Lecture #3: Metaheuristics

COMP3608 - Intelligent Systems

Inzamam Rahaman

Last Lecture

- Last lecture we looked at gradient descent for solving optimisation problems

- … and backtracking for solving CSPs

- This lecture, we look at nature-inspired methods for solving such problems

- In particular, we shall look at 0th order methods that do not require the gradient

The Problem about problems

- Many optimisation problems are very difficult to solve!
- Sometimes, we have to acknowledge that we can’t find optimal design point(s) to certain problems.
- Nevertheless, we can still try to get something passable or good, even if its not the best
  - Something is better than nothing

Heuristics

- Recall that optimisation problems are regarded as search problems
- Have a space of potential solutions
- Want to find design point that gives us optimal

Heuristics

- Sometimes, our problem as a structure that allow us to make good guesses about where good regions or good design points are located in the search space

- A rule of thumb for a very specific type of problem is called a heuristic

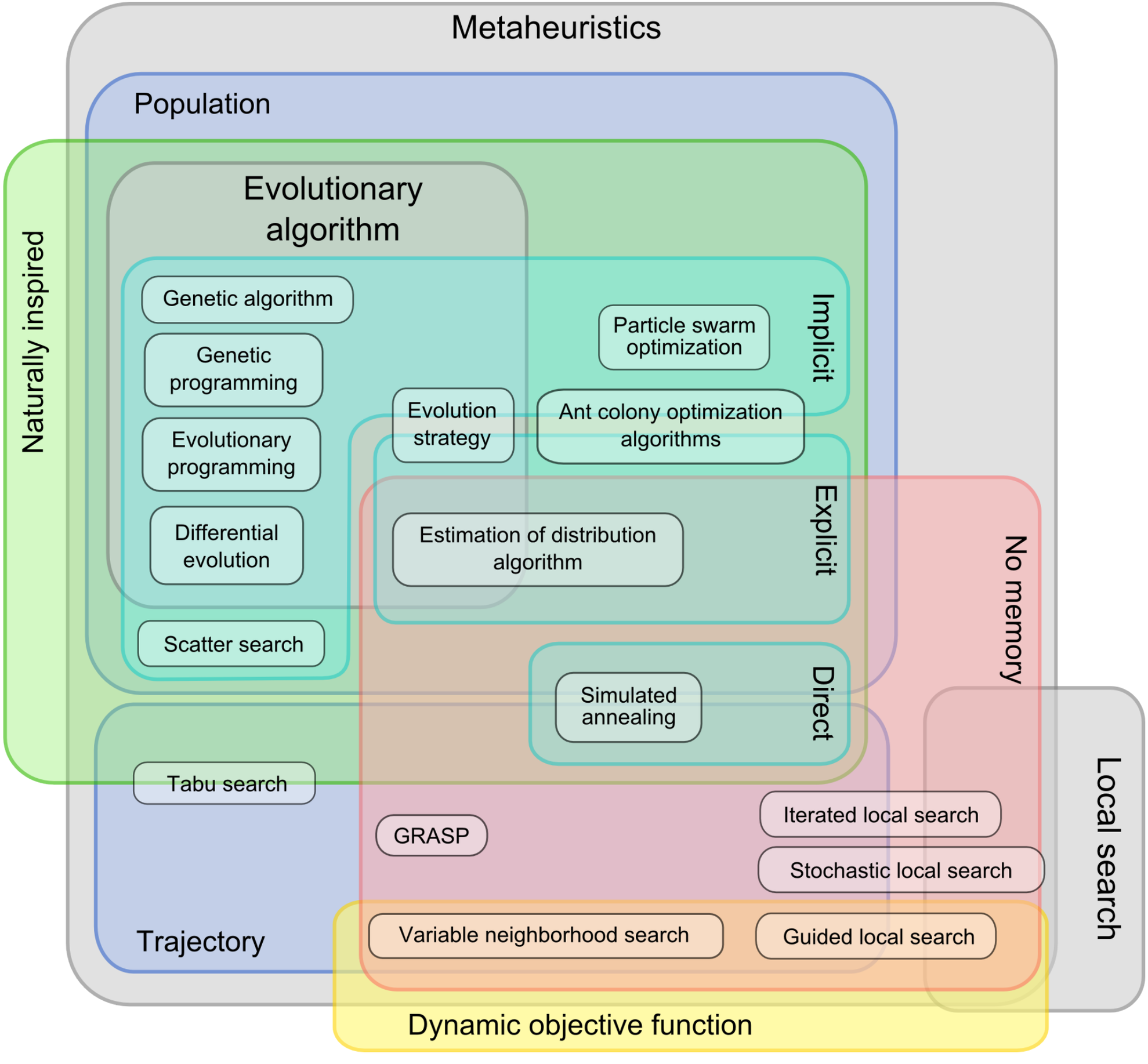
- Example: When we have different servers at different costs for different tasks, we have heuristics that we can use to assign servers to tasks

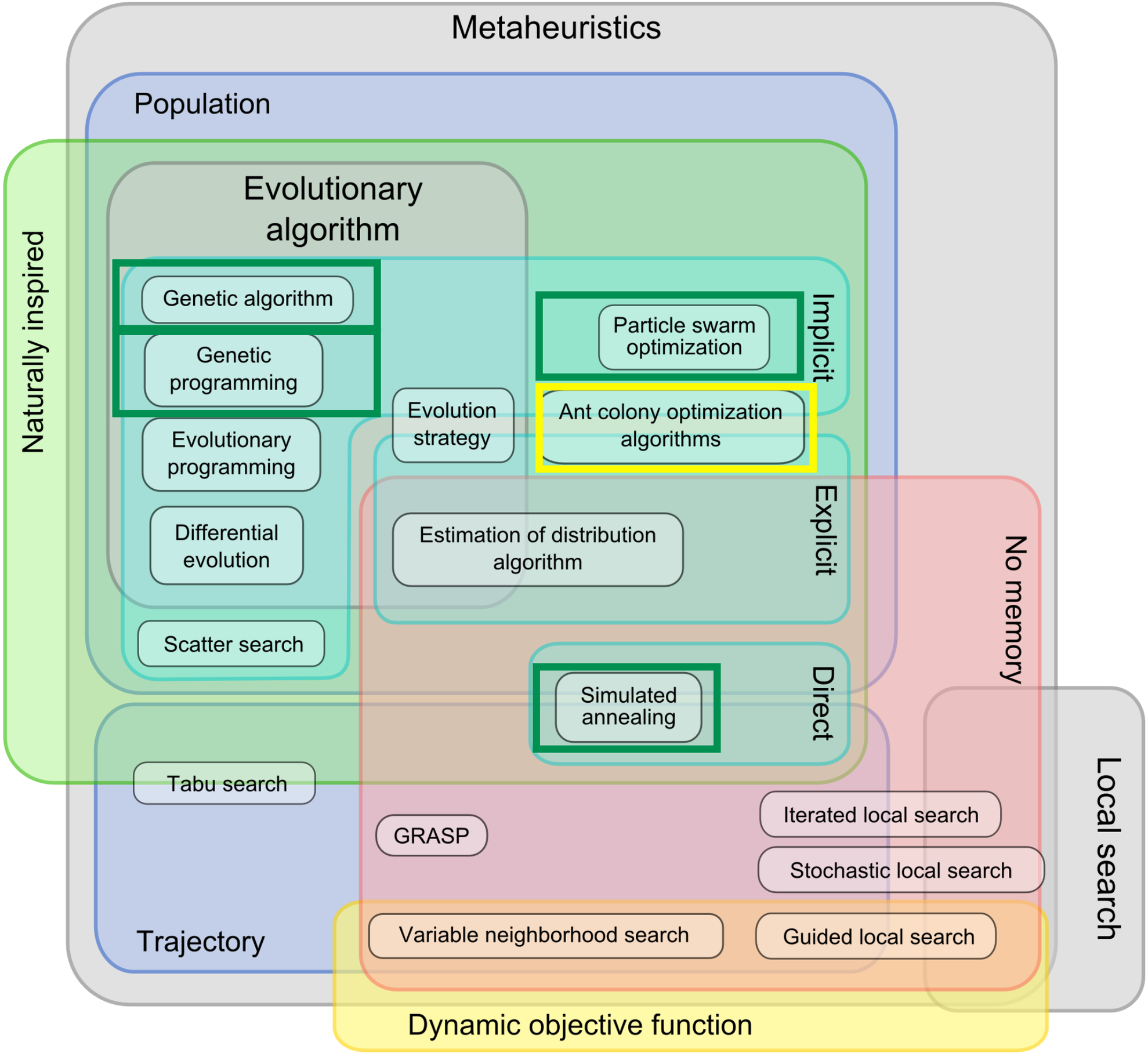
The problem with heuristics

- Heuristics, however, have problems
- For one, there are very problem and instance specific
  - Much effort must be applied to derive them
  - Much effort must be applied in their engineering
- Remember we like algorithms to be as general as possible!

Metaheuristics to the rescue!

- Metaheuristics are like heuristics in that they offer good, efficient strategies but often no guarantees of optimality
- But they differ in that they are more generally applicable
  - Rely on more generic ideas to guide search through space of potential solutions
  - No magic bullet, but very helpful
  - Many different types and general ideas
    - Exists an interesting taxonomy over metaheuristics


Metaheuristics
Population
Naturally inspired
Evolutionary algorithm
Genetic algorithm
Genetic programming
Evolutionary programming
Differential evolution
Scatter search
Implicit
Particle swarm optimization
Evolution strategy
Ant colony optimization algorithms
Explicit
Estimation of distribution algorithm
Direct
Simulated annealing
No memory
Local search
Tabu search
GRASP
Iterated local search
Stochastic local search
Trajectory
Variable neighborhood search
Guided local search
Dynamic objective function


Metaheuristics
Population
Naturally inspired
Evolutionary algorithm
Genetic algorithm
Genetic programming
Evolutionary programming
Differential evolution
Scatter search
Evolution strategy
Implicit
Particle swarm optimization
Ant colony optimization algorithms
Explicit
Estimation of distribution algorithm
Direct
Simulated annealing
No memory
Tabu search
GRASP
Iterated local search
Stochastic local search
Trajectory
Variable neighborhood search
Guided local search
Local search
Dynamic objective function

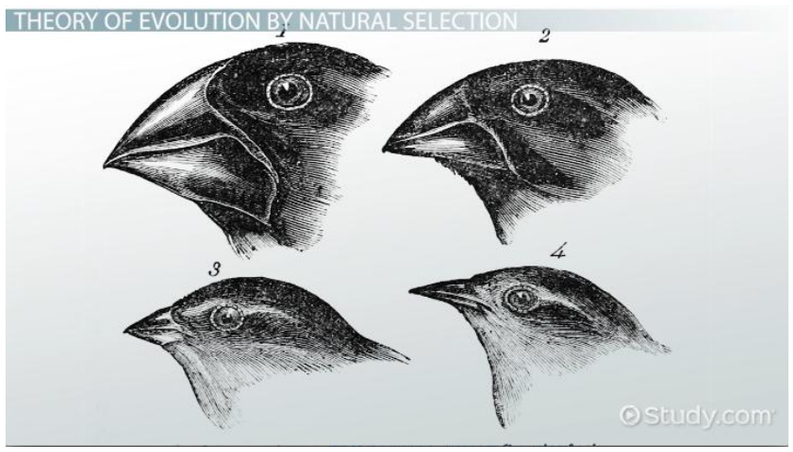
Metaheuristics

- Can have an entire course devoted to metahueristics
- There are actually entire journals and conferences devoted just to metaheuristics
- GECCO is the most prominent venue in the area

The power of metaphor

- As you might have already guessed, many metaheuristics are nature-inspired

- Assume that the natural world solves optimisation problems (implicitly) all the time!



The power of metaphor

- Genetic Algorithm and Genetic Programming - Natural Selection

- Particle Swarm Optimisation - flocking behaviour of birds and insects

- Simulated Annealing - the cooling of molten metal into solid metal

- Ant Colony Optimisation - how ants forage for food and communicated indirectly by depositing pheromones

Genetic Algorithm

17-year-old crashes car while driving with a beanie pulled over her eyes as part of viral 'Bird Box' challenge

Alexandra Ma Jan 12, 2019, 7:19 AM

Natural Selection - A very brief summary

- Mechanism by which organisms evolve and develop traits
- Environment acts as a form of pressure on a population of organisms
- Organism’s fitness is a result of how well it manages in its environment
- Organisms that survive mate
    - Fit organisms tend to want to mate with other fit organisms
    - Produce offspring
    - Offspring might acquire some mutation during gestation
- Offspring then continue the cycle

Natural Selection - A very brief summary

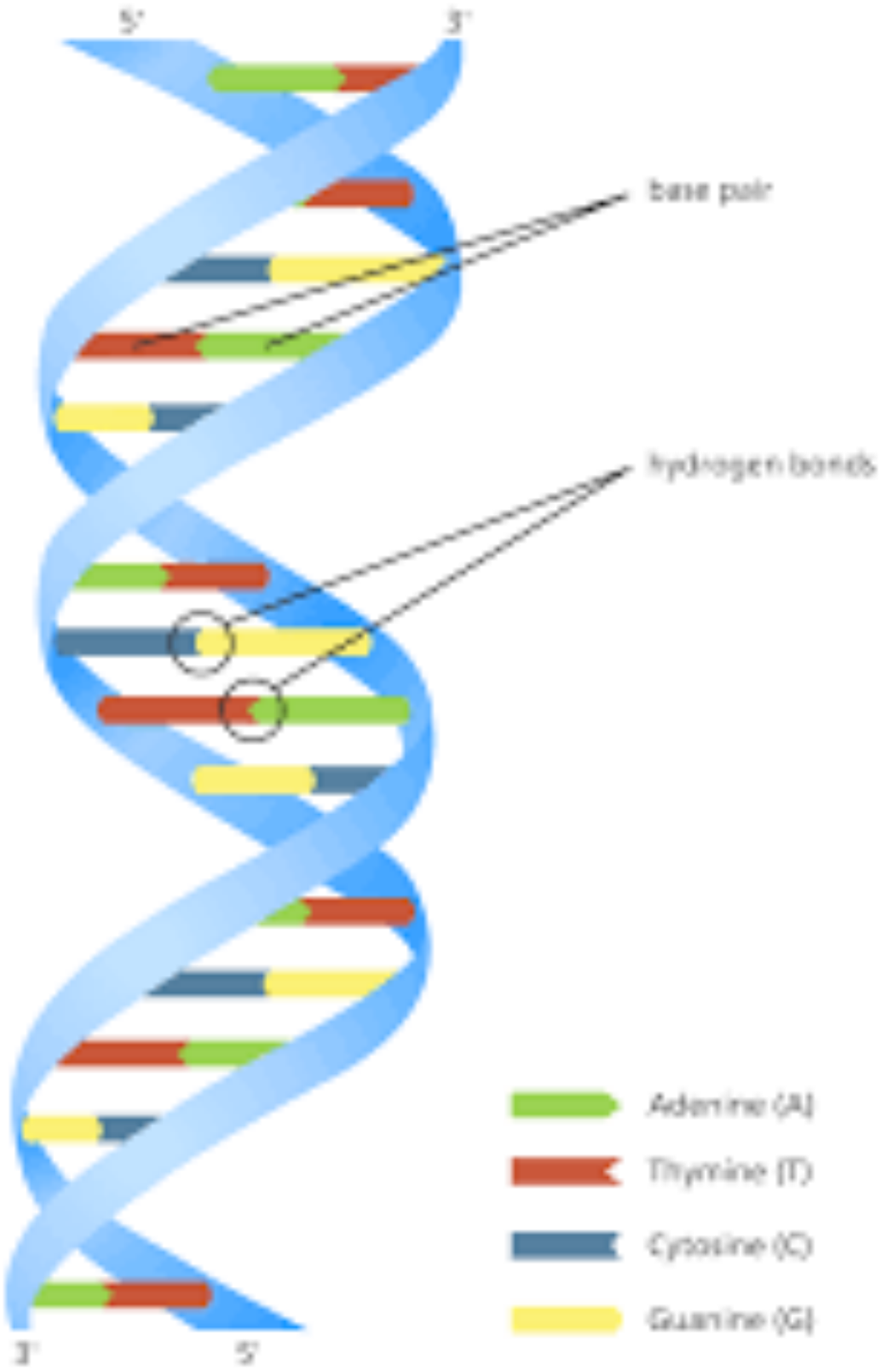
- Mechanism by which organisms evolve and develop traits
- Environment acts as a form of pressure on a **population** of **organisms**
- Organism's **fitness** is a result of how well it manages in its environment
- Organisms that survive mate
    - Fit organisms tend to want to **mate** with other fit organisms
    - **Produce** offspring
    - Offspring might acquire some **mutation** during gestation
- Offspring then **continue** the cycle

A bit of genetics

- Recall that the features of an organism are partially encoded by their genetics

- Hence, their fitness for their environment is at least partially determined by their genetics!

- DNA and chromosomes comprises of a sequence genes that work in concert with one another to encode features


5'
3'
base pair
hydrogen bonds
3'
5'
Adenine (A)
Thymine (T)
Cytosine (C)
Guanine (G)

If DNA/chromosomes is a sequence of genes, from a CS perspective, what is DNA?

ARRAYS!

Key insights of genetic algorithm

- We can represent features (such as the components) of our design points as genes

- Represent the entire design point as an array of said features (like a chromosome)

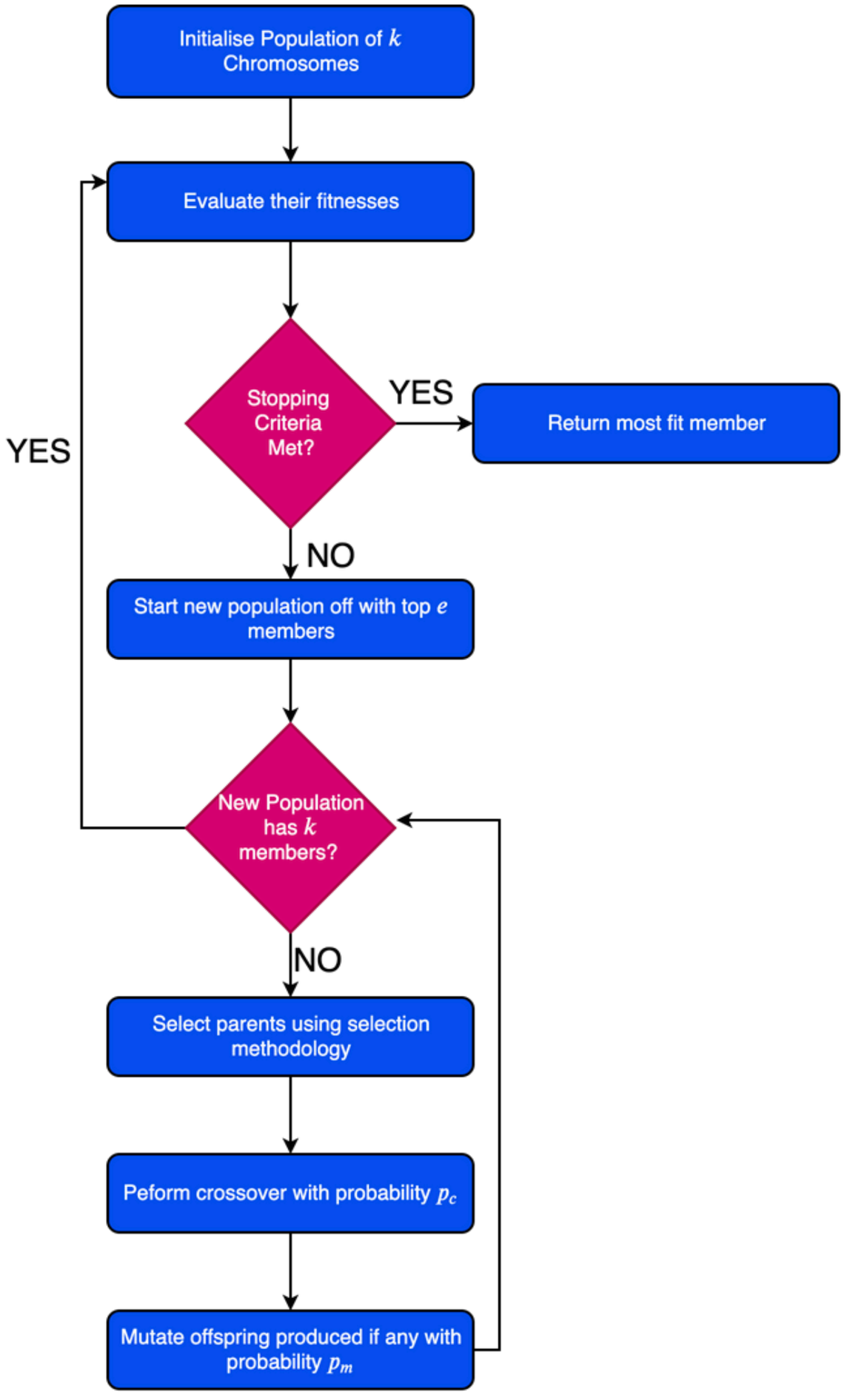
- Use chromosome as input into some fitness function that assess how good the design point is

Key insights of genetic algorithm

- When organisms mate, their genetics are melded together - so we can combine our artificial chromosomes together similarly

- When organisms mutate, their genes change - can change values inside of the array

- We repeatedly apply this procedure to maximise some objective function that acts as our fitness function


Initialise Population of k Chromosomes
Evaluate their fitnesses
Stopping Criteria Met?
YES
Return most fit member
NO
Start new population off with top e members
New Population has k members?
YES
NO
Select parents using selection methodology
Peform crossover with probability p_c
Mutate offspring produced if any with probability p_m

Decisions, decisions, decisions

- The above is an outline
- We need to decide on some problem specific details
- Need to define
  - Representation
  - Crossover operation
  - Mutation
  - And of course fitness

Representation of Chromosomes

- We need to design an encoding of our design points as arrays.
- Each problem is different
- Hence would have to select our representation to suite
- Choosing a good representation is vital
- Our choice of representation also dictates how we will randomly initialise our population
- And how we apply other operations

Representation of Chromosomes

- Three main ways:
    - Real vectors - used for problems over real vector space. Each gene is component of vector is a real number representing a component
    - Binary arrays - each entry is 0 or 1 representing true or false for some feature associated with that entry. These can also be used to represent integer values
    - Permutations - the entire chromosome is a permutation

Selection strategy

- Roulette Wheel selection - all valid chromosomes in a population are considered for mating. Parents are selected stochastically in proportion to their fitness value

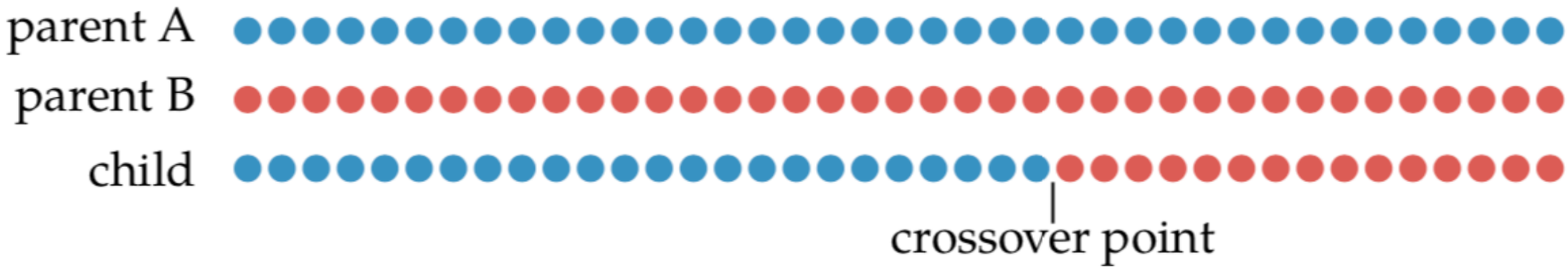
- Tournament Selection - t chromosomes are randomly selected. The top 2 are taken and then mated

- Truncation Selection - pick top t chromosomes, randomly choose 2 and mate them

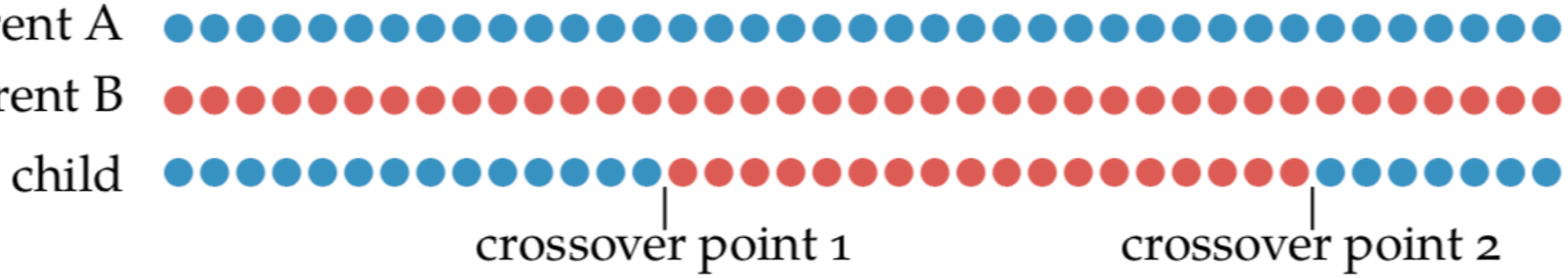
Crossover Methods

- Different methods of crossover
  - Single-point crossover
  - Multiple-point crossover
  - Uniform crossover
  - Interpolation crossover

Single Point Crossover



Double-point crossover



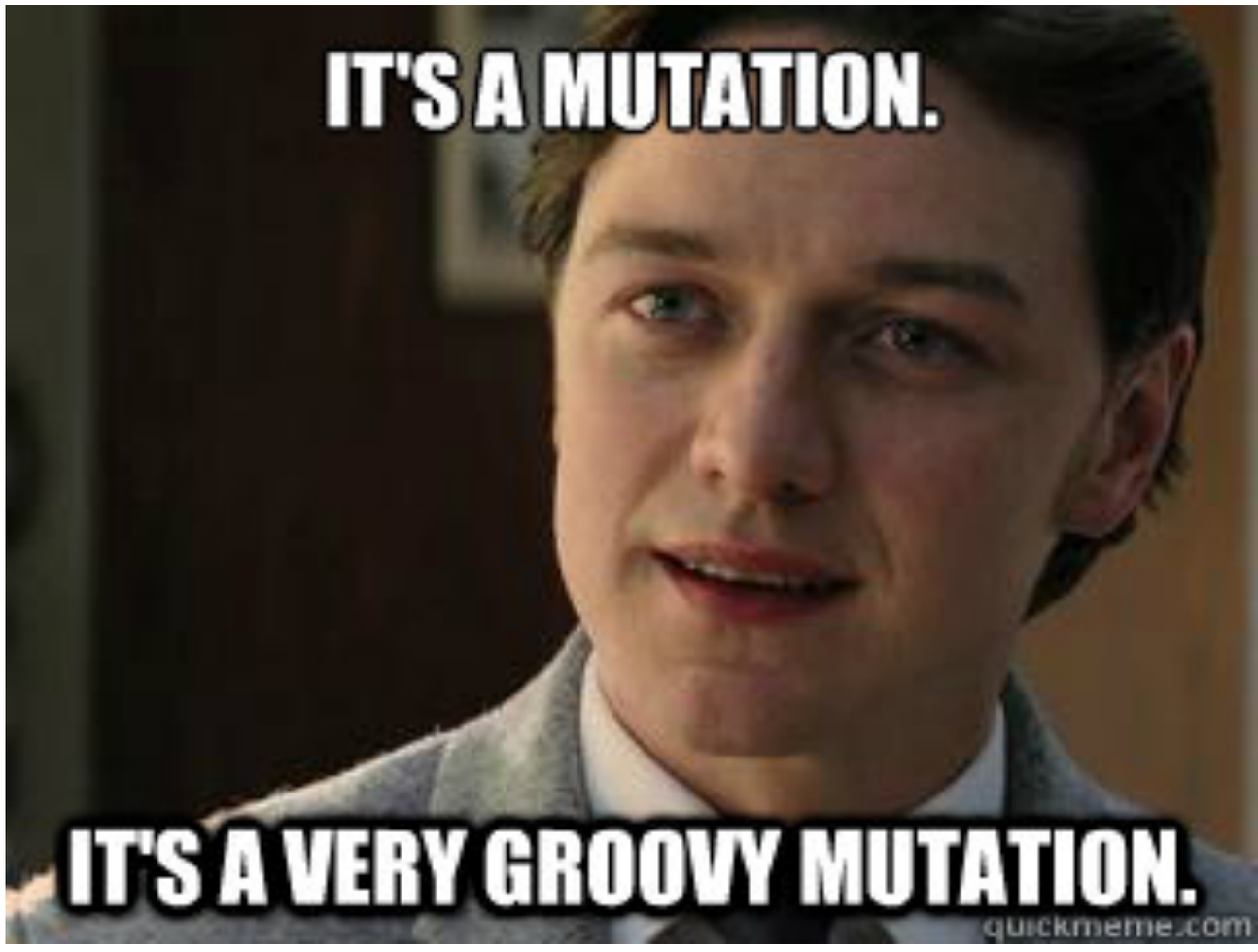
Uniform Crossover

Interpolation Crossover

- Consider there being a line between two vectors x_1 and x_2. This line is defined by the expression $(1 - \lambda)x_1 + \lambda x_2$ where $0 \leq \lambda \leq 1$
- We can look at crossover as choosing a point on this line.
  - We either randomly select a λ
  - Or set λ to a fixed value (usually 0.5)
- Interpolation crossover can only be done on chromosomes that are real vectors

Mutation

- Mutation can take on many different forms depending on the nature the representation
- Recall that our chromosome can be conceived as existing in some space of potential chromosomes
- Mutation makes this chromosome randomly jump to another neighbouring solution
- Mutation helps us avoid getting stuck in a bad local minimum

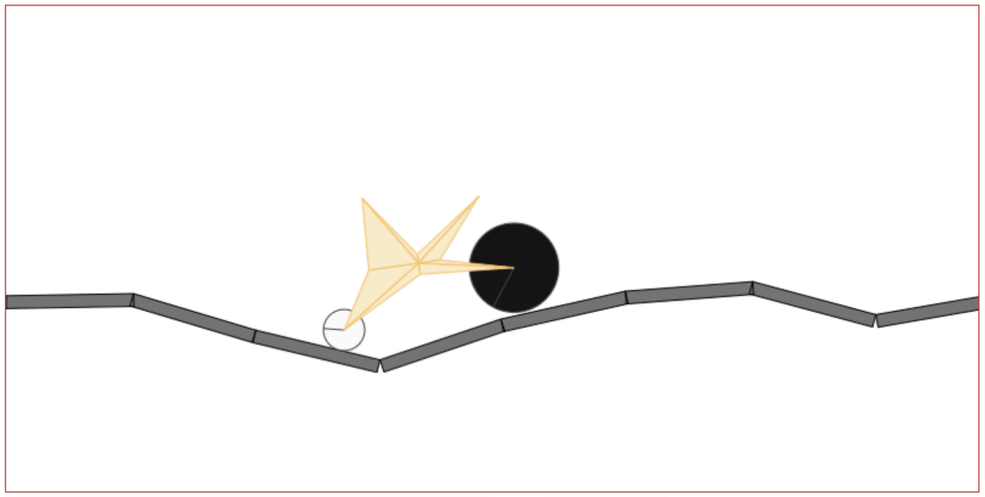


Mutation

- Possible interpretations
  - Adding standard Gaussian noise to a real vector chromosome
  - Flipping a bit in a bit-based chromosome
  - Swapping elements in a permutation based chromosome

GA Usage

- Using GAs require a deal of thought
- But they tend to work well when our search space is high, but number of feasible points is low
- Examples:
  - Scheduling
  - Automated design



Donate

Save Population | Restore Saved Population

Surprise! | New Population

Fast Forward

Create new world with seed

Enter any string | Go!

Generation	0
Cars alive	2
Distance	51.84 meters
Height	-0.34 meters
Mutation rate:	5%
Mutation size:	100%
Floor:	fixed
Gravity:	Earth (9.81)
Elite clones:	1

View top replay

Watch Leader

† | †
2 | †
† | †
† | 6
† | †
† | †
† | †
† | †

Particle Swarm Optimisation

- Based off of flocking behaviour of birds and insects
- “Follow” the leader type strategy
- Each particle has a memory of its previous best state
- Each solution represented as a particle
  - Usually a real vector
- At the end of every iteration
  - Compute a velocity of each particle that gives it a direction to travel in
  - Add velocity to particle to compute new direction of particle
- Unlike GA, we can naturally use PSO for both maximisation and minimisation problems

Velocity Computation

- To decide on a particle's velocity, we take into account three pieces of information
  - Inertia - the resistance to change velocity
  - Exploration factor - tendency to explore the current globally best behaviour
  - Exploitation factor - tendency to follow previously known best personal behaviour

PSO Notation

- $x^{(i)}$ is the ith particle
- $v^{(i)}$ is the velocity of the ith particle
- $x_{\text{best}}^{(i)}$ is the best state of particle i in its entire history
- x_{best} is the state of the best performing particle over all iterations
- w - inertial factor, usually set to a value between 0 and 1
- $r1$ - exploitation factor, randomly choose each time from $\mathcal{U}(0,1)$
- $r2$ - exploration factor, randomly choose each time from $\mathcal{U}(0,1)$

$$v^{(i)} = wv^{(i)} + r_1(x^{(i)}_{\text{best}} - x^{(i)}) + r_2(x_{\text{best}} - x^{(i)})$$

$$x^{(i)} = v^{(i)} + x^{(i)}$$

```
function pso(f, k,w, num_iter):
    population = []
    p_best = None
    best_score = -inf
    for i = 1 to k do
        population.append(init_particle())
    for i = 1 to num_iter:
        for p in population do
            f_p = f(p.state)
            if f_p > p.prev_best_score:
                p.prev_best = p.state
            if f_p > best_score:
                best_score = f_p
                p_best = p.state
    for p in population do:
        r_1 = random_uniform(0, 1)
        r_2 = random_uniform(0, 1)
        velocity = w * p.prev_velocity + r_1(p.prev_best - p.state) + r_2(p_best - p.state)
        p.state = p.state + velocity
        p.prev_velocity = velocity
return p_best
```

Hill-Climbing Techniques

- GA and PSO are what are referred to as population based meta-heuristics

- They generate a random population, and use them in concert with one another to find a good design point

- Some methods, however, instead focus on randomly generating a single design point and iteratively refining it

- Hill-climbing techniques are a family of such techniques

Hill-Climbing Techniques

- Core idea, start at random point
- Each point has neighbouring points
- Examine some neighbouring points and test if they are better than our current point
  - If better move to that point
  - If equal or worse stay where we are
  - continue until converge criteria met

Hill-Climbing Techniques

- There is a major problem with vanilla hill-climbing techniques
- Vanilla hill-climbing techniques, more formally called deterministic hill climbing techniques can get a stuck in a bad local optimal point
- Sometimes, we need to get worse before we get better
- Stochastic hill-climbing techniques avoid this problem by computing some probability of moving to a neighbouring point, and performing a Bernouilli Trial

Simulated Annealing

- SA is a variant of stochastic hill climbing
- Inspired by the process of smelting metal
- We provide a temperature schedule that controls volatility or willingness to accept a worse point in hopes of going to a better point in the future
- As time goes on (our iterations increase), temperature falls, thereby lowering volatility

Simulated Annealing - Probabilistic Jumping

- Suppose that we are a point x_1 and we then jump to a point x_2

- Assuming that we want to maximise f, we let $\Delta f = f(x_2) - f(x_1)$.

- We go to point x_2 with the probability p where

$$p \begin{cases} 1 \text{ if } \Delta f \leq 0 \\ e^{-\frac{\Delta f}{t_i}} \text{ otherwise} \end{cases}$$

Simulated Annealing - Probabilistic Jumping

- Suppose that we are a point x_1 and we then jump to a point x_2

- Assuming that we want to minimise f, we let $\Delta f = f(x_2) - f(x_1).$

- We go to point x_2 with the probability p where

$$p \begin{cases} 1 \text{ if } \Delta f \leq 0 \\ e^{-\frac{\Delta f}{t_i}} \text{ otherwise} \end{cases}$$

← Metropolis Criteria

Simulated Annealing - Temperature Schedules

- Different types of temperature schedules
  - Fixed schedules - we come up with temperature values before hand
  - Non-fixed schedules - give starting temperature and parameters of schedule and temperature is dynamically calculated

Simulated Annealing - Temperature Schedules

- Exponential annealing:
  - $t_i = \gamma^i t_0, \qquad 0 < \gamma < 1$
- Fast annealing
  - $t_i = \frac{t_0}{i}$
- Logarithmic annealing
  - $t_i = \frac{t_0 \log(2)}{\log(\mathrm{i} + 1)}$

```
function simulated_annealing(f, x, T, t, k_max)
    y = f(x)
    x_best, y_best = x, y
    for k in 1 : k_max
        x′ = x + rand(T)
        y′ = f(x′)
        Δy = y′ - y
        if Δy ≤ 0 || rand() < exp(-Δy/t(k))
            x, y = x′, y′
        end
        if y′ < y_best
            x_best, y_best = x′, y′
        end
    end
    return x_best
end
```