

# Working with Evolutionary Algorithms

## Lecture 9

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

# 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** (see tutorial on paper writing)

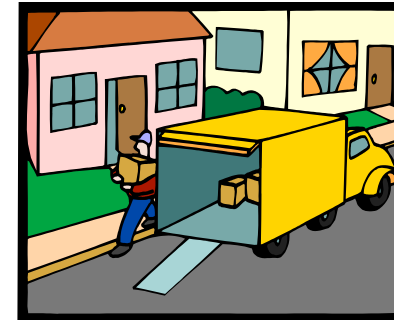
# EA experimentation

- ▶ EA objectives determined by problem context:
- ▶ Design (engineering) problems – single ‘good’ solution required.
- ▶ Control (optimization) problems – requiring many ‘good’ yet ‘timely’ solutions.

# Example: Production Perspective

## ► Optimizing Internet shopping

delivery routes



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

# Example: Design Perspective

- ▶ Optimizing 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*



# Perspectives of an EA's goals

- Design perspective:

find a **very good** solution at least **once**

- Production perspective:

find a **good** solution at **almost every run**

- Academic perspective:

must meet **scientific standards**

These perspectives have very different implications when evaluating EA results.

# Algorithm design

- ▶ Design a representation
- ▶ Design a way of mapping a genotype to a phenotype
- ▶ 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: initialisation method
- ▶ Decide how to stop: termination criterion



# Test problems for experimental comparisons

- ▶ Use problem instances from an academic repository
- ▶ Use randomly generated problem instances
- ▶ Use real life problem instances

# Test problems for experimental comparisons

- ▶ 5 DeJong functions
- ▶ 25 “hard” objective functions
- ▶ Frequently encountered or otherwise important variants of given practical problem
- ▶ Selection from recognised benchmark problem repository (“challenging” by being NP--- ?!)
- ▶ Problem instances made by random generator

Choice has severe implications on

- generalisability and
- scope of the results

# Bad example

- ▶ 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?

# Bad example

- ▶ What did I (my readers) did 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?
  - Can the results be **generalised** ? (Is the tricky GA applicable for other problems? Which ones?)

# Getting Problem Instances 1

- ▶ Testing on **real data**
- ▶ Advantages:
  - Results are application oriented
- ▶ Disadvantages
  - 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 generalise

# Getting Problem Instances 2

- ▶ 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](http://www.ics.uci.edu/~mlearn/MLRepository.html)
- ▶ Advantage:
  - Tried and tested problems and instances (hopefully)
  - Much other work on these → results comparable
- ▶ Disadvantage:
  - Not real – might miss crucial aspect
  - Algorithms get tuned for popular test suites

# Getting Problem Instances 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 systematic investigation of an objective function parameter range
  - Can be shared allowing comparisons with other researchers
- ▶ **Disadvantage:**
  - Not real – might miss crucial aspect
  - Given generator might have hidden bias

# 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 competition limits
- use the same performance measures



# 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
- ▶ ...

# 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

# 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
  - ...

# Performance measures

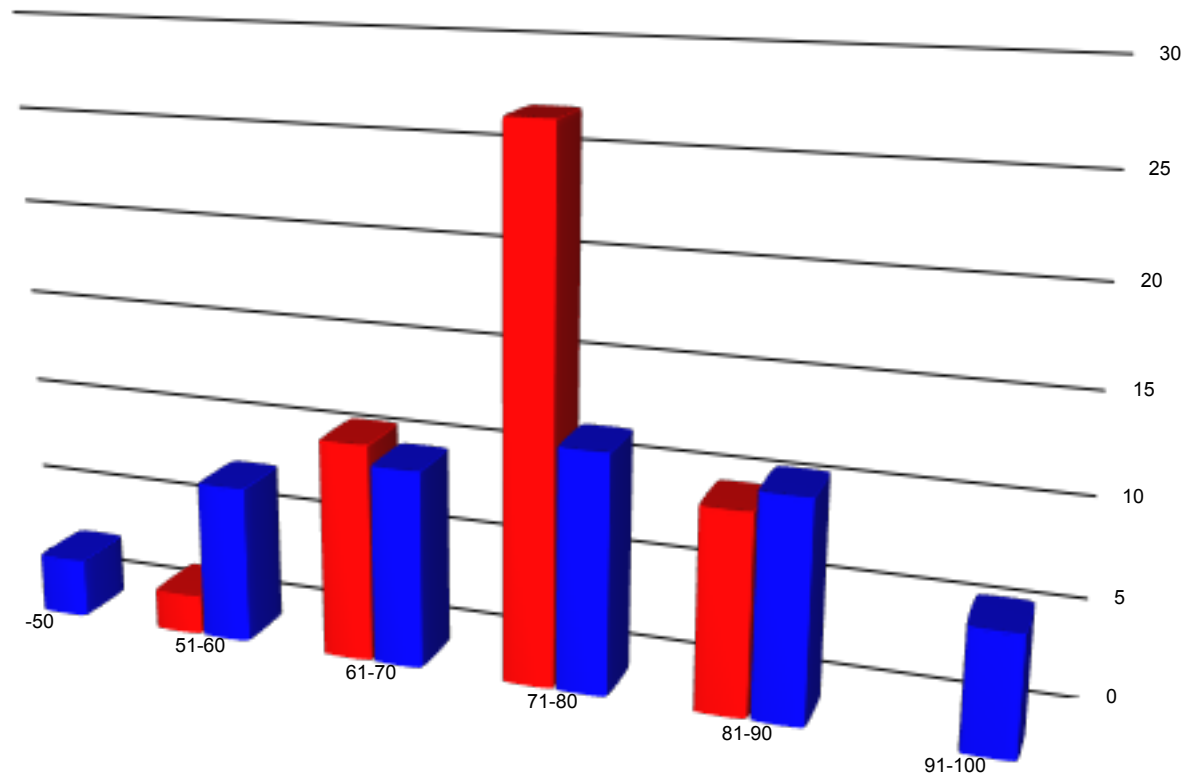
- ▶ No. of generated points in the search space  
= no. of fitness evaluations  
(don't use no. of generations!)
- ▶ **AES: average no. of evaluations to solution**
- ▶ **SR: success rate** = % of runs finding a solution  
(individual with acceptable quality / fitness)
- ▶ **MBF: mean best fitness** at termination, i.e., best per run, mean over a set of runs
- ▶  $SR \neq MBF$ 
  - Low SR, high MBF: good approximator (more time helps?)
  - High SR, low MBF: "Murphy" algorithm

# Fair experiments

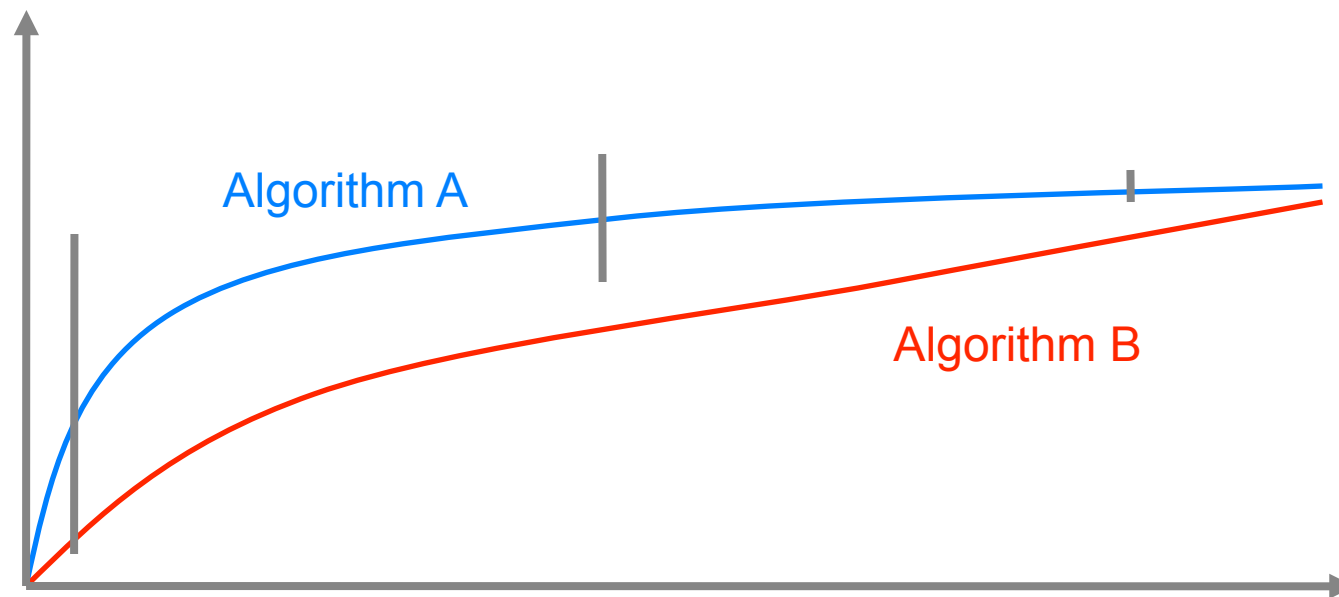
- ▶ Basic rule: use the same computational limit for each competitor
- ▶ Allow each EA the same no. of evaluations, but
  - Beware of hidden labour, e.g. in heuristic mutation operators
  - Beware of possibly fewer evaluations by smart operators
- ▶ EA vs. heuristic: allow the same no. of steps:
  -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

# Example: off-line performance measure evaluation

Which algorithm  
is better?  
Why?  
When?



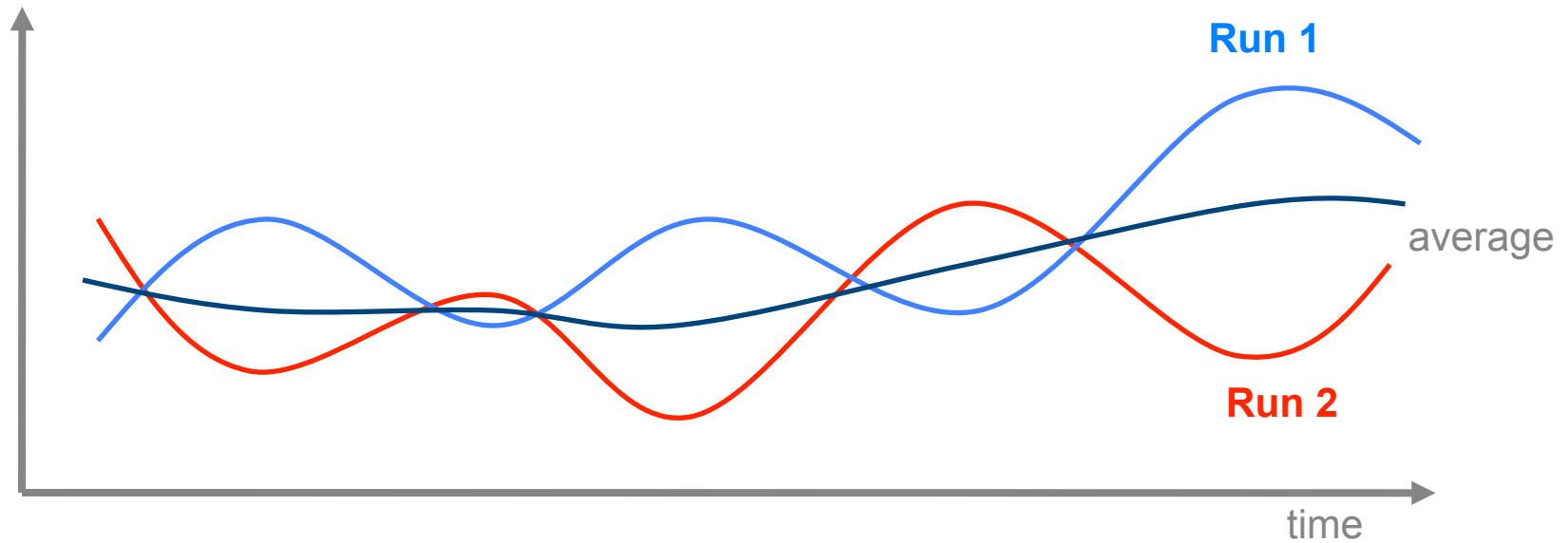
# Example: on-line performance measure evaluation



Populations mean (best) fitness

Which algorithm is better? Why? When?

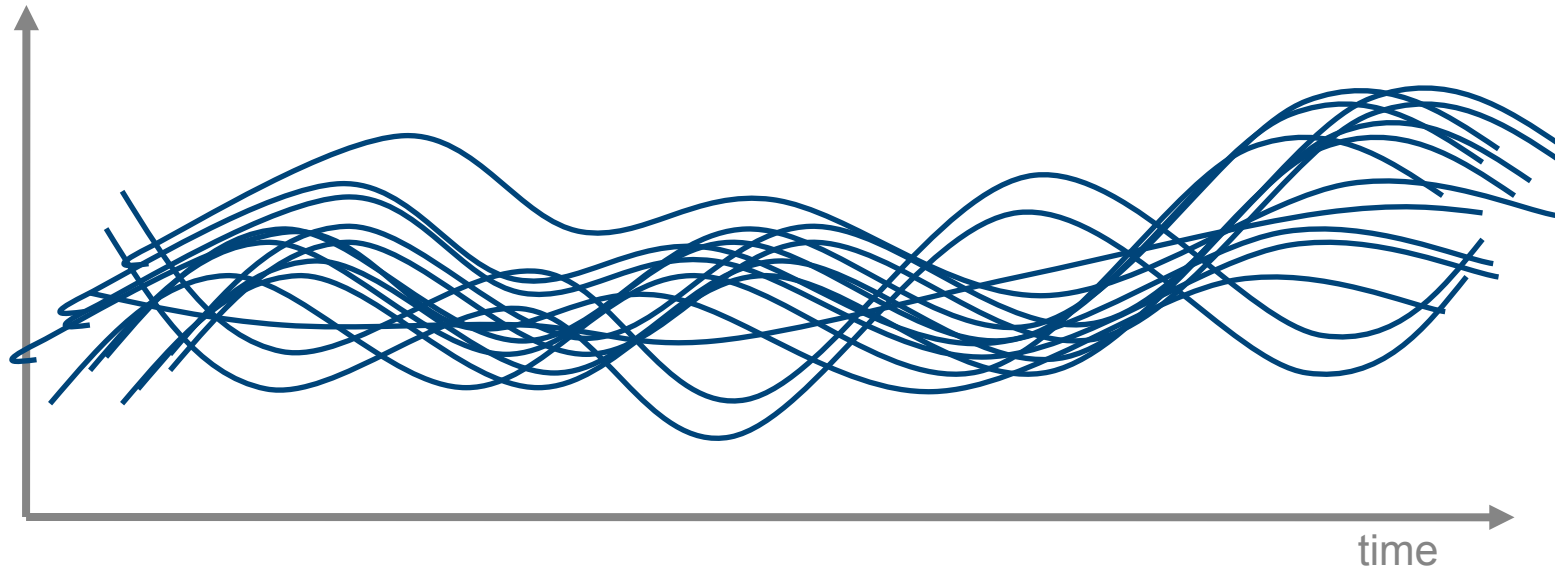
# Example: averaging on-line measures



Averaging can “choke” interesting information



# Example: overlaying on-line measures



Overlay of curves can lead to very “cloudy” figures

# Statistical Comparisons and Significance



- ▶ Algorithms are stochastic
- ▶ Results have element of “luck”
- ▶ Sometimes can get away with less rigour – e.g. parameter tuning
- ▶ For scientific papers where a claim is made: “Newbie recombination is better than uniform crossover”, need to show statistical significance of comparisons

# Example

Trial	Old Method	New Method
1	500	657
2	600	543
3	556	654
4	573	565
5	420	654
6	590	712
7	700	456
8	472	564
9	534	675
10	512	643
Average	545.7	612.3

Is the new method better?

# Example (cont'd)

Trial	Old Method	New Method
1	500	657
2	600	543
3	556	654
4	573	565
5	420	654
6	590	712
7	700	456
8	472	564
9	534	675
10	512	643
Average	545.7	612.3
SD	73.5962635	73.5473317
T-test	<b>0.07080798</b>	

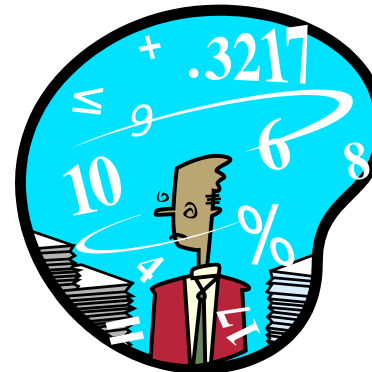
- Standard deviations supply additional info
- T-test (and alike) indicate the chance that the values came from the same underlying distribution (difference is due to random effects) E.g. with 7% chance in this example.

# Statistical tests

- ▶ T-test assumes:
  - Data taken from continuous interval or close approximation
  - Normal distribution
  - Similar variances for too few data points
  - Similar sized groups of data points
- ▶ Other tests:
  - Wilcoxon – preferred to t-test where numbers are small or distribution is not known.
  - F-test – tests if two samples have different variances.

# Statistical Resources

- ▶ <http://fonsg3.let.uva.nl/Service/Statistics.html>
- ▶ <http://department.obg.cuhk.edu.hk/ResearchSupport/>
- ▶ <http://faculty.vassar.edu/lowry/webtext.html>
- ▶ R, Mathematica
- ▶ <http://www.octave.org/>



# 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

# 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 (AES for benchmark?)
- ▶ Put my program code and the instances on the Web



# 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”
- ▶ Analysed the specific features and the niches of each algorithm thus answering the “why question”
- ▶ Learned a lot about problem X and its solvers
- ▶ Achieved generalisable results, or at least claims with well-identified scope based on solid data
- ▶ Facilitated reproducing my results → further research

# Some tips

- ▶ **Be organised**
- ▶ 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)
- ▶ Present results well (figures, graphs, tables, ...)
- ▶ Watch the scope of your claims
- ▶ Aim at generalisable results
- ▶ Publish code for reproducibility of results (if applicable)