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
  - Inge Wortel Inge.Wortel@ru.nl
- Teaching assistants:
  - Ankur Ankan Ankur.Ankan@ru.nl
  - Franka Buytenhuijs franka.buytenhuijs@ru.nl
  - Gijs Schröder gijs.schroder@ru.nl

Course structure

- 7 lectures on five topics
  - Evolutionary Computation (EC) [Elena] 1 February, 8 February
  - Swarm Intelligence (SI) [Elena] 15 February
  - Cellular Automata (CA) [Inge] 22 February, 8 March
  - Learning Classifier Systems (LCS) [Inge] 15 March
  - Ensemble Learning (EL) [Gijs] 12 April
- Assignments:
  - 5 home exercises (in teams): one for each topic
  - Project (in teams): design, implement and test thoroughly a method based on Natural Computing to tackle a problem at your choice, give a flash talk, write a report

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

Information

- On Brightspace:
  - Syllabus with pointers to literature and software
  - Slides of 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

5

Schedule with deadlines

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

- 1, 8 February: Evolutionary Computing (EC)
  - Assignment 1 (EC): deadline 15 Feb
- 15 February: Swarm Intelligence (SI)
  - Assignment 2 (SI): deadline 25 Feb
- 22 February, 8 March: Cellular Automata (CA)
  - Assignment 3 (CA): Deadline 18 March
- 15 March: Learning Classifier Systems (LCS)
  - Assignment 4 (LCS): deadline 1 April
- 12 April: Ensemble Learning (EL)
  - Assignment 5 (EL): deadline 25 April
- Project proposal: deadline 30 April
- Week 19-23: work at project
  - Project report: deadline 19 June (end week 24)

6

Course assistants for each home assignment

For questions about the assignments please contact the following assistants:

- Assignment 1 (EC): Ankur Ankan
- Assignment 2 (SI): Franka Buytenhuijs
- Assignment 3 (LCS): Gijs Schröder
- Assignment 4 (CA): Inge Wortel
- Assignment 5 (EL): Ankur Ankan, Franka Buytenhuijs, Gijs Schröder and Inge Wortel

7

Evolutionary Computing, part 1: Evolutionary Algorithms

8

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 of 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 as algorithm process

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*.

But we should not misuse metaphors in natural computing …

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.

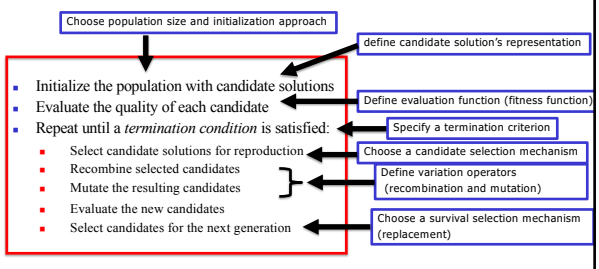
Evolutionary Algorithm

- 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 low level of individual variation

Evolutionary Algorithm design



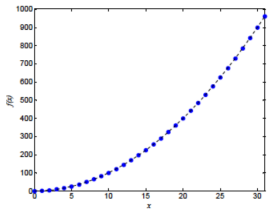
A toy example

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.



13

Candidate solutions 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)

14

Choose population size and initialization method

Random 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

15

Define evaluation function (fitness function)

- Decoding

01000 → Decode → $x = 8$ → Evaluation → $f(8) = 8^2 = 64$

- Results

Individual No.	Initial population	x value	$f(x)$
1	0 1 1 0 1	13	169
2	1 1 0 0 0	24	576
3	0 1 0 0 0	8	64
4	1 0 0 1 1	19	361

⇧ Fitness

16

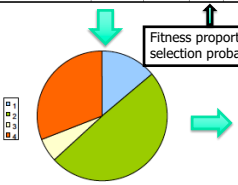
Choose a candidate selection mechanism

Fitness proportional selection:

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

Expected number in mating pool

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

17

Define variation operators: crossover

One-point 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

18

Define variation operators: mutation

Bit flip 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

19

Choose survival selection mechanism

Generational: place the created individuals in a new population which replaces the old population

- 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

20

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)

This was an example of so-called 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)
Fitness proportional selection
One-point crossover
Bit flip mutation
Replacement: the entire population is replaced at each iteration (generational)

Canonical Genetic Algorithm (CGA) 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

- The schema theorem explains why canonical GAs work by analyzing the effect of selection, crossover and mutation on *schemas*.
- Example of a schema: 0*11* represents the set
  00110, 01110, 00111, 01111
  (* is the 'don't care' symbol that matches either 0 or 1)
- Schema theorem: a schema with above average fitness, a small number of fixed bits (*order*) and a small *length* (longest distance between two fixed positions), is more likely to "survive" after selection, crossover and mutation.

25

Schema

A schema represents a set of binary strings.

Schema 0* represents the set 00, 01

String 10 of length l=2 is an instance of (belongs to) $2^l = 2^2$ different schemas: 10, *0, 1*, **

26

Schema theorem in words

Expected nr of instances of schema H at generation k+1 =

N × prob. of *selecting* schema H at gen. k ×

prob. of *surviving crossover* at gen. k ×

prob. of *surviving mutation* at gen. k

N: population size

27

Notation

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

Example:

H = 0*1 has order o(H) = 2

H = 0** has order o(H) = 1

A schema with a small order is more difficult to be "disrupted"

28

Notation

The *defining length* of a schema is the longest distance between two fixed positions.

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

A schema with a small defining length is more difficult to be disrupted

29

Notation

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

Example:

Initial (k=0 generation) population of size 5: (00, 00, 00, 00, 10)

Schema 0* contains 4 instances, m(0*,0)=4

Schema 1* contains 1 instance, m(1*,0)=1

30

Fitness of a schema

- Fitness of H in the *k*-th generation:

$$f(H,k) = \frac{1}{m(H,k)} \sum_{x \text{ in pop. } k \text{ and in } H} m(x,k) f(x)$$

fitness of x

number of instances matching H in pop. k

number of occurrences of x in pop. k matching H

Example:

At k=0 population: (00, 00, 00, 00, 10), fitness function: $f(00) = 1, f(10) = 0.5$

$$f(0*,0) = 1$$
$$f(1*,0) = 0.5$$

Average fitness $f = \sum_{i=1}^{5} f(x_i)/5 = \frac{4.5}{5} = 0.9$

31

Schema theorem

Expected nr of instances of schema H =

N × prob. of *selecting* schema H × prob. of *surviving crossover* × prob. of *surviving mutation*

32

Prob. of selecting schema H

- Assumption: fitness proportional selection
- Prob. of selecting a schema H at generation k

$$p(H,k) = \sum_{\substack{x \text{ in pop. } k \\ \text{and in } H}} m(x,k) p_s(x)$$

- $p_s(x)$ probability of selecting individual x

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

$x_1 \ldots x_N$ population at generation k

Prob. of selecting schema H

$$p(H,k) = \sum_{\substack{x \text{ in pop. } k \\ \text{and in } H}} m(x,k) p_s(x)$$

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

$$p(H,k) = \frac{m(H,k)}{m(H,k)} \sum_{\substack{x \text{ in pop. } k \\ \text{and in } H}} m(x,k) \frac{f(x)}{\sum_{i=1}^{N} f(x_i)}$$

$$= \frac{1}{N} m(H,k) \frac{f(H,k)}{f}$$

$f = \sum_{i=1}^{N} f(x_i)/N$ average fitness of population at generation k

Prob. of selecting schema H

$$p(H,k) = \frac{1}{N} m(H,k) \frac{f(H,k)}{f}$$

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

Schema theorem

Expected nr of instances of schema H =
N × prob. of *selecting* schema H ×
prob. of *surviving crossover* ×
prob. of *surviving mutation*

Prob. of *surviving crossover*

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

37

Crossover Survival Examples

Consider H = *10**

P1 = 1 1 0 | 1 0 ∈ H ⇨ S1 = 1 1 0 1 1 ∈ H

P2 = 1 0 1 | 1 1 ∉ H S2 = 1 0 1 1 0 ∉ H

Schema H survived

P1 = 1 1 | 0 1 0 ∈ H ⇨ S1 = 1 1 1 1 1 ∉ H

P2 = 1 0 | 1 1 1 ∉ H S2 = 1 0 0 1 0 ∉ H

Schema H destroyed

38

Crossover Operation

- Suppose a parent is an instance of a schema H. When the crossover 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 point is within the bits of the defining length is d(H)/(l - 1)

39

Crossover Operation

Example:

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

40

Crossover Operation

$D_c(H)$: probability of schema H being destroyed

p_c: crossover probability

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

$\leq$ instead of $=$ because we do not consider the part "unless the other parent repairs the destroyed portion"

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

41

Prob. of *surviving crossover*

- A lower bound on the probability $S_c(H)$ that H survives crossover 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

42

Schema theorem

Expected nr of instances of schema H =
N × prob. of *selecting* schema H ×
prob. of *surviving crossover* ×
prob. of *surviving mutation*

43

Mutation Operation

- Assumption: mutation (bit flipping) is applied to each position with probability p_m
- For a schema H to survive, all fixed bits must remain unchanged

44

Prob. of *surviving mutation*

- Probability of a bit not being flipped after mutation: $(1 - p_m)$
- 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

45

Schema Theorem

Expected nr of instances of schema H at gen. k+1 >

N × Prob. of selecting schema H ($m(H,k)\frac{f(H,k)}{f}$) ×

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

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

46

Schema Theorem

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

- A schema with above average fitness, short defining length and low order is more likely to survive

47

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.

48

Arguments against the validity of the BBH

- Superposition of fit schemata possibly generates larger (higher order and bigger 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

$$f(x_1, \ldots, x_n) = \begin{cases} 1 \; if \; (x_1, \ldots, x_n) = (0, \ldots, 0) \\ 0 \; otherwise \end{cases}$$

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 $(x_1, \ldots, x_n)$ is computed as a linear combination of all its elements

$$f(x_1, \ldots, x_n) = c + \sum_i c_i x_i$$

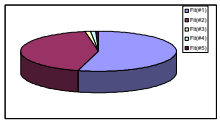
Its optimal value can be determined for every position independently (only depending on the sign of the scaling factors c_i).

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 selected,*
- the one with best fitness is selected as a parent.

- TS is efficient to code 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

A GA with Varying Population Size (GAVaPS):

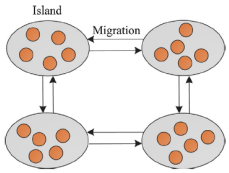
- At birth individuals are assigned a *lifetime* (number of generations that the individual stays alive). Those with above-average fitness are assigned a higher lifetime than those with below-average fitness.
- At birth (through initialization or reproduction) individuals have *age* zero, and age increases at each generation.
- At every generation, a fraction ρ of the current population is randomly selected for reproduction.

No explicit selection operator, but indirect through lifetime. The better an individual is, the longer it stays in the population, and therefore increases the chance to propagate its traits to future individuals.

See e.g. 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 with a migration scheme



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.

57

Hybridation with local search (Memetic Algorithms)

- 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

58

Memetic Algorithms

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

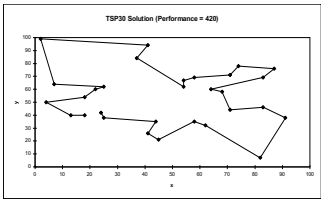
59

A popular benchmark problem

60

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: 1/(total distance traveled)

Representation

- *Permutation of* cities

Example:

1) London 3) Dunedin 5) Beijing 7) Tokyo
2) Venice 4) Singapore 6) Phoenix 8) Victoria

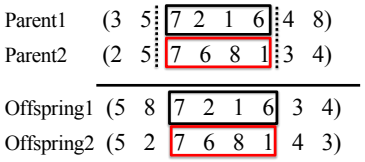
Two GA candidate solutions:

(3 5 7 2 1 6 4 8)

(2 5 7 6 8 1 3 4)

Crossover

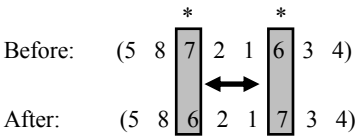
- **Order Crossover**: (1) Choose 2 cut points. (2) Copy between cut points to offsprings. (3) Starting from 2nd cut point in one parent, fill missing cities in order they appear in other parent.



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 replacement strategy

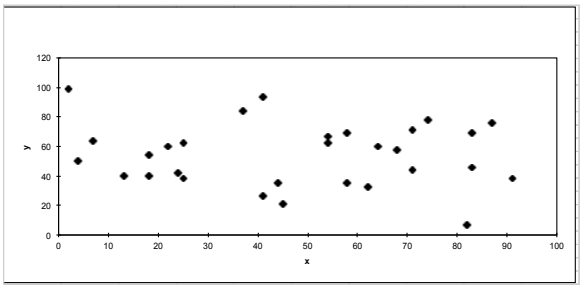
Contreras-Bolton, C., & Parada, V. (2015). Automatic combination of operators in a genetic algorithm to solve the traveling salesman problem. *PloS one*, *10*(9), e0137724.

Hybridization

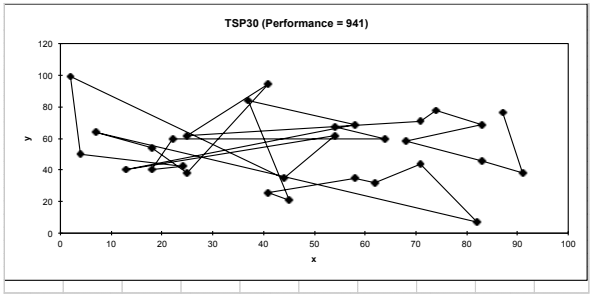
- Memetic algorithm: incorporate a local search method for the TSP in the GA
- One of the Evolutionary Computation home exercises!

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

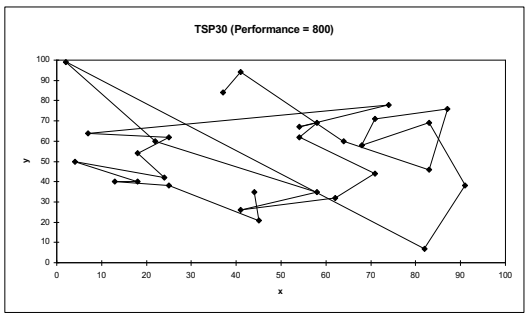
TSP Example run: 30 Cities



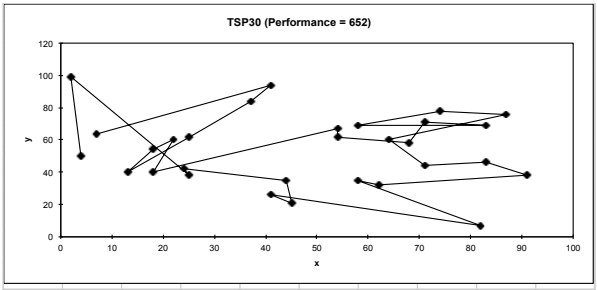
Solution $_i$ (fitness = 941)



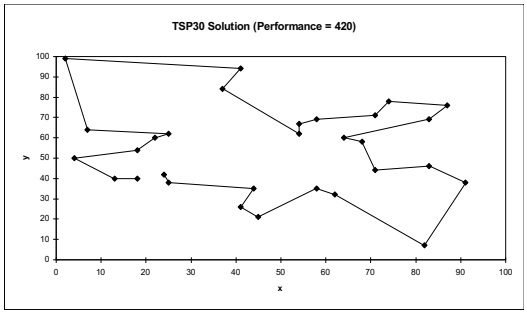
A best solution at 3rd iteration (fitness = 800)



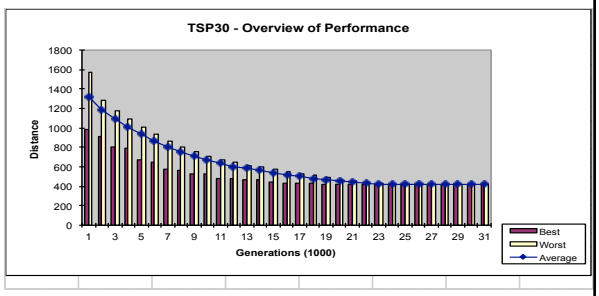
A best solution at 7th iteration (fitness = 652)



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")

75

Your project on Evolutionary Computing?

- Look for inspiration from GECCO papers https://gecco-2021.sigevo.org/Accepted-Papers
- Choose your own problem and solve it using EC (motivate the need for EC versus other approaches!)

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 15 February