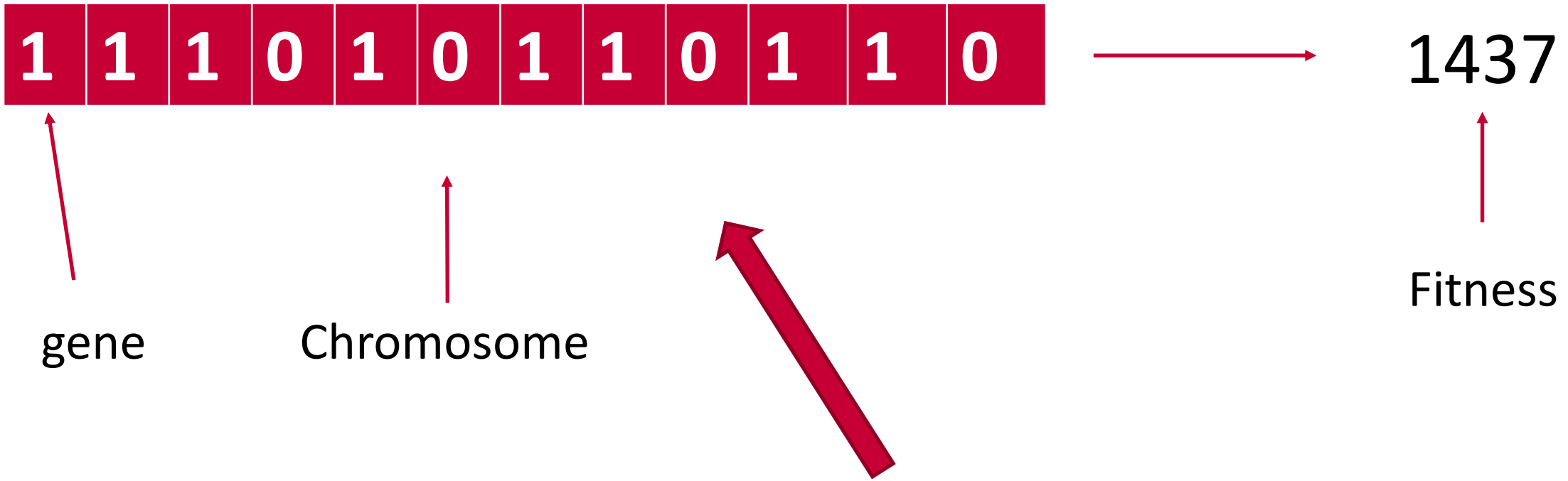
When should we use GA?

- When other simpler methods are not good enough
- When we don’t have much knowledge about the problem that we are trying to solve
- … and when the search space is very large (making complete enumeration unfeasible).

Encoding

- Problem solutions have to be encoded in some sort of structure (traditionally binary strings, but there are many other possibilities)
- We need a way of quantifying the value (fitness) of any given structure (solution)
  - Or given two solutions decide which is the better one
- The fitness value is obtained through an objective function

Terminology



Terminology

$$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_3) \wedge (x_5 \vee \overline{x_1})$$

Number of unsatisfied clauses

1	1	0	0	0

$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 0$ → Cost = 2

1	1	1	0	1

$x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 0, x_5 = 1$ → Cost = 0

Gene/ Variable

Chromosome/ Assignment

Fitness

A Simple GA

Generate a population with N random individuals.

WHILE not happy enough with the solution quality DO

- Compute the fitness of every individual in the population
- Select better individuals
- Do crossover between pairs of selected individuals with probability Pc
- Mutate each gene with probability Pm

END WHILE

One iteration of the GA

	Generation at time t
1	$Sol_{1,t}$
2	$Sol_{2,t}$
N-1	$Sol_{N-1,t}$
N	$Sol_{N,t}$

	Generation at time t+1
1	$Sol_{1,t+1}$
2	$Sol_{2,t+1}$
N-1	$Sol_{N-1,t+1}$
N	$Sol_{N,t+1}$

Selection, Recombination, and Mutation

Selection

- Simulates survival of the fittest
- The stronger have a better opportunity to reproduce
- The weaker tend to disappear from the population
- Various methods to implement this operator (roulette wheel, ranking, tournament, and many others)

Let's assume random selection

Example: binary tournament selection

- Pick a pair of individuals from the population
- The winner survives, the loser dies
- Repeat N times (N is the population size)

Example

individual	Fitness
A	8
B	2
C	4
D	5

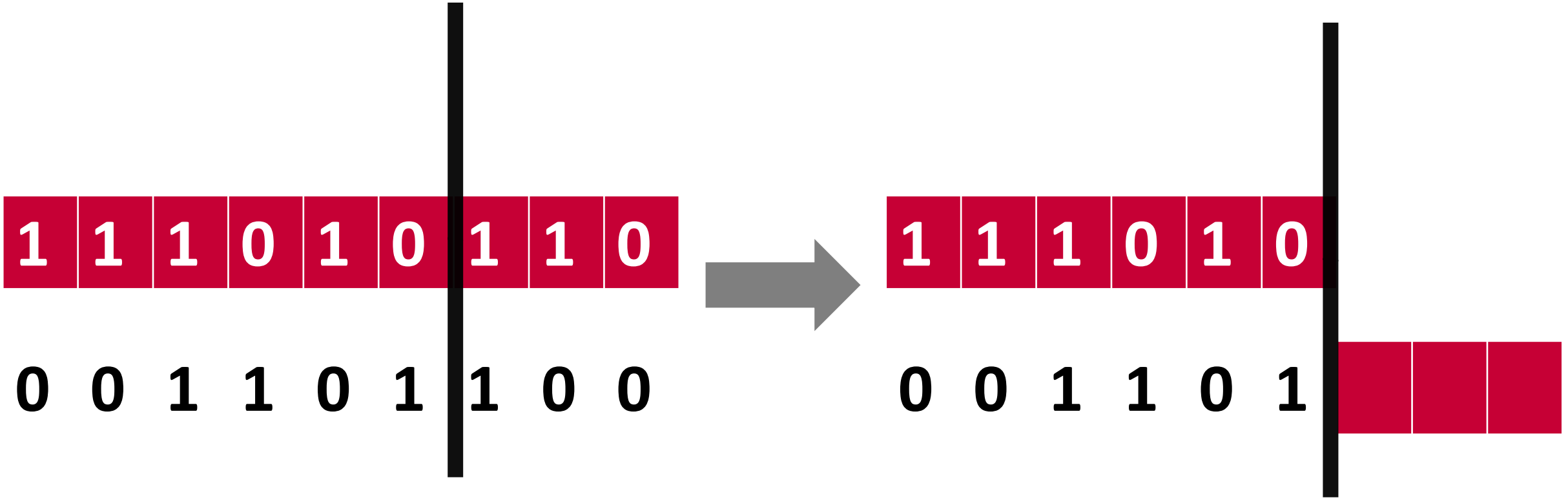
4 tournaments:	(A, D),	(B, C),	(A, B),	(C, D)
4 winners:	A,	C,	A,	D

Crossover

- Recombine 2 individuals (father and mother) to obtain two new individuals (the children)

- In the previous example, **A** recombines with **C**, and **A** recombines with **D**

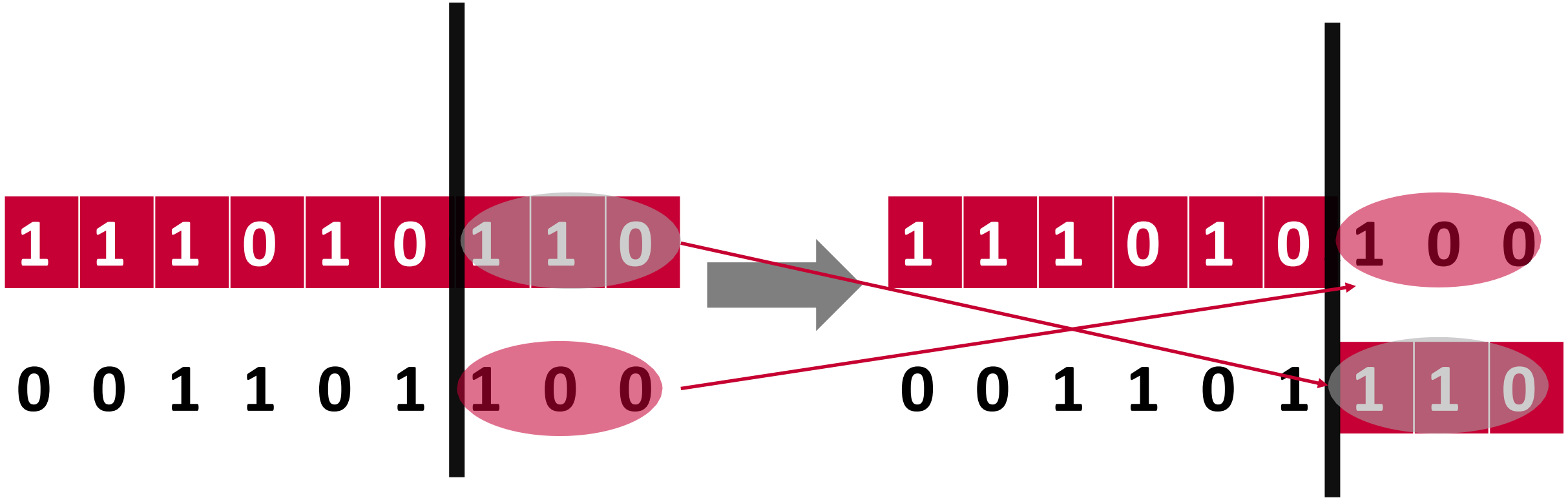
Example of crossover



Before recombination

After recombination

Example of crossover



Before recombination

After recombination

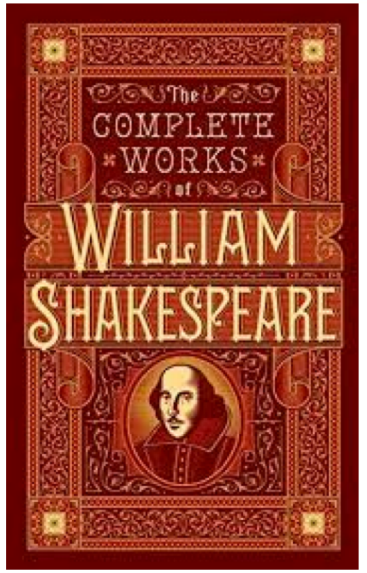
Mutation

- With probability P_m, flip a gene from 0 to 1, or from 1 to 0
- In traditional GAs, this operator is typically used with a low probability

Gene 7 was mutated

Demo

Target → “To be or not to be”
Fitness → percentage of correct symbols
nr5**b**kRB/O.?6,zbnDt→ 0.05



n	r	5	b	k	R	B	/	O	.	?	6	,	z		n	D	t

Correct symbol: Fitness → 1/size(To be or not to be) → 1/18 → 0.05

T	o		b	e		o	r		n	o	t		t	o		b	e

Example demo

```
class Individual:
    def _init_(self, num):
        self.fitness = -1
        self.genes = []
        self.genSize = num
        for i in range(0, num):
            self.genes.append( chr(random.randint(32, 128)) )

    def getPhrase(self):
        return "".join(str(x) for x in self.genes)

    def getFitness(self):
        return self.fitness

    def computeFitness(self, target):
        score = 0.0
        for i in range(0, len(self.genes)):
            if self.genes[i] == target[i]:
                score+=1
        self.fitness = score/len(target)
```

Chromosome, e.g., [HeLlo World]

Random values

percentage of correct characters

Example demo

```
class GA:

    #Crossover
    def crossover(self, ind1, ind2):
        child = Individual(self.genSize)
        midPoint = random.randint(0, self.genSize)
        for i in range(0, self.genSize):
            if(i > midPoint):
                child.genes[i] = ind1.genes[i]
            else:
                child.genes[i] = ind2.genes[i]
        return child

    #Mutation
    def mutate(self, ind):
        for i in range(0, self.genSize):
            if(random.random() < self.mutationRate):
                ind.genes[i] = chr(random.randint(32, 128))
```

Midpoint crossover
Half from one and half from the other

Mutation rate

Random modifications for a given gene

Example demo

```
def GAStep(self):
    matingPool = []
    for gene_i in self.population:
            elementsInPool = int(gene_i.getFitness() * 100)
            for i in range(0, elementsInPool):
                    matingPool.append(gene_i)

    for gene_i in range(0, len(self.population)):
        indexPartnerA = random.randint(0, len(matingPool)-1)
        indexPartnerB = random.randint(0, len(matingPool)-1)
        partnerA = matingPool[indexPartnerA]
        partnerB = matingPool[indexPartnerB]

        child = self.crossover(partnerA, partnerB)
        self.mutate(child)
        child.computeFitness(self.target)

        self.population[gene_i] = child
        if child.getFitness() > self.best.getFitness():
            self.best = child
```

Mating Pool
Candidate Individuals

Selection

Update fitness

Example demo

```
def search(self):
    i=0
    while i < self.maxIterations and self.best.getFitness() < 1:
            self.GAStep()
            i+=1
    print ("i: ",i)
```

1. Selection
2. Crossover
3. Mutation

Why do they work?

- We can make an analogy with the way humans are creative and innovative (Goldberg)
- Humans are creative and innovative when they combine notations that work well in some other context
- Likewise, GAs can be creative when combining pieces of a good solution, with pieces of a another good solution

Components of a GA

- Initialization
  - Usually at random, but can use prior knowledge
- Selection
  - Give preference to better solutions
- Replacement
  - Combine original population with newly created solutions
- Variation
  - Create new solutions through crossover and mutation

Variation operators in GAs

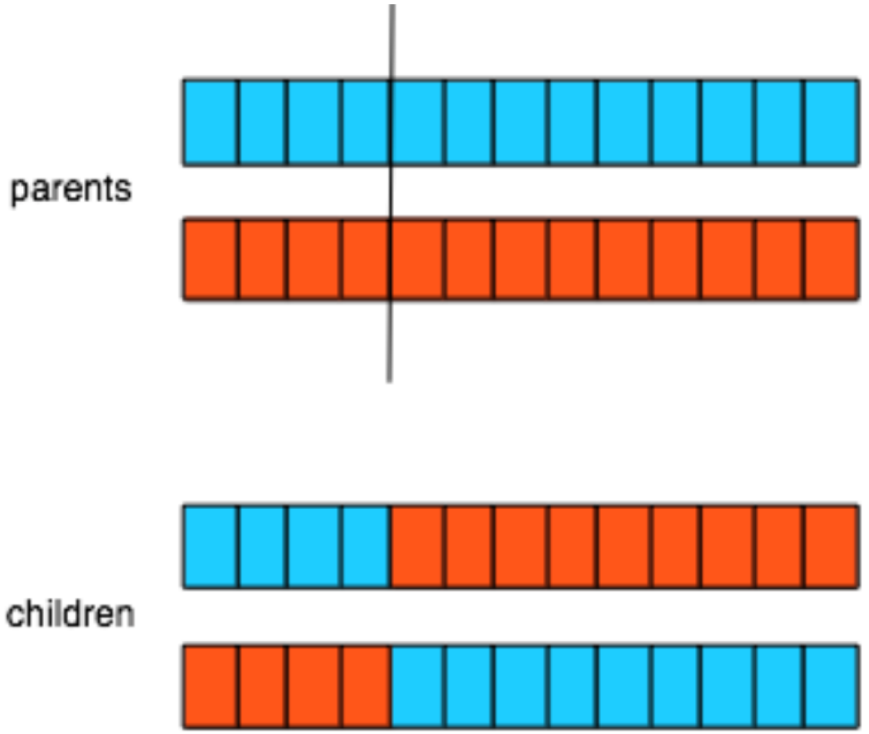
- Let’s look at some of the commonly used variation operators for Gas
  - For binary strings
  - For real-valued vectors and permutations

Commonly used crossover operators for binary strings

- One-point, two-point, *k*-point crossover
- Uniform crossover

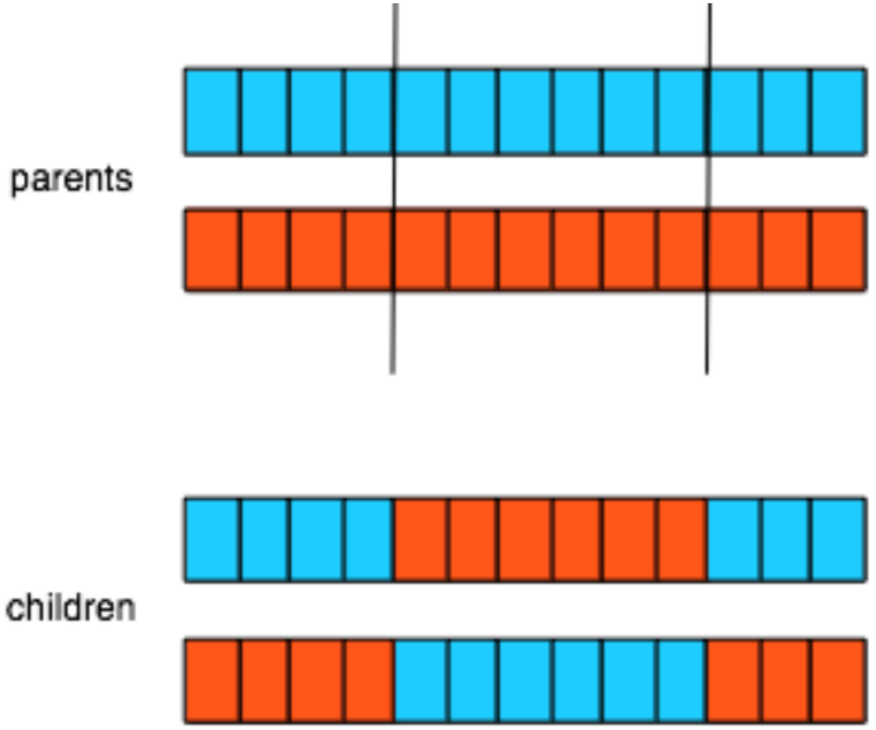
One-point crossover

- A crossing position is chosen at random and the two parents exchange all their bits after that position (like we did in our example by hand)



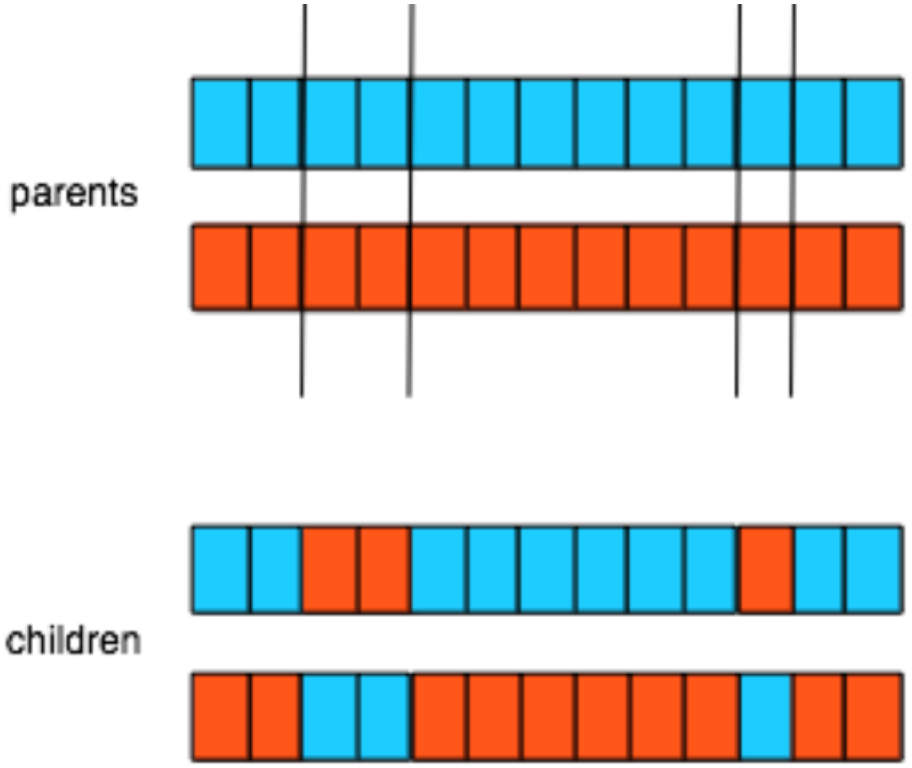
Two-point crossover

- Two crossing positions are chosen at random and the two parents exchange all their bits between the two locations



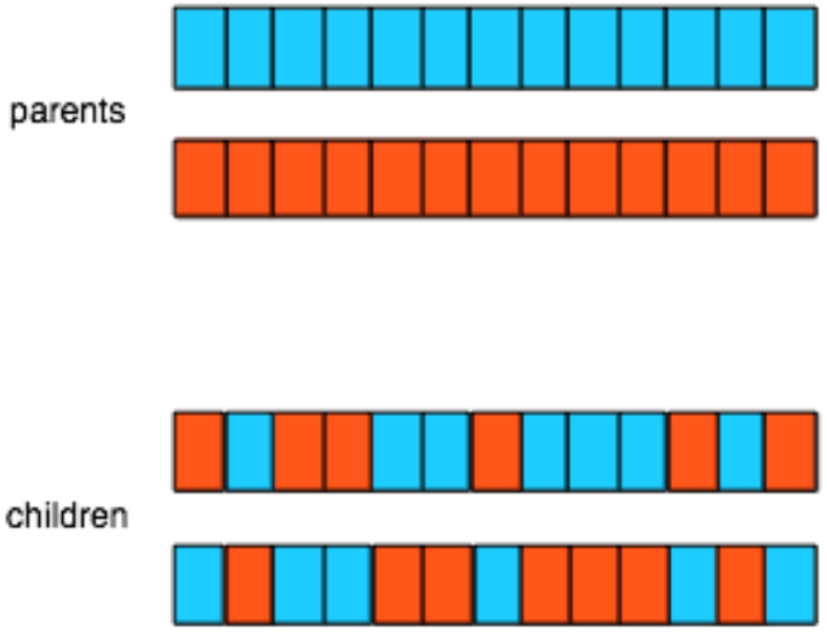
k-point crossover

- Can generalize method to k crossing positions. Here is an example with k=4



Uniform crossover

- For each position, exchange genes with a given probability (typically 0.5). Here's an example with '1011001000101'
  - 1 → exchange
  - 0 → don't exchange



k-point vs. Uniform crossover

- 1-point, 2-point, …, k-point crossover establish an implicit linkage among genes (string positions)
  - Positions close to each other are likely to be treated together (either exchanged or not)

- Uniform crossover treats every position independently
  - It is independent of the ordering of the genes

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, xover-only-EA would not work
- Adding crossover to mutation-only GA makes the GA less sensitive to the parameter Pm

Crossover or Mutation?

- **Exploration / Diversification**: Discovering promising areas in the search space, i.e. gaining information on the problem
- **Exploitation / Intensification**: Optimizing within a promising area, i.e. using information
- There is co-operation AND competition between them
- Crossover explore, it makes a big jump to an area somewhere ”in between” two parents areas
- Mutation is exploitative, it creates random small diversions, thereby staying near (in the area of) the parent

Crossover or Mutation?

- 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 crossovers)
- To hit the optimum you often need a 'lucky' mutation

TSP -- Representation

TSP – Matrix representation

- Binary n x n matrix M represents ordering information, where m_{ij}=1 if and only if city i precedes city j

e.g. the tour 1-5-2-6-3-4-1 is represented as

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

TSP - Matrix representation

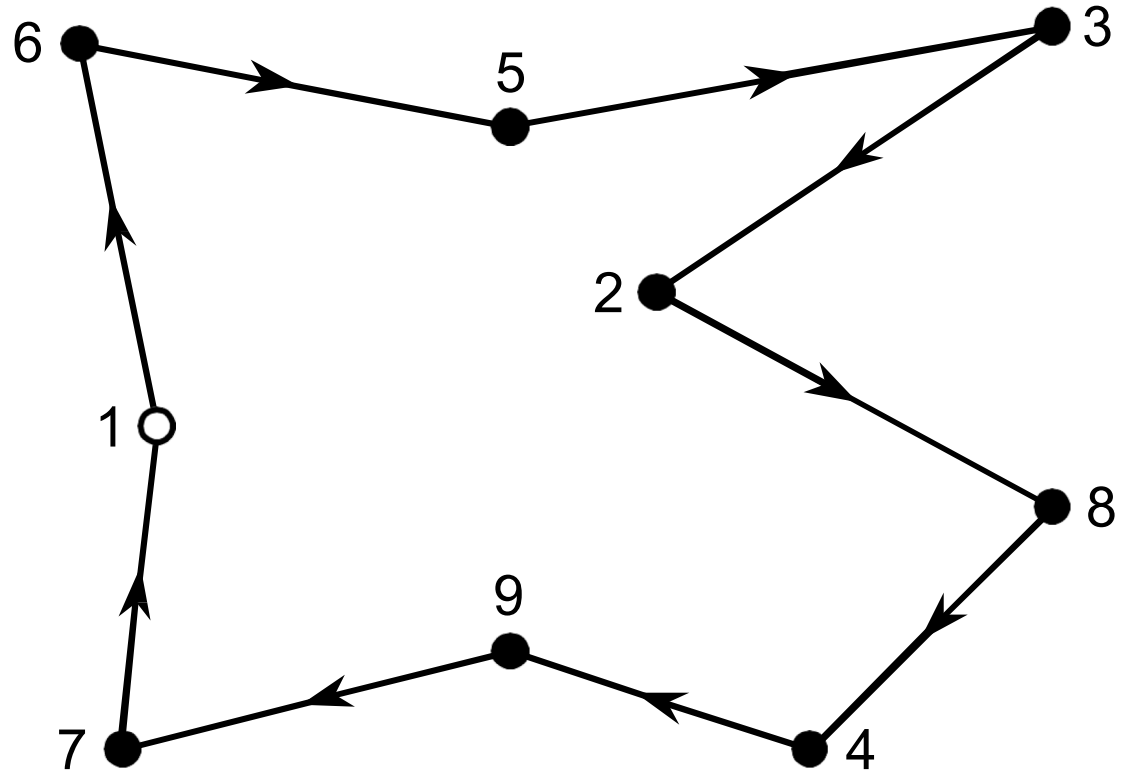
- Binary n x n matrix M represents ordering information, where m_{ij}=1 if and only if city i p

e.g. the tour 1 ented as

	1	2	3	4	5	6
1	0	1	1	1	1	1
2	0	0	1	1	0	1
3	0	0	0	1	0	0
4	0	0	0	0	0	0
5	0	1	1	1	0	1
6	0	0	1	1	0	0

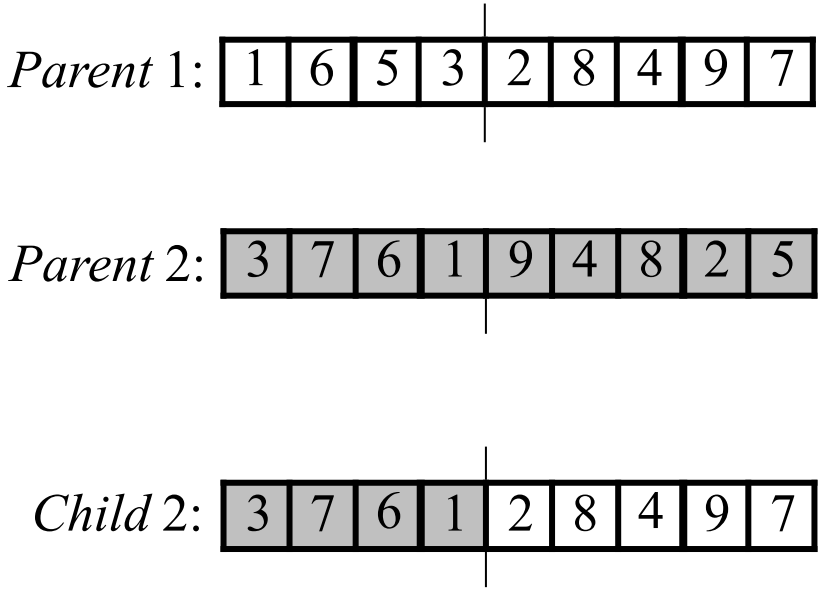
Not very practical due to memory requirements

Traveling Salesman Problem -- TSP



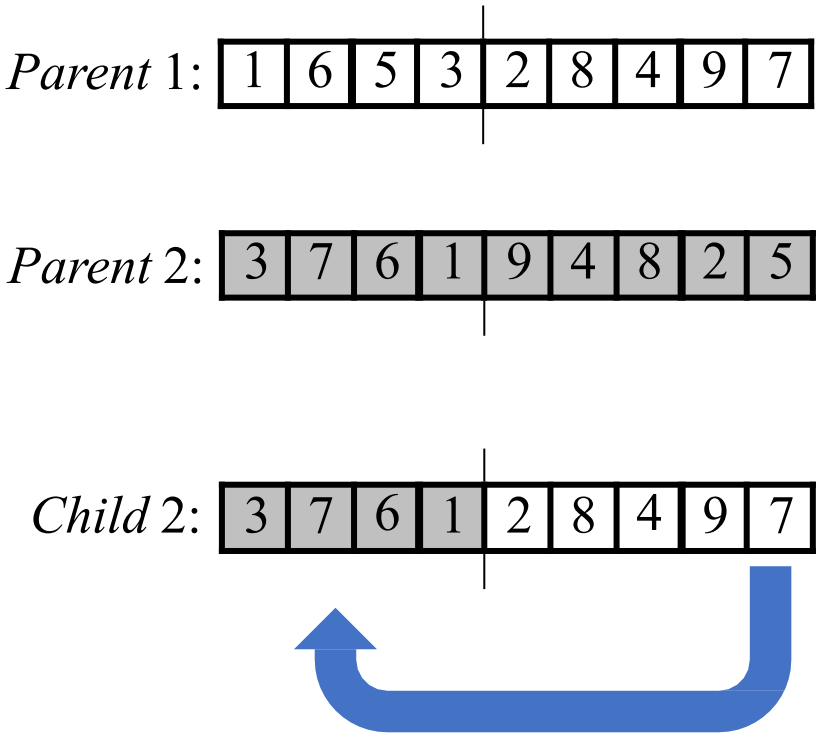
1	6	5	3	2	8	4	9	7

How does the crossover operator work?



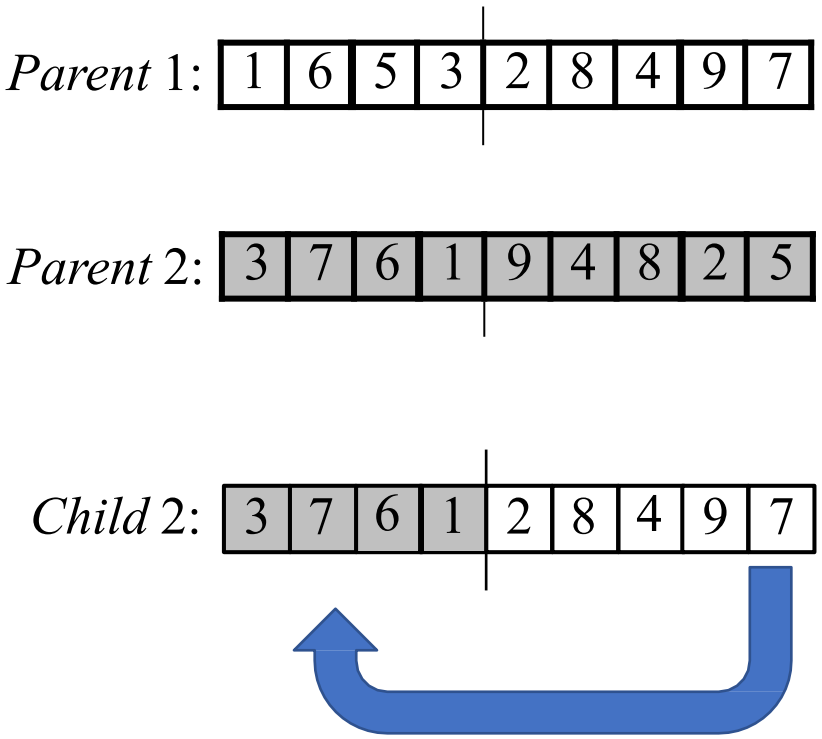
Is this OK?

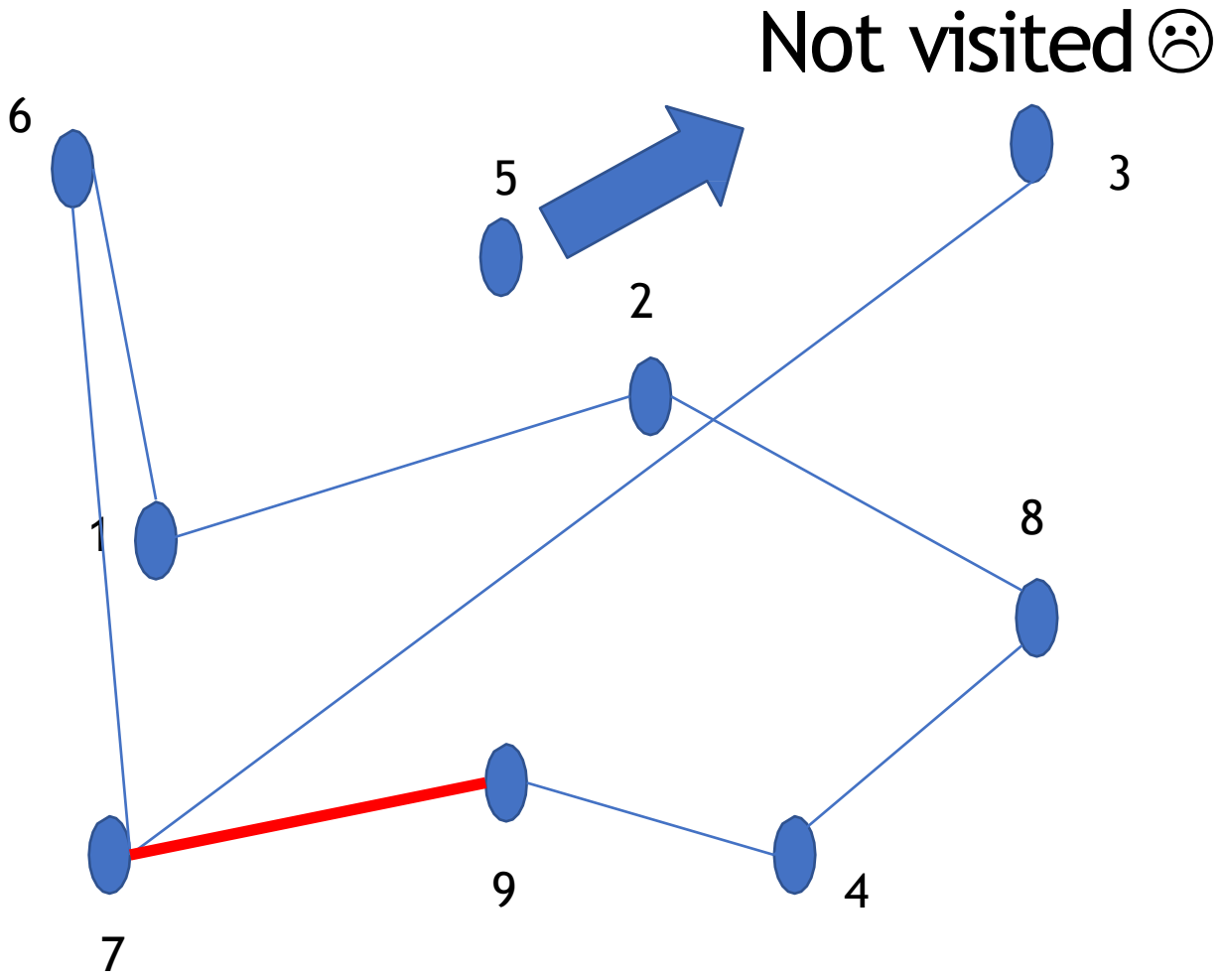
How does the crossover operator work?



One point crossover

How does the crossover operator work?

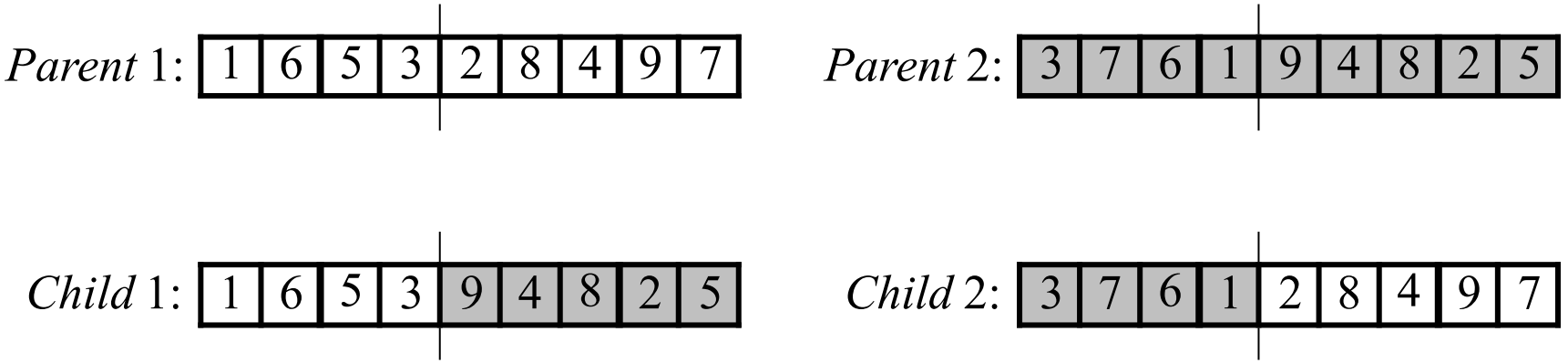




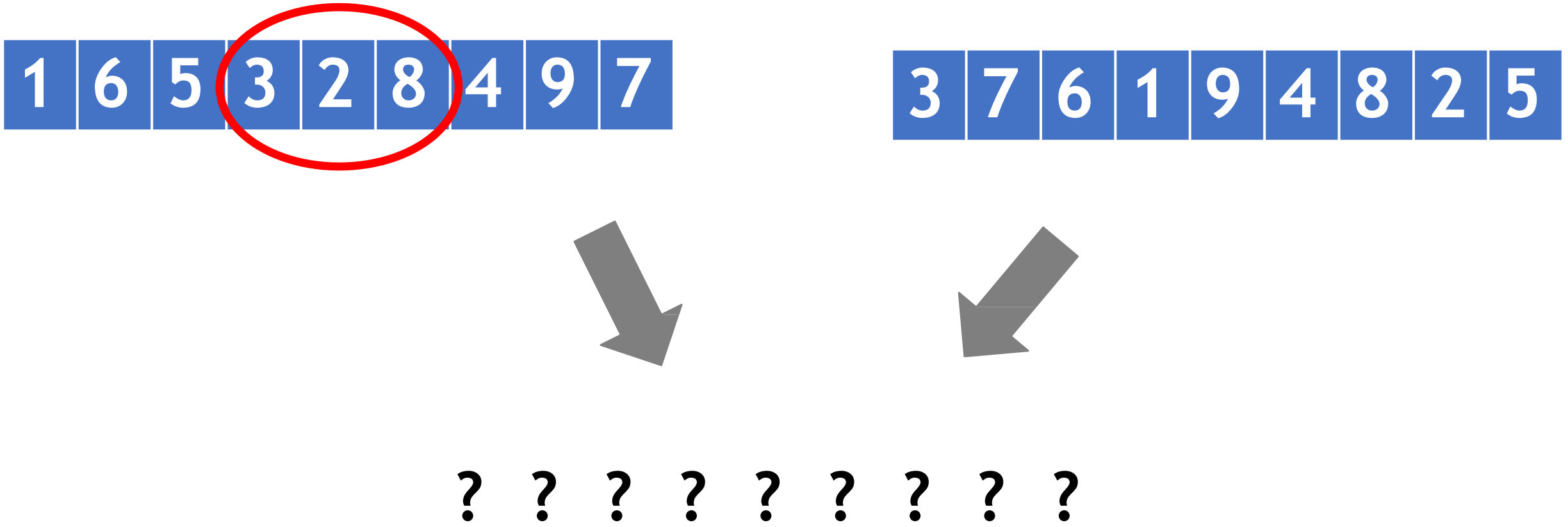
How does the crossover operator work?

The crossover operator in its classical form cannot be directly applied to the TSP.

A simple exchange of parts between parents would produce illegal routes containing duplicates and omissions - some cities would be visited twice while some others would not be visited at all.

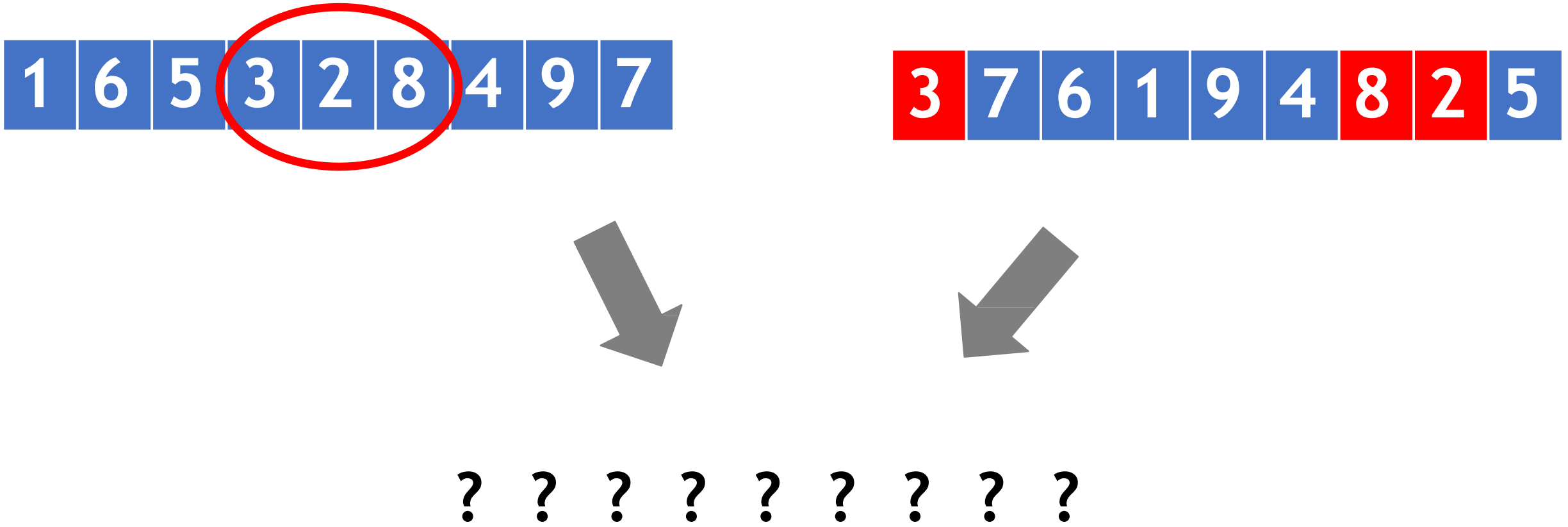


How does the crossover operator work?



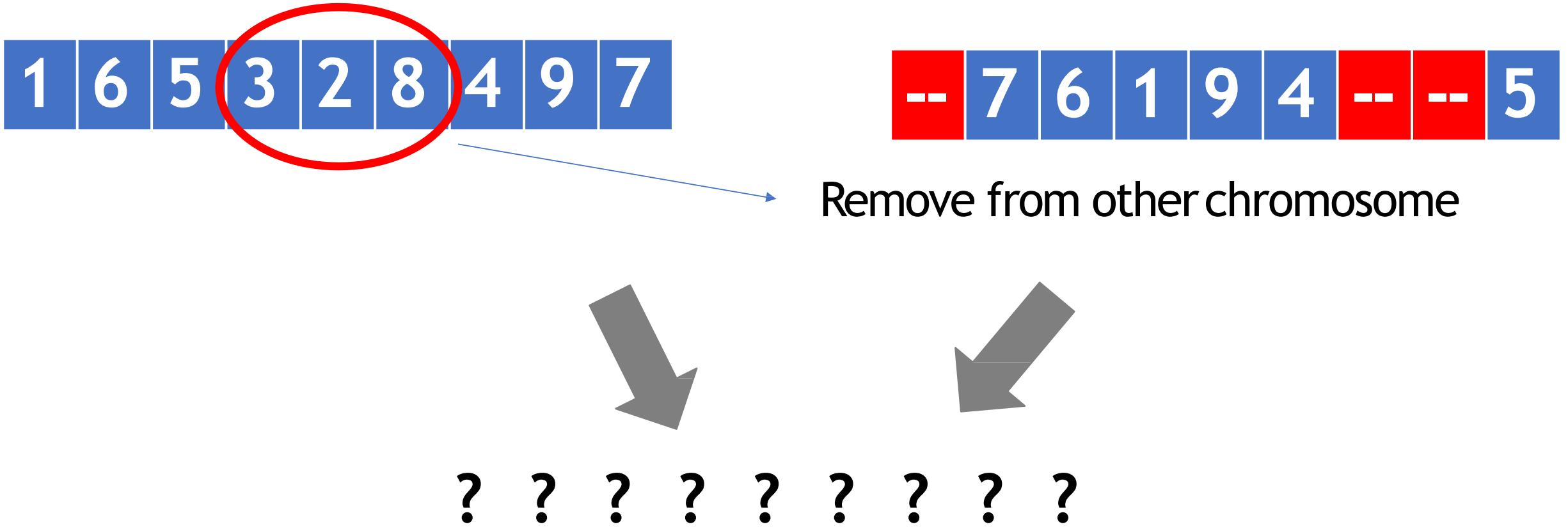
Order 1 crossover

How does the crossover operator work?



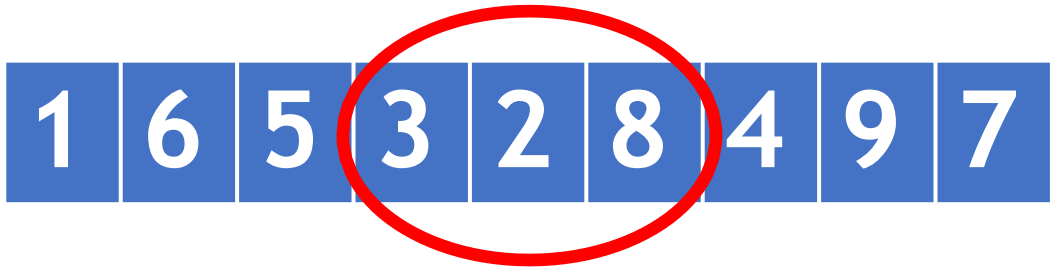
Order 1 crossover

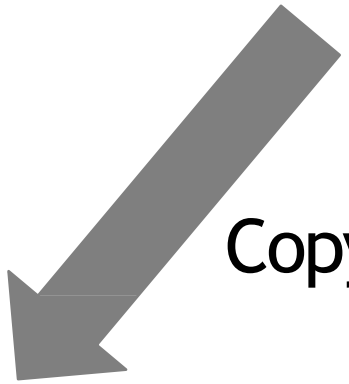
How does the crossover operator work?



Order 1 crossover

How does the crossover operator work?



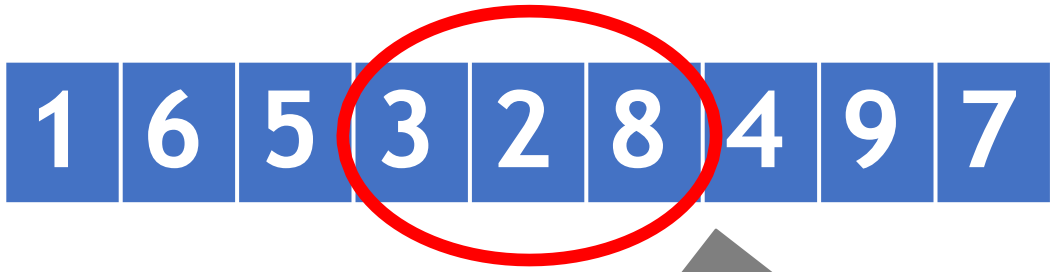


Copy the current tour in the child

7 6 1 9 4 5 ? ? ?

Order 1 crossover

How does the crossover operator work?



We can also change the order

Add selected sequence in our new child

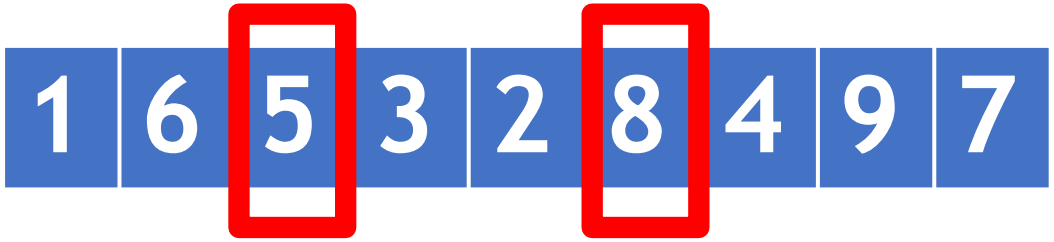
7 6 1 9 4 5 3 2 8

Order 1 crossover

How does the mutation operator work?

- Can’t just take one value and replace it with a random value
- Number of possible mutation operators for permutation problems, e.g. **reciprocal exchange** and **inversion**
- The **reciprocal exchange** operator simply swaps two random selected cities in the chromosome
- The **inversion operator** selects two random points along the chromosome string and reverses the order of the cities between these two points

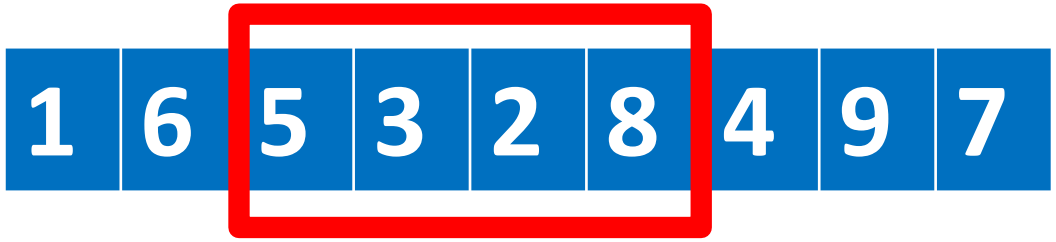
How does the mutation operator work?



Reciprocal Exchange

1 6 8 3 2 5 4 9 7

How does the mutation operator work?

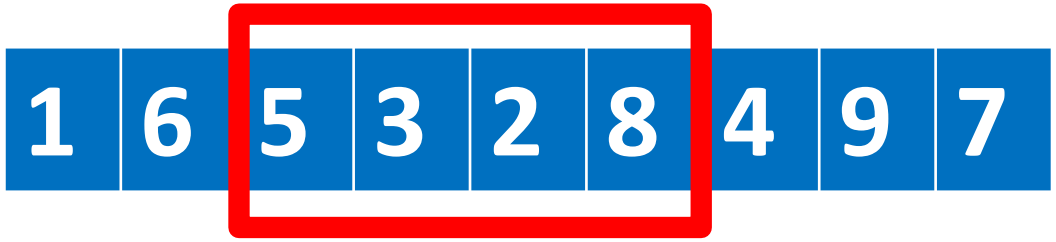


Inversion Operator

1 6 8 2 3 5 4 9 7

Alternative: Scramble Mutation

- Shuffles the elements between the selected locations



Scramble Mutation

1 6 2 5 8 3 4 9 7