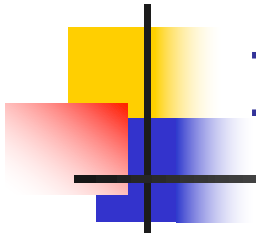
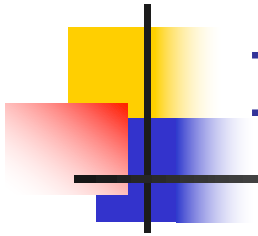
Natural Computing

Material sources from Eiben & Smith's book, R. Frankel, L. Nunes de Castro, W. Williams, and other colleagues.



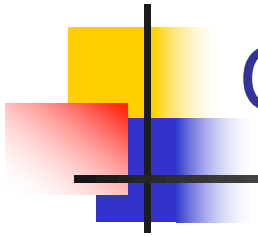
Information

- Teachers:
  - Elena Marchiori elenam@cs.ru.nl
  - Gijs van Tulder gijs.vantulder@ru.nl
  - Johannes Textor Johannes.Textor@ru.nl
- Teaching assistants:
  - Ankur Ankan Ankur.Ankan@radboudumc.nl
  - Franka Buytenhuijs franka.buytenhuijs@ru.nl
  - Inge Wortel Inge.Wortel@ru.nl



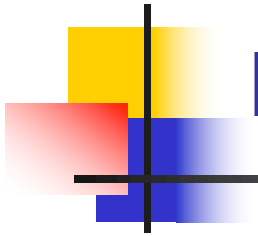
Information

- On Brightspace:
  - Syllabus with pointers to literature and software
  - Slides of the lectures and other material
  - Assignments:
    - Home exercises
    - Project (includes flash talks)
  - Announcements posted during the course, for example schedule of project meetings and flash talks



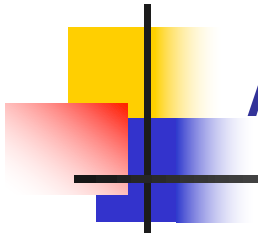
Course structure

- 7 lectures on five topics
- Assignments:
  - 5 home exercises
  - Project



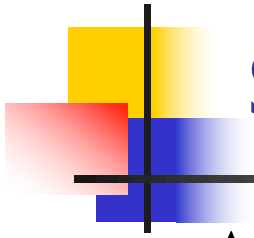
Lectures

- Evolutionary Computation (EC) [Elena] 26 January, 2 February
- Swarm Intelligence (SI) [Elena] 9 February
- Cellular Automata (CA) [Johannes] 16, 23 February
- Artificial Immune System (AIS) [Johannes] 2 March
- Ensemble Learning (EL) [Gijs] 9 March



Assignments

- Home exercises:
  - one for each topic
- Project:
  - Design, implement and test thoroughly a method based on Natural Computing to tackle a problem at your choice
  - Give a flash talk



Schedule

Available on Brightspace: first click on Content, then click on Course Schedule and then on Full Schedule

26/01 Evolutionary Computing (1/2)

2/02 Evolutionary Computing (2/2)

9/02 Swarm Intelligence

23/02 Artificial Immune Systems

2/03 Cellular Automata (1/2)

9/03 Cellular Automata (2/2)

6/04 Ensemble Learning

15/04 deadline project proposal

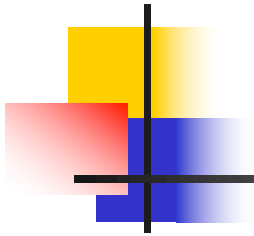
20/04 work at project, meeting with teacher(s) (during the week)

11/05 flash talks, work at project, meeting with teacher(s) (during the week)

18/05 work at project, meeting with teacher(s) (during the week)

25/05 finalize project report and source code

1/06 finalize project report and source code



Deadlines

Home exercises:

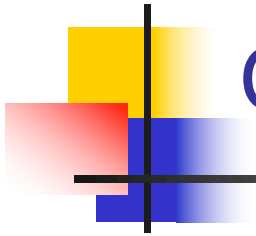
1. Evolutionary Computing, Feb 9, 13:30
2. Swarm Intelligence, Feb 20, 13:30
3. Artificial Immune Systems, Mar 2, 13:30
4. Cellular Automata, Mar 30, 13:30
5. Ensemble Learning, Apr 13, 13:30

Project:

1.Project proposal, Apr 15, 13:30

2. Slide flash talk, May 10, 13:30

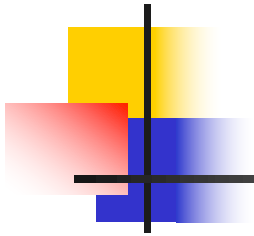
3. Project report deadline, Jun 10, 13:30



Grading

- Team grade: 1) home exercises 40%, 2) project 60%
- Grade for 1) and 2) should be >=5
- Individual final grade: team grade max +1, min -1, based on peer assessment and self-evaluation

Evolutionary Algorithms



Evolution in biology

- Evolution in biology is the study of the diversity of life, the differences and similarities among organisms, and the adaptive and non-adaptive characteristics of organisms
- The main features of evolution and evolutionary systems are:
  - Population(s) of individuals
  - Reproduction with inheritance
  - Genetic variation
  - Natural selection
- *Evolution* can be understood and represented in an abstract and common terminology as an *algorithmic process*

Evolution algorithmically

Evolutionary algorithms (EAs) embody the major processes involved in the theory of biological evolution:

- a *population* of individuals that
- *reproduce with inheritance*, and are subject to
- *variation* and
- *natural selection*.

Interesting reading: A critical tutorial on the use of metaphors in natural computing (like "intelligent" water drops …): Sörensen, Kenneth. "Metaheuristics—the metaphor exposed." *International Transactions in Operational Research* 22.1 (2015): 3-18.

Evolutionary Algorithm

- Initialize the population with candidate solutions
- Evaluate the quality of each candidate
- Repeat until a *termination condition* is satisfied:
  - Select candidate solutions for reproduction
  - Recombine selected candidates
  - Mutate the resulting candidates
  - Evaluate the new candidates
  - Select candidates for the next generation

Three main types of termination condition:

1. A satisfying candidate solution has been obtained;
2. A predefined number of iterations (also called generations) has been reached;
3. The population has converged to a certain level of individual variation.

Evolutionary Algorithm

- Initialize the population with candidate solutions
- Evaluate the quality of each candidate
- Repeat until a *termination condition* is satisfied:
  - Select candidate solutions for reproduction
  - Recombine selected candidates
  - Mutate the resulting candidates
  - Evaluate the new candidates
  - Select candidates for the next generation

Design choices:

- Representation (definition of candidate solutions)
- Population size and initialization
- Evaluation function (fitness function)
- Candidate selection mechanism
- Variation operators (recombination and mutation)
- Survival selection mechanism (replacement)

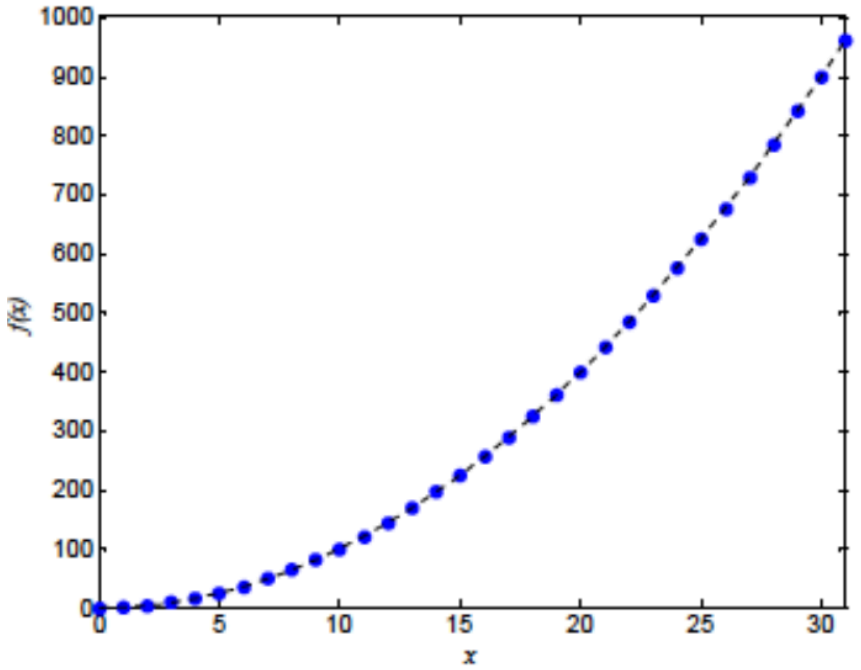
A toy example

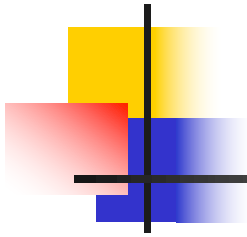
Consider the following maximization problem:

$$\max_{x} f(x)$$

for $f(x) = x^2$ and x integer between 0 and 31.

$f(x)$ has its maximum value 961 at x=31.





Representation

- Use unsigned binary integer of length 5

10110 ⇨ $1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = \mathbf{22}$

- A five-bit unsigned binary integer can have values between 0(0000) and 31(11111)

Population size and initialization

- Initial populations are randomly generated
- Suppose that the population size is 4
- An example initial population

Individual No.	Initial population	x value
1	0 1 1 0 1	13
2	1 1 0 0 0	24
3	0 1 0 0 0	8
4	1 0 0 1 1	19

Fitness

- Decoding

- Results

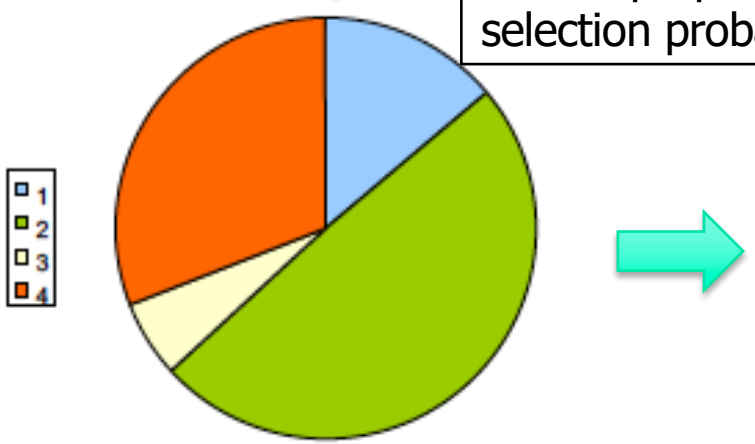
Individual No.	Initial population	x value	$f(x)$	$f_i / \Sigma f$	Expected number
1	**0 1 1 0 1**	13	169	0.14	0.56
2	**1 1 0 0 0**	24	576	0.49	1.96
3	**0 1 0 0 0**	8	64	0.06	0.24
4	**1 0 0 1 1**	19	361	0.31	1.24

Fitness proportional selection probability

Candidate selection mechanism

Individual No.	Initial population	x value	$f(x)$	$f_i / \Sigma f$	Expected number
1	**0 1 1 0 1**	13	169	0.14	0.56
2	**1 1 0 0 0**	24	576	0.49	1.96
3	**0 1 0 0 0**	8	64	0.06	0.24
4	**1 0 0 1 1**	19	361	0.31	1.24

Fitness proportional selection probability



No.	Mating pool
1	**0 1 1 0 1**
2	**1 1 0 0 0**
3	**1 1 0 0 0**
4	**1 0 0 1 1**

Variation operators: crossover

- Two individuals are randomly chosen from mating pool
- Crossover occurs with the probability of $p_c = 1$
- Crossover point is chosen randomly

Mating pool	Crossover point	New population	x	$f(x)$
0 1 1 0 | 1	4	0 1 1 0 0	12	144
1 1 0 0 | 0		1 1 0 0 1	25	625
1 1 | 0 0 0	2	1 1 0 1 1	27	729
1 0 | 0 1 1		1 0 0 0 0	16	256

Variation operators: mutation

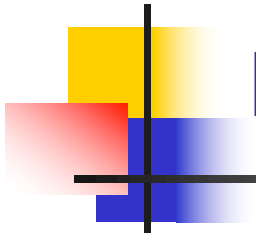
- Applied on a bit-by-bit basis
- Each gene mutated with probability of $p_m = 0.001$

Before Mutation	After Mutation	x	$f(x)$
0 1 1 0 0	**0 1 1 0 0**	12	144
1 1 0 0 1	**1 1 0 0 1**	25	625
1 1 0 1 1	**1 1 0 1 1**	27	729
1 0 0 0 0	**1 0 0 1 0**	18	324

Fitness of new individuals

- Fitness values of the new population are calculated

Old population	x	$f(x)$	New population	x	$f(x)$
0 1 1 0 1	13	169	**0 1 1 0 0**	12	144
1 1 0 0 0	24	576	**1 1 0 0 1**	25	625
0 1 0 0 0	8	64	**1 1 0 1 1**	27	729
1 0 0 1 1	19	361	**1 0 0 1 0**	18	324
	sum	1170		sum	1756
	avg	293		avg	439
	max	576		max	729



Repeat until termination

- Repeat these steps (generate mating pool, apply crossover, apply mutation, evaluate new individuals, generated next population) until the termination condition is satisfied

The Canonical Genetic Algorithm (CGA)

The previous is an example of Canonical (or Simple) GA:

- Initialize the population of candidate solutions
- Evaluate the quality of each candidate
- Repeat until a *termination condition* is satisfied:
  - Select candidate solutions for reproduction
  - Recombine selected candidates
  - Mutate the resulting candidates
  - Evaluate the new candidates
  - Select candidates for the next generation

Binary representation

Fixed population size

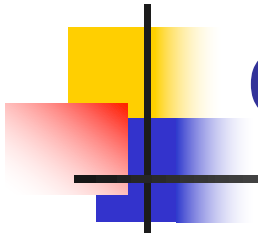
Evaluation function (fitness function)

Roulette wheel selection

Single point crossover
Point mutation

The entire population is replaced at each iteration (generation gap)

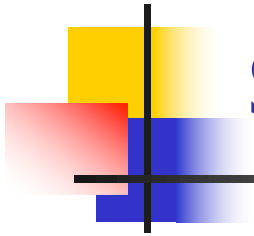
CGA theoretical analysis



Convergence to a global optimum

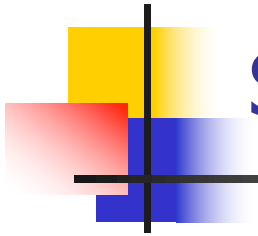
Convergence to a global optimum: *The probability that the population contains the global optimum converges to 1 when time (i.e. number of iterations) goes to infinity.*

- The CGA does not converge to a global optimum
- The CGA with elitist selection scheme, in which the best individual of the previous generation always survives, converges to a global optimum



Schema Theorem

- Schema theorem serves as the analysis tool for the GA process
- Explain why GAs work by showing the expectation of *schema* survival
- Applicable to a Canonical GA
  - binary representation
  - fixed length individuals
  - fitness proportional selection
  - single point crossover
  - gene-wise mutation



Schema

A schema represents a set of binary strings.

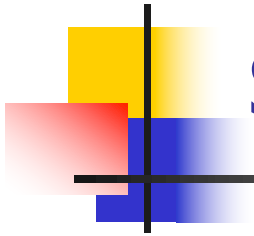
Example:

H = 0*11*, where * is the ‘don’t care’ symbol that matches either 0 or 1.

00110 matches H

H represents the set 00110, 01110, 00111, 01111

The string 10 of length l=2 belongs to $2^l = 2^2$ different schemas: 10, *0, 1*, **.

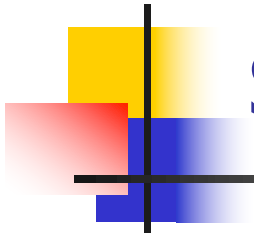


Schema order o(H)

The *order* of a schema is the number of its fixed bits, i.e. the number of bits that are not '*' in the schema H

Example:

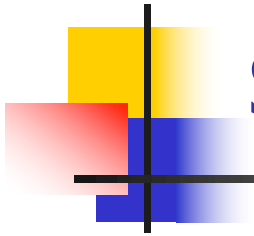
If H = 0*1 then o(H) = 2.



Schema: defining length d(H)

The *defining length* of a schema is the difference between its last and first fixed string position.

- Example: if H = *1*01 then d(H) = 5 – 2 = 3
- Example: if H = 0**** then d(H) = 1 – 1 = 0



Schema: Count

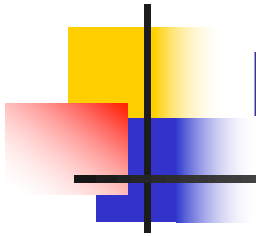
- Suppose x is an individual that belongs to the schema H, then we say that x is an instance of H ($x \in H$)
- *m(H,k)* denotes the number of instances of H occurring in the population at the k-th generation

Schema: Fitness

- The average fitness of H in the k-th generation is

$$f(H,k) = \frac{1}{m(H,k)} \sum_{\substack{x \text{ in the population} \\ \text{matching } H}} m(x,k) f(x)$$

where *m(H, k)* denotes the number of instances of H in the population at generation k, *m(x,k)* the number of copies of x in the population at generation k, and *f(x)* the fitness of x.



Effect of CGA On A Schema

- Effect of Selection
- Effect of Crossover
- Effect of Mutation
- = Schema Theorem

Effect of Selection on Schema

- Assumption: fitness proportional selection
- Selection probability for the individual x

$$p_s(x) = \frac{f(x)}{\sum_{i=1}^{N} f(x_i)}$$

where the N is the total number of individuals in the population

Effect of Selection on Schema

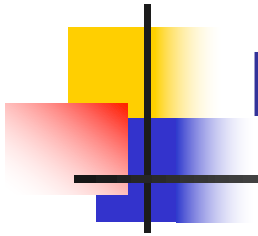
- The expected number M(H,k) of instances of H in the mating pool is

$$M(H,k) = \frac{\sum_{\substack{x \text{ in pop. } k \\ matching\ H}} m(x,k) f(x)}{\bar{f}}$$

$$= m(H,k) \frac{f(H,k)}{\bar{f}}$$

where $\bar{f} = \sum_{i=1}^{N} f(x_i)/N$

- **Schemas with fitness greater than the population average are likely to appear more in the next generation**



Effect of Crossover on Schema

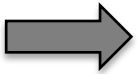
- Assumption: single-point crossover
- Schema H survives crossover operation if
  - one of the parents is an instance of the schema H **AND**
  - one of the offspring is an instance of the schema H

Crossover Survival Examples

Consider H = *10**

P1 = 1 1 0 | 1 0 ∈ H

P2 = 1 0 1 | 1 1 ∉ H



S1 = 1 1 0 1 1 ∈ H

S2 = 1 0 1 1 0 ∉ H

Schema H survived

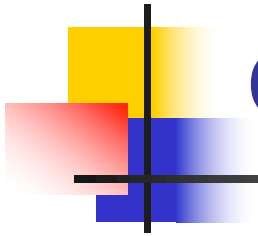
P1 = 1 1 | 0 1 0 ∈ H

P2 = 1 0 | 1 1 1 ∉ H

S1 = 1 1 1 1 1 ∉ H

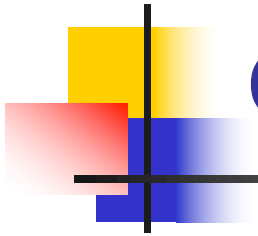
S2 = 1 0 0 1 0 ∉ H

Schema H destroyed



Crossover Operation

- Suppose a parent is an instance of a schema H. When the crossover is occurred within the bits of the defining length, it is destroyed unless the other parent repairs the destroyed portion
- Given a string with length l and a schema H with defining length d(H), the probability that the crossover occurs within the bits of the defining length is $d(H)/(l - 1)$



Crossover Operation

Example:

- Suppose H = *1**0.
- We have $l = 5$, d(H) = 5 – 2 = 3.
- Thus, the probability that the crossover occurs within the defining length is d(H)/(l - 1)=3/4

Crossover Operation

- The probability $D_c(H)$ that the schema H being destroyed is such that:

$$D_c(H) \leq p_c \frac{d(H)}{l-1}$$

where p_c is the crossover probability

Example: Suppose $p_c = 0.8$; $l = 100$

If d (H) = 3 then $D_c(H) \leq 0.8 \frac{3}{99} = 0.024$

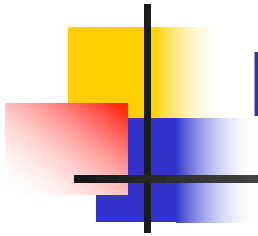
If d(H) = 50 then $D_c(H) \leq 0.8 \frac{50}{99} = 0.404$

Net Effect of Crossover

- The lower bound on the probability $S_c(H)$ that H survives is

$$S_c(H) = 1 - D_c(H) \geq 1 - p_c \frac{d(H)}{l-1}$$

- **Schemas with small defining length are more likely to survive**



Mutation Operation

- Assumption: mutation is applied gene by gene with probability p_m
- For a schema H to survive, all fixed bits must remain unchanged
- Probability of a gene not being changed is

$$(1 - p_m)$$

Net Effect of Mutation on Schema

- The probability a schema H survives under mutation

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

- **Schemas with low order are more likely to survive**

Schema Theorem

Expected # of Schema H in Generation k+1 $>$

Expected # in Mating Pool $(M(H,k) = m(H,k)\frac{f(H,k)}{\bar{f}})$

Prob. of Surviving Crossover $(S_c(H) \geq 1 - p_c \frac{d(H)}{l-1})$

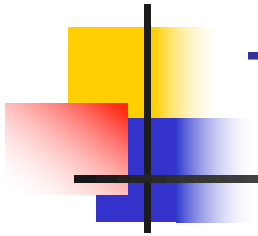
Prob. of Surviving Mutation $(S_m(H) = (1-p_m)^{o(H)})$

= number of instances

Schema Theorem Mathematically

$$E(m(H,k+1)) \geq m(H,k)\frac{f(H,k)}{\bar{f}}\left(1-p_c\frac{d(H)}{l-1}\right)(1-p_m)^{o(H)}$$

- The schema theorem states that the schema with above average fitness, short defining length and low order is more likely to survive



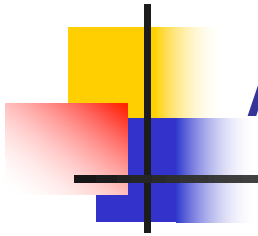
The Building Blocks Hypothesis

- The *Schema Theorem identifies the building blocks* of a good solution although it only addresses the disruptive effects of recombination
- How do we address the constructive effects of recombination? Why a GA should work?

Building Block Hypothesis (BBH):

A GA creates stepwise *better solutions* by selecting, crossing, and mutating *short (that is, small defining length), low-order, high-fitness schemata*, called building blocks.

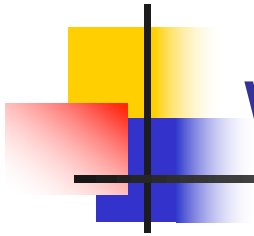
Crossover combines short, low-order schemata into increasingly fit candidate solutions to create even higher fitness schemata.



Arguments against the validity of the BBH

- Superposition of fit schemata possibly generates larger (higher order and defining length) schemata which are less likely to survive
- In populations of realistic size, the observed fitness of instances of a schema may be arbitrarily far from the average fitness used in the definition of schema fitness, even in the initial population.

John J. Grefenstette. Deception Considered Harmful. 1992



When BBH does not hold

- BBH does not hold when there is no information available which could guide a GA to the global optimum through the composition of partial sub-optimal solutions

- EXAMPLE. Consider a needle-in-haystack problem, where

$$f(x) = \begin{cases} 1 \; if \; x = s \\ 0 \; otherwise \end{cases}$$

Values on single positions do not provide any information for guiding a GA to the global optimum

When BBH does hold

- The more the positions in candidate solutions can be evaluated independently, the easier it is for a GA to find the global optimum.

- EXAMPLE. The fitness of x is computed as a linear combination of all its elements

$$f(x) = c + \sum_i c_i x_i$$

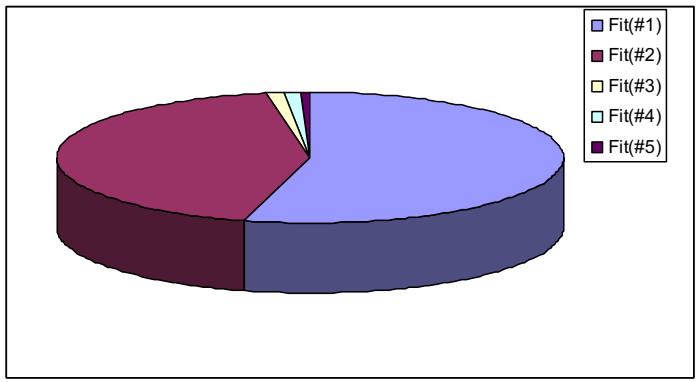
Its optimal value can be determined for every position independently (only depending on the sign of the scaling factors c_i).

Extensions of the CGA

Selection operator

Drawback of roulette-wheel selection:

Relatively superfit individuals will dominate the population.
Too much exploitation, too few exploration, likely to yield premature convergence to a local optimum.



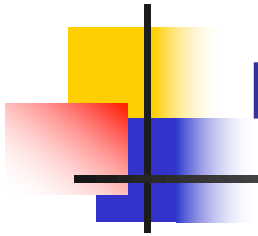
Selection operator: Tournament selection

Tournament selection (TS) with parameter K:

- K individuals are *randomly chosen*
- the fittest one is selected as a parent.

- TS is efficient to code, works on parallel architectures and allows the selection pressure to be easily adjusted

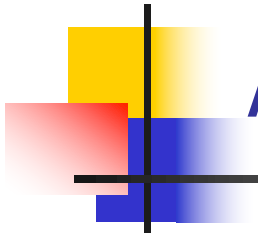
Miller, Brad; Goldberg, David (1995). Genetic Algorithms, Tournament Selection, and the Effects of Noise. Complex Systems. ***9****: 193–212.*



Population replacement/update

- Steady state:
  - Add and remove one individual at each generation
  - Only use part of the population for reproduction
  - The offspring can replace:
    - Parents
    - Worst member of population
    - Randomly selected individuals

- Elitism: Pass best chromosome(s) to next generation. Often beneficial in practice



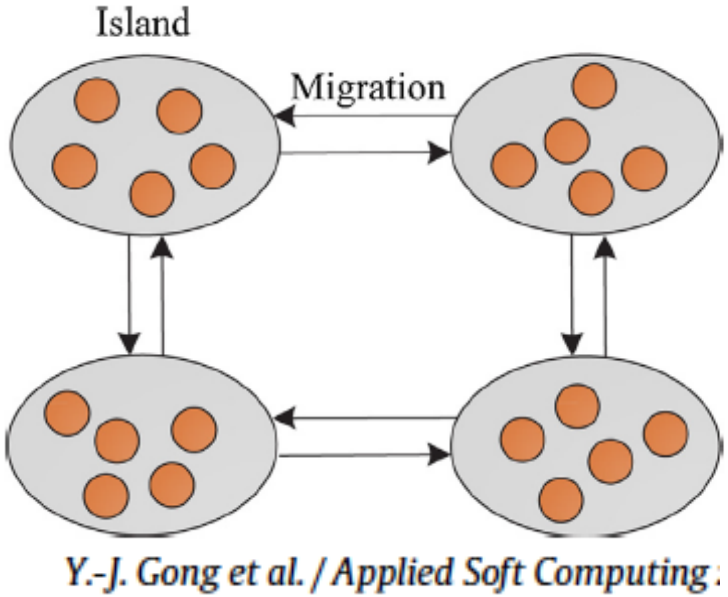
Adaptive population size

1. At birth, every individual is assigned a *lifetime* which corresponds to the number of generations that the individual stays alive in the population
2. The population *grows* either when (1) there is an improvement in best fitness, or (2) there is no improvement in the best fitness for a "long time"

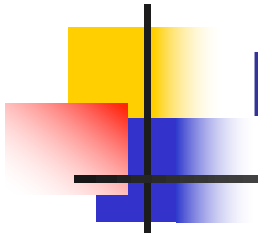
Lobo, F. G., & Lima, C. F. (2005, June). A review of adaptive population sizing schemes in genetic algorithms. In *Proceedings of the 7th annual workshop on Genetic and evolutionary computation* (pp. 228-234). ACM.

Parallelization

- Parallelize execution of fitness-evaluation
- Use multiple populations



Gong, Y. J., Chen, W. N., Zhan, Z. H., Zhang, J., Li, Y., Zhang, Q., & Li, J. J. (2015). Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Applied Soft Computing*, *34*, 286-300.



Hybridizations: Memetic Algorithms

- Memetic algorithms help the GA to find good local optima in shorter time
- They search in a reduced space: search among locally optimal solutions, rather than among any candidate solution in the solution space

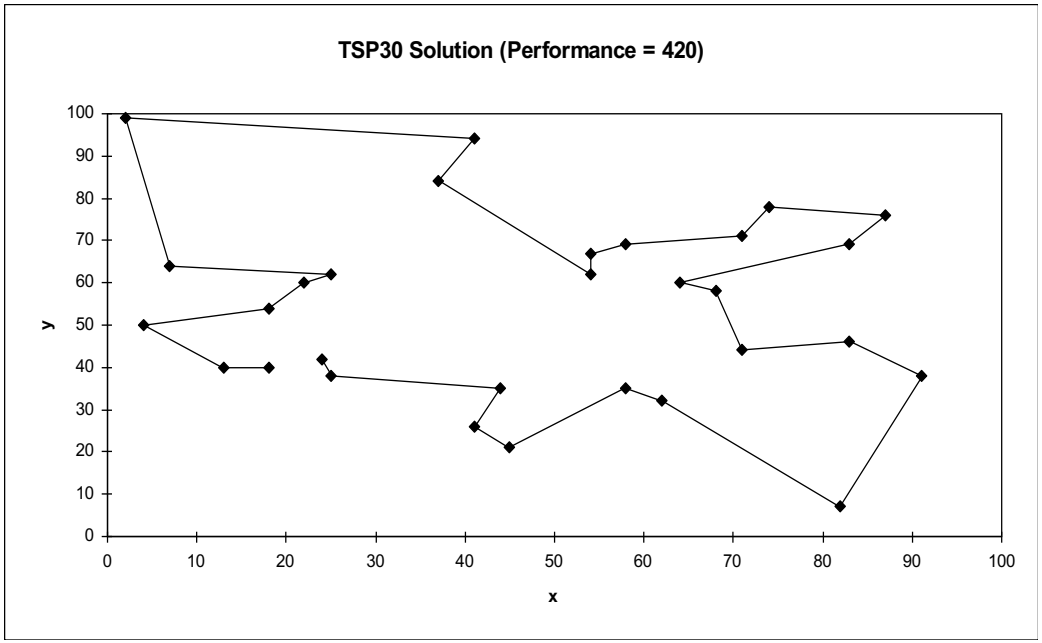
Memetic Algorithm scheme

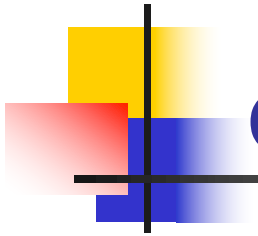
- Initialize the population with random candidate solutions
- Apply LOCAL SEARCH to each individual
- Evaluate the quality of each candidate
- Repeat until termination condition is satisfied:
  - Select parents for reproduction
  - Recombine selected parents
  - Mutate the resulting individuals
  - Apply LOCAL SEARCH to each individual
  - Evaluate the new candidates
  - Select individuals for the next generation

A popular benchmark problem

The Traveling Salesperson Problem

- Find a tour of a given set of cities such that
  - each city is visited only once
  - the total distance traveled is minimized

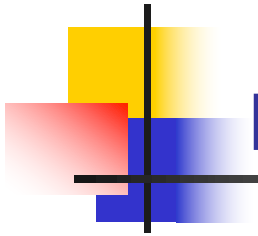




GA for the TSP

Design choices:

- Fitness
- Representation
- Crossover, mutation
- Selection
- Population operators (size, initialization, replacement scheme)



Fitness

- Fitness function: total distance traveled

(here lower value → better fitness)

Representation

- *Permutation of* cities

Example:

1) London 3) Dunedin 5) Beijing 7) Tokyo

2) Venice 4) Singapore 6) Phoenix 8) Victoria

Two GA candidate solutions:

(3 5 7 2 1 6 4 8)

(2 5 7 6 8 1 3 4)

Crossover

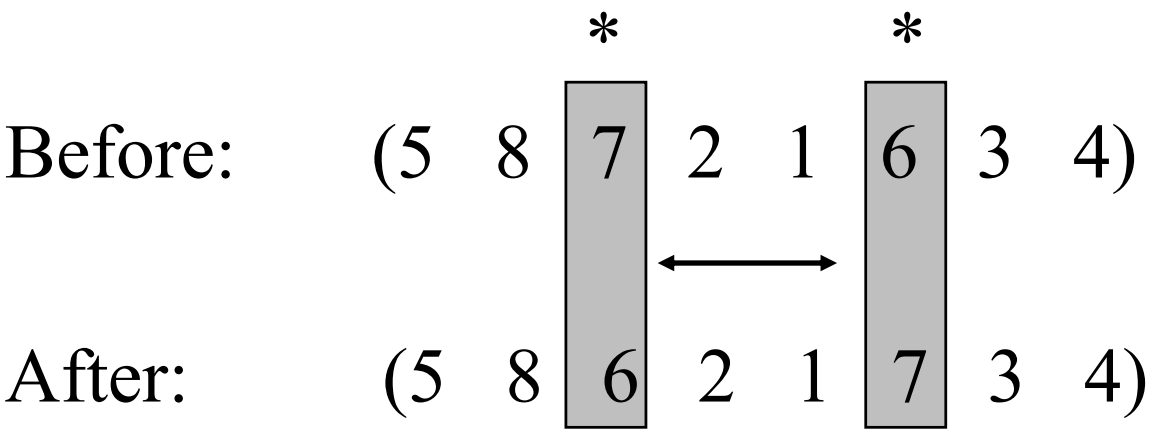
- **Order Crossover**: (1) Choose 2 cut points. (2) Copy between cut points to offsprings. (3) Starting from 2nd cut point in one parent, fill missing cities in order they appear in other parent.

Parent1	(3	5	7	2	1	6	4	8)
Parent2	(2	5	7	6	8	1	3	4)
Offspring1	(5	8	7	2	1	6	3	4)
Offspring2	(5	2	7	6	8	1	4	3)

A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem. International Journal of Computer Applications (0975 – 8887) Volume 31– No.11, October 2011.

Mutation

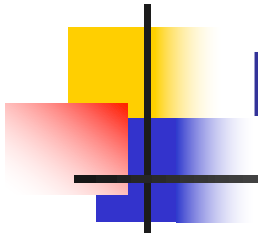
Reverse Sequence Mutation : swap values of two positions



Population, Selection and replacement

- Random initialization of the population
- Fixed-size population
- Binary tournament selection
- Generational gap replacement strategy

Contreras-Bolton, C., & Parada, V. (2015). Automatic combination of operators in a genetic algorithm to solve the traveling salesman problem. *PloS one*, *10*(9), e0137724.

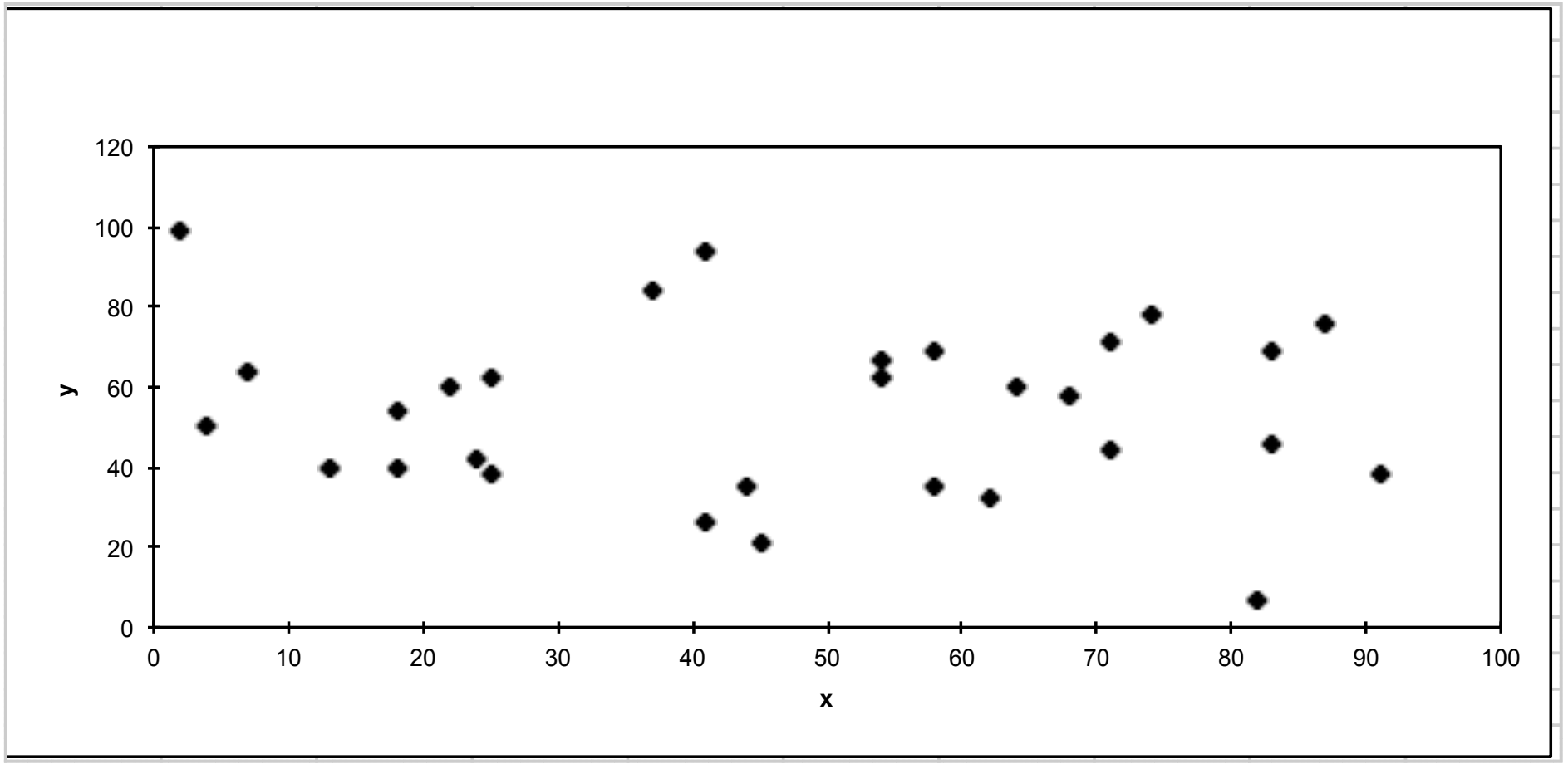


Hybridization

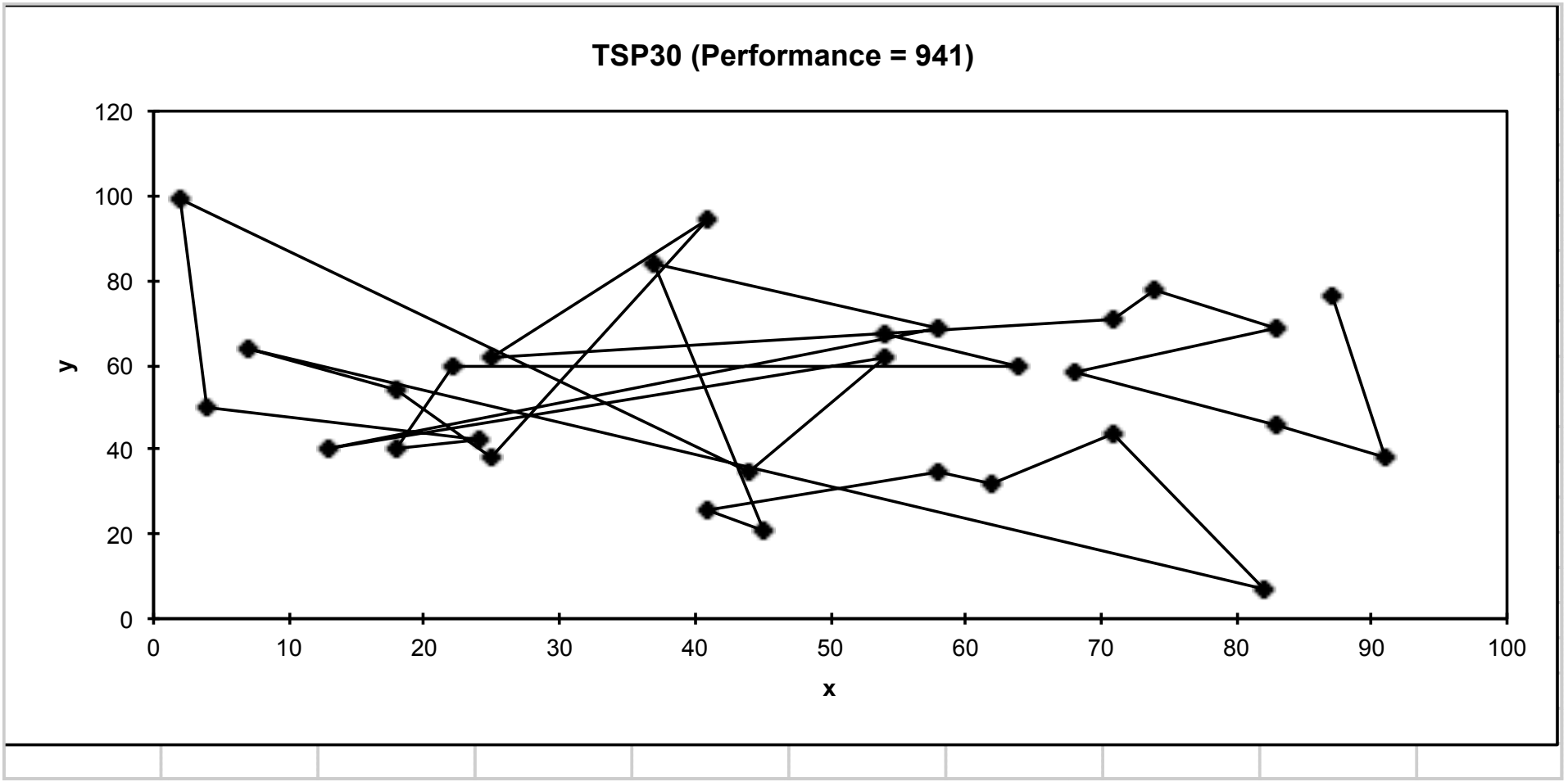
- Memetic algorithm: incorporate a local search method for the TSP in the GA
- One of the Evolutionary Computation home exercises!

Merz, Peter, and Bernd Freisleben. "Memetic algorithms for the traveling salesman problem." *Complex Systems* 13.4 (2001): 297-346.

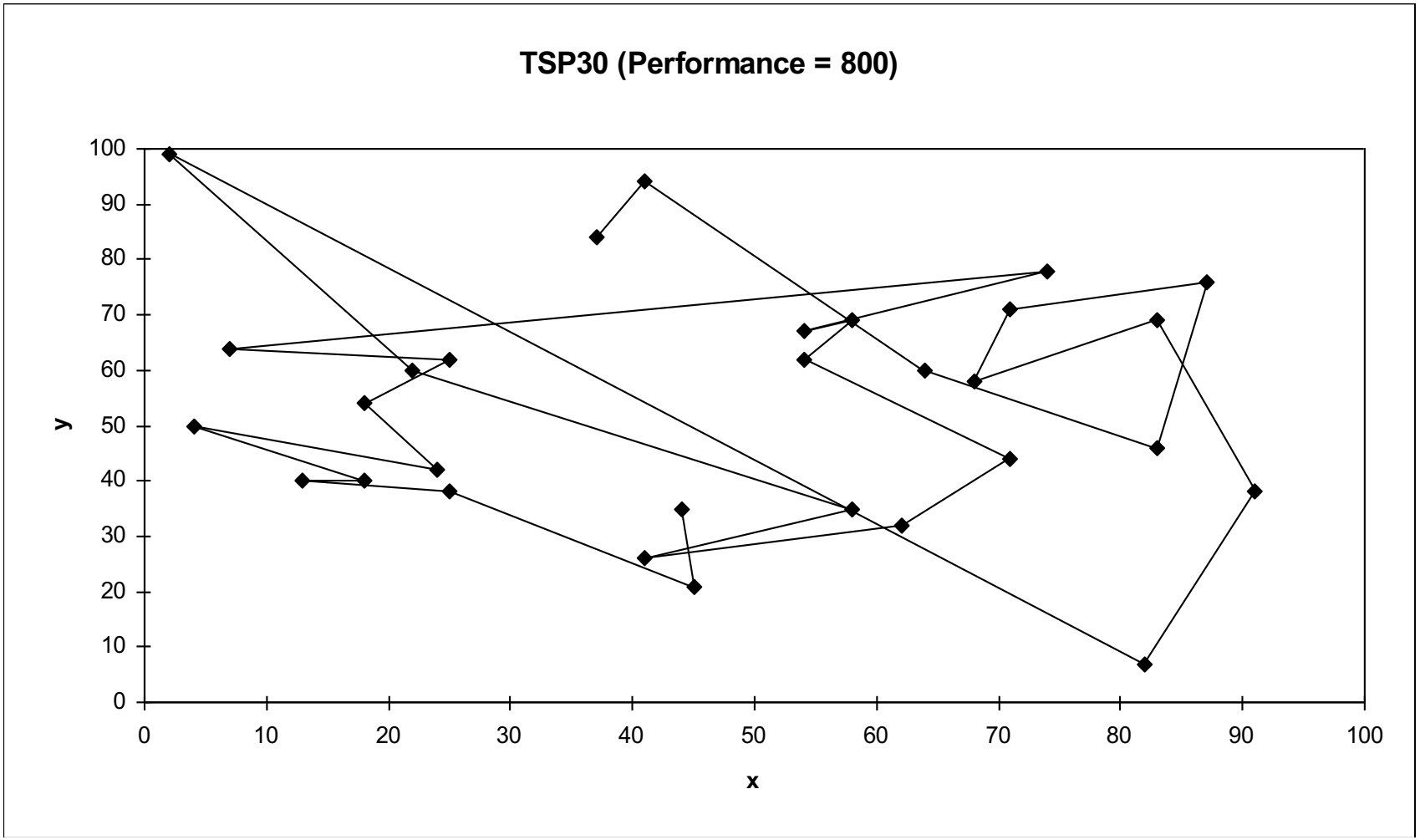
TSP Example run: 30 Cities



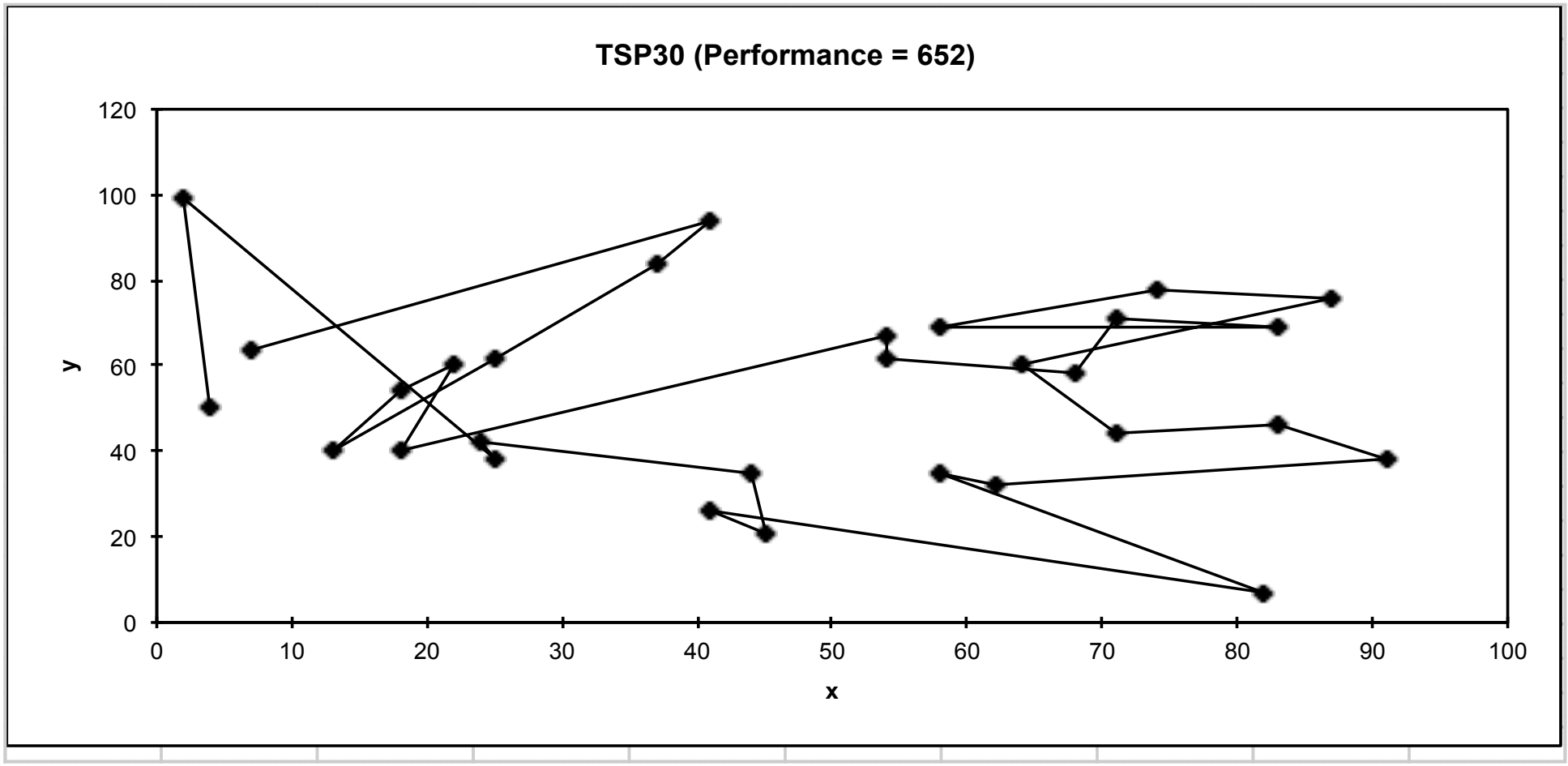
Solution $_i$ (fitness = 941)



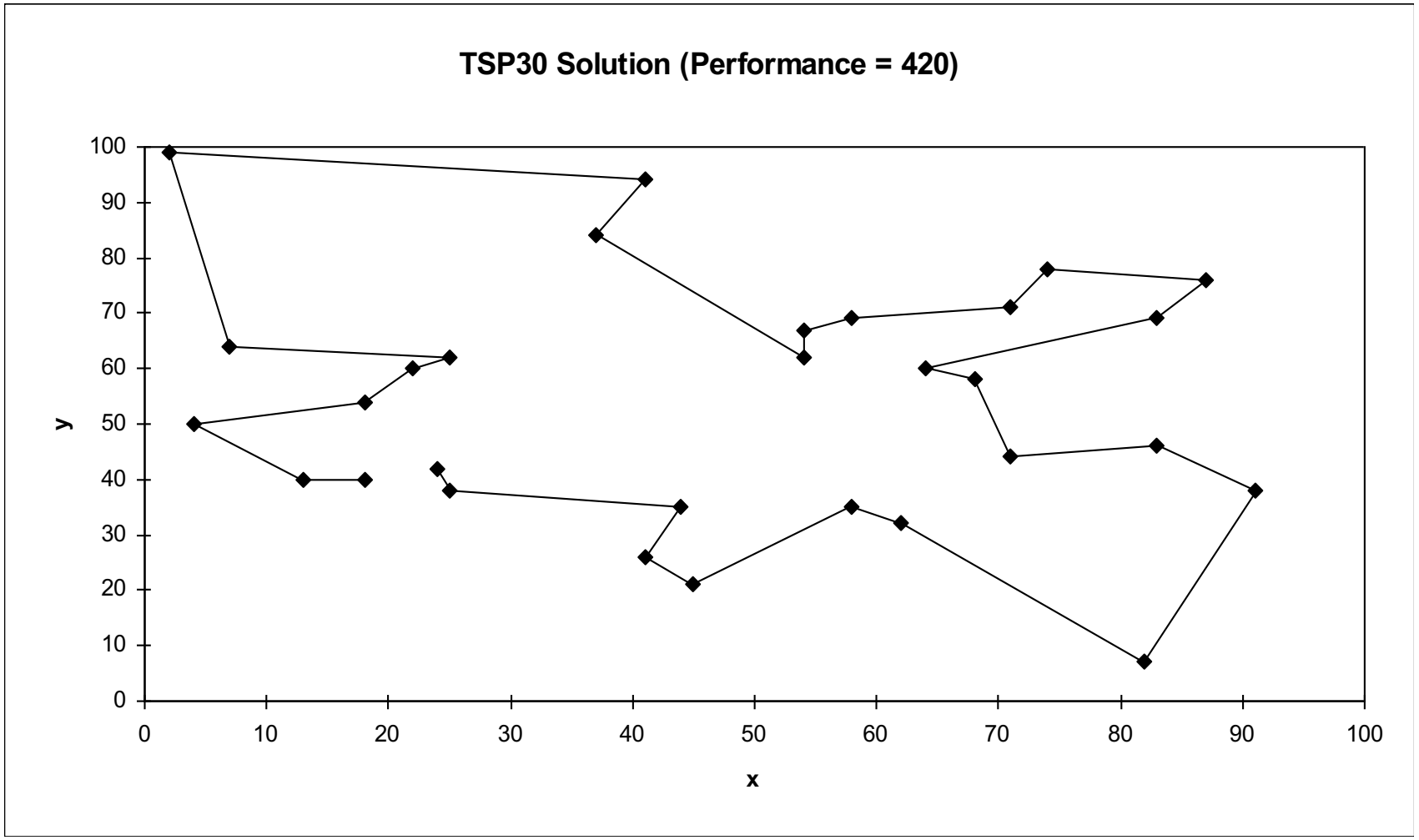
A best solution at 3rd iteration (fitness = 800)



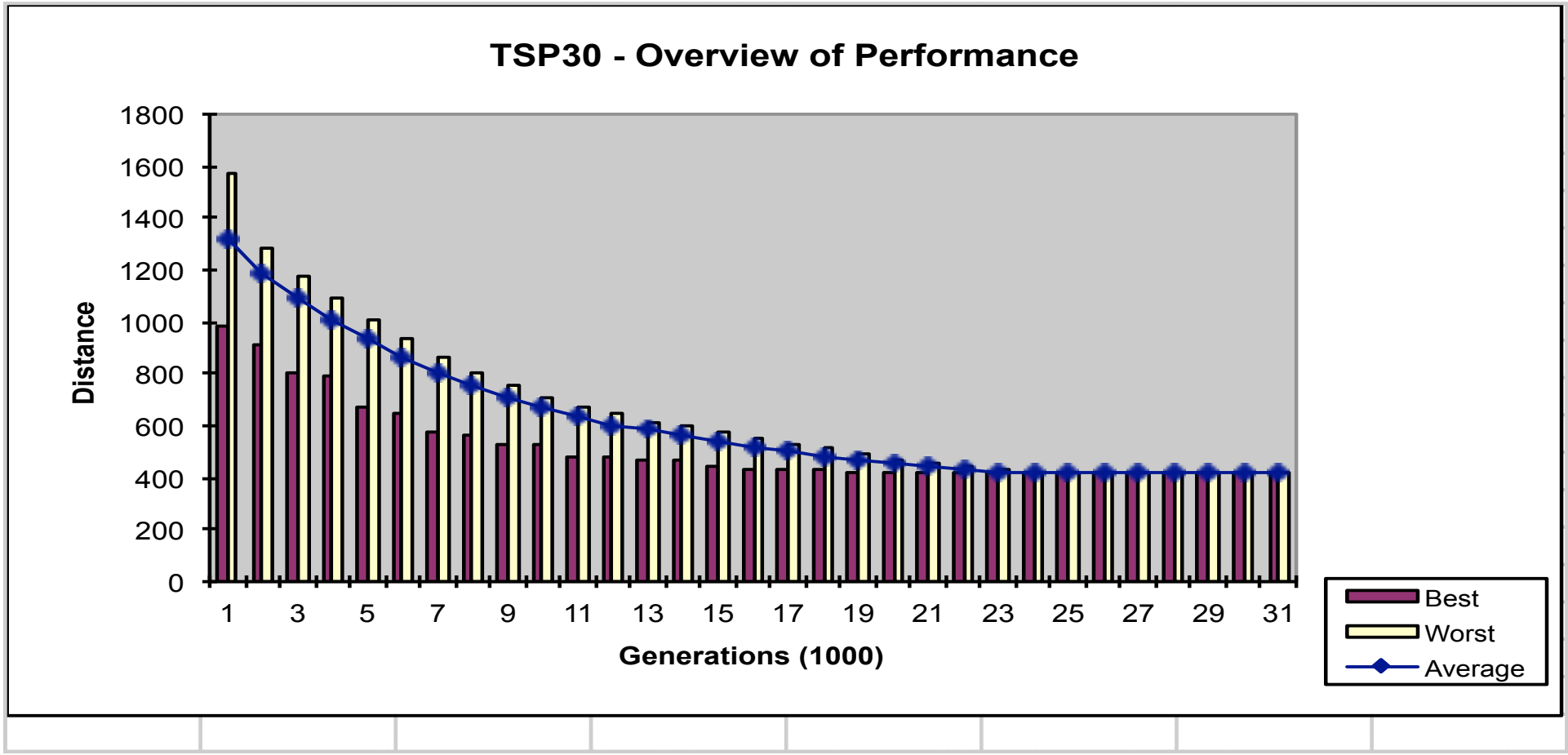
A best solution at 7th iteration (fitness = 652)

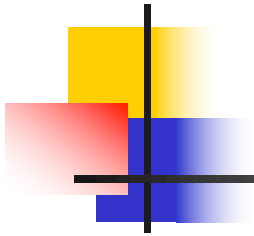


Best solution found after about 30 iterations (fitness = 420)



Overview of Performance



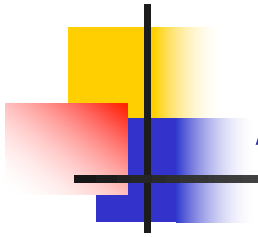


Evolutionary Algorithms in practice

- Perform in different environments and do not break down in the face of e.g. discontinuity
- Applicability in search, optimization, machine learning, …
- Anytime problem solving behavior (return a solution even if interrupted before termination)
- Easy to implement
- Easy to hybridize with local search methods
- Easy to run on parallel machines

Slides on EC's in practice available on Brightspace

(click on "Content" and select "EC in practice")



About the Project

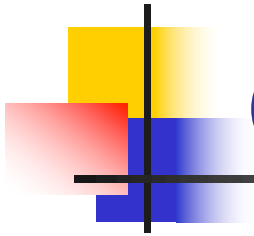
For EA related projects:

- Choose a topic a your choice!
- Look for inspiration at
  - GECCO 2020 papers

https://gecco-2020.sigevo.org/index.html/Accepted+Papers

  - Evolve deep neural networks architectures https://arxiv.org/pdf/1703.00548.pdf
  - GECCO 2021 competitions:

https://gecco-2021.sigevo.org/Competitions



Conclusion

- EA's evolve a population of candidate solutions using stochastic reproduction and selection operators
- Choice of representation, fitness function and genetic operators is crucial
- Elitism and hybridization with local optimization are useful in real-life applications
- EA's are mainly used to tackle discrete-valued optimization problems
- Next week: Evolutionary strategies and Genetic Programming

1st assignment available in Brightspace

deadline 9 February