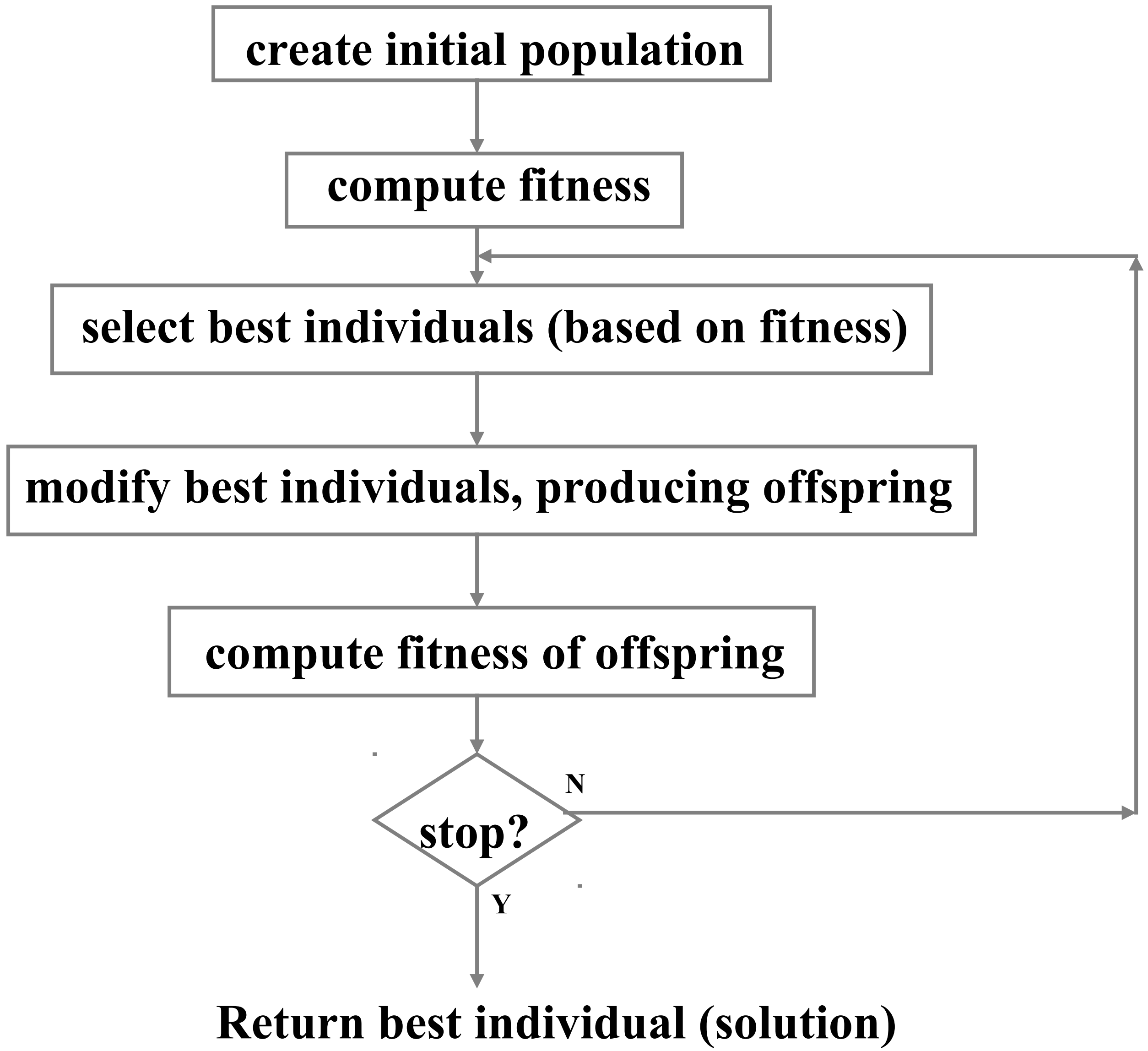
Introduction to Intelligent Systems CO528

An Introduction to Evolutionary Algorithms

Dominique Chu

Basic Flow Chart of EAs



Simple Genetic Algorithm (SGA) (1)

- **Clearly described in Goldberg's book [1989]**
- **Fixed-size population of individuals (solutions)**
- **Individual (or "chromosome") representation: binary string - e.g., 001101**
- **Proportionate selection, simulating a Roulette Wheel**
- **Simple genetic operators (to produce offspring):**
  - **Simple one-point crossover**
  - **Simple Mutation: flip a bit**

Simple Genetic Algorithm (2)

- **Basic Operators of a Simple GA (binary strings):**
  - **Single-point crossover & single-point mutation**

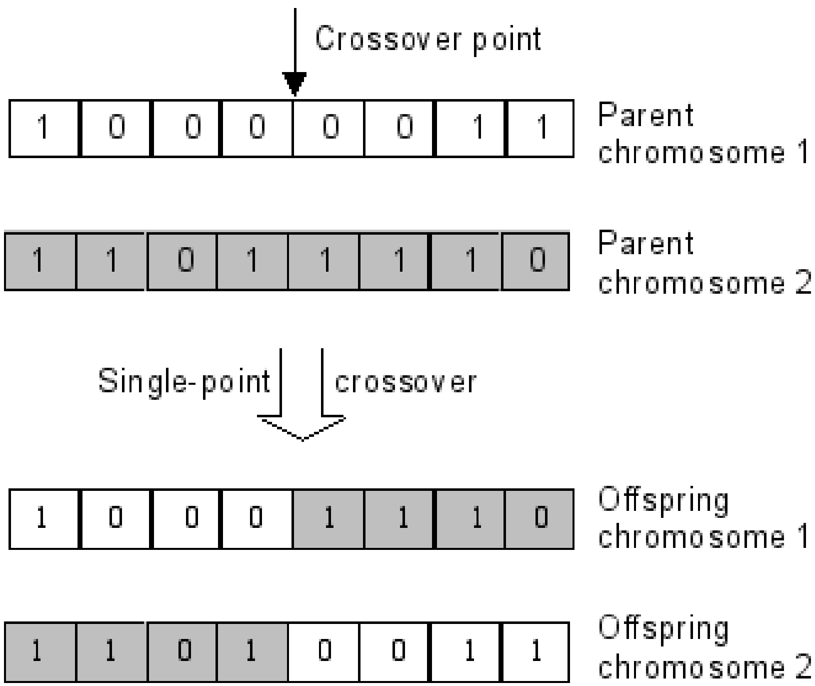


Fig. 1: Single-point crossover

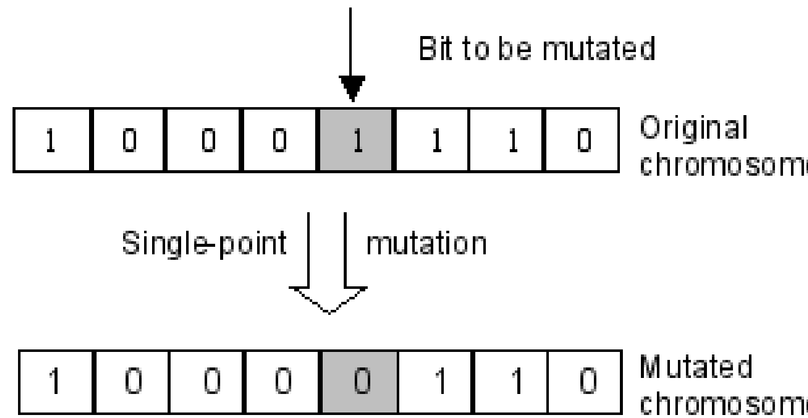


Fig. 2: Single-point mutation

A pedagogical example of SGA (1)

Goldberg's book [1989]

- "Toy" problem: finding the maximum value of the function x^2 in the interval [0..31]
- Individual encoding: five bits representing x in [0..31]
- Fitness function: x^2
  (the larger the fitness, the better the individual)
- Fitness evaluation: obtain x, decode the 5 bits (*genotype*), and then compute x^2 (*phenotype*)
- Initial population (randomly generated):
  0 1 1 0 1
  1 1 0 0 0
  0 1 0 0 0
  1 0 0 1 1

A pedagogical example of SGA (2)

- **Measuring fitness of each individual in the population:**

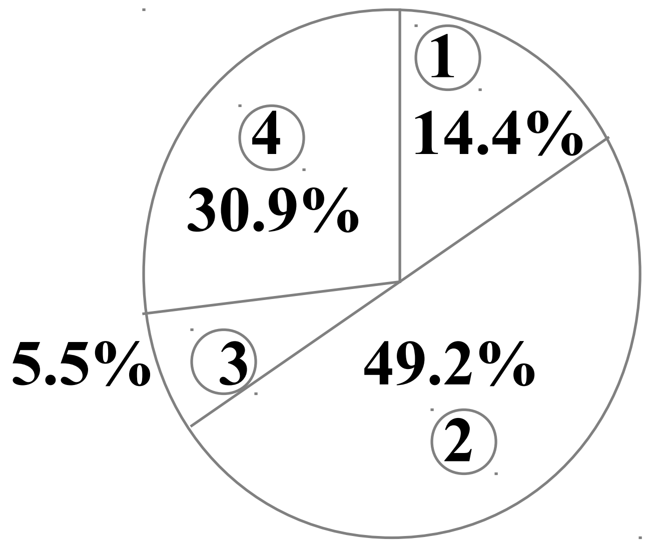
No.	individual	x	(fitness) x^2	% of total fitness
1	0 1 1 0 1	13	169	14.4
2	1 1 0 0 0	24	576	49.2
3	0 1 0 0 0	8	64	5.5
4	1 0 0 1 1	19	361	30.9

E.g.: decoding individual 1:

$0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 13$

A pedagogical example of SGA (3)

- **Selection of the best individuals for reproduction**



- **To select an individual for reproduction, we “spin” the above “biased” roulette wheel, whose slots have sizes proportional to the fitness of the individuals**
- **This is called *roulette-wheel selection***

A pedagogical example of SGA (4)

- **Suppose the selected individuals are**
  - **one copy of individual No. 1**
  - **one copy of individual No. 4**
  - **two copies of individual No. 2**
  - **(individual No. 3 was not selected)**

- **Selected individuals (*parents)* probably undergo crossover, to produce two new individuals (*children*)**
  - **User-defined crossover probability: about 80%**

- **Children can also undergo mutation**
  - **User-defined mutation probability: 1%**

    (in nature most mutations are harmful)

A pedagogical example of SGA (5)

- **Example of one-point crossover:**

  crossover of individuals No. 1 and 2, after the 4th bit

parents	children
0 1 1 0 | 1	0 1 1 0 | 0
1 1 0 0 | 0	1 1 0 0 | 1

  crossover of individuals No. 2 and 4, after the 2nd bit

parents	children
1 1 | 0 0 0	1 1 | 0 1 1
1 0 | 0 1 1	1 0 | 0 0 0

- **Note: crossover point is randomly chosen**

A pedagogical example of SGA (6)

Population at generation 0

individual	x	x² (fitness)
0 1 1 0 1	13	169
1 1 0 0 0	24	576
0 1 0 0 0	8	64
1 0 0 1 1	19	361
	Average:	293
	Maximum:	576

Population at generation 1

individual	x	x² (fitness)
0 1 1 0 0	12	144
1 1 0 0 1	25	625
1 1 0 1 1	27	729
1 0 0 0 0	16	256
	Average:	439
	Maximum:	729

Note: Generation 1 has better individuals than generation 0, i.e., the population evolves...

Another Selection Technique

- ***Tournament Selection*: a small subset of K individuals is chosen at random, then the best individual in this set (the tournament winner) is selected**
- **K = tournament size (a user-specified parameter)**
- **The higher the value of K, the higher the “selective pressure”**

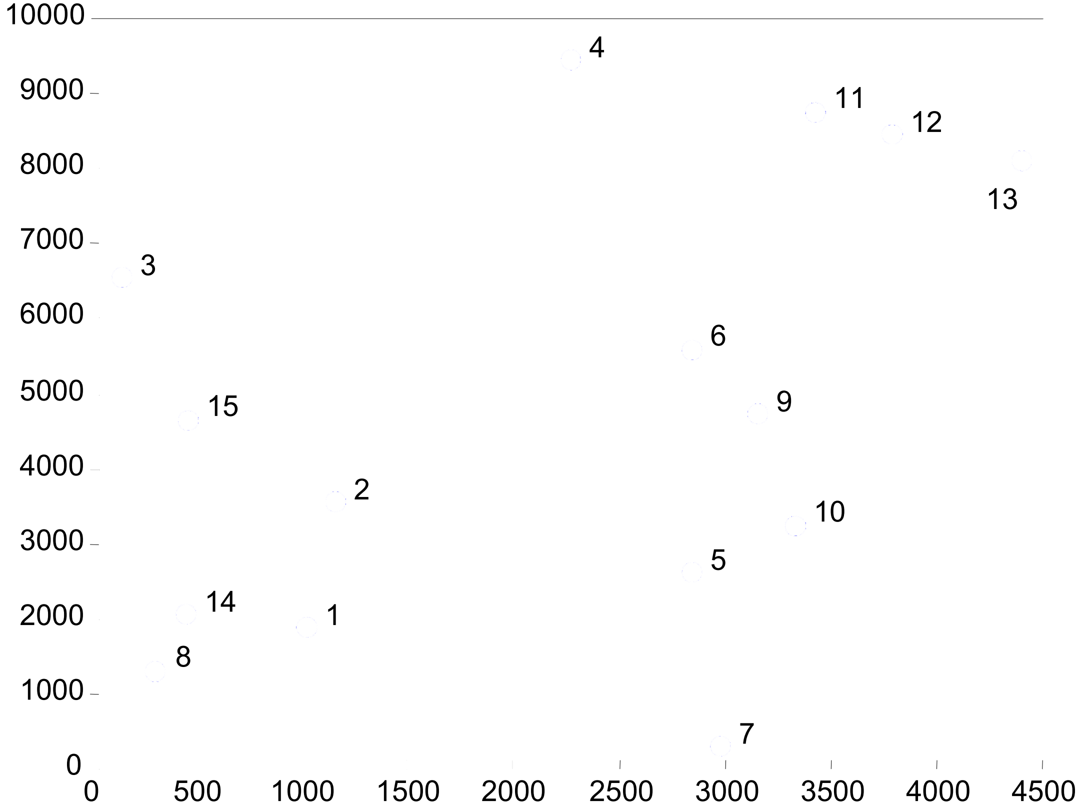
Questions:

1) What happens if K = 1?

2) What happens if K = population size?

The Travelling Salesman Problem

- **Well-known combinatorial optimisation problem**

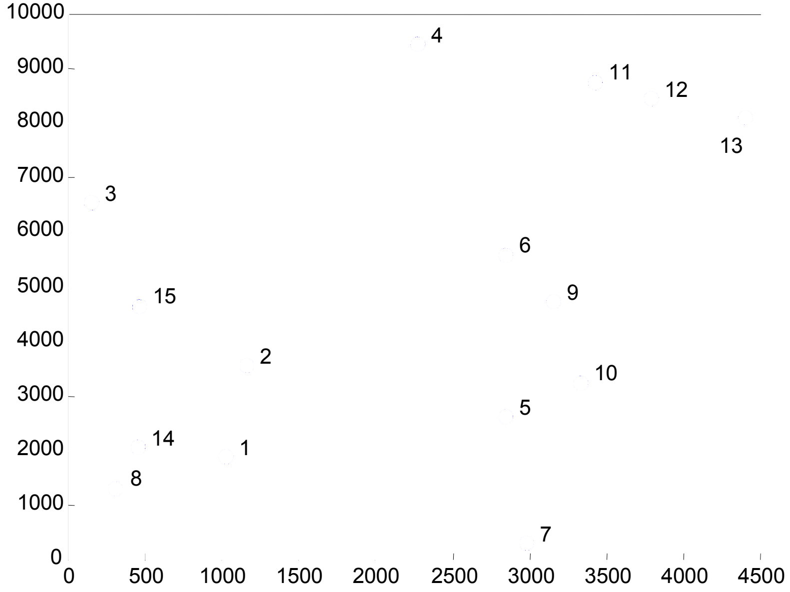


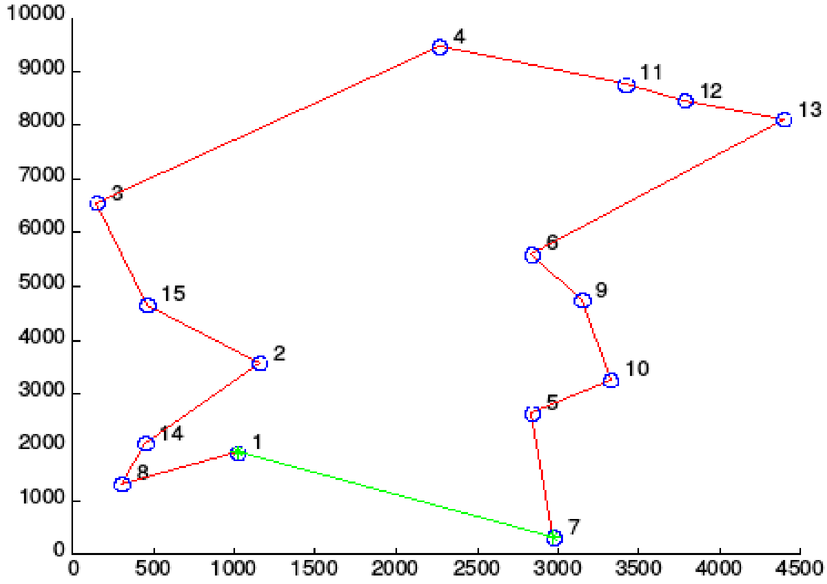
City coordinates

1, 1020, 1900
2, 1152, 3573
3, 142, 6561
4, 2269, 9471
5, 2832, 2642
6, 2839, 5614
7, 2969, 318
8, 306, 1331
9, 3145, 4745
10, 3326, 3269
11, 3415, 8778
12, 3779, 8476
13, 4395, 8125
14, 450, 2089
15, 455, 4671

GA for the Travelling Salesman Problem (1)

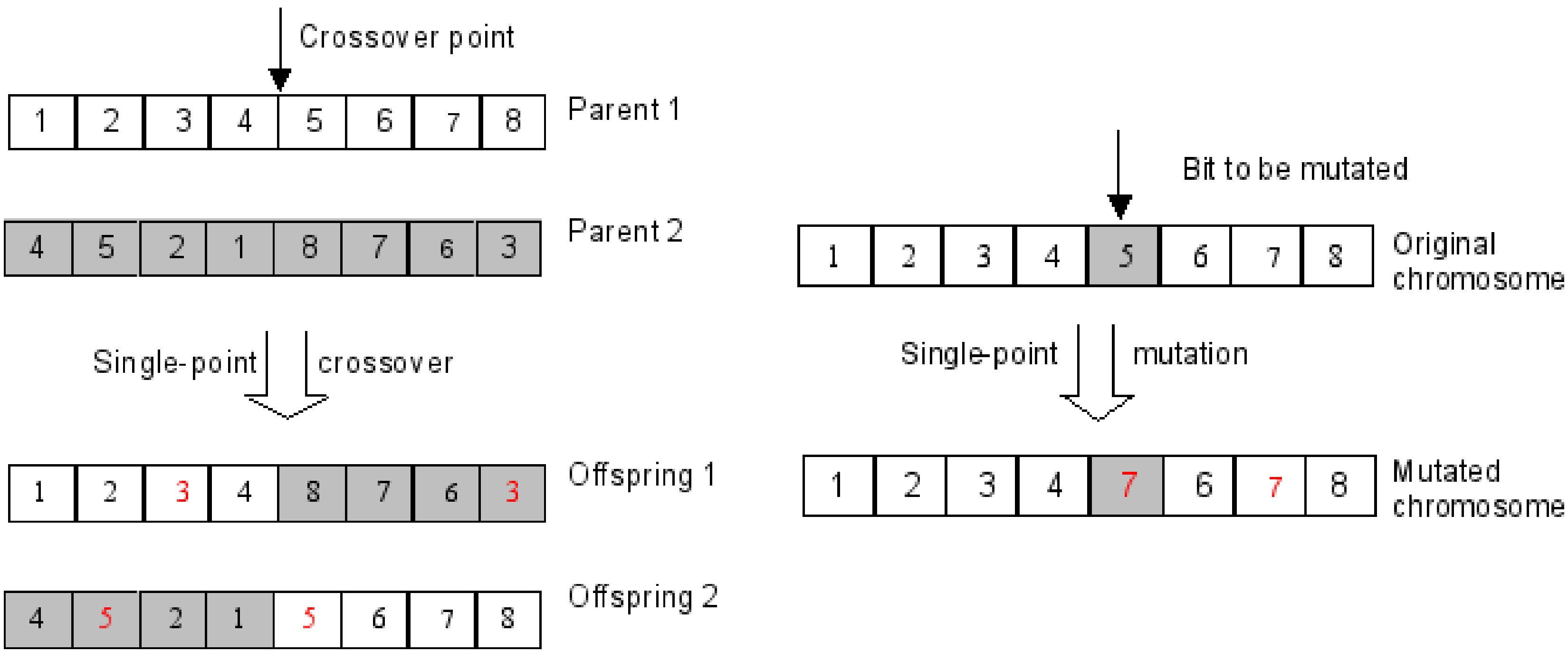
- **Individual (candidate solution) representation:**
  - A permutation of integer numbers (each gene = a city index):
  - € ‹1, 8, 14, 2, 15, 3, 4, 11, 12, 13, 6, 9, 10, 5, 7⟩
- **Fitness function:**
  - distance (length) of each tour





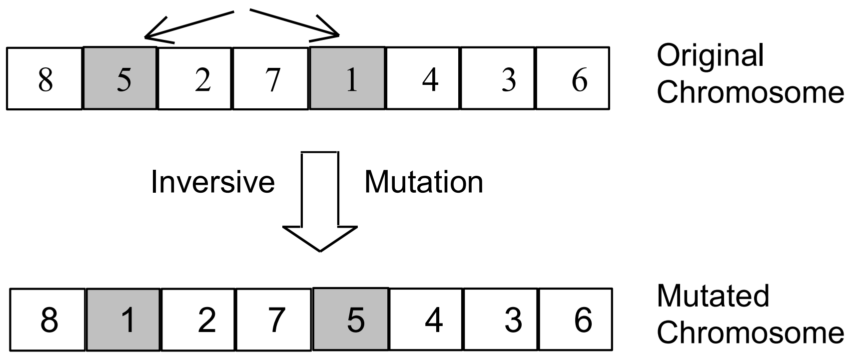
GA for the Travelling Salesman Problem (2)

- **Conventional single-point crossover and single-bit mutation would produce invalid offspring:**



GA for the Travelling Salesman Problem (3)

- **Operator specific for permutation problems:**
  - **Swapping two genes of the same chromosome**
  - **Creates one new child from a given parent**



Advantages of Genetic Algorithms

- Perform a “global” search in the search space
  - work with a *population* of individuals (candidate solutions), rather than with just one candidate solution at a time)
  - Avoid the use of “greedy” heuristics (e.g., start at a given city and visit one city at a time, choosing the nearest city at each step)
  - Global search allows a broader exploration of the search space

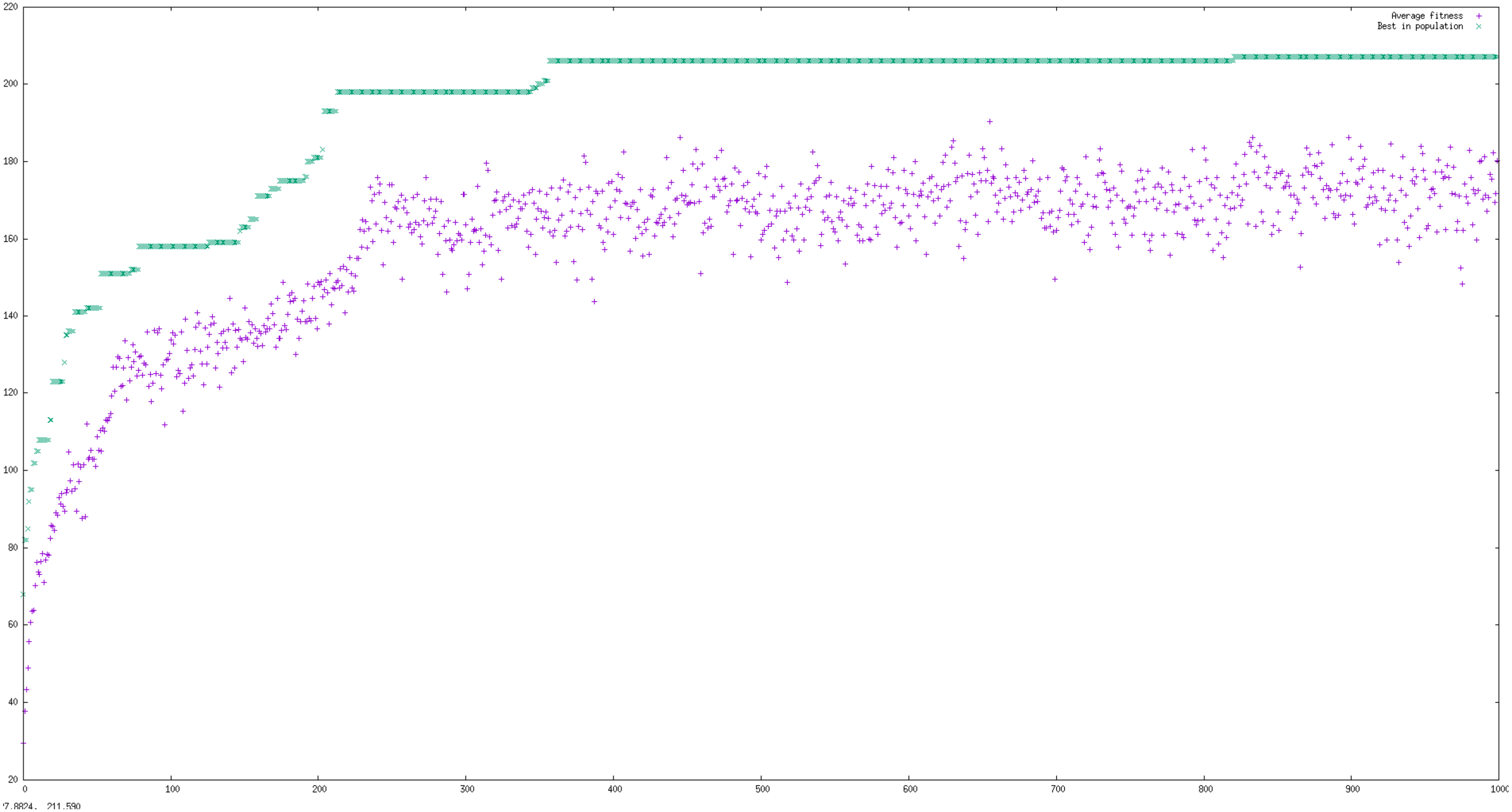
- Represent a candidate solution in a declarative way, independent of the method used to search for a solution, so they allow the easy specification of constraints on the type of solutions to be found

- Easy to implement

Disadvantages of Genetic Algorithms

- Do not offer any guarantee of finding the optimal solution, nor any lower bound on the quality of the solutions to be found
- Are computationally expensive in large-scale problems
- Have several parameters (crossover probability, mutation probabilities, population size, number of generations, etc.), whose "optimisation" is not a trivial task

Typical GA run



References

- D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989. Chapters 1 and 3.

- A.E. Eiben and J.E. Smith. Introduction to Evolutionary Computing. Springer, 2003. Chapters 1 and 2; sections 3.3, 3.4, 3.7.

Assessment 1

- Solve two different problems using genetic algorithms.
- You need to implement 2 different GAs (easiest).
- The fitness function is given to you as a class (Assess.class).
- Your mark will depend partially on the quality of the result (i.e. how good a solution you find).
- You are allowed a maximum time to run code. This time will be measured on raptor (not your home machine)!!! Careful!

Asessment: Exercise 1

- **It takes an array of 20 doubles as input.**
- **A fitness function is implemented in Asess.class, hence you do not need to worry about that.**
- **You need to find the minimum fitness.**
- **You can limit the search to the interval [-5, 5] for all variables.**

Asessment: Example 2

- The second exercise is an instance of the suitcase packing problem.
- You have a maximum weight that you are allowed to pack.
- You want to maximise your utility.
- I am not giving you a table of the values for the utility and the weight. These are encoded in the class file and you do not need this information.
- Calling a method in Assess.class gives you the relevant weights and utility.
- You need to be careful to take into account the weight constraint. You need to construct a fitness function to reflect that.

Dos and don’ts

- Do follow the instructions in every detail.
- Do not add any external libraries or graphical user interfaces.
- Do make sure your code compiles from the command line and Example.java contains the Main method.
- Do not put your code into subdirectories. This will result in points deductions.
- Check the runtime of your code on raptor.
- Do not forget to check in the results once you generated it.
- Do not check in fixed solutions, but the ones you generated. You will be marked on a (numerically) different problem!