2. Evolutionary computing and the simple genetic algorithm

What is evolutionary computing?

- Evolutionary Computing (EC) techniques:
  - are a diverse collection of `nature-inspired' techniques
  - heuristic: not guaranteed to find the best solution
    - the opposite of an `exact' algorithm
  - stochastic: they use random number generators
    - the opposite of deterministic algorithms
  - intended for hard optimization problems (e.g., NP-hard)
- Some examples:
  - Genetic Algorithms (GA)
  - Genetic Programming (GP)
  - Evolutionary Strategies (ES)
  - ...

What is evolutionary computing?

- In contrast, many other heuristic algorithms do not use a population of solutions; e.g.,
  - Hillclimbing
  - Simulated annealing
  - Tabu search
- ...

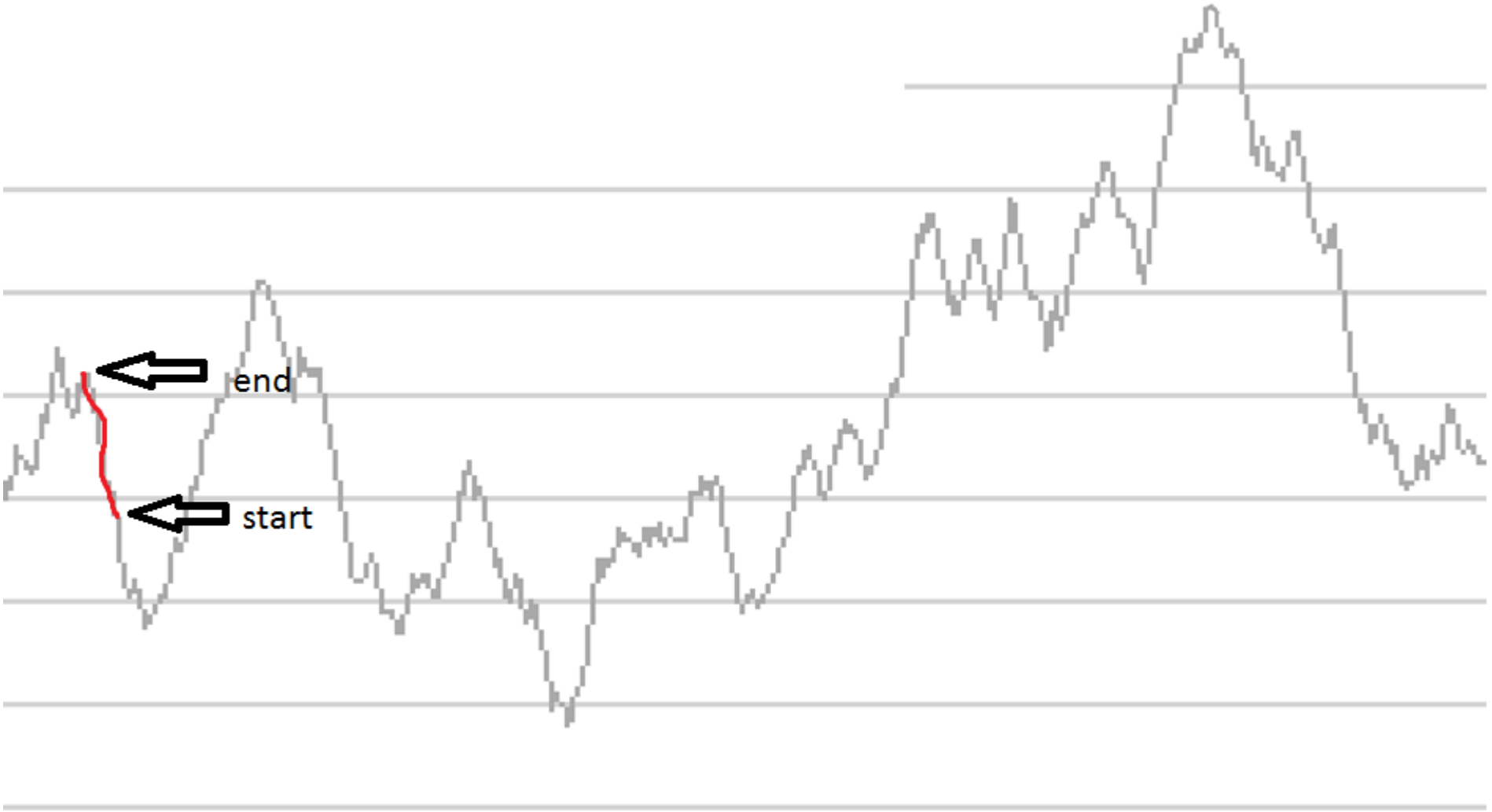
Simple Hillclimbing

New current solution = the first fitter point it finds in the neighbourhood of the current point

```
Start with any point p
repeat
  q = chooseNextPoint(p)
  if p == q then return p as final solution

chooseNextPoint(p):
for each neighbour q of p
   if q is fitter than p return q
return p
```

Simple Hillclimbing



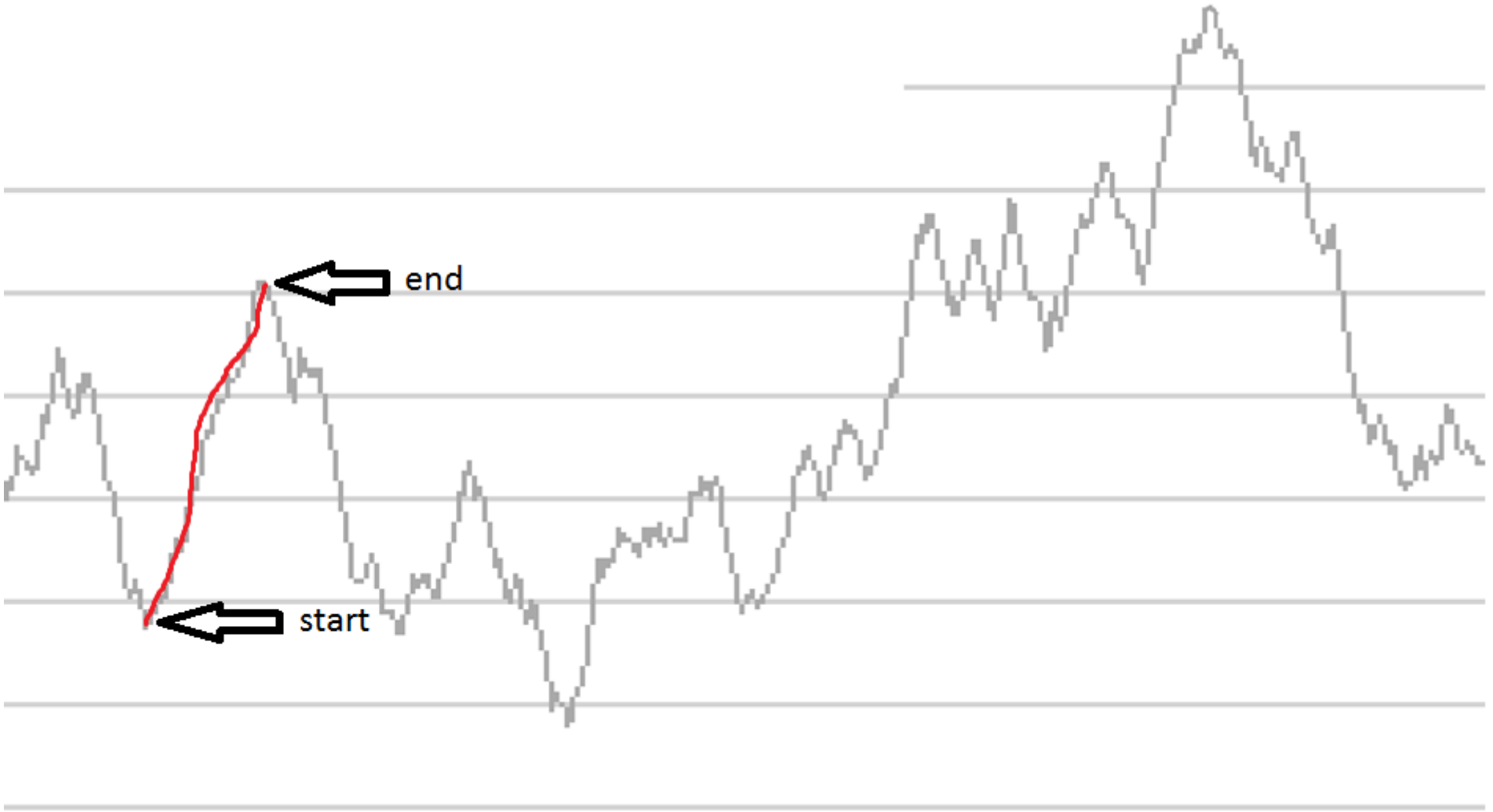
Steepest ascent hillclimbing

Evaluates all neighbours of the current point and selects the fittest of them

```
start with any point p
repeat
  q = chooseNextPoint(p)
  if p == q then return p as final solution

chooseNextPoint(p):
fittest = p
for each neighbour q of p
  if q is fitter than fittest then fittest = q
return fittest
```

Steepest ascent hillclimbing



Simulated annealing

Randomly choose a different solution that is better than current solution, or even worse than current solution if temperature is higher.

Gradually reduce temperature.

https://en.wikipedia.org/wiki/Simulated_annealing

The explore/exploit balance

Hillclimbing **exploits** current knowledge of search space.

Simulated annealing also **explores** search space.

The explore/exploit balance is very important in RL (later).

Disadvantages of EC

- EC is heuristic
  - not guaranteed to find best possible solution
- EC is stochastic
  - can produce a different solution every time we run it
  - we often want to run it repeatedly in case the next run returns a better solution
    - how many times should we run it?
- Evolutionary algorithms are complex systems
  - complex or chaotic behaviour can emerge from simple interactions between simple parts
  - very hard to predict how overall system will behave
  - one approach is to run algorithm many times and see what happens
    - but that can be slow, especially if you can change many parameters
- Given these disadvantages, why use them?

Computational compromises

- Instead of EC, we can use a simple algorithm which is guaranteed to find the best possible solution:

```
bestSolution = null;
for each solution s in possibleSolutions do
    if isBetter(s, bestSolution) then bestSolution = s;
end
```

- If this is fast enough for your problem, use it!
  - But for many problems, it scales too slowly
- For many computer science problems there is a simple, exact algorithm ... which is too slow to use
- So instead we either:
  1. Accept an approximate (suboptimal) solution
  2. Or find an exact solution to an approximation of the original problem
- EC takes the first approach

Alternatives to EC

- Many other optimization methods exist
  - Some are very efficient
  - Some are also exact, or at least guaranteed to improve the longer you run them
  - But they may require information about the problem that you do not have
    - We'll return to this when we cover Dynamic Programming
  - Or require conditions that do not hold
    - E.g., that a function be differentiable

Alternatives to EC

- Nonetheless, there are *many* useful optimization methods, which we will not cover; e.g.,
  - deterministic search algorithms (e.g., depth-first search)
  - linear programming
  - integer programming
  - constraint satisfaction
  - optimal control
  - ... and many more. See mathematical optimization in wikipedia for an overview
- Always use the right tool for the job

Darwin, Mendel, and the modern synthesis

EC is inspired by `neo-Darwinian' evolutionary theory

- 1859: publication of *Origin of Species* by Charles Darwin
  - Explained evolution as due to natural selection acting on heritable variation
  - Very similar ideas proposed by A. R. Wallace at about the same time
- 1865 - Gregor Mendel presented his work on *Experiments in Plant Hybridisation*
  - He demonstrated the particulate nature of inheritance (i.e., genes are discrete particles)
  - Darwin had assumed genes were blended between parents (like averaging two numbers)
- These two achievements together paved the way for the `Modern Synthesis'...

The modern synthesis (neo-Darwinism)

- One problem that had worried Darwin was `regression' of traits
  - Darwin assumed blending of heritable material
  - Hence the `value' of any trait would tend to converge in a population as evolution progressed
  - The resulting lack of variation in the population would give natural selection nothing to act on, so evolution would stall
- Mendel's results went unnoticed by evolutionists for about 30 years

The modern synthesis (neo-Darwinism)

- Once rediscovered, Mendel's laws of particulate (i.e., genetic) inheritance were incorporated into Darwinian evolutionary theory
- The result was the `modern synthetic theory of evolution', or `neo-Darwinism'
  - Natural selection acting on genetic variation in populations
  - Primarily mathematical in nature, modelling gene spread in populations
    - Population genetics
    - Quantitative genetics
- Subsequently, the physical basis of particulate inheritance was found
  - Discovery of DNA by James Watson and Francis Crick, 1962 Nobel Laureates in Medicine

The emergence of evolutionary computing

- Evolutionary ideas were being applied in optimization as early as the mid 20th century; e.g.,
  - Box (1957) Evolutionary operation: a method for increasing industrial productivity. Applied Statistics 6, 81-101
  - Bremermann (1962) Optimization through evolution and recombination. In: Self-Organizing Systems.
- 1970s: Genetic Algorithms became a prominent research area
  - John Holland presented a mathematical definition of GAs, and theory explaining their performance
    - Holland (1975) Adaptation in Natural and Artificial Systems
  - Holland was mainly interested in adaptive systems, but his student Ken De Jong catalysed research on GAs for optimization, formulating a test suite of problems
    - De Jong (1975) An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan

The simple genetic algorithm

- What is the 'Simple Genetic Algorithm' ?
- Genetic algorithms have many variations on a central theme
  - Choice of selection operator
  - Choice of genetic operators
  - ...
- The SGA is the canonical description of what a GA is
  - Here we will describe the SGA according to Holland's original proposals
- As an aside, the term “SGA" also refers to Michael Vose's mathematical description of a GA
  - Vose (1999) The Simple Genetic Algorithm: Foundation and Theory

Terminology

- *Chromosome* - a solution to the problem the GA is solving, encoded as a fixed length string in some finite alphabet
- *Gene* - a position on the chromosome which can take a particular value
- *Locus* - see gene
- *Allele* - a value that a gene can take (e.g., 0 or 1 in binary chromosomes)
- *Population* - the set of chromosomes the GA is acting on
- *Fitness* - the objective value of the solution a chromosome encodes for a problem
- *Fitness/objective function* - the function that returns the objective value of a given solution

Terminology

- *Selection* - the mechanism for choosing chromosomes to reproduce, based on their fitness
- *Parent* - a chromosome selected for reproduction
- *Offspring* - the chromosome(s) resulting from reproduction of one or more parents
- *Crossover* - the process of recombining alleles from multiple parents during reproduction to produce offspring
- *Mutation* - the process of randomly replacing offsprings' alleles with randomly chosen alternatives during reproduction

The simple GA

- The Simple GA

  generate initial population;
  while *termination criterion unsatisfied* do
    evaluate population;
    select parents;
    reproduce using genetic operators;
    replace population with offspring;
  end

- Genetic operators
  - Single-point crossover - recombines alleles from two parents into offspring
  - Point mutation - randomly changes alleles in offspring at a very low rate

Selection

- A key idea of the GA is that parents should be selected based on their fitness
- Simplest way of doing this is using roulette wheel selection
  - Each chromosome *i* in population *P* is given a probability of selection p_i based on its fitness f_i as fraction of total population fitness:

$$p_i = \frac{f_i}{\sum_{j \in P} f_j}$$

- So for a given population we construct a single-pointer `roulette wheel' where each chromosome's proportion of the wheel is given by the equation above
- To select a parent for reproduction we uniformly sample from [0, 1)
  - i.e., we spin the roulette wheel once and see which chromosome its pointer indicates

Selection

- Having selected a parent, we then reproduce that parent according to our operators, parameters, etc.
  - According to our parameters, we may apply crossover, and/or mutation, and/or any other operators
  - Often crossover rate (probability of applying crossover during reproduction) is denoted by χ
  - Similarly mutation rate (per gene probability of mutation during reproduction) by μ
- Usually population size is held constant...
- ...we continue selecting and reproducing until we have enough offspring to replace the current generation
  - Hence such a GA is called a `generational' GA

Representation

- The SGA uses binary strings
  - The simplest representation
  - Widely used
  - Have some nice theoretical properties
- Other algorithms also use:
  - higher cardinality alphabets
  - more complex data structures: trees, graphs

Single-point crossover

- Holland's initial proposal for genetic recombination in GAs was single-point crossover, sometimes denoted 1X
- For a pair of parent chromosomes of length l, we (uniformly) choose at random an integer from {1, 2, …, l-1}
- We then exchange portions of each chromosome, combining the substring left of the cutpoint from one chromosome with the substring right of the cutpoint from the other chromosome, and vice versa
  - e.g., the parent chromosomes 11111 and 00000 with the randomly selected crossover point at position 3...
  - ...yield the offspring chromosomes 11100 and 00011
- We now have two offspring chromosomes, each containing alleles from both parent chromosomes
- Optionally we can discard one of the offspring if we only require one offspring

Point mutation

- During reproduction, mutation occurs at a (usually low) rate, in addition to crossover
- Normally mutation rate is expressed as a per-gene probability
  - Called “point mutation" as we mutate 1 point (gene) at a time
- If a gene is mutated, its current allele is exchanged for a different randomly selected allele
  - If our chromosomes are binary strings, there is only one other possible allele and the gene is simply bit-flipped
- Typical mutation rate about 0.01 probability of mutation per gene `transcribed' (copied) from parent to offspring

A simple example - 1-max

- Let's work through a simple example of the SGA solving a problem
  - Problem: 1-Max (binary fixed-length chromosome; fitness is number of ones in chromosome)
    - 1-Max is not a good problem to apply a GA to as the fitness function is linearly separable (e.g., hill-climbing will do much better), but it is often used for illustration
  - Chromosome length (l) = 5
  - Population size (N) = 4
  - Crossover rate (1X) (χ) = 1.0
  - Mutation rate (μ) = 0.25 (which is unrealistically high for most problems)