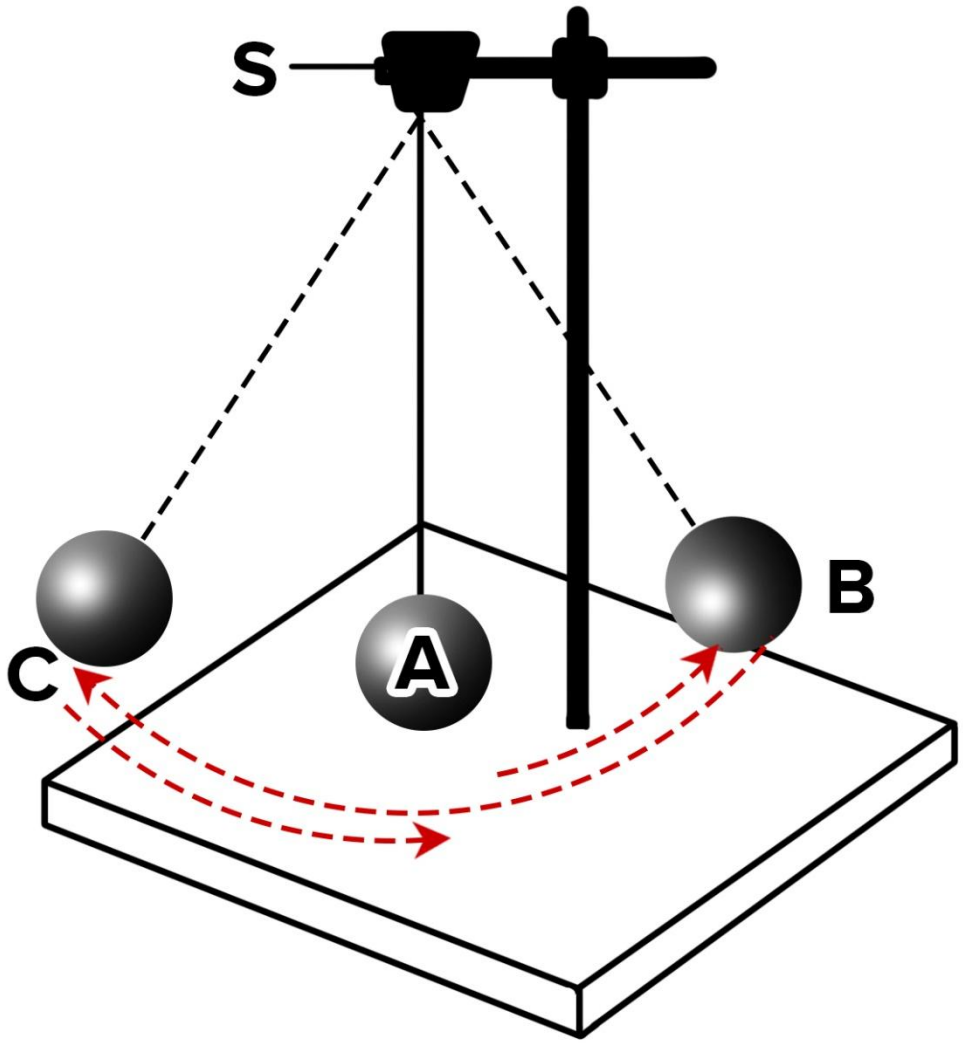
THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 06: Genetic Algorithms and Function Discovery

Sergei V. Kalinin

Functions from data

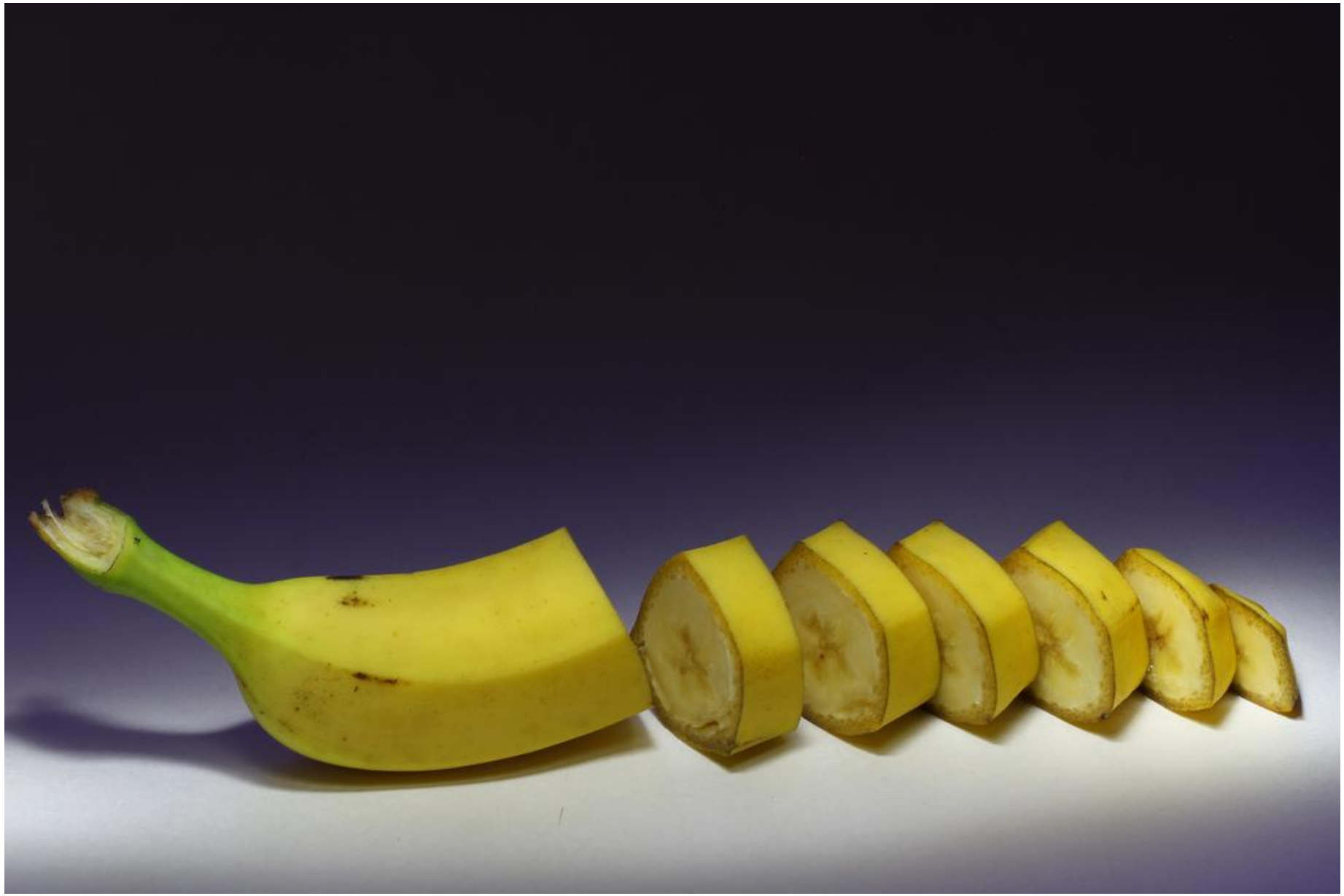
- If the function is known, we can fit it to data
- If we have several possible functions, we can fit all of them and compare the quality of fits
- We can also make some judgements based on the parameter values
- But what if we do not know the functions?



https://www.sciencealert.com/fruits-vegetables-before-domestication-photos-genetically-modified-food-natural

https://www.sciencealert.com/fruits-vegetables-before-domestication-photos-genetically-modified-food-natural

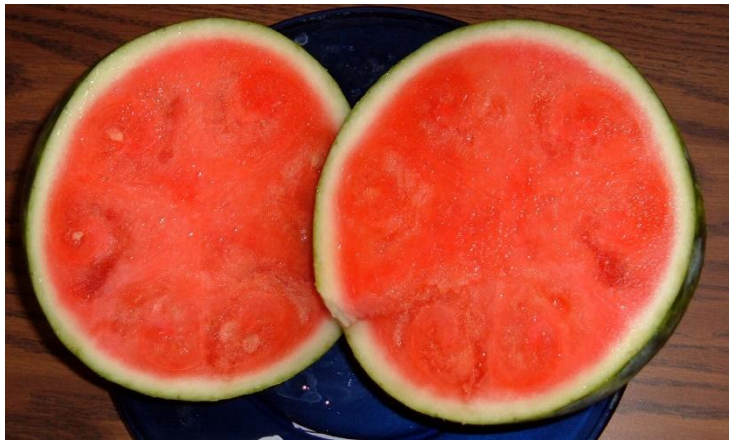
https://www.sciencealert.com/fruits-vegetables-before-domestication-photos-genetically-modified-food-natural



https://www.sciencealert.com/fruits-vegetables-before-domestication-photos-genetically-modified-food-natural

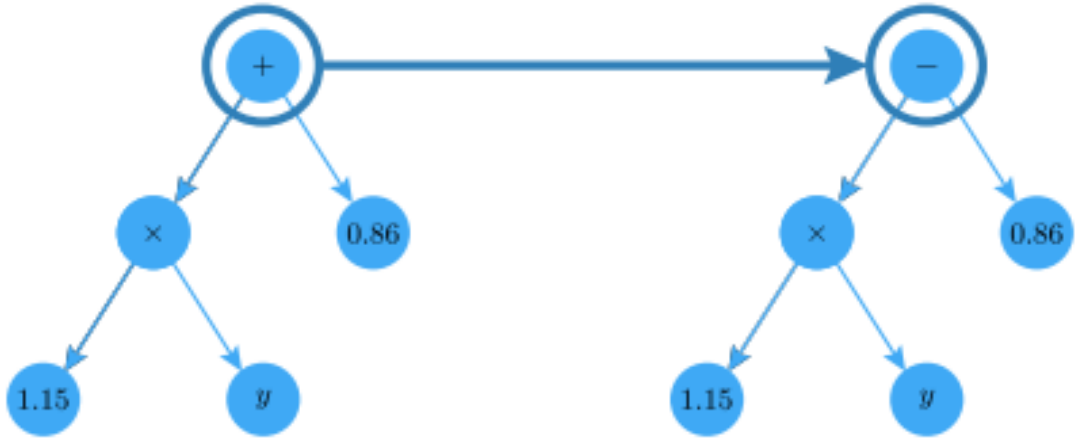




https://www.sciencealert.com/fruits-vegetables-before-domestication-photos-genetically-modified-food-natural

Functions can be represented as trees of operations!



$1.15y + 0.86$ $1.15y - 0.86$

https://arxiv.org/pdf/2305.01582.pdf

Genetic Algorithms

Genetic Algorithms (GAs) are a part of Evolutionary Computing (EC), which is a rapidly growing area of Artificial Intelligence (AI). It inspired by the process of biological evolution based on Charles Darwin's theory of natural selection, where fitter individuals are more likely to pass on their genes to the next generation.

- The GA, developed by John Holland and his collaborators in the 1960s and 1970s.
- As early as 1962, John Holland's work on adaptive systems laid the foundation for later developments.
- By the 1975, the publication of the book "*Adaptation in Natural and Artificial Systems*", by Holland and his students and colleagues.
- The GA got popular in the late 1980s by was being applied to a broad range of subjects that are not easy to solve using other techniques.
- In 1992, John Koza has used genetic algorithm to evolve programs to perform certain tasks. He called his method "*genetic programming*" (GP)[3].

https://towardsdatascience.com/an-introduction-to-genetic-algorithms-c07a81032547

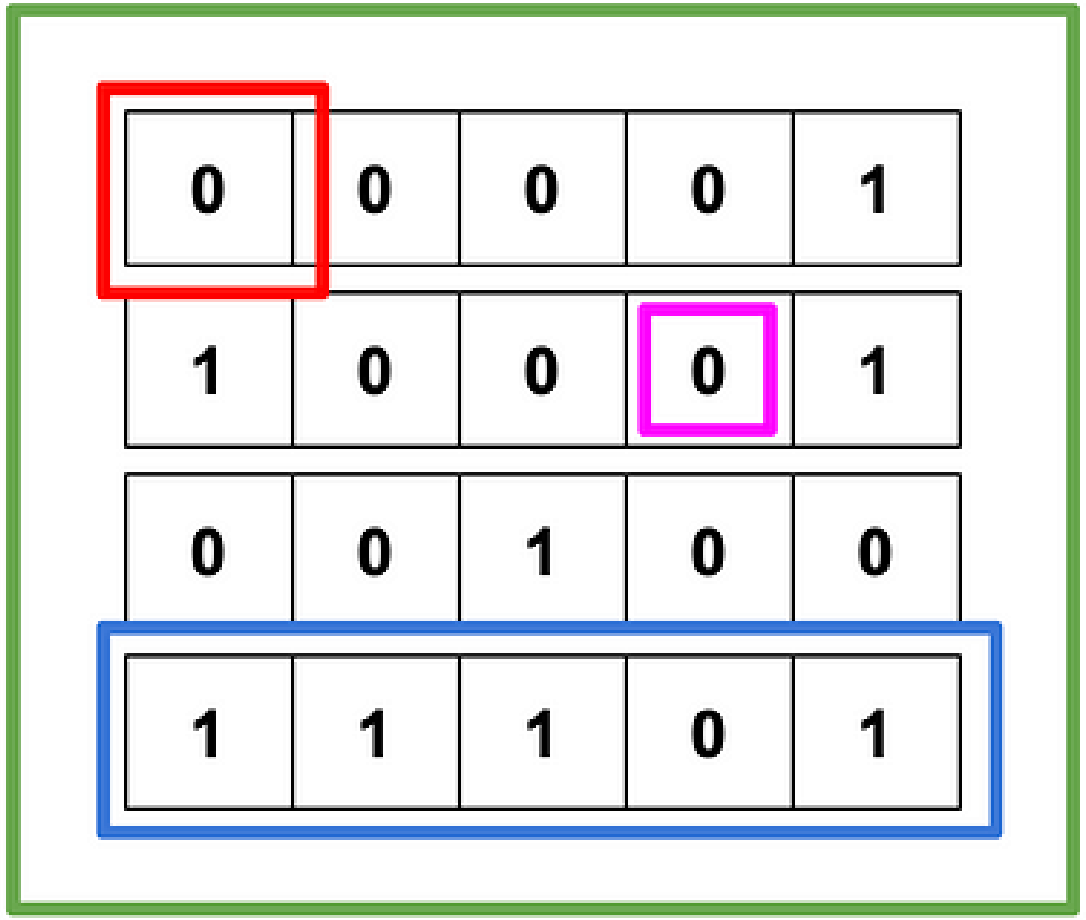
Genetic Algorithms in Nature

- Each cell of a living thing contains chromosomes — strings of DNA.
- Each chromosome contains a set of genes — blocks of DNA
- Each gene determines some aspect of the organism (like eye color)
- A collection of genes is sometimes called a genotype
- A collection of aspects (like eye color) is sometimes called a phenotype
- Reproduction (crossover) involves recombination of genes from parents and then small amounts of mutation (errors) in copying
- The fitness of an organism is how much it can reproduce before it dies
- Evolution based on “survival of the fittest”

https://towardsdatascience.com/from-biology-to-computing-an-introduction-to-genetic-algorithms-b39476743483

https://towardsdatascience.com/an-introduction-to-genetic-algorithms-c07a81032547

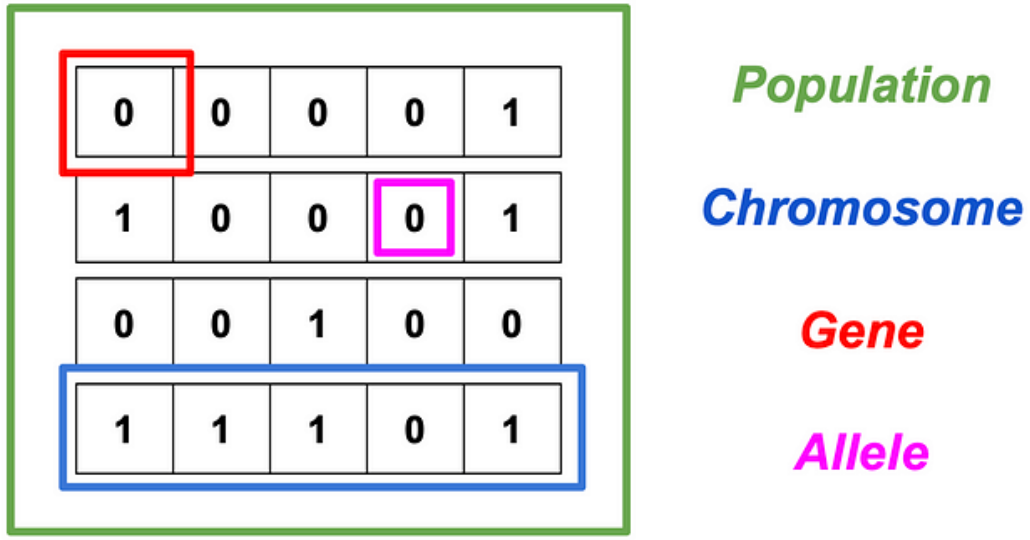
Key element: representation



https://towardsdatascience.com/an-introduction-to-genetic-algorithms-c07a81032547

Key element: representation

Traveling Salesman Problem (TSP): Each chromosome is a sequence (array) of city numbers, representing a specific route.



Optimizing Neural Network Architectures: Each chromosome could be a string or array representing the layers in the network, where each gene represents the type of layer (e.g., convolutional, pooling), its size, activation function, etc.

Portfolio Optimization: Each chromosome can be an array where each element represents the proportion of total funds allocated to a specific investment.

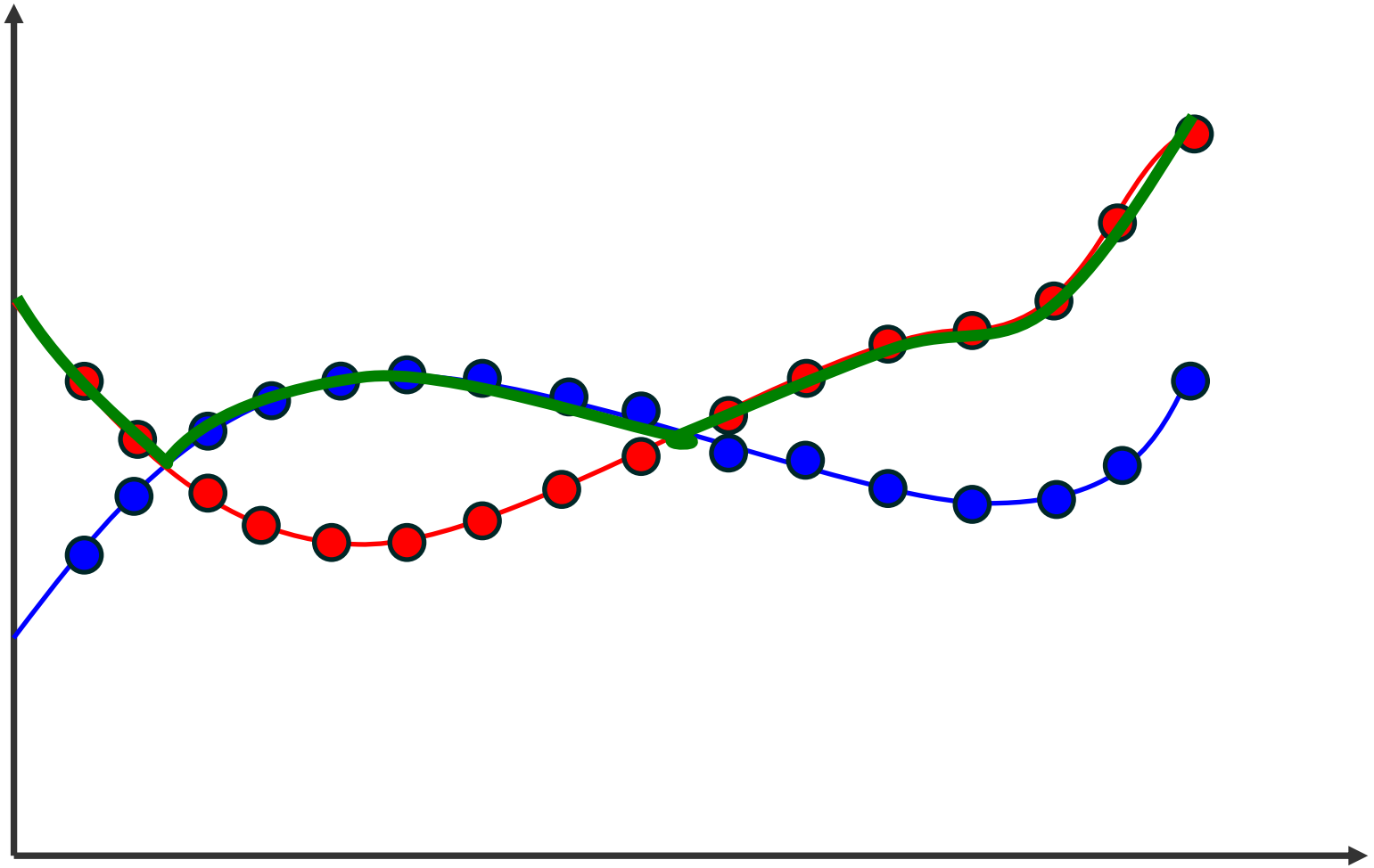
Genetic Programming: Chromosomes are tree structures representing computer programs. Nodes and branches of the tree represent operations and operands in the program.

Molecular design: representations are SMILES and SELFIES

Caveat: we need to design the crossover/mutation for each classes of problems

https://towardsdatascience.com/an-introduction-to-genetic-algorithms-c07a81032547

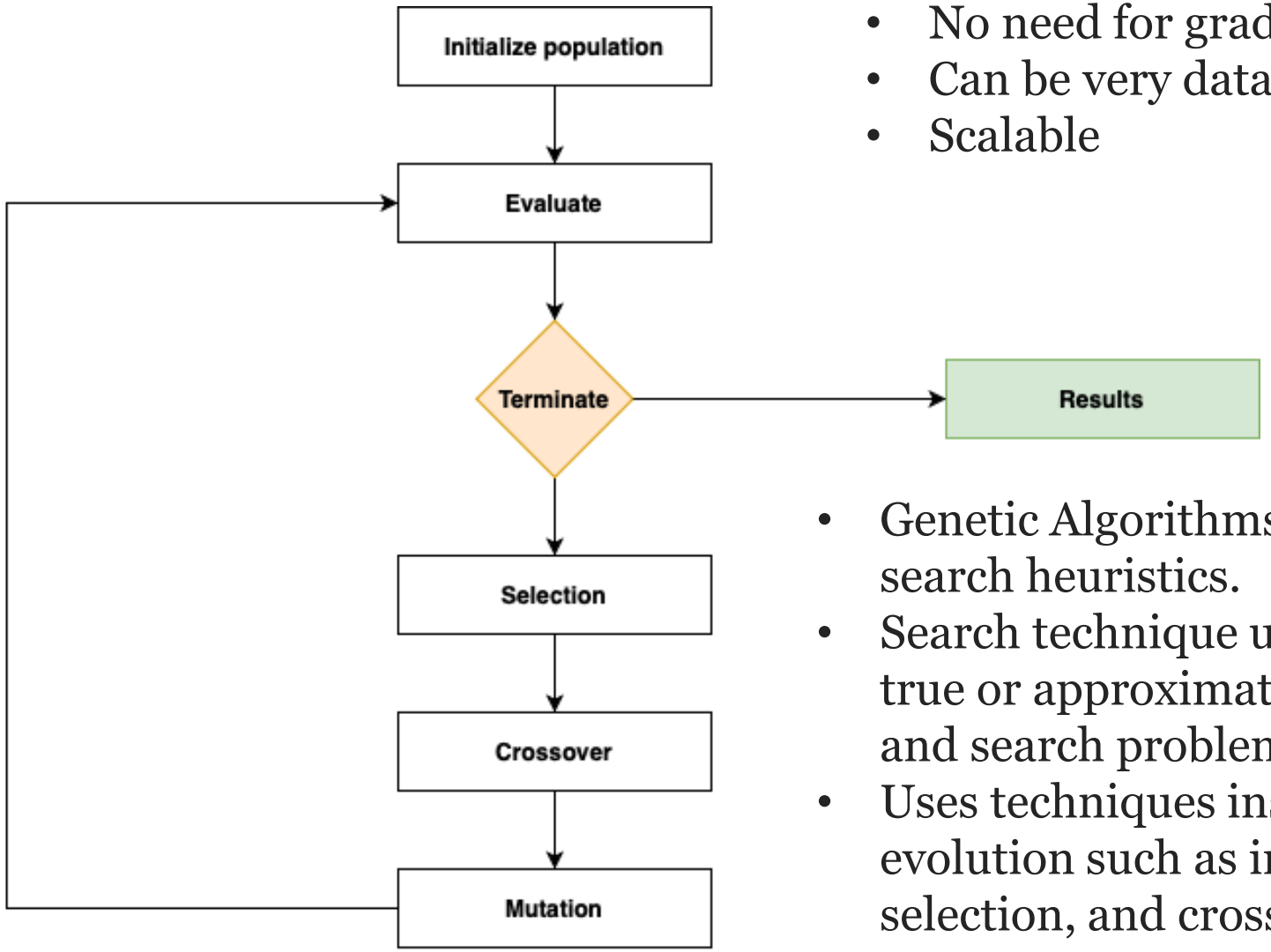
Key element: representation



Representing functions:

1. Just list of values
2. … or expansion in the polynomial or any other series

General principle of GA



- No need for gradients
- Can be very data intensive
- Scalable

- Genetic Algorithms are categorized as global search heuristics.
- Search technique used in computing to find true or approximate solutions to optimization and search problems.
- Uses techniques inspired by biological evolution such as inheritance, mutation, selection, and crossover.

https://towardsdatascience.com/an-introduction-to-genetic-algorithms-c07a81032547

General principle of GA

Initialize population: genetic algorithms begin by initializing a **Population** of candidate solutions. This is typically done randomly to provide even coverage of the entire search space. A candidate solution is a C**hromosome** that is characterized by a set of parameters known as **Genes**.

Evaluate: next, the population is evaluated by assigning a fitness value to each individual in the population. In this stage we would often want to take note of the current fittest solution, and the average fitness of the population.

After evaluation, the algorithm decides whether it should terminate the search depending on the **termination conditions** set. Usually this will be because the algorithm has reached a fixed number of generations or an adequate solution has been found.

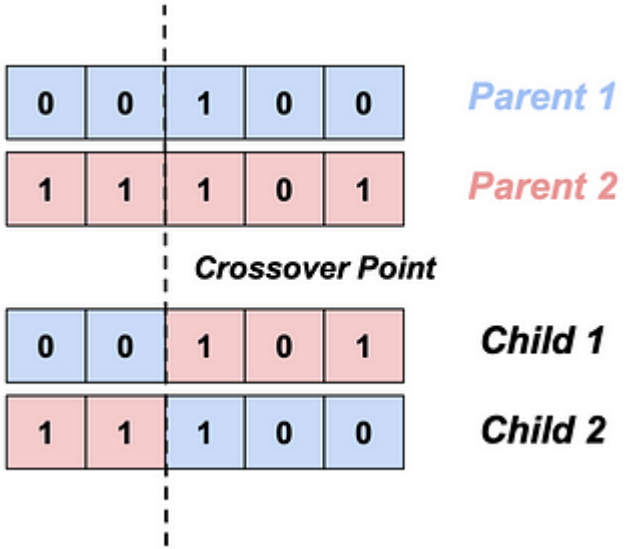
When the termination condition is finally met, the algorithm will break out of the loop and typically return its final **search results** back to the user.

General principle of GA

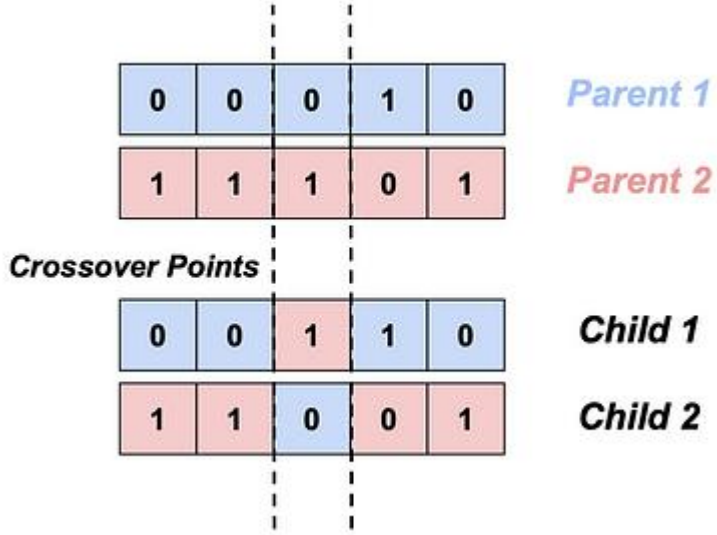
- **Selection:** if the termination condition is not met, the population goes through a selection stage in which individuals from the population are selected based on their fitness score, the higher the fitness, the better chance an individual has of being selected.
- Two pairs of selected individuals called **parents.**
- **Crossover:** the next stage is to apply crossover and mutation to the selected individuals. This stage is where new individuals (**children**) are created for the next generation.
- **Mutation:** at this point the new population goes back to the evaluation step and the process starts again. We call each cycle of this loop a generation.

Cross-over

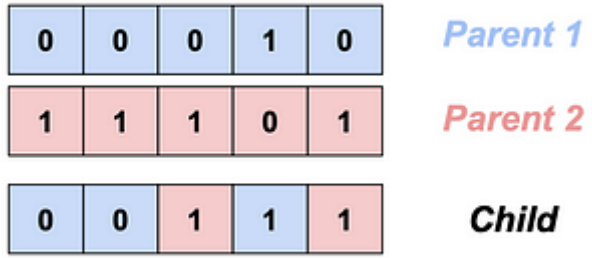
One-point crossover: *Swap the genes from a selected point on the parents' chromosomes:*



Two-point crossover: *Swap the genes from two selected points on the parents' chromosomes:*



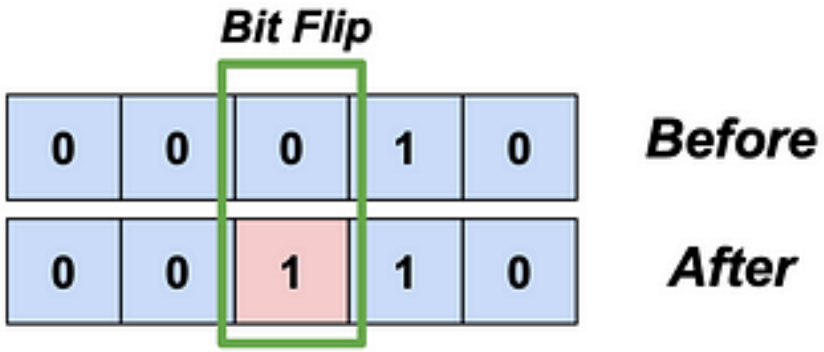
•**Uniform crossover:** *Each gene is randomly selected from the corresponding genes from the two parents:*



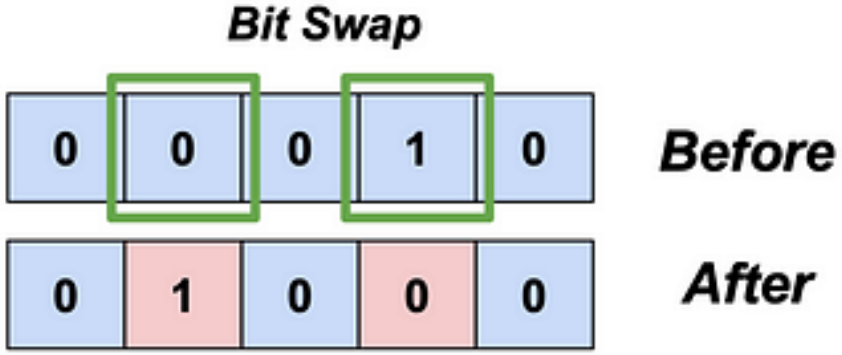
https://towardsdatascience.com/from-biology-to-computing-an-introduction-to-genetic-algorithms-b39476743483

Mutation

Bit flip: *Iterate through the genes in a chromosome and with each pass randomly flip a bit with a small probability:*



Swap: *Select two genes, with low probability, in the chromosome and swap them:*



https://towardsdatascience.com/from-biology-to-computing-an-introduction-to-genetic-algorithms-b39476743483

Selection methods

After we have our population, we need to select solutions (parents) to take forward to produce offspring for the next generation. The parents are selected based on their fitness to ensure the 'best' genes are passed on.

- **Elitism:** Choose a certain number of solutions in the current population with the best fitness function.

- **Roulette Wheel:** Sample from a probability distribution of the solutions (chromosomes) in the current population where each solution's probability, p_s, is proportional to its fitness score, f_s:

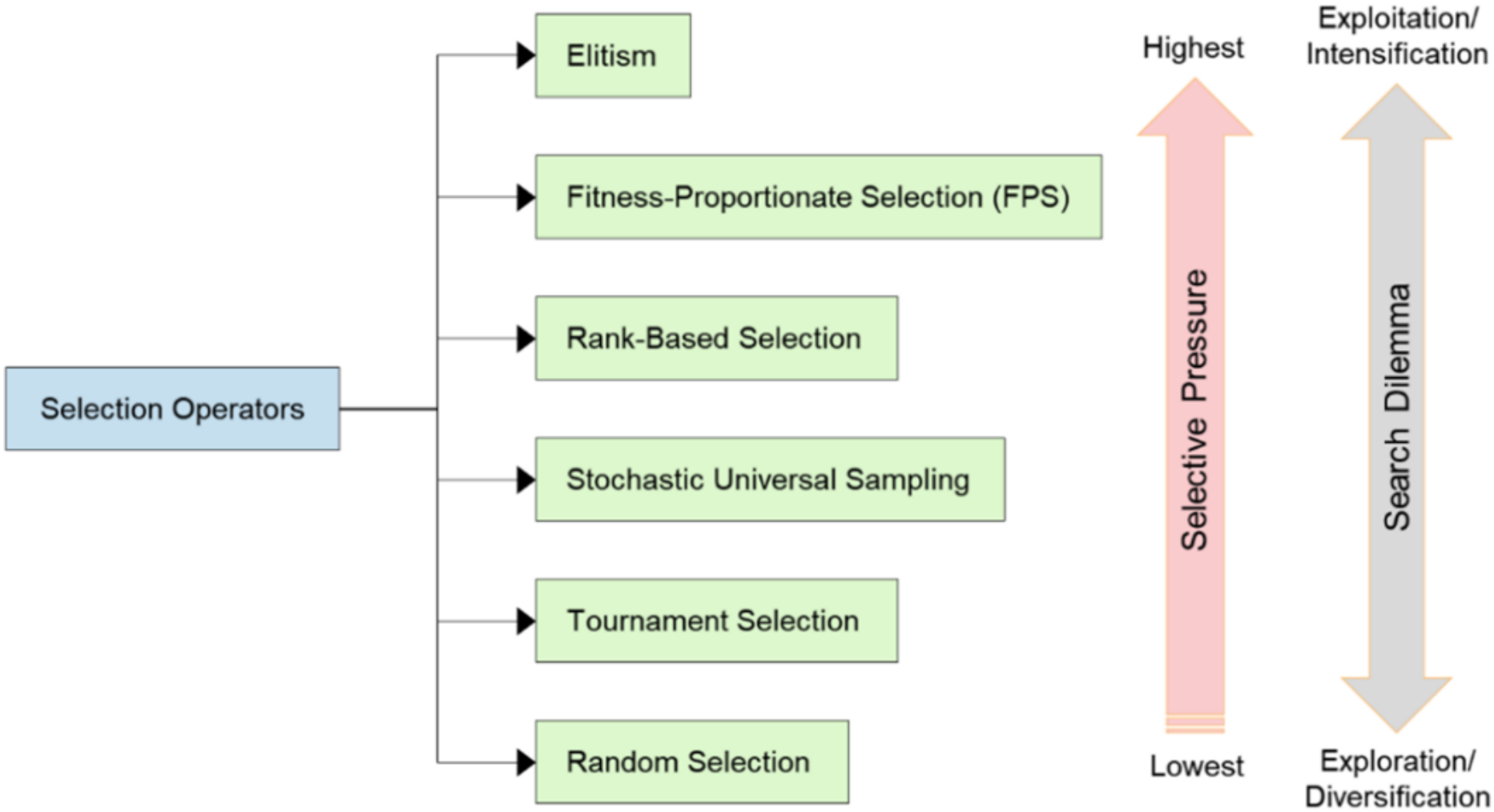
$$p_s = \frac{f_s}{\sum_{s=1}^{N} f_s}$$

- **Tournament:** This involves selecting solutions at random and running tournaments with these subset solutions where the winner is the one with the best fitness score.

There are many other methods for the selection process and different variants of the techniques listed above. You can also combine selection procedures to generate hybrid methods as well. There is no 'one size fits all' and it's best to experiment with various types.

https://towardsdatascience.com/from-biology-to-computing-an-introduction-to-genetic-algorithms-b39476743483

Selection methods

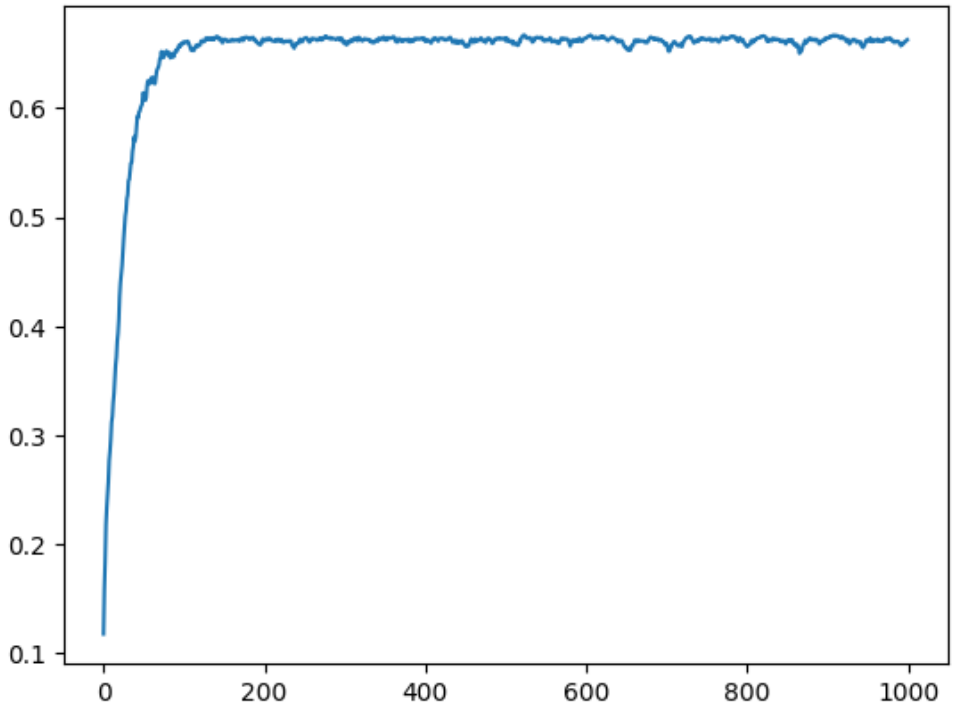


https://livebook.manning.com/book/optimization-algorithms/chapter-7/v-11/139

Termination

- A certain number of generations reached
- A desired fitness is achieved
- Computational resource exhausted
- Fitness score has plateaued

```
RCFMcsMTDo,,ds
iJrjvRnbMPraOP
psQjeuocixTcfz
.msh,!QraH!KAu
pZ QzsjTdUVAPx
qfZgqrFG.IILhW
S,iQKcNiGAUJW!
tfFGWZa vqyUwM
YO.vNv.aCRnUEM
!huDlOebruGmys
```

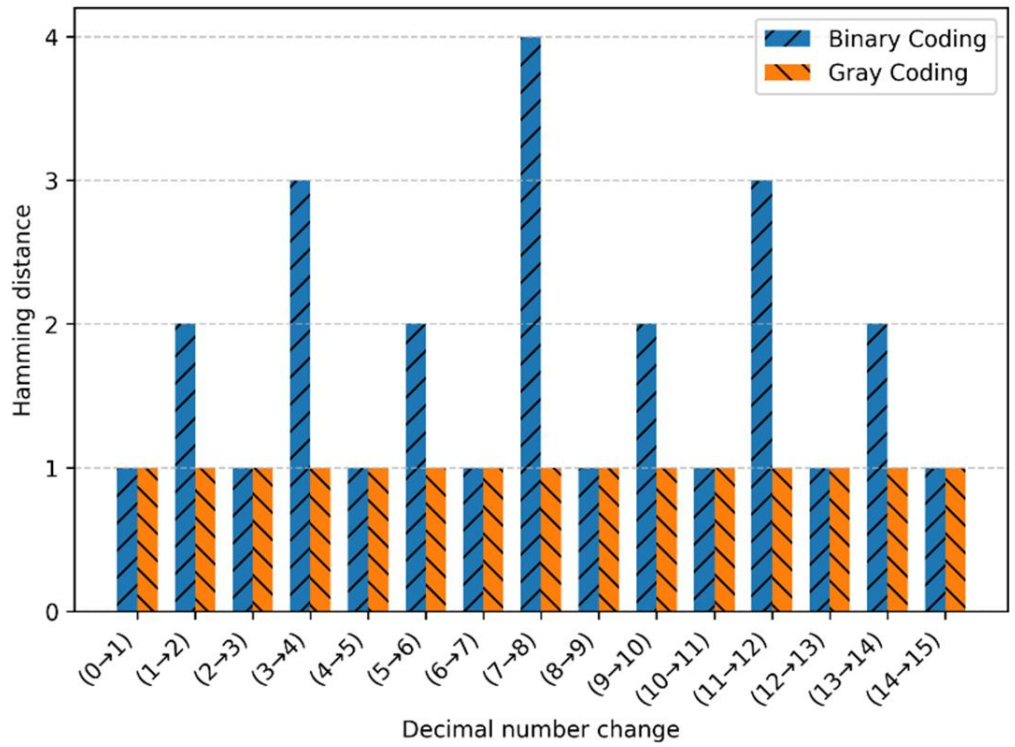



```
HellgoloWrld
Helllo  Wold
Hell o  Wrd!
Hell olcWor!
Helloo oWol!
Hell o  Wrr!
Hell,o  Wol!
Helloo  aol!
Helloo  Wol!
Hell o oWol!
Hell,ol Wold
Hell o oWol!
Helllo YWod!
Helleo  Wold
```

https://towardsdatascience.com/from-biology-to-computing-an-introduction-to-genetic-algorithms-b39476743483

Colab!

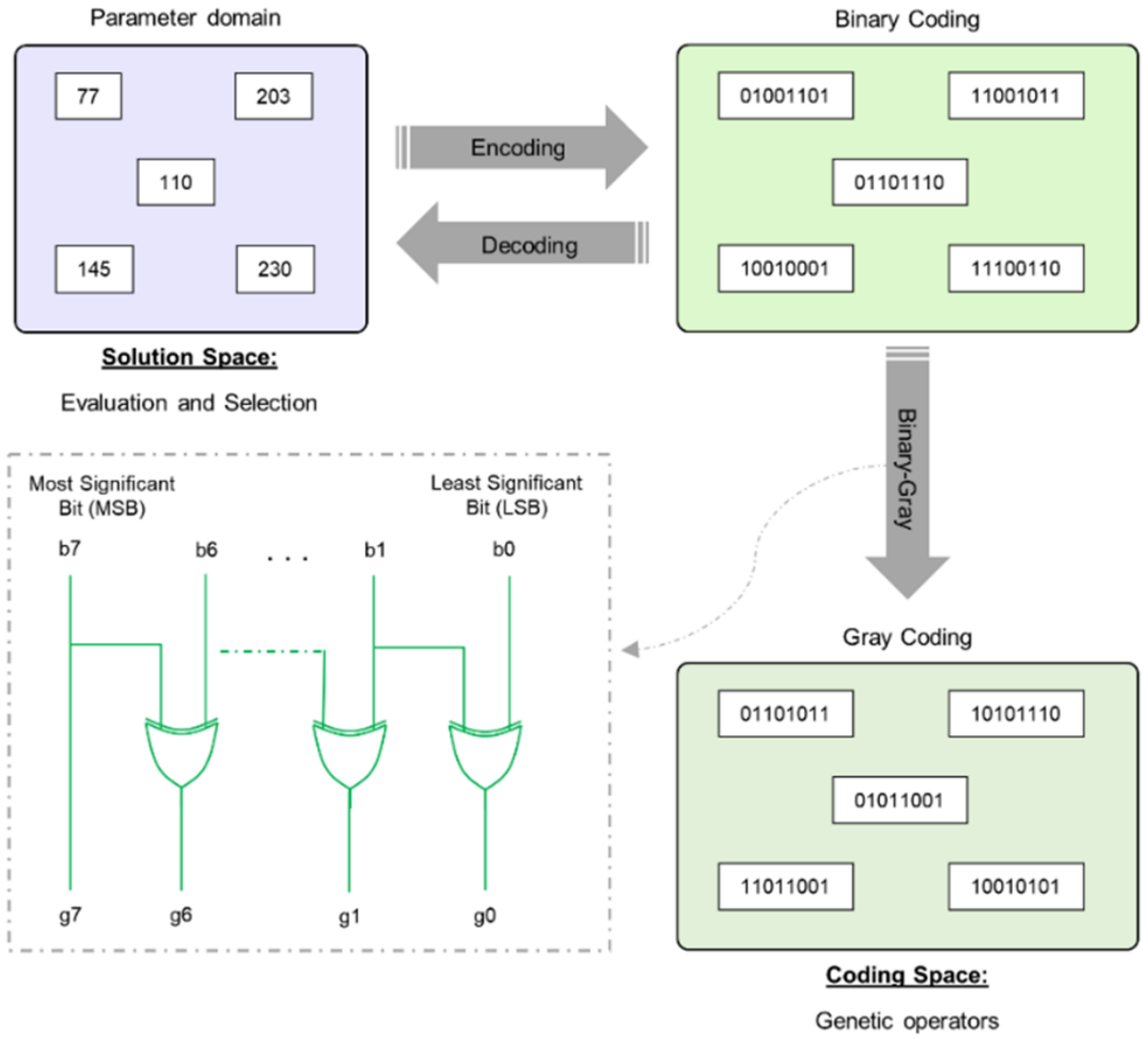
Humming Cliffs



0: 00000
1: 00001
2: 00010
3: 00011
4: 00100
5: 00101
6: 00110
7: 00111
8: 01000
9: 01001
…

One of the drawbacks of encoding variables as binary strings is the presence of Hamming cliffs. In binary-coded GAs, a small change in the encoded value (e.g., flipping a single bit) can lead to a significant change in the decoded value, especially if the flipped bit is located towards the most significant bit position. This abrupt change between two adjacent numbers in the search space is referred to as the “Hamming Cliff”. This problem negatively affects binary-coded GAs by disrupting the search space's smoothness, causing poor convergence, and leading to inefficient exploration and exploitation.

Solution: Gray coding



https://livebook.manning.com/book/optimization-algorithms/chapter-8/v-11/11

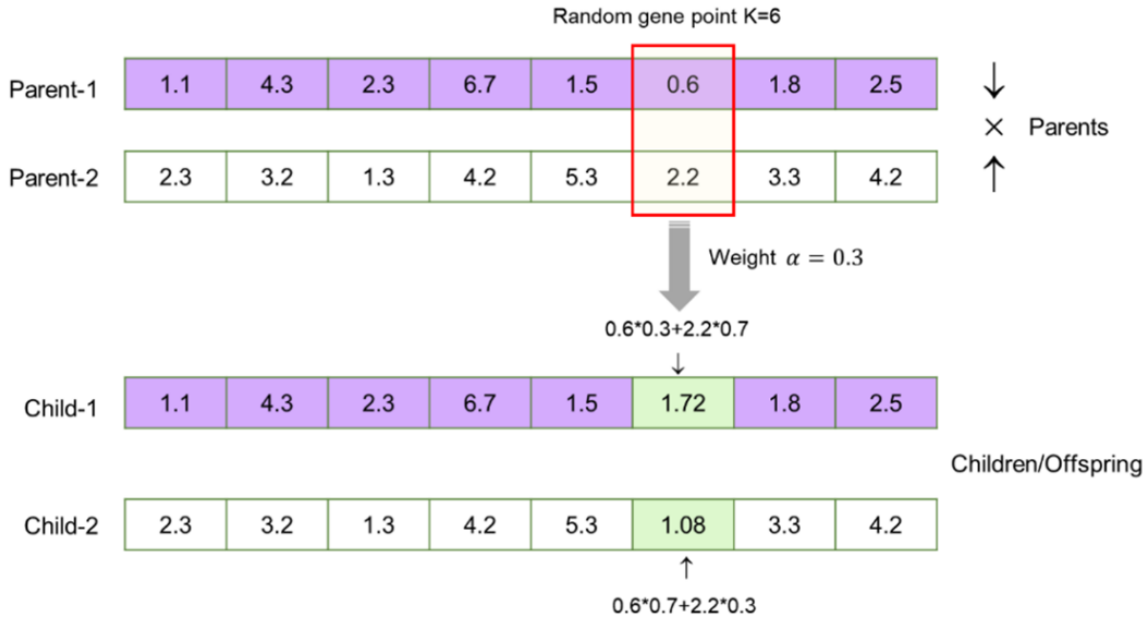
Humming Cliffs

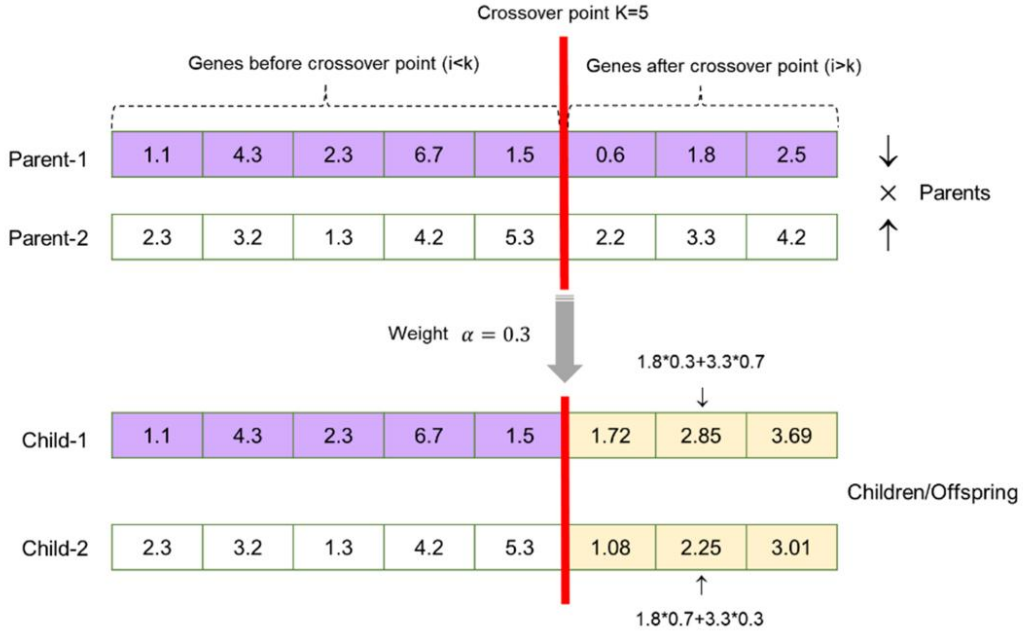
Table 8.1 Decimal, binary and Gray coding

Decimal Number	Binary Code	Gray Code	Decimal Number	Binary Code	Gray Code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

https://livebook.manning.com/book/optimization-algorithms/chapter-8/v-11/11

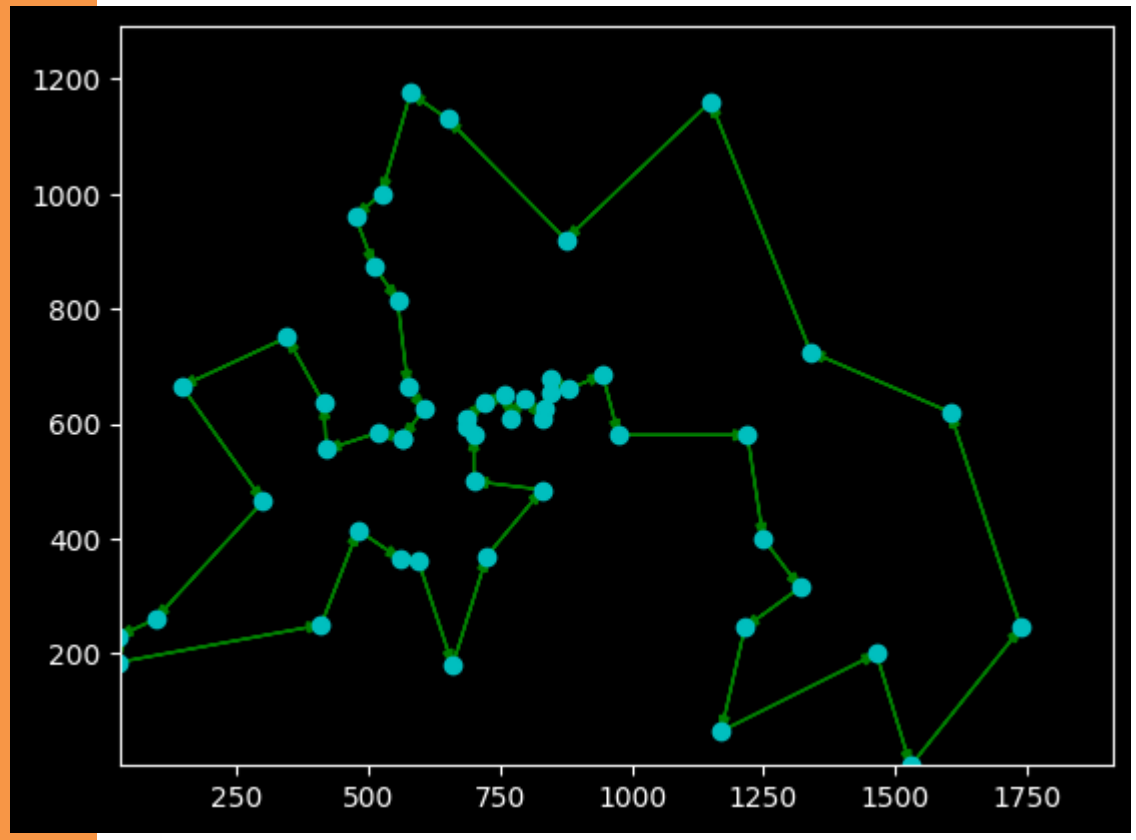
Encoding by continuous vectors

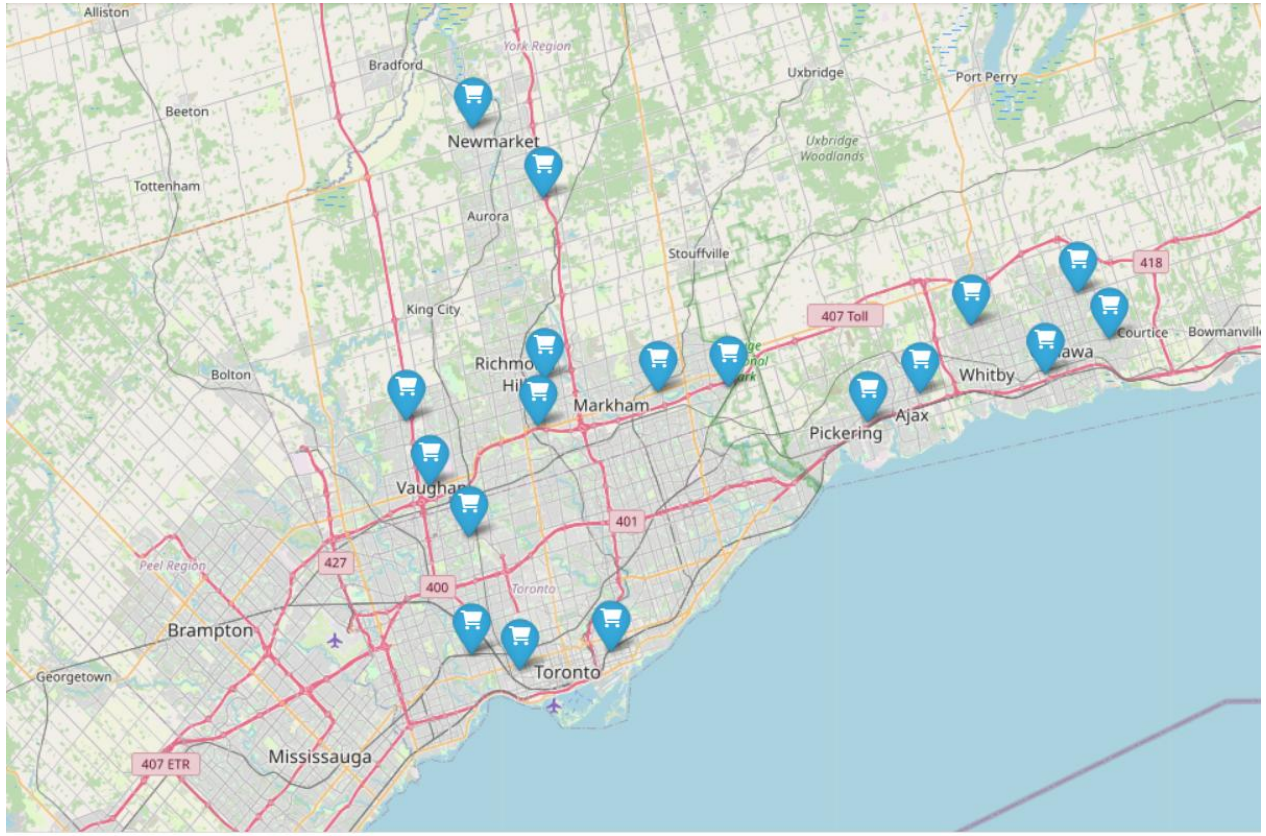




https://livebook.manning.com/book/optimization-algorithms/chapter-8/v-11/11

Permutation GA

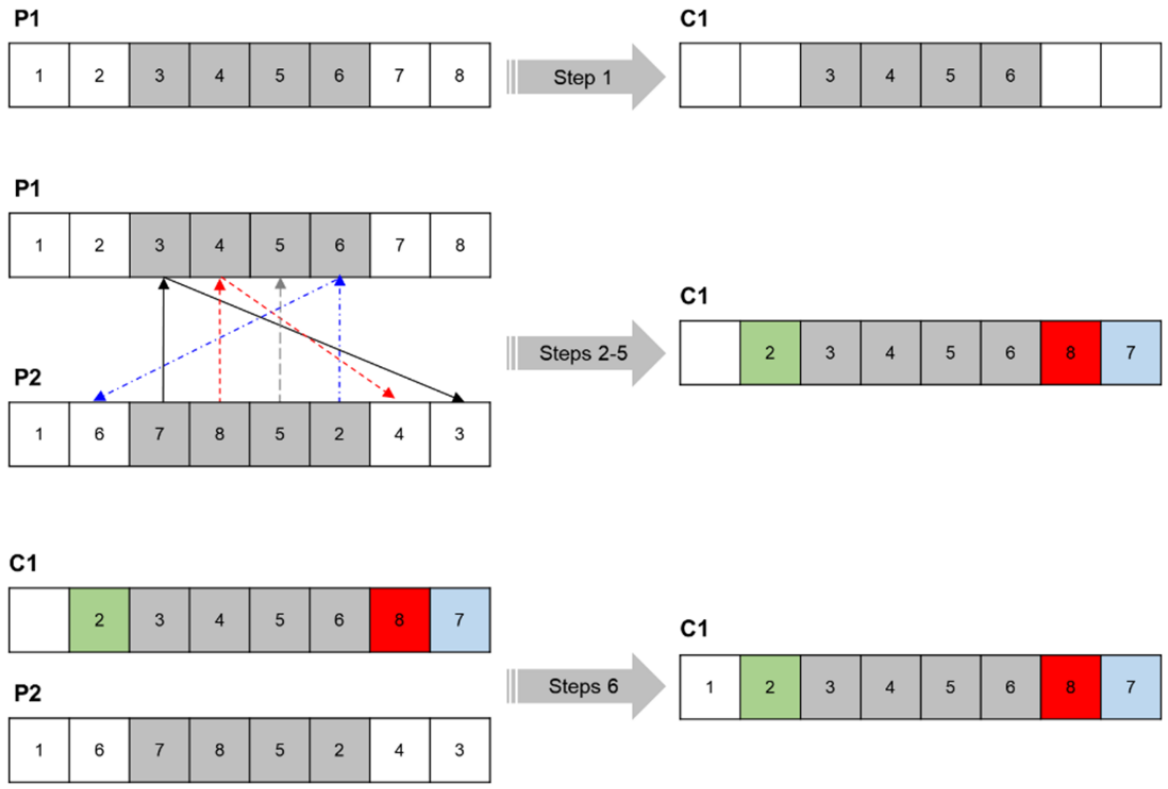




https://livebook.manning.com/book/optimization-algorithms/chapter-8/v-11/11

Partially Mapped Crossover (PMX)

1. Initialize: Choose two random crossover points and copy the segment between these two points from parent P1 to child C1 and from the second parent P2 to the second child C2.



2. For each element in the copied segment of C1:
3. Find the corresponding element in the P2's segment.
4. If the corresponding element is not already in the C1:
5. Replace the element in the C1 at the same position as in P2 with the corresponding element from P2
6. Fill the remaining positions in the offspring with the elements from the other parent, ensuring that no duplicates are introduced.
7. Repeat steps 2-6 for the second offspring, using the P1's segment as a reference.
8. Return C1 and C2

https://livebook.manning.com/book/optimization-algorithms/chapter-8/v-11/11

```
import random

def partially_mapped_crossover(parent1, parent2):
    n = len(parent1)
    point1, point2 = sorted(random.sample(range(n), 2))
    child1 = [None] * n
    child2 = [None] * n
    child1[point1:point2+1] = parent1[point1:point2+1]
    child2[point1:point2+1] = parent2[point1:point2+1]

    for i in range(n):
        if child1[i] is None:
            value = parent2[i]
            while value in child1:
                value = parent2[parent1.index(value)]
            child1[i] = value

        if child2[i] is None:
            value = parent1[i]
            while value in child2:
                value = parent1[parent2.index(value)]
            child2[i] = value

    return child1, child2
```

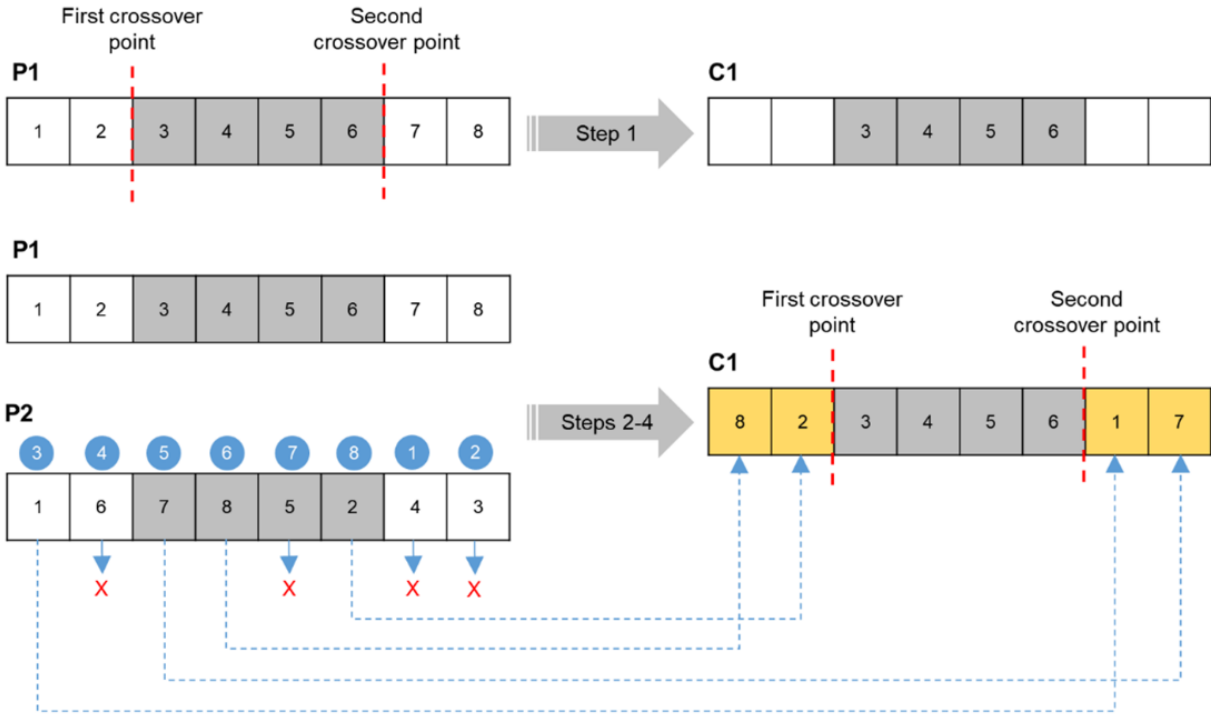
Other crossovers for permutation GAs

Edge Crossover (EC)

Order 1 Crossover (OX1)

Cycle cross-over:

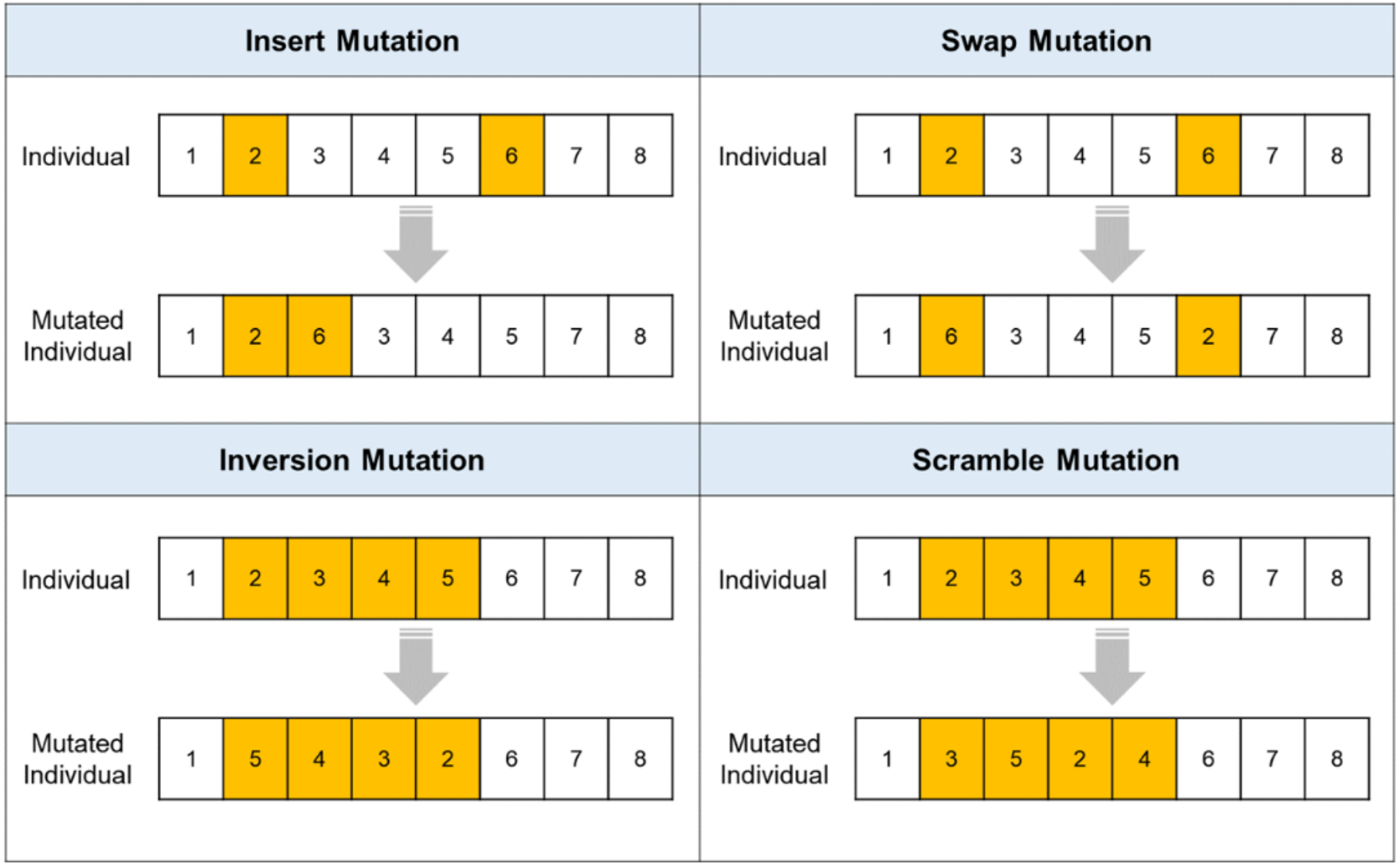
Order 1 Crossover (OX1)



We choose cross-over methods so that each number appears only once

https://livebook.manning.com/book/optimization-algorithms/chapter-8/v-11/11

Mutations for permutation GAs



https://livebook.manning.com/book/optimization-algorithms/chapter-8/v-11/11

Repair operator

If the result of mutation or crossover is not a legitimate solution, we can use **repair** operator

Wait, there is more!

- Genetic Algorithm (GA): is a search algorithm that mimics natural evolution, where each individual is a candidate solution encoded as a binary or real-valued vector.
- Differential Evolution (DE): is an algorithm that uses real-valued vectors as individuals and generates new solutions by adding weighted differences between pairs of existing solutions. It is similar to GA, differing in the reproduction mechanism used.
- Genetic Programming (GP): is a special case of GA, where each individual is a computer program encoded as a variable-length tree. This tree structure is used to represent functions and operators, such as if-else statements and mathematical operations .
- Evolution Strategy (ES): is an algorithm that uses real-valued vectors as individuals and adapts mutation and recombination parameters during evolution (i.e. the evolution of evolution). Plus-selection, comma-selection, greedy selection and distance-based selection are used as selection methods in ES.

Wait, there is more!

- Cultural Algorithm (CA): incorporate social learning from a shared belief space into the traditional population-based evolution process. The concept behind CA is to model the evolution of a population's culture and how it influences the genetic and phenotypic evolution of individuals

- Co-evolution (CoE): is based on the process of reciprocal evolutionary change that occurs between interacting populations, where each represents a given species, together optimizing coupled objectives.

Overall:

- Scalability is not as good as other optimization algorithms, hence long computing time
- Difficulty to adequately fine-tune hyperparameters such as mutation and selection can lead to non-convergence or useless results
- Doesn't guarantee finding the global optimum, however, this is the trade-off of all meta-heuristic methods
- An expensive and complex fitness function can also lead to long computation duration

Great method when we can represent solution as a string and design mutation and crossover operators .

We can even tune neural networks using GA

https://towardsdatascience.com/genetic-algorithm-6aefd897f1ac

Universality matters!

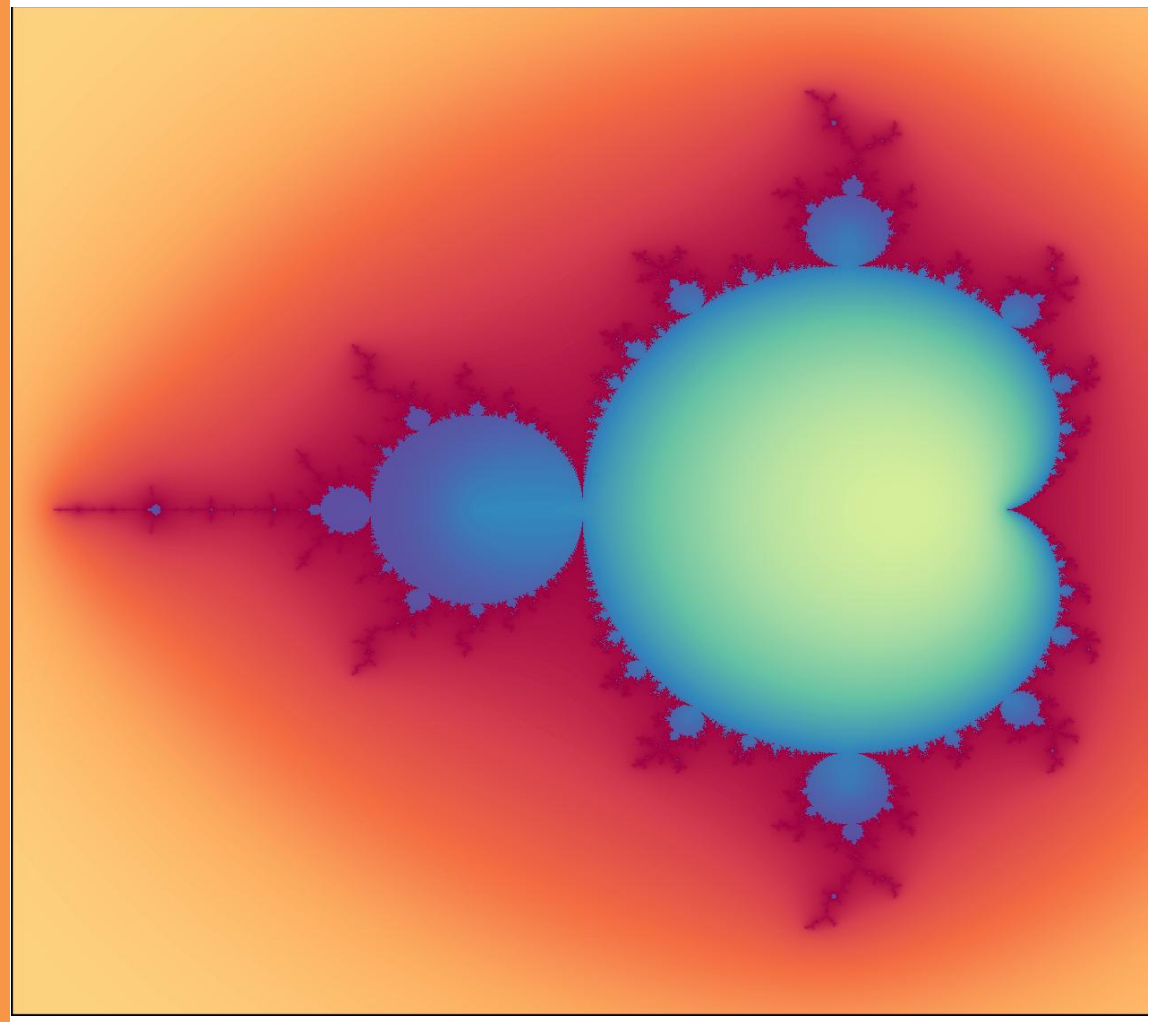
Computer vision: Deep convolutional networks

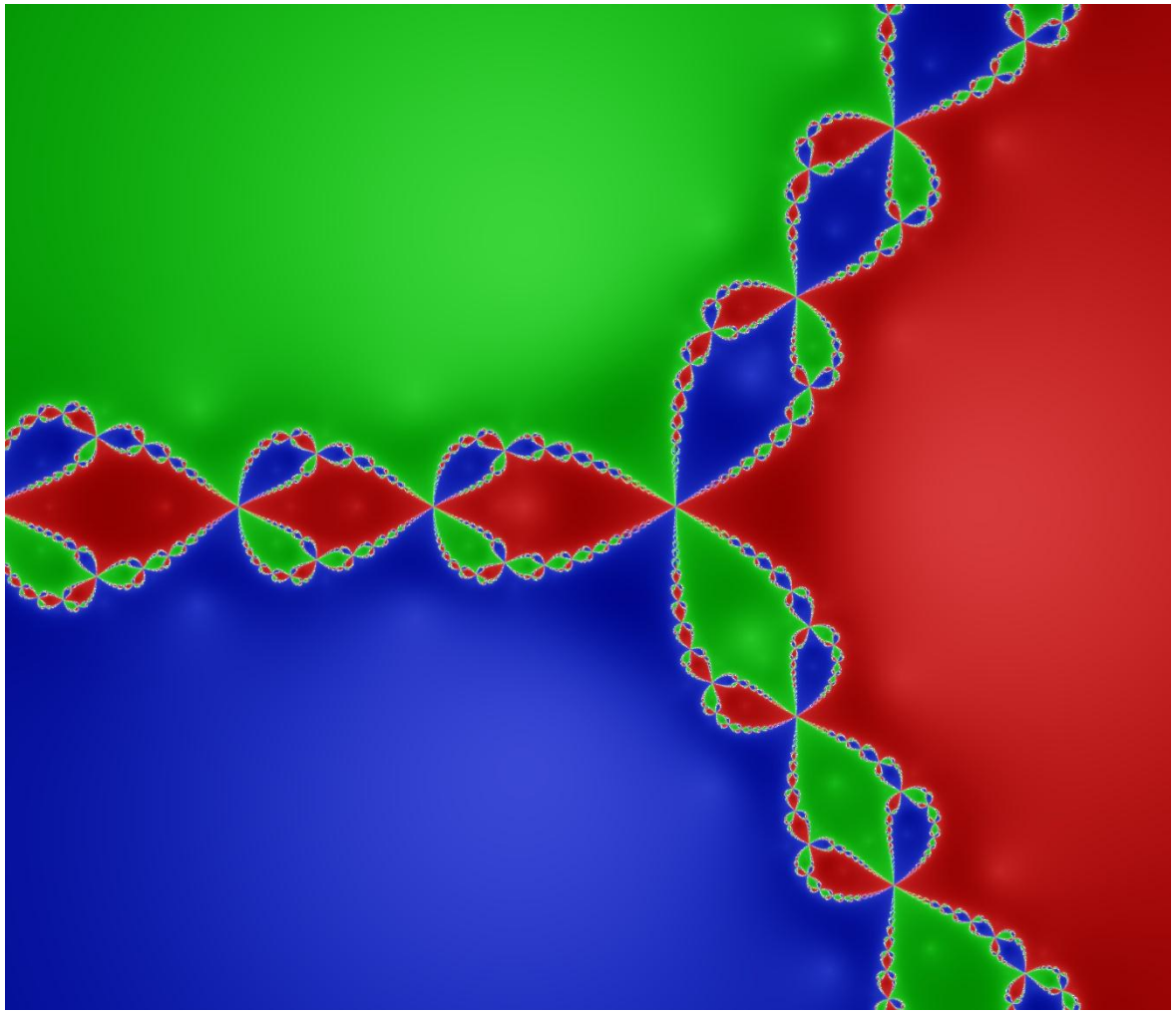
Radiation effects in materials

Medical imaging and radiology

Autonomous driving and robotics

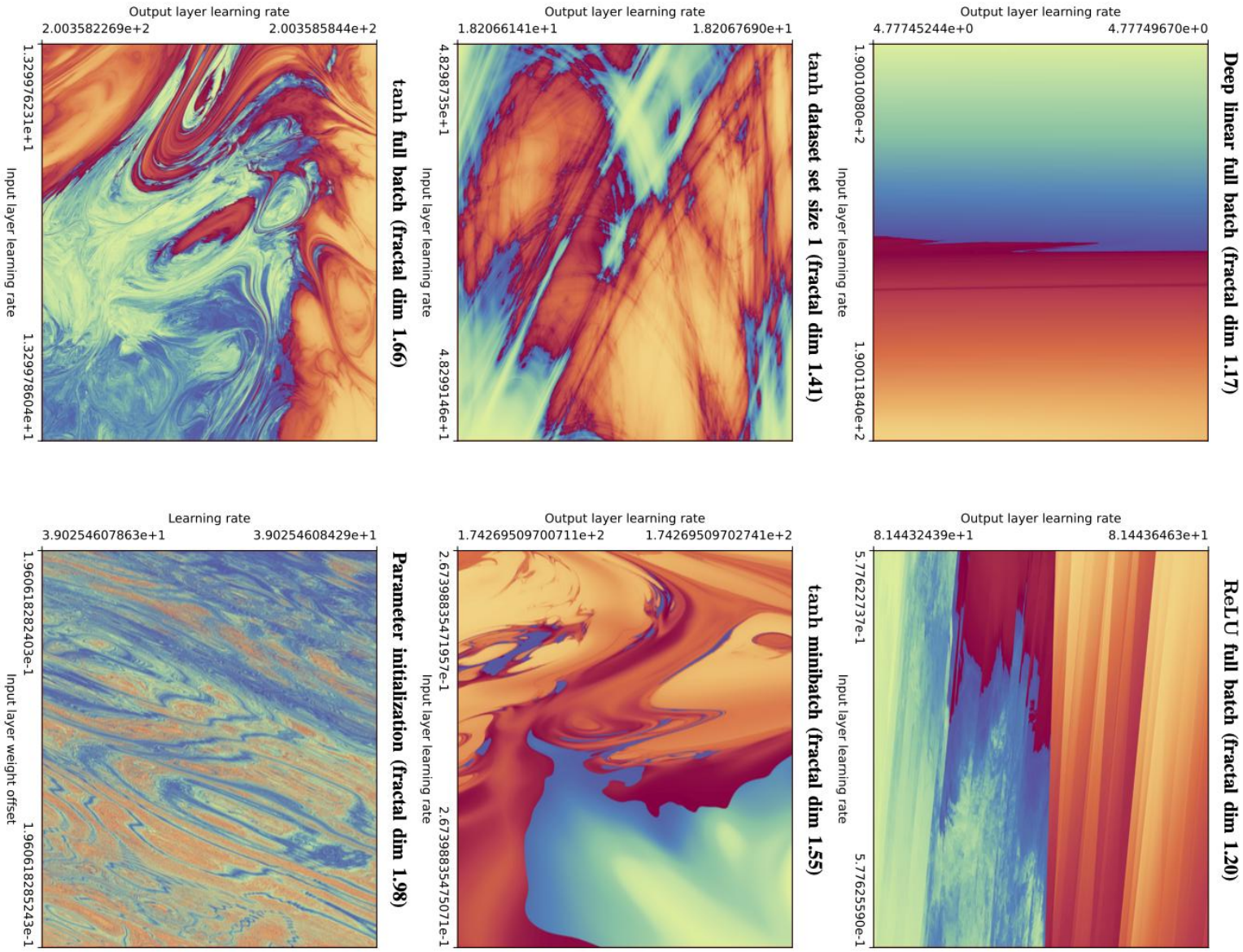
Cool science





https://sohl-dickstein.github.io/2024/02/12/fractal.html

Cool science



https://sohl-dickstein.github.io/2024/02/12/fractal.html