Zürcher Hochschule
für Angewandte Wissenschaften
School of
Engineering
IAMP Institut für Angewandte
Mathematik und Physik

MSE Algorithms

L06: Genetic Algorithms, Work on Santas Challenge

Samuel Beer

Course Overview

Zürcher Hochschule
für Angewandte Wissenschaften
School of
Engineering
IAMP Institut für Angewandte
Mathematik und Physik

1. Introduction: Basic problems and algorithms
2. Constructive methods: Random building, Greedy
3. Local searches
4. Randomized methods
5. Threshold accepting, Simulated Annealing
6. Decomposition methods: Large neighborhood search
7. Learning methods for solution improvement: Tabu search
8. Application: Santa's Challenge
9. Learning methods for solution building: Artificial ant systems
10. **Methods with a population of solutions: Genetic algorithms**
11. **Work on Santa's Challenge**
12. Final Lecture

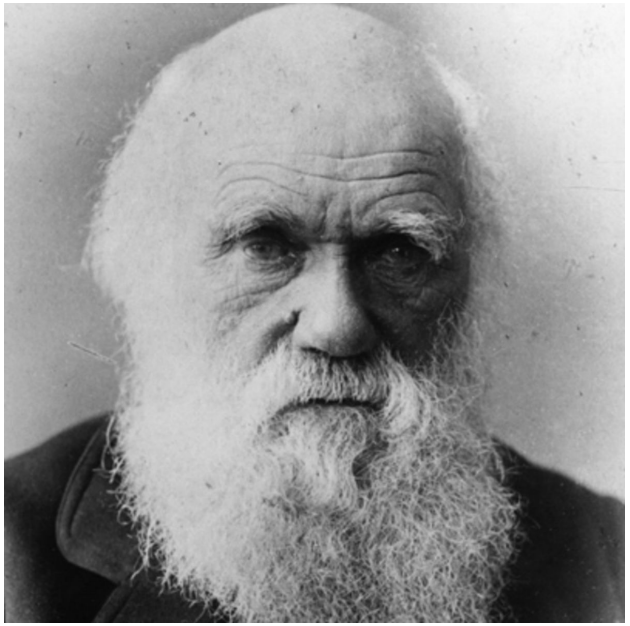
Genetic Algorithms

These slides are based on a course by Prof. Moshe Sipper: https://www.moshesipper.com/

Genetic Algorihtms are used to **improve** solutions

Introduction to Genetic Algorithms

- Inspired by the biological evolution process
- Uses concepts of “Natural Selection” and “Genetic Inheritance” (Darwin 1859)
- Core Idea: "Survival of the Fittest"
- Originally developed by John Holland (1975)
- Subclass of Evolutionary Algorithms, together with Genetic Programming



Core Idea

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.**

Operators:

1. **Selection** replicates the most successful solutions (called individuals in this context) found in a population at a probability rate proportional to their relative quality determined by an appropriate fitness function.
2. **Recombination (cross-over)** decomposes two distinct solutions (parents) and then randomly mixes their parts (genes) to form novel solutions (children or offspring).
3. **Mutation** randomly perturbs a candidate individual.
4. **Replacement** replaces part of the population with modified individuals.

Basic Genetic Algorithm

produce an initial population of individuals

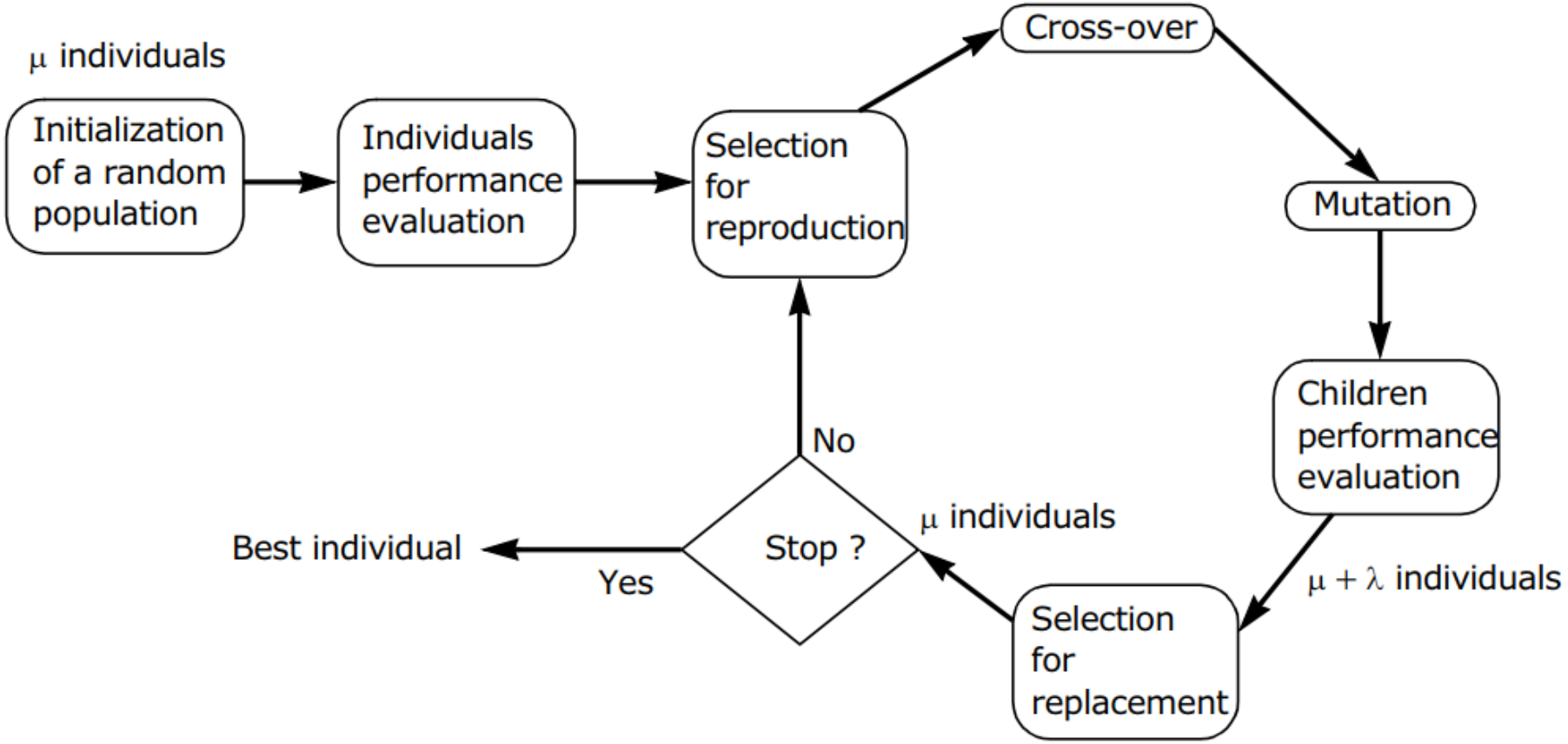
evaluate the fitness of all individuals

while termination condition not met **do**

- Select fitter individuals for reproduction
- Recombine between individuals (cross-over)
- Mutate individuals
- Evaluate fitness of the modified individuals
- Generate a new population

End while

Working Scheme



Example: MAXONE

Example: The MAXONE-Problem

Goal: Given a binary string of l digits, maximize the number of Ones ("1") in the string

- Trivial example, but good to show basic principles of Genetic Algorithms.
- Can be easily modified to maximize an arbitrary complex function.

- An individual is encoded (naturally) as a string of l binary digits.
- The fitness f of a candidate solution to the MAXONE problem is the number of ones in its genetic code.

MAXONE: Initialization

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

- We start with a population of n random strings. Suppose that $l = 10$ and $n = 6$
- We toss a fair coin 60 times and get the following initial population:

$s_1 = 1111010101$ $\quad f(s_1) = 7$

$s_2 = 0111000101$ $\quad f(s_2) = 5$

$s_3 = 1110110101$ $\quad f(s_3) = 7$

$s_4 = 0100010011$ $\quad f(s_4) = 4$

$s_5 = 1110111101$ $\quad f(s_5) = 8$

$s_6 = 0100110000$ $\quad f(s_6) = 3$

MAXONE: Selection

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

- Randomly select *n* individuals with probability proportional to their fitness
- Solutions can be selected more than once

- Example: after performing selection, we get the following population (again with 6 individuals):

s_1` = 1111010101 (s_1)

s_2` = 1110110101 (s_3)

s_3` = 1110111101 (s_5)

s_4` = 0111000101 (s_2)

s_5` = 0100010011 (s_4)

s_6` = 1110111101 (s_5)

MAXONE: Crossover

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

- Next we mate strings for **crossover**. For each couple we decide according to crossover probability (for instance 0.6) whether to actually perform crossover or not.
- Suppose that we decide to actually perform crossover only for couples (s_1`, $s_2$`) and (s_5`, $s_6$`).
- For each couple, we randomly extract a crossover point, for instance 2 for the first and 5 for the second couple.
- We then swap the sections **behind** the crossover point for each couple:
- Before crossover

s_1` = 11111010101 $s_5$` = 01000010011

s_2` = 11110110101 $s_6$` = 11101111101

- After crossover

s_1`` = 11110110101 $s_5$`` = 01000011101

s_2`` = 11111010101 $s_6$`` = 11101110011

MAXONE: Mutation

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

- Next we apply **random mutation**: for each bit that we are to copy to the new population we allow a small probability of error (for instance 0.1)

Before mutation:	After applying mutation:	Fitness:
$s_1'' = 1110110101$	$s_1''' = 1110100101$	$f(s_1''') = 6$
$s_2'' = 1111010101$	$s_2''' = 1111110100$	$f(s_2''') = 7$
$s_3'' = 1110111101$	$s_3''' = 1110101111$	$f(s_3''') = 8$
$s_4'' = 0111000101$	$s_4''' = 0111000101$	$f(s_4''') = 5$
$s_5'' = 0100011101$	$s_5''' = 0100011101$	$f(s_5''') = 5$
$s_6'' = 1110110011$	$s_6''' = 1110110001$	$f(s_6''') = 6$

MAXONE: Mutation

- Finally in this example we set the new population to be s1’’’, …, s6’’’.

Before mutation:	After applying mutation:	Fitness:
$s_1'' = 1110110101$	$s_1''' = 1110100101$	$f(s_1''') = 6$
$s_2'' = 1111010101$	$s_2''' = 1111110100$	$f(s_2''') = 7$
$s_3'' = 1110111101$	$s_3''' = 1110101111$	$f(s_3''') = 8$
$s_4'' = 0111000101$	$s_4''' = 0111000101$	$f(s_4''') = 5$
$s_5'' = 0100011101$	$s_5''' = 0100011101$	$f(s_5''') = 5$
$s_6'' = 1110110011$	$s_6''' = 1110110001$	$f(s_6''') = 6$

MAXONE: Evaluation

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

- Finally in this example we set the new population to be $s_1''', \ldots, s_6'''$.
- In one generation, the total population fitness changed from 34 to 37, thus improving by ~9%
- From this point, we go through the same process all over again, until a stopping criterion is met

Components of a Genetic Algorithm

Components of a GA

A problem instance as input, and

- Encoding principles (genes, chromosomes)
- Initialization procedure (creation of a population)
- Selection of parents (reproduction)
- Genetic operators (cross-over, mutation)
- Evaluation function (fitness, environment)
- Generational Update (population evolvement)
- Termination condition

Representation (Encoding, Genotype)

Possible encoding of individuals:

- Bit strings (0101 ... 1100)
- Real numbers (43.2 -33.1 ... 0.0 89.2)
- Permutations of elements (E11 E3 E7 ... E1 E15)
- Lists of rules (R1 R2 R3 ... R22 R23)
- Program elements (genetic programming)
- Any data structure…

Representation (continued)

When choosing an encoding method, rely on the following key ideas:

- Use a data structure as close as possible to the natural representation
- Define appropriate genetic operators as needed
- If possible, ensure that all genotypes correspond to feasible solutions
- If possible, ensure that genetic operators preserve feasibility

Initialization

Start with a population *n* individuals.

Potential sources:

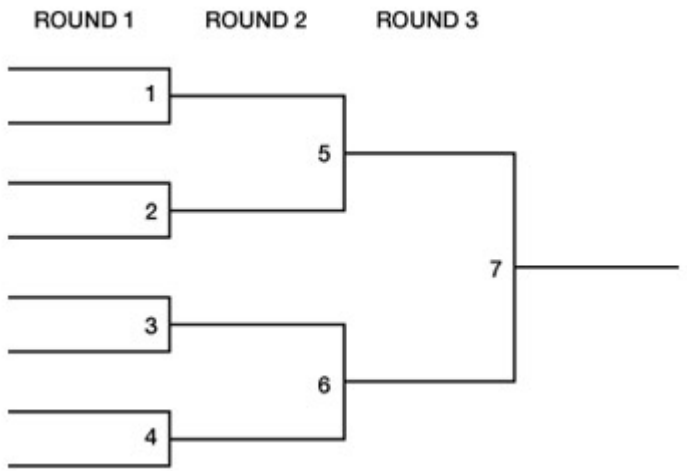
- Randomly generated individuals
- A previously saved population
- A set of solutions provided by human experts
- A set of solutions provided by another heuristic algorithm

Selection of parents

- Purpose: Focus the search to promising regions of the solution space
- Inspiration: Darwin's "survival of the fittest"
- Trade-off between *exploration* and *exploitation* of the search space
- A higher *selective pressure* will generally result in the population converging faster towards a single solution (i.e. all individuals look the same), but that this solution might not be very good.

Local Tournament Selection

- Extract *k* individuals from the population with uniform probability (without re-insertion) and make them play a “tournament”, where the probability for an individual to win is generally proportional to its fitness
- Selection pressure is directly proportional to the number *k* of participants

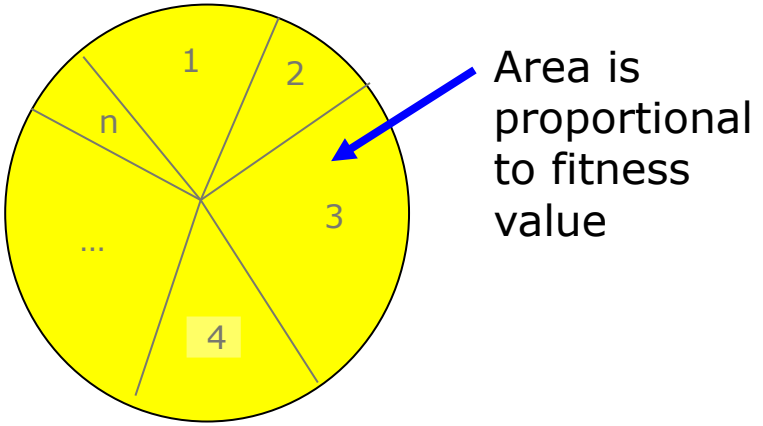


Selection with Roulette Wheel Method

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

Idea

- Size of area in a "Roulette Wheel" is proportional to the fitness value of each solution (also called "Proportional Selection")
- Selects randomly 1 out of n solutions per spin
- Each individual *i* has probability $\frac{f(i)}{\sum_j f(j)}$ to be chosen
- Individuals with high fitness will occur often but must not necessarily be chosen



Random choice in [1, 36]

i	1	2	3	4	5	6
f(i)	10	7	2	5	4	8

Selection with Roulette Wheel Method

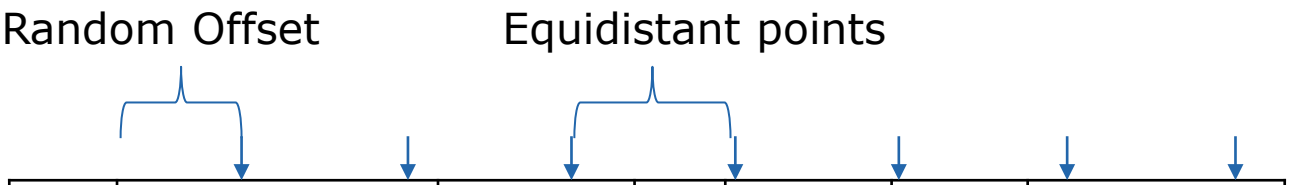
Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

```
rouletteSelection()
    totalScore = 0
    for i in population
        totalScore += f(i)

    rnd = uniformRandom(0, 1) * totalScore

    runningScore = 0
    for i in population
        if rnd >= runningScore AND rnd < runningScore + f(i)
            return i
        runningScore += f(i)
```

Other Selection Methods

Stochastic Universal Sampling

- Similar to Roulette Wheel, but *m* **equidistant** points are selected
- Uses a random offset at the beginning



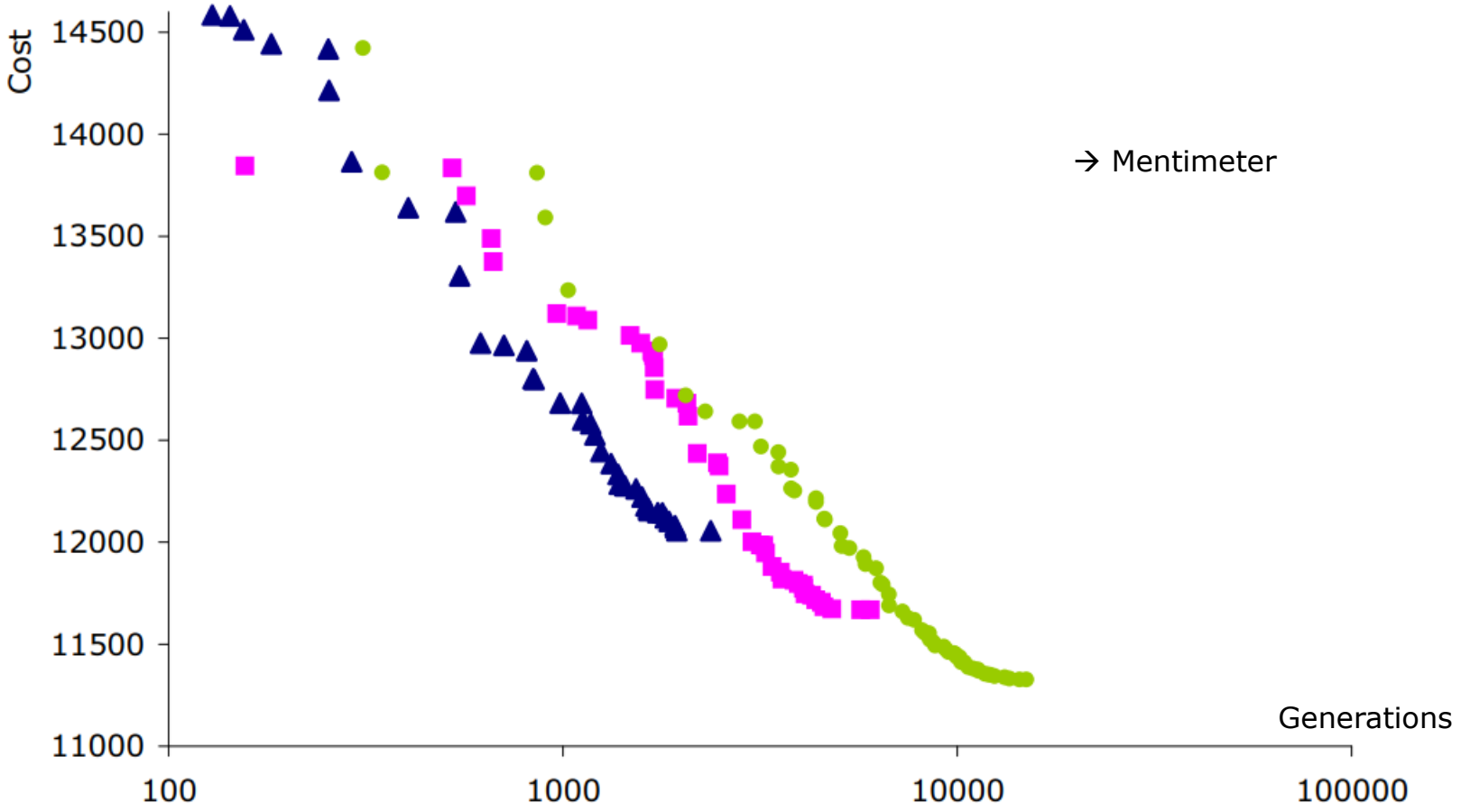
i	1	2	3	4	5	6
f(i)	10	7	2	5	4	8

Rank

- Sort all individuals according to their fitness
- The *rank* of a solution is its position in the sorted list, i.e. fittest solution has rank *n*, next has rank *n-1*.. and least fittest solution has rank *1*
- Apply Roulette Wheel selection using the *rank* instead of *fitness*

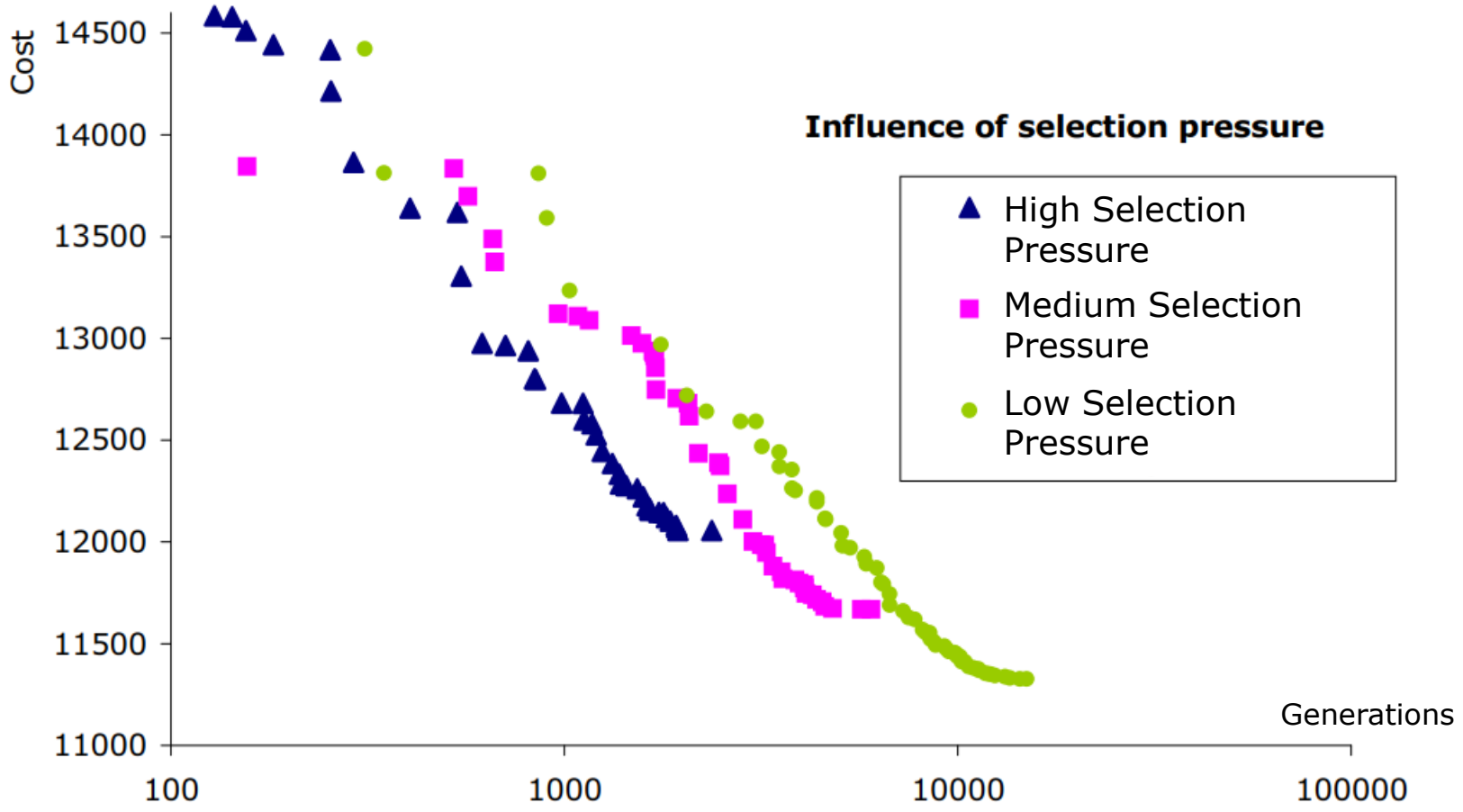
Question: Influence of Selection Pressure?

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik



→ Mentimeter

Solution: Influence of Selection Pressure



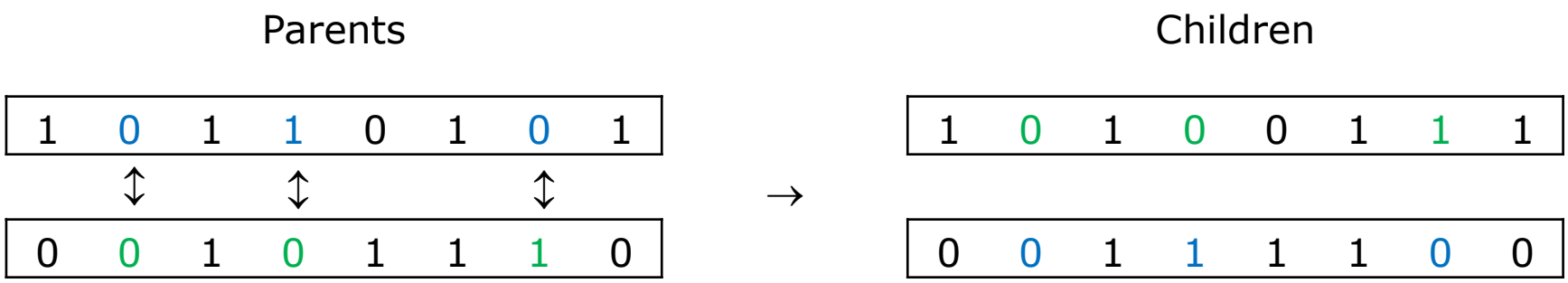
Recombination (=Crossover)

- Enables the evolutionary process to move toward promising regions of the search space
- Matches good parents' sub-solutions to construct better offspring
- Idea: Randomly choose components of the parents and combine them to new solutions (offspring)

Crossover Operation in 0-1-Vectors

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

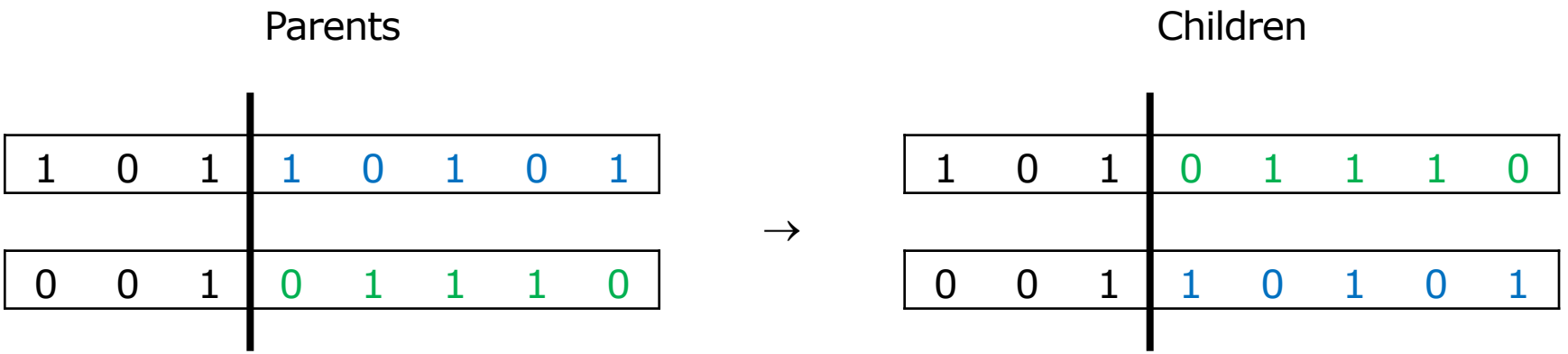
Random Bit Exchanges



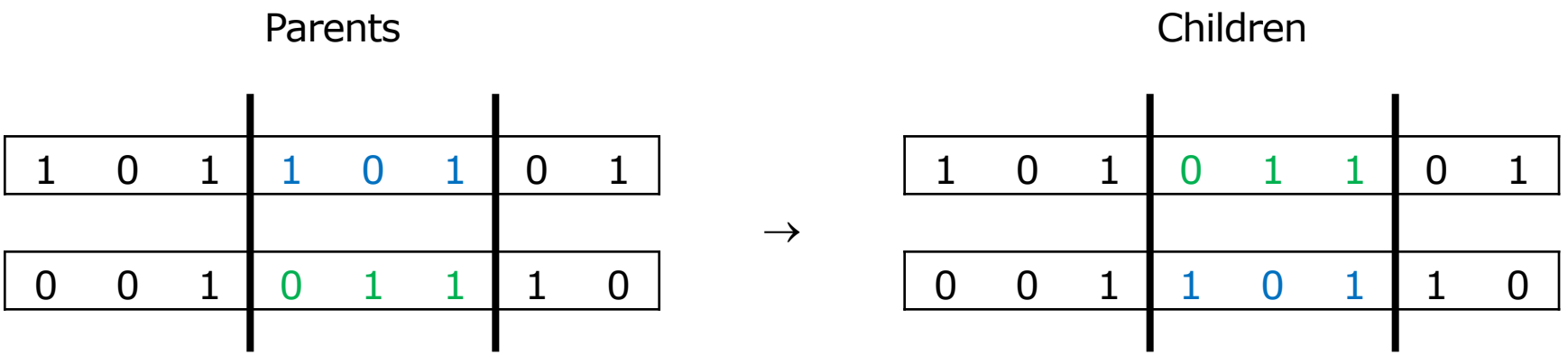
Crossover Operation in 0-1-Vectors

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

1-Point Exchange



2-Point Exchange



Crossover in Permutations

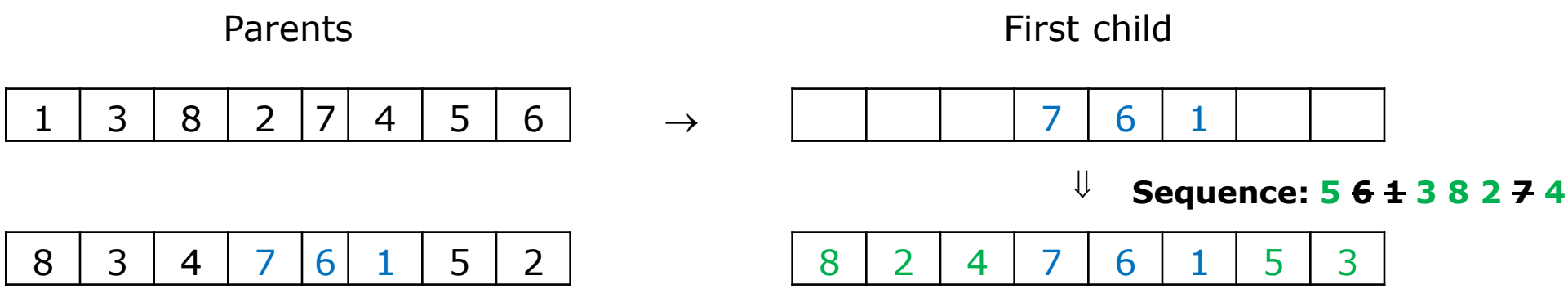
Crossover operations for bit vectors are easy to define. For permutations, the situation is not as simple, because it is not trivial to guarantee that the result is a valid permutation.

How can crossover operations be defined for permutations?

Crossover in Permutations

Order-1 Crossover (OX)

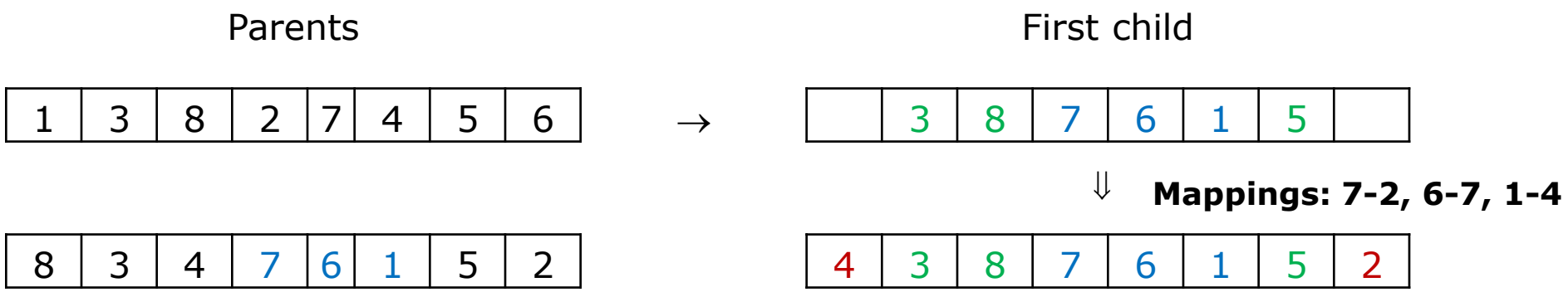
- Select range in both parents
- For first child (second child analogously):
  - Insert range from second parent
  - Enumerate all elements from first parent in a sequence, starting to the RIGHT of the range and skipping elements that are already in the child
  - Insert the missing elements, starting to the RIGHT of the range -> keeps ordering as in first parent



Crossover in Permutations

Partially Mapped Crossover (PMX)

- Select range in both parents
- For first child (second child analogously):
  - Insert range from second parent (blue)
  - Copy all elements from first parent that do not have a conflict (i.e. do not appear in range) (green)
  - Create mapping within the range from second to first parent
  - Add remaining elements accoding to the mapping (red)



Mutation

Purpose:

- To simulate the effect of errors that happen with low probability during reproduction

Effects:

- Movement in the search space
- Restoration of "lost" information to the population

Realization:

- 0 – 1 vectors: Modify each bit randomly, with a given probability, or modify a given number of bits randomly
- Other cases: apply random "moves" as known from local search (e.g. 2-opt for TSP)

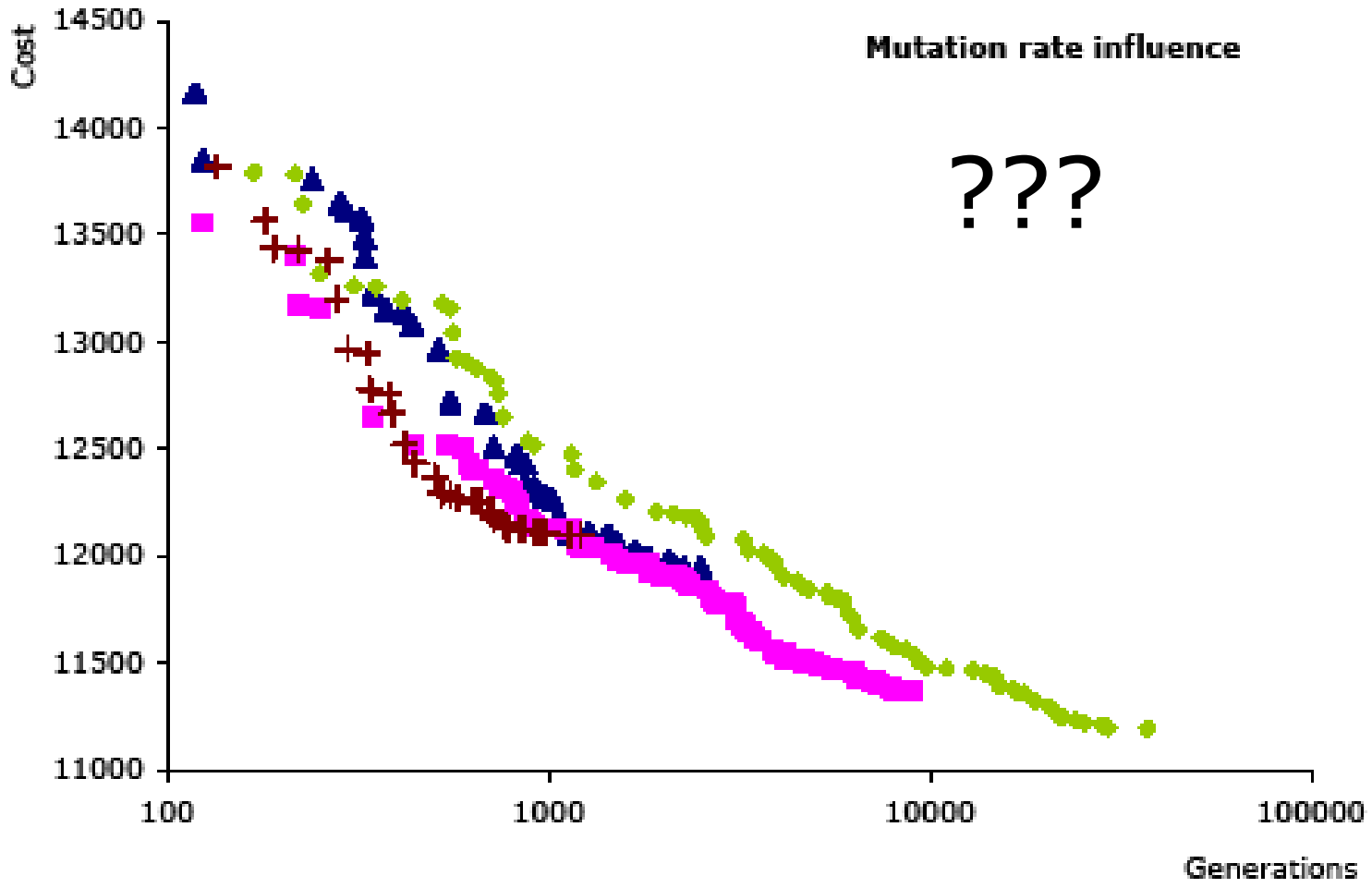
Mutation Operation in Permutations

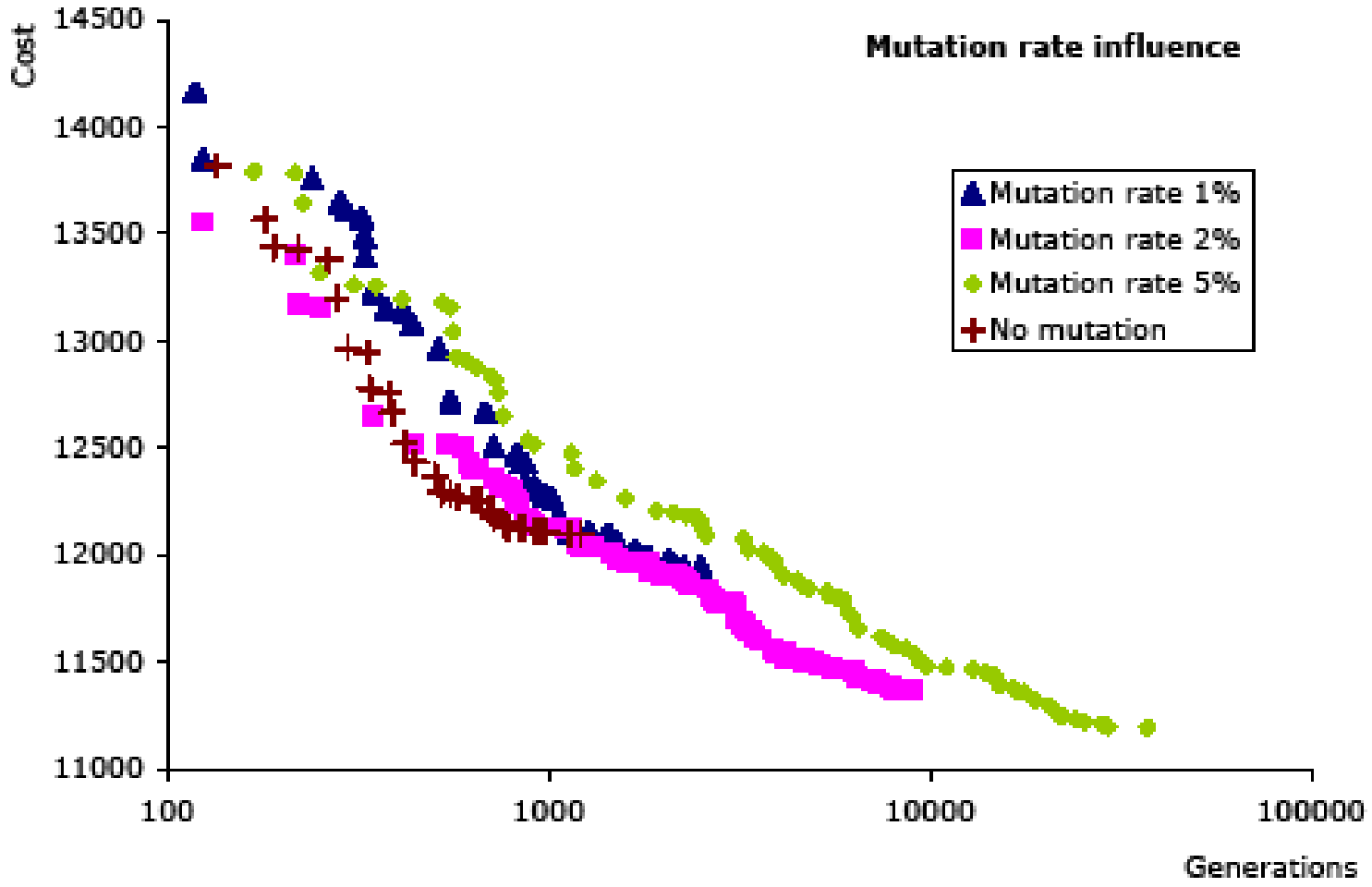
Path Inversion (2-opt)

- The sub-string between two randomly selected points in the path is reversed (this corresponds to a 2-opt move)

- Example: (1 2 3 4 5 6 7 8 9) is changed into (1 2 7 6 5 4 3 8 9)

- Such simple mutations guarantee that the resulting offspring is a legal permutation


Mutation rate influence
???
Cost
14500
14000
13500
13000
12500
12000
11500
11000
100
1000
10000
100000
Generations


Mutation rate influence
Cost
14500
14000
13500
13000
12500
12000
11500
11000
100
1000
10000
100000
Generations
Mutation rate 1%
Mutation rate 2%
Mutation rate 5%
No mutation

Evaluation (Fitness Function)

- Genetic Algorithms are only as good as the evaluation function; choosing a good (=promising) one is often the hardest part
- Problem-specific
- Similar-encoded solutions should have a similar fitness

Generational Update: Examples of Strategies

Zürcher Hochschule
für Angewandte Wissenschaften
School of
Engineering
IAMP Institut für Angewandte
Mathematik und Physik

Let μ be the current population size, and λ the number of offsprings generated during replication phase

Generational replacement

Only children are selected from one population to the next one (parents are removed)

($\lambda = \mu$)

(μ, λ)-Evolutionary Strategy

Only the μ best children are kept ($\lambda > \mu$)

Steady state replacement

Replace λ parents by children (λ small, typically 1 or 2)

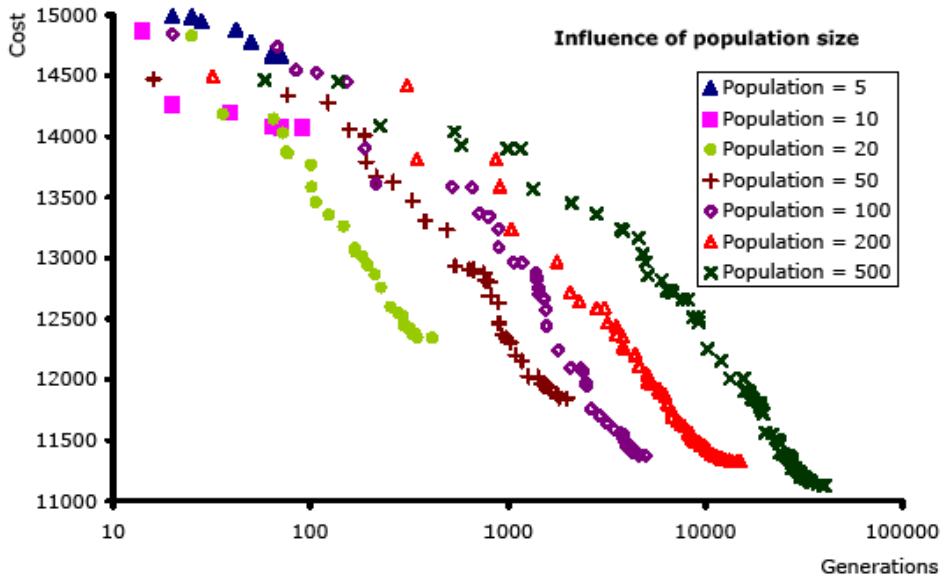
Elitist strategy

From the complete population of $\mu + \lambda$ individuals, keep only the μ best

Influence of Population Size

A large population implies:

- … larger diversity
- … better solution obtained before convergence (on average)
- … lower convergence rate, larger computational times



Termination Condition

Examples:

- A pre-determined number of generations or time has elapsed
- A satisfactory solution has been achieved
- No improvement in solution quality has taken place for a pre-determined number of generations

Remarks

- The repetition of the loop *selection* — *cross-over* — *replacement* tends to make the population homogeneous
- Finally, the population contains multiple identical individuals – "the solution"

Slowing down the convergence speed

- Increase population size
- Lower selection pressure (for reproduction and replacement)
- Increase mutation rate

Difficulty of implementing a genetic algorithm

- Choice of the cross-over operator
- Choice of the mutation operator
- Choice of the fitness function
- Choice of parameters (population size etc.)

References

- C. Darwin. On the Origin of Species by Means of Natural Selection; or, the Preservation of flavored Races in the Struggle for Life. John Murray, London, 1859.
- W. D. Hillis. Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure. Artificial Life 2, vol 10, Addison-Wesley, 1991.
- J. H. Holland. Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, Michigan, 1975.
- Z. Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin, third edition, 1996.
- M. Sipper. Machine Nature: The Coming Age of Bio-Inspired Computing. McGraw-Hill, New-York, first edition, 2002.
- M. Tomassini. Evolutionary algorithms. In E. Sanchez and M. Tomassini, editors, Towards Evolvable Hardware, volume 1062 of Lecture Notes in Computer Science, pages 19-47. Springer-Verlag, Berlin, 1996.

Course Overview

Zürcher Hochschule
für Angewandte Wissenschaften
School of
Engineering
IAMP Institut für Angewandte
Mathematik und Physik

1. Introduction: Basic problems and algorithms
2. Constructive methods: Random building, Greedy
3. Local searches
4. Randomized methods
5. Threshold accepting, Simulated Annealing
6. Decomposition methods: Large neighborhood search
7. Learning methods for solution improvement: Tabu search
8. Application: Santa's Challenge
9. Learning methods for solution building: Artificial ant systems
10. Methods with a population of solutions: Genetic algorithms
11. Work on Santa's Challenge
12. Final Lecture