UiO : Department of Informatics
University of Oslo

INF3490 - Biologically inspired computing

Lecture 4: Eiben and Smith,

Working with evolutionary algorithms (chpt 9)
Hybrid algorithms (chpt 10)
Multi-objective optimization (chpt 12)

Kai Olav Ellefsen

Key points from last time (1/3)

- Selection pressure
- Parent selection:
  - Fitness proportionate
  - Rank-based
  - Tournament selection
  - Uniform selection
- Survivor selection
  - Age-based vs fitness based
  - Elitism

Key points from last time (2/3)

- Diversity maintainance:
  - Fitness sharing
  - Crowding
  - Speciation
  - Island models

Key points from last time (3/3)

Name	Representation	Crossover	Mutation	Parent selection	Survivor selection	Specialty
Simple Genetic Algorithm	Binary vector	1-point crossover	Bit flip	Fitness proportional	Generational replacement	None
Evolution Strategies	Real-valued vector	Discrete or intermediate recombination	Gaussian	Random draw	Best N	Strategy parameters
Evolutionary Programming	Real-valued vector	None	Gaussian	One child each	Tournament	Strategy parameters
Genetic Programming	Tree	Swap sub-tree	Replace sub-tree	Usually fitness proportional	Generational replacement	None

Chapter 9: Working with Evolutionary Algorithms

1. **Types of problem**
2. Algorithm design
3. Measurements and statistics
4. Test problems
5. Some tips and summary

Main Types of Problem we Apply EAs to

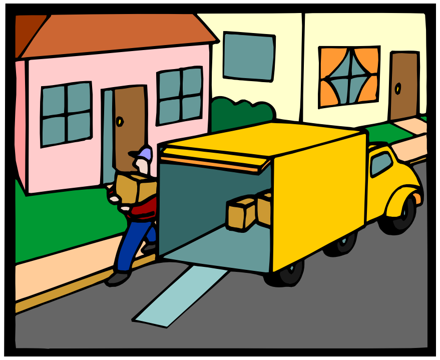
- Design (one-off) problems
- Repetetive problems
  - Special case: On-line control
- Academic Research

Example Design Problem

- 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***

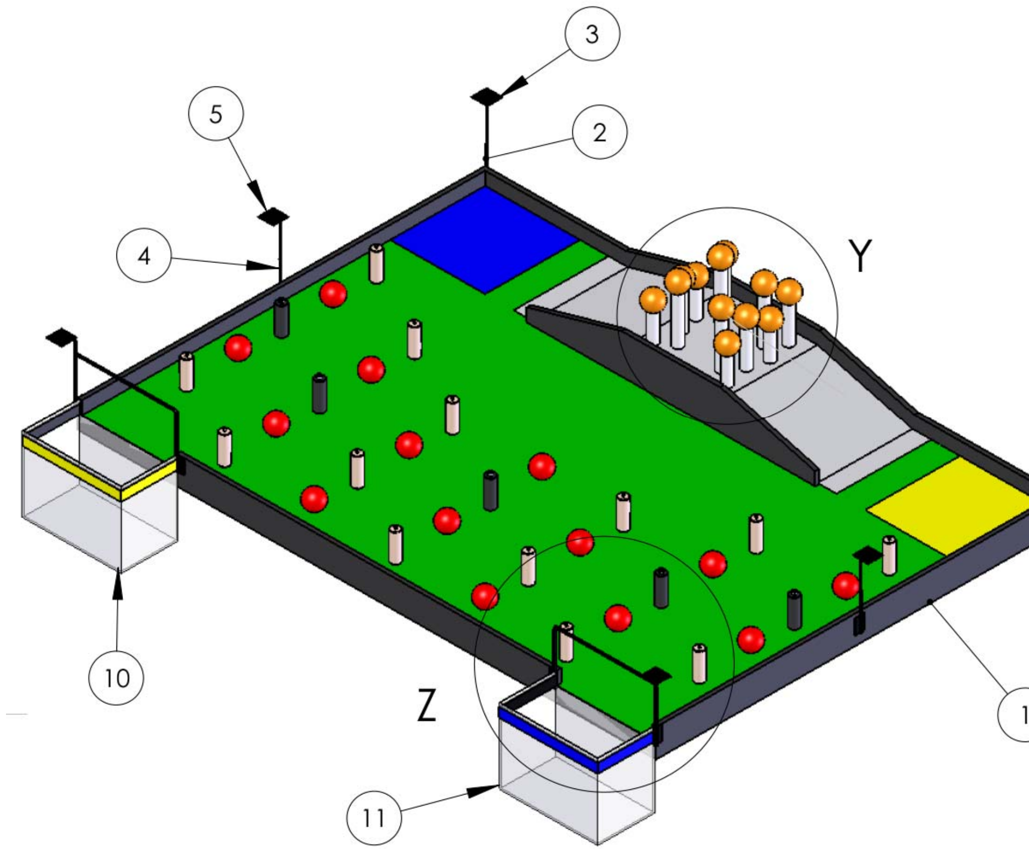
Example Repetitive Problem

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



Example On-Line Control Problem

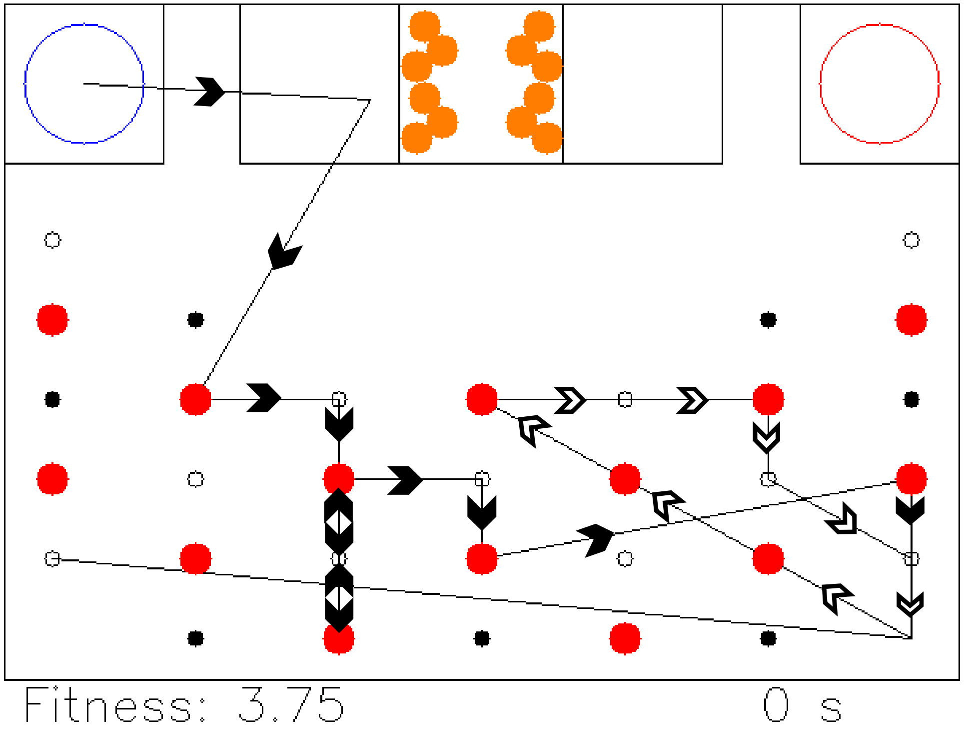
- Robotic competition
- Goal: Gather more resources than the opponent
- Evolution optimizes strategy before *and during* competition

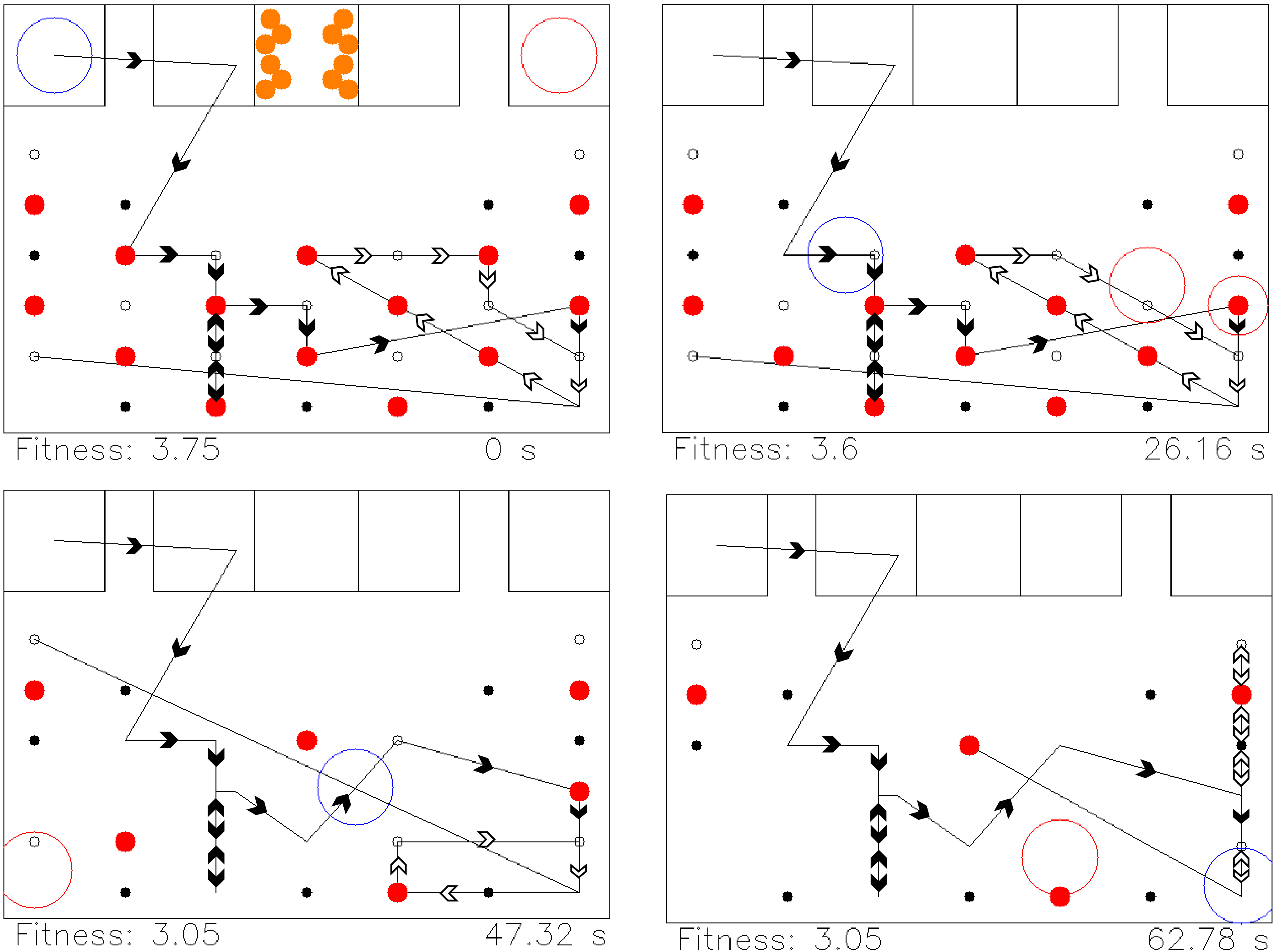


Example On-Line Control Problem

Example On-Line Control Problem

- Representation:
  Array of object IDs:
  [1 5 7 34 22 ….]
- Fitness test:
  Simulates rest of match, calculating our score (num. harvested resources)

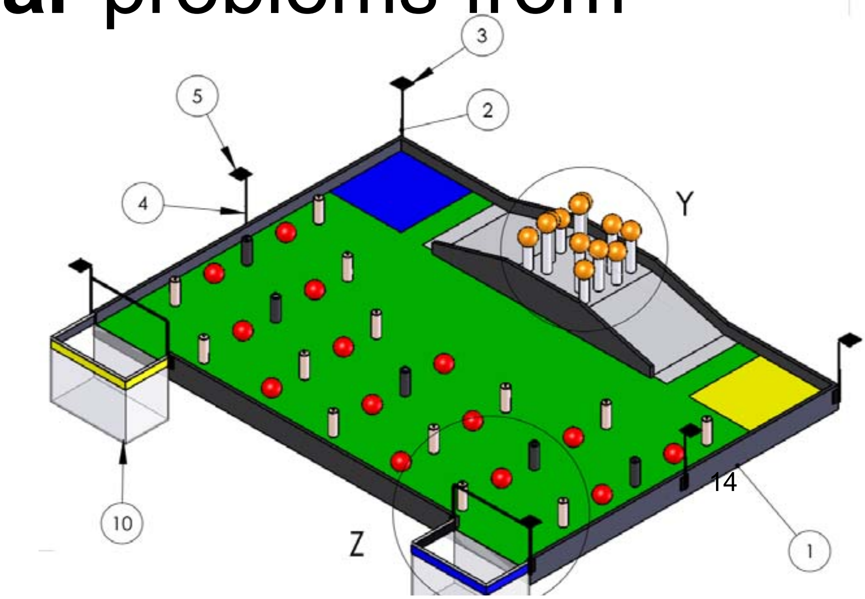




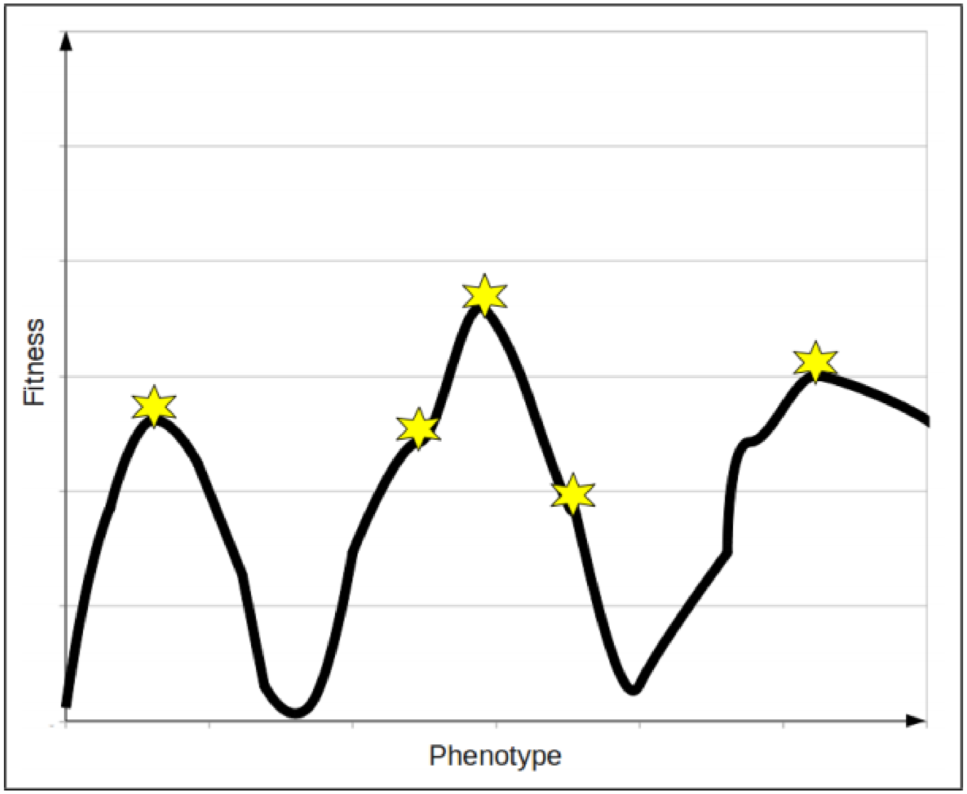
Fitness: 3.75
0 s
Fitness: 3.6
26.16 s
Fitness: 3.05
47.32 s
Fitness: 3.05
62.78 s

On-Line Control

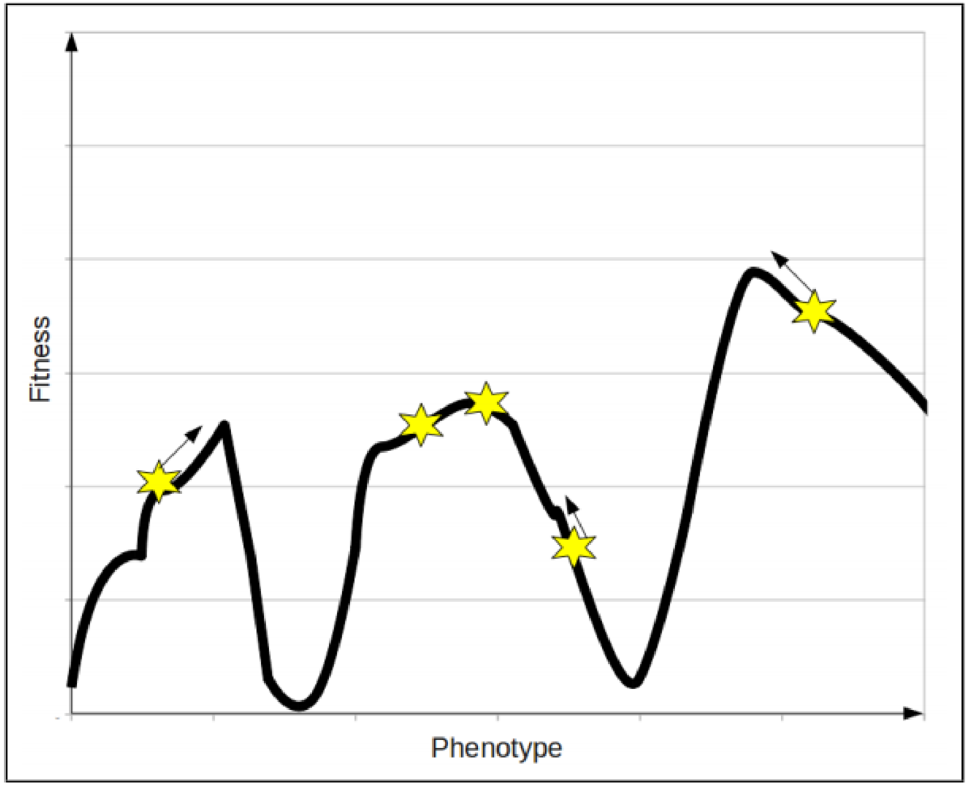
- Needs to **run regularly/repetitively**
- **Limited time** to run algorithm
- Must *always* deliver ***reasonably* good** solution in **limited time**
- Requires **relatively similar** problems from one timestep to the next



Why we require similar problems: Effect of changes on fitness landscape



Before environmental change



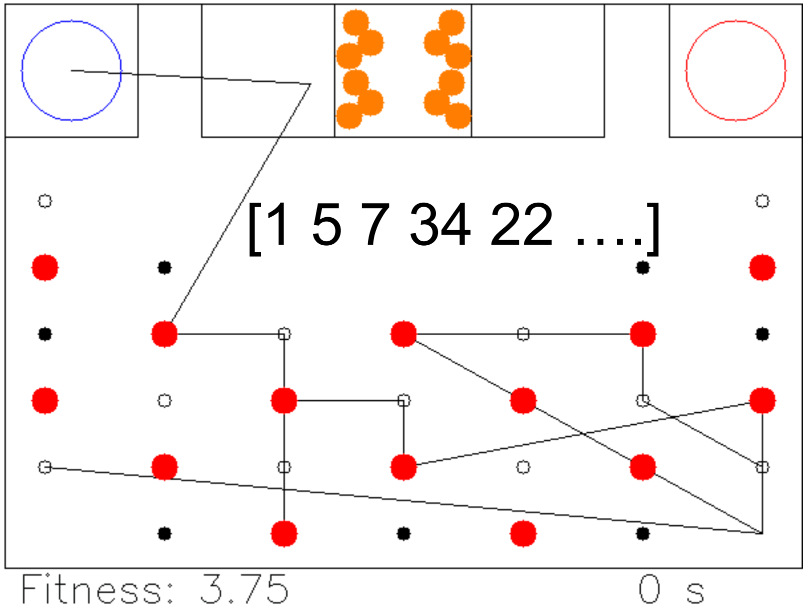
After environmental change

Goals for Academic Research on EAs

- Show that EC is applicable in a **(new) problem domain** (real-world applications)
- Show that *my_EA* is **better than** *benchmark_EA*
- Show that EAs outperform **traditional** algorithms
- Optimize or study **impact of parameters** on the performance of an EA
- Investigate **algorithm behavior** (e.g. interaction between selection and variation)
- See how an EA **scales-up** with problem size
- …

Working with Evolutionary Algorithms

1. Types of problem
2. **Algorithm design**
3. Measurements and statistics
4. Test problems
5. Some tips and summary



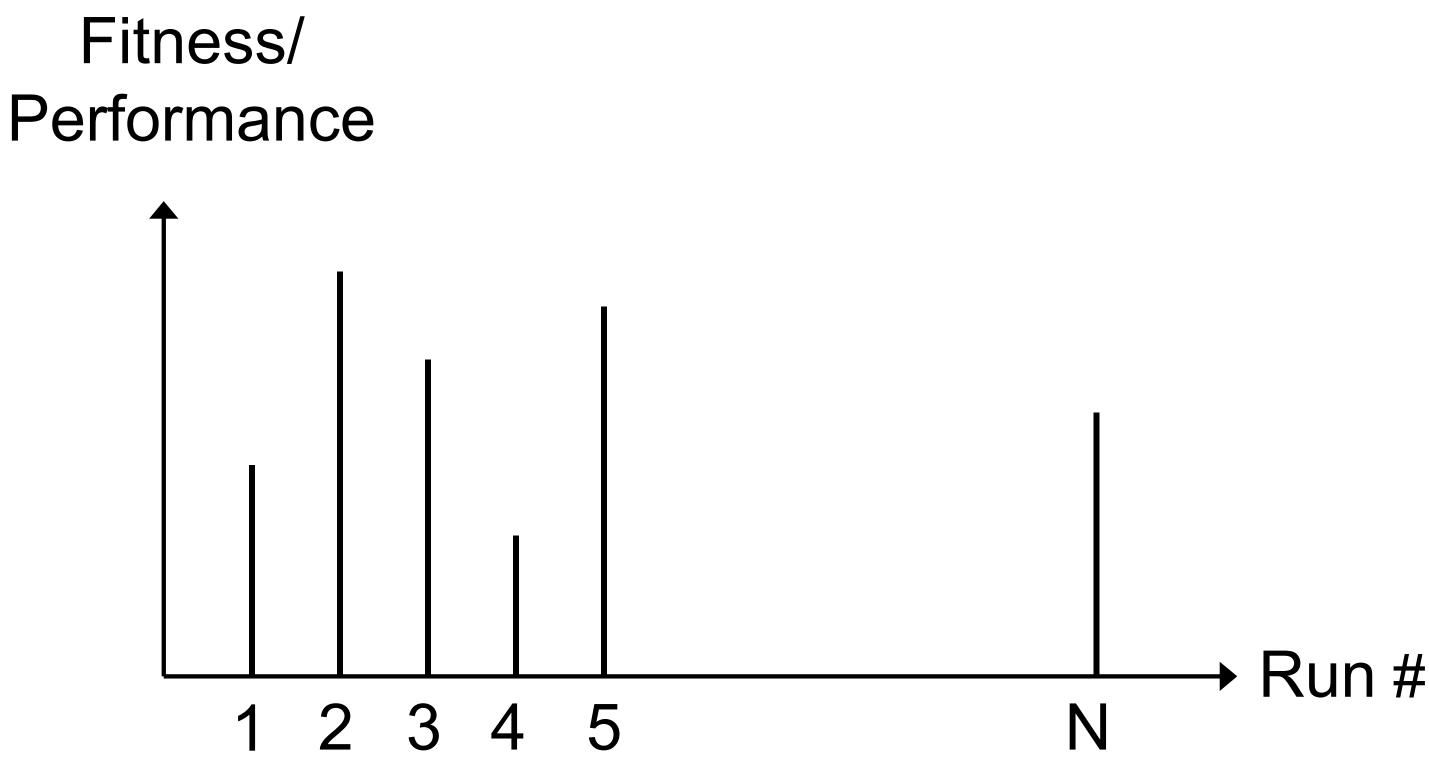
Algorithm design

- 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: initialization method
- Decide how to stop: termination criterion

Working with Evolutionary Algorithms

1. Types of problem
2. Algorithm design
3. **Measurements and statistics**
4. Test problems
5. Some tips and summary

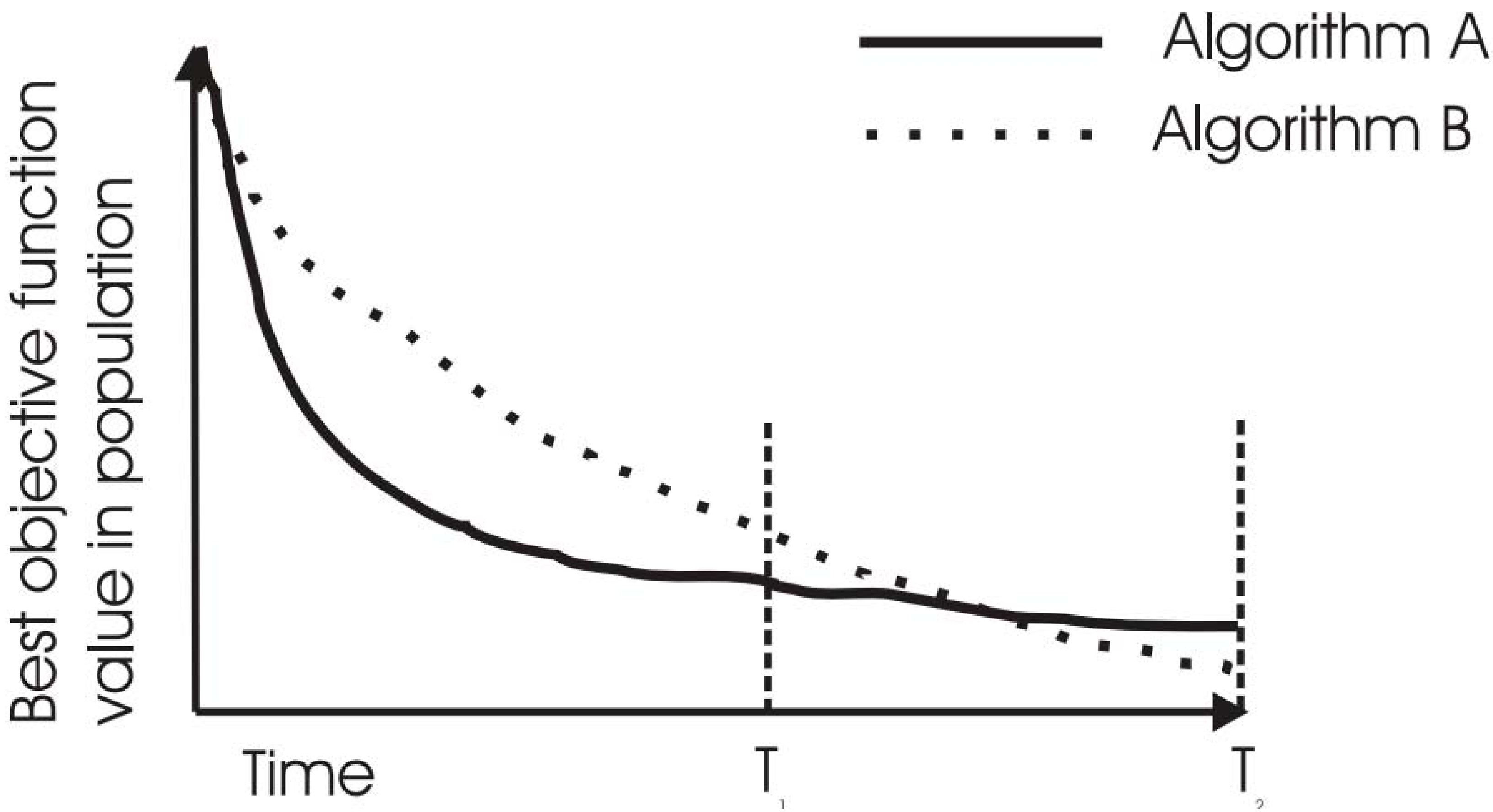
Typical Results from Several EA Runs



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 (to cope with turtle/hare effect)
  - use the same performance measures

Turtle/hare effect

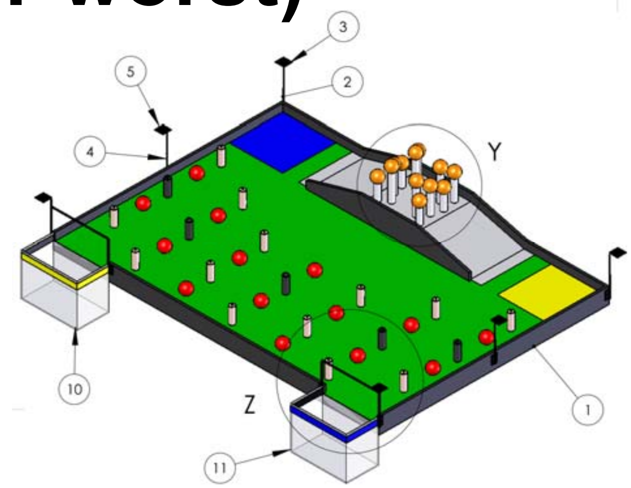


How to Compare EA Results?

- **Success Rate**: Proportion of runs within x% of **target**
- **Mean Best Fitness:** Average best solution over *n* runs
- Best result ("Peak performance") over *n* runs
- Worst result over *n* runs

Peak vs Average Performance

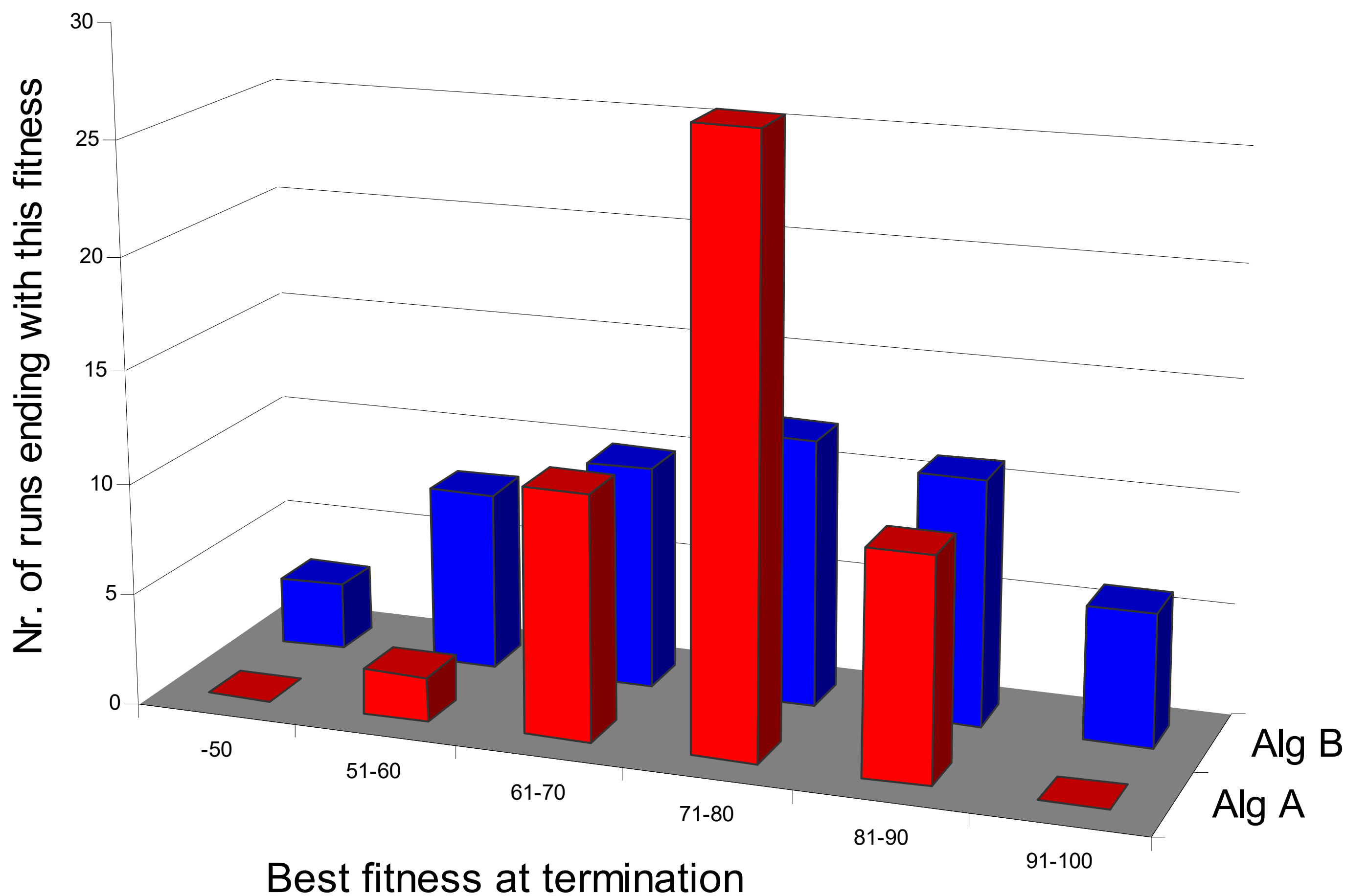
- For repetitive tasks, **average (or worst) performance** is most relevant



- For design tasks, **peak performance** is most relevant

Example: off-line performance measure evaluation

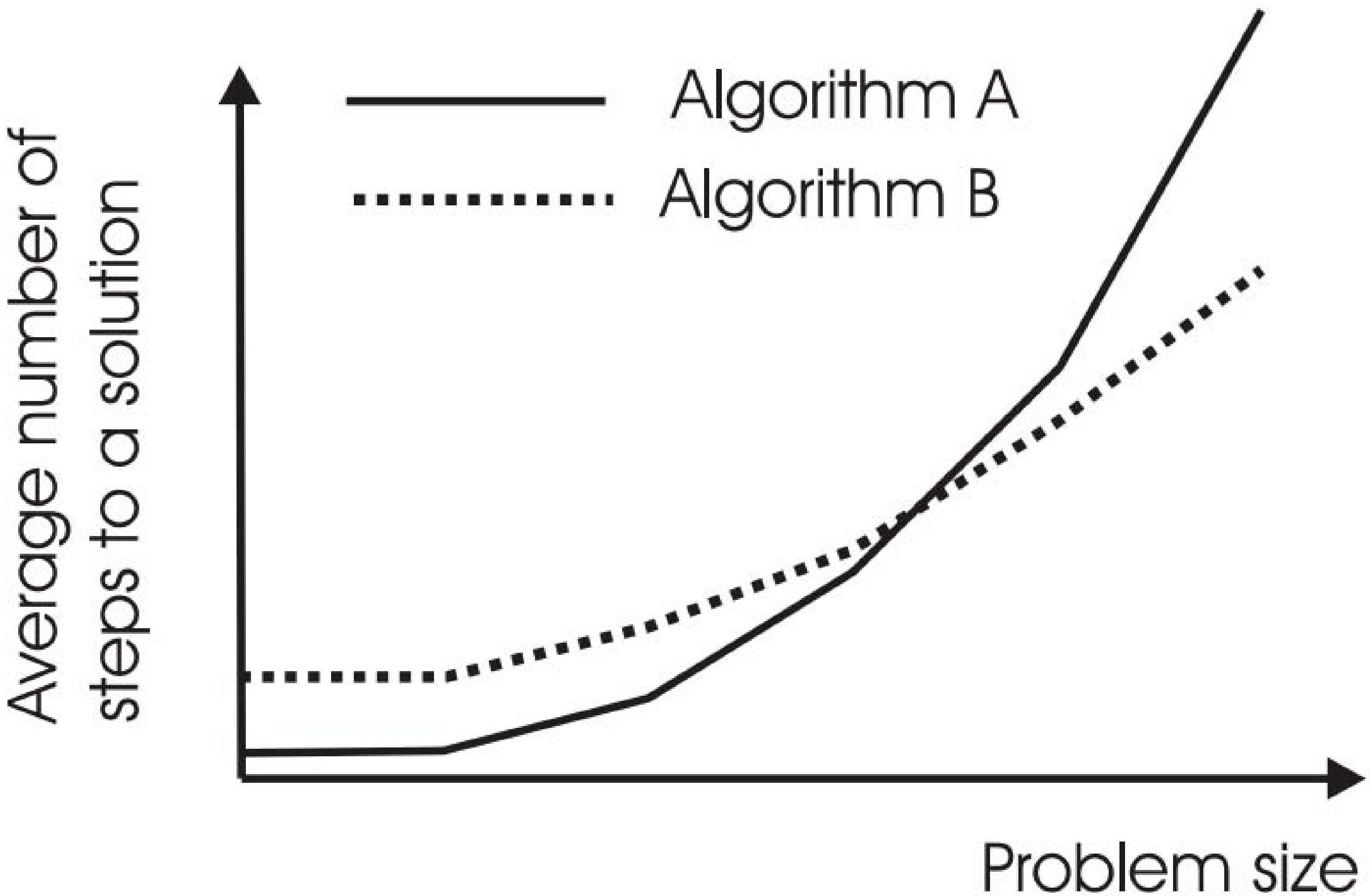
Which algorithm is better? Why? When?



Measuring Efficiency: What time units do we use?

- Elapsed time?
  - Depends on computer, network, etc…
- CPU Time?
  - Depends on skill of programmer, implementation, etc…
- Generations?
  - Incomparable when parameters like population size change
- Evaluations?
  - Other parts of the EA (e.g. local searches) could “hide” computational effort.
  - Some evaluations can be faster/slower (e.g. memoization)
  - Evaluation time could be small compared to other steps in the EA (e.g. genotype to phenotype translation)

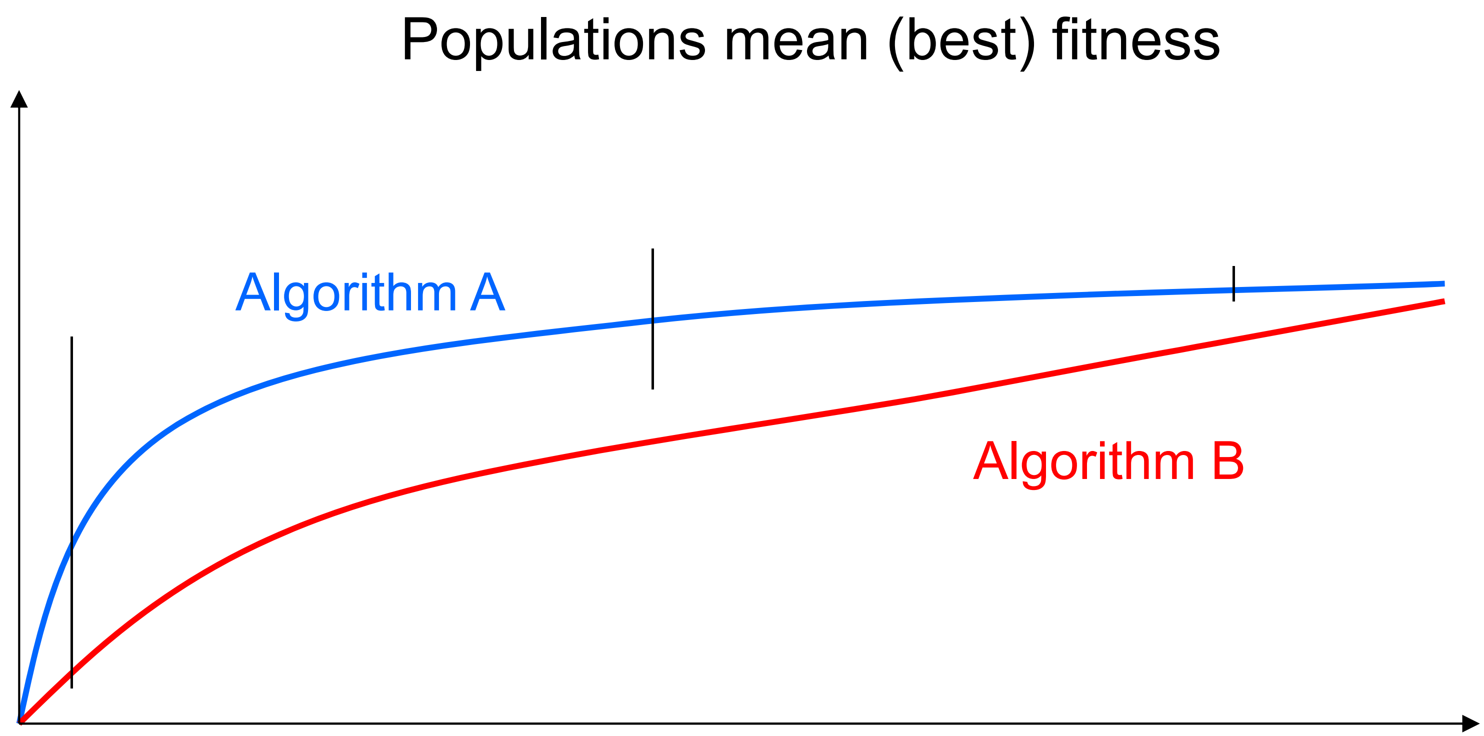
Scale-up Behavior



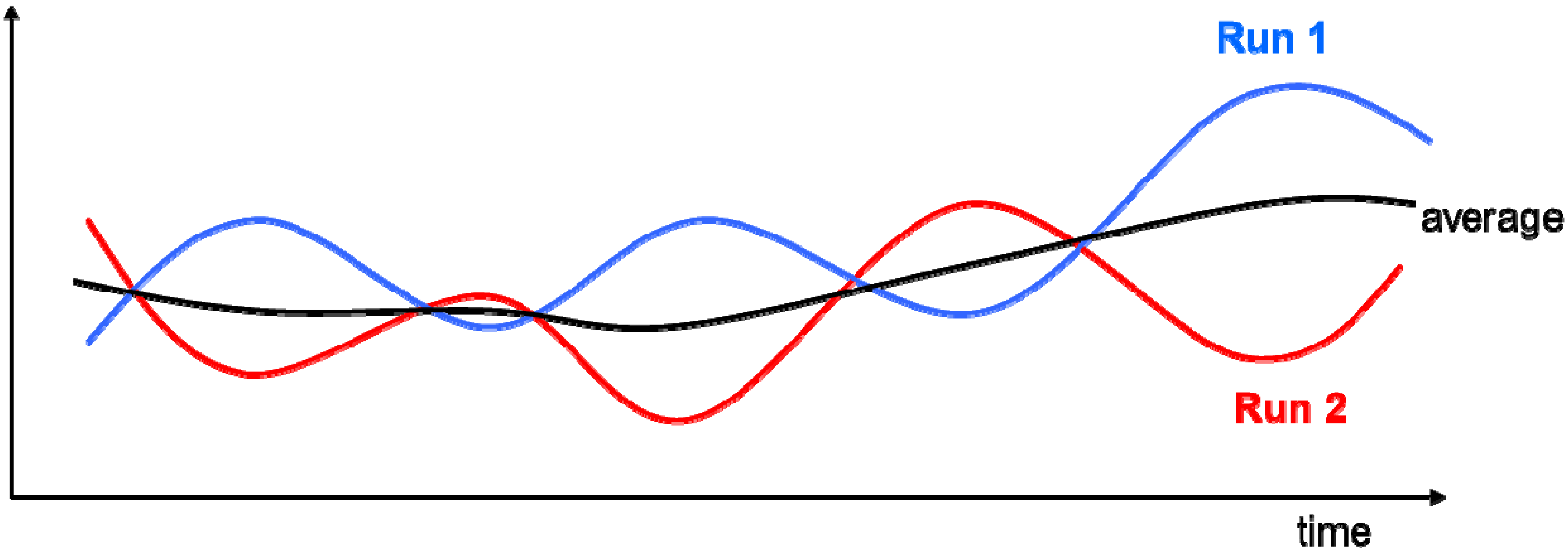
Measures

- **Performance measures (off-line)**
  - **Efficiency** (alg. speed, also called performance)
    - Execution time
    - Average no. of evaluations to solution (AES, i.e., number of generated points in the search space)
  - **Effectiveness** (solution quality, also called accuracy)
    - Success rate (SR): % of runs finding a solution
    - Mean best fitness at termination (MBF)
- **“Working” measures (on-line)**
  - Population distribution (genotypic)
  - Fitness distribution (phenotypic)
  - Improvements per time unit or per genetic operator
  - …

Example: on-line performance measure evaluation

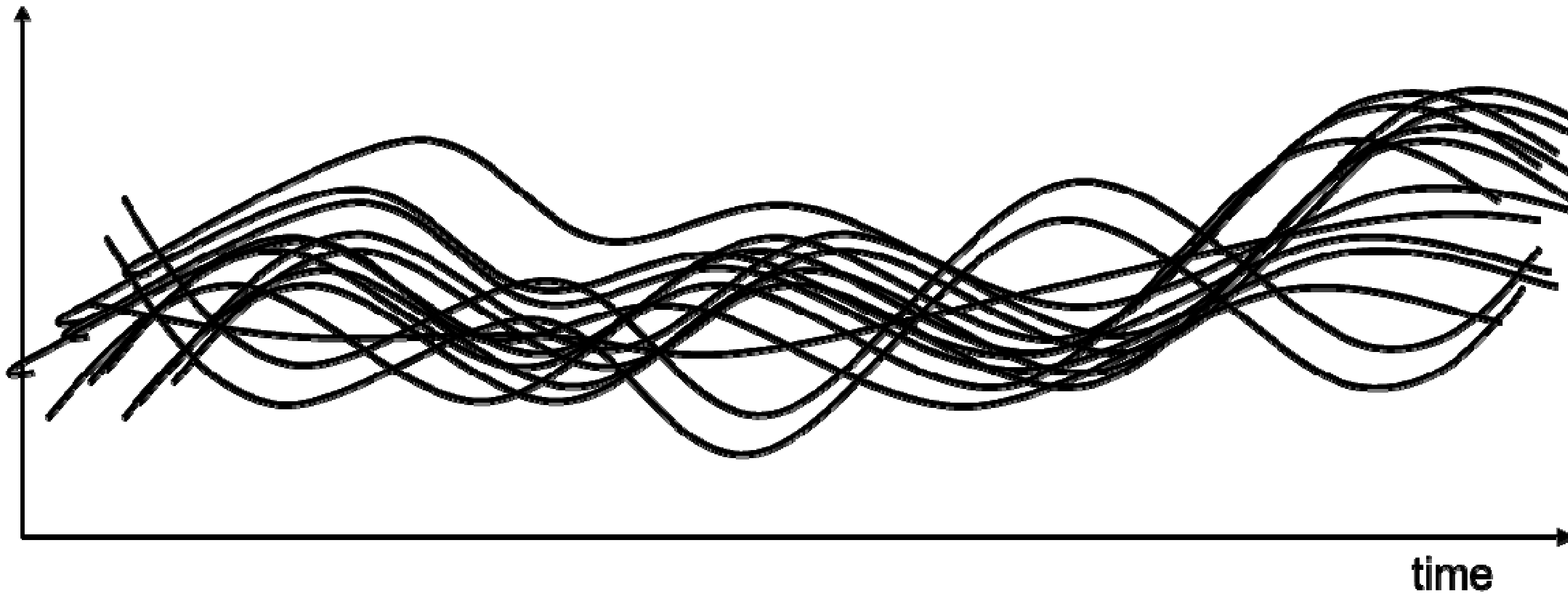


Example: averaging on-line measures



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”
- If a claim is made “Mutation A is better than mutation B”, need to show **statistical significance** of comparisons
- Fundamental problem: two series of samples (random drawings) from the SAME distribution may have DIFFERENT averages and standard deviations
- Tests can show if the differences are significant or not

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	**0.07080798**	

- 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.

Working with Evolutionary Algorithms

1. Types of problem
2. Algorithm design
3. Measurements and statistics
4. **Test problems**
5. Some tips and summary

Where to Find Test Problems for an EA?

1. Recognized **benchmark problem** repository (typically “challenging”)
2. Problem instances made by **random generator**
3. Frequently encountered or otherwise important variants of given **real-world problems**

Choice has severe implications on:

- generalizability and
- scope of the results

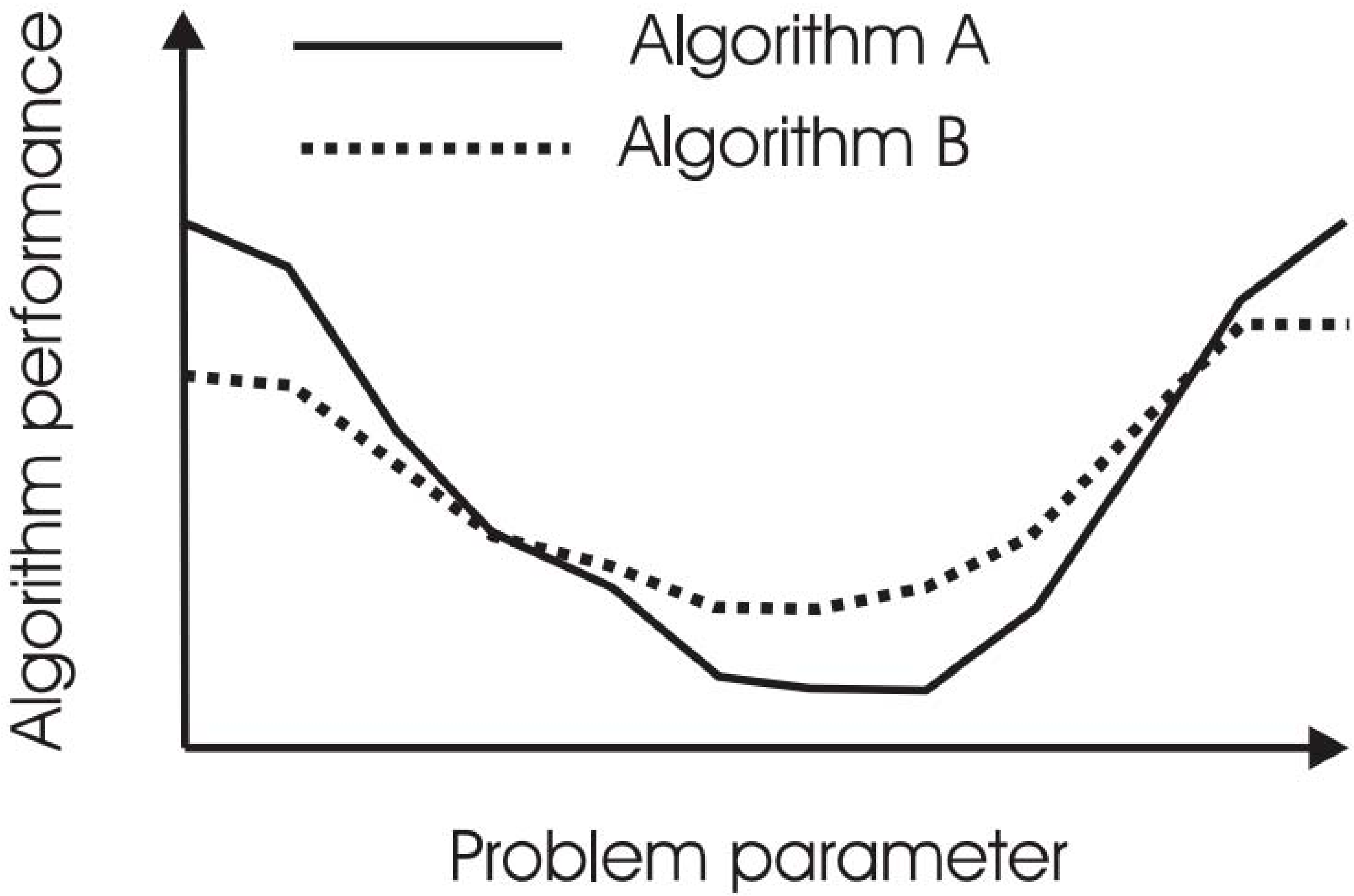
Getting Problem Instances (1/4) Benchmarks

- Standard data sets in problem **repositories**, e.g.:
  - OR-Library
    www.brunel.ac.uk/~mastjjb/jeb/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:
  - Not real – might miss crucial aspect
  - Algorithms get tuned for popular test suites

Getting Problem Instances (2/4) Problem instance generators

- **Problem instance generators** produce simulated data for given parameters, e.g.:
  - GA/EA Repository of Test Problem Generators
    http://vlsicad.eecs.umich.edu/BK/Slots/cache/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

Getting Problem Instances (3/4)
Problem instance generators



Getting Problem Instances (4/4)
Real-world problems

- Testing on (own collected) **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

Working with Evolutionary Algorithms

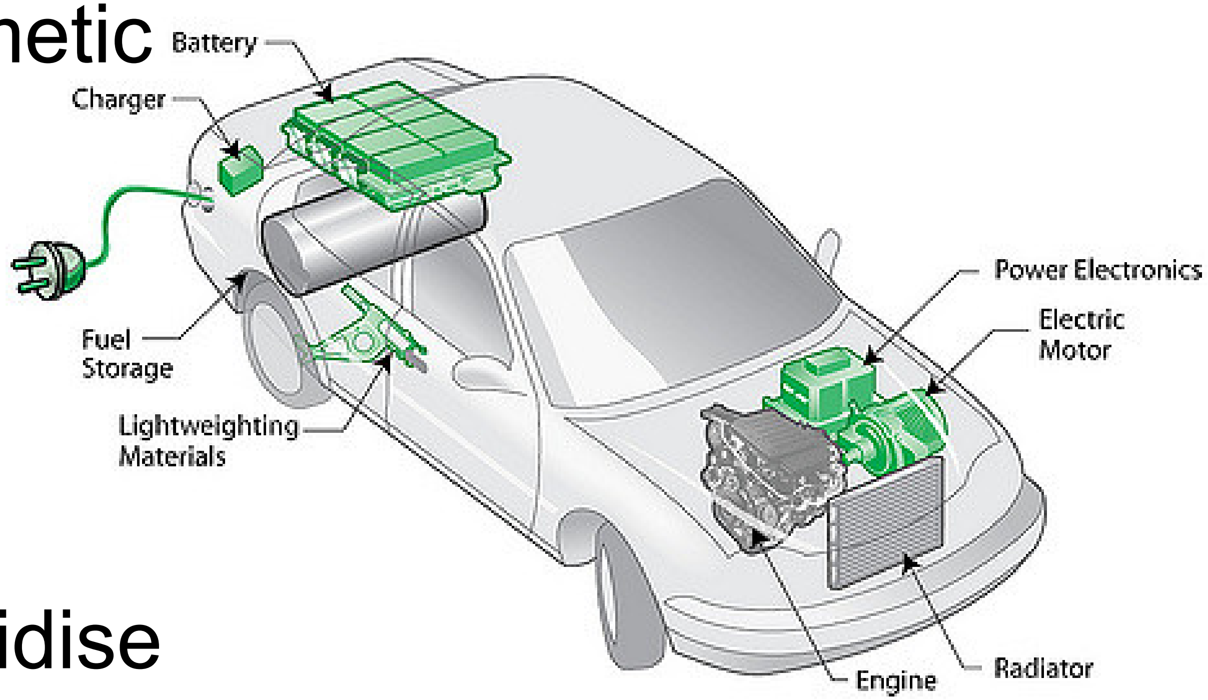
1. Types of problem
2. Algorithm design
3. Measurements and statistics
4. Test problems
5. **Some tips and summary**

Summary of tips for experiments

- **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)
- Include in publications all necessary parameters to make **others able to repeat** your experiments
- 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)

Chapter 10: Hybridisation with Other Techniques: Memetic Algorithms

1. Why Hybridise?
2. What is a Memetic Algorithm?
3. Local Search
   – Lamarckian vs. Baldwinian adaptation
4. Where to hybridise



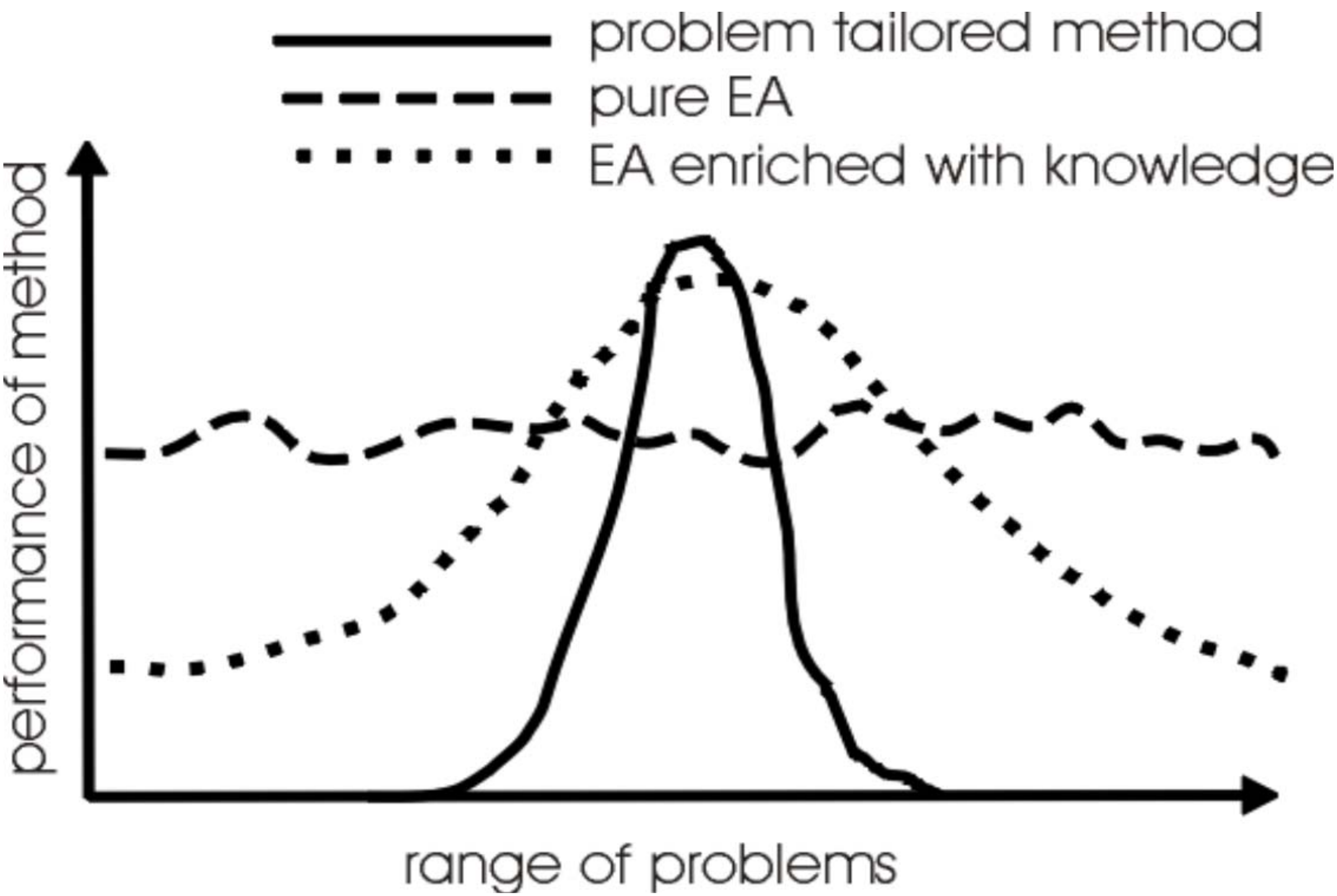
1. Why Hybridise

- Might be looking at **improving on existing techniques** (non-EA)
- Might be looking at **improving EA search** for good solutions

1. Why Hybridise: One-Max Example

- The One-Max problem: maximize the number of 1's in a binary string: [1 0 0 1 0 1 … 1]
- A GA gives rapid progress initially, but very slow towards the end
- Integrating a local search in the EA speeds things up

1. Why Hybridise
Michalewicz's view on EAs in context

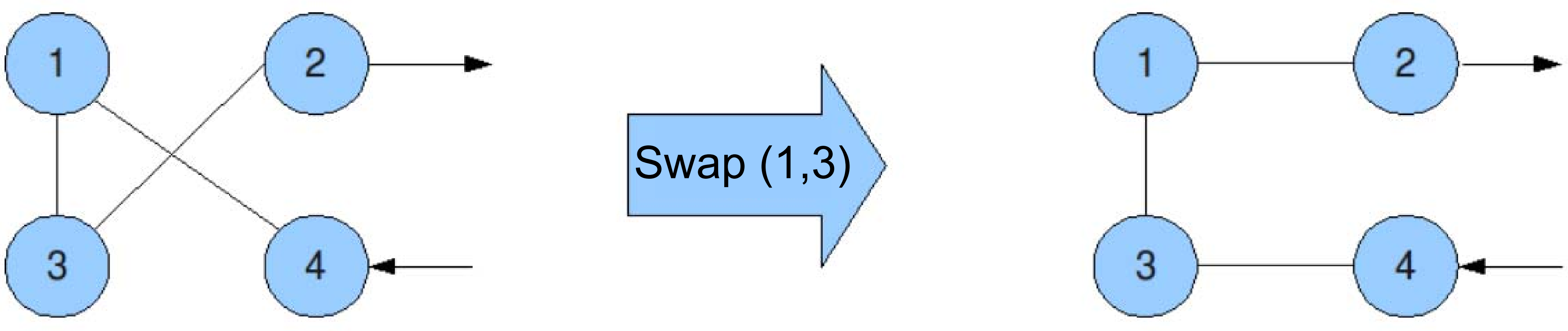


2. What is a Memetic Algorithm?

- The combination of Evolutionary Algorithms with **Local Search Operators** that work within the EA loop has been termed “**Memetic Algorithms**”
- Term also applies to EAs that use **instance-specific knowledge**
- Memetic Algorithms have been shown to be orders of magnitude **faster and more accurate** than EAs on some problems, and are the “state of the art” on many problems

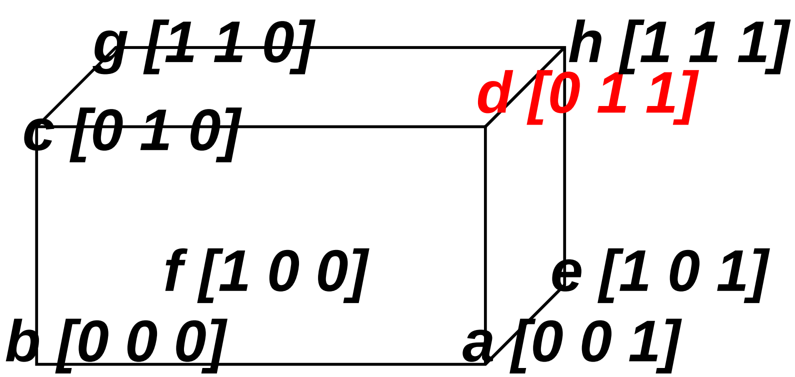
3. Local Search: Main Idea (simplified)

- Make a small, but intelligent (problem-specific), change to an existing solution
- If the change improves it, keep the improved version
- Otherwise, keep trying small, smart changes until it improves, or until we have tried all possible small changes



3. Local Search: Local Search

- Defined by combination of ***neighbourhood*** and ***pivot rule***
- *N(x)* is defined as the set of points that can be reached from *x* with one application of a move operator
  - e.g. bit flipping search on binary problems



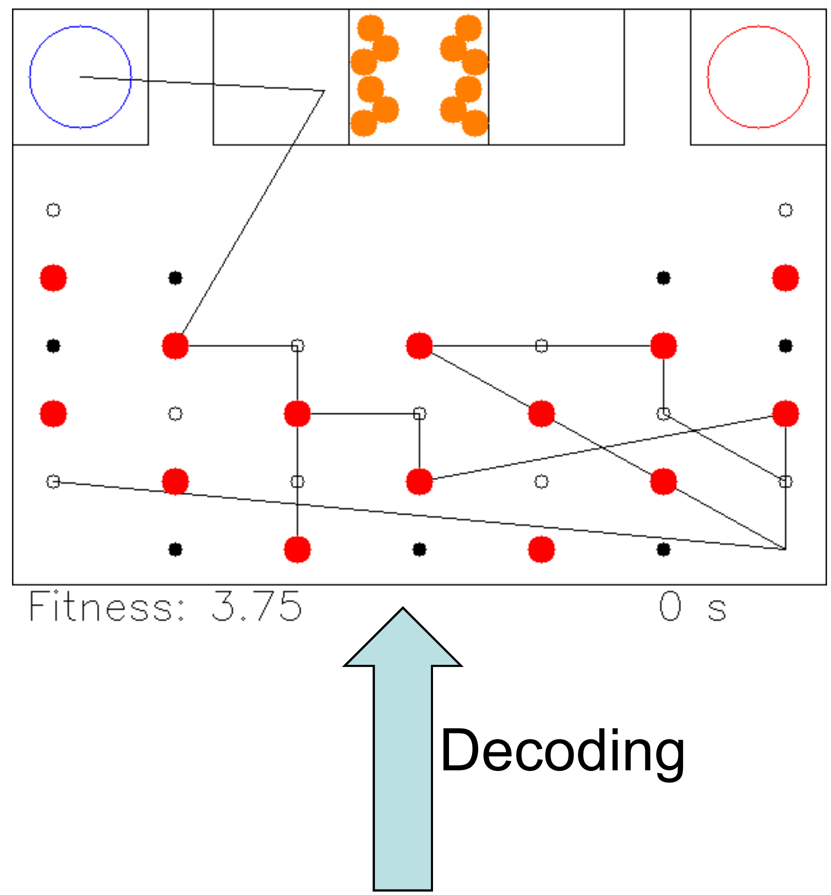
N(d) = {a,c,h}

3. Local Search:
Pivot Rules

- Is the neighbourhood searched randomly, systematically or exhaustively ?
- does the search stop as soon as a fitter neighbour is found (***Greedy Ascent***)
- or is the whole set of neighbours examined and the best chosen (***Steepest Ascent***)
- of course there is no one best answer, but some are quicker than others to run ....

3. Local Search: Example

- Genotype: Array of integers
- Greedy local search:
  - Select N random pairs of integers (u, v)
  - Test swapping u and v
  - If a swap gives better plan: Return new plan
  - Else: Move to next (u,v)



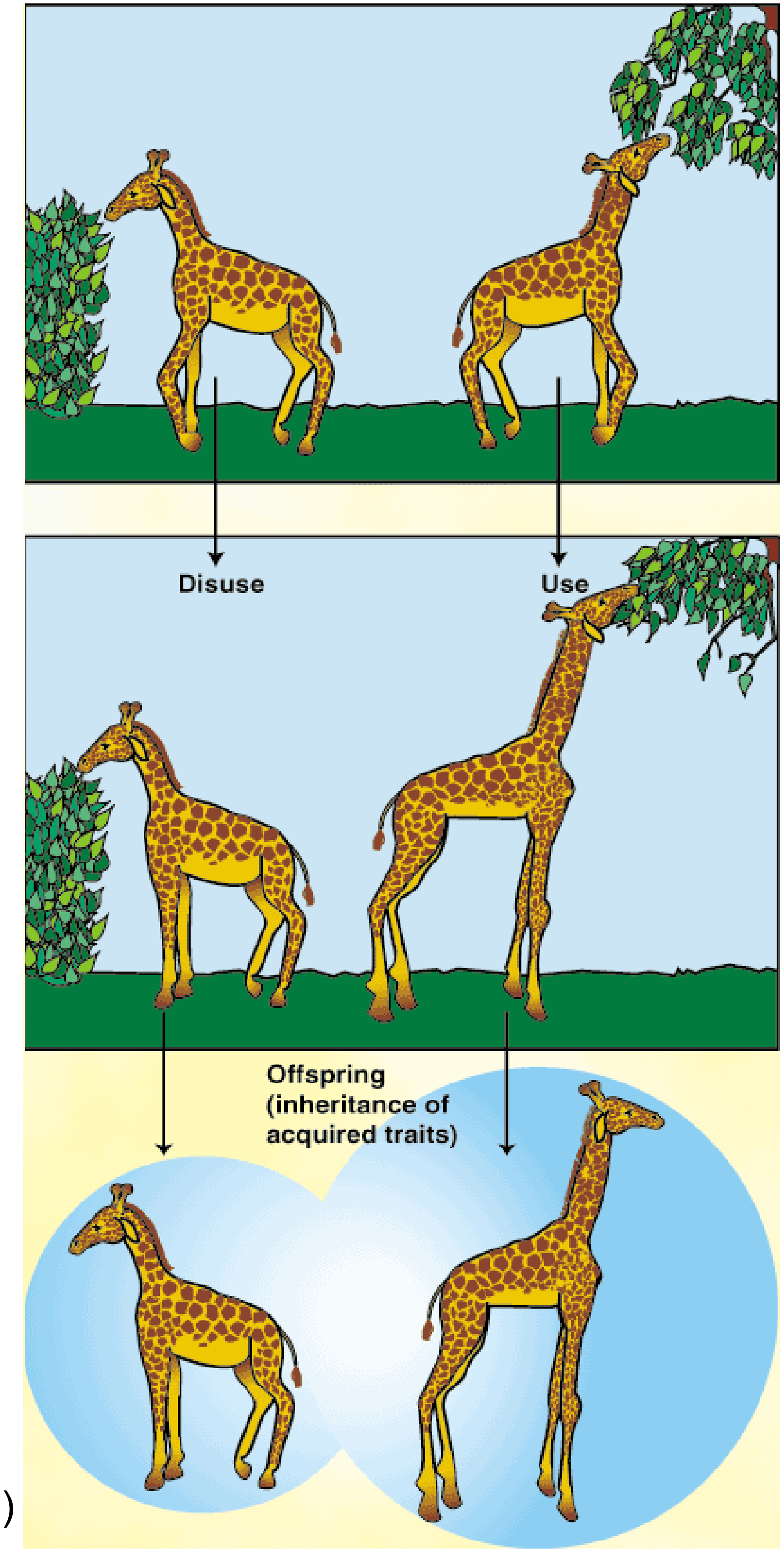
[1 5 7 34 22 ….]

4. Local Search and Evolution

- Do offspring inherit what their parents have "learnt" in life?
  - Yes - Lamarckian evolution
    - Improved fitness and genotype
  - No - Baldwinian evolution
    - Improved fitness only

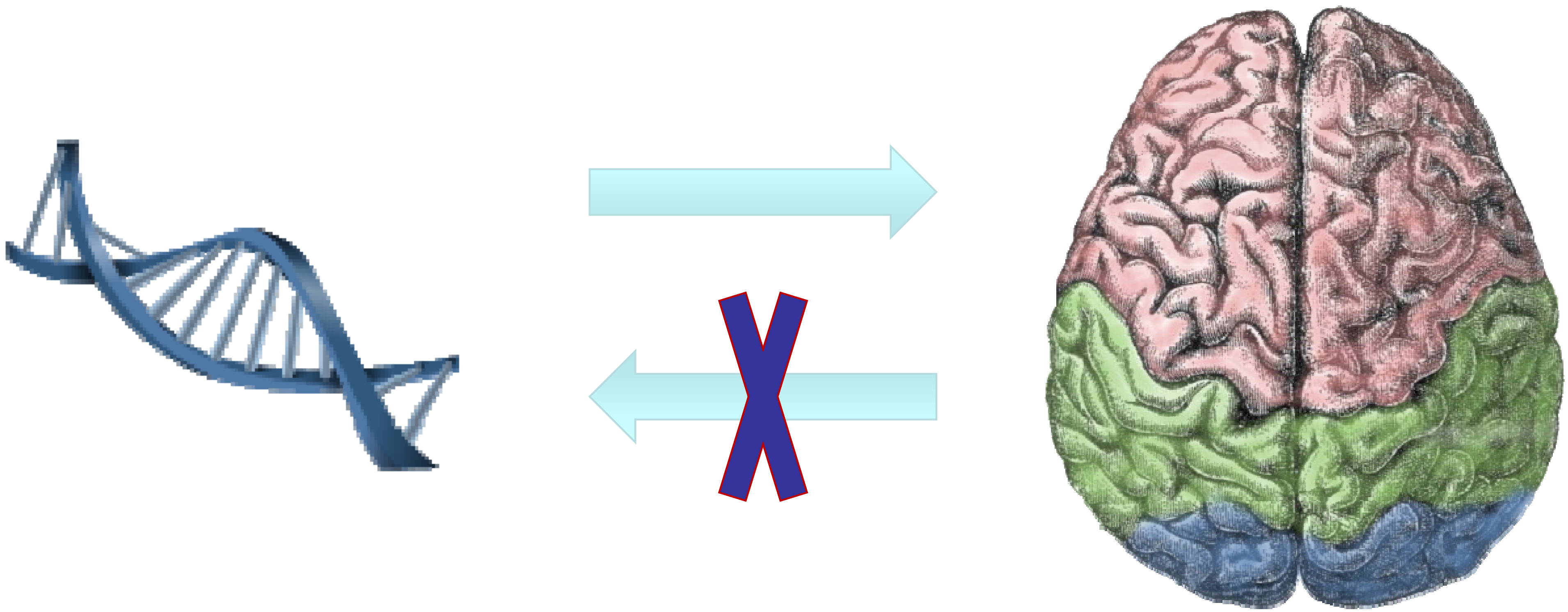
4. Lamarckian Evolution

- Lamarck, 1809: Traits acquired in parents' lifetimes can be inherited by offspring
- This type of direct inheritance of acquired traits is not possible, according to modern evolutionary theory



(Image from sparknotes.com)

4. Inheriting Learned Traits?

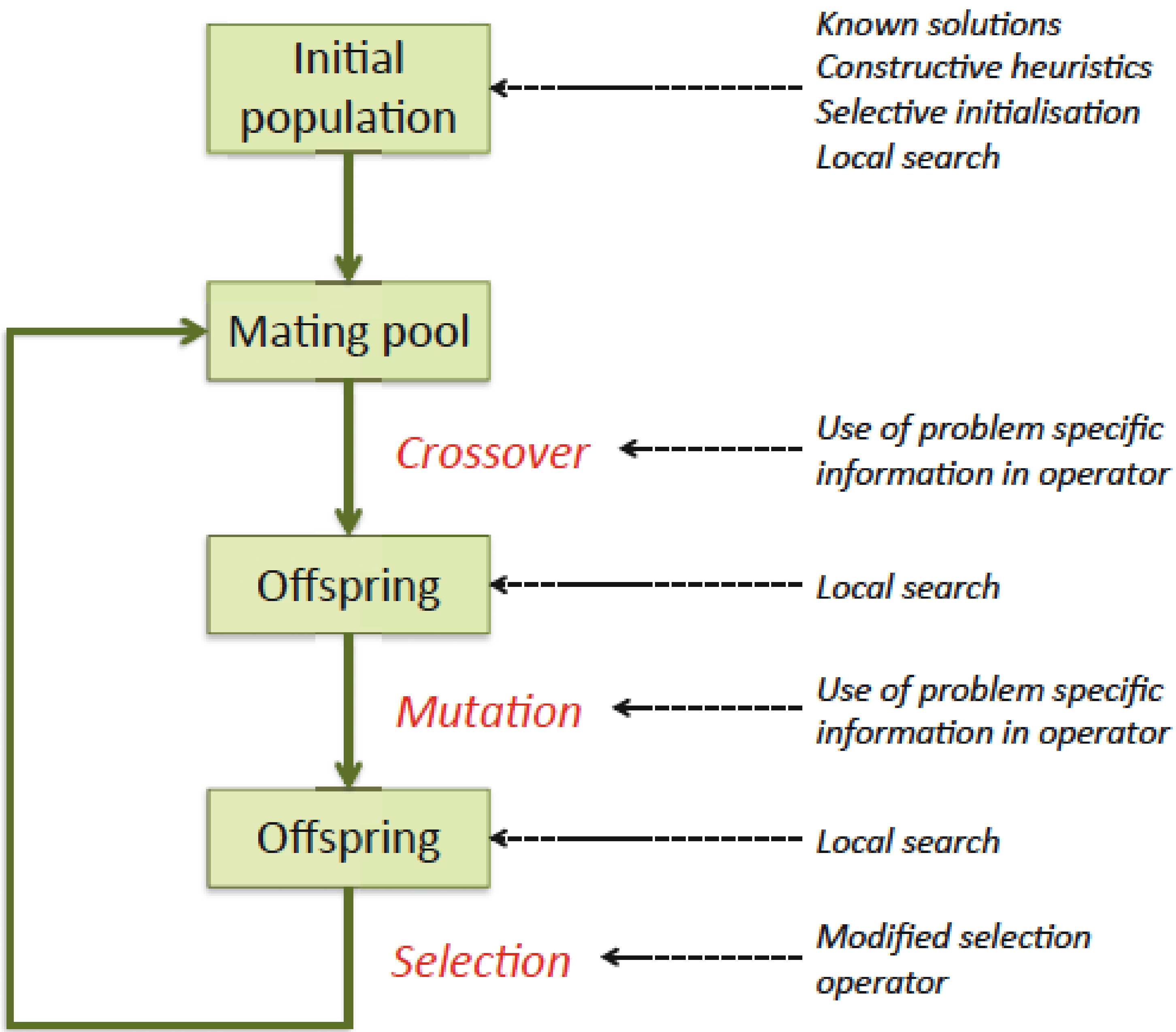


(Brain from Wikimedia Commons)

4. Local Search and Evolution

- In practice, most recent Memetic Algorithms use:
  - Pure Lamarckian evolution, or
  - A stochastic mix of Lamarckian and Baldwinian evolution

5. Where to Hybridise:



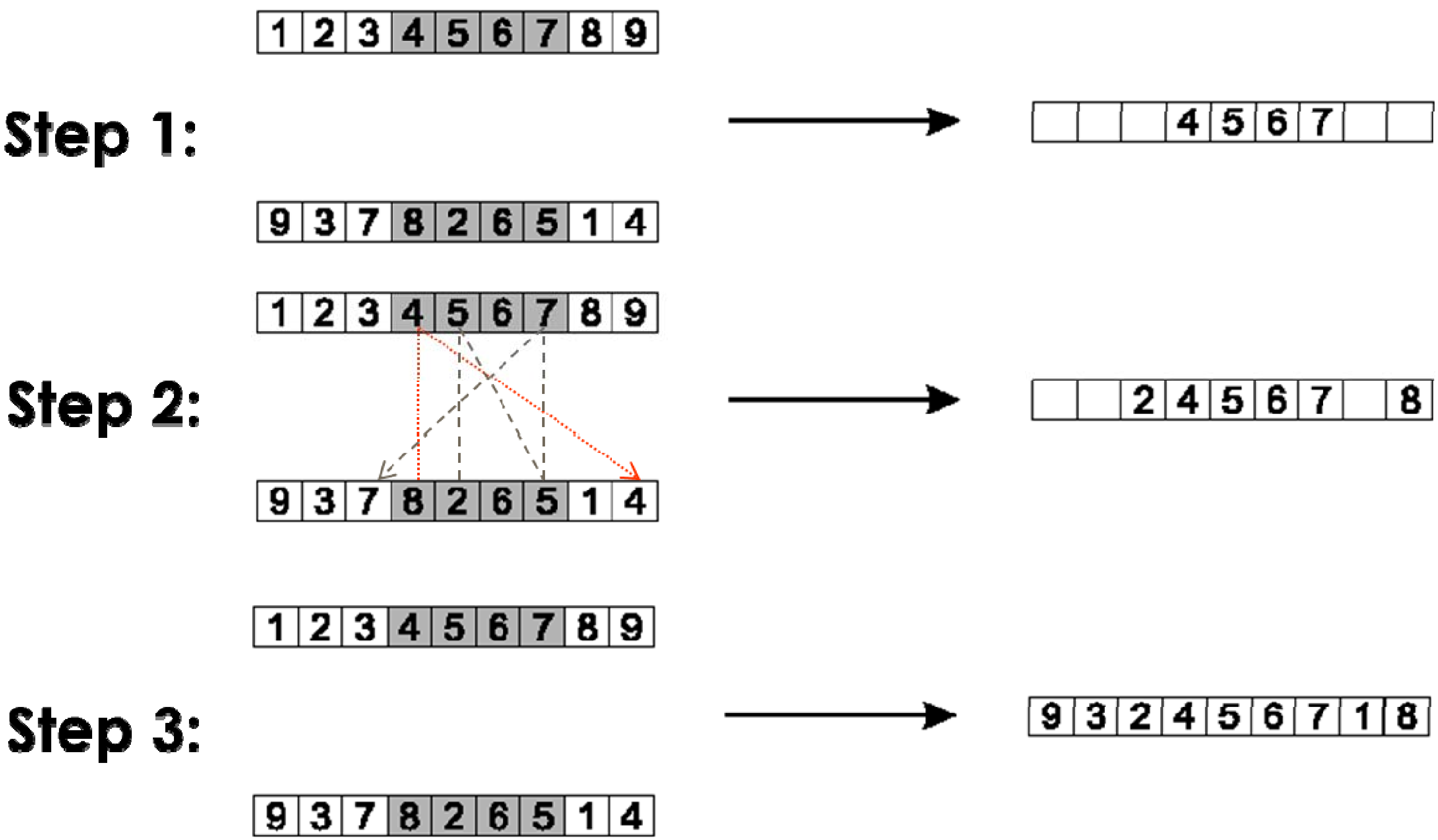
5. Where to Hybridise: In initialization

- Seeding
  - Known good solutions are added
- Selective initialization
  - Generate kN solutions, keep best N
- Refined start
  - Perform local search on initial population

5. Where to Hybridise: Intelligent mutation and crossover

- Mutation bias
  - Mutation operator has bias towards certain changes
- Crossover hill-climber
  - Test all 1-point crossover results, choose best
- “Repair” mutation
  - Use heuristic to make infeasible solution feasible

Note: We already saw examples of this. E.g. Partially mapped crossover



Hybrid Algorithms Summary

- It is **common** practice **to hybridise EA's** when using them in a real world context.
- This may involve the use of operators from other algorithms which have already been used on the problem, or the incorporation of domain-specific knowledge
- Memetic algorithms have been shown to be orders of magnitude faster and more accurate than EAs on some problems, and are the "state of the art" on many problems

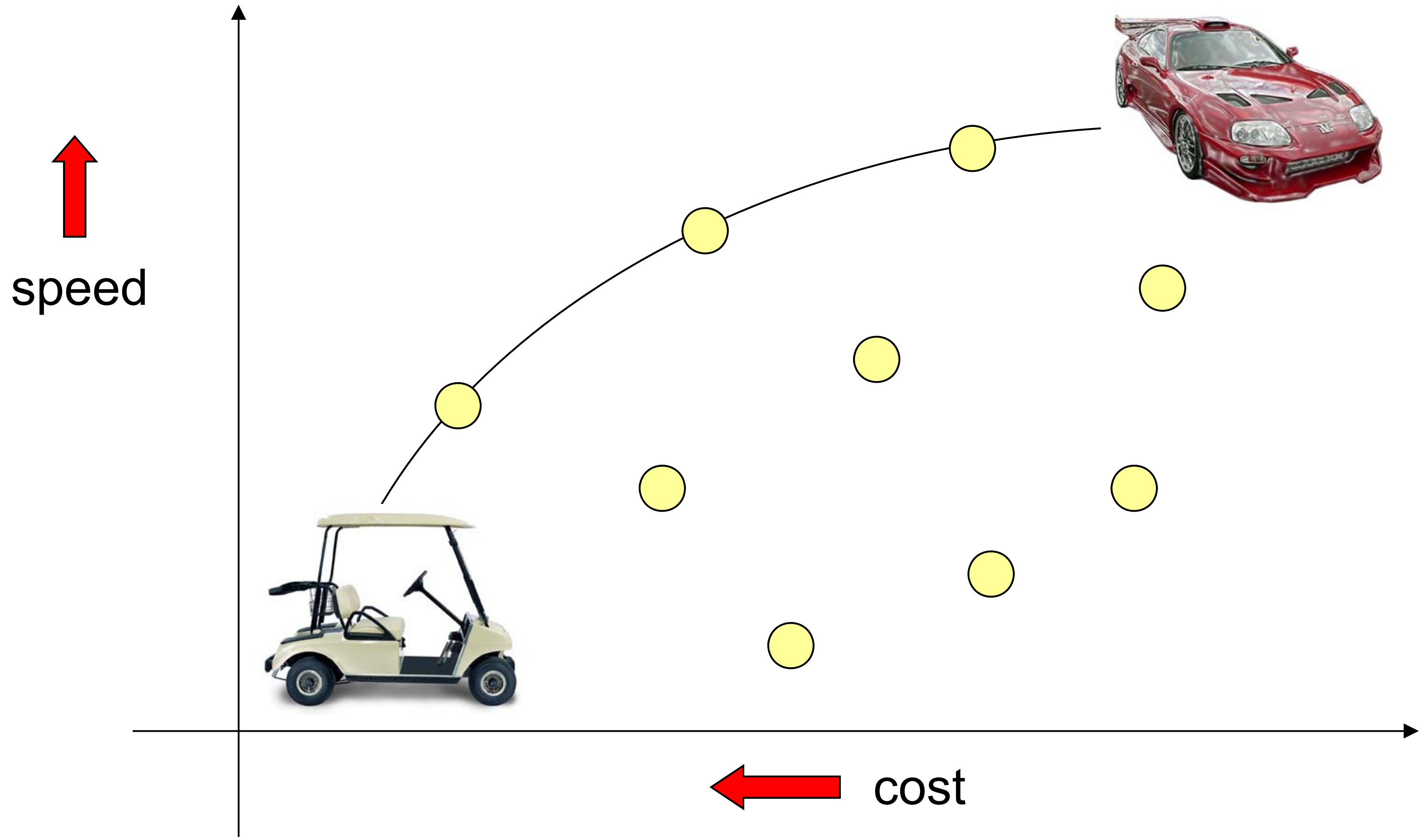
Chapter 12: Multiobjective Evolutionary Algorithms

- Multiobjective optimisation problems (MOP)
  - Pareto optimality
- EC approaches
  - Selection operators
  - Preserving diversity

Multi-Objective Problems (MOPs)

- Wide range of problems can be categorised by the presence of a number of ***n* possibly conflicting objectives**:
  - buying a car: speed vs. price vs. reliability
  - engineering design: lightness vs. strength
- Two problems:
  - finding set of good solutions
  - choice of best for the particular application

An example: Buying a car



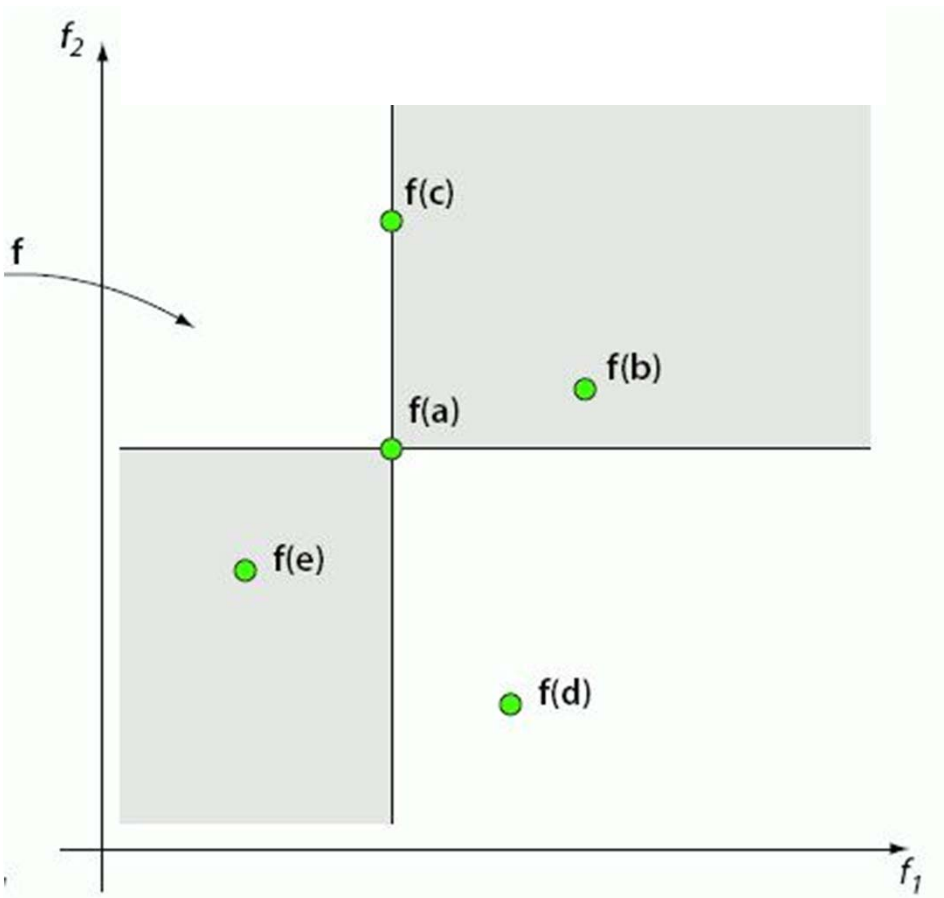
Two approaches to multiobjective optimisation

- **Weighted sum (scalarisation):**
  - transform into **a single objective** optimisation method
  - compute a weighted sum of the different objectives

- **A set of multi-objective solutions (Pareto front):**
  - The **population-based** nature of EAs used to ***simultaneously*** search for a set of points approximating Pareto front

Comparing solutions

Objective space

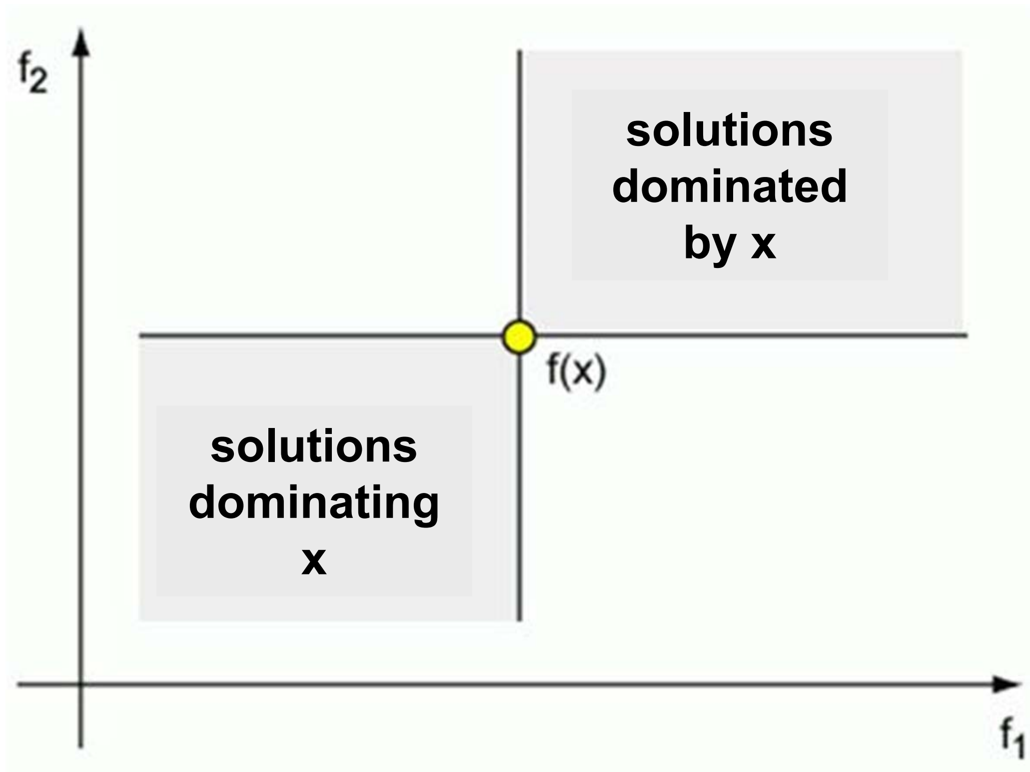


- Optimisation task:
  Minimize both f_1 and f_2

- Then:
  a is better than b
  a is better than c
  a is worse than e
  a and d are incomparable

Dominance relation

- Solution x dominates solution y, ($x \preceq y$), if:
  - x is better than y in at least one objective,
  - x is not worse than y in all other objectives



Pareto optimality

- Solution x is **non-dominated** among a set of solutions Q if no solution from Q dominates x
- A set of non-dominated solutions from the entire feasible solution space is the **Pareto set**, or **Pareto front**, its members Pareto-optimal solutions

Illustration of the concepts

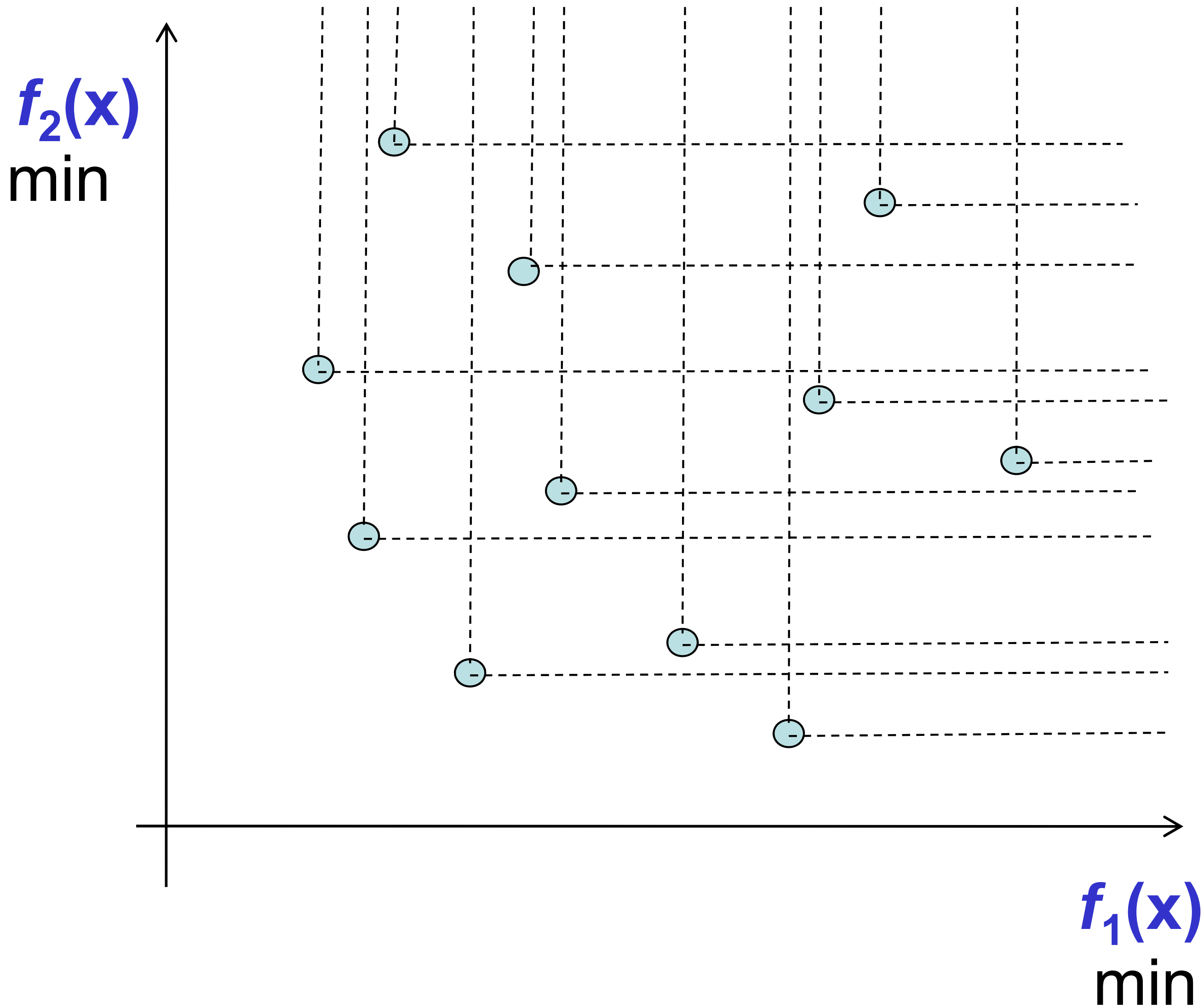
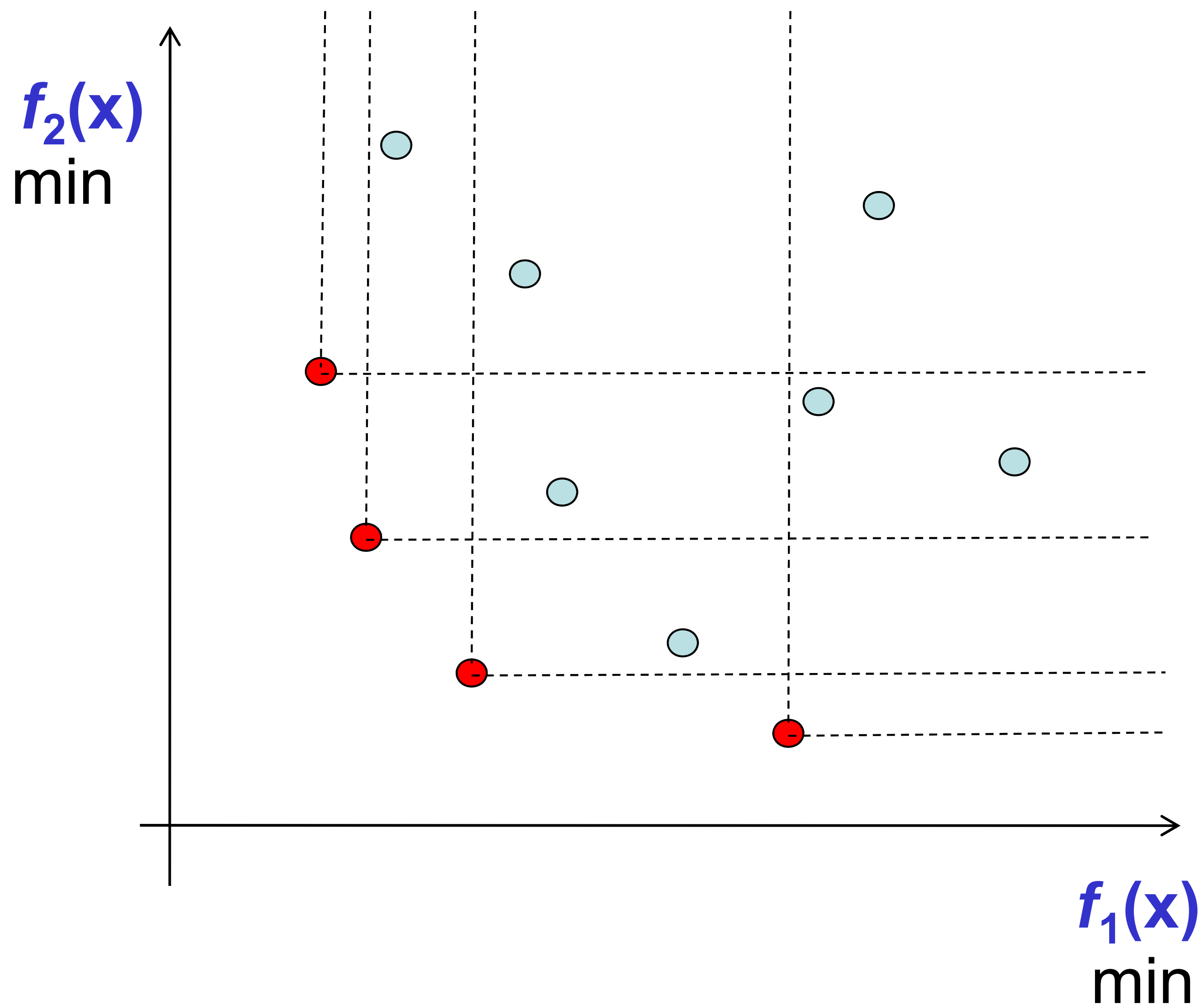
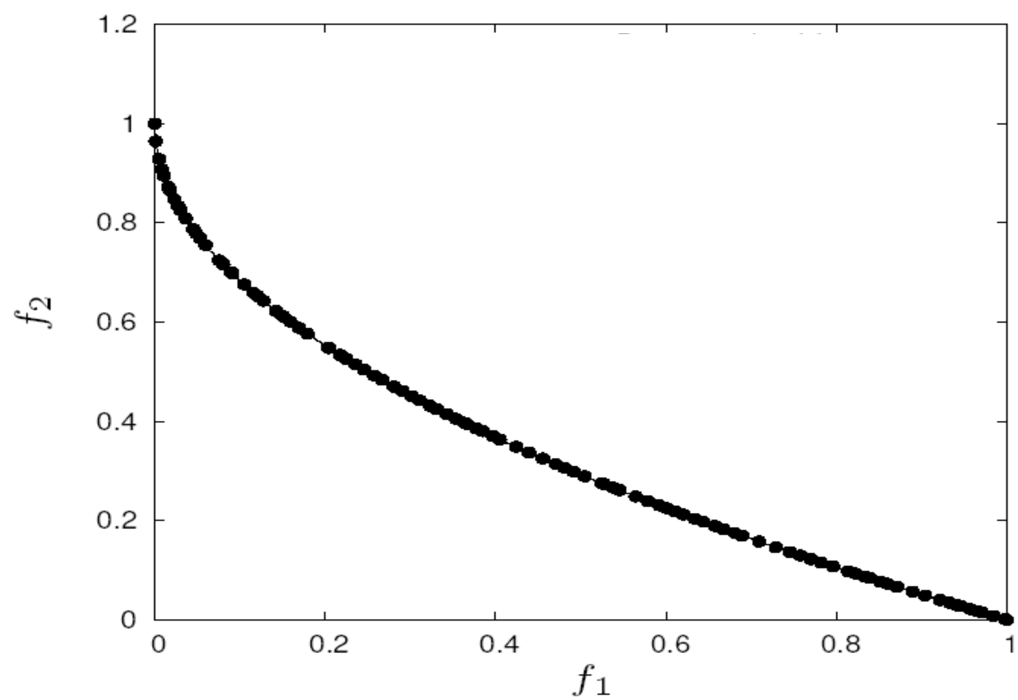


Illustration of the concepts



Goal of multiobjective optimisers

- Find a set of non-dominated solutions (**approximation set**) following the criteria of:
  - **convergence** (as close as possible to the Pareto-optimal front),
  - **diversity** (spread, distribution)



EC approach: Requirements

1. Way of assigning fitness and **selecting individuals**,
   – usually based on dominance

2. Preservation of a **diverse set of points**
   – similarities to multi-modal problems

3. Remembering all the **non-dominated points** you have seen
   – usually using elitism or an archive

EC approach:
1. Selection

- Could use aggregating approach and change weights during evolution
  - no guarantees
- Different parts of population use different criteria
  - no guarantee of diversity
- Dominance (made a breakthrough for MOEA)
  - ranking or depth based
  - fitness related to whole population

Example: Dominance Ranking in NSGA-II

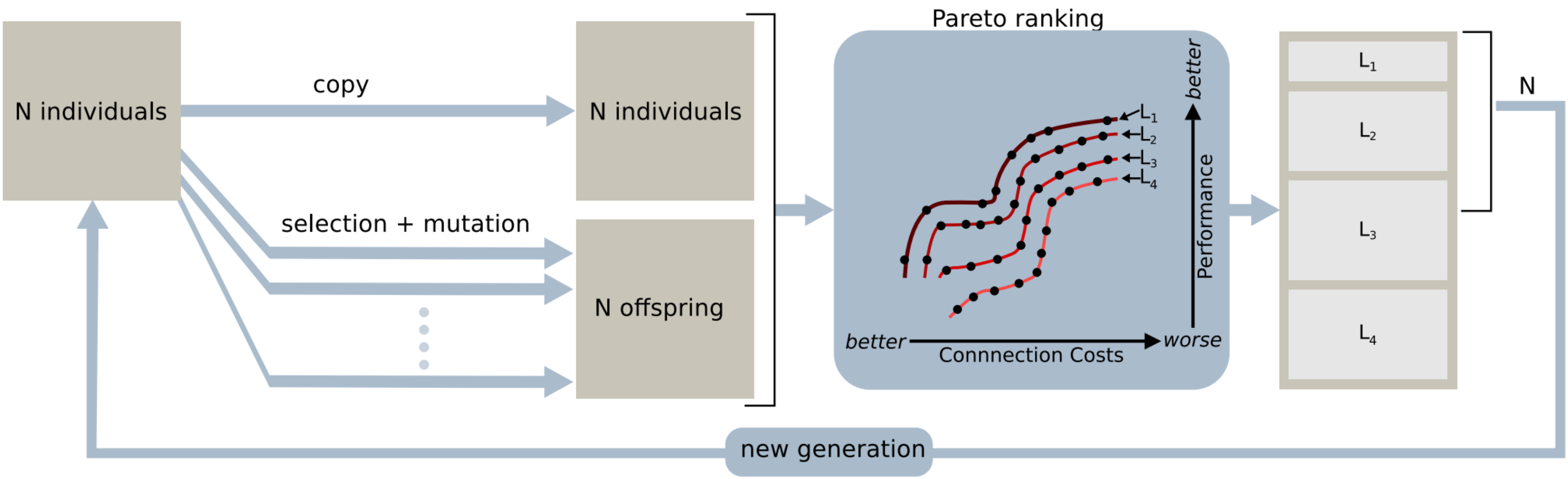


Figure from Clune, Mouret & Lipson (2013): "The evolutionary origins of modularity"

EC approach:
2. Diversity maintenance

- Aim: Evenly distributed population along the Pareto front
- Usually done by niching techniques such as:
  - fitness sharing
  - adding amount to fitness based on inverse distance to nearest neighbour
- All rely on some distance metric in genotype / phenotype / objective space

EC approach:
3. Remembering Good Points

- Could just use elitist algorithm, e.g. ($\mu + \lambda$) replacement

- Common to maintain an archive of non-dominated points
  – some algorithms use this as a second population that can be in recombination etc.
  – others divide archive into regions too

Multi objective problems - Summary

- MO problems occur very frequently
- EAs are very good in solving MO problems
- MOEAs are one of the most successful EC subareas