

Genetic Algorithms

Sebastian JORDAN-MONTAÑO

sebastian.jordan@inria.fr

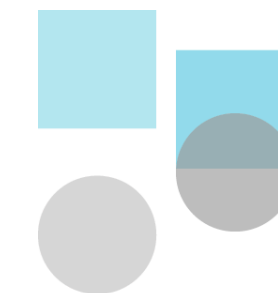
Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL



Université
de Lille



centralelille
ÉCOLE CENTRALE DE LILLE



CRIStAL
Centre de Recherche en Informatique,
Signal et Automatique de Lille



September 2023

Part 1: Genetic Algorithms

- Natural evolution
- Traveling salesman problem
- Genetic algorithms
- Limitations

Part 2: Hands on

- Modelling the traveling salesman problem
- Implementing the genetic algorithm
- Visualizing the evolution

Part 1:

Genetic Algorithms

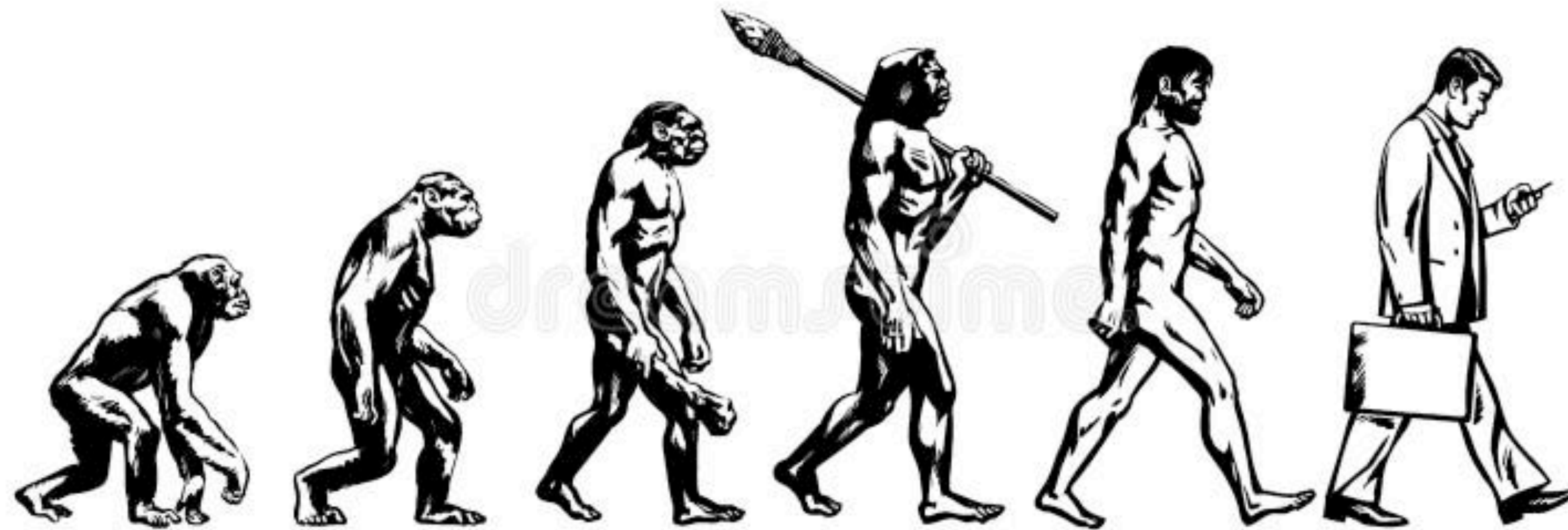
Natural Evolution

Biological evolution

- In 1859 Charles Darwin presented his famous theory. It states: « stating that species of organisms arise and develop through the natural selection of small, inherited variations that increase the individual's ability to compete, survive, and reproduce » [wikipedia]
- This process created all the forms of life is called evolution and how it is done it's called natural selection.

Natural selection

- The organisms compete between them to survive. They fight since when they were small molecules, fighting for perpetuating their DNA primordial chains against other molecules.



How evolution occurs

- Selection: the individuals that are better fitted in the fight for surviving are the one that reproduce the most, while the less fitted have problems.
- Reproduction: two individuals give part of their genes to produce a new complete individual that is unique.
- Mutation: sometime, with a low probability, the nature makes some error when combining the gens of the parents or makes some random modification.

Genetic Algorithms

The idea

1950 - 1959

- Alan Turing, considered as the father of computer science, author of the algorithm that unencrypted the messages from the Nazis called ENIGMA; established an idea about programs that can modify themselves like the organisms improve themselves in nature.
 - John Von Neumann also worked to discover mechanisms based on the evolution of species.
- Pedro Díaz Carrizo: Técnicas de computación evolutivas
 - Consuelo Puente: UCB 2018

The idea

1950 - 1959

- Nils Barricelli was in favor of adding to the selection and mutation proposals, the recombination operator (syntogenesis) to simulate artificial life.
- R. Friedberg tried to apply Darwin's ideas to make small modifications (mutations) to sequences of instructions selected for "good behavior" with a reward system.

- Pedro Díaz Carrizo: Técnicas de computación evolutivas
- Consuelo Puente: UCB 2018

The idea

1960 - 1969

- Bledroe y Bremerman were the first to think in the evolution theory as a way of solving problem, but they didn't have to much success in their first try.
- In 1965 Rechenberg established an optimisation method based on the evolution of an individual with a mutation operator.

- Pedro Díaz Carrizo: Técnicas de computación evolutivas

- Consuelo Puente: UCB 2018

The idea

1970 - 1979

- John Holland is considered as the author of the first genetic algorithm, in 1975. In his publication « Adaptation in Natural and Artificial Systems » he tried to import the natural selection to the resolution of problems in computer science.

Each one of the GA elements that we will study in this presentation were proposed by him.

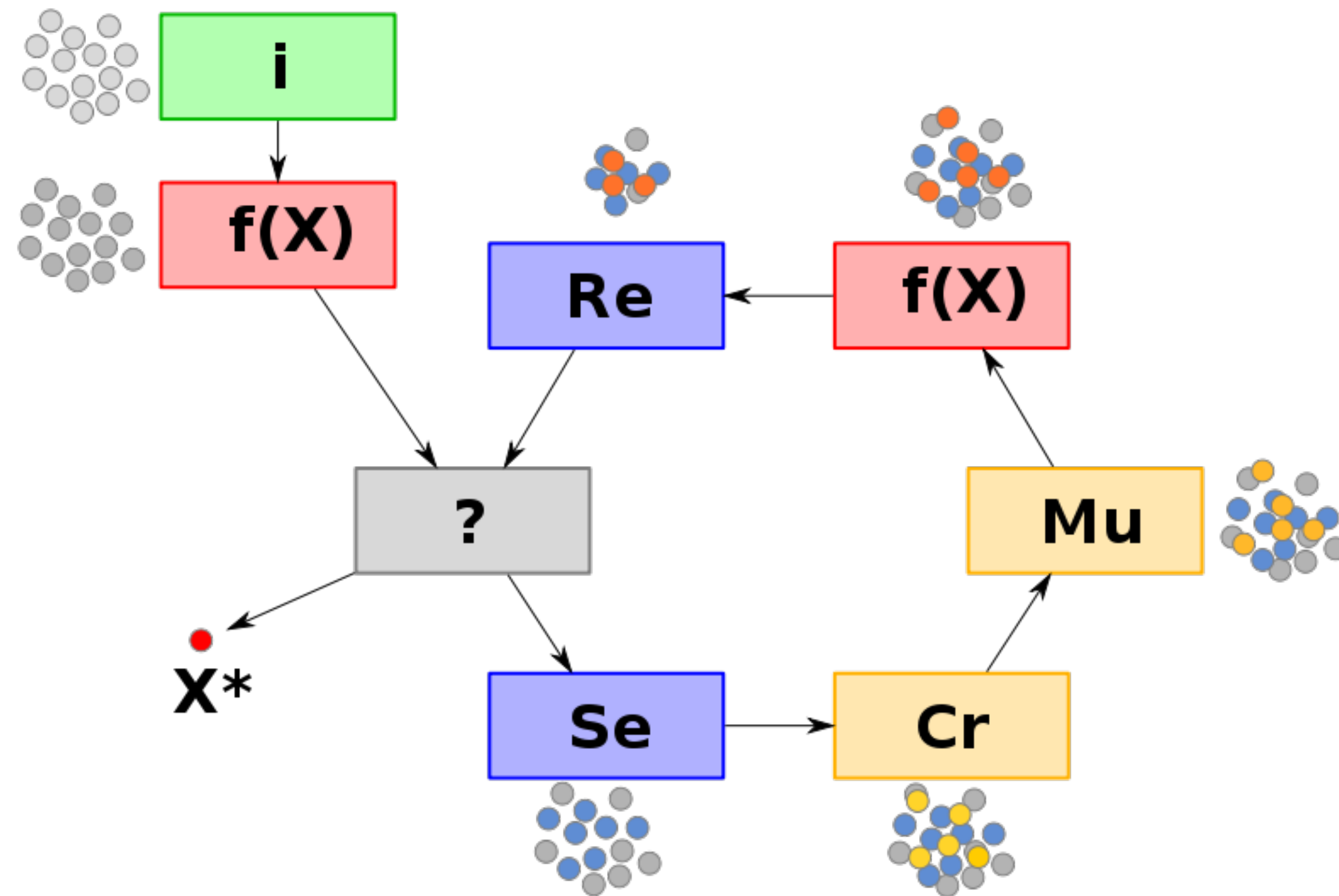
- Genetic Algorithms de Melanie Mitchell 1996
- Consuelo Puente: UCB 2018

Genetic algorithms

They are adaptive procedures for finding solutions in complex spaces, inspired by biological evolution, with operations based on the Darwinian principle of reproduction and survival of the individuals that are best adapted to the environment in which they live.

- Genetic Algorithms de Melanie Mitchell 1996
- Consuelo Puente: UCB 2018

Genetic algorithm graphical representation



Genetic algorithm i: initialisation, $f(X)$: evaluation, ?: termination condition, Se: selection, Cr: crossover, Mu: mutation, Re: replacement, X^* : best solution.

Apply natural selection to a problem requires

- To model the problem as a chain of characters. We will call this a chromosome.
- To choose the population size.
- Generate the individuals that will be in the initial population
- Define a fitness function that estimates the optimal individual.
- To choose the initial probabilities, crossing probability, mutation probability.
- To define the reduction operator

Simple genetic algorithm pseudocode

```
population := self generatePopulation « with an initial size of P ».
```

```
[ self doesItConverge ] whileFalse: [
```

```
    descendants := OrderedCollection empty.
```

```
    numberOfCrosses := self calculateNumberOfCrosses.
```

```
    numberOfCrosses timesRepeat: [
```

```
        « Choose father and mother randomly. The more fit has more probability of  
        being chose » -> father, mother.
```

```
        Cross father with mother -> childOne, childTwo
```

```
        Mutate childOne, childTwo with a probability
```

```
        descendants addAll: { childOne . childTwo } ].
```

```
    self reducePopulation « to size P » ].
```

```
^ population bestChromosome
```


Non essential variants of the classic genetical algorithm

- The crossover operation is not trivial when the chromosomes belong in a state space that is made up of permutations.
- In the case of mutation, a character in a chromosome could not be changed when it would break some chromosome formation criterion or rule.
- Reduction is not always so strict. It may be desirable for the success of the algorithm that the population can vary in size within a certain range. It may be important to keep low adaptation chromosomes along with the other

When a population converges

When the individuals of the population as a whole are approaching a unique solution, i.e. when the whole population has evolved towards a "best solution".

The time it takes for a genetic algorithm until the population converges to a unique solution depends on the size of the population under consideration and is $O(n \log n)$. At least that's what Goldberg found, which is excellent news, since he says that the time required to finish depends exclusively on the size of the input in a nearly linear fashion.

Holland thought that it was enough for half of the chromosomes to have a value of the chromosomes to have a high fitness value.

Important considerations

A genetic algorithm searches the state space of possible solutions to the problem, encoded as chromosomes, for the best of all or one of the best. That is, it prefers those with the best adaptation, as measured by their evaluation function.

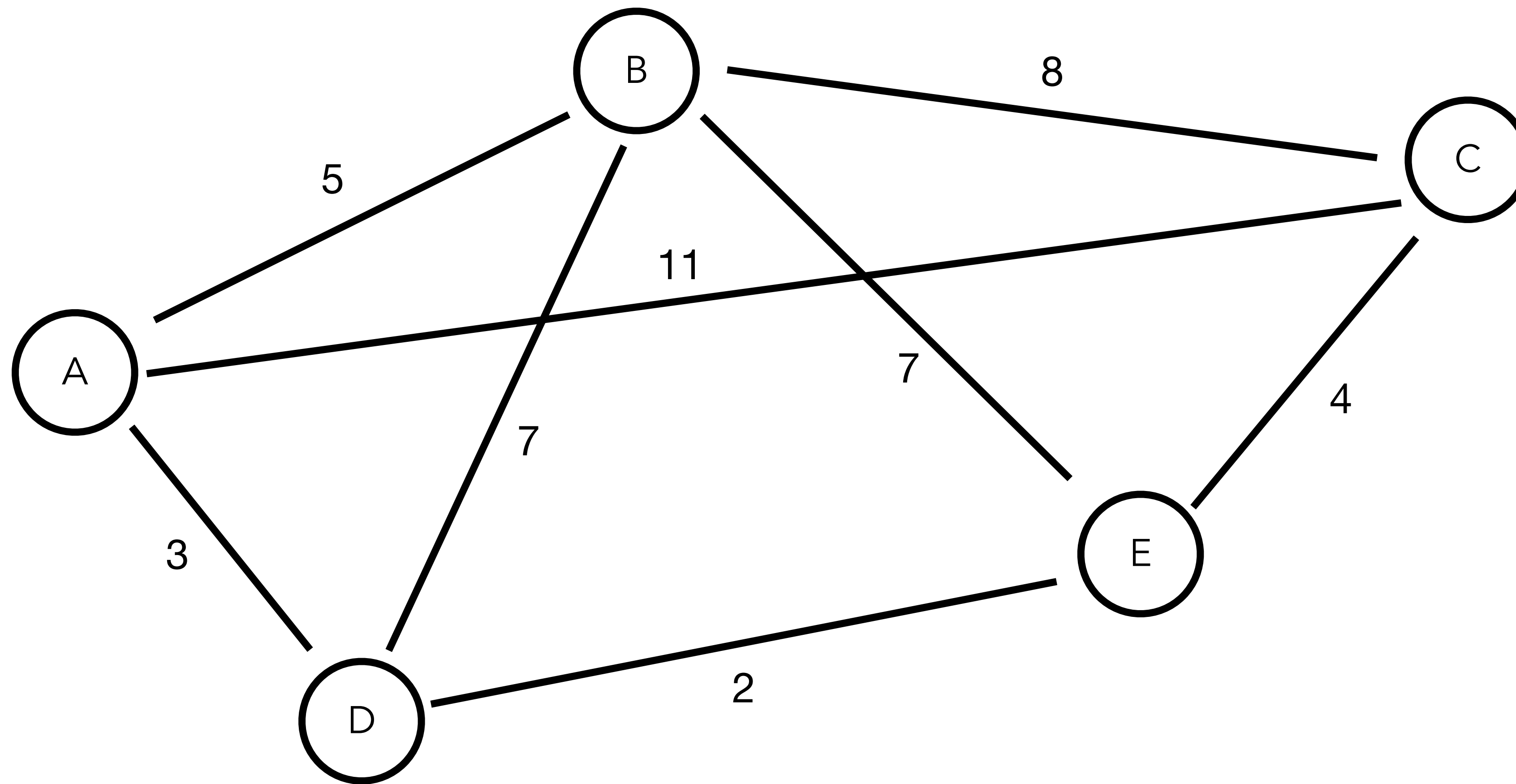
To have a good chance of success, we have to be sure that the space is explored "everywhere", i.e. that reproduction and mutation guarantee the emergence of individuals in the most diverse and hidden places of the state space. This is important to avoid local minima (or maxima).

Some limitations

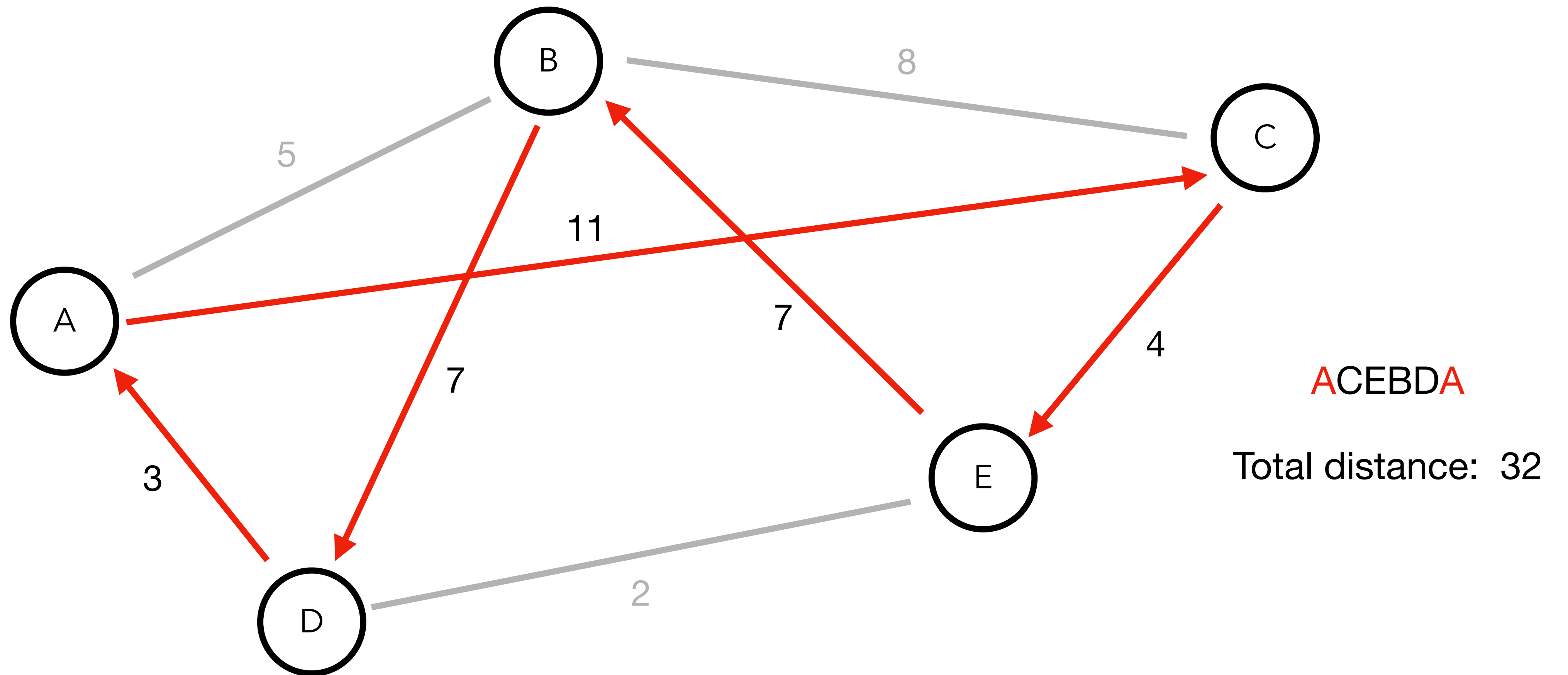
- Multiple fitness function evaluation
- Exponential increase in the search space when the modelling of the problem is too complex
- The stopping criterion is not clear. We don't know in advance the best solution.
- Solution can converge to a local minima (or maxima).

Traveling Salesman Problem

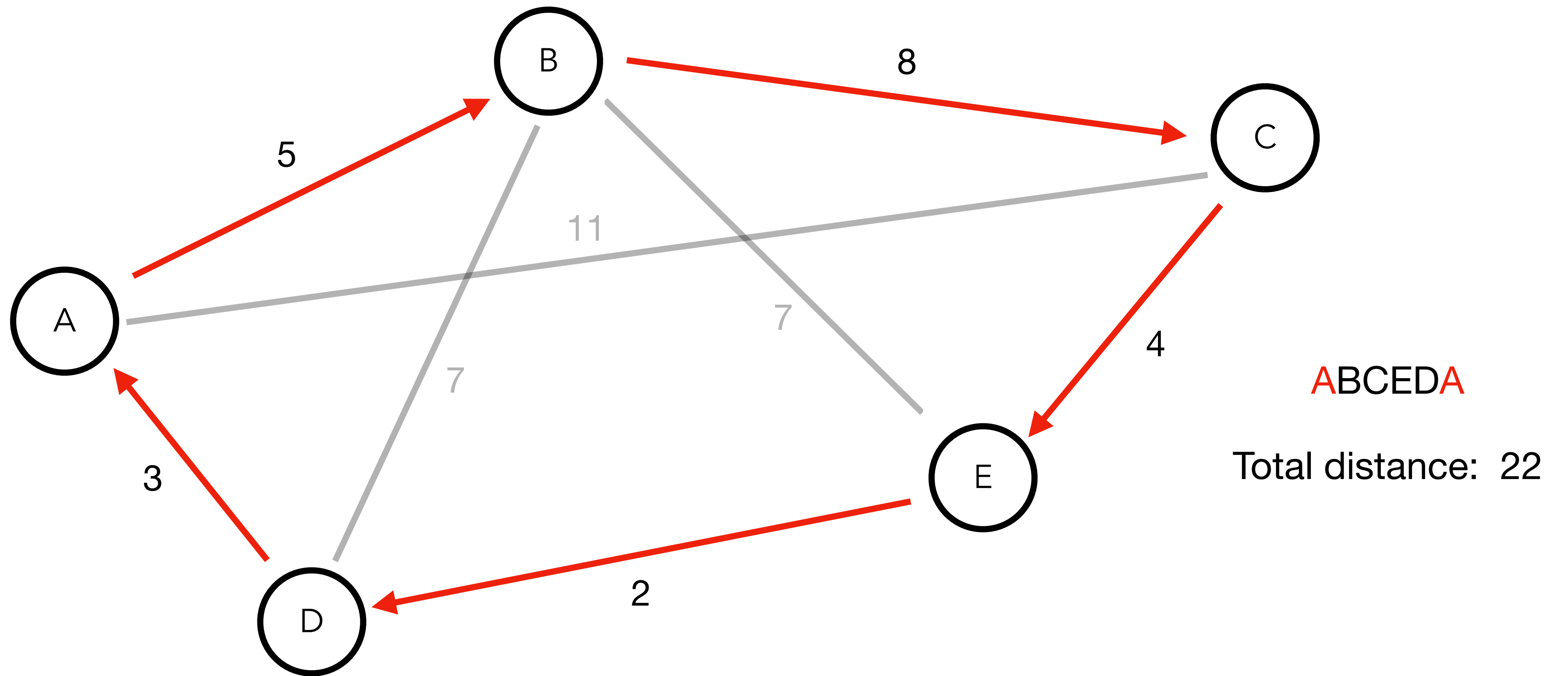
Traveling Salesman Problem



Traveling Salesman Problem



Traveling Salesman Problem



Modelling the traveling salesman problem: Adjacency matrix

All nodes are connected to all nodes. One can put an infinity distance to simulate that two nodes are not connected.

	A	B	C	D
A	0	d1	d2	d3
B	d1	0	d5	d6
C	d2	d5	0	d8
D	d3	d6	d8	0

Part 2:

Hands on

An individual

An individual must know

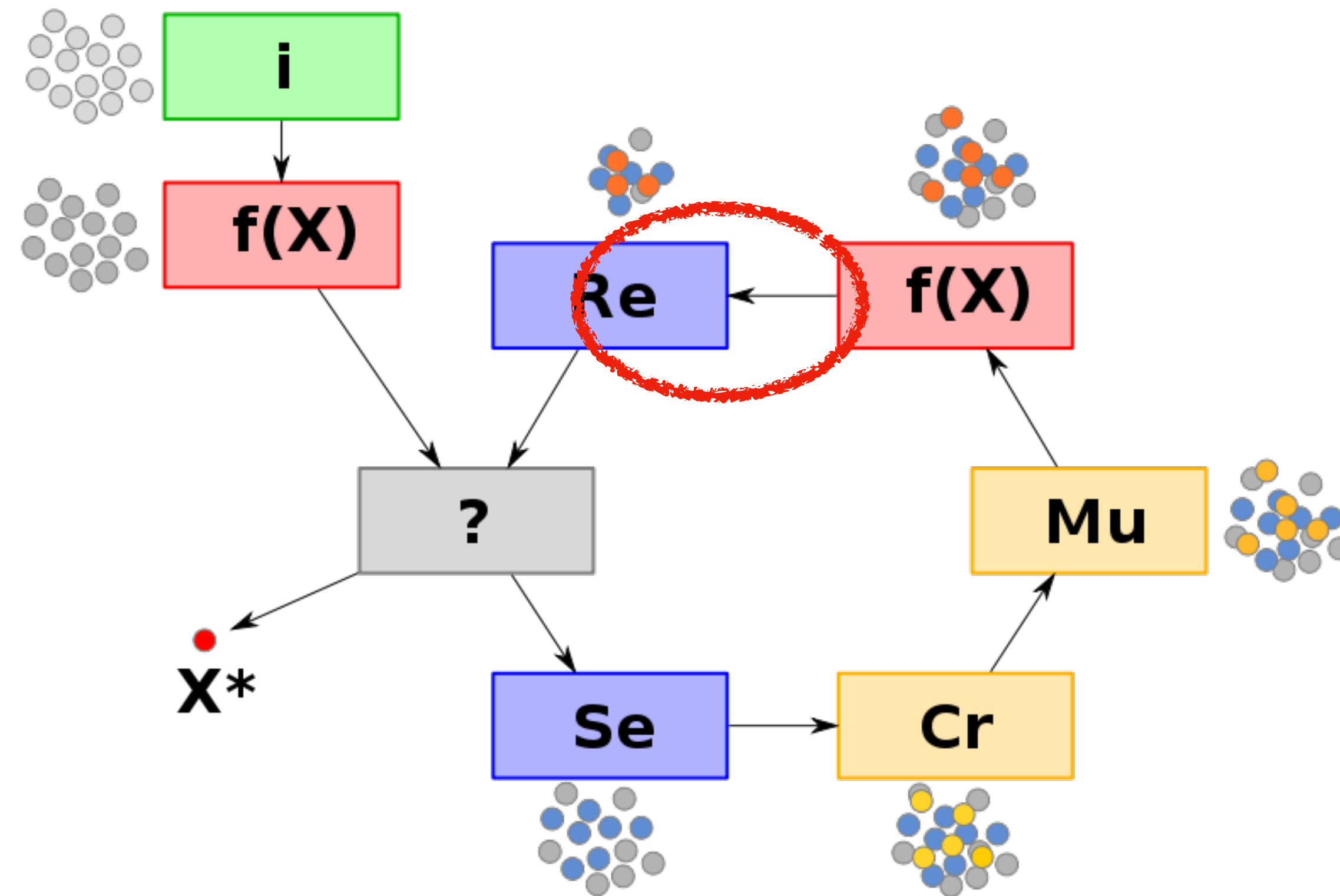
- How to cross himself with another individual
- How to mutate itself
- To calculate its fitness function (how well is adapted to solve the problem)
- To random generate another individual (helper method)

Generate an individual randomly

- The initial city is always the first and the last
- Randomly put the rest of the cities

ACFEBHGDIA

Genetic algorithm graphical representation



Genetic algorithm i: initialisation, $f(X)$: evaluation, $?$: termination condition, Se: selection, Cr: crossover, Mu: mutation, Re: replacement, X^* : best solution.

Cross two individuals

①

ACFEBHGDIA

②

AIHBGCEDFA

A A

Cross two individuals

①

ACFEBHGDI A

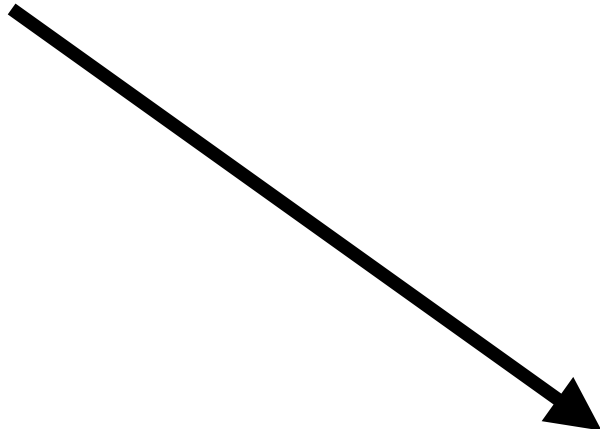
4

7

②

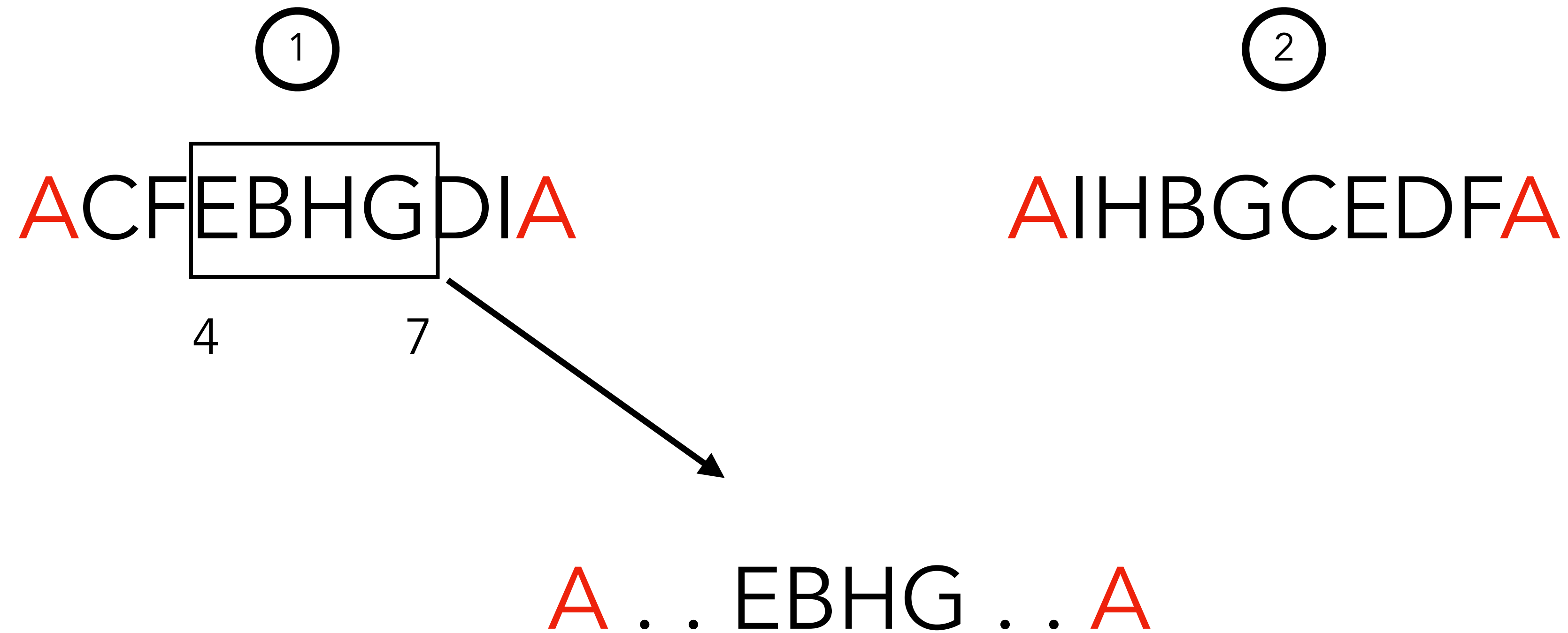
AIHBGCEDFA

Randomly select the range

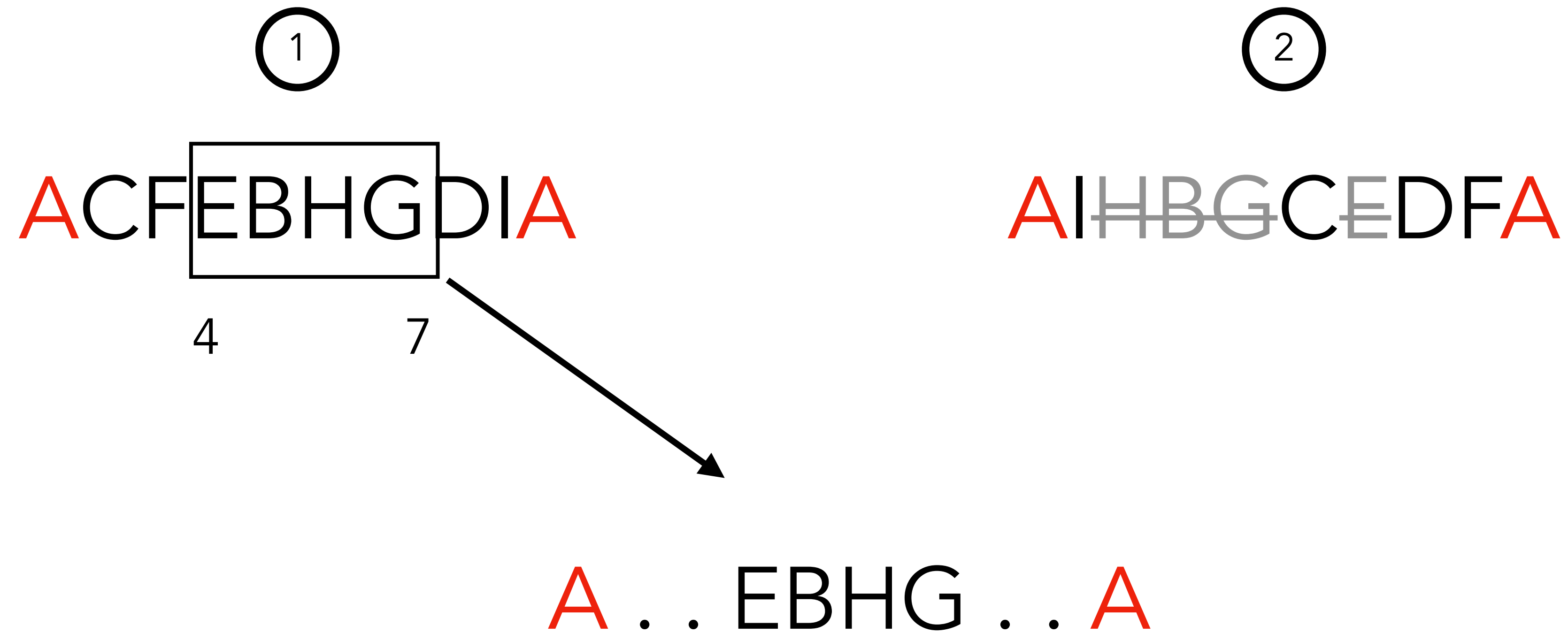


A A

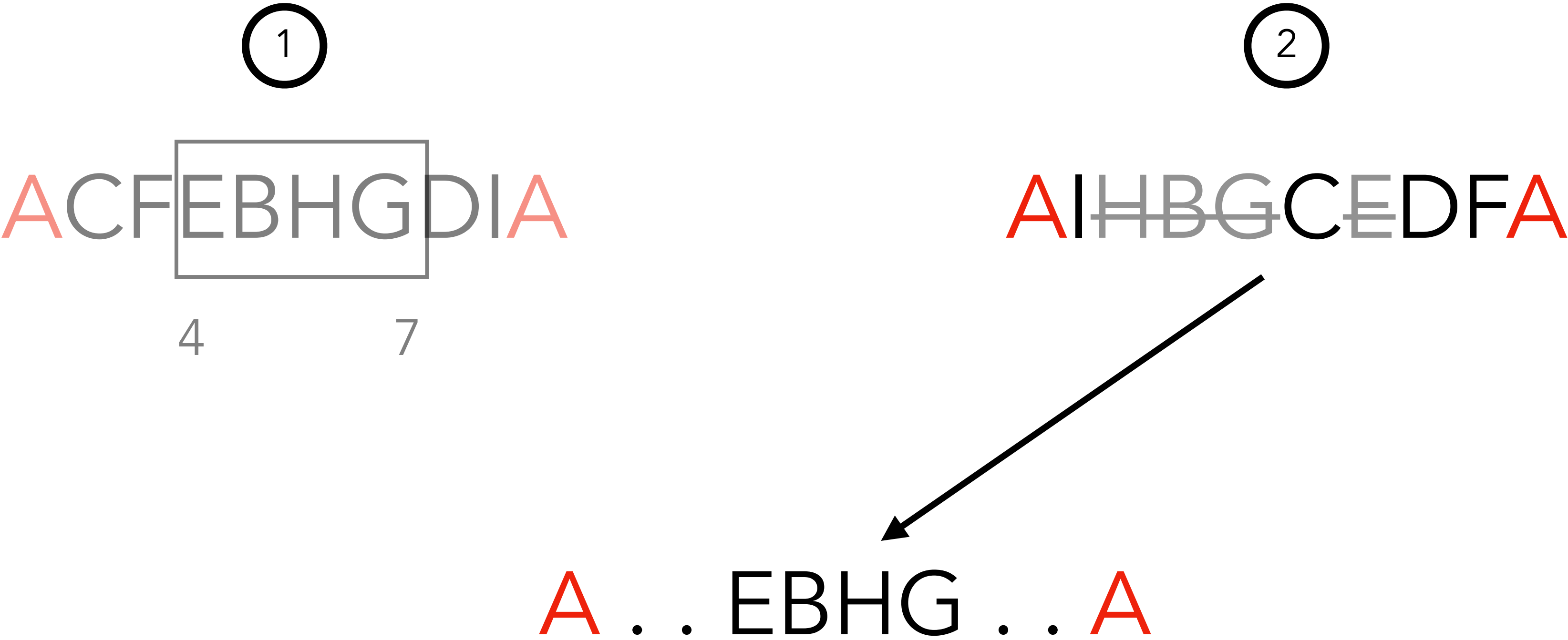
Cross two individuals



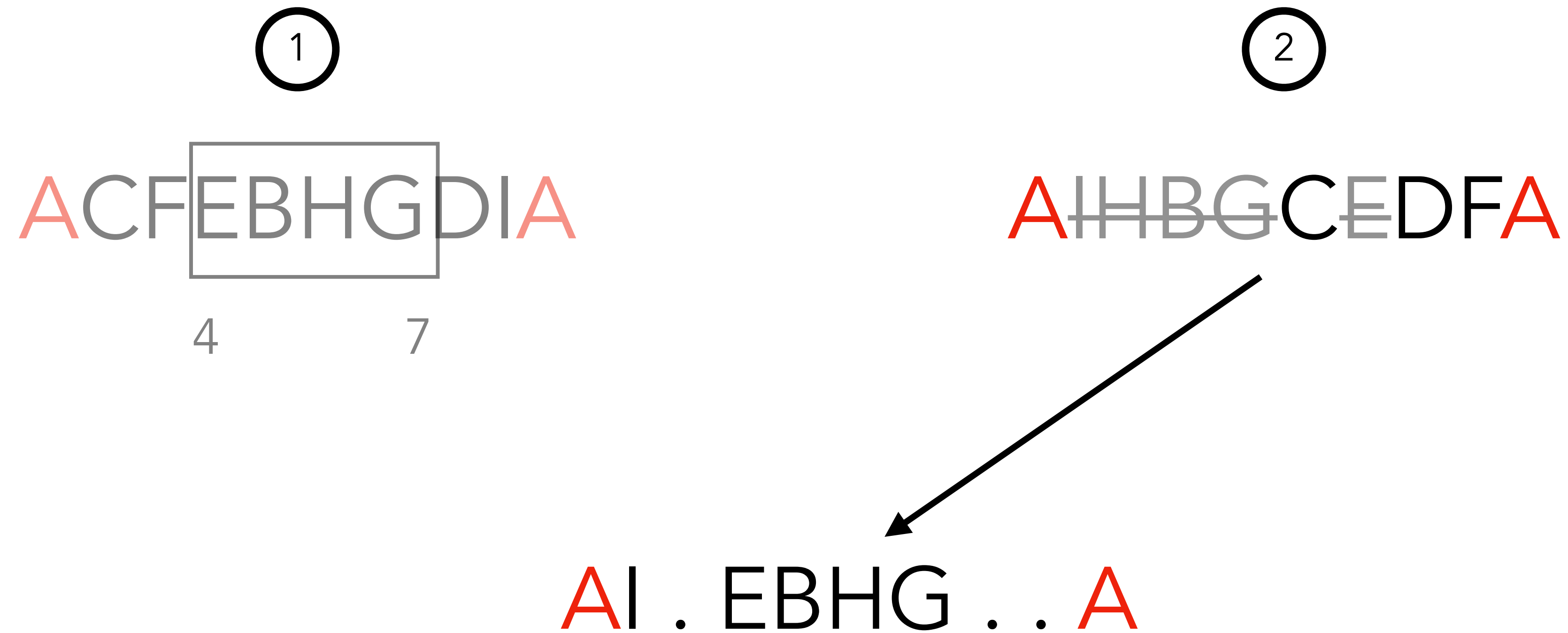
Cross two individuals



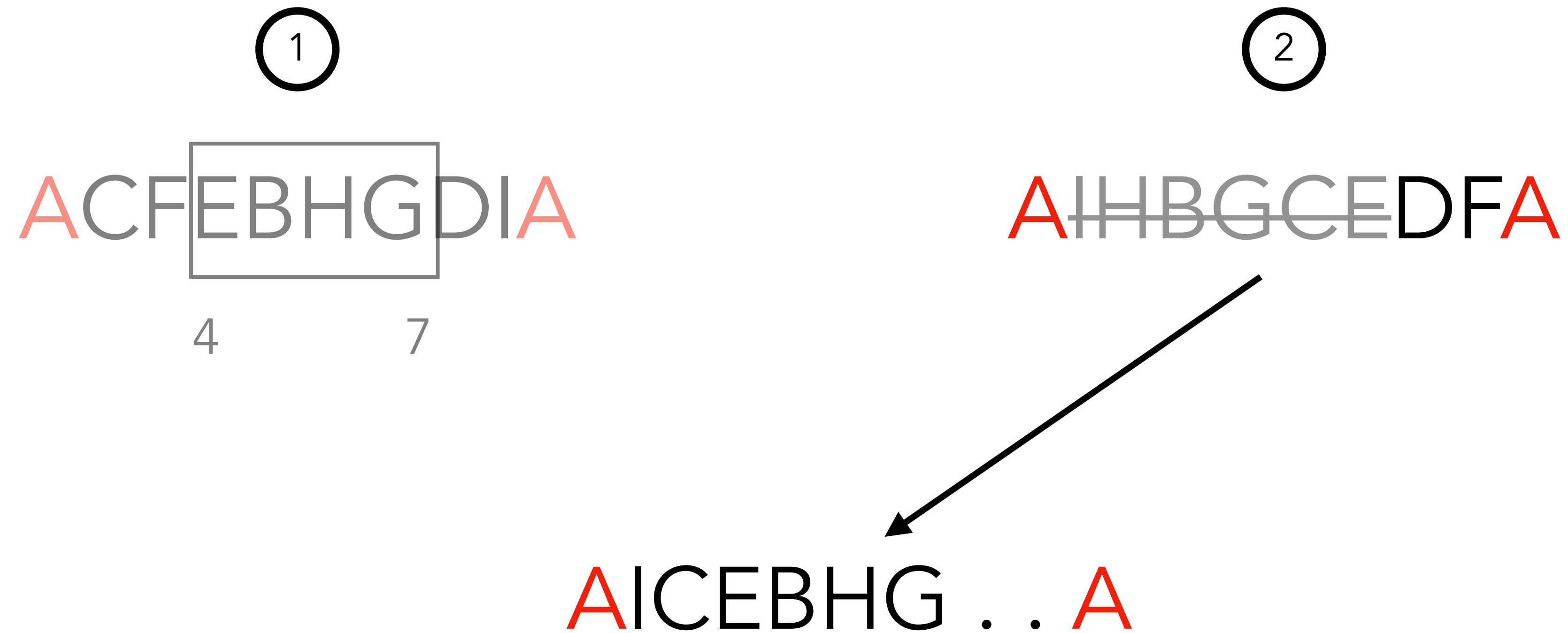
Cross two individuals



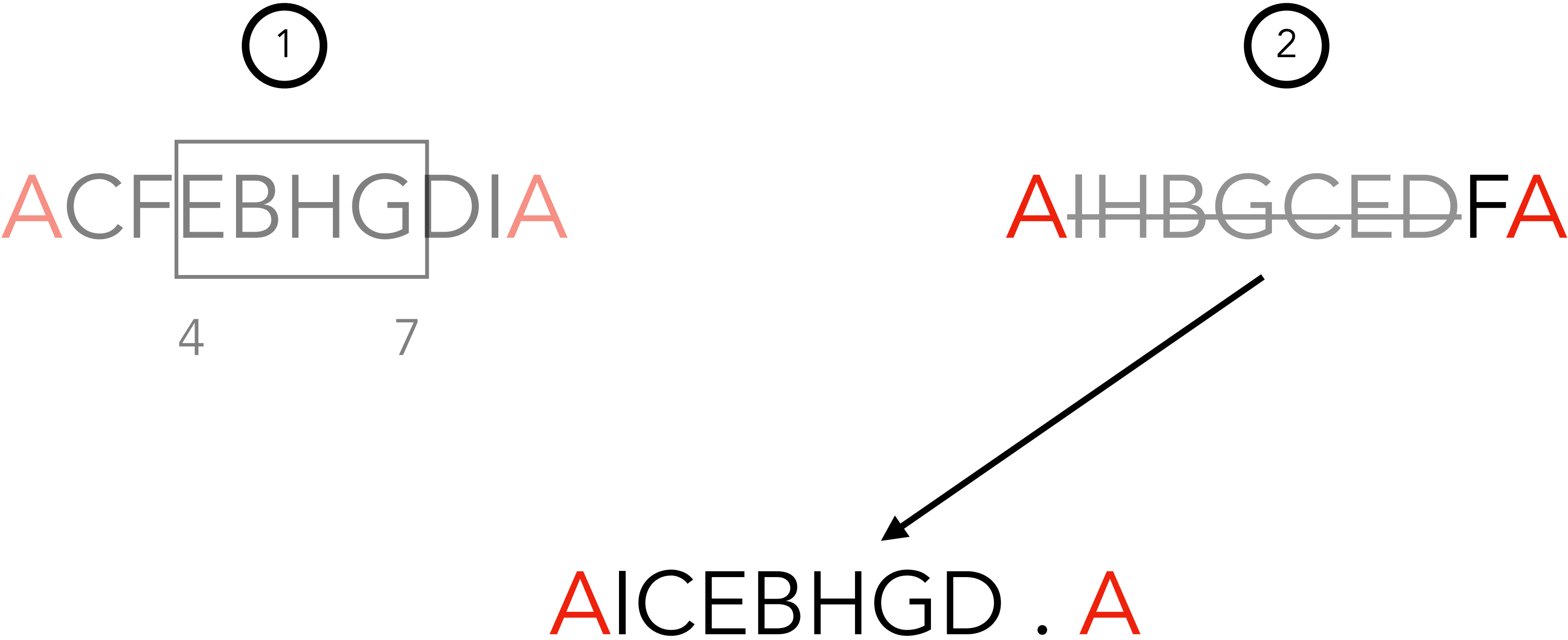
Cross two individuals



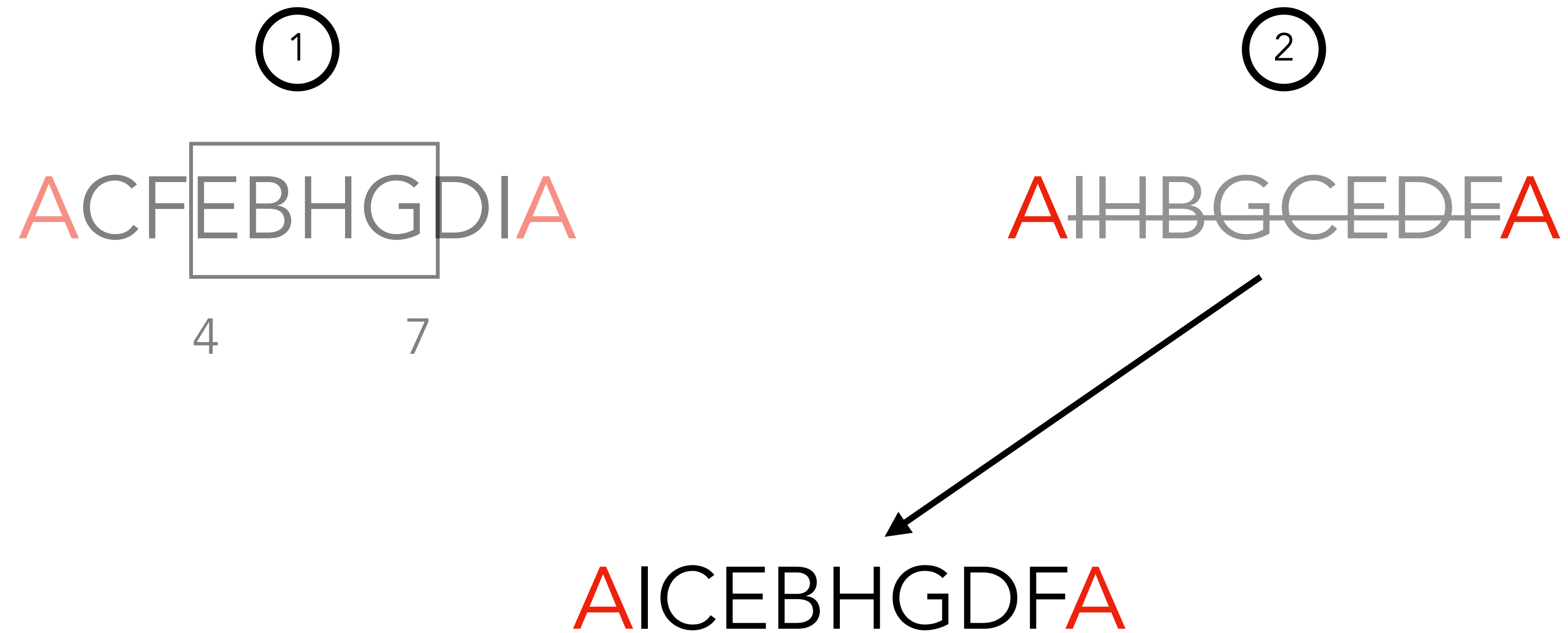
Cross two individuals



Cross two individuals

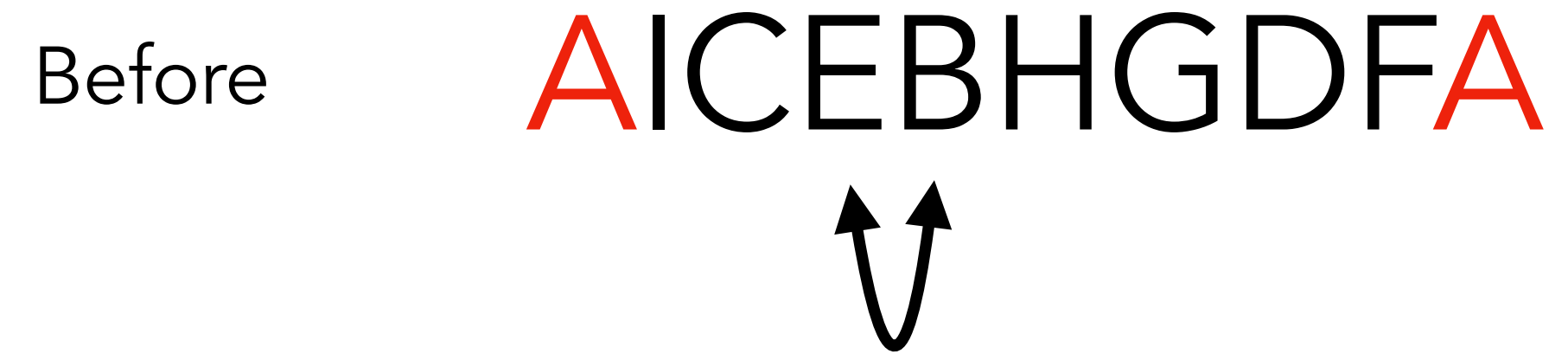


Cross two individuals



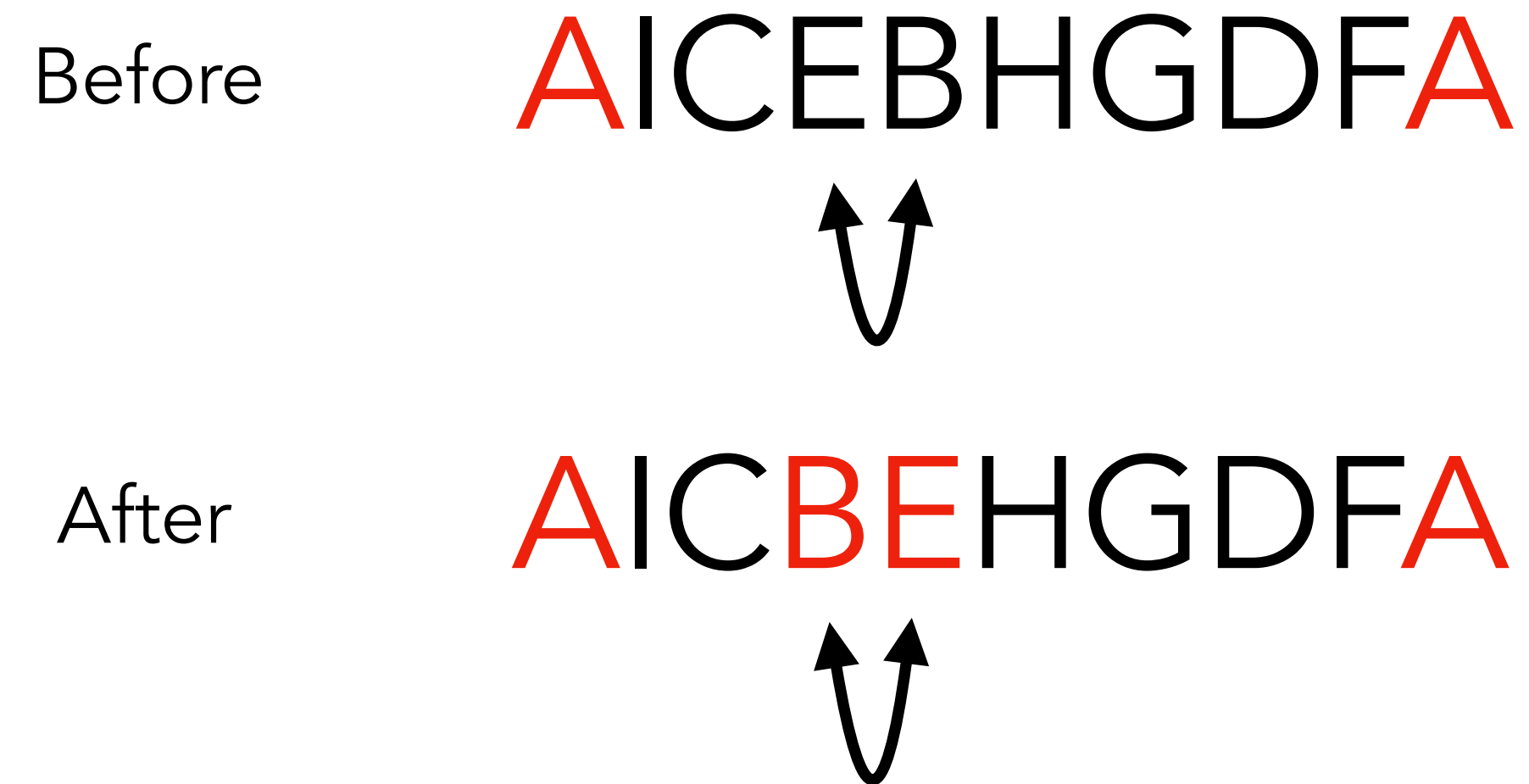
Mutating one individual

Before **A**ICEBHGD**F**A



We randomly swap two cities

Mutating one individual



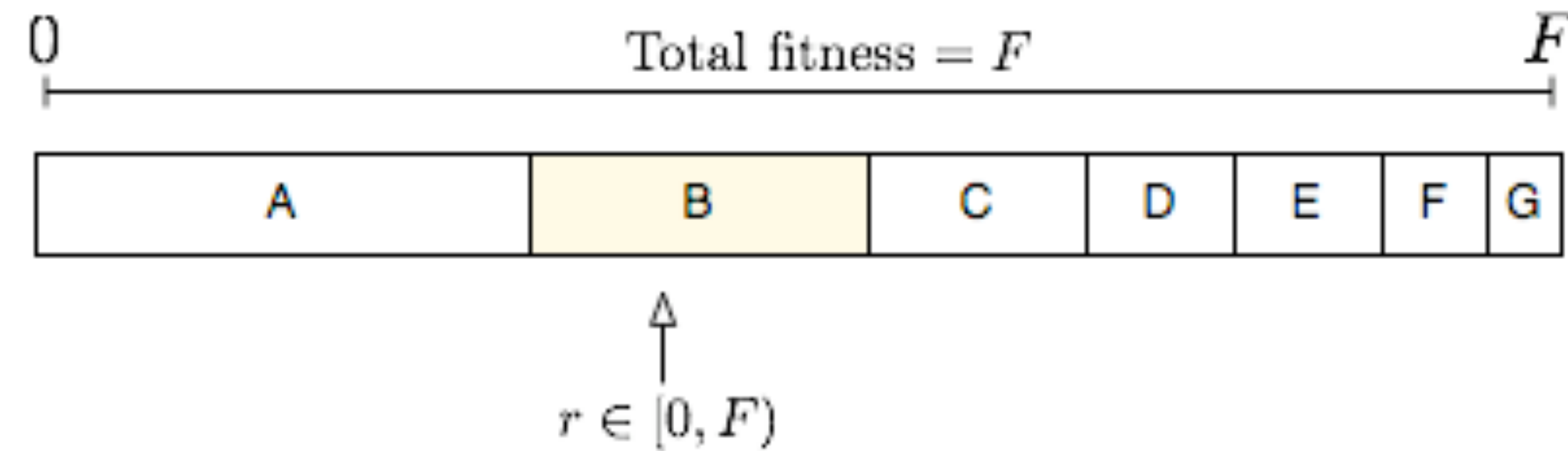
We randomly swap two cities

Parent selection

- The individuals with a good fitness function should have a higher probability to reproduce.
- The individuals with a low fitness function should have a lower probability.
- The selection of the individuals must be done randomly but respecting the probabilities.

How do we implement it?

Roulette wheel selection



Source: wikipedia.com

Fitness calculation

$$\text{fitness} = \text{maxDistance} - \text{totalDistance}$$

Fitness calculation

To not have 0

$$\text{fitness} = \text{maxDistance} + 2 - \text{totalDistance}$$

Reduce the population

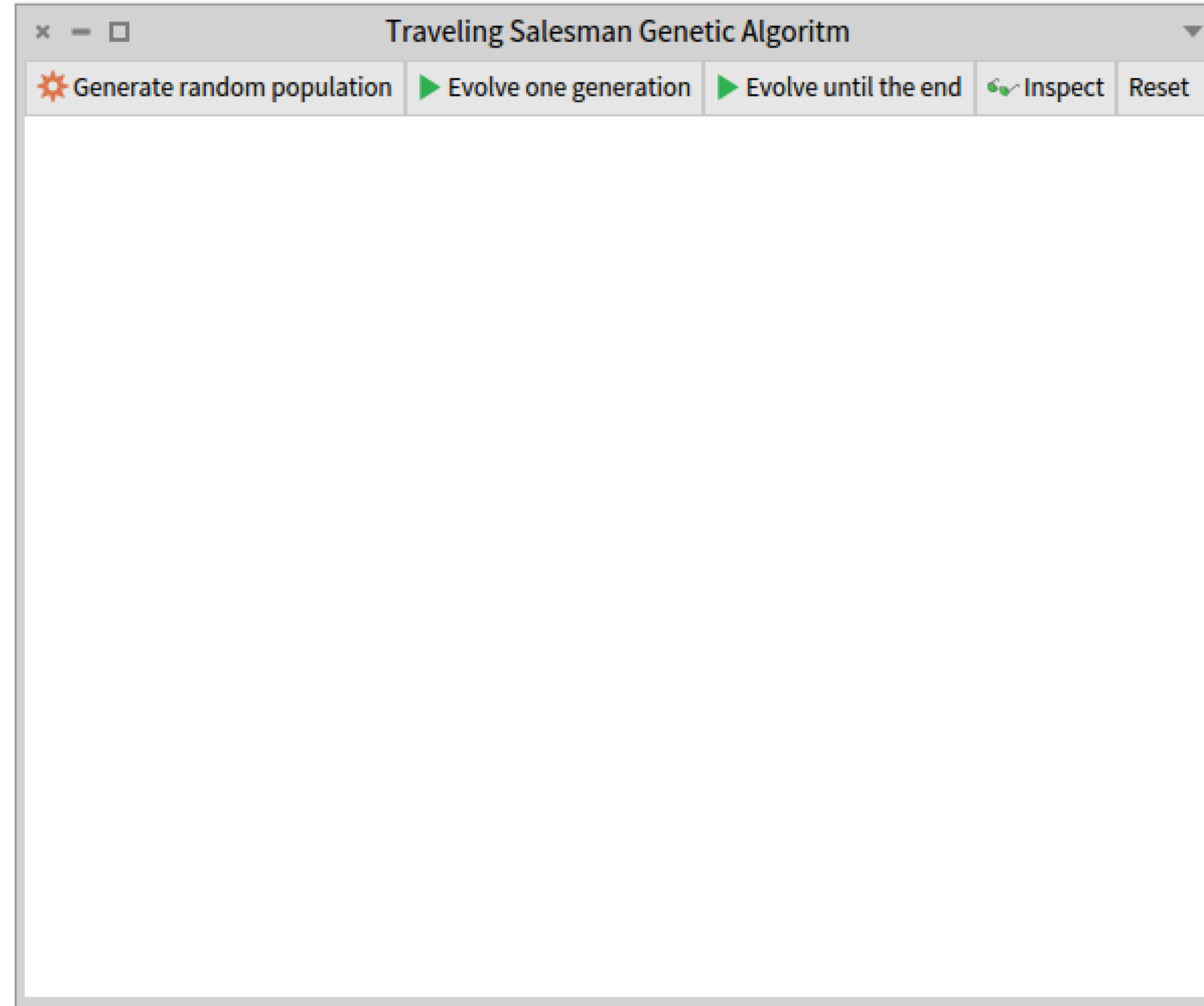
Basically, selecting the n individuals with the better fitness value

Terminating the algorithm

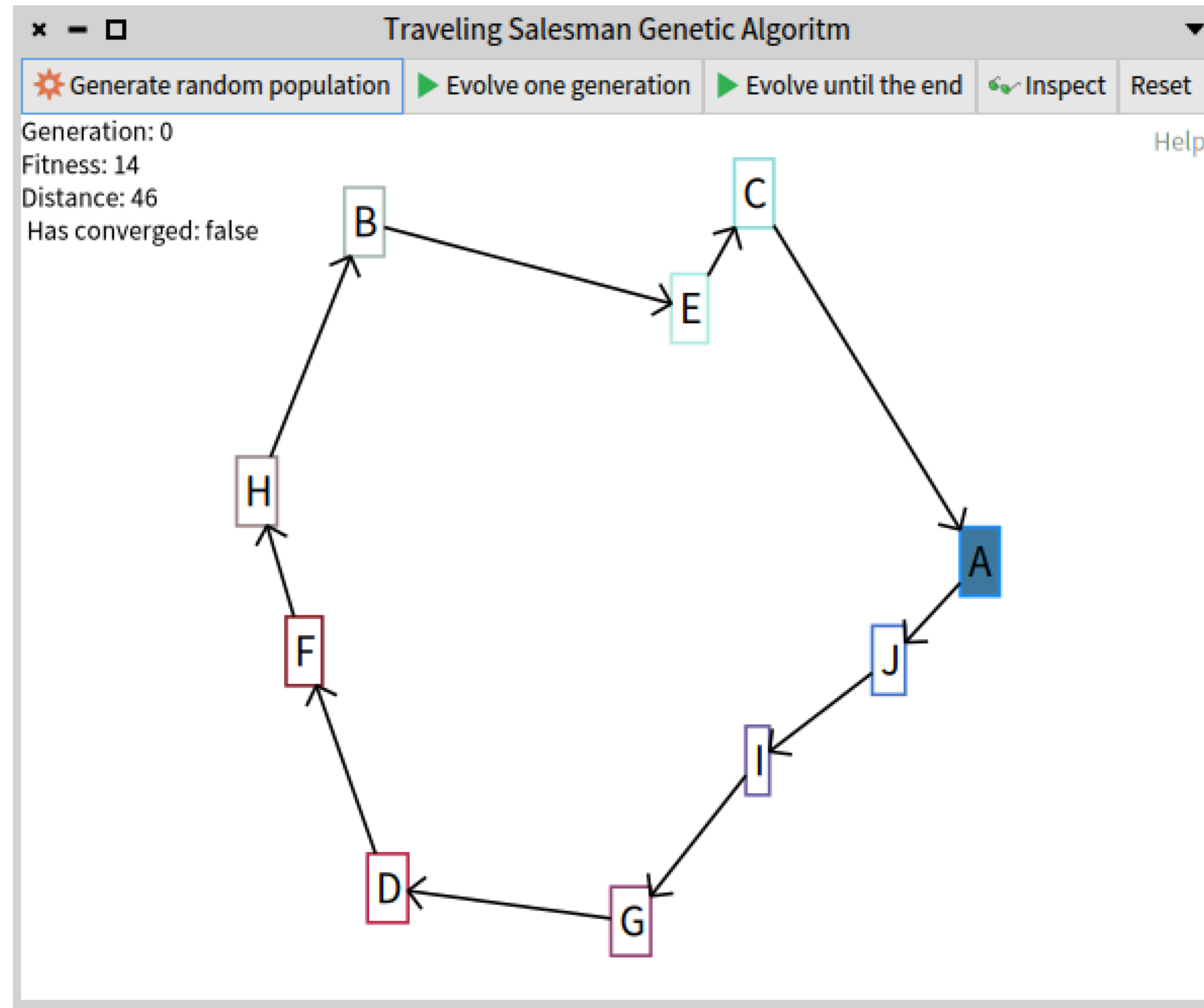
The algorithm has converged into a solution when:

n% of the individuals have the same fitness value (e.g. 90%)

Making a little application



Generate random population



For the visualization

Use the class:

`ChromosomeVisualizerPresenter`

For example:

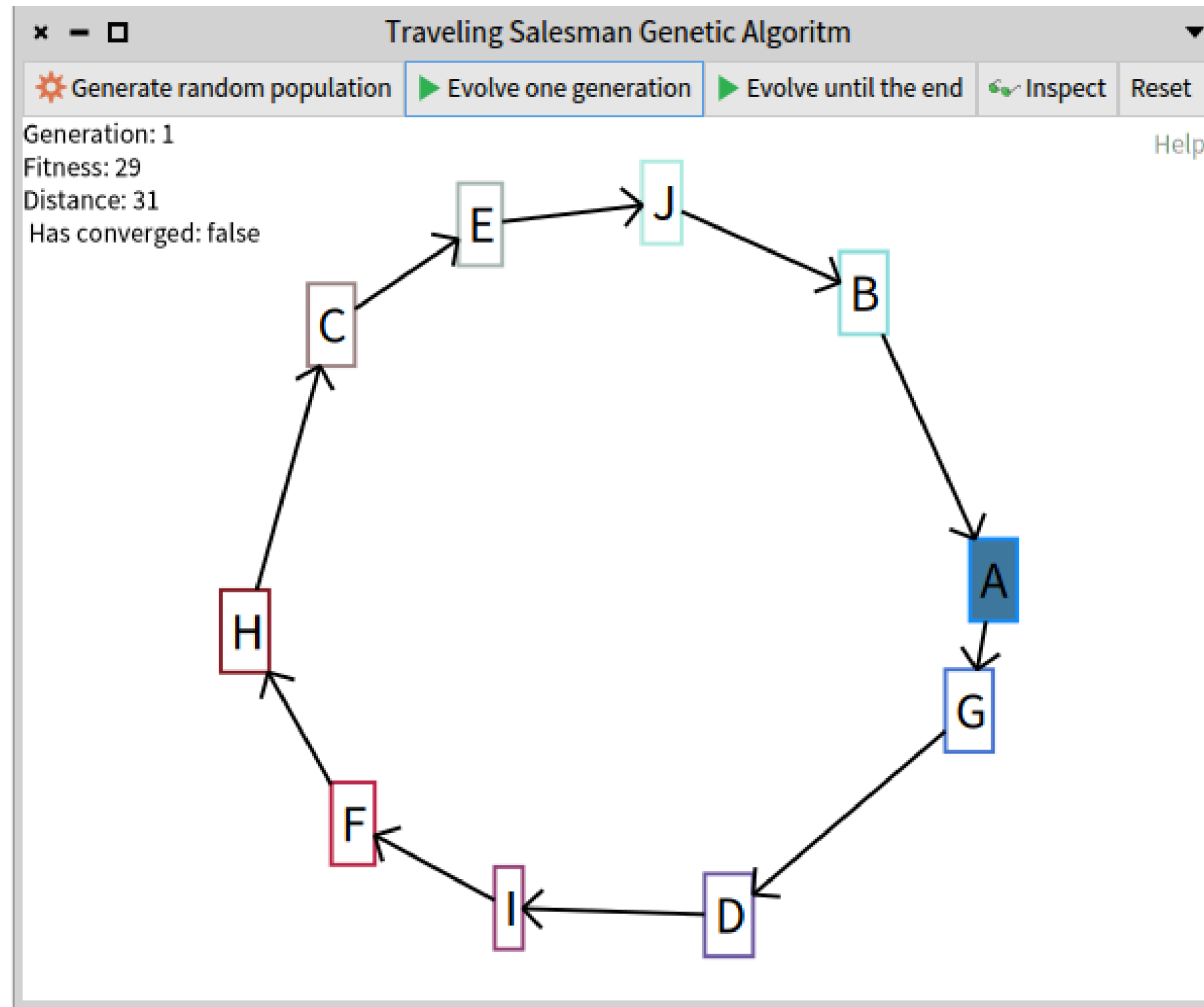
```
chromosomeVisualizerPresenter
```

```
    geneticAlgorithm: geneticAlgorithm;
```

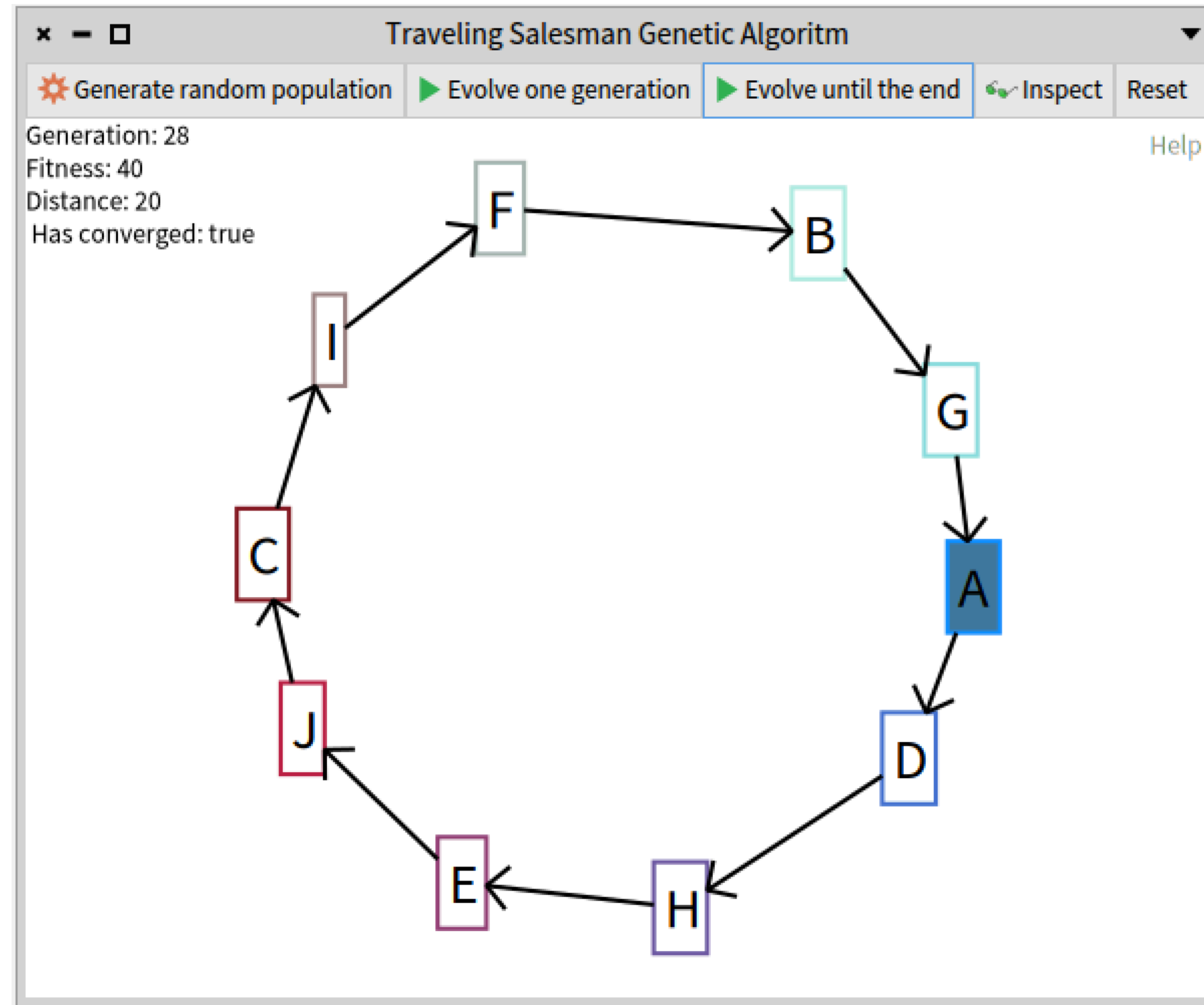
```
    individual: geneticAlgorithm population atRandom;
```

```
    refresh
```

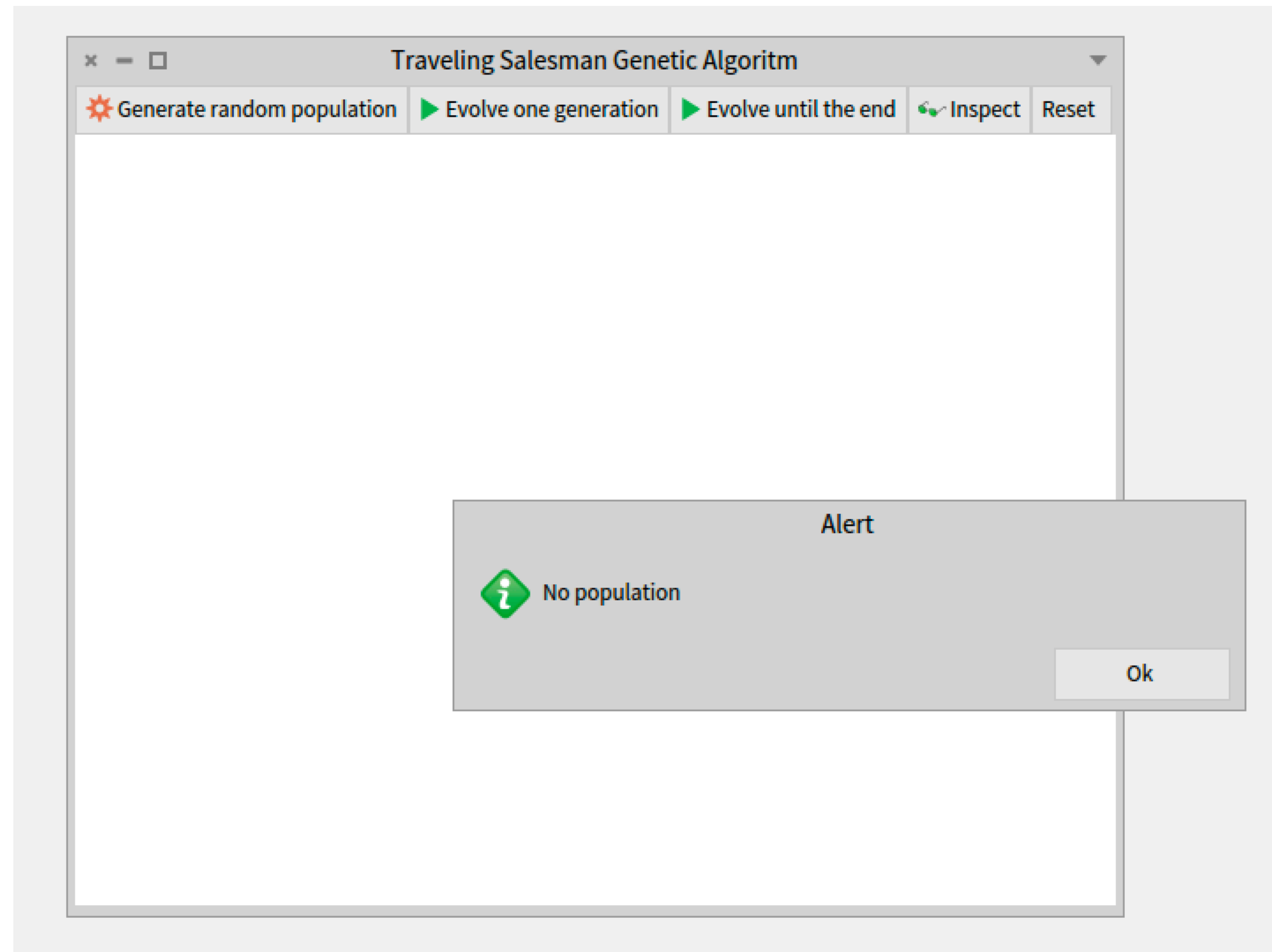
Evolve one generation



Evolve until the end



Alert then trying to evolve without a population



For the modal

Look at the class:

`SpInformPresenter.`

You can instantiate it inside the presenter with:

```
self application newInform
```

The reset and the inspect button

