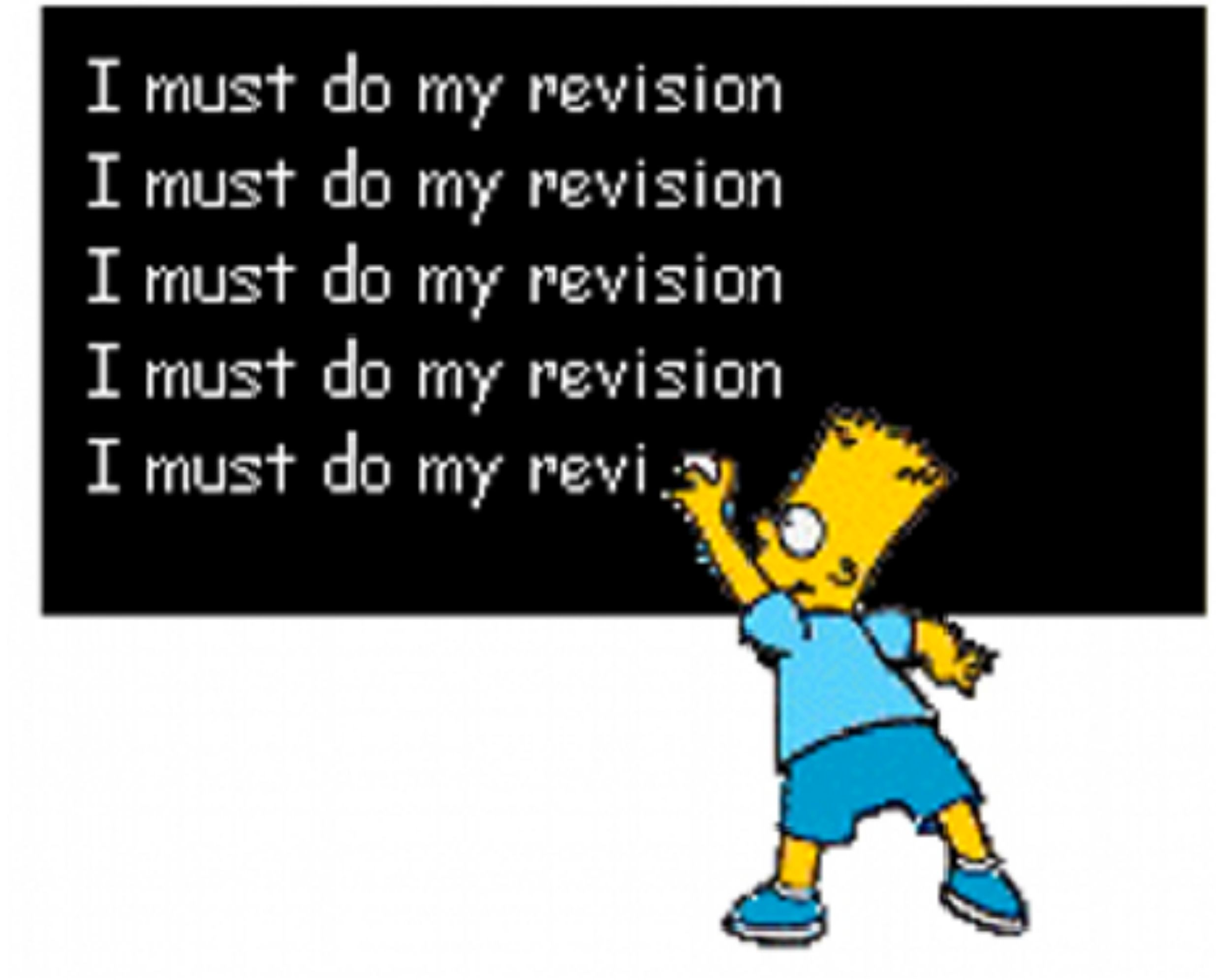
CO3091 - Computational Intelligence and Software Engineering

Lecture 27



Feature Selection and Revision of Optimisation Algorithms

Leandro L. Minku

Image from: http://www.literaturedelights.com/wp-content/uploads/2016/03/revision-simpsons.jpg

Overview

- Brute-force search
- Hill-Climbing
- Simulated Annealing
- Evolutionary Algorithms
- Constraint Handling
- Multi-objective Evolutionary Algorithms
- What would be a good algorithm / design for feature selection?

Doodle Poll

- The doodle poll to sign up for an optional lab session this Friday will be released this afternoon.

Brute-Force Search

- Brute-force search = exhaustive search = generate and test.
- Systematically enumerate all possible candidates for the solution and check which one is the best.
- What are the advantages and disadvantages of brute force?
- Guaranteed to find the optimal solution.
- Can we use brute-force search to solve optimisation problems?

Image from: http://vignette2.wikia.nocookie.net/creepypasta/images/8/80/Creepycemetery.jpg/revision/latest?cb=20131013135547

Problem: high computational complexity.

Computational Intelligence

- No single definition.
- Heuristic algorithms, which aim to find good solutions to problems in a reasonable amount of time.*
  - Typically not guaranteed to find the optimum, but able to find near-optimal solutions.
  - What type of problems could benefit from heuristic algorithms?
    - Problems where we cannot afford enumerating all possible solutions to guarantee optimality.
    - Problems where no exact algorithm exists to solve the problem in polinomial time.
    - Problems where a near-optimal solution is acceptable.
- Learning algorithms that can build models based on data.

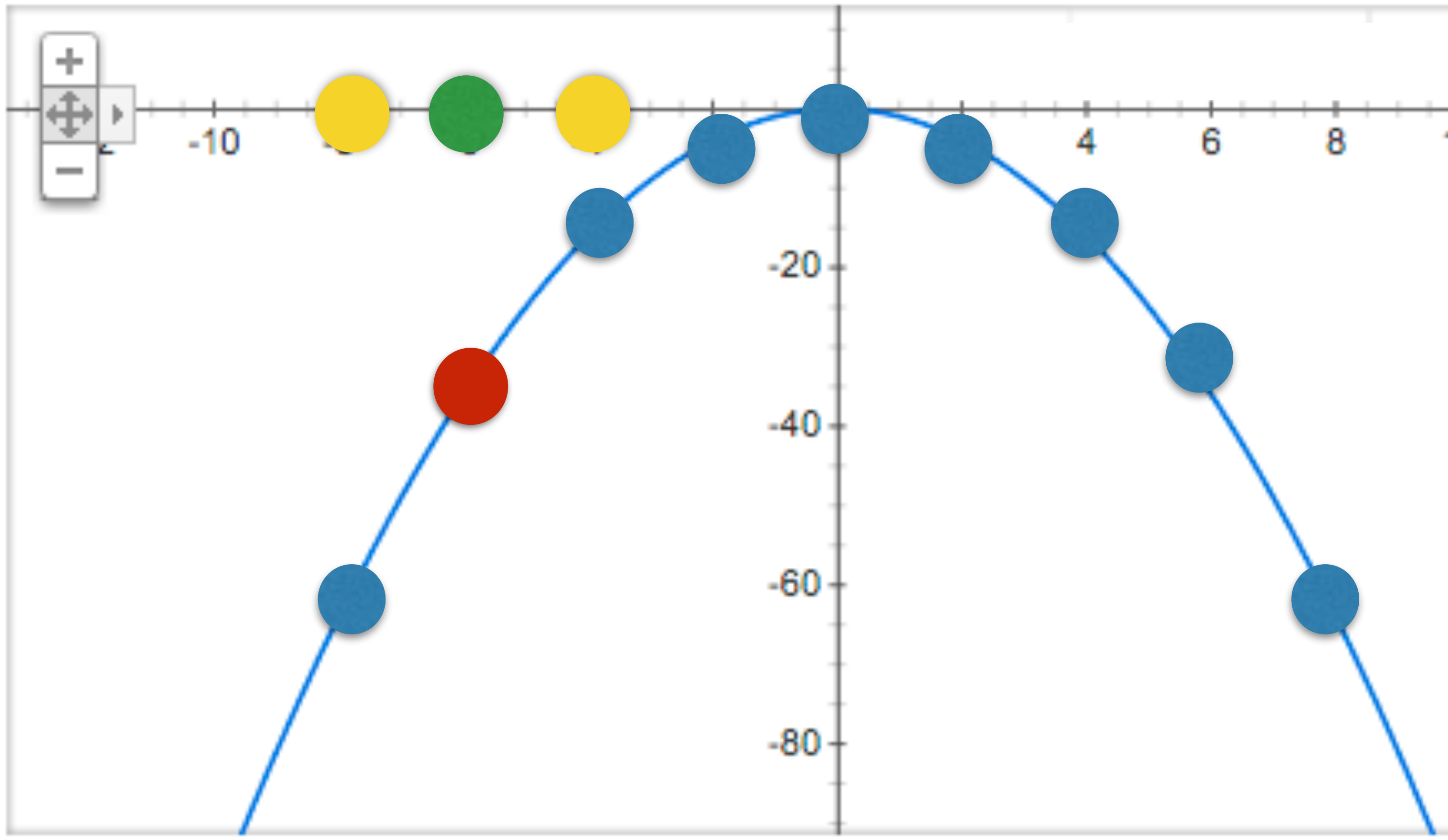
* http://ukci.cs.manchester.ac.uk/intro.html

Hill-Climbing — Illustrative Example

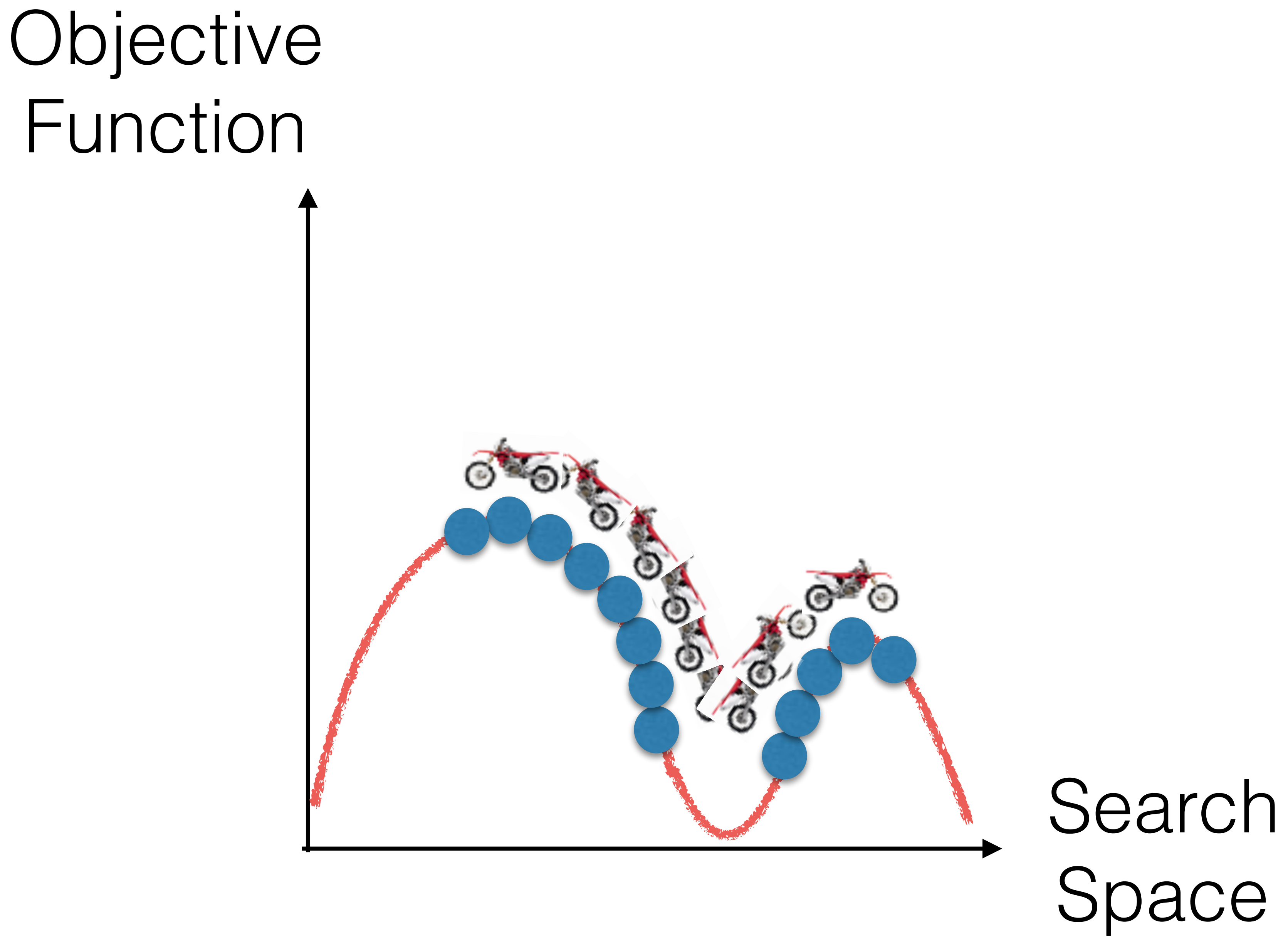
Hill-Climbing (assuming maximisation)

```
1. current_solution = generate initial solution randomly

2. Repeat:

    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour
```

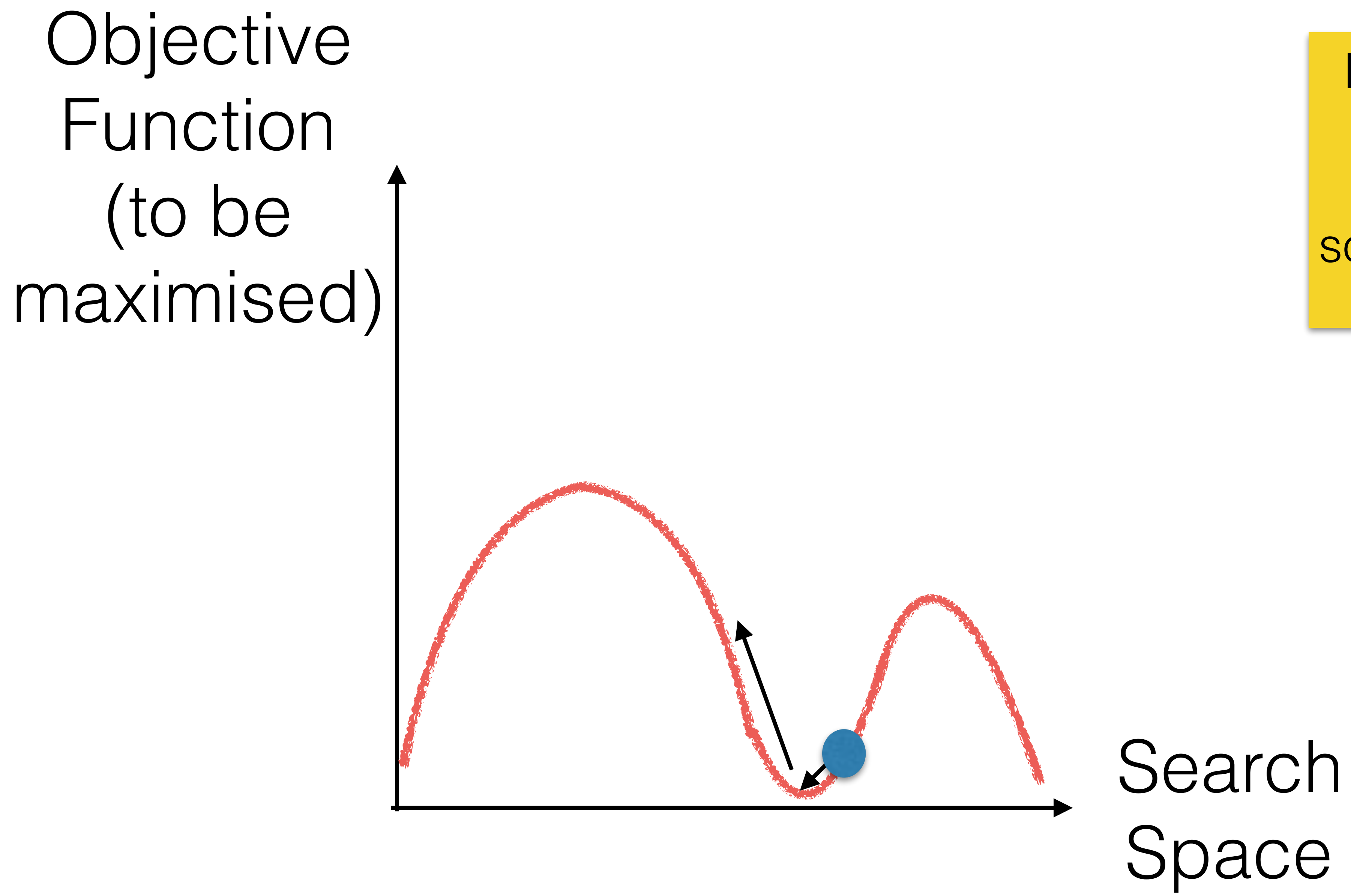
maximise $f(x) = -x^2$



Hill-Climbing — General Idea



Simulated Annealing — Motivation



If we could sometimes accept a downward move, we would have some chance to move to another hill.

In simulated annealing, instead of taking the best neighbour, we pick a random neighbour and give some chance to accept it, even if its quality is worse than that of the current solution.

Metallurgy Annealing

- A blacksmith heats the metal to a

High temperature = high probability

molecules can move fast and randomly.

Image from: http://www.stormthecastle.com/indeximages/sting-steel-thumb.jpg

- The blacksmith then lets it cool down slowly.

Low temperature = low probability

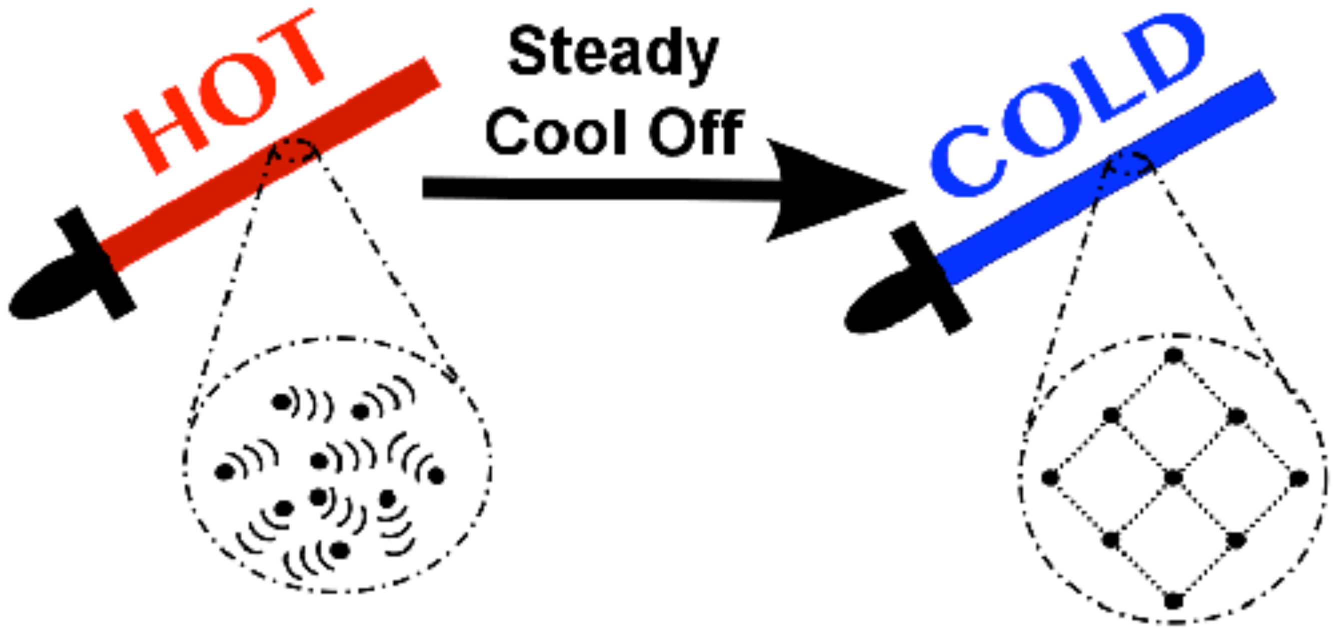
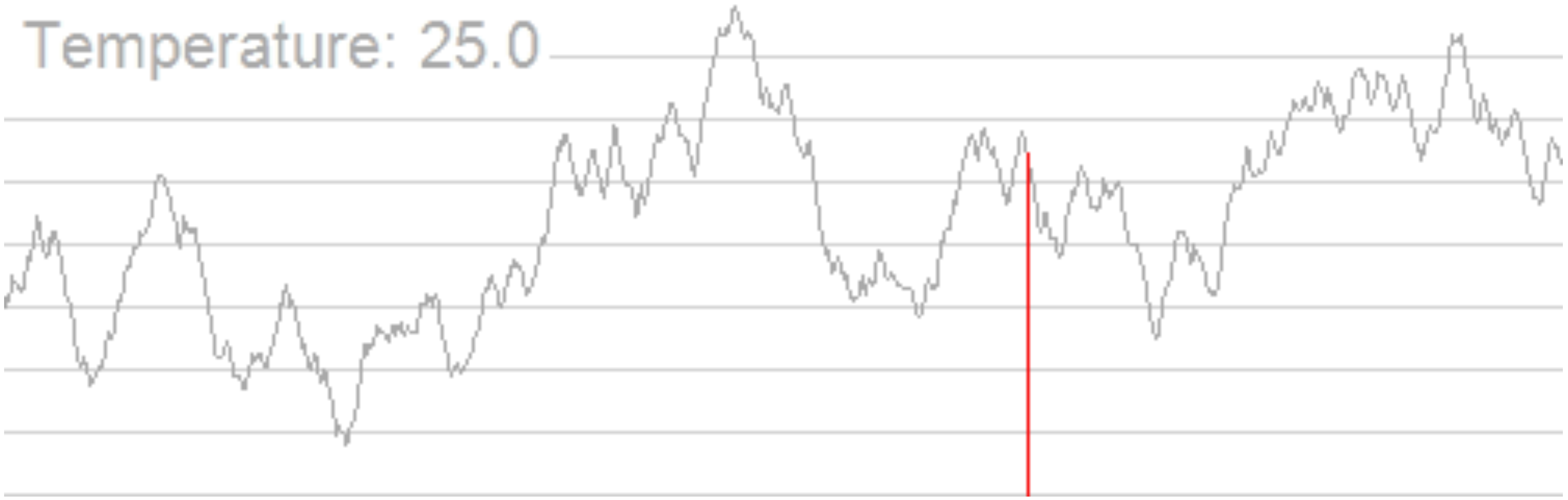


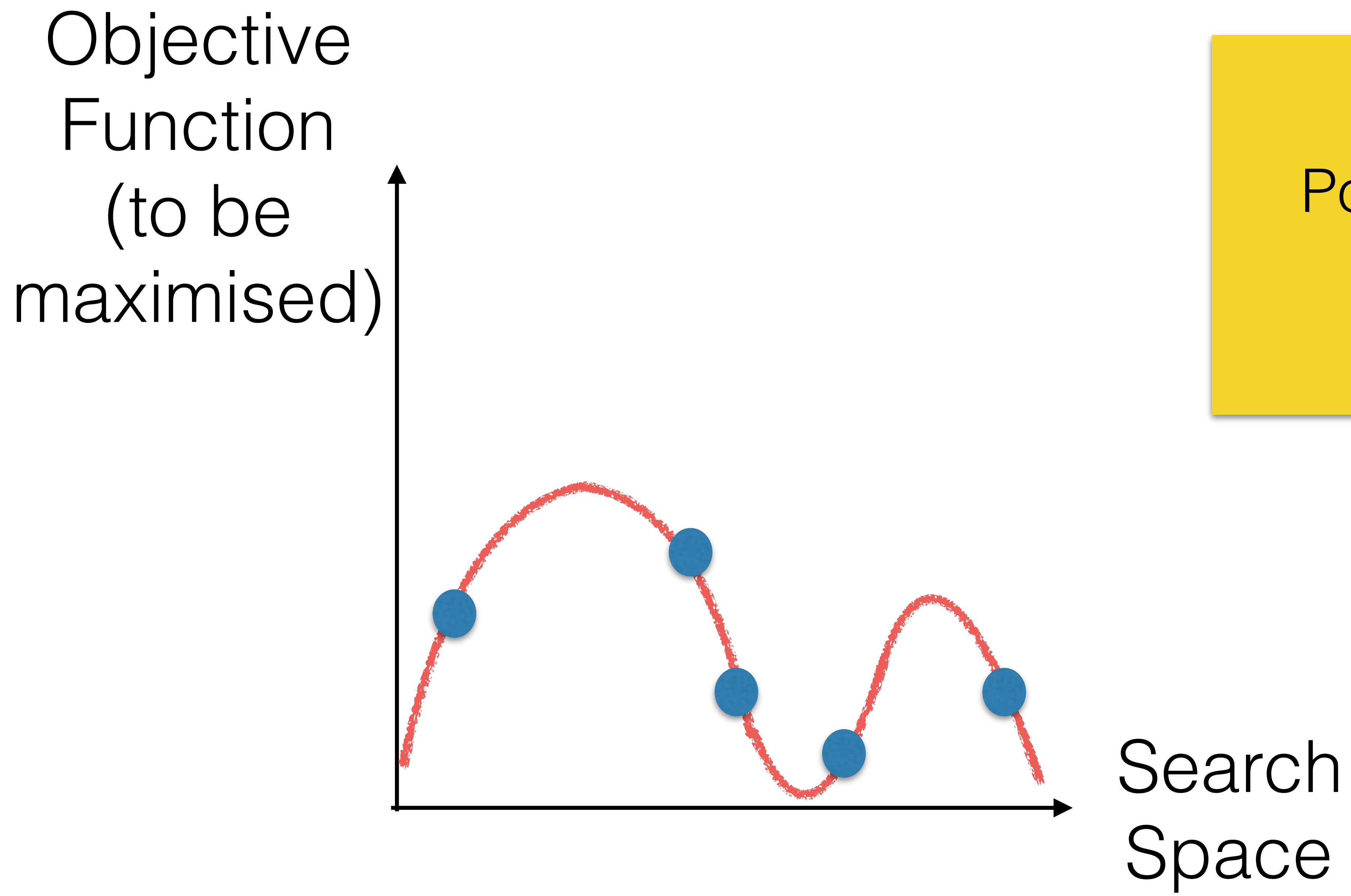
Image from: http://2.bp.blogspot.com/--kOlrodykkg/UbfVZ0_I5HI/AAAAAAAAAJ4/0rQ98g6tDDA/s1600/annealingAtoms.png

- This makes the sword stronger than the untreated steel.



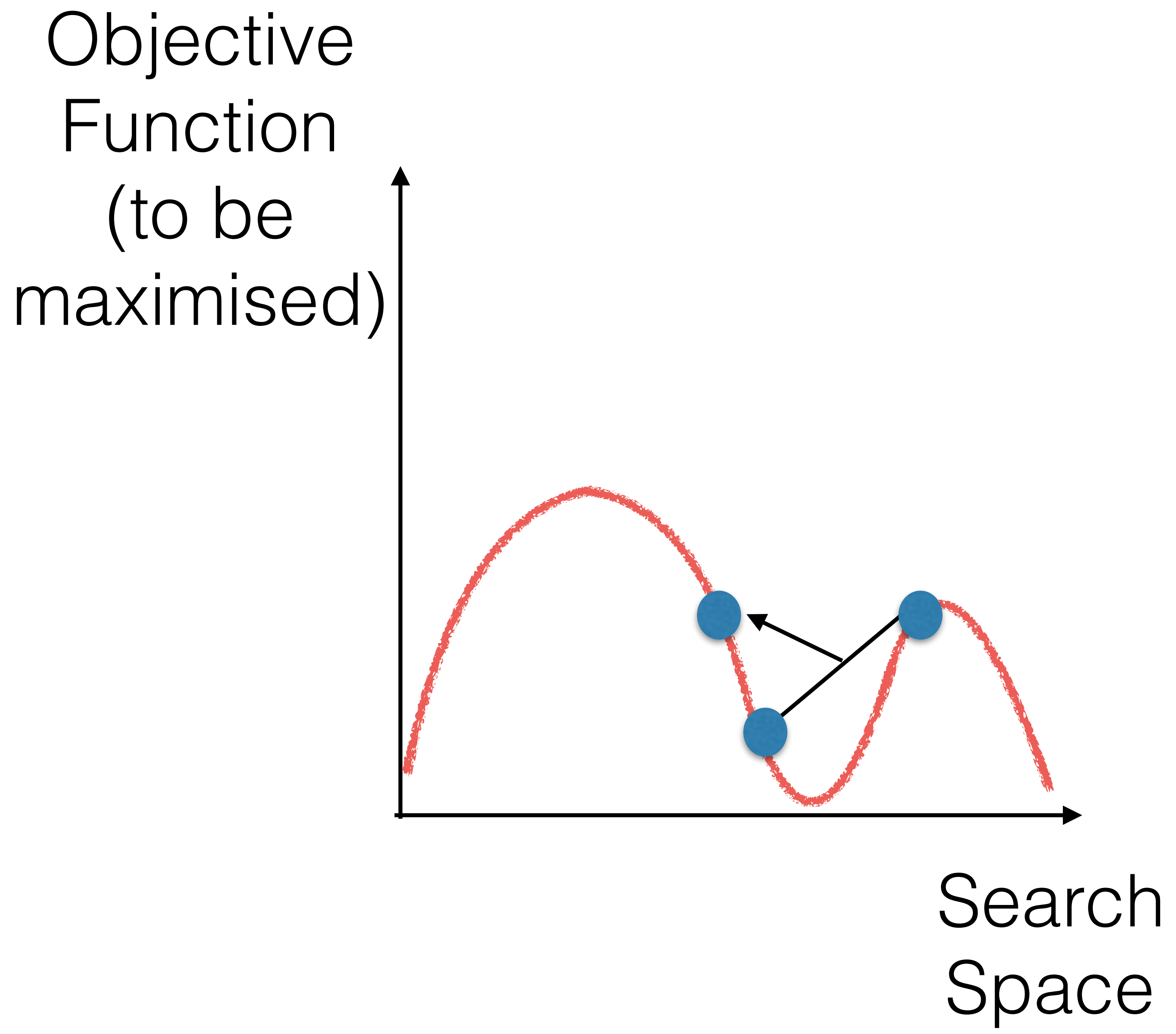
[By Kingpin13 - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=25010763]

Evolutionary Algorithms (EAs)



Population helps exploration.

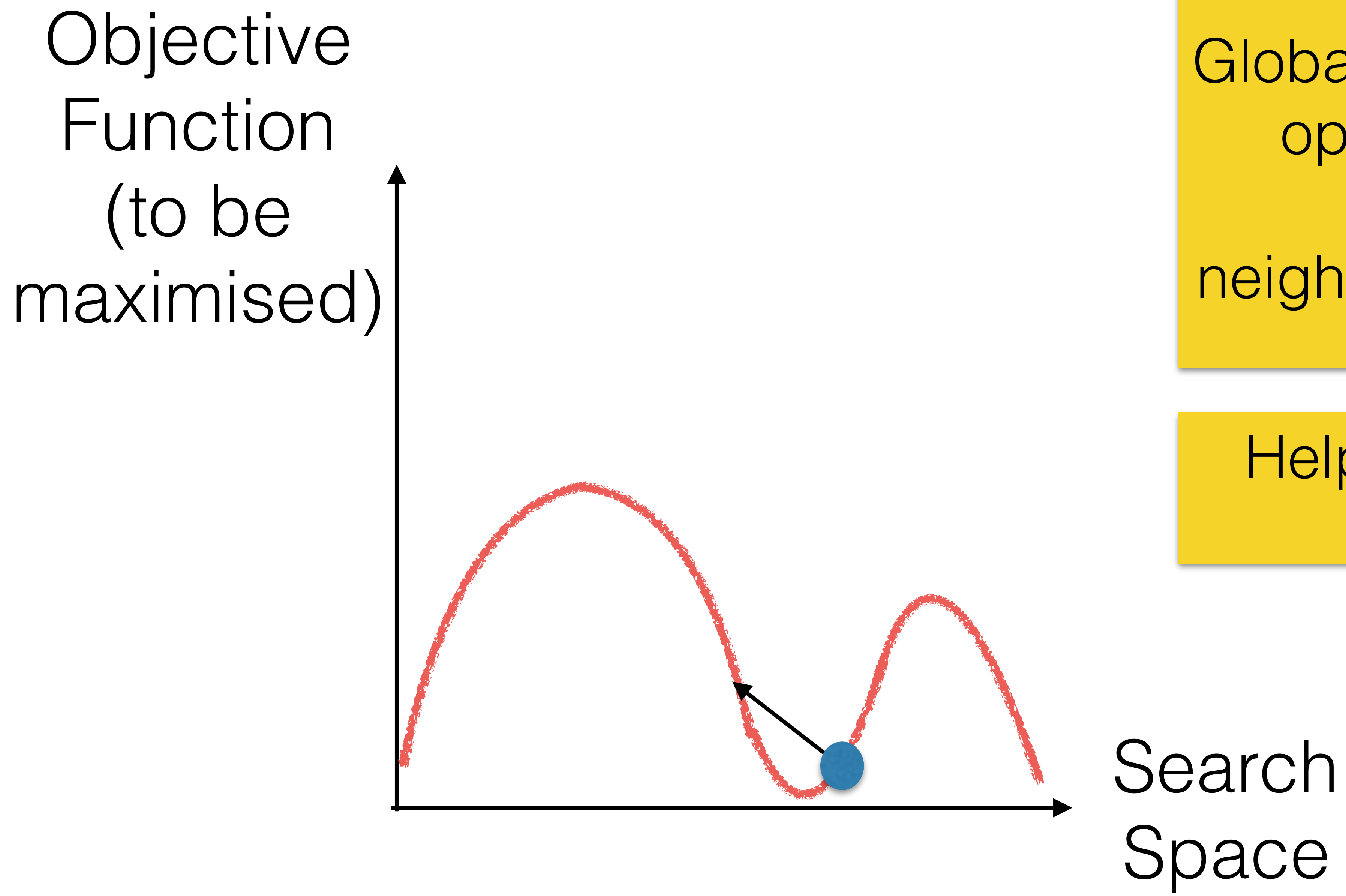
Evolutionary Algorithms (EAs)



Bad individuals have some (small) chance to reproduce.

Helps to avoid local optima.

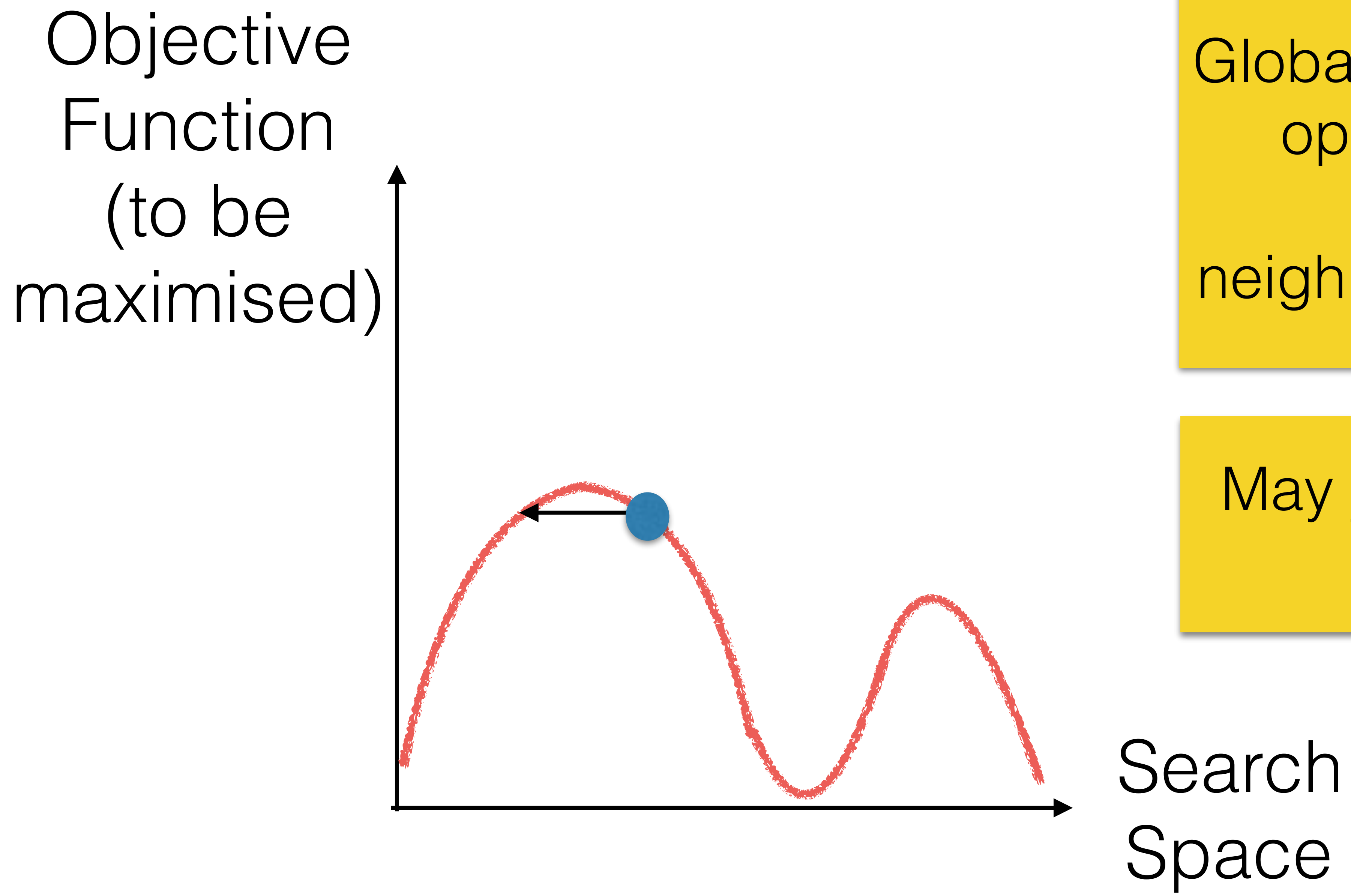
Evolutionary Algorithms (EAs)



Global search: variation operators are not restricted to neighbouring solutions.

Helps to avoid local optima.

Evolutionary Algorithms (EAs)



Global search: variation operators are not restricted to neighbouring solutions.

May jump passed the optimum.

EA's Pseudocode

```
Evolutionary Algorithm
  1. Initialise population
  2. Evaluate each individual (determine their fitness)
  3. Repeat (until a termination condition is satisfied)
      3.1 Select parents
      3.2 Recombine parents with probability Pc
      3.3 Mutate resulting offspring with probability Pm
      3.4 Evaluate offspring
      3.5 Select survivors for the next generation
```

Parents and/or survivors selection is based on a selective pressure towards fitter (better) individuals.

EA Design

- Representation
- Initialisation
- Recombination
- Mutation
- Parent selection
- Survivor selection
- Termination criteria

- Fitness function
- How to deal with constraints

Feature Selection Problem Formulation

- Design variable:
    - $v \in \{0,1\}^N$, where $v_i = 0$ represents that input feature i is not selected, $v_i = 1$ represents that input feature i is selected, and N is the number of available input features.
- Objective function:
    - Validation error of the model built based on the training set using only the selected features.
- Constraints:
    - None.

What optimisation algorithm would you choose to solve this problem?

EA Design

Evolutionary algorithms are popular choices of algorithms due to their ability to avoid local optima.

- Representation
- Initialisation
- Recombination
- Mutation
- Parent selection
- Survivor selection
- Termination criteria

- Fitness function
- How to deal with constraints

EA Design

Evolutionary algorithms are popular choices of algorithms due to their ability to avoid local optima.

Representation
Initialisation
Recombination
Mutation
Parent selection
Survivor selection
Termination criteria

Fitness function
How to deal with constraints

Summary of Variants of Evolutionary Algorithms

- Representation:
  - Binary strings
  - Integer vectors
  - Floating-point vectors
  - Permutations
  - Matrices
  - Etc
- Crossover for binary, integer and floating point representation
  - 1-Point Crossover
  - Multi-parent diagonal crossover
  - Uniform
- Crossover for floating point representation
  - Intermediate
- Crossover for permutations
  - 1-point crossover for permutations
- Mutation for binary representation
  - Bitwise bit-flipping
- Mutation for integer representation
  - Random reset
  - Creep mutation
- Mutation for floating-point representation
  - Uniform
  - Non-uniform
- Mutation for permutations
  - Swap mutation
- Parents selection:
  - Roulette wheel
  - Tournament selection
- Survival selection:
  - Age-based
    - Generational
  - Fitness-based
    - Delete-worst
    - Elitism

Optimisation Problems May Have Implicit or Explicit Constraints

- Optimisation problem:
  - Maximise / minimise $f(x)$, where x is the design variable
  - Subject to:
    - $g_i(x) \leq 0$, where $i=1,\ldots,n$
      [inequality constraints]
    - $h_j(x) = 0$, where $j=1,\ldots,p$
      [equality constraints]

We will refer to both inequality and equality constraints by $\phi_i(x)$, $i = 1,\ldots,m$, where $m = n + p$ is the total number of constraints.

How to Deal with Constraints?

- Penalty functions.
- Maintaining only feasible solutions based on special representation and genetic operators.

How to Deal with Constraints?

- **Penalty functions** - add a penalty to the fitness of infeasible solutions (or replace the fitness by this penalty).
- Maintaining only feasible solutions based on special representation and genetic operators.

Image from: http://globe-views.com/dcim/dreams/death/death-02.jpg

Death Penalty

Penalty Based on the Level of Infeasibility

$$Q(x) = \begin{cases} 0 & \text{if } x \text{ is feasible} \\ c \cdot \phi(x)^2 & \text{otherwise} \end{cases}$$

Violated constraint (squared)

Very large positive constant

$$Q(x) = \begin{cases} 0 & \text{if } x \text{ is feasible} \\ c \cdot [\phi_1(x)^2 + \phi_2(x)^2 + \ldots + \phi_k(x)^2] & \text{otherwise} \end{cases}$$

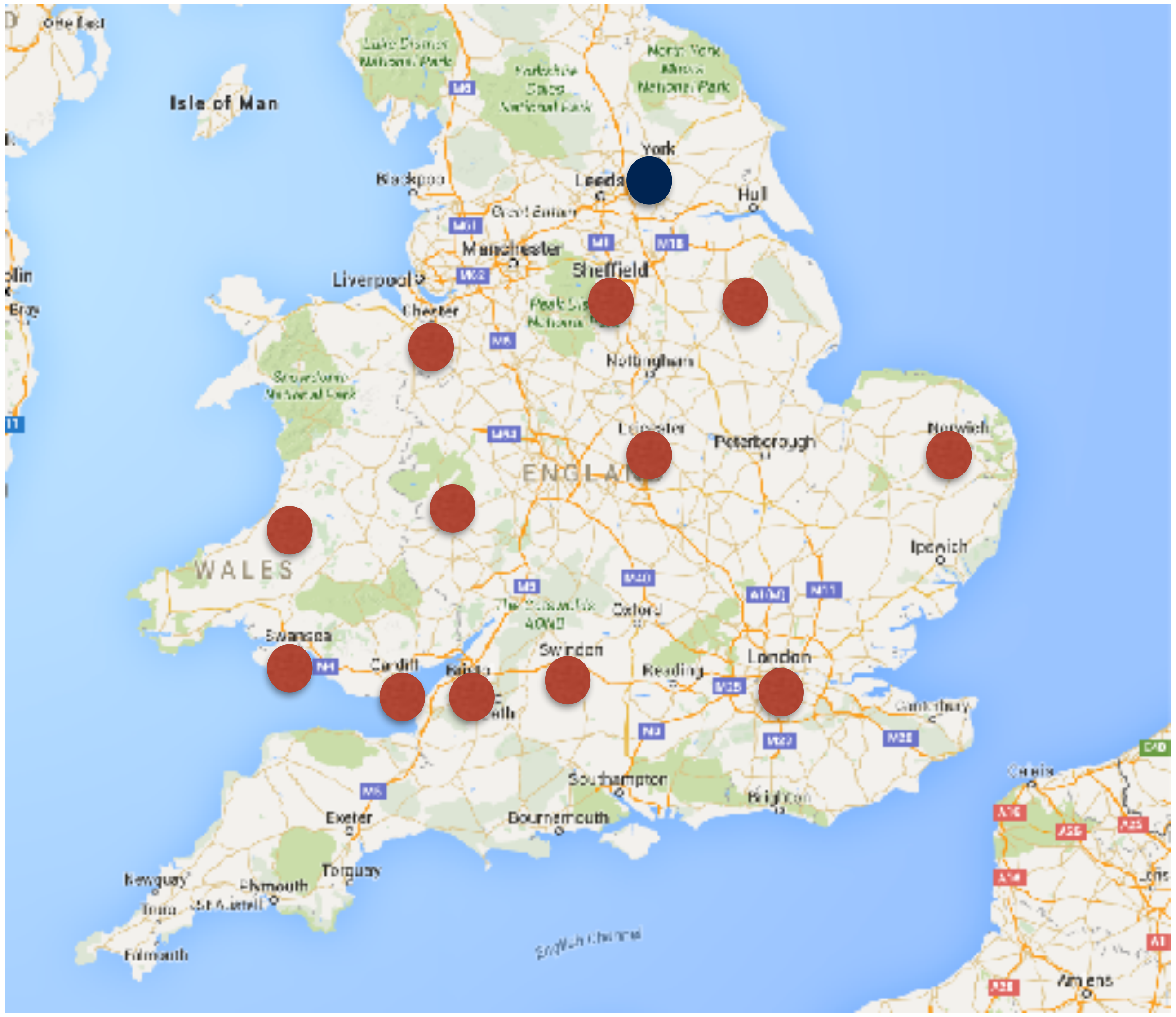
ϕ_i, $1 \leq i \leq k$, are only the violated constraints

How to Deal with Constraints?

- Penalty functions.
- **Maintaining only feasible solutions based on special representation and genetic operators.**

Example: Traveling Salesman Problem

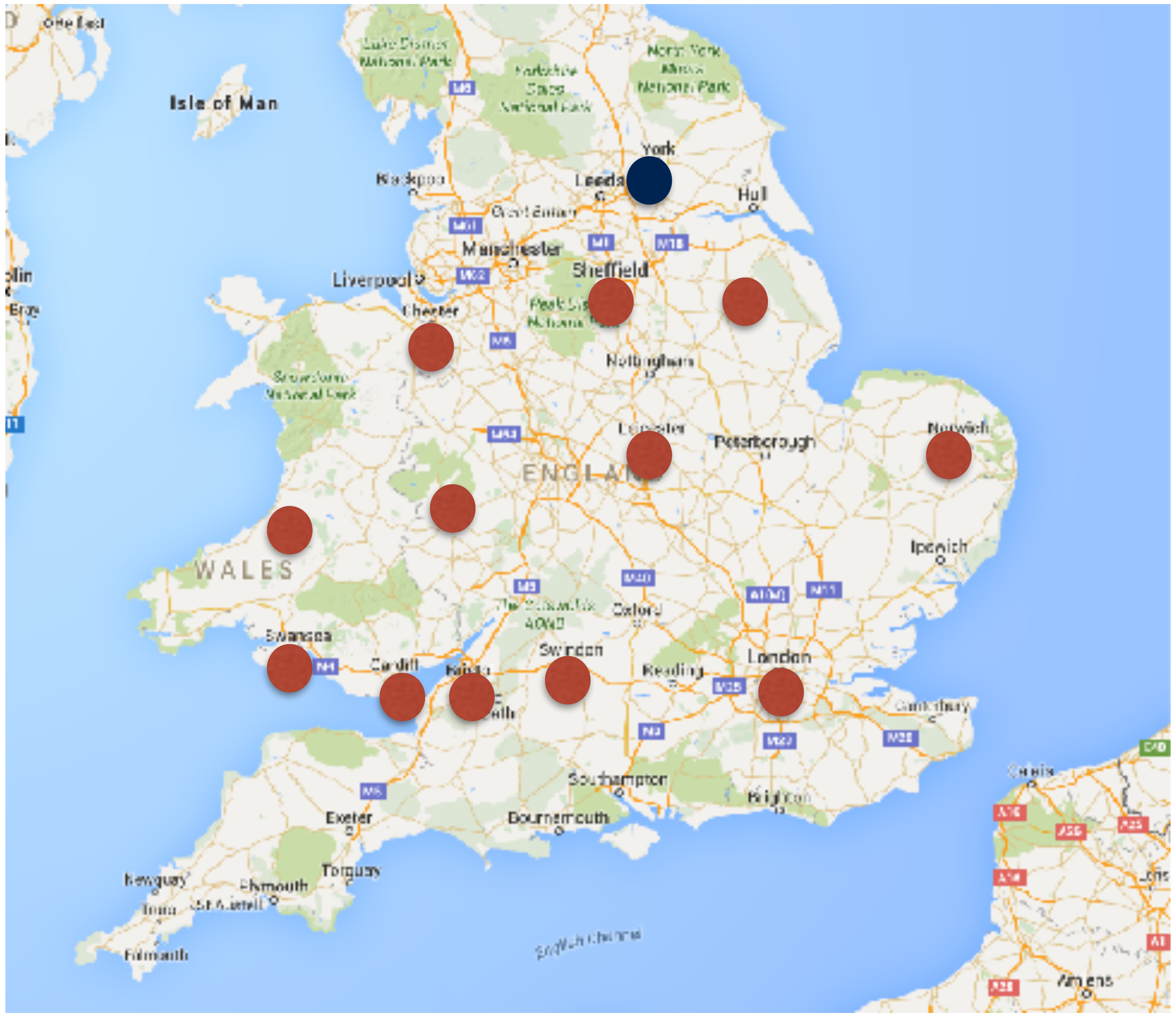
- Traveling Salesman Problem:
  - A salesman must travel passing through *N* cities.
  - Depending on the route the salesman takes, he/she will need to travel longer / shorter distances.



Problem: find a route that minimises traveling distance.

Example: Traveling Salesman Problem

- Traveling Salesman Problem:
  - A salesman must travel passing through *N* cities.
  - Depending on the route the salesman takes, he/she will need to travel longer / shorter distances.



Representation: permutation
Mutation: swap mutation
Crossover: 1-point crossover for permutations

Repair and Decoding

- Repair:
  - Fix the genotype to make it feasible.
  - E.g., remove products from genotype if they exceed the lorry’s maximum supported weight.

- Decoding:
  - Make the solution feasible as you decode it from its genotype to phenotype, without changing the genotype.
  - E.g., normalisation in software project scheduling.

Multi-Objective Optimisation Problems

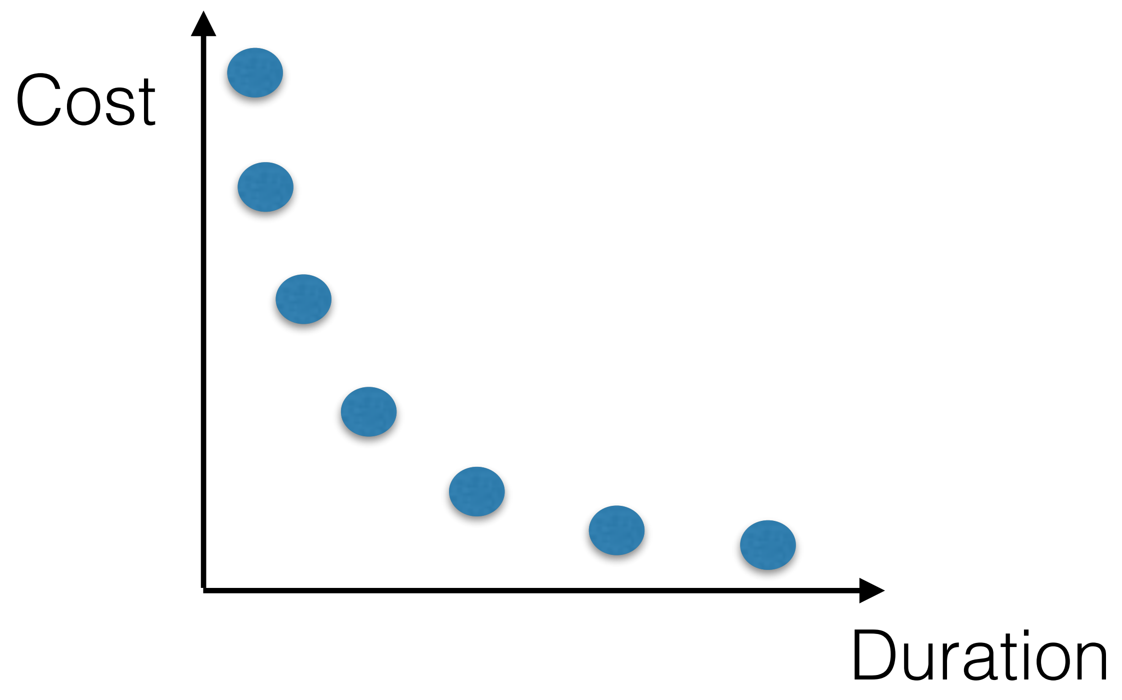
- Real world problems frequently have more than one objective.
- Software project scheduling problem:
    - Cost and duration (to be minimised).

Combining Multiple Objectives into One Objective

- Advantage:
    - Once weights are set, any single-objective optimisation algorithm can be used.
- Disadvantages:
    - Finding suitable weights is not easy before knowing what different trade-offs among objectives may be available.
    - If the different objectives have different scales, weights must reflect not only preferences towards certain objectives, but also treat the different scales.
    - The weights will strongly affect the results.

Multi-Objective Evolutionary Algorithms

- Consider different objectives separately, instead of combining them into a single fitness function.



Overview

- Optimisation problems
- Brute-force search
- Hill-Climbing
- Simulated Annealing
- Evolutionary Algorithms
- Constraint Handling
- Multi-objective Evolutionary Algorithms