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@radboudumc.nl
- Teaching assistants:
  - Ankur Ankan Ankur.Ankan@radboudumc.nl
  - Inge Wortel Inge.Wortel@radboudumc.nl

Information

- In Brightspace
  - Syllabus with pointers to literature and software
  - Slides of the lectures
  - Assignments:
    - Home exercises
    - Project (includes flash talks)
  - Announcements for schedule of project meetings with the teachers and of flash talks

Course structure

- 7 lectures
- Assignments:
  - 5 home exercises
  - Project (includes flash talks)

Teams of 3 students

REGISTER YOUR TEAM **TODAY**

registration form

https://docs.google.com/forms/d/e/1FAIpQLSd7pU1jKj3RArlO4-ZKGLRv6HSAxoIiFurG7_nKmRAadqt9vA/viewform

Lectures

- Evolutionary Computation (EC) [Elena] 4, 11 February
- Swarm Intelligence (SI) [Elena] 18 February
- Cellular Automata (CA) [Johannes] 3,10 March
- Artificial Immune System (AIS) [Johannes] 17 March
- Ensemble Learning (EL) [Gijs] 14 April

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

Grading

- Grading: home exercises 40%, project 60%.
- Individual final grade: team grade max +1, min -1, based on peer assessment and self-evaluation

7

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 evolutionary systems are:
  - Population(s) of individuals
  - Reproduction with inheritance
  - Genetic variation
  - Sorting of variations, also called natural selection
- *Evolution* can be understood and represented in an abstract and common terminology as an *algorithmic process*

9

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 suffer *variation* and *natural selection*.
- Evolutionary algorithms metaphor linking biological evolution and problem solving:
  - Individual -> candidate solution
  - Fitness -> quality
  - Environment -> problem solving

10

Evolutionary Algorithm scheme

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

Three main types of termination condition:

1. A satisfying 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.

11

Evolutionary Algorithm components

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

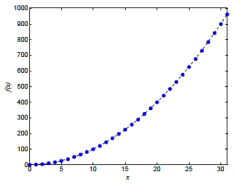
12

A toy example

- Consider the following maximization problem

$$\max \quad f(x) = x^2$$

where x is an integer between 0 and 31



$f(x) = x^2$ has its maximum value 961 at $x = 31$

Representation of each individual

- 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

Initialize the population

- 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

Evaluate the fitness of each individual

- Decoding

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

- 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

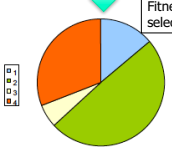
⇧ Fitness proportional selection probability

16

Generate a mating pool

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

17

Perform 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

Perform 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

Evaluate 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

20

Repeat until termination

- Repeat this procedure until the termination condition is met.

21

The Canonical Genetic Algorithm (CGA)

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

- Binary representation of individuals
- Single-point crossover
- Point mutation
- Roulette wheel selection
- Fixed population size
- The entire population is replaced at each iteration (generation gap)

22

CGA theoretical analysis

23

Convergence

In the analysis of the behavior of genetic algorithms there are two types of convergence: convergence to a global optimum and population convergence.

- Convergence to a global optimum: *The probability that the population contains the global optimum converges to 1 when time t goes to infinity.*
  - The Canonical GA does not converge in this sense.
  - Convergence is guaranteed if we extend the Canonical GA by using an elitist selection scheme in which the best individual of the previous generation always survives.

24

Convergence

- Population convergence: *The Hamming distance between individuals convergences to 0 when t goes to infinity.*
  - **Mutation** helps to maintain diversity in the population to prevent convergence to sub-optimal solutions. Too high mutation rates favors exploration of the search space, and prevents converge, too low rates favor exploitation and lead to convergence to a suboptimal solution.
  - The **size of the population** also influences convergence. If it is too small, convergence may occur as in the case of a too low mutation rate; if it is too large, convergence slows down. Typical population sizes are in the range of 30 to 1000; between 50 and 200 is a good choice for many problems.
  - **Selection pressure** also influences convergence. How much should fitter structures be favored in the selection process? If this pressure is too high, less fit population members are quickly driven out, which can lead to convergence to suboptimal solutions. However, a low selection pressure slows down the convergence process.

25

Schema

To analyze the similarities between strings and the correlations of these similarities with fitness Holland introduced the *schema:* a similarity template, for example (0 # 1 1 #) where # is a placeholder, describing a subset of strings with similarities at certain string positions.

- The *order* of a schema is its number of fixed positions
- The *defining length* of a schema, denoted by d (H), is the distance between the first and the last specific string position

EXAMPLE

H = (0 # 1 1 #)
(00110) matches H
H represents the set {(00110), (01110), (00111), (01111)}
H has order 3
H has defining length 3

26

How does CGA work?
The Schema theorem (J. Holland, 1975)

- How does selection improve fitness?
- What is the fate of the schemata in face of selection, mutation and and crossover?

27

Effect of selection 1/3

Let *f(x)* denote the fitness of individual *x*. At reproduction, *x* is selected with a probability

$$\frac{f(x)}{\sum_y f(y)}$$

$\sum_y f(y)$ denotes the sum of fitness of individuals of the current population.
Let m(x, t): number of copies of x in the current population (i.e. at generation t)

$$E(m(x,t+1)) = m(x,t)\frac{f(x)}{average(f)}$$

E(-) is the expected value,
$\frac{1}{n}\frac{f(x)}{average(f)}$: probability of selecting x
$average(f) = \frac{1}{n}\sum_y f(y)$ average fitness of current population,
n: size of the population

28

Effect of selection 2/3

Suppose *x* has above-average fitness:

$$f(x) = average(f)(1+c) \quad c>0.$$

Then $E(m(x,t+1)) = \frac{f(x)}{average(f)}m(x,t) = (1+c)\frac{average(f)}{average(f)}m(x,t)$

If *c* is constant over time then

$$E(m(x,t+1)) = m(x,0)(1+c)^t$$ Exponential growth

- Growth is self-limiting: The relative advantage shrinks because with more fit individuals also the average fitness increases. So fit solutions tend to dominate the population (crossover and mutation being ignored for the moment).
- Analogously: Exponential decay for c<0.

29

Effect of selection 3/3

If individuals all sample the same schema H their fitness define the (average) fitness of H at time t:

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

Where *m(H, t)* is the number of instance of H in the population at time t. How many instances of H can be expected after selection?

$$E(m(H,t+1)) = m(H,t)\frac{f(H)}{average(f)}$$

30

Disruptive effect of 1-point crossover

- 1-point crossover with probability p_c
- d (H) is the defining length of H = distance between first and last fixed position
- H = (# #10 # 1 # #) d (H) = 3
- In a single crossover there are l-1 crossover points:
- (1 0 1 0 0 1 0 0) 7 crossover points
- Of these, d (H) points will disrupt the schema

$$\Pr(disruption) = p_c \frac{d(H)}{l-1}$$

where *l* is the length of the (bit string) individual

- Higher chance of survival if d (H) is low
- Example: Suppose p_c = 0:8; d (H) = 3, l = 100

$$\Pr(disruption) = 0.8\frac{3}{99} = 0.0242$$

Disruptive effect of mutation

- Single-point mutation with probability p_m (applied to each bit in turn)
- o(H) is the order of H
- H = (# # 1 0 # 1 # #) o(H) = 3
- H = (1 1 1 0 # 1 # 1) o(H) = 6
- Probability that a bit survives is $1 - p_m$
- Flipping a defined bit always disrupts a schema, so the probability that the schema survives is

$$\Pr(survival) = (1 - p_m)^{o(H)}$$

For small values of $p_m \ll 1$ we can approximate p_t as

$$\Pr(survival) \approx 1 - p_m o(H).$$

- Best chances for surviving crossover and mutation when d(H) and o(H) are both low

Putting things together: Schema Theorem

Schemata are not only being destroyed, but can also be restored/created though crossover and mutation. So we should write an inequality

$$E(m(H, t+1)) \geq m(H, t)\frac{f(H)}{average(f)}(1 - p_c\frac{d(H)}{l-1})(1 - p_m o(H))$$

Highest when

- $f(H)$ is large: fit
- d(H) is small: short
- o(H) is small: small number of defined bits

The Schema Theorem in words:

Short (i.e. small defining length), low-order, above-average schemata receive exponentially increasing trials in subsequent generation of a genetic algorithm.

The Building Blocks Hypothesis

- The Schema Theorem identifies the building blocks of a good solution although it only addresses the disruptive effects of crossover (and the constructive effects of crossover are supposed to be a large part of why GA work). How do we address the constructive effects?

Building Block Hypothesis (BBH):

A genetic algorithm creates stepwise *better solutions* by recombining, 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: 1) short low-order, high-fitness schemata; 2) stepping stone solutions which combine pairs of schemata to create even higher fitness schemata.

Arguments against the validity of the BBH

- Compositionality: Superposition of fit schemata does not guarantee larger schemata that are more fit and these are less likely to survive.
- Fitness variance within schemata: In populations of realistic size, the observed fitness of a schema may be arbitrarily far from the static average fitness, even in the initial population.

John J. Grefenstette. Deception Considered Harmful. 1992

BBH: When does and does not hold? 1/2

- When there is no information available which could guide a GA to the global solution through partial sub-optimal solutions the BBH does not hold.
- The more the positions can be judged independently, the easier it is for a GA.
- The more positions are coupled (a phenomenon called epistasis), the more difficult it is for a GA (and for any other optimization method).
- Empirical studies have shown that GAs are appropriate for problems with medium epistasis.

BBH: When does and does not hold? 2/2

- YES. The fitness of s is computed as a linear combination of all its elements

$$f(s) = c + \sum_i c_i s_i$$

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

- NO. Consider a needle-in-haystack problem, where

$$f(s) = \begin{cases} 1 \; if \; s = s_0 \\ 0 \; otherwise \end{cases}$$

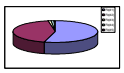
Certain values on single positions (e.g. schemata) do not provide any information for guiding an optimization algorithm to the global optimum.

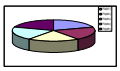
Extensions of the CGA

Selection operator: Tournament selection

With roulette-wheel selection:

1. Relatively superfit individuals will dominate the population. Too much exploitation, too few exploration, likely to premature convergence to a local optimum.
2. Fitness of all individuals is similar. No selection pressure, too few exploitation, too much exploration, after many generations the average fitness converges, but no global optimum is found.





Tournament selection (TS): *n* 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

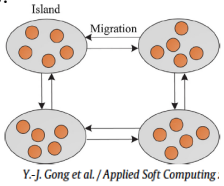
Adaptive population size

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.



Y.-J. Gong et al. / Applied Soft Computing

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

To boost the algorithm to find good local optima we can search among locally optimal solutions, rather than among any candidate solution in the solution space.

- Initialize the population of 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

Goal: Find a tour of a given set of cities such that

- each city is visited only once,
- the total distance traveled is minimized.

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

Fitness

Find a tour of a given set of cities so that

- each city is visited only once,
- the total distance traveled is minimized.

- Fitness: total distance.

Representation

Representation is an ordered list of city numbers known as an *order-based* GA.

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

CityList1 (3 5 7 2 1 6 4 8)
CityList2 (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)
          ------------------------
Child     (5  8  7  2  1  6  3  4)
```

Based on the observation that order is important, not necessarily position.

Other crossover operators for TSP: see paper

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 :

Before: (5 8 7 2 1 6 3 4)

After: (5 8 6 2 1 7 3 4)

Selection and population operators

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

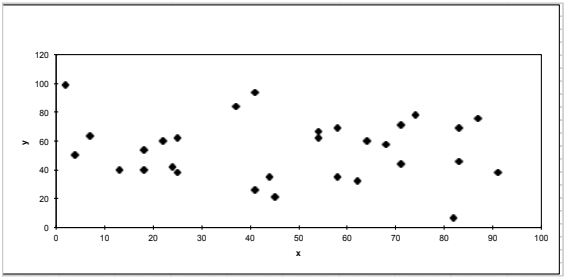
Contreras-Bolton, C., & Parada, V. (2015). Automatic combination of operators in a genetic algorithm to solve the traveling salesman problem. *PloS one*, *10*(9), e0137724.

Hybridization

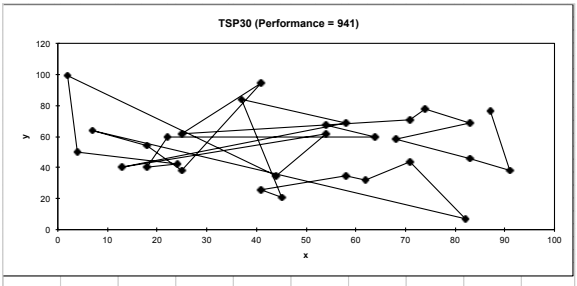
- Hybridizations incorporating a local search method for the TSP (memetic algorithms).
- Home exercise.

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

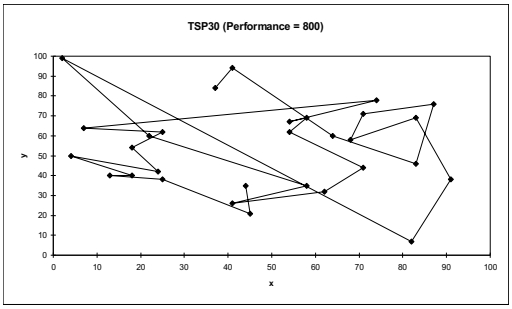
TSP Example: 30 Cities



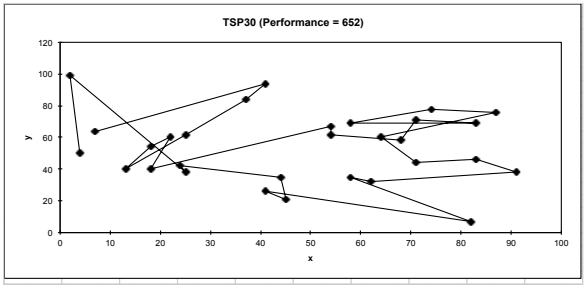
Solution $_i$ (fitness = 941)



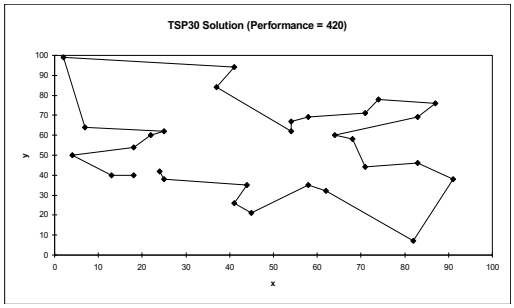
Solution $_j$(fitness = 800)



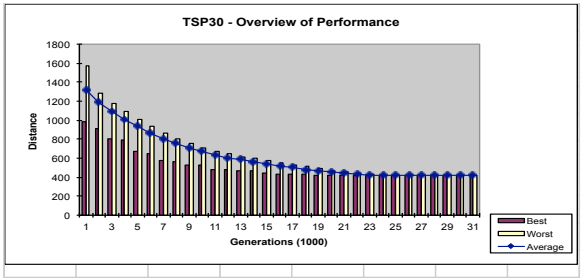
Solution $_k$(fitness = 652)



Best Solution (fitness = 420)



Overview of Performance



EA' s in practice

58

Working with Evolutionary Algorithms

- Experiment design
- Algorithm design
- Test problems
- Measurements and statistics
- Some tips and summary

Slides adapted from chapter 9 of Eiben and Smith' s book Introduction to EC
http://www.evolutionarycomputation.org/slides/

59

Experimentation

- Has a goal or goals
- Involves algorithm design and implementation
- Needs problem(s) to run the algorithm(s) on
- Amounts to running the algorithm(s) on the problem(s)
- Delivers measurement data, the results
- Is concluded with evaluating the results in the light of the given goal(s)
- Is often documented

60

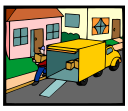
Experimentation: Goals

- Get a good solution for a given problem
- Show that EC is applicable in a (new) problem domain
- Show that *my_EA* is better than *benchmark_EA*
- Show that EAs outperform traditional algorithms (sic!)
- Find best setup for parameters of a given algorithm
- Understand algorithm behavior (e.g. pop dynamics)
- See how an EA scales-up with problem size
- See how performance is influenced by parameters
- …

61

Example: Production Perspective

- Optimising Internet shopping delivery route
  - Different destinations each day
  - Limited time to run algorithm each day
  - Must *always* be *reasonably* good route in limited time



62

Example: Design Perspective

- Optimising spending on improvements to national road network
  - Total cost: billions of Euro
  - Computing costs negligible
  - Six months to run algorithm on hundreds computers
  - Many runs possible
  - Must produce *very* good result just *once*

63

Perspectives of goals

- Design perspective:
  find a very good solution at least once
- Production perspective:
  find a good solution at almost every run
- Publication perspective:
  must meet scientific standards (huh?)
- Application perspective:
  good enough is good enough (verification!)

These perspectives have very different implications on evaluating the results (yet often left implicit)

64

Algorithm design

- Design a representation
- Design a way of evaluating an individual
- Design suitable mutation operator(s)
- Design suitable recombination operator(s)
- Decide how to select individuals to be parents
- Decide how to select individuals for the next generation (how to manage the population)
- Decide how to start: initialization method
- Decide how to stop: termination criterion

65

Test problems

- 5 DeJong functions
- 25 “hard” objective functions
- Frequently encountered or otherwise important variants of given practical problem
- Selection from recognized benchmark problem repository (e.g. TSP benchmark instances from the OR repository)
- Problem instances made by random generator

Choice has severe implications on
- generalizability and
- scope of the results

66

Bad example (1/2)

- I invented "tricky mutation"
- Showed that it is a good idea by:
  - Running standard (?) GA and tricky GA
  - On 10 objective functions from the literature
  - Finding tricky GA better on 7, equal on 1, worse on 2 cases
- I wrote it down in a paper
- And it got published!
- Q: what did I learned from this experience?
- Q: is this good work?

67

Bad example (2/2)

- What did I (my readers) not learn:
  - How relevant are these results (test functions)?
  - What is the scope of claims about the superiority of the tricky GA?
  - Is there a property distinguishing the 7 good and the 2 bad functions?
  - Are my results generalizable? (Is the tricky GA applicable for other problems? Which ones?)

68

Getting Problem Instances (1/3)

- Testing on real data
- Advantages:
  - Results could be considered as very relevant viewed from the application domain (data supplier)
- Disadvantages
  - Can be over-complicated
  - Can be few available sets of real data
  - May be commercial sensitive – difficult to publish and to allow others to compare
  - Results are hard to generalize

69

Getting Problem Instances (2/3)

- Standard data sets in problem repositories, e.g.:
  - OR-Library
    http://www.ms.ic.ac.uk/info.html
  - UCI Machine Learning Repository
    www.ics.uci.edu/~mlearn/MLRepository.html
- Advantage:
  - Well-chosen problems and instances (hopefully)
  - Much other work on these → results comparable
- Disadvantage:
  - Might still miss crucial aspects
  - Algorithms get tuned for popular test suites

70

Getting Problem Instances (3/3)

- Problem instance generators produce simulated data for given parameters, e.g.:
  - GA/EA Repository of Test Problem Generators
    http://www.cs.uwyo.edu/~wspears/generators.html
- Advantage:
  - Allow very systematic comparisons for they
    - can produce many instances with the same characteristics
    - enable gradual traversal of a range of characteristics (hardness)
  - Can be shared allowing comparisons with other researchers
- Disadvantage
  - Not real – might miss crucial aspect
  - Given generator might have hidden bias

71

Basic rules of experimentation

- EAs are stochastic →
  never draw any conclusion from a single run
  - perform sufficient number of independent runs
  - use statistical measures (averages, standard deviations)
  - use statistical tests to assess reliability of conclusions
- EA experimentation is about comparison →
  always do a fair competition
  - use the same amount of resources for the competitors
  - try different comp. limits
  - use the same performance measures

72

Things to Measure

Many different ways. Examples:

- Average result in given time
- Average time for given result
- Proportion of runs within % of target
- Best result over *n* runs
- Amount of computing required to reach target in given time with % confidence
- …

73

What time units do we use?

- Elapsed time?
  - Depends on computer, network, etc…
- CPU Time?
  - Depends on skill of programmer, implementation, etc…
- Generations?
  - Difficult to compare when parameters like population size change
- Evaluations?
  - Evaluation time could depend on algorithm, e.g. direct vs. indirect representation

74

Measures

- Performance measures (off-line)
  - Efficiency (alg. speed)
    - CPU time
    - No. of steps, i.e., generated points in the search space
  - Effectivity (alg. quality)
    - Success rate
    - Solution quality at termination
- "Working" measures (on-line)
  - Population distribution (genotypic)
  - Fitness distribution (phenotypic)
  - Improvements per time unit or per genetic operator
  - …

75

Performance measures

- No. of generated points in the search space
  = no. of fitness evaluations
  (don't use no. of generations!)
- AES: average no. of evaluations to solution
- SR: success rate = % of runs finding a solution (individual with acceptabe quality / fitness)
- MBF: mean best fitness at termination, i.e., best per run, mean over a set of runs
- SR ≠ MBF
  - Low SR, high MBF: good approximizer (more time could possibly help)
  - High SR, low MBF: "Murphy" algorithm (few very bad runs)

76

Fair experiments

- Basic rule: use the same computational limit for each competitor
- Allow each EA the same no. of evaluations, but
  - Beware of hidden labour, e.g. in heuristic mutation operators
  - Beware of possibly fewer evaluations by smart operators
- EA vs. heuristic: allow the same no. of steps:
  - Defining "step" is crucial, might imply bias!
  - Scale-up comparisons alleviate this bias

77

Better example: problem setting

- I invented myEA for problem X
- Looked and found 3 other EAs and a traditional benchmark heuristic for problem X in the literature
- Asked myself when and why is myEA better

78

Better example: experiments

- Found/made problem instance generator for problem X with 2 parameters:
  - n (problem size)
  - k (some problem specific indicator)
- Selected 5 values for k and 5 values for n
- Generated 100 problem instances for all combinations
- Executed all alg's on each instance 100 times (benchmark was also stochastic)
- Recorded AES, SR, MBF values w/ same comp. limit
- Put my program code and the instances on the Web

79

Better example: evaluation

- Arranged results "in 3D" (n,k) + performance (with special attention to the effect of n, as for scale-up)
- Assessed statistical significance of results
- Found the niche for my_EA:
  - Weak in … cases, strong in - - - cases, comparable otherwise
  - Thereby I answered the "when question"
- Analyzed the specific features and the niches of each algorithm thus answering the "why question"
- Learned a lot about problem X and its solvers
- Achieved generalizable results, or at least claims with well-identified scope based on solid data
- Facilitated reproducing my results → further research

80

Some tips also for your project

- Be organized
- Decide what you want & define appropriate measures
- Choose test problems carefully
- Make an experiment plan (estimate time when possible)
- Perform sufficient number of runs
- Keep all experimental data (never throw away anything)
- Use good statistics ("standard" tools from Web, MS, R)
- Present results well (figures, graphs, tables, …)
- Watch the scope of your claims
- Aim at generalizable results
- Publish code for reproducibility of results (if applicable)
- Publish data for external validation (open science)

81

What do Evolutionary Algorithms offer

- Robust problem solving ability (it performs reasonably well in many different environments, and does not break down in the face of multimodality (multiple optima), discontinuity, noise, and other difficulties)
- Wide applicability in search, optimisation, and machine learning
- Good performance on dynamic problems (e.g. job-shop scheduling)
- Ease of implementation
- Hybridization with other methods
- Anytime problem solving behaviour (an anytime algorithm is an algorithm that can return a valid solution to a problem even if it is interrupted before it ends)
- Easy to run on parallel machines

82

Conclusion

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

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.

1st assignment on EC available today, deadline 17 February

Possible topics for your project

- Evolve deep neural networks architectures
  https://arxiv.org/pdf/1703.00548.pdf
- GECCO 2020 competitions:
  https://gecco-2020.sigevo.org/index.html/Competitions