Biologically Inspired Computation

Dr Marta Vallejo
m.vallejo@hw.ac.uk

Lecture 8

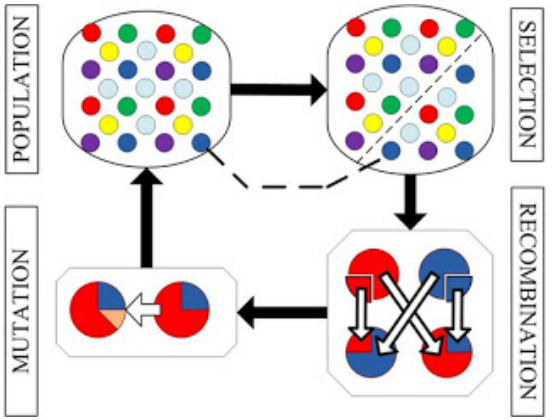
Introduction to Evolutionary Algorithms

- What are Evolutionary Algorithms?
- Classification of EA methods and related heuristics
- History of Evolutionary Algorithms
- Domains of Application
- Natural Evolution as a Problem Solving Method
- A Generic Evolutionary Algorithm
- The Fitness Function
- Landscapes
- Pros & Cons
- Hill Climbing
- Local Search
- Population-based Algorithms
- Examples and Bibliography

What are Evolutionary Algorithms?

Evolutionary Algorithm: metaheuristic that operates on a population of potential solutions (population-based algorithm) applying the principle of **evolutionary biology**:

- Inheritance
- Mutation
- Selection
- Crossover/recombination



A type of **Guided** Random Search Used for optimisation problems. Niche: poorly-understood problems (non-linear, discontinuous, multiple optima,…) where no other method works well

Classification of EA Methods

Genetic Algorithm (GA)

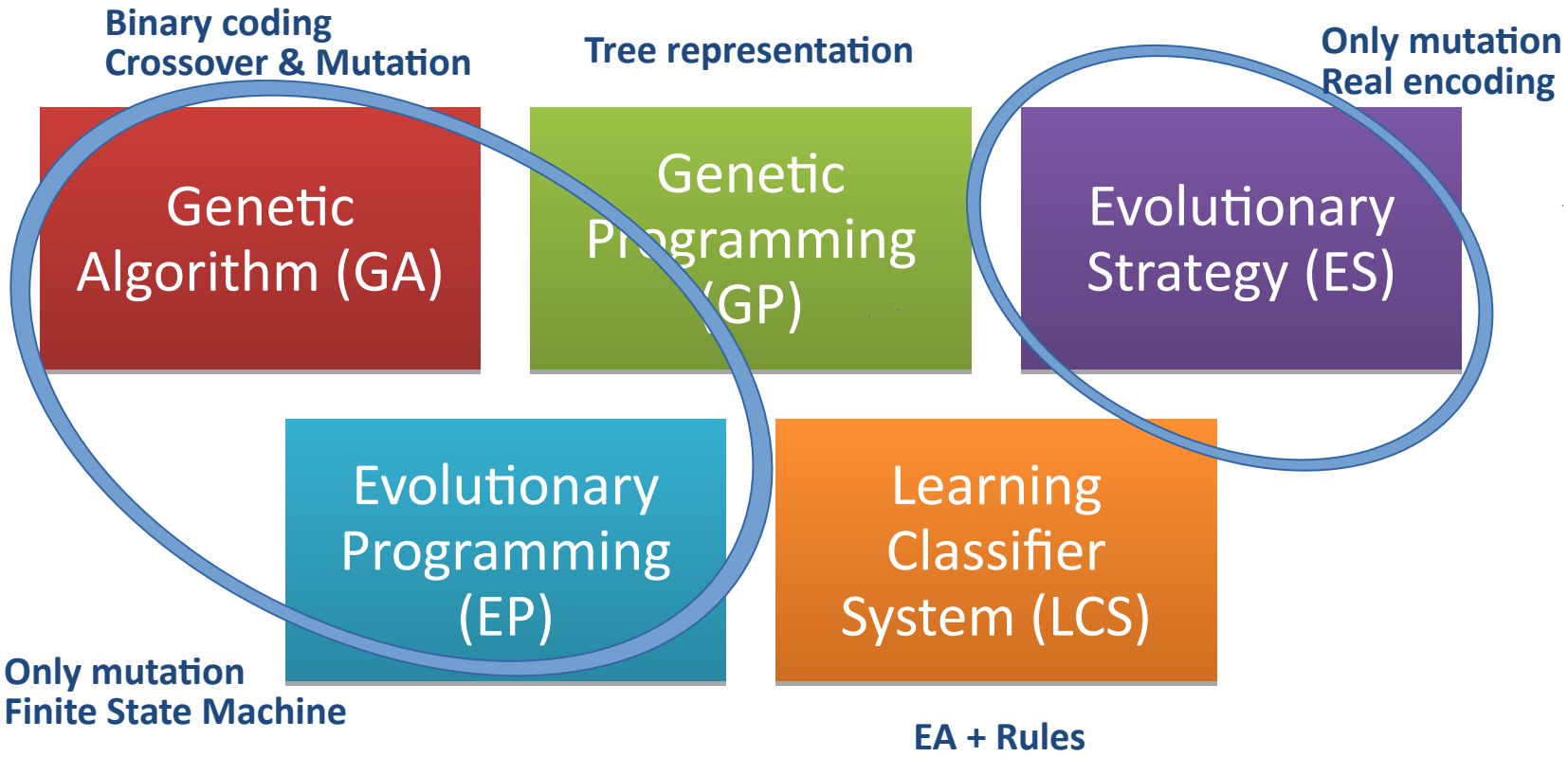
Genetic Programming (GP)

Evolutionary Strategy (ES)

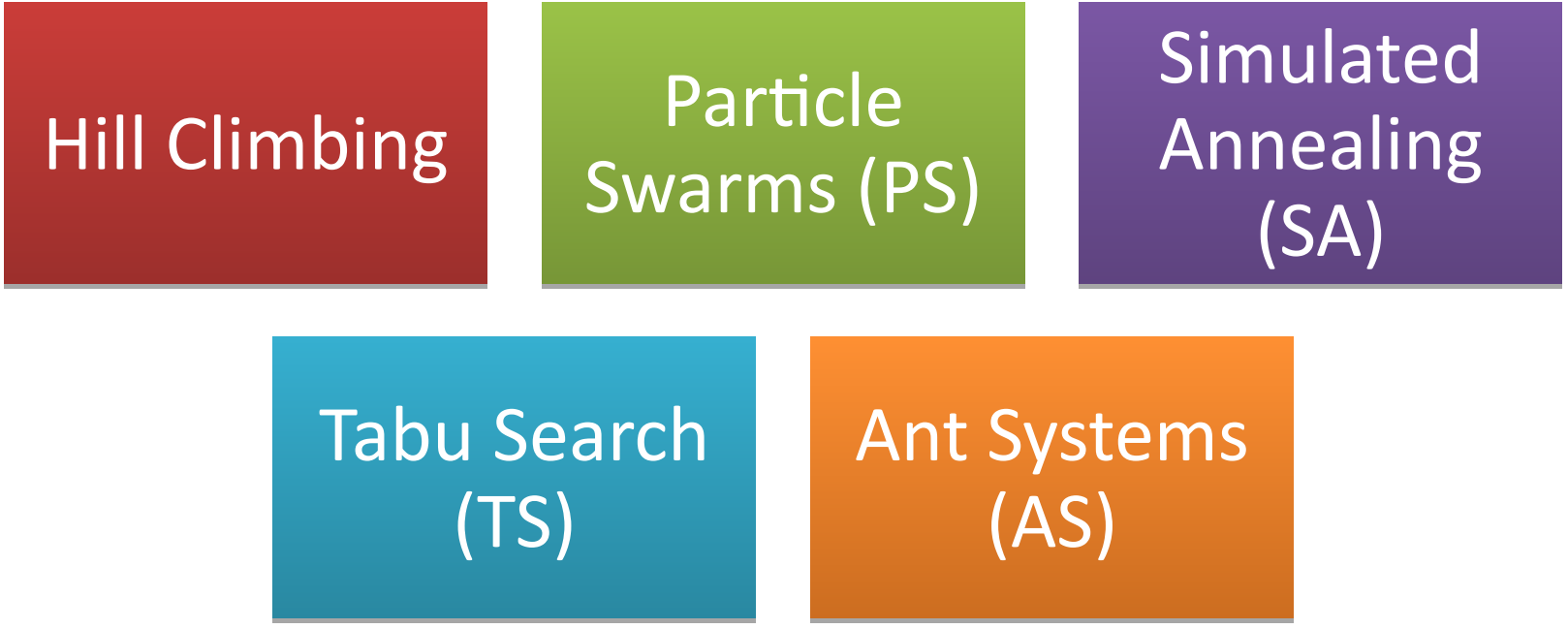
Evolutionary Programming (EP)

Learning Classifier System (LCS)

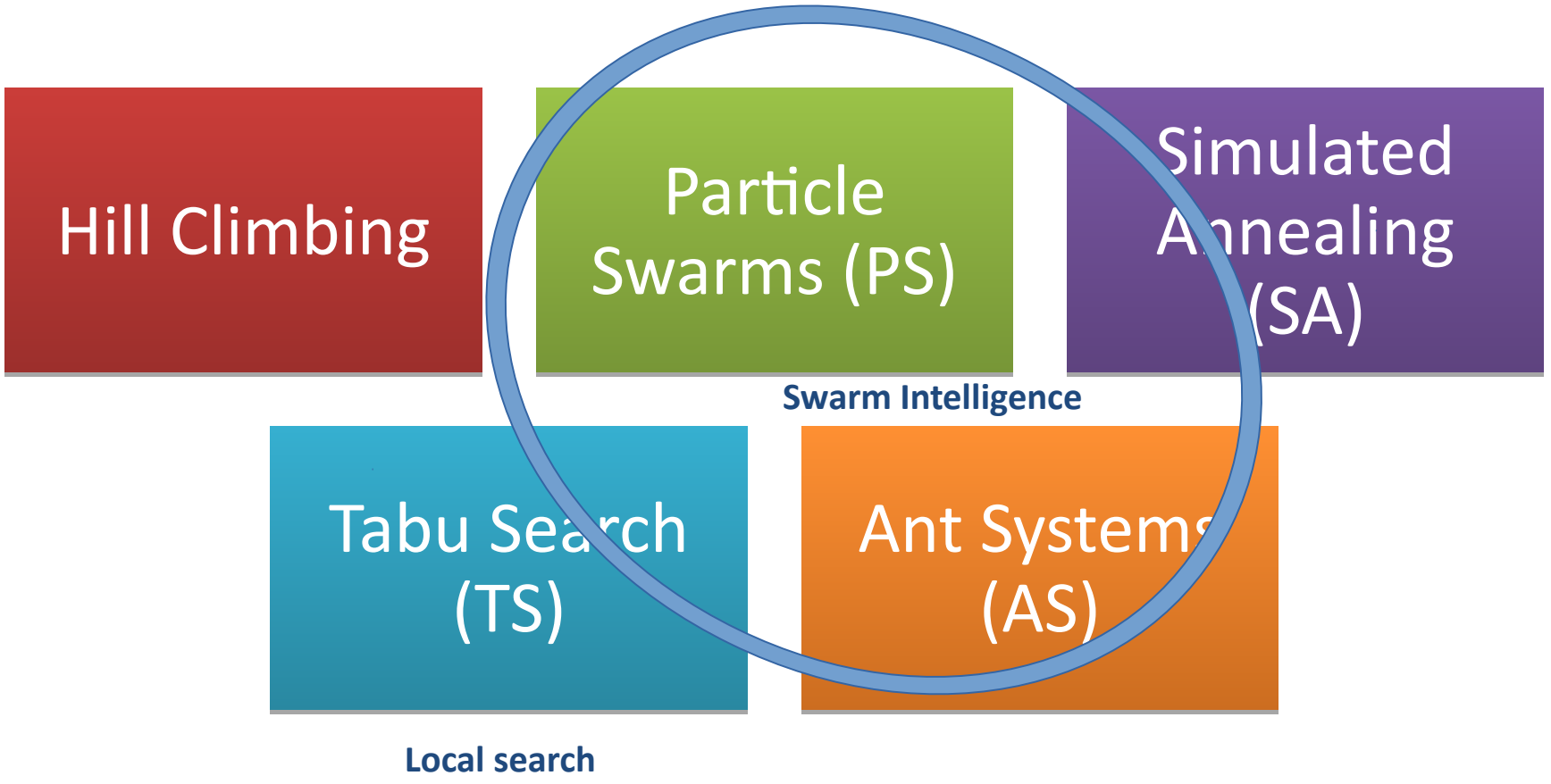
Classification of EA Methods



Related Search Heuristics



Related Search Heuristics



History of Evolutionary Algorithms

- L. Fogel 1962 (San Diego, CA): Evolutionary Programming
- J. Holland 1962 (Ann Arbor, MI): Genetic Algorithms

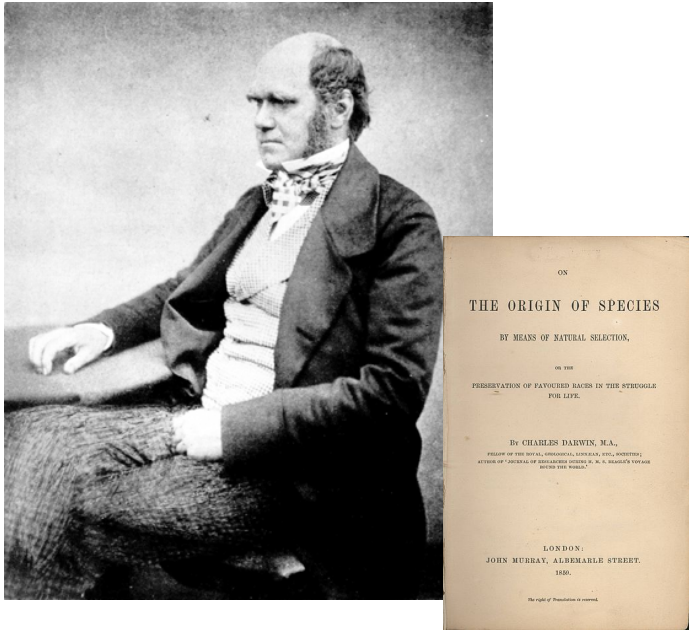
  He wrote the first book on Genetic Algorithms 'Adaptation in Natural and Artificial Systems' in 1975.
- I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany): Evolution Strategies
- J. Koza 1989 (Palo Alto, CA): Genetic Programming. Variation of genetic algorithms to evolve programs that perform certain tasks.

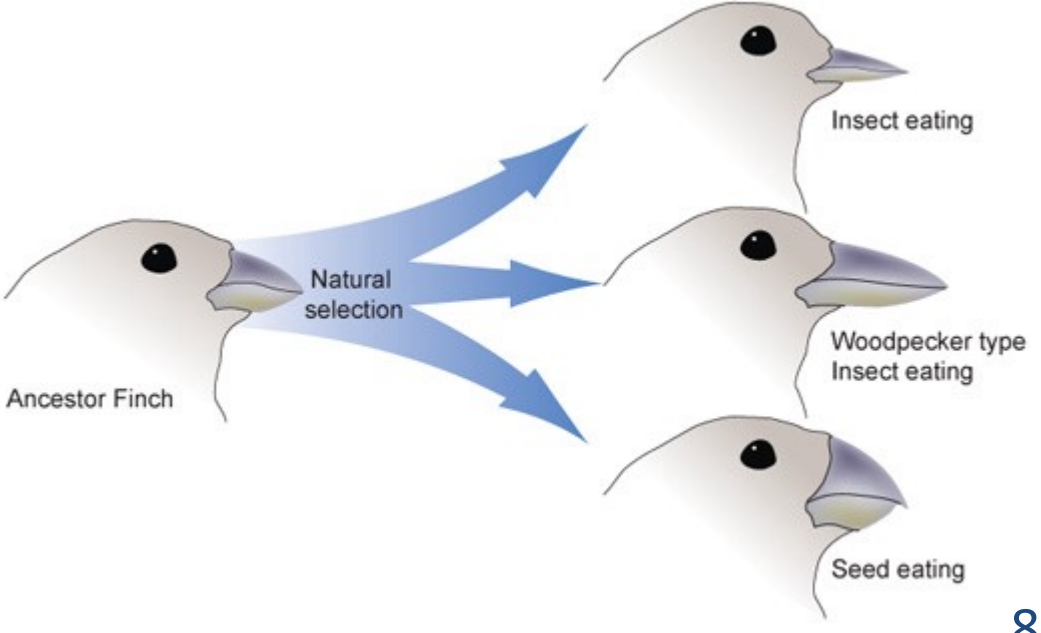
Domains of Application

- Numerical, Combinatorial Optimisation
- Automatic Programming
- System Modelling and Identification
- Planning and Control
- Engineering Design
- Data Mining
- Machine Learning
- Artificial Life

Evolution/Survival of the Fittest

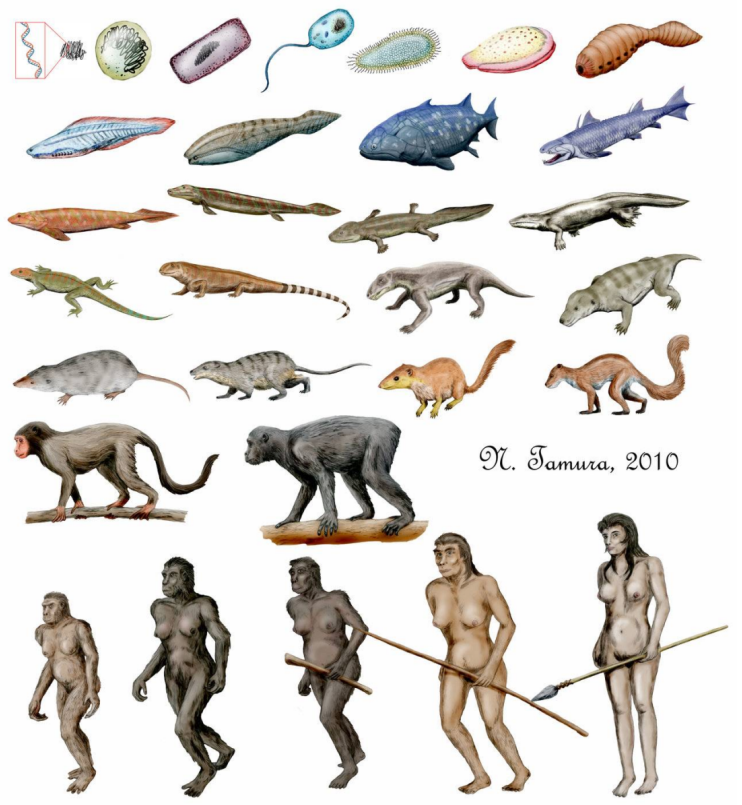
The **theory of evolution** states that all species on Earth have arisen by evolution from one or more very simple self-reproducing molecules in the primeval soup. We have evolved by the accumulation of countless advantageous mutations over countless generations, and species have diversified to occupy environmental niches, as a result of different environments favouring different mutations.



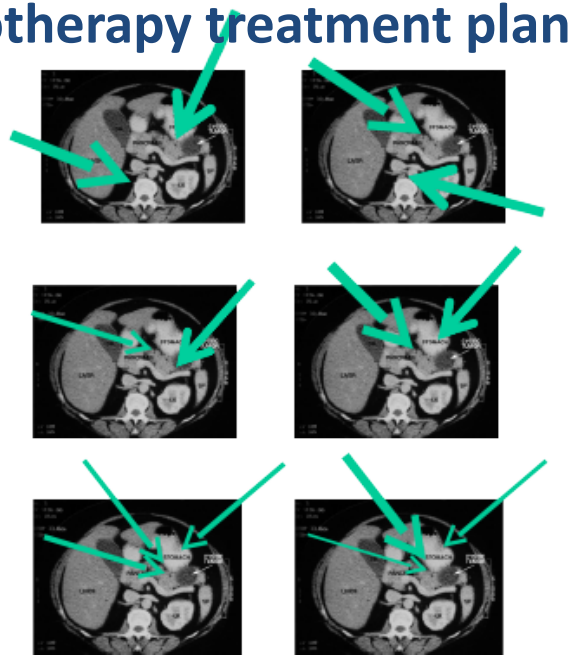


Using the Evolution

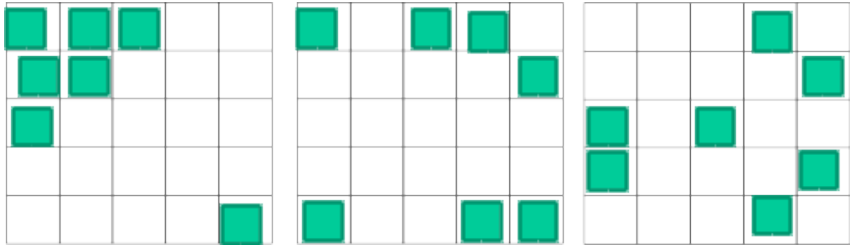
Human Evolution



Radiotherapy treatment plans



Lecture timetables



Natural Evolution as a Problem Solving Method

Given:

- a population of organisms that **can reproduce** (generate new organisms) in a challenging/changing environment
  (… their chances of reproduction depends on how well they cope with their environment)

- a way of continually **generating diversity** in new `child' organisms (so – new organisms are not simply copies of old ones)

A **survival of the fittest** principle will naturally emerge: future generations will have mixes of characteristics that tend to go along with being good at surviving in this environment.

Evolution as an Optimisation Method

Can view evolution as a way of solving the problem
How can I best survive in this environment?

The basic evolution method to do that is trial and error - something like this:

1. *Come up with a new solution by randomly changing an old one. Does it work better than previous solutions? If yes, keep it and throw away one of the old ones. Otherwise, discard it.*
2. *Go to 1.*

But this appears to be a recipe for problem solving algorithms which take forever, with little or no eventual success!

Concepts taken from Nature

There are certain things we learn from natural evolution, which lead to a generally superb problem solving methods called Evolutionary Algorithms:

Lesson 1 Natural evolution is driven by a complex environment – essentially this calculates an **organism's 'fitness'** over its lifetime. We can replace that with a much faster calculation!

Lesson 2 Keep a population/collection of different things (solutions) on the go.

Lesson 3 Select `parents' with a relatively weak bias towards the fittest. It's not really plain survival of the fittest, what works is the fitter you are, the more chance you have to reproduce, and it works best if even the least fit still have some chance.

Lesson 4 Use randomised Mutation and/or Recombination (aka crossover) to generate new candidate solutions from the selected 'parents'

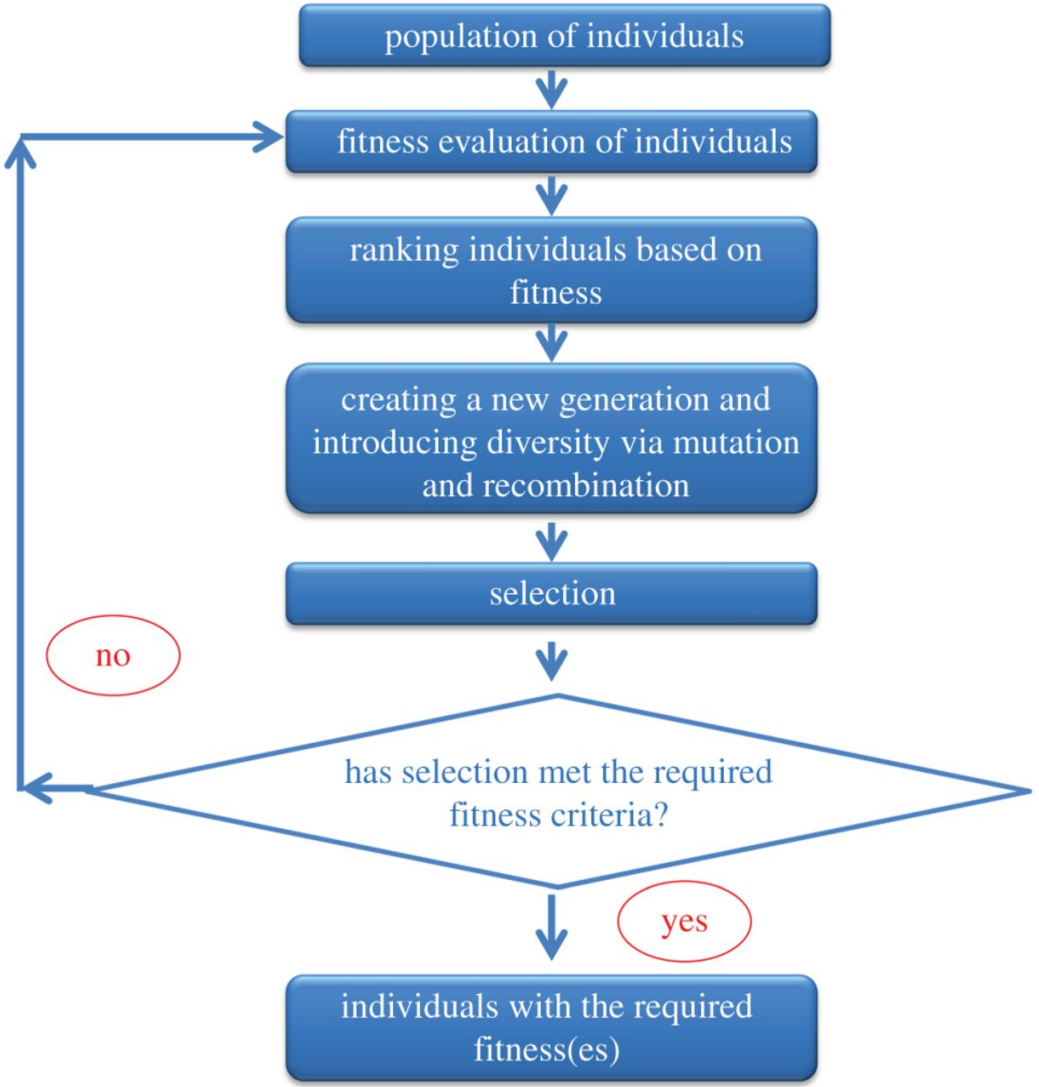
A Generic Evolutionary Algorithm

Suppose you have to find a solution to a given problem and suppose, given any candidate solution *s* you have a **fitness function** *f(s)* which measures how good *s* is as a solution to the problem.

Generate an initial population P of randomly generated solutions (this is typically 100 or 500 or so). Use *f(s)* to evaluate the fitness of each. Then:

Repeat until a termination condition is reached:

- **Selection**: Choose some of P to be parents
- **Variation**: Apply 'genetic operators' to the parents to produce some children, and then evaluate the fitness of the children.
- **Population update**: Update the population P by retaining some of the children and removing some of the incumbents.


population of individuals
fitness evaluation of individuals
ranking individuals based on fitness
creating a new generation and introducing diversity via mutation and recombination
selection
has selection met the required fitness criteria?
no
yes
individuals with the required fitness(es)

Key elements to take into account

How to select? some of P to be parents
Always select the best? Bad results, quickly
Select almost randomly? Great results, too slowly

How to encode?
The 'Encoding' or 'Representation', is the approach used to specify a solution as a data structure. This is intricately tied up with:

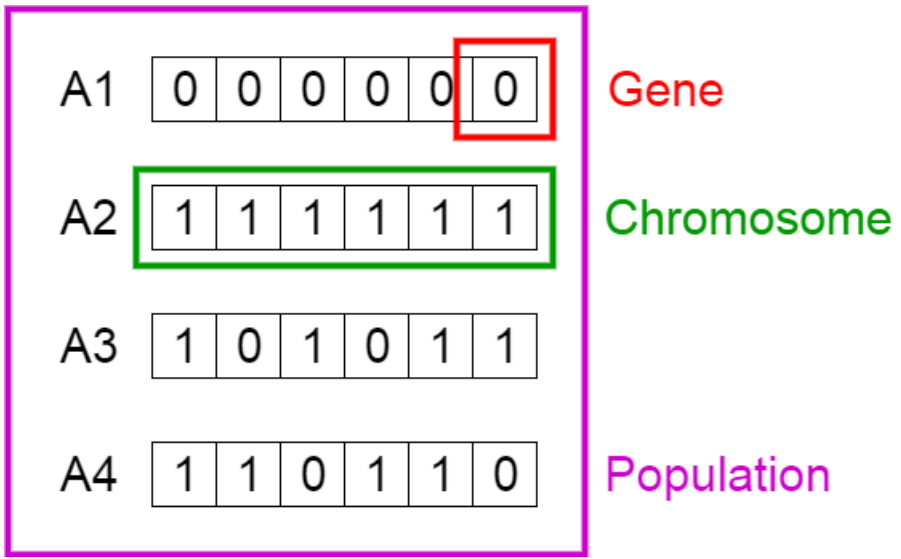
How to vary?
Apply genetic operators: mutation, recombination, etc…
small-step mutation preferred, recombination seems to be a principled way to do large steps, but large steps are usually abysmal.

How to update the population?
replace entire population with the new children?
choose best |P| from P and the new ones?

What parameters? How to adapt with time?

Terminology for Evolutionary Algorithms

- **Individual:** Any possible solution
- **Population:** Group of all individuals
- **Search Space:** All possible solutions to the problem
- **Chromosome:** Blueprint for an individual
- **Gene/Trait:** Possible aspect (features) of an individual
- **Allele:** Possible settings of a trait (black, blond, etc.)
- **Locus:** The position of a gene on the chromosome
- **Genome:** Collection of all chromosomes for an individual EA



http://geneticalgorithms.ai-depot.com/Tutorial/Overview.html

The Fitness Function

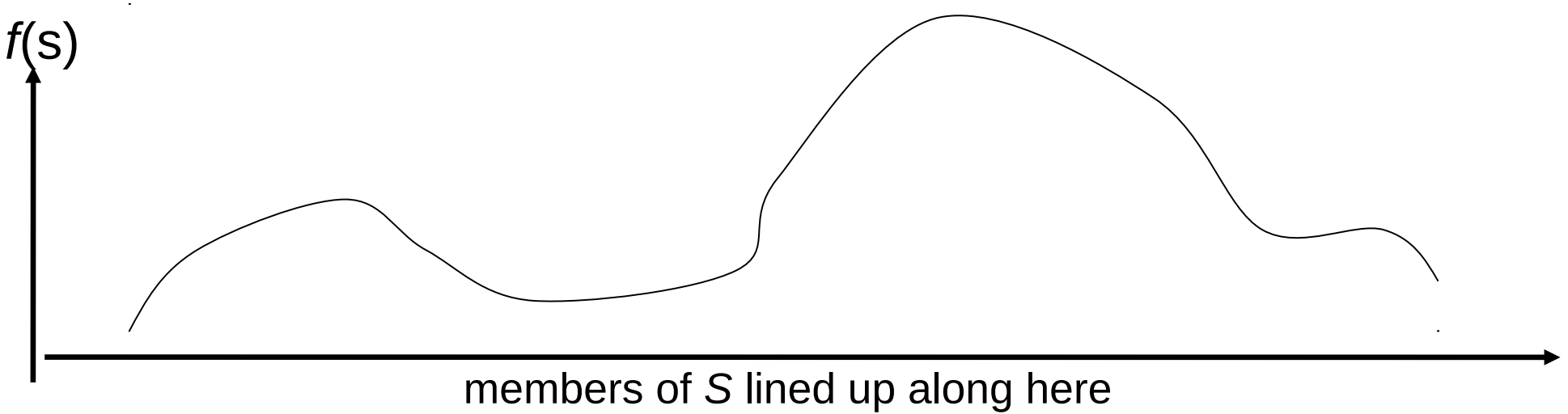
In EA often we do not understand totally the nature of the problem, but we have an idea about **how to measure good solutions**.

Every candidate solution *s* in the set of all solutions *S* can be given a score, or a "fitness", by a fitness function. We usually write *f(s)* to indicate the fitness of solution *s*.

GOAL: we want to find the *s* in *S* with the best score → optimise its value.

Landscapes

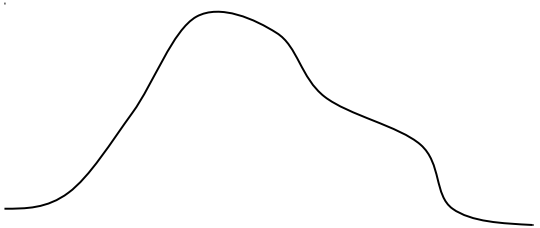
Recall *S* is the search space, and *f(s)* is the fitness of a candidate solution in *S*



The structure we get by imposing f(s) on S is called a **landscape**. In practice, a solution will have multiple variables, and a corresponding axis for each variable. Then it is challenging to visualise any fitness landscape with more than two variables

Landscapes

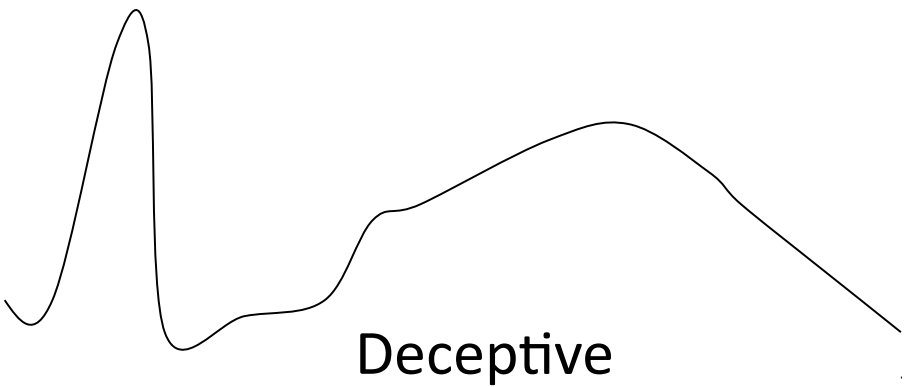
There are various classes of fitness landscape. Here are some common examples:



Unimodal

Plateau

Multimodal



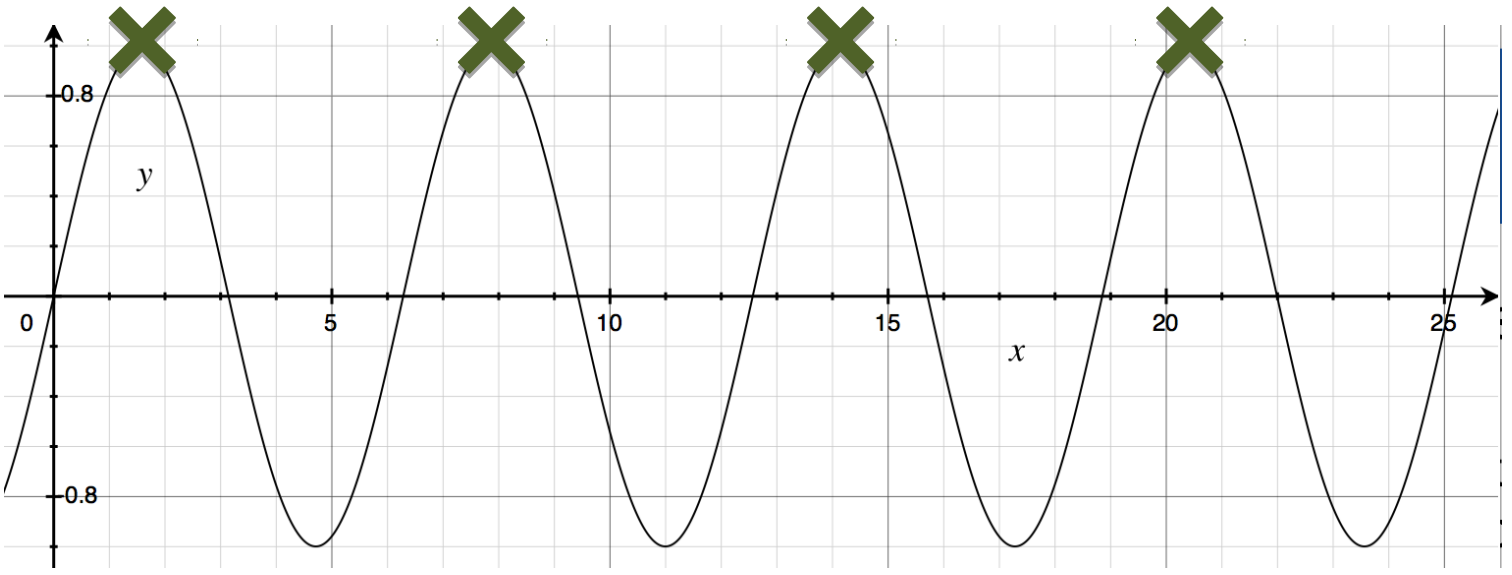
Deceptive

Landscapes

Fitness landscapes are very diverse, but most of them contain lots of local optima. Some can even contain **multiple global optima**, how?

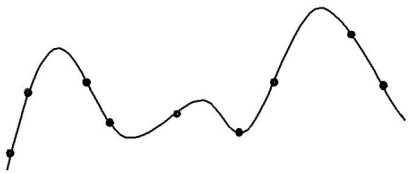
Landscapes

Fitness landscapes are very diverse, but most of them contain lots of local optima. Some can even contain **multiple global optima**, if each optima is equal to the rest

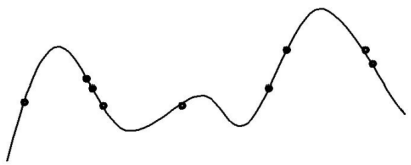


Typical behaviour of an EA

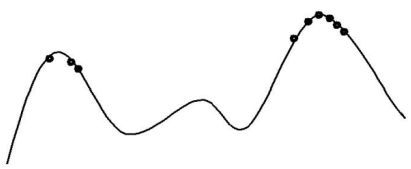
Phases in optimising on a 1-dimensional fitness landscape:



Early phase:
quasi-random population distribution



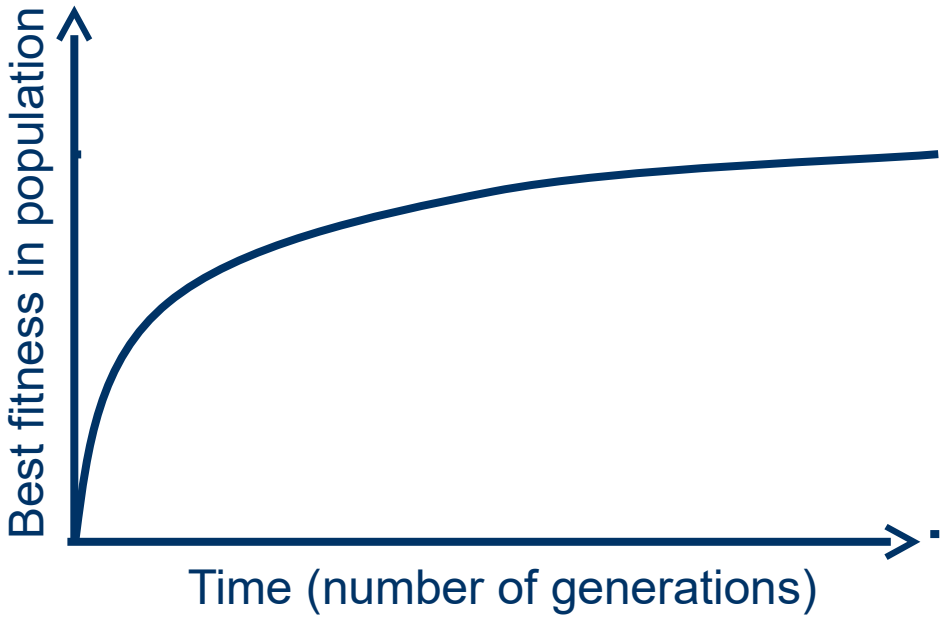
Mid-phase:
population arranged around/on hills



Late phase:
population concentrated on high hills

Typical run: progression of fitness

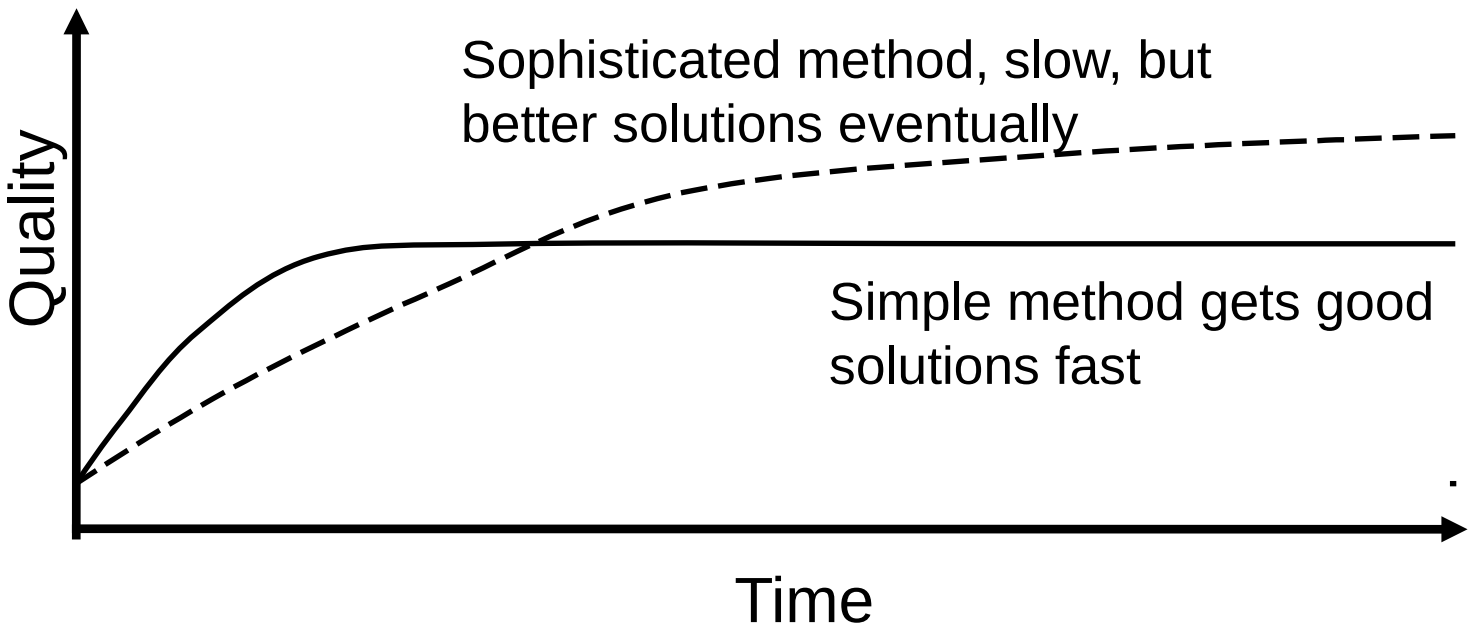
EA is a metaheuristic specially good for large and complex **search spaces**



A typical run of an EA shows "**anytime behaviour**": An algorithm that returns as low-cost solutions as possible at any moment of its execution

Typical Performance of Approximate Methods

Evolutionary Algorithms turn out to be one of the most successful traditional Machine Learning algorithm. They often take **a long time** though – it's worth getting used to the following curve which tends to apply across the board.



Pros & Cons

Advantages:

- No presumptions about the problem space
- Easily applicable
- Low development & application costs
- Easy to incorporate other methods
- Solutions are interpretable (unlike ANN)
- Can be run interactively, accommodate user proposed solutions
- Provide many alternative solutions to a given problem. The final choice is lef to the user

Disadvantages:

- No guarantee for optimal solution within finite time
- Weak theoretical basis
- May need parameter tuning
- Often computationally expensive, i.e. slow

Hill Climbing

Almost all metaheuristics build upon a basic algorithmic known as **hill climbing**:

- Generate a solution, usually randomly
- Evaluate the solution using an objective function
- Make one or more random changes to the solution
- Evaluate the modified solution
- If it's better than the original solution, replace it

Algorithm 4 *Hill-Climbing*

1: $S \leftarrow$ some initial candidate solution
2: **repeat**
3: $R \leftarrow \text{Tweak}(\text{Copy}(S))$
4: **if** $\text{Quality}(R) > \text{Quality}(S)$ **then** ▷
5: $S \leftarrow R$
6: **until** S is the ideal solution or we have run out of time
7: **return** S

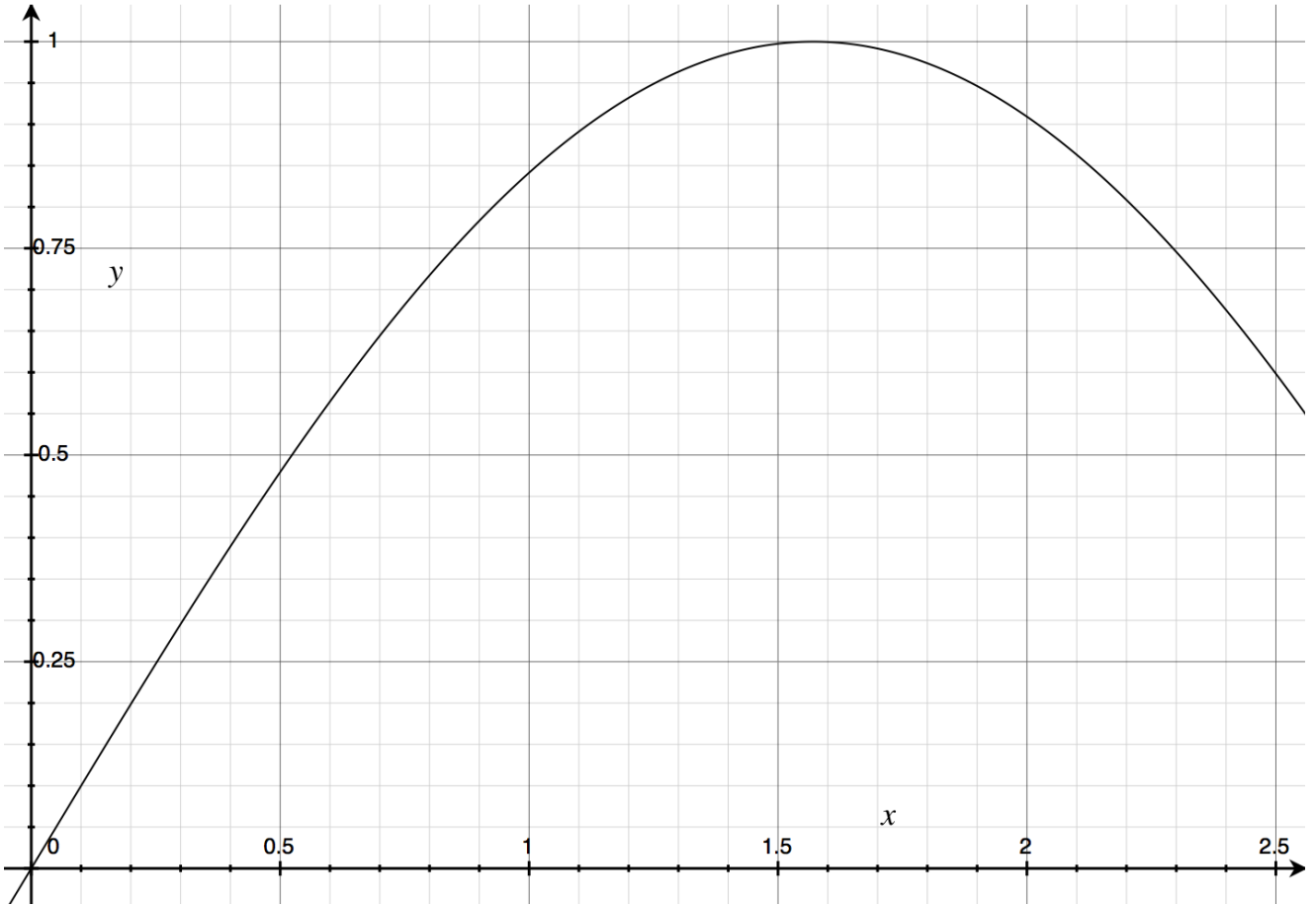
In the bio-inspired community, a "tweak" is usually called a "mutation"

"Quality" (how good a solution i) is usually referred to as "fitness" or "objective value"

Essentials of Metaheuristics, p17

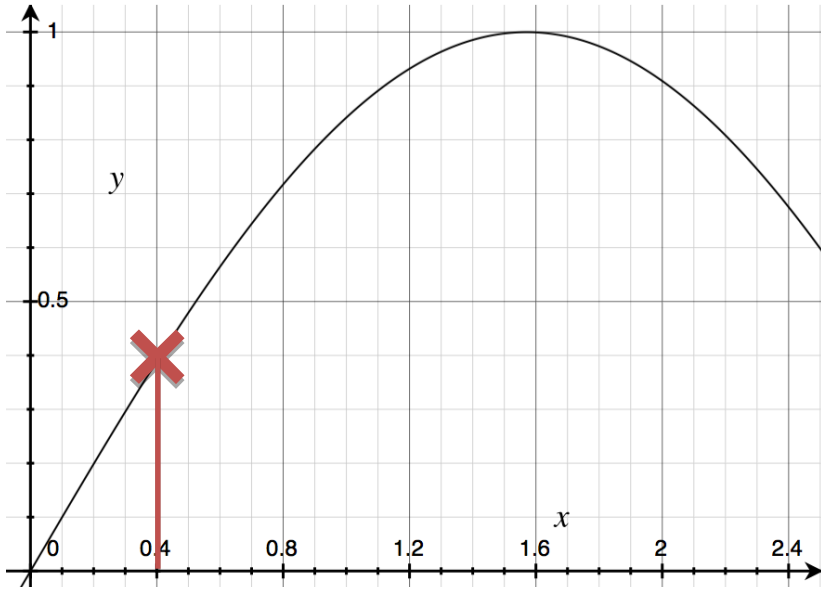
Hill Climbing

Consider optimising (find the maximum) the function **sin(x)**. Its **search space** is:

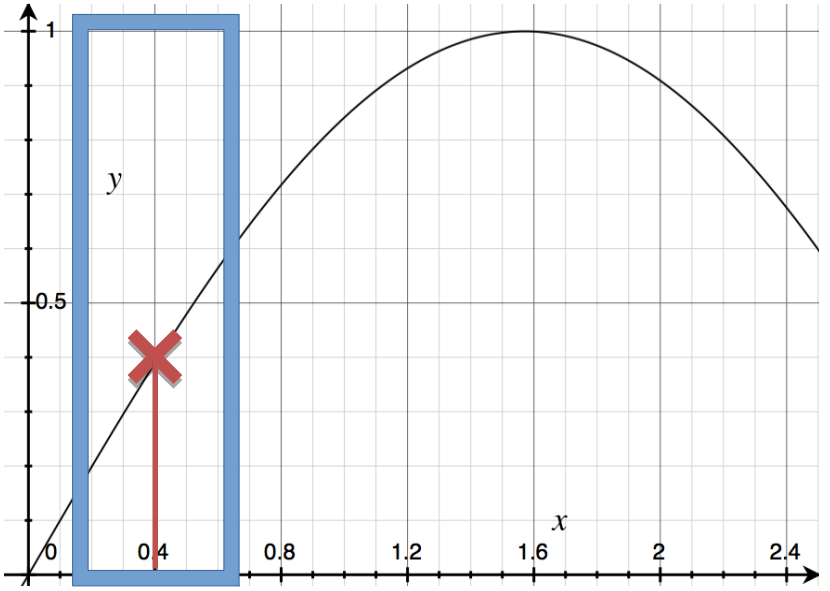


Hill Climbing

We could start off with a random solution, x=0.4

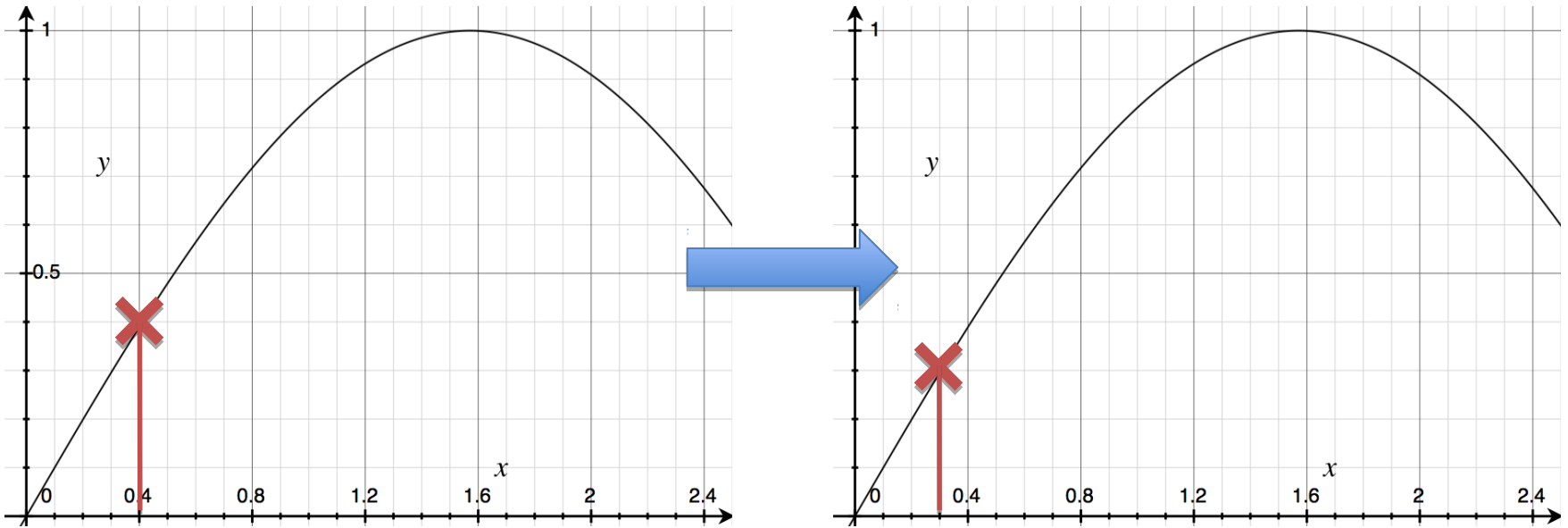


We then need to choose a **neighbourhood**
Let's say this is any x within ±0.3 of the current value



Hill Climbing

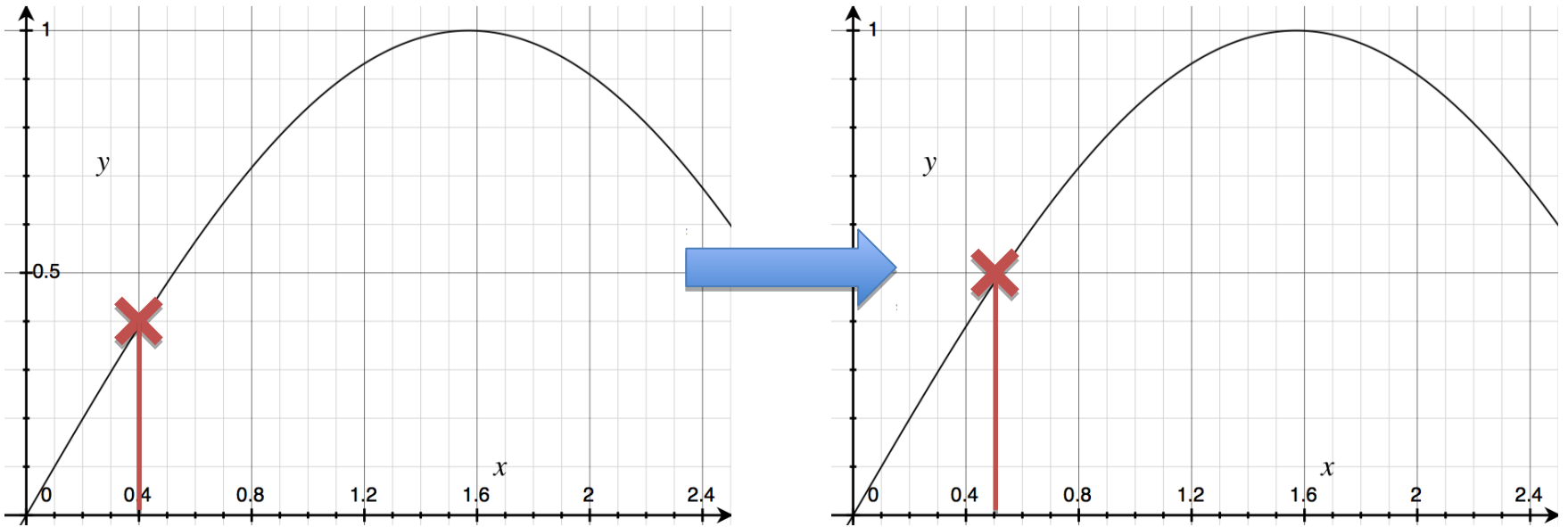
Randomly choose a neighbouring solution: x=0.3. Is it better?



This solution is worse, so it is rejected

Hill Climbing

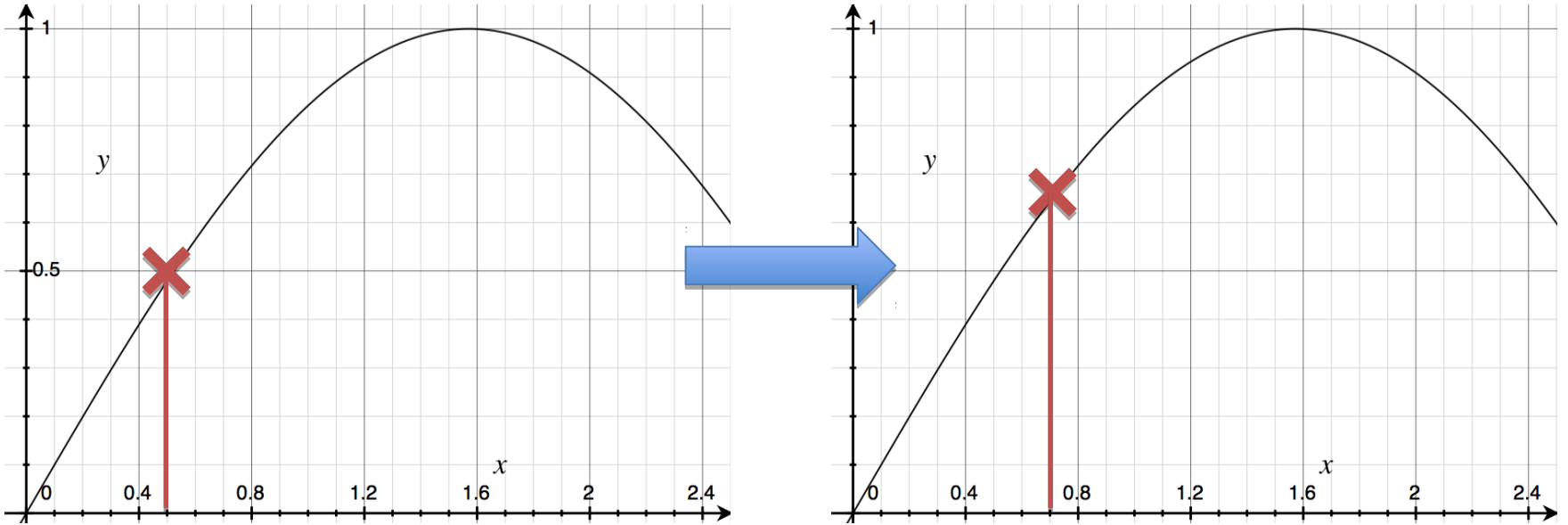
We try another neighbouring solution: x=0.5. Is it better?



This solution is better, so it replaces the original

Hill Climbing

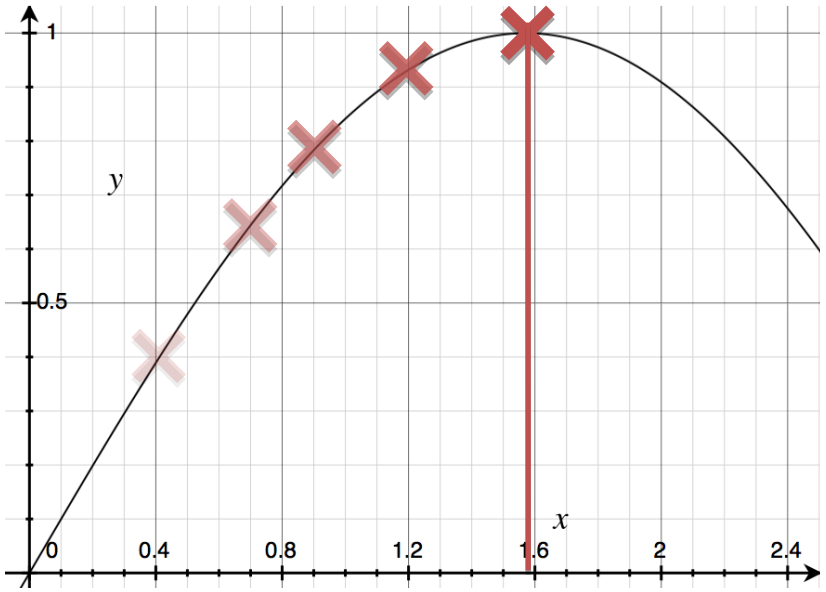
Again, we try another neighbouring solution (±0.3) now from x=0.5, like for example x=0.7 Is it better?



This solution is again better, so it replaces the original

Hill Climbing

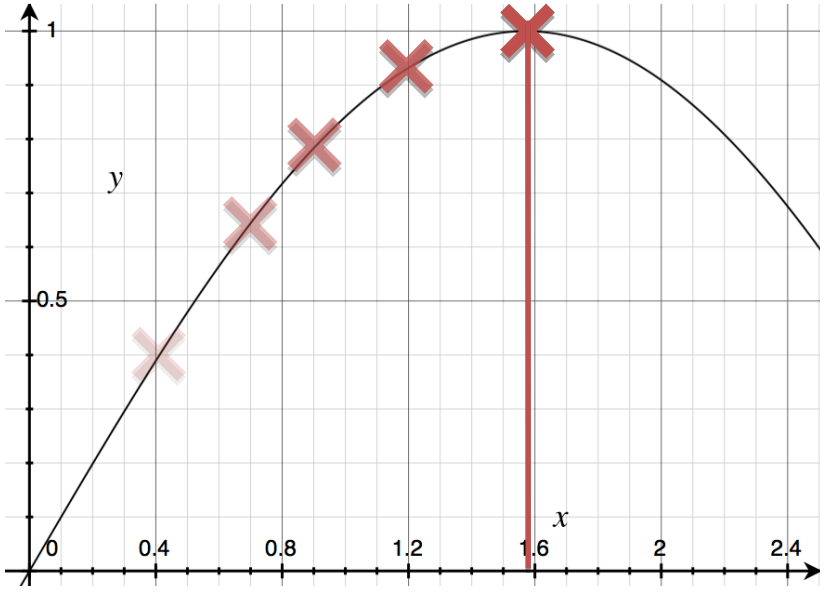
We continue hill climbing back and forth until the optimum is reached



This task is easy because sin() is a ..... function?

Hill Climbing

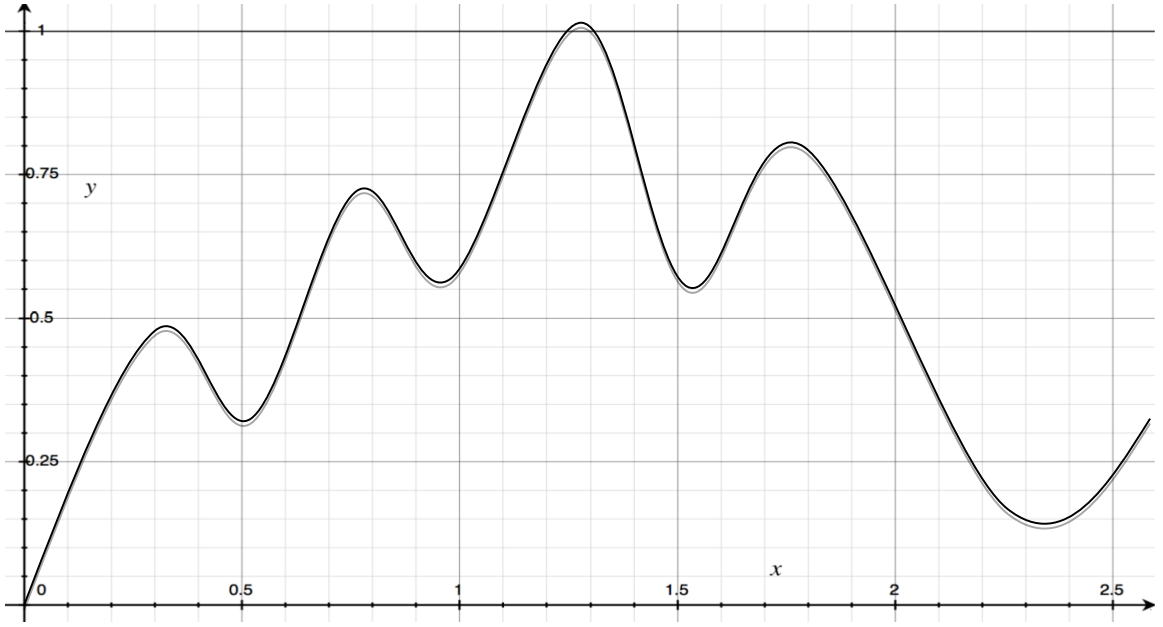
We continue hill climbing back and forth until the optimum is reached



This task is easy because sin() is a **convex** function?

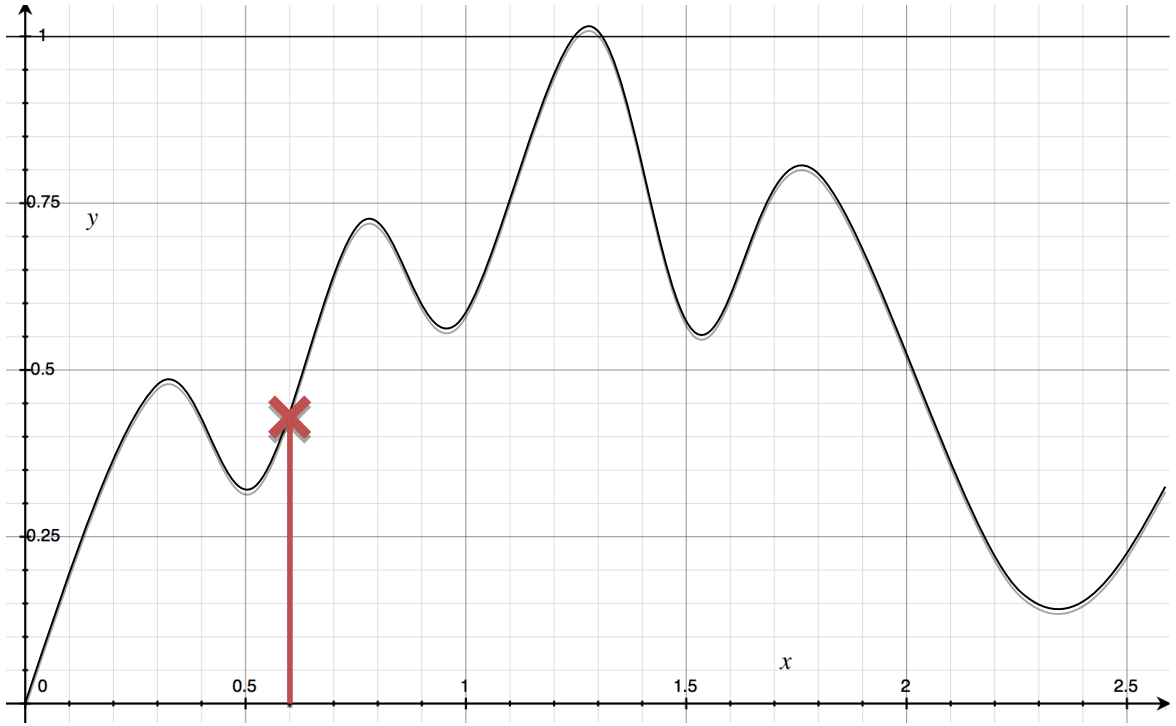
Hill Climbing

How well would hill-climbing do on this function?



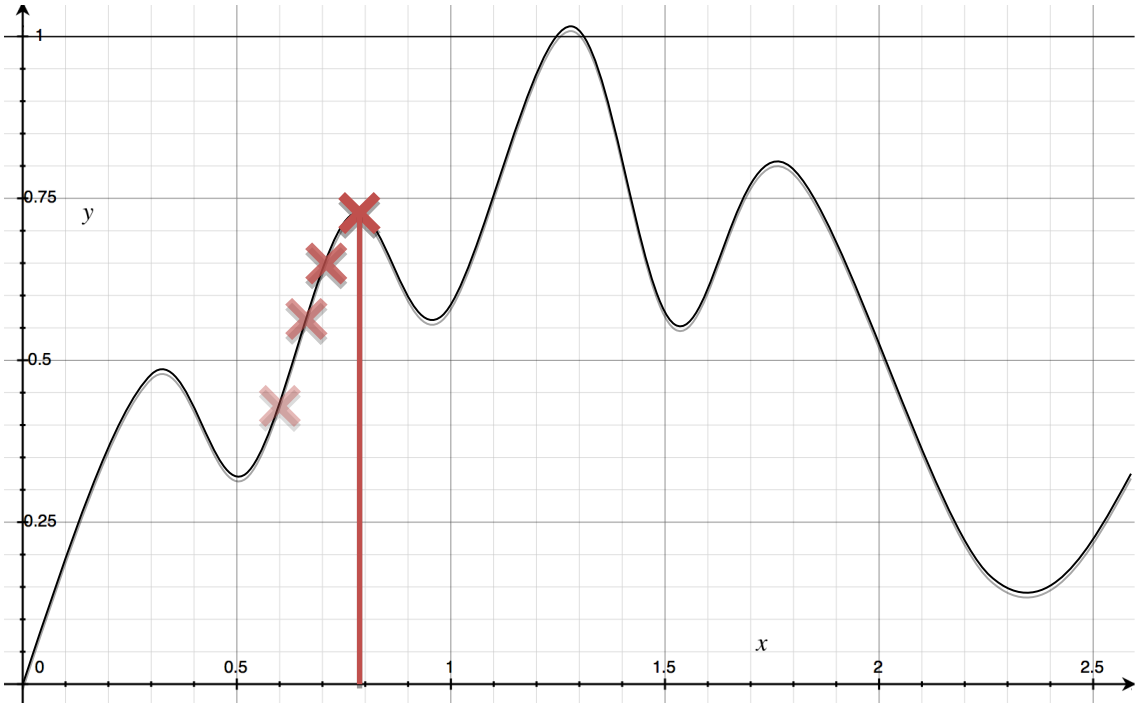
Hill Climbing

As before, you choose a random value for x. In this case 0.6 and we apply the previous algorithm



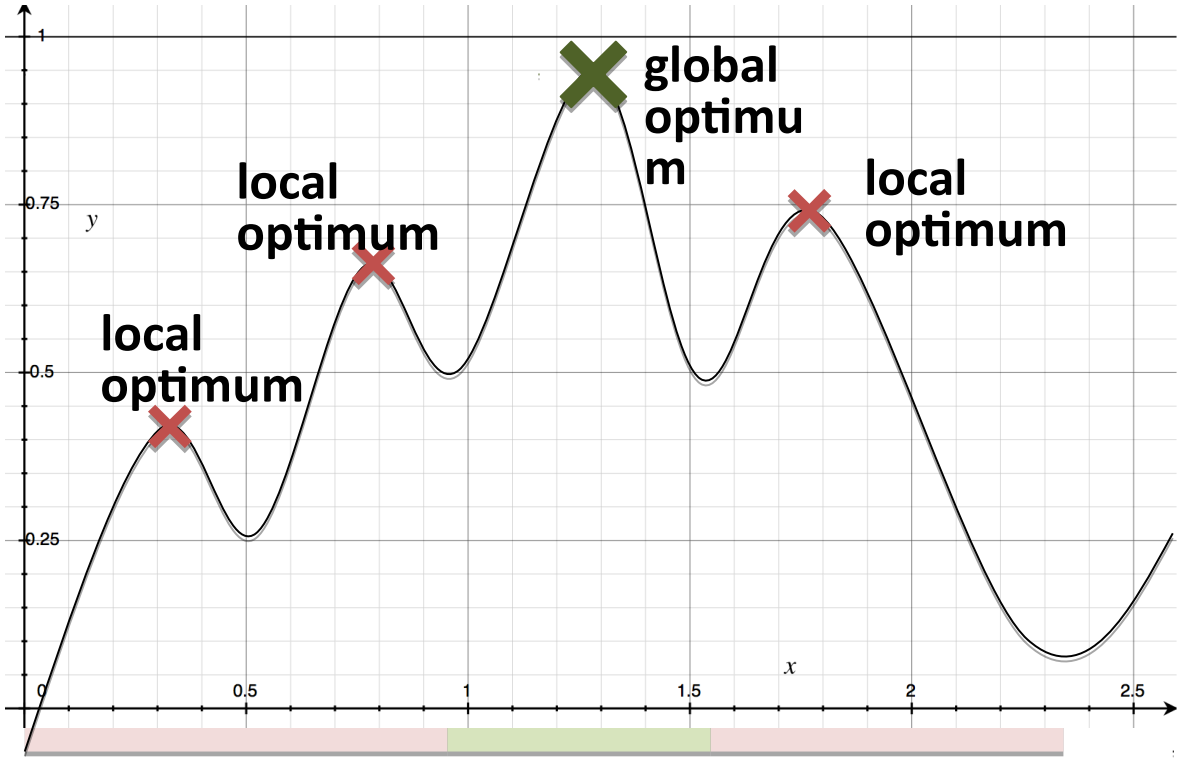
Hill Climbing

We improve our search until we achieve…….?



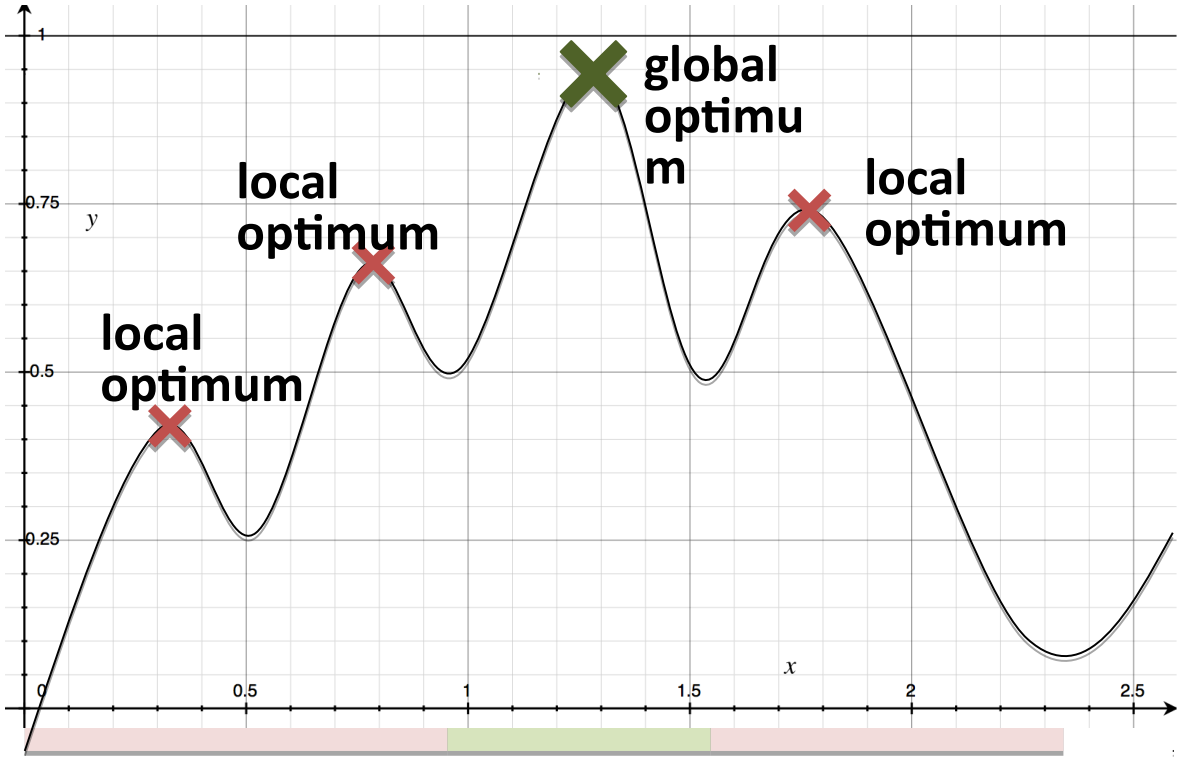
Hill Climbing

In a function like that, hill climbing most probable will get stuck in a local maxima



But why?

Hill Climbing



If the initial search point is in the **red region**, hill climbing wont be able to improve more than its closest local optimum

Local Search

Local search algorithms are derivatives of hill climbing designed to escape local optima. Examples of this kind of algorithms:

- **Simulated annealing** permits occasional downhill moves, allowing it to escape from local optima
- **Tabu search** avoids revisiting the same areas, thereby pushing itself away from local optima
- **Multi-start local search:** start again at a new position once they have converged to a local optimum

For more information, read Chapters 1 and 2 of Essentials of Metaheuristics

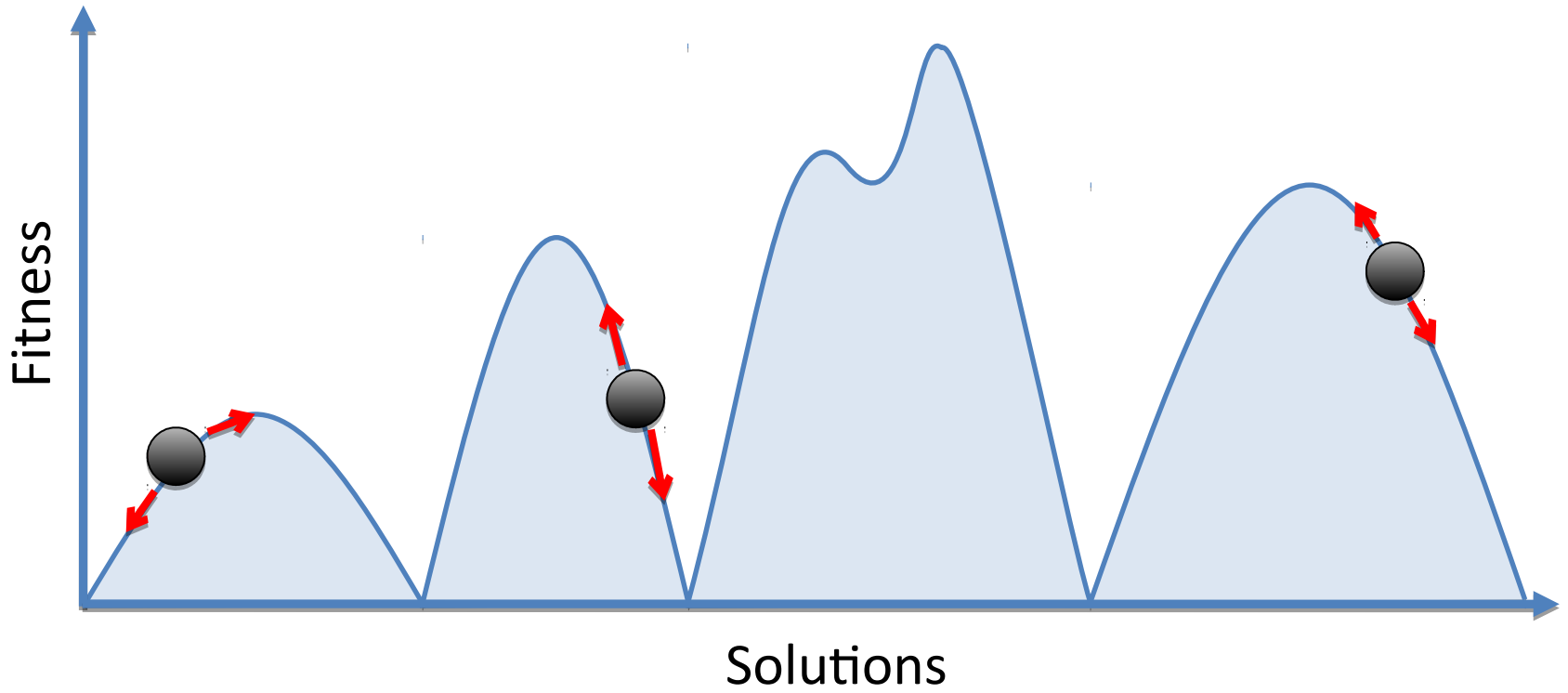
Population-based Algorithms

The strategy used by bio-inspired metaheuristics is to maintain a **population** of search points

- This really helps to avoid local optima, something that even the best local search algorithms struggle with
- In a way, it is a bit like having multiple local search processes running **in parallel**
- However, it is more than that, because the local search processes **can interact** with each other via selection and crossover
- The use of a population has been central to the success of EAs and other bio-inspired optimisation methods

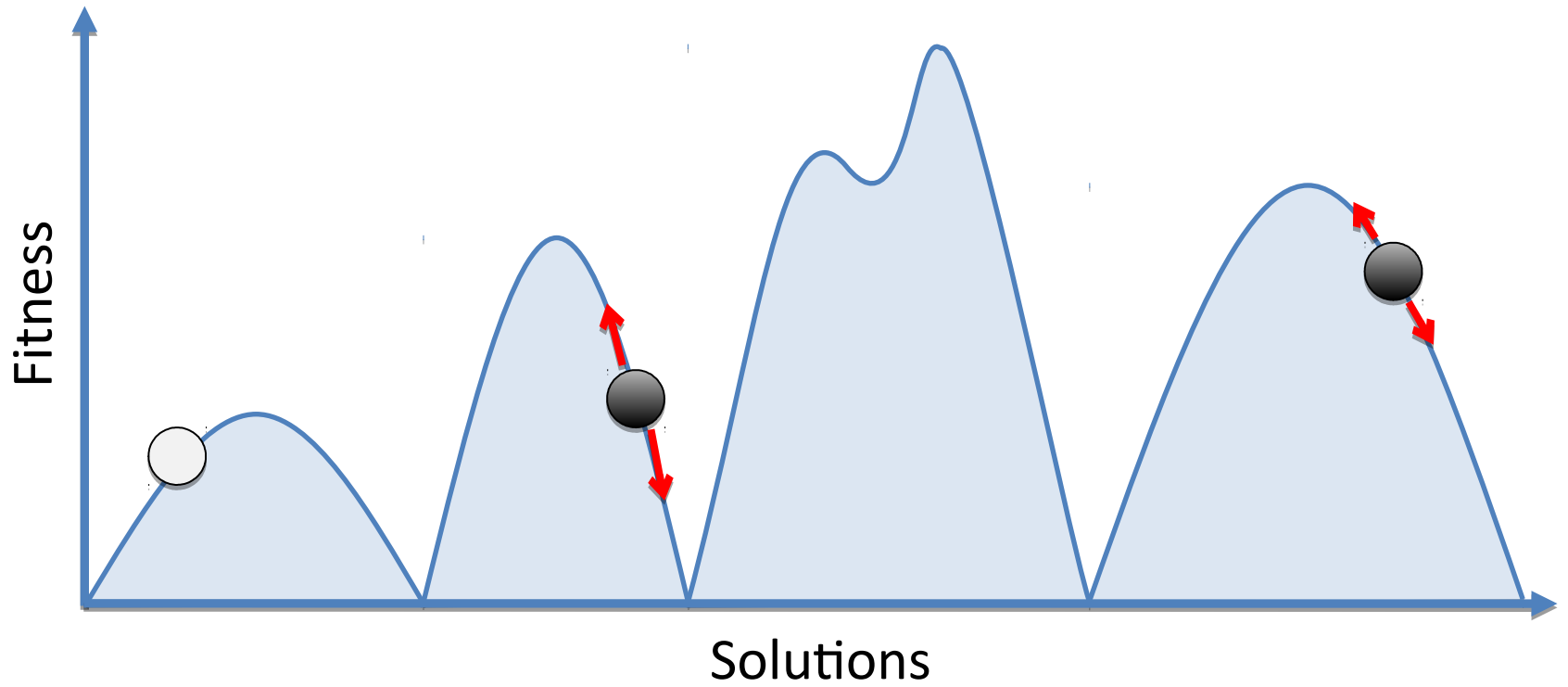
Population-based Algorithms

Consider again an imaginary **fitness landscape**. An initial random population of three solutions/chromosomes might look like this:



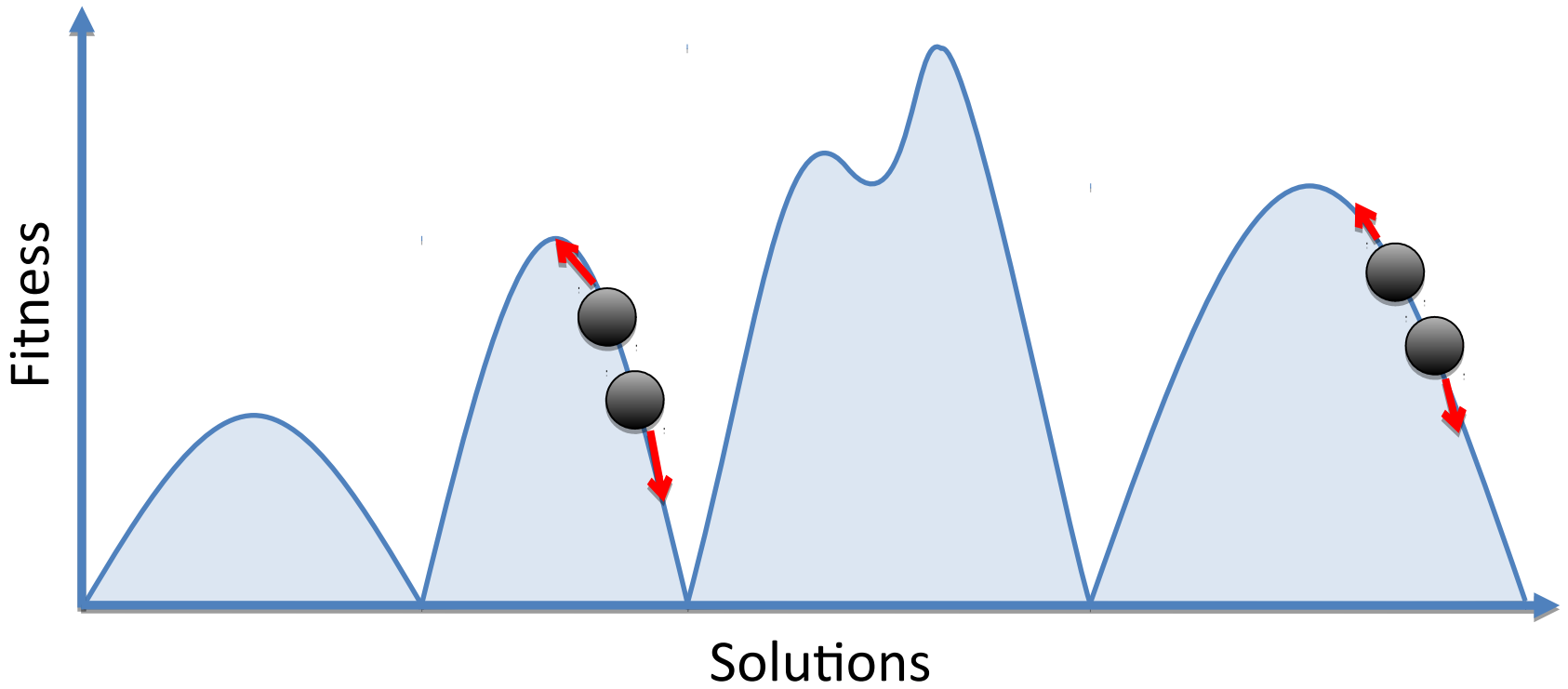
Population-based Algorithms

The selection process might remove some points. Unproductive search processes tend to be **killed off**, which is **more efficient** than just parallel local search



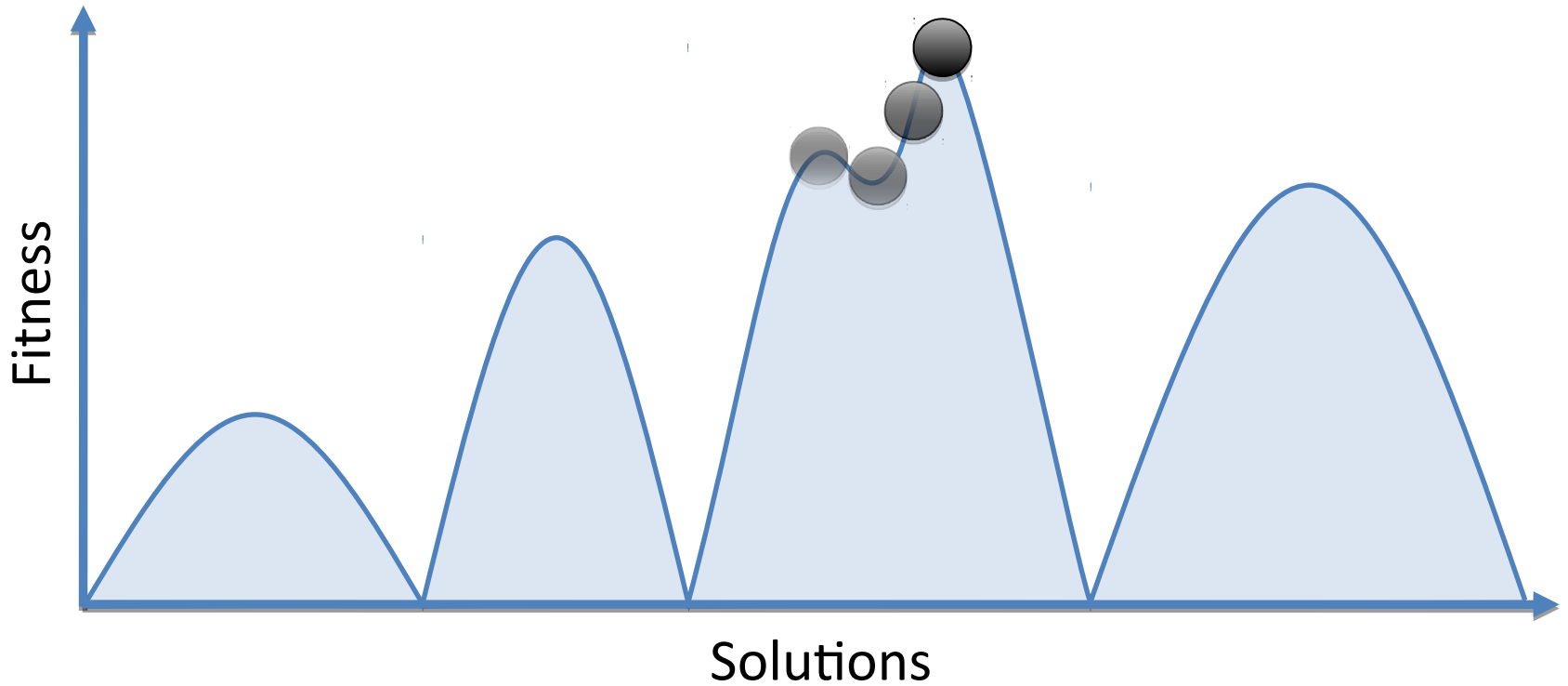
Population-based Algorithms

The mutation process might then add some points, by sampling new points near existing points. Mutation and selection can then lead to **the next optima**



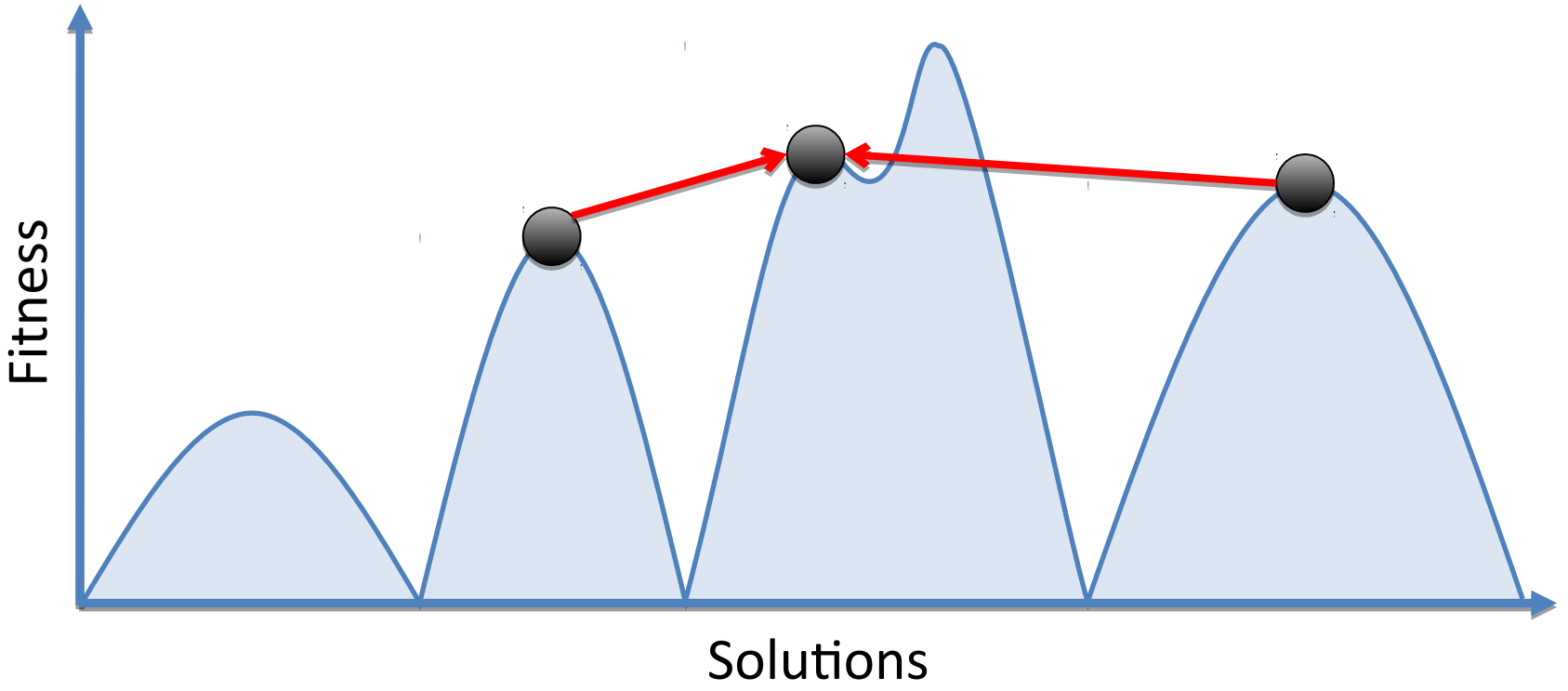
Population-based Algorithms

Mutation and selection can escape local optima, because they are stochastic, sometimes a good search point will be replaced by a less good search point, if they are close to each other



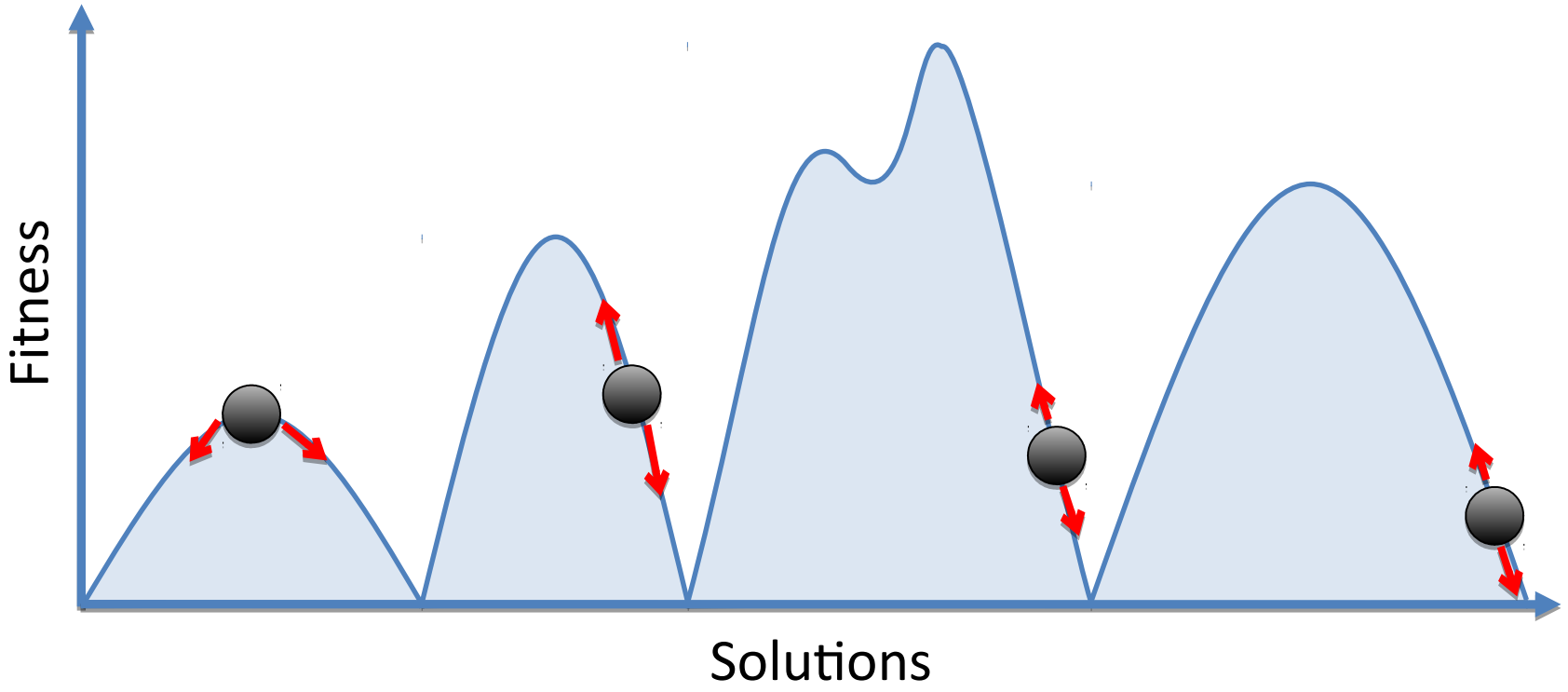
Population-based Algorithms

But if they are far, this is where crossover can be helpful, since allows large jumps



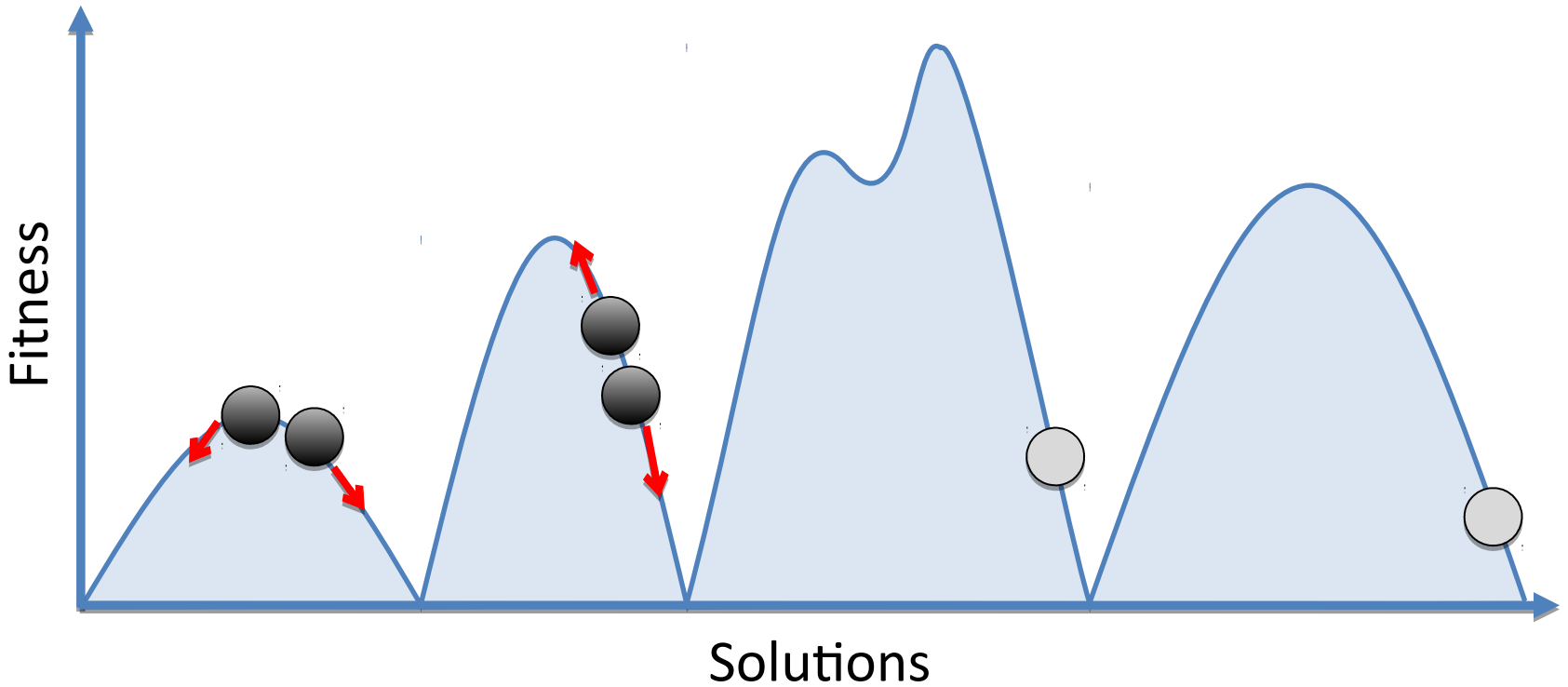
Population-based Algorithms

However, there are potential **pitfalls**: **Too much selection pressure** can impede progress. Consider this initial population:



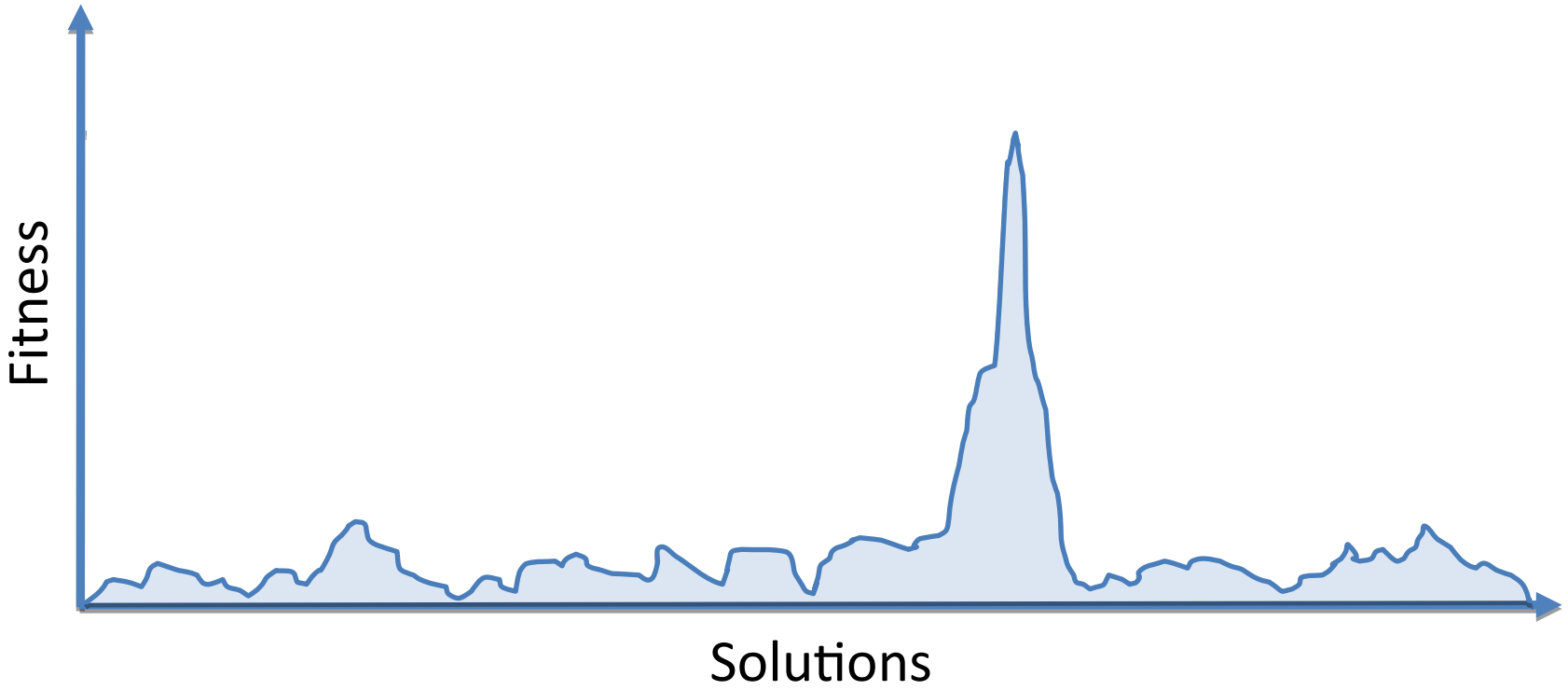
Population-based Algorithms

Here, the less promising chromosomes has been removed and this has moved the search away from the best areas



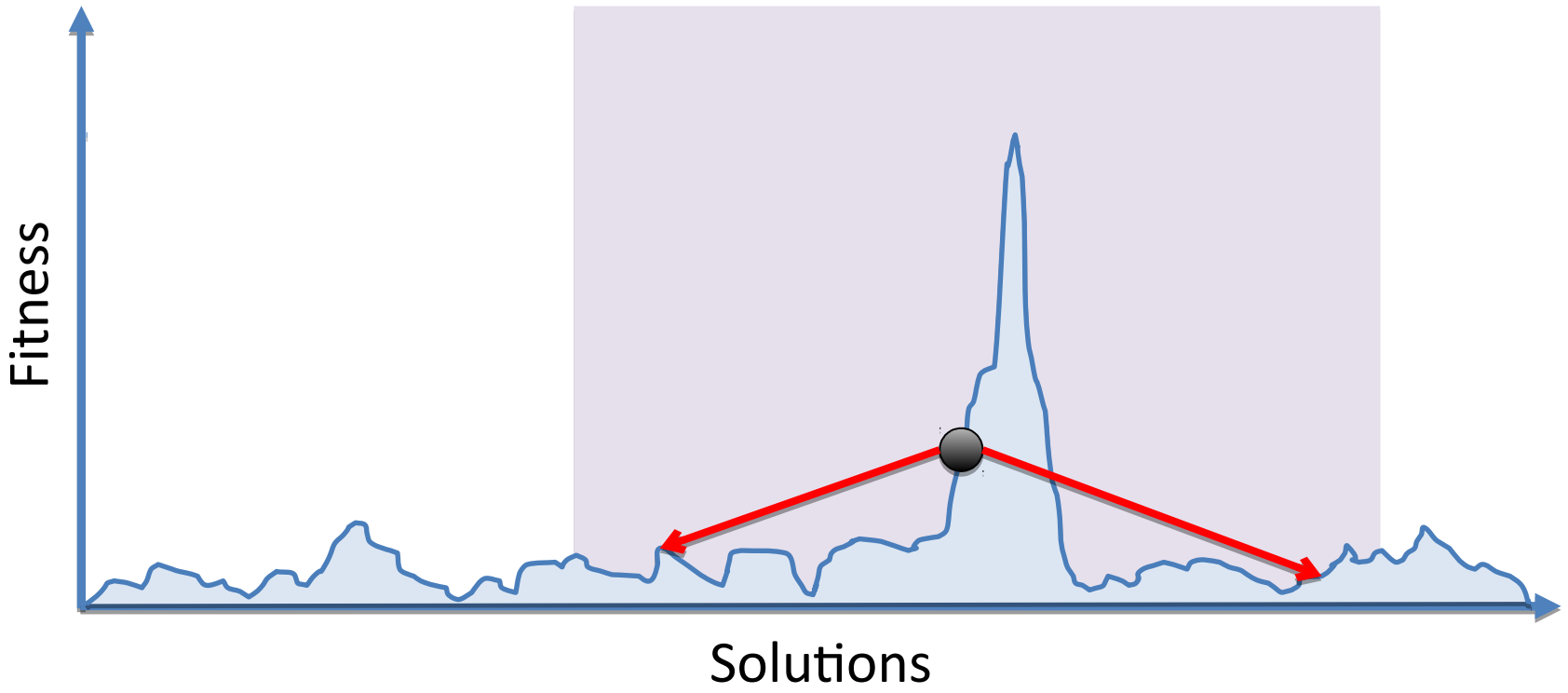
Population-based Algorithms

You also have to be careful with crossover rates. In large real world problems, there are often tiny areas where the decent solutions lurk



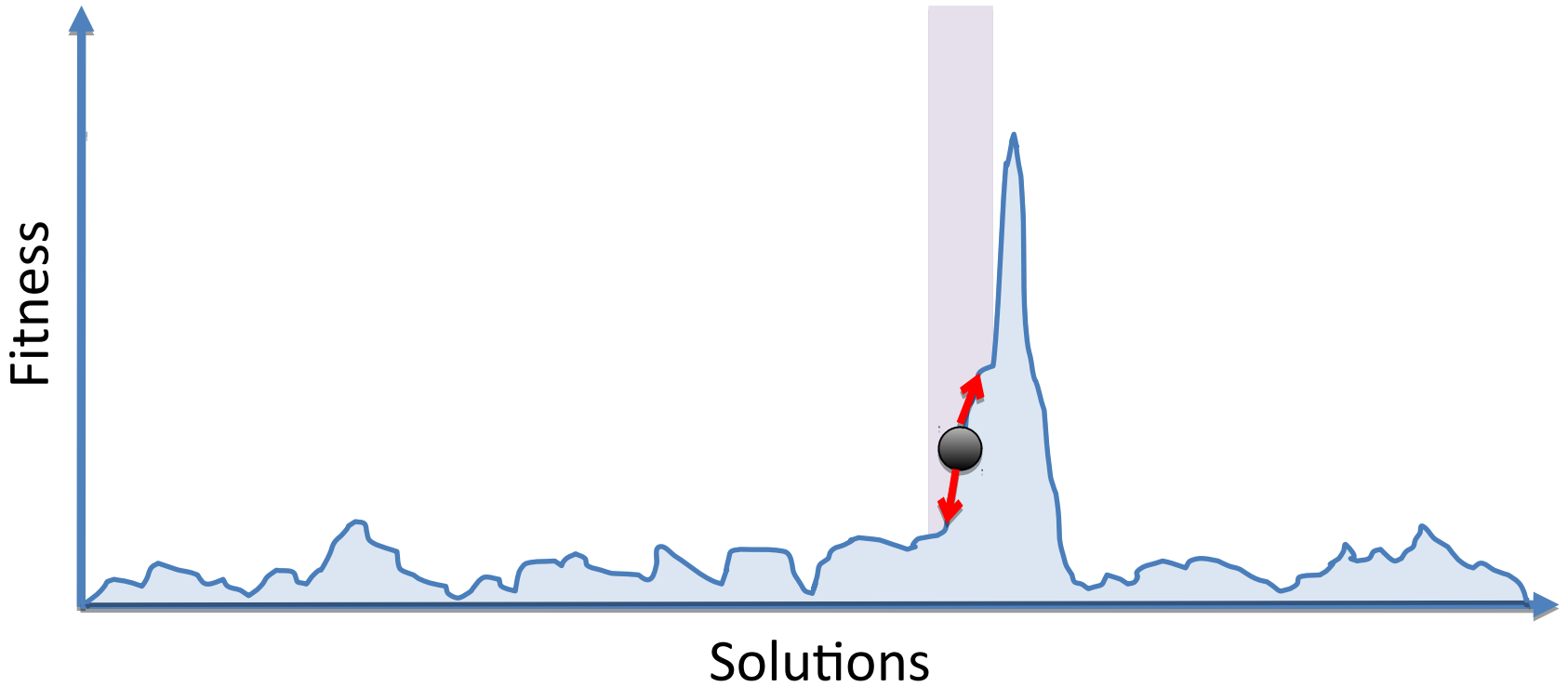
Population-based Algorithms

You also have to be careful with crossover. Big jumps will most likely take search away from productive regions of the search space



Population-based Algorithms

For this reason, its usually best to use mutation operators and rates that restrict the neighbourhood



Problems and Algorithms

Choosing the best algorithm for a problem is itself a difficult and unsolved problem

- It's hard to tell what a particular problem's fitness landscape looks like
- This means it is hard to know which optimisation strategies will work best for a particular problem
- Evolutionary algorithms are a good choice because they work over lots of different fitness landscapes
- Though it is still hard to know which are the best parameter settings
- You usually have to do experiments to work out the right mutation rate, selection pressure etc.

One of the very first applications. Determine the internal shape of a two-phase jet nozzle that can achieve the maximum possible thrust under given starting conditions

Ingo Rechenberg was the very first, with pipe-bend design. This is slightly later work in the same lab, by Schwefel

Starting point

Result

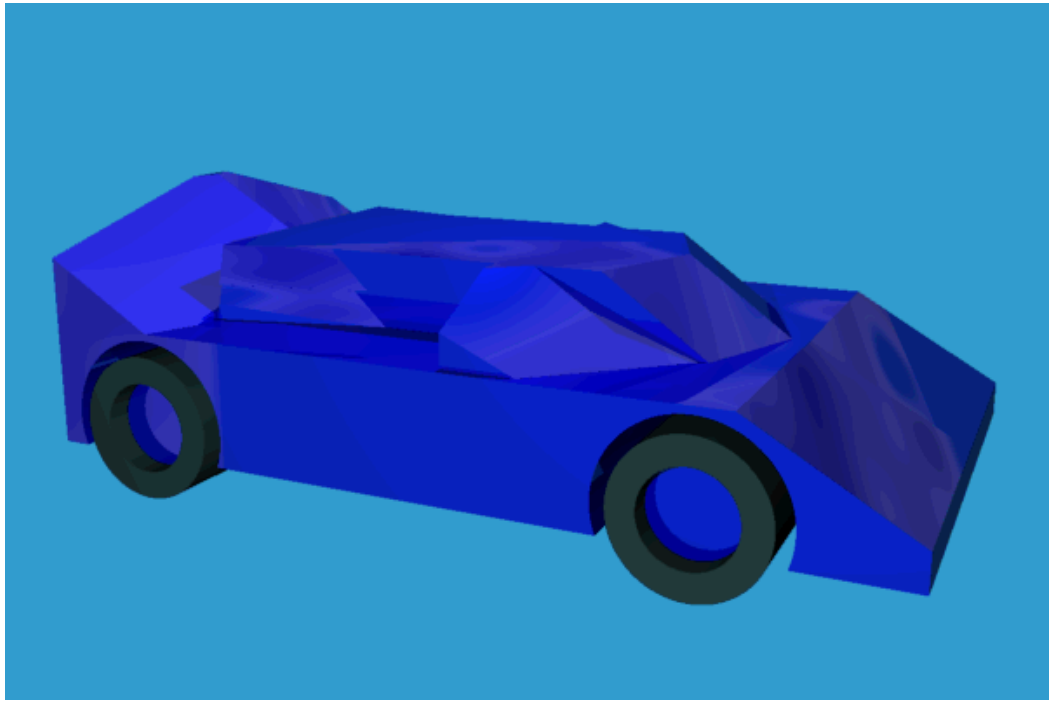
EA (ES) running

Design freedom → entirely new and better designs based on principles we don't yet understand.

Some extra slides if time, illustrating some high-profile EAs

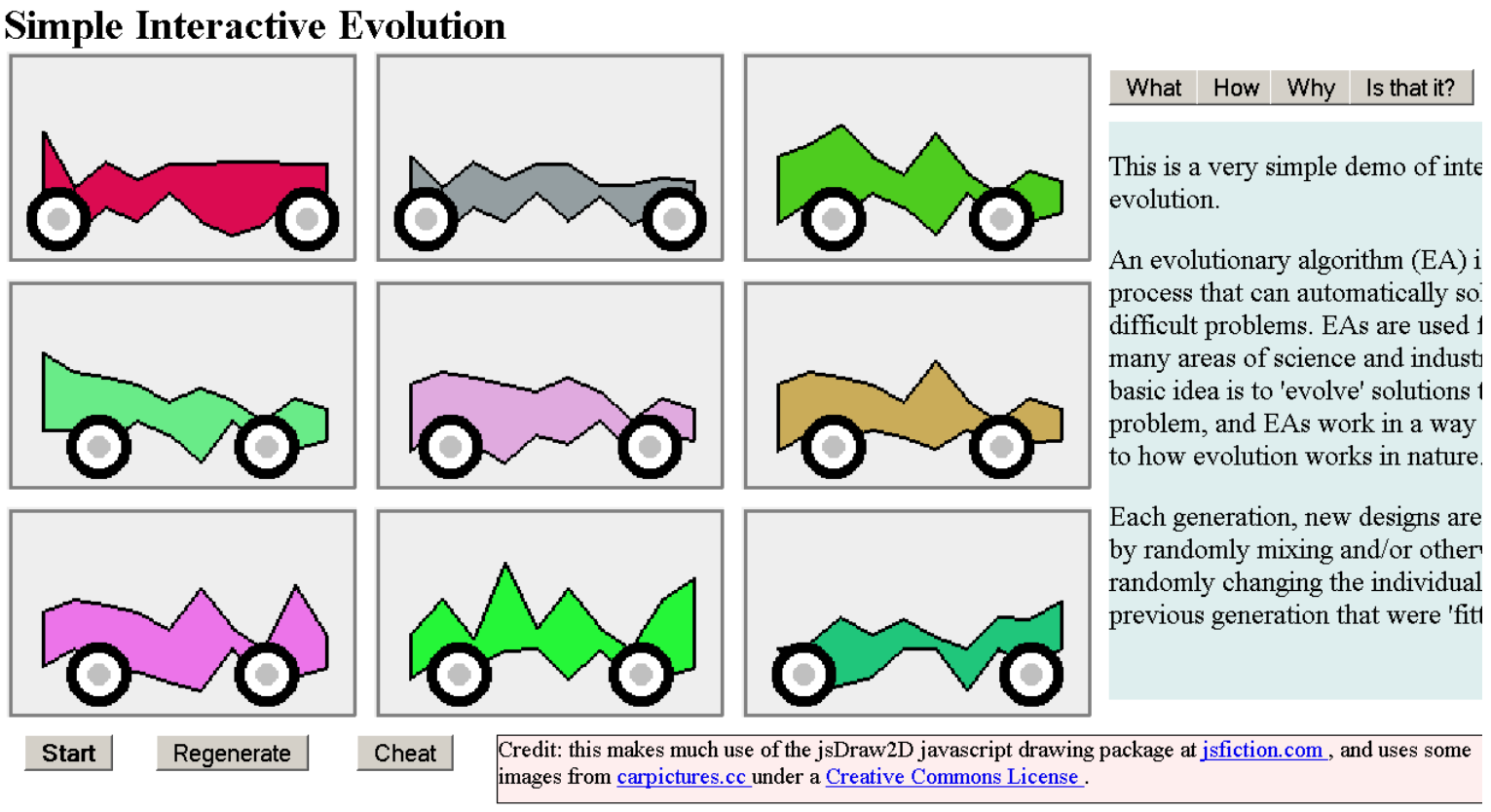
- An innovative EC-designed
- Propellor from Evolgics GmbH,
- Associated with Rechenberg's group.

Bentley’s thesis work

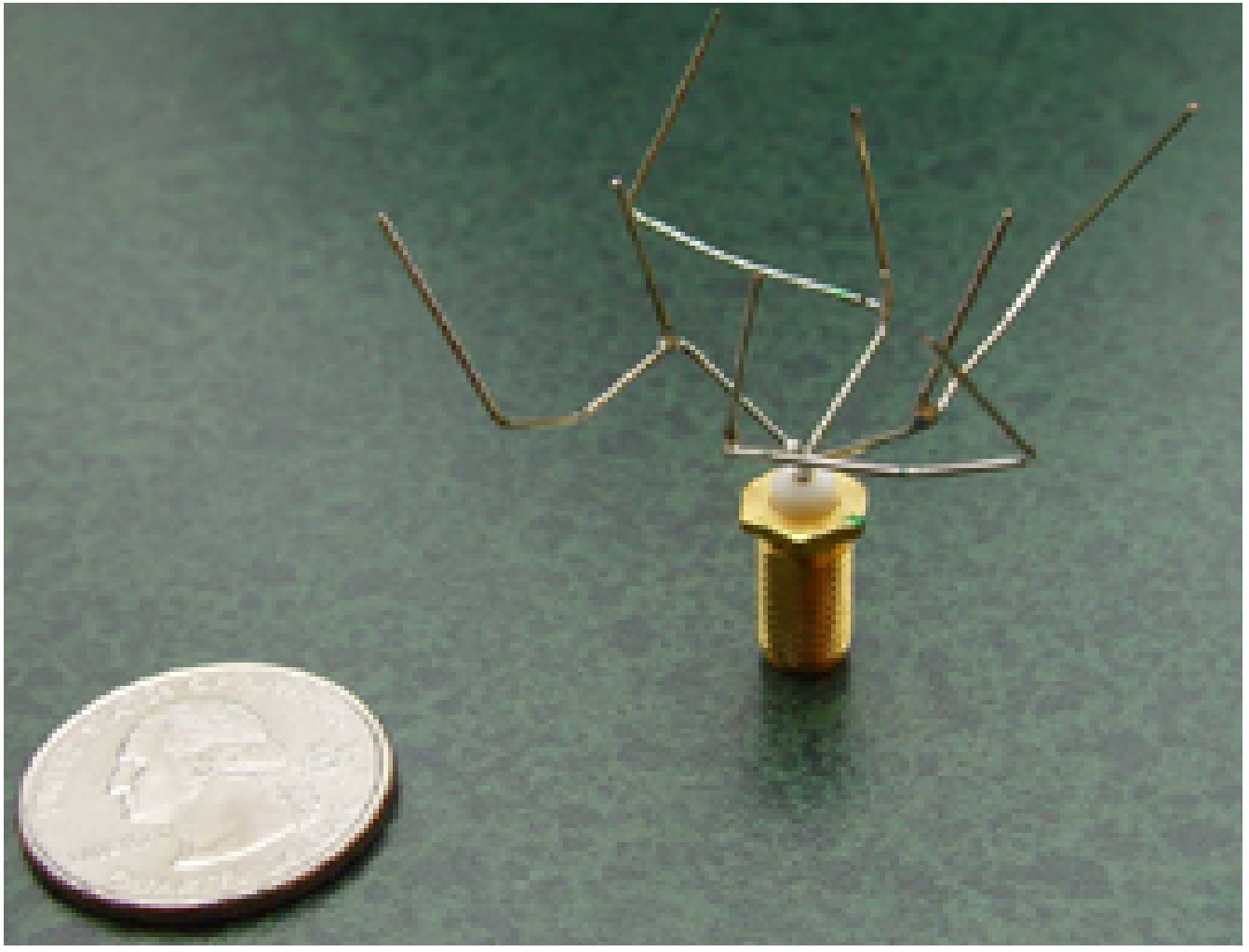


- Fixed wheel positions
- Constrained bounding area,
- Chromosome is a series of slices\fitnesses evaluated via a simple airflow simulation

http://www.macs.hw.ac.uk/~dwcorne/Teaching/iea.html



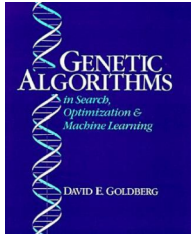
We knew something like this would happen



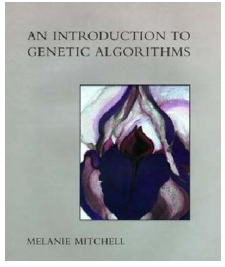
Credit: Jason Lohn

Bibliography for Evolutionary Algorithms

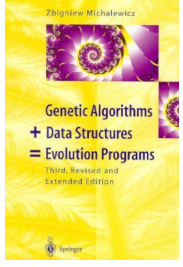
Three traditional books about EA/GA:



1.- Genetic Algorithms in Search, Optimization and Machine Learning
1989 by David E. Goldberg



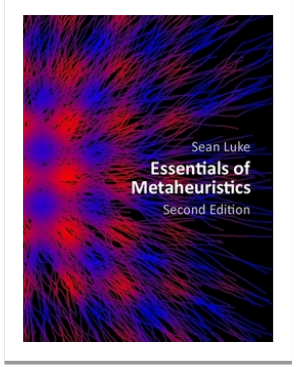
2.- An Introduction to Genetic Algorithms (Complex Adaptive Systems)
1998 by Melanie Mitchell



3.- Genetic Algorithms + Data Structures = Evolution Programs
1998 by Z. Michalewicz

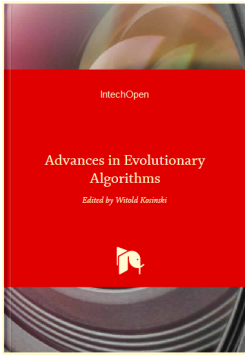
Free eBooks for Evolutionary Algorithms

Essentials of Metaheuristics



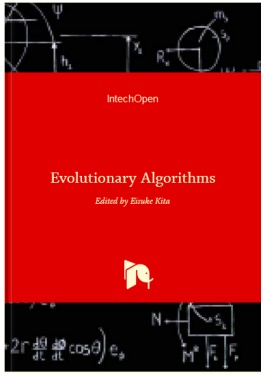
https://cs.gmu.edu/~sean/book/metaheuristics/

Advances in Evolutionary Algorithms



https://www.intechopen.com/books/advances_in_evolutionary_algorithms

Evolutionary Algorithms



https://www.intechopen.com/books/evolutionary-algorithms

Global Optimization Algorithms – Theory and Application –

http://www.it-weise.de/projects/bookNew.pdf