Global Optimisation Methods

Pablo Brito-Parada
p.brito-parada@imperial.ac.uk

ACSE7 L12
2021

Optimisation Problems

A generic maximisation (or minimisation) problem involves having an objective function that is dependent on a number of input variables for which a maximum (or minimum) value is required

Optimisation Algorithms

Optimisation algorithms can be classified in different ways

i. Gradient-based or Gradient-free

ii. Deterministic or Stochastic

iii. Trajectory-based or Population-based

iv. Local or Global

The classification is not always straightforward as hybrid algorithms are not uncommon

Gradient-based methods

- For objective functions that are continuous functions of the input variables, gradient search methods are often employed
  - E.g. conjugate gradient methods
- For problems where the input variables are either discrete and/or the objective function is discontinuous, gradient search methods are not appropriate and gradient-free methods (suitable for global optimisation) can be used instead

Global methods

- A global optimisation algorithm is intended to locate a global optima. It explores the entire input search space and aims at getting close to (or finding exactly) the extrema of the function.
- Global optima and local optima – the difficulty associated to global optimisation
- Iterative generation of a candidate solution or a population of candidate solutions
- High computational cost

Global methods

These methods can be classified into ***exact methods*** and ***heuristics*****.**

An exact (or deterministic) method guarantees to find and verify global solutions. Otherwise, it is called a heuristic (or meta-heuristic).

Heuristics try to find global solutions without verifying global optimality – they involve a set of rules

They can be used as stand-alone solvers or as an acceleration tool in deterministic methods.

Meta-heuristics

- Recent literature tends to refer to algorithms with stochastic components, as well as to all modern nature-inspired algorithms, as meta-heuristic

- *Heuristic* → *finding/discovering by trial and error*

- Here *meta-* means *beyond* or *higher level*

- A trade-off of randomization and local search is used

Meta-heuristics

- A top-level strategy that guides and modifies an underlying or subordinate heuristic
- Iterative generation process
- Combines **exploration** and **exploitation** of the search spaces
- Uses learning strategies, a guiding process and an application process, to structure information so that near-optimal solutions can be found efficiently

Meta-heuristics

- Exploration / Diversification
  - Generate diverse solutions via randomisation to better explore the search space

- Exploitation / Intensification
  - If a current good solution is found in a local region, the region becomes the focus of the search

A balance of the two is essential and determines convergence rates

Global methods – Meta-heuristics

Examples of heuristic search strategies:

- **Simulating annealing**
- **Memetic algorithms**
- **Particle swarm optimisation**
- **Genetic algorithms**

An ever increasing number of “nature inspired” algorithms (ant colony, artificial bee colony, firefly, cuckoo search)

Simulated annealing

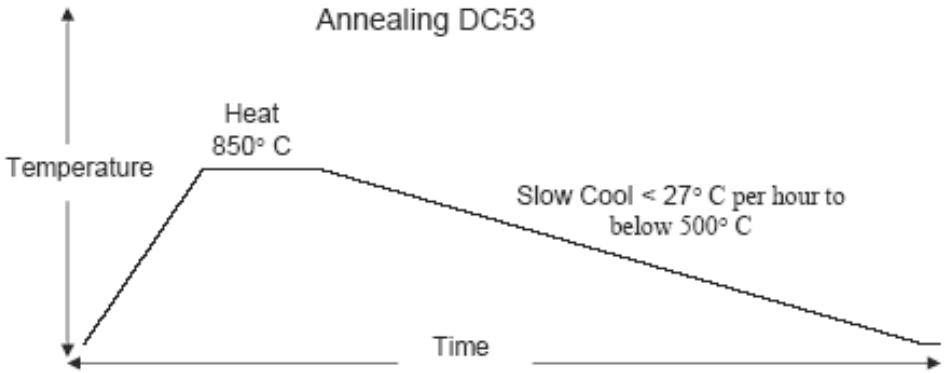
Simulated annealing

Annealing is a metallurgical process in which a material is heated above its recrystallization temperature for a certain amount of time, and then is slowly cooled.

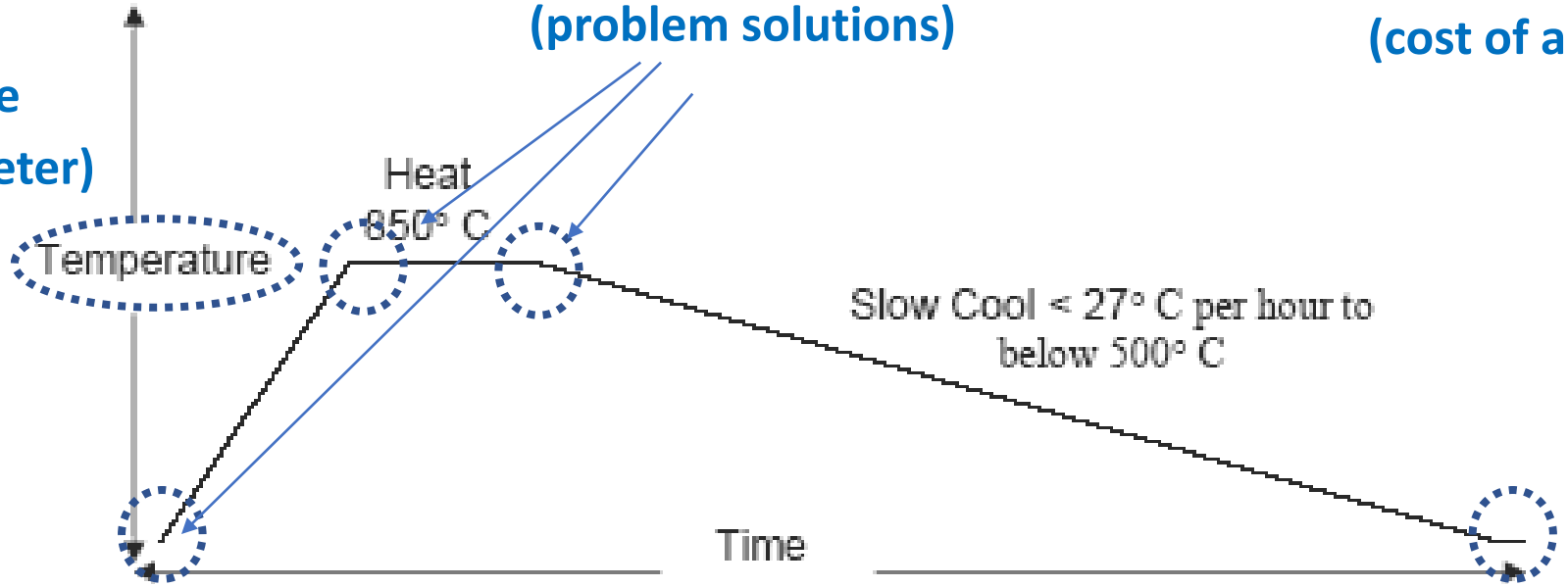
The slow cooling brings the metal to a crystalline state, altering their physical/chemical properties to increase its ductility and reduce its hardness.

In *simulated annealing* points of a sample set are modified by applying a descent step with a random search direction and an initially large step-size that is gradually decreased during the optimisation process.

Typically used in discrete, but very large, configuration spaces.



Simulated annealing analogy



Physical material states
(problem solutions)

Energy of state
(cost of a solution)

Temperature
(control parameter)

Defect-free crystal
(global minimum energy configuration)

Simulated annealing

At each iteration, a new point is randomly generated. The extent of the search is based on a probability distribution (with a scale proportional to the “temperature”).

Importantly, the algorithm accepts new points that minimise the objective function but also, with a certain probability, points that do not, in order to avoid being trapped in local minima.

An *annealing schedule* is selected to systematically decrease the temperature as the algorithm proceeds. As the temperature decreases, the algorithm reduces the extent of its search to converge to a minimum.

Simulated annealing

$$P_\alpha = \frac{1}{Z} e^{\frac{-E_\alpha}{k_B} T}$$

P_α Probability of a physical system being in state α

E_α Energy in state α

T Temperature

k_B Boltzmann's constant

Z Partition function

At high T , the Boltzmann distribution exhibits uniform preference for all the states, regardless of the energy.

When T approaches zero, only the states with minimum energy have nonzero probability of occurrence.

Simulated annealing

• **At high *T***	The system ignores small changes in the energy and approaches thermal equilibrium rapidly	It performs a coarse search of the space of global states and finds a good minimum
• **As *T* is lowered**	The system responds to small changes in the energy	It performs a fine search in the neighbourhood of the already determined minimum and finds a better minimum
• **At *T* = 0**	The system must reach equilibrium	The best minimum is accepted as the global minimum

Simulated annealing

- Theoretically, a high probability to reach a global minimum
- *T* is used as a control parameter, the *computational temperature*. It controls the magnitude of the perturbations of the energy function $E(\boldsymbol{x})$.
- The probability of a state change is determined by the Boltzmann distribution of the energy difference of the two states:

$$P = e^{-\frac{\Delta E}{T}}$$

- The probability of uphill moves in the energy function ($\Delta E > 0$) is large at high T , and is low at low T .

Simulated annealing algorithm

1. Initialize the system configuration.
   Randomize $\boldsymbol{x}(0)$.
2. Initialize T with a large value.
3. **Repeat**:
   a. **Repeat**:
      i. Apply random perturbations to the state $\boldsymbol{x} = \boldsymbol{x} + \Delta\boldsymbol{x}$.
      ii. Evaluate $\Delta E(\boldsymbol{x}) = E(\boldsymbol{x} + \Delta\boldsymbol{x}) - E(\boldsymbol{x})$:
         if $\Delta E(\boldsymbol{x}) < 0$, keep the new state;
         otherwise, accept the new state with probability $P = e^{-\Delta E/T}$.
      until the number of accepted transitions is below a threshold level.
   b. Set $T = T - \Delta T$.

until T is small enough.

Note that a simulated annealing algorithm can result in different local minima

→ a problem needs to be run several times before the solution can be accepted as the global optimum.

Simulated annealing

- The cooling schedule is critical in terms of efficiency
  - If *T* is decreased too rapidly – potential of converging to local minimum
  - If *T* is decreased too slowly – very slow convergence
- Two common ways to control the cooling rate are linear and geometric annealing schedules
- Some studies have shown that a logarithmic decrease almost ensure convergence to a the global optimum:

$$T(t) \geq \frac{T_0}{\ln(1+t)}, \qquad t = 1, 2, \ldots$$

Memetic algorithms

Memetic algorithms

These are evolutionary algorithms, based on the evolution of a concept.

Memes, building blocks of meaningful information that is transmissible and replicable, are the social analogue of genes and can be thought of as schemata that is modified and passed on over a learning process.

The typical memetic algorithm uses an additional mechanism to modify schemata and that refinement can be passed on to an individual's offspring.

In the memetic computation paradigm, the basic mechanisms of evolution are augmented with domain-knowledge expressed as computationally encoded memes

Memetic algorithms

The notion of memes is limited to mathematical procedures or heuristics that serve as local search schemes which are subsequently hybridized with some population-based stochastic global optimiser.

It combines the evolutionary adaptation of a population with individual learning of its members.

This strategy can be seen as a balance between exploration and exploitation of the search space.

Success of memetic algorithms relies strongly on careful manual adjustment of the local search procedures

Memetic algorithms

The memetic evolutionary process is primarily driven by imitation.

Imitation occurs during transmission of memes. Individuals make choices and imitate others who have obtained high payoffs in previous rounds.

Memetic selection – *whom* to imitate

Memetic transmission –*how* to imitate

Memetic variation –*wha*t is imitated or assimilated.

Genetic Algorithms

Genetic Algorithms

- Based on the principles derived from the dynamics of natural population genetics.
- Most popular evolutionary algorithms.
- Population-based and the basis for many modern evolutionary algorithms.
- Applications range from graph colouring to pattern recognition, from discrete systems (such as the travelling salesman problem) to continuous systems (e.g., the efficient design of airfoil in aerospace engineering), and from financial markets to multiobjective engineering optimization.

Genetic Algorithms

A popular approach to problems with discrete variables

- Note that the method can be used with continuous variables (either some or all of the variables continuous)
- If all input variables and the objective function are continuous then gradient search methods are likely to perform better

- Encoding of solutions as arrays of bits or character strings (chromosomes)
- Manipulation of strings by genetic operators and a selection based on their fitness to find a solution to a given problem.

Genetic Algorithms

Instead of proposing a single solution to an optimisation problem, a genetic algorithm generates many possible solutions that form a population.

Then solutions are scored using a fitness function (objective function) to decide which solutions are better than others.

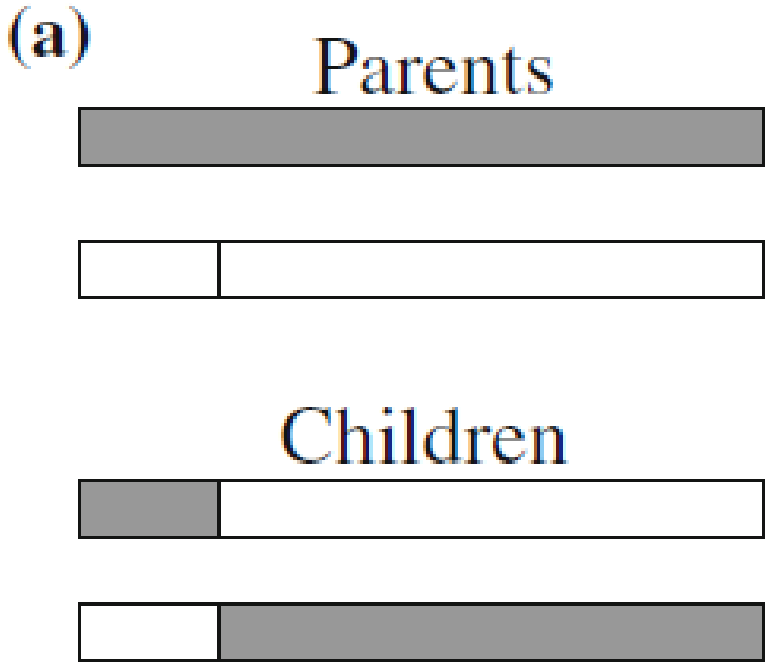
These candidates solutions are recombined so that the best solutions reproduce to create a new generation of solutions with the best traits of the previous solutions.

This continuous until improvement stops or the max number of generations is reached.

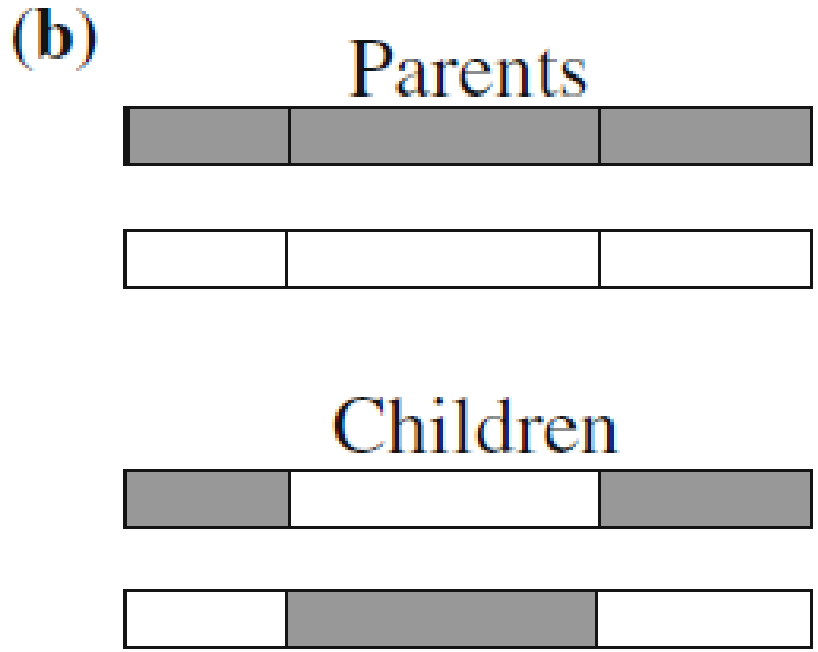
Genetic Algorithms – Producing Children

- The heart of a Genetic Algorithm is the production of “child” vectors from a list of “parent” vectors
- Done using two operations:
  - ***Crossover***
    - This is roughly equivalent to sexual reproduction.
    - A portion of one parent vector is swapped with a portion of another parent vector to produce two child vectors
    - Motivation for swapping a portion of a parent vector with another rather than swapping individual values randomly is that, over successive generations, values that work well together will end up next to one another in the vector (roughly equivalent to genes)
  - ***Mutations***
    - Random changes in the numbers in the vector.

Genetic Algorithms – Crossover

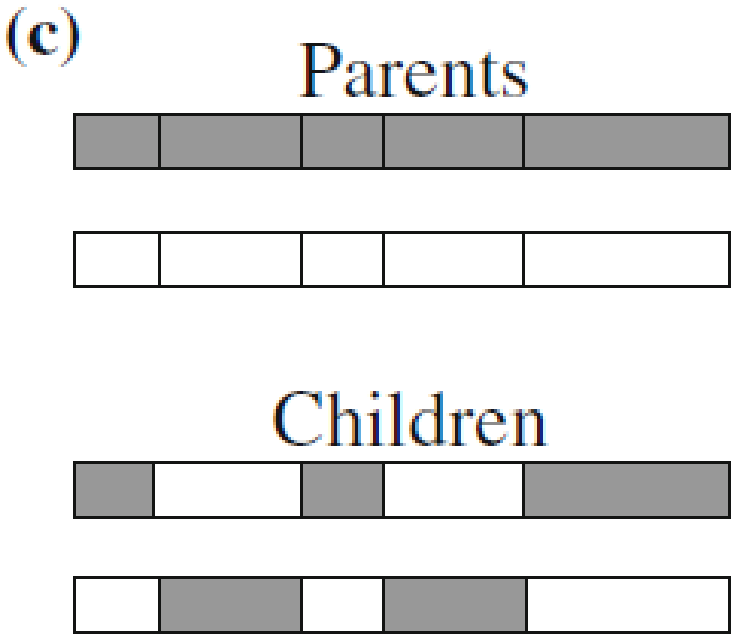


One-point crossover

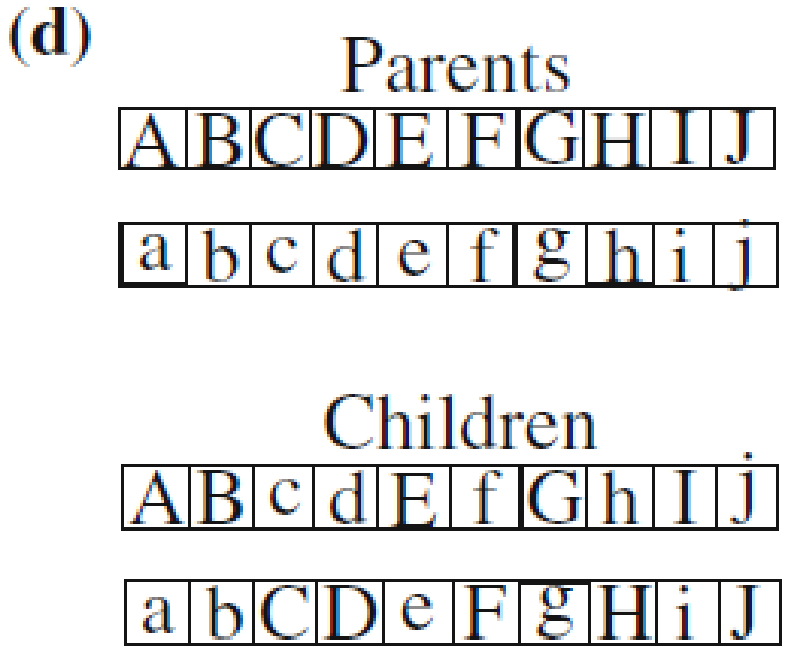


Two-point crossover

Genetic Algorithms – Crossover

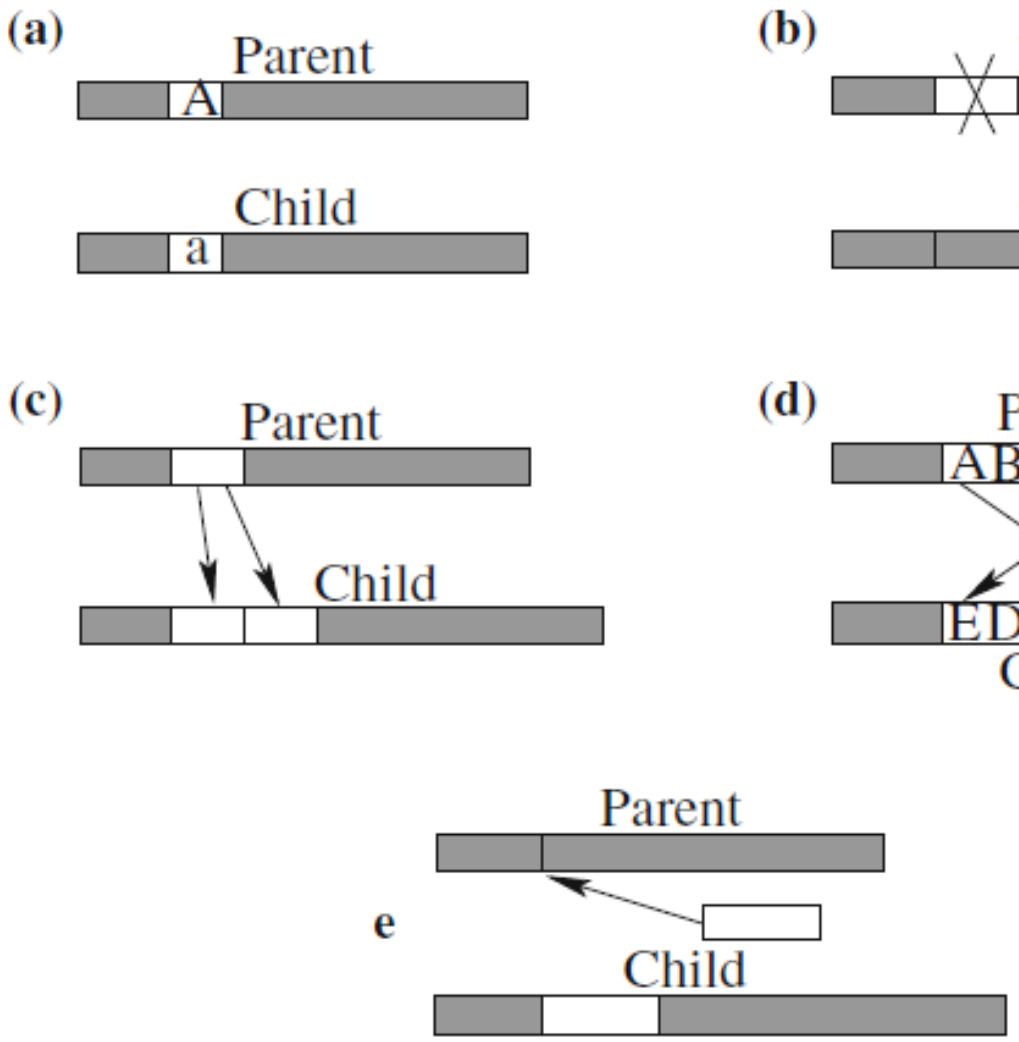


Multipoint crossover



Uniform crossover

Genetic Algorithms – Mutation



- **a Substitution**
- **b Deletion**
- **c Duplication**
- **d Inversion**
- **e Insertion**

Genetic Algorithms – Selection of fittest

- Selection of the fittest is carried out according to the fitness of the chromosomes.
- To ensure the best chromosomes remain in the population, they are transferred to the next generation without much change (called elitism).
- A proper criterion for selection is very important. It determines how chromosomes with higher fitness are preserved and transferred to the next generation (with a degree of elitism).

- The basic form is to select the best chromosome (in each generation) which will be carried over to the new generation without being modified by the genetic operators. This ensures that a good solution is attained more quickly.

Genetic Algorithms – general procedure

1) Definition of an encoding scheme
2) Definition of a fitness function or selection criterion
3) Creation of a population of chromosomes
4) Evaluation of the fitness of every chromosome in the population
5) Creation of a new population (children) by performing fitness-proportionate crossover and mutation (also possibly selection and recombination)
6) Replacement of the old population by the new one.

Steps 4), 5) and 6) are then repeated for a number of generations. At the end, the best chromosome is decoded to obtain a solution to the problem.

Particle swarm

Particle swarm optimisation

It uses a simple mechanism that mimics swarm behaviour (eg birds flocking, fish schooling) to guide the particles to search for global optimal solutions.

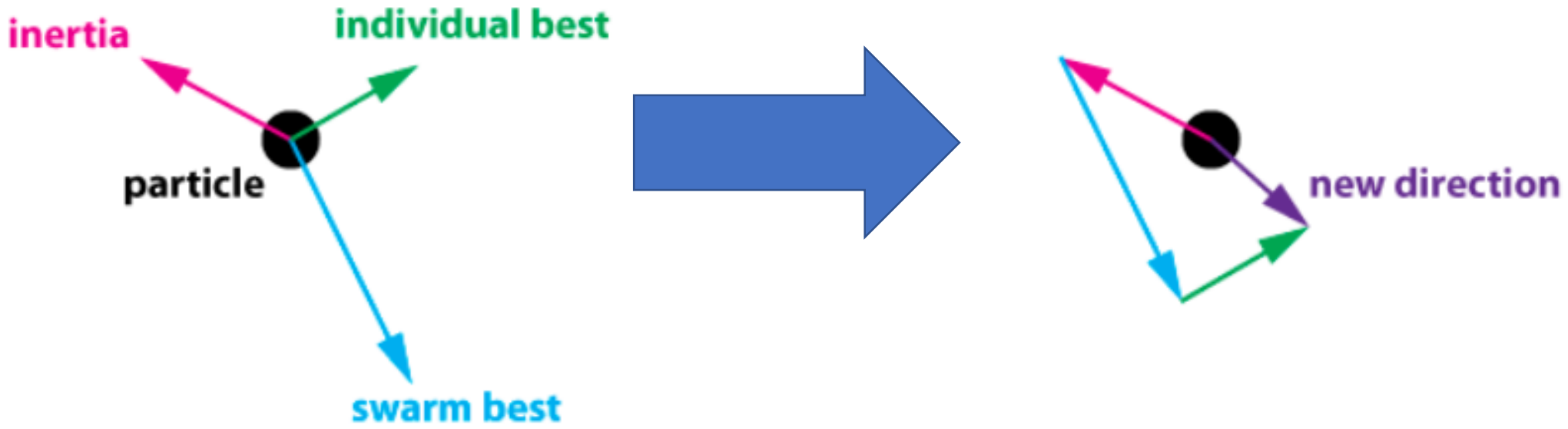
- Useful for the optimisation of irregular problems that are noisy and change over time.
- It evolves populations or swarms of individuals → particles.

It finds the global best solution by adjusting the moving vector of each particle according to its personal best and the global best positions of particles in the entire swarm at each iteration.

Relevant parameters:

- number of particles
- position of agent in the solution space
- velocity of agents
- neighbourhood of agents

Particle swarm optimisation



https://nathanrooy.github.io/posts/2016-08-17/simple-particle-swarm-optimization-with-python

Particle swarm optimisation

Basic algorithm:

1. Set $t = 1$.
   Initialise each of the N_P particles in the population by randomly selecting values for its position x_i and velocity $\boldsymbol{v}_i$, $i = 1, \ldots, N_P$.
2. **Repeat**:
   a. Calculate the fitness value of each particle i .
      If the fitness value for each particle i is greater than its best fitness value found so far, $\boldsymbol{x}^*_i$ then revise $\boldsymbol{x}^*_i\,(t)$.
   b. Determine the location of the particle with the highest fitness and revise the global best, $\boldsymbol{x}^g(t)$, if necessary.
   c. **for each** particle i, calculate its velocity
   d. Update the location of each particle i
   e. Set $t = t + 1$.
   until stopping criteria are met.

$$\begin{aligned} \boldsymbol{v}_i(t+1) &= \boldsymbol{v}_i(t) + cr_1\left[\boldsymbol{x}_i^*(t) - \boldsymbol{x}_i(t)\right] + cr_2\left[\boldsymbol{x}^g(t) - \boldsymbol{x}_i(t)\right], \\ \boldsymbol{x}_i(t+1) &= \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1), \quad i = 1, \ldots, N_P, \end{aligned}$$

Particle swarm optimisation

Computationally inexpensive in terms of both memory requirements and speed.

It can locate the region of the optimum fast, but once in this region the progress is slowly (due to the fixed velocity step size), which is an aspect that different variants of the algorithm try to address.

Final remarks

Global optimisation / gradient-free

- Easy to implement, in that they don’t require derivatives!
- Slow for large problems
- No guarantee of optimal solution
- Several versions available
- A large amount of research

Global Optimisation Methods

Pablo Brito-Parada
p.brito-parada@imperial.ac.uk

ACSE7 L12
2021