Neuroevolution

Nuno Antunes Ribeiro

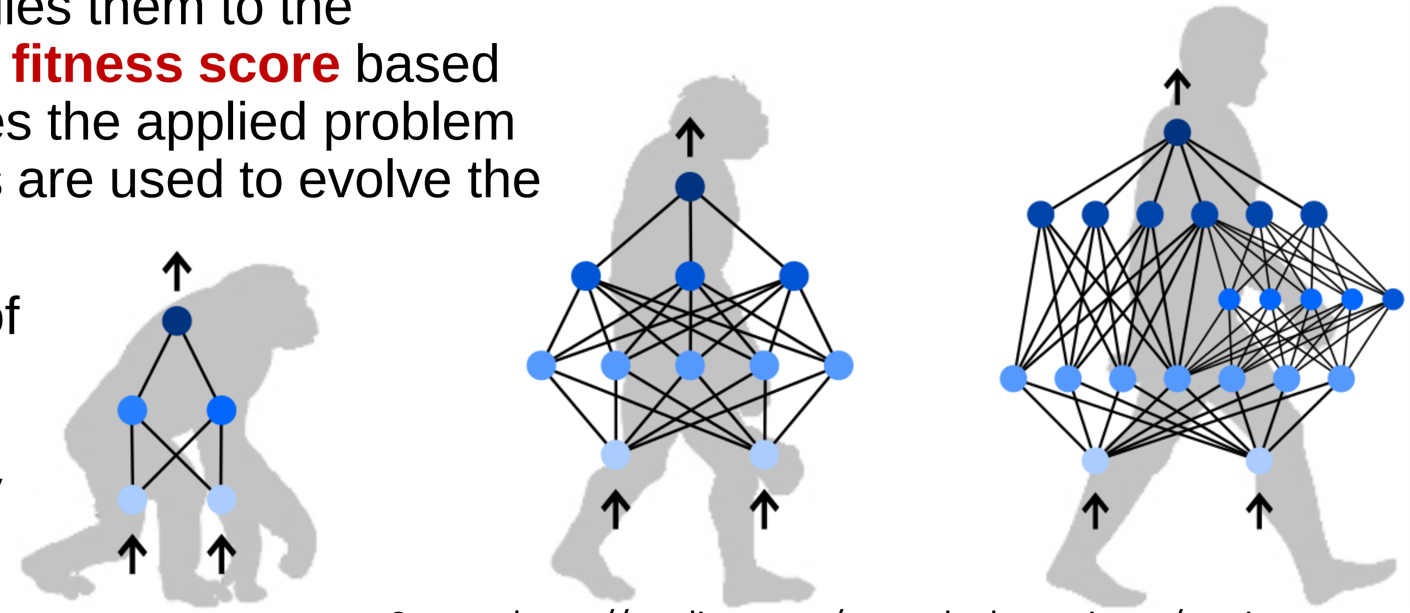
Assistant Professor

Engineering Systems and Design

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Neuroevolution

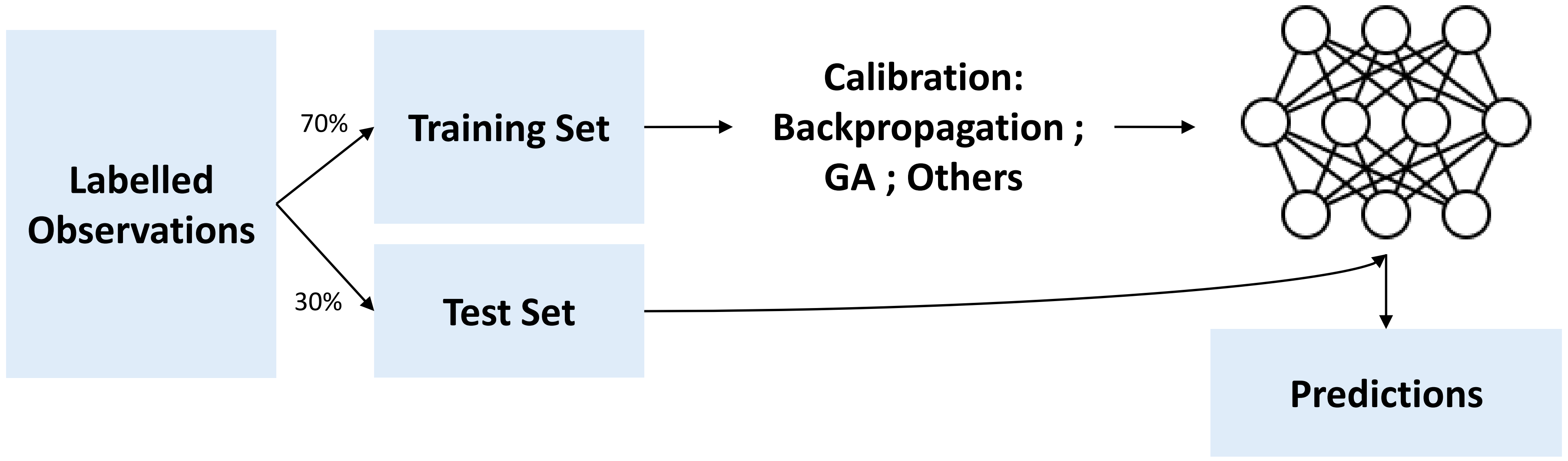
- **Neuroevolution** is a **machine learning technique** that improves the rough abstractions of brains that are artificial neural networks (abbr. ANNs) by generating increasingly better topologies, weights and hyperparameters by the means of evolutionary algorithms.
- The neuroevolution process initializes with a set of random **initial neural networks**. Then it applies them to the **problem environment** after which a **fitness score** based on how well the neural network solves the applied problem is estimated. Evolutionary algorithms are used to evolve the neural networks.
- The fitness score can be a function of the accuracy achieved in an image recognition task, how close a robotic arm moved to the intended trajectory or how well an agent performed in a virtual environment such as a video game.



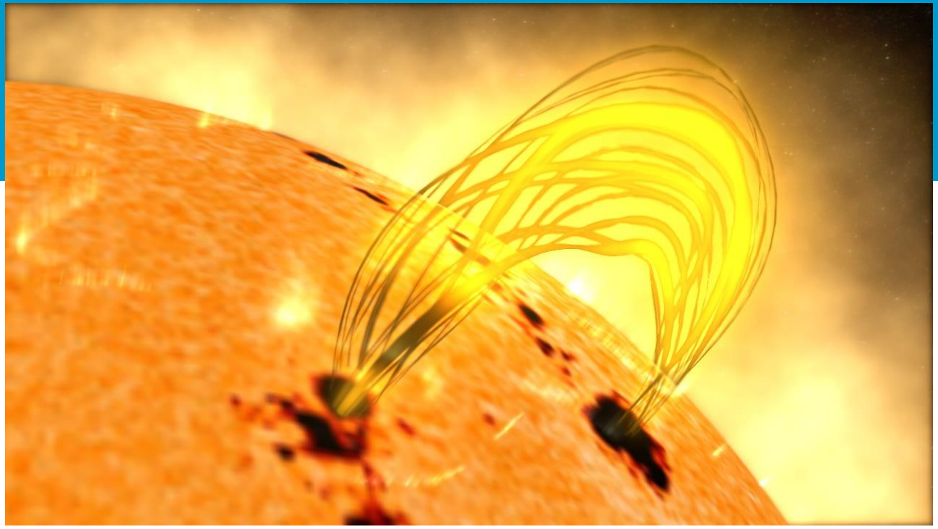
Source: https://medium.com/towards-data-science/a-primer-on-the-fundamental-concepts-of-neuroevolution-9068f532f7f7

Supervised Learning

- Supervised learning relies on labelled input and output training data. You use the training data to infer the parameters and hyperparameters and verify with the test data whether the approximated function performs well on unseen data.
- In supervised learning the decisions are independent of each other, so labels are given to each decision.
- Applications: Image, Text and Speech recognition ; Spam Detection ; Risk Evaluation; Fraud Detection; Forecasts and Predictive Analytics; etc.



Supervised Learning



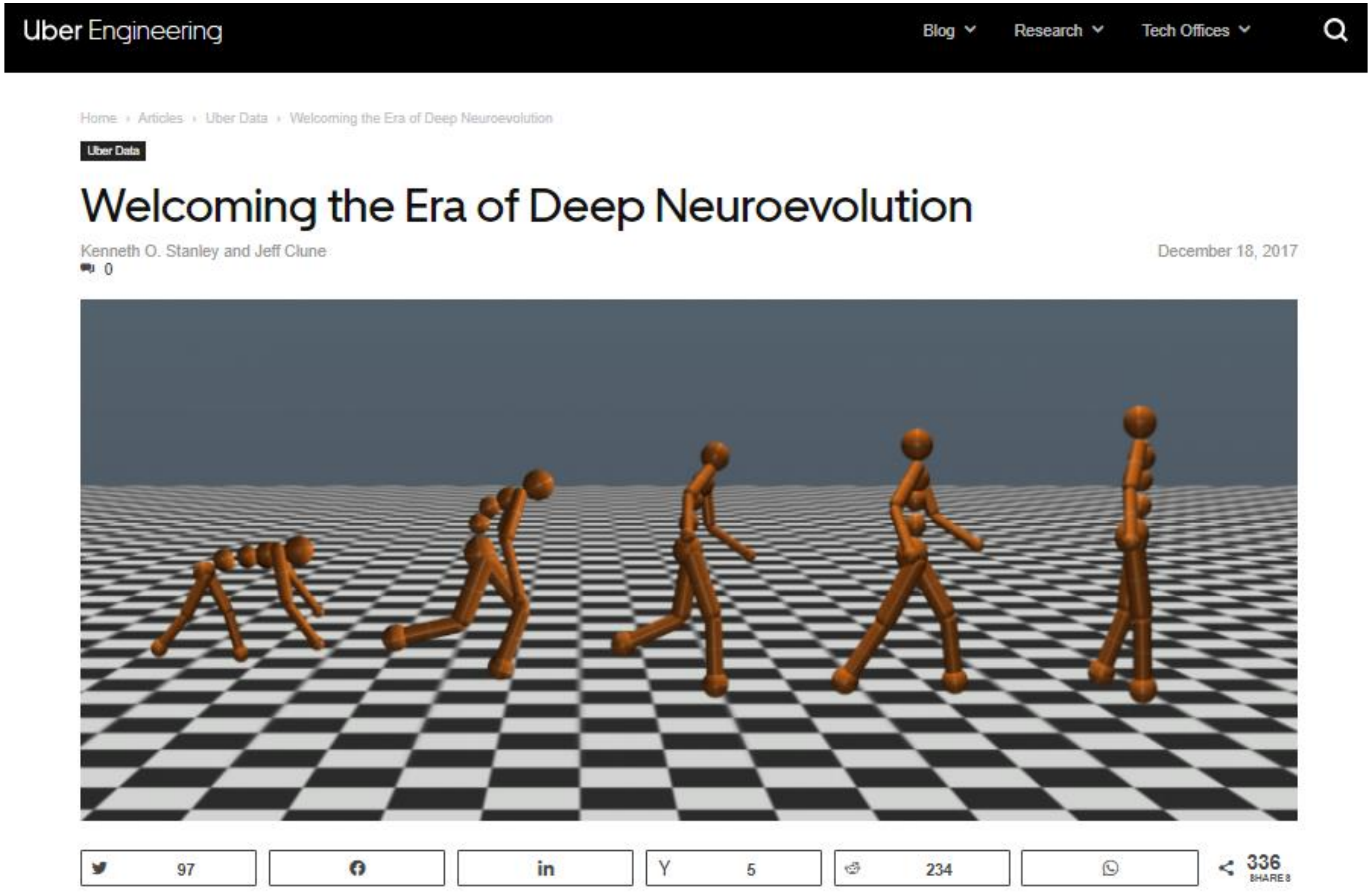
- **Example:** Predicting Sunspot Cycle
- Observations:
  - Results have shown that neuroevolution performs slightly better than gradient-based algorithms;
  - However, the improvements obtained from using neuroevolution are minor
  - Also, neuroevolution takes considerably longer to calibrate the NN than gradient-based algorithms.

Genetic Algorithm Results		Backpropagation	
Validation	**Testing**	**Validation**	**Testing**
595	**613**	602	615
666	674	720	**661**
632	659	739	**601**
651	**700**	1896	734
646	**626**	1876	636
656	705	688	**657**
621	**670**	649	693
625	**677**	651	689
595	**607**	660	662
624	576	634	**574**

Supervised Learning

- **Advantages of using neuroevolution**:
  - First, in neuroevolution we only need to use the forward pass. The backwards pass (backpropagation) is usually the most time consuming.
  - Second, gradient-based algorithms can get stuck in local minima whereas evolutionary algorithms tend to avoid this problem
  - Third, neuroevolution allow us to not only optimize the weights of a NN, but also the **NN architecture**
- **Disadvantages of using neuroevolution**:
  - Often gradient-based algorithms are very effective when enough gradient information is available. As a consequence, it is very rare that you will use genetic algorithms in a supervised context.
  - However, if you are working in a **Reinforcement Learning** context, then neuroevolution provides a viable alternative!

Is neuroevolution the future of deep learning?



Home › Articles › Uber Data › Welcoming the Era of Deep Neuroevolution

Uber Data

Welcoming the Era of Deep Neuroevolution

Kenneth O. Stanley and Jeff Clune

0

December 18, 2017

By Kenneth O. Stanley, Jeff Clune

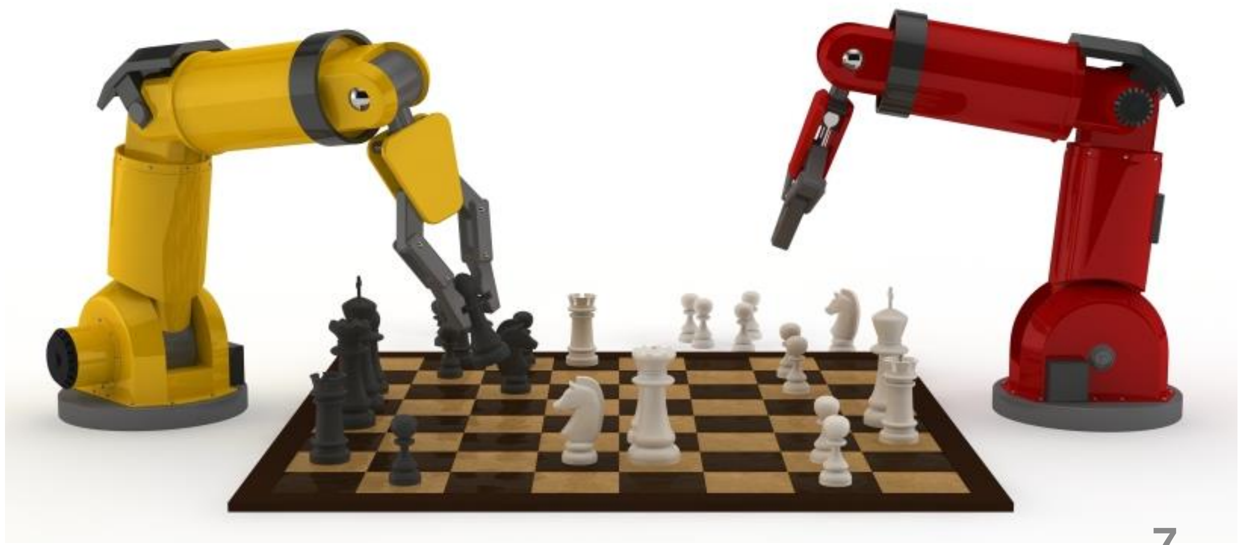
https://eng.uber.com/deep-neuroevolution/

"(GA) can train deep convolutional networks with over 4 million parameters to play Atari games from pixels, and on many games outperforms modern deep reinforcement learning (RL) algorithms (e.g. DQN and A3C)"

"This result is surprising both because GAs, which are not gradient-based, were not expected to scale well to such large parameter spaces and also because matching or outperforming the state-of-the-art in RL using GAs was not thought to be possible."

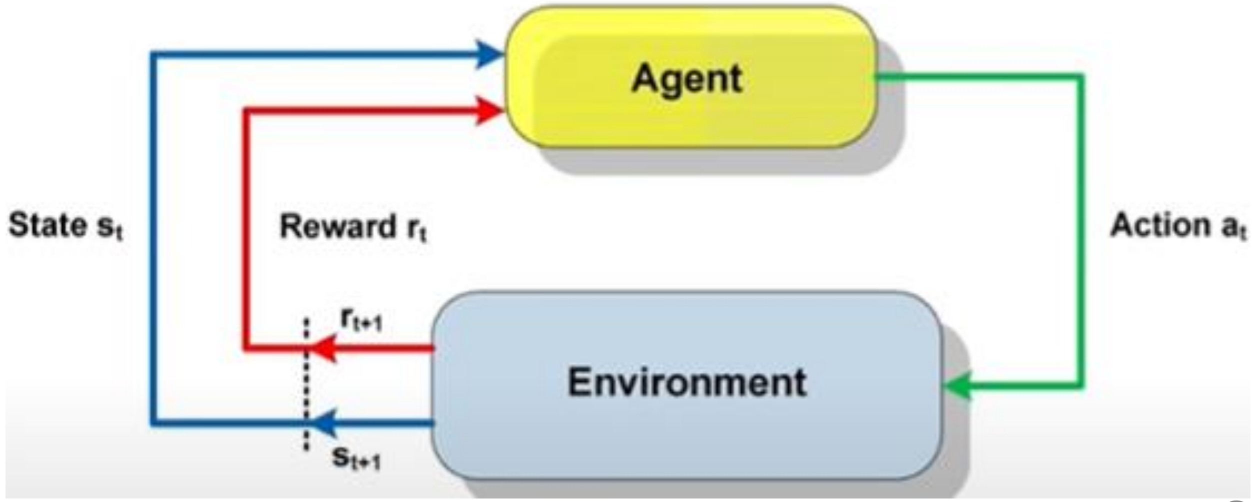
Reinforcement Learning

- In supervised learning we provide inputs to a neural network, and we know the output that the NN should produce. We can easily apply backpropagation to compute gradients, and train our NN to produce the expected outputs.
- Now, imagine you want to train a **neural network to play a game**.
- What you would do in a supervised setting is having a good gamer, playing the game for a couple of hours. Then you would create a dataset where you would log all the actions made by the gamer in all frames of the game. You could then use this dataset to calibrate your NN using backpropagation.
- However, there are 2 main downsides of using this approach:
  - The generation of the dataset may be difficult, requiring a **large amount of data** – time and space consuming
  - Your agent will eventually end-up by mimicking the gamer - in theory the **NN will never be able to outperform the gamer** itself.



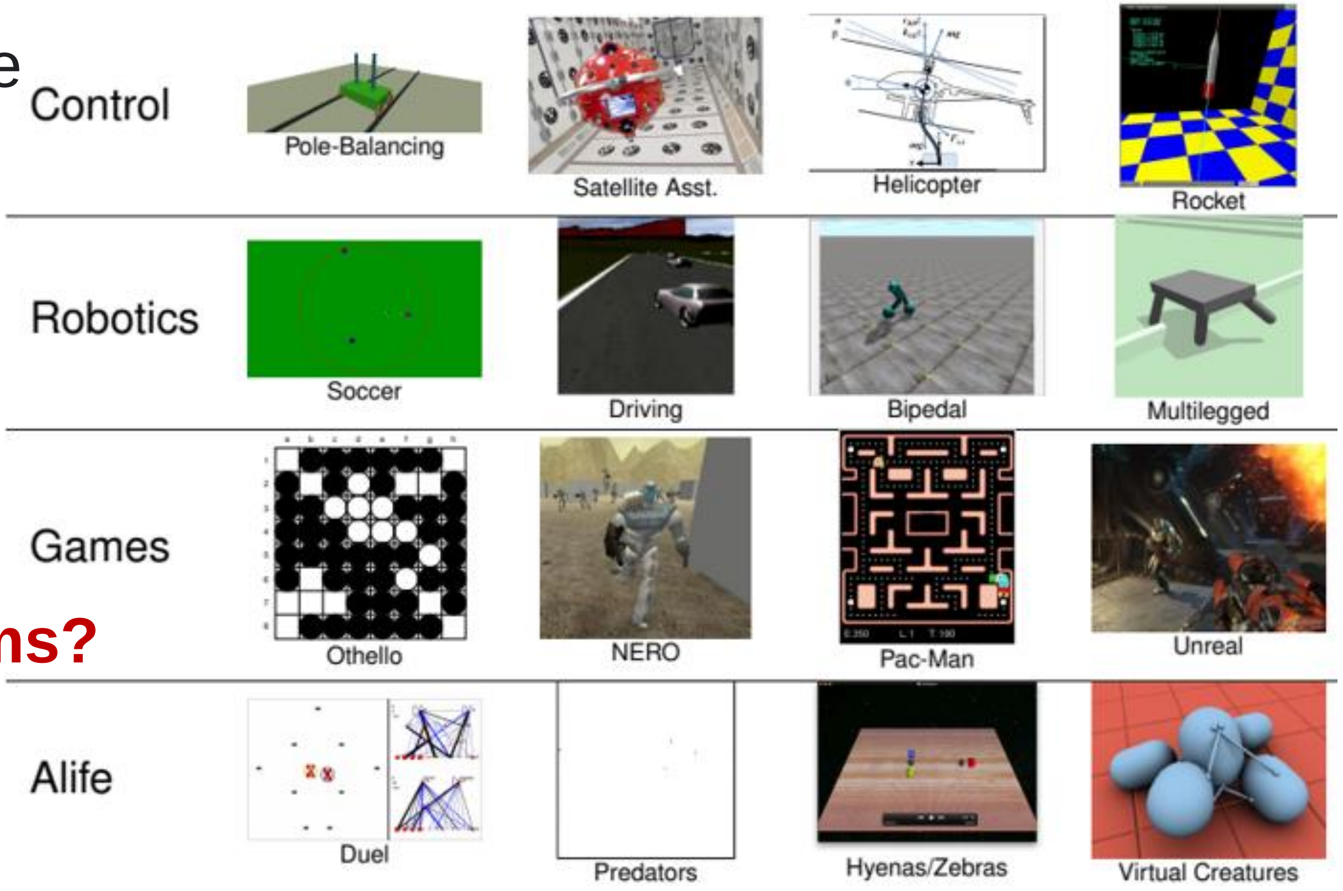
Reinforcement Learning

- Reinforcement Learning (RL) aims to learn from “online” interaction with the environment – similarly to how humans learn (i.e. trial and error)
- In order to evaluate taken actions, favorable choices are rewarded. The actor is called agent and aims to maximize the accumulated rewards
- Essentially, the agent (encoded as a NN) plays the game several times. The reward obtained at the end of each game is given by the score of the game (e.g. did the agent win the game? how long did it take to be eliminated? what was the score? etc.).
- The agent learns from experience every time it plays the game (by calibrating the weights and bias of the NN).



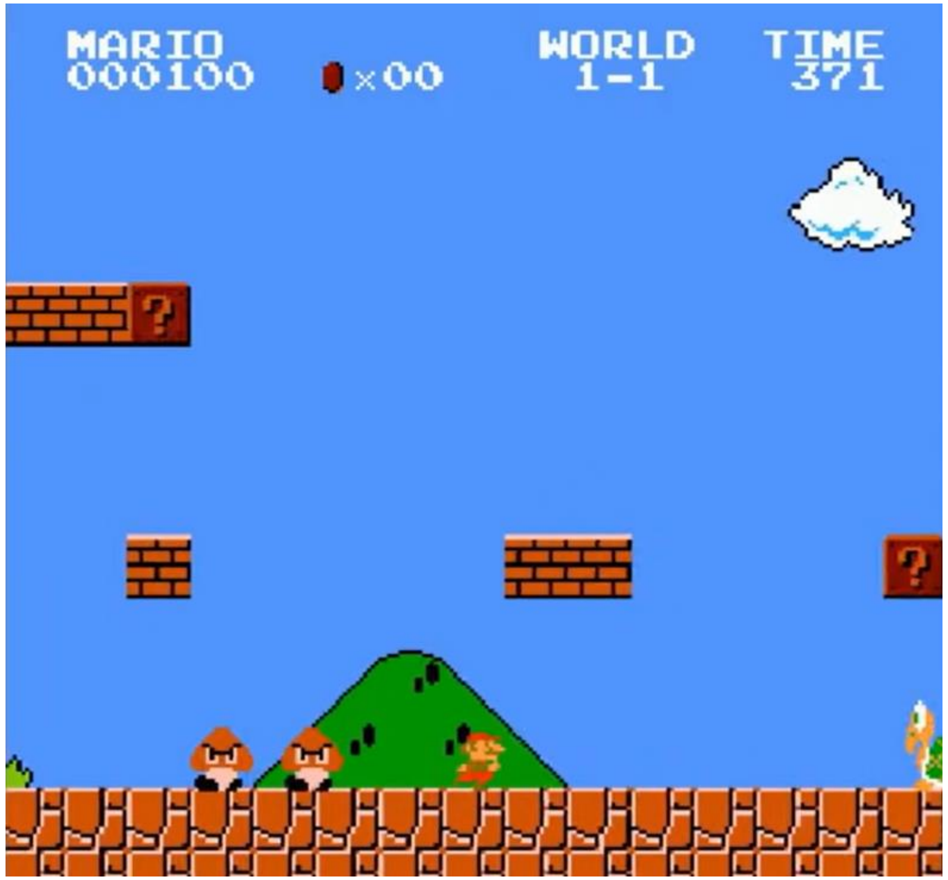
Reinforcement Learning Applications

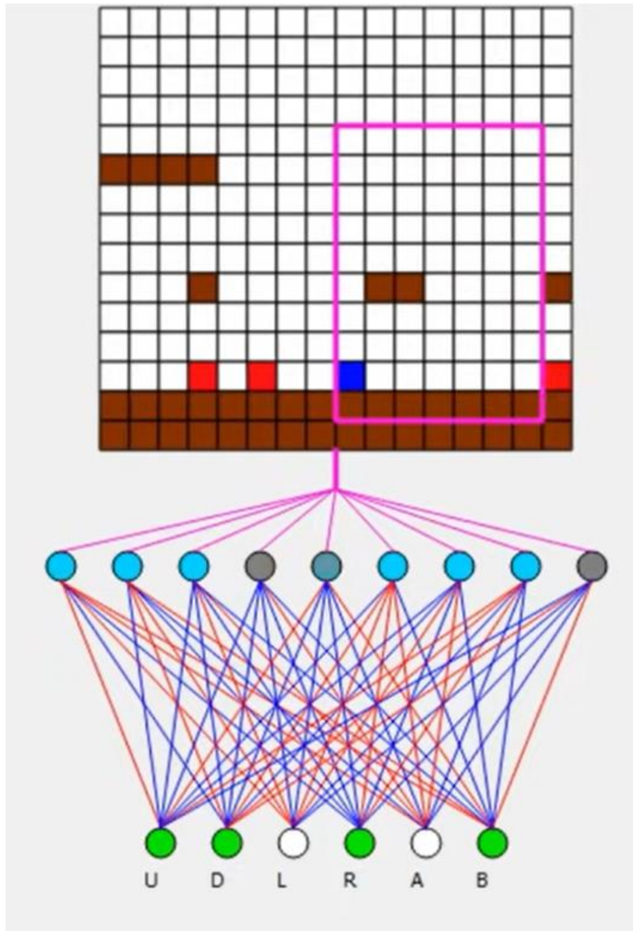
- Different methods are used to solve Reinforcement Learning:
  - Q-Learning,
  - SARSA,
  - DQN,
  - DDPG
  - Etc.
- **What if we use Genetic Algorithms?**



Conventional Neuroevolution

- Given a specific neural network topology we aim to optimize the weights and bias.
- **Let's Play Super Mario**





https://www.youtube.com/watch?v=CI3FRsSAa_U&t=126s

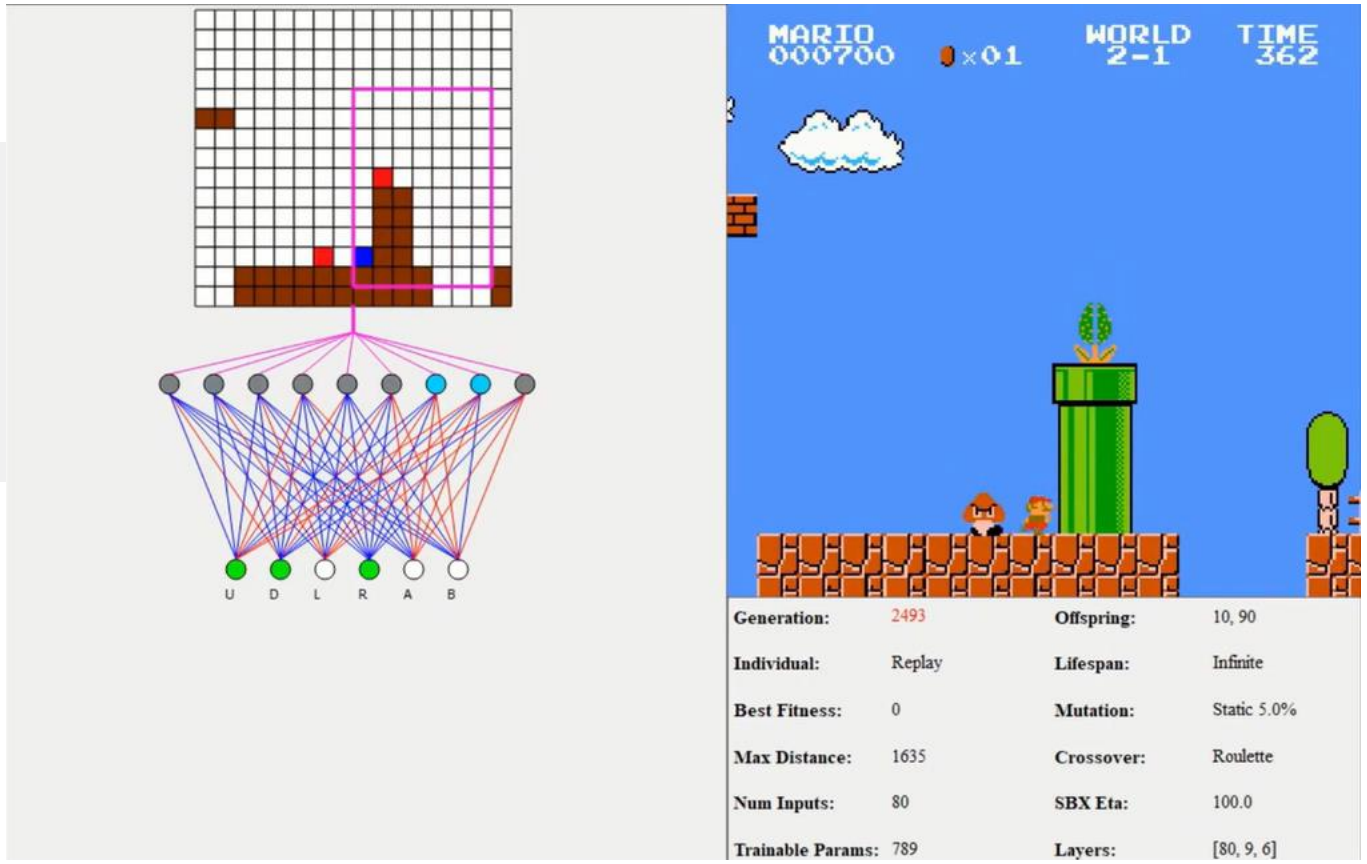
Conventional Neuroevolution

- Given a specific neural network topology we aim to optimize the weights and bias.
- Let’s Play Super Mario (**3 weeks of calibration**)

Fitness Function

$$D_{bonus} = \text{distance}^{1.8}$$
$$T_{penalty} = \text{frames}^{1.5}$$
$$E_{bonus} = \min(\max(\text{distance} - 50, 0), 1) * 2500$$
$$W_{bonus} = \text{did\_win} * 10^{6}$$

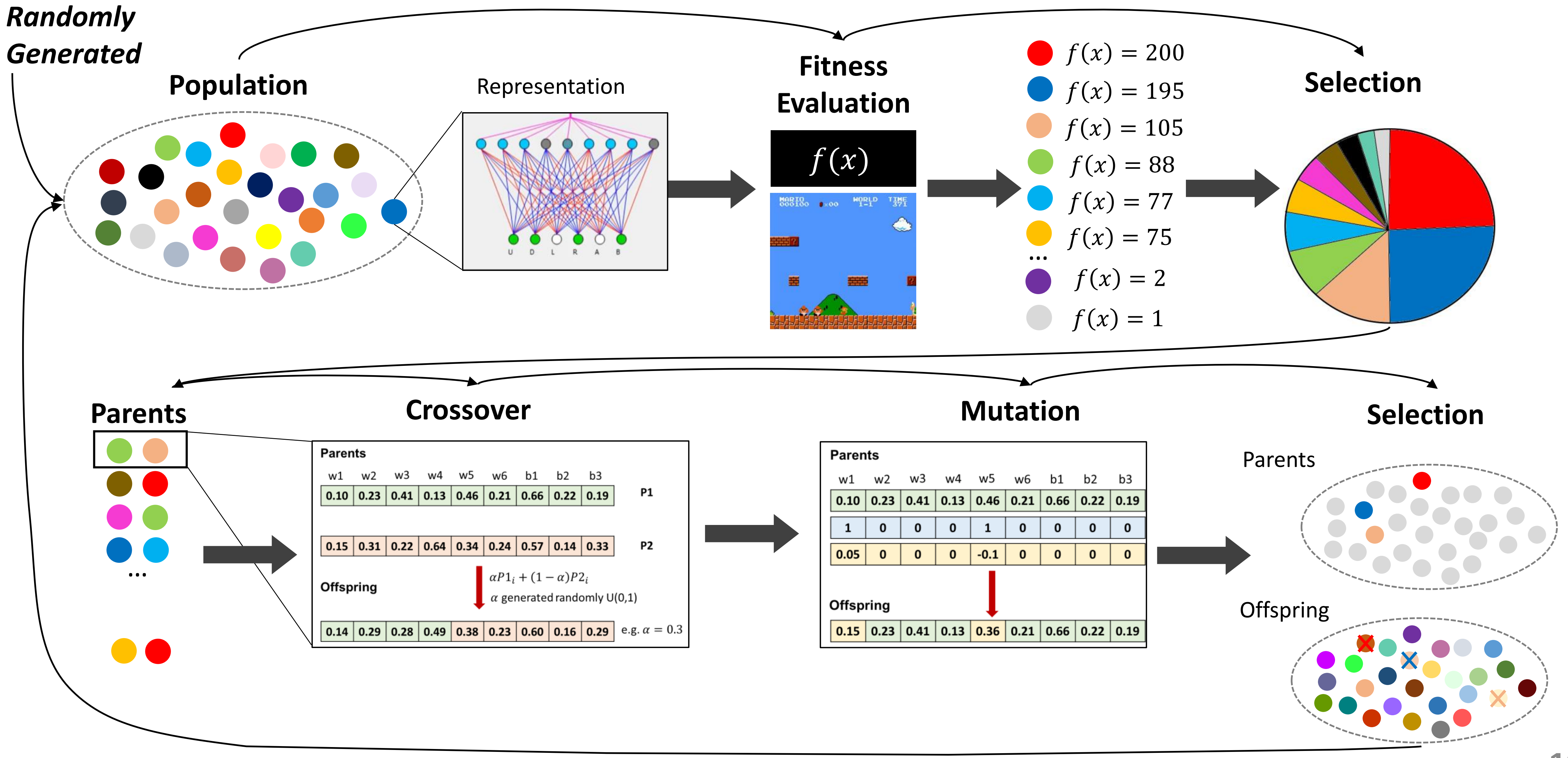
$$\text{Fitness} = \max(D_{bonus} - T_{penalty} + E_{bonus} + W_{bonus}, 10^{-5})$$



Generation:	2493	Offspring:	10, 90
Individual:	Replay	Lifespan:	Infinite
Best Fitness:	0	Mutation:	Static 5.0%
Max Distance:	1635	Crossover:	Roulette
Num Inputs:	80	SBX Eta:	100.0
Trainable Params:	789	Layers:	[80, 9, 6]

https://www.youtube.com/watch?v=CI3FRsSAa_U&t=126s

Conventional Neuroevolution



Crossover

Parents

	w1	w2	w3	w4	w5	w6	b1	b2	b3
P1	0.10	0.23	0.41	0.13	0.46	0.21	0.66	0.22	0.19
P2	0.15	0.31	0.22	0.64	0.34	0.24	0.57	0.14	0.33

$\alpha P1_i + (1-\alpha) P2_i$

α generated randomly U(0,1)

Offspring

0.14	0.29	0.28	0.49	0.38	0.23	0.60	0.16	0.29

e.g. $\alpha = 0.3$

Mutation

Parents

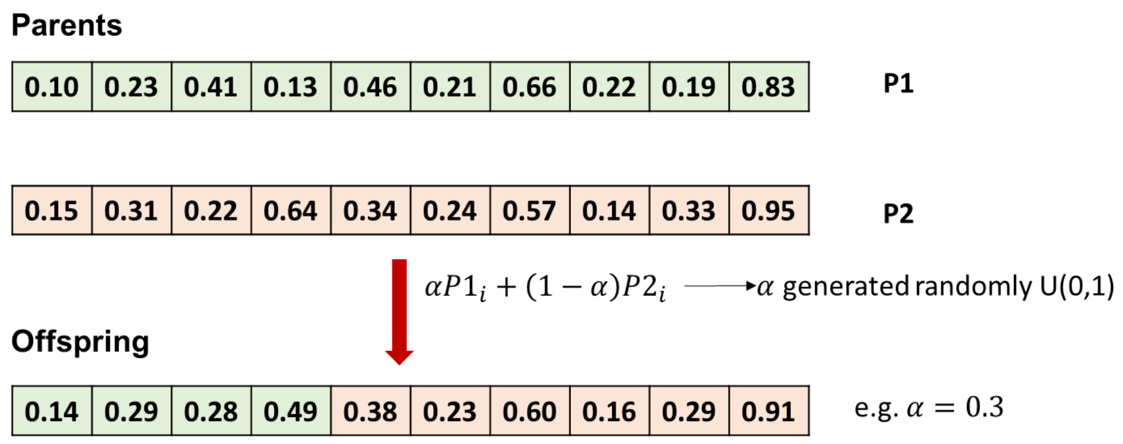
w1	w2	w3	w4	w5	w6	b1	b2	b3
0.10	0.23	0.41	0.13	0.46	0.21	0.66	0.22	0.19
1	0	0	0	1	0	0	0	0
0.05	0	0	0	-0.1	0	0	0	0

Offspring

0.15	0.23	0.41	0.13	0.36	0.21	0.66	0.22	0.19

Recall: EA for Continuous Optimization

- Randomized crossover operators for continuous problems might be very ineffective.
- Evolution Strategies and Differential Evolution are to Evolutionary Algorithms used to more effectively search for better solutions in a continuous search space
- **Typical crossover operator used in Genetic Algorithms**



Parents

0.10	**0.23**	**0.41**	**0.13**	**0.46**	**0.21**	**0.66**	**0.22**	**0.19**	**0.83**	**P1**
0.15	**0.31**	**0.22**	**0.64**	**0.34**	**0.24**	**0.57**	**0.14**	**0.33**	**0.95**	**P2**

$\alpha P1_i + (1-\alpha)P2_i \longrightarrow \alpha$ generated randomly U(0,1)

Offspring

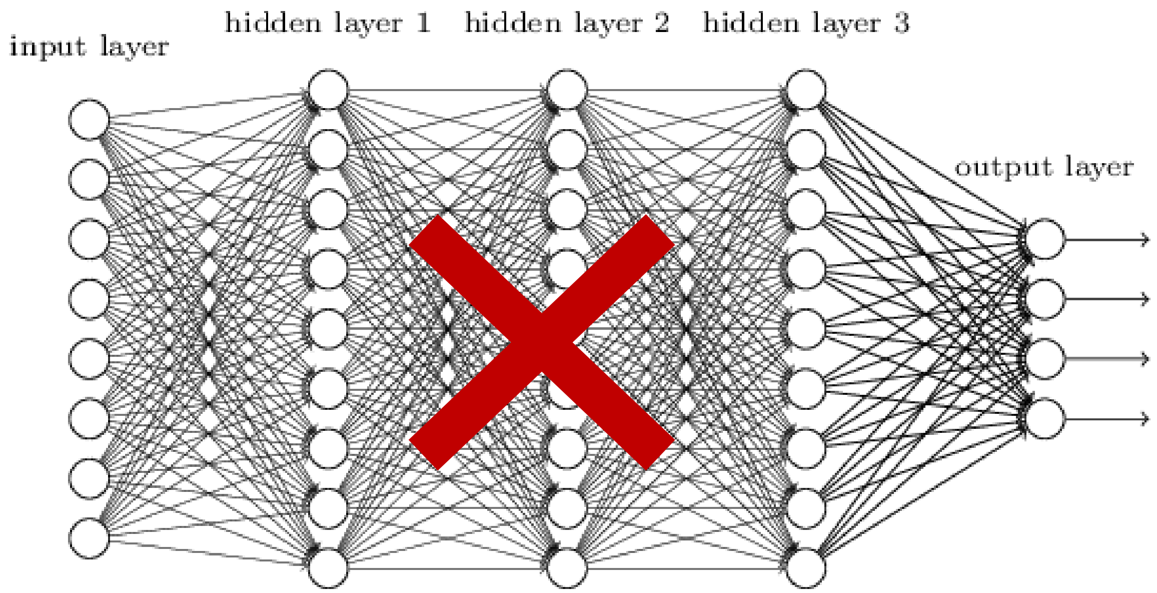
0.14	**0.29**	**0.28**	**0.49**	**0.38**	**0.23**	**0.60**	**0.16**	**0.29**	**0.91**

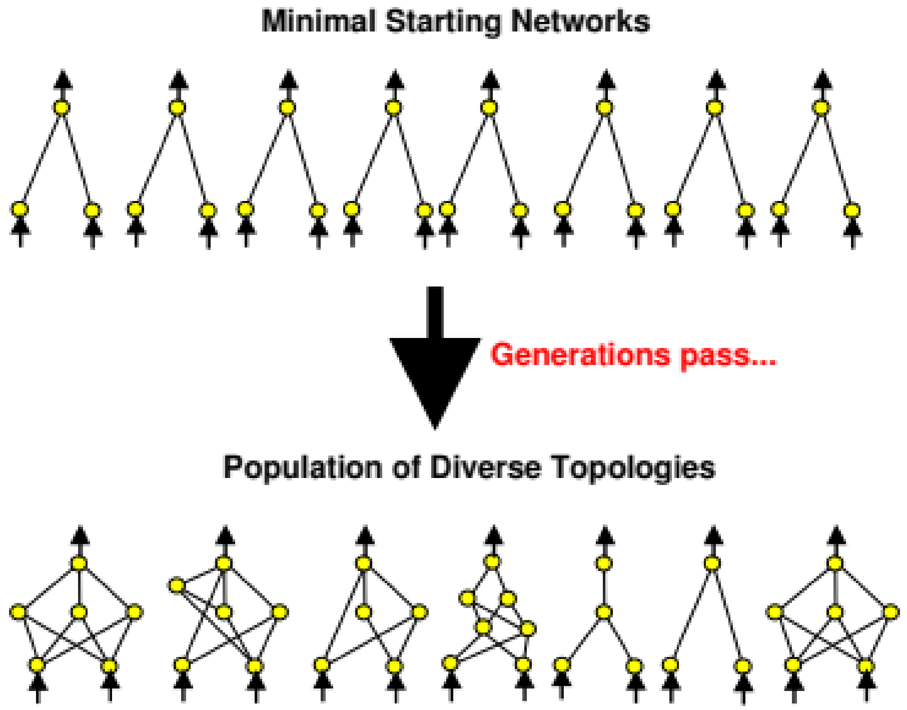
e.g. $\alpha = 0.3$

Evolution Strategies	Differential Evolution
~1964	~1997
Vector of **real** variables	Vector of **real** variables
(Recombination) + Mutation	Mutation + Crossover
Very effective on optimization problems with **continuous search space**	**Simple and fast**
Only applicable to problems with **continuous** search spaces	Convergence to **local optima** – **only real** domains ca be considered
Continuous Optimization Reinforcement Learning	Continuous Optimization **Control problems that require fast computation**

Neuroevolution of Augmenting Topologies (NEAT)

- Instead of relying on a fixed **topology** for a neural network, why not allow it to **evolve** through a genetic algorithm?
- Why Complexification?
  - Complexification keeps the search tractable (i.e. start simple, add more sophistication)
  - Incremental construction of intelligent agents





Key Questions in NEAT

- Is there a **genetic encoding** representation that allows diferente topologies to crossover in a meaningful way?
- How can **topological innovation** that **needs a few generations to be optimized** be protected so that it does not disappear from the population prematurely
- How **can topologies be minimized** throughout evolution without the need for a specially contrived fitness function that measures complexity?

Genetic Encoding

Innovations

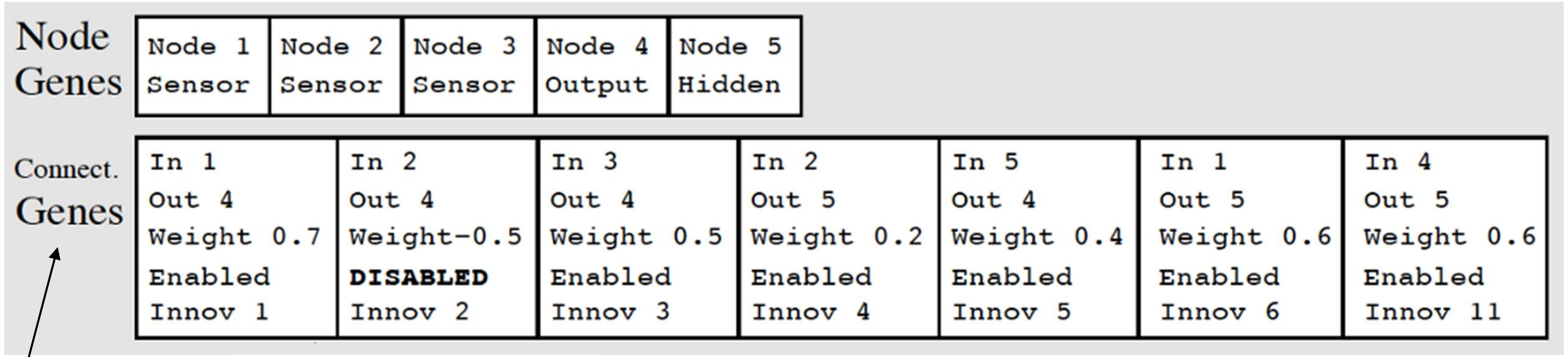
Speciation

Explicit Fitness Sharing

Minimal Structure Initialization

Genetic Encoding

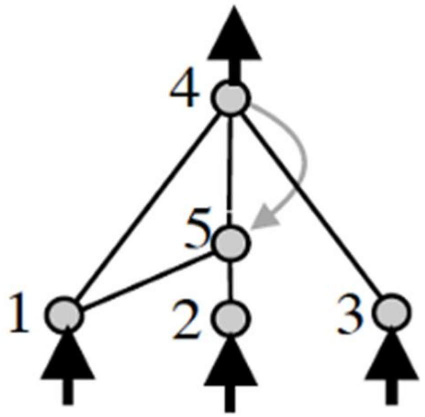
- Genetic Encoding in NEAT is a little bit more complex than a simple tree or binary encoding, however, it is still straightforward to understand.



Node Genes					
	Node 1 Sensor	Node 2 Sensor	Node 3 Sensor	Node 4 Output	Node 5 Hidden

Connect. Genes							
	In 1	In 2	In 3	In 2	In 5	In 1	In 4
	Out 4	Out 4	Out 4	Out 5	Out 4	Out 5	Out 5
	Weight 0.7	Weight-0.5	Weight 0.5	Weight 0.2	Weight 0.4	Weight 0.6	Weight 0.6
	Enabled	**DISABLED**	Enabled	Enabled	Enabled	Enabled	Enabled
	Innov 1	Innov 2	Innov 3	Innov 4	Innov 5	Innov 6	Innov 11

A NN link is called a gene in Neuroevolution

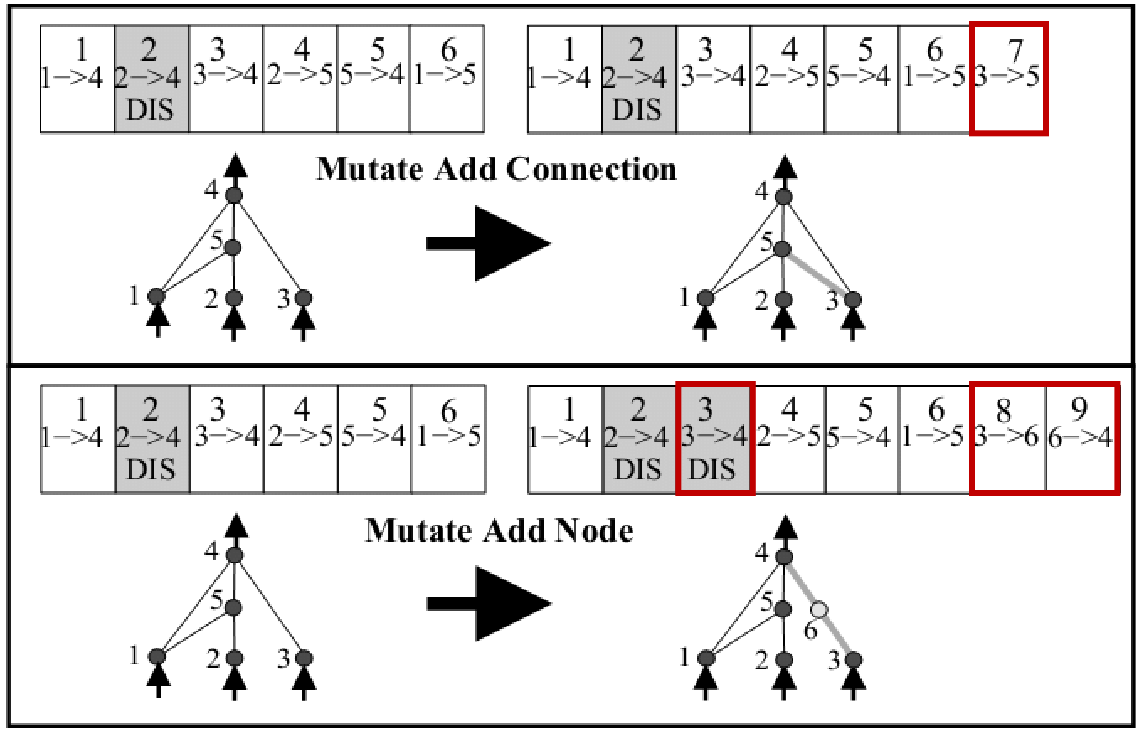


Simplified Encoding

1	**2**	**3**	**4**	**5**	**6**	**11**
1->4	**2->4**	**3->4**	**2->5**	**5->4**	**1->5**	**4->5**
	DISAB					

NEAT Mutations

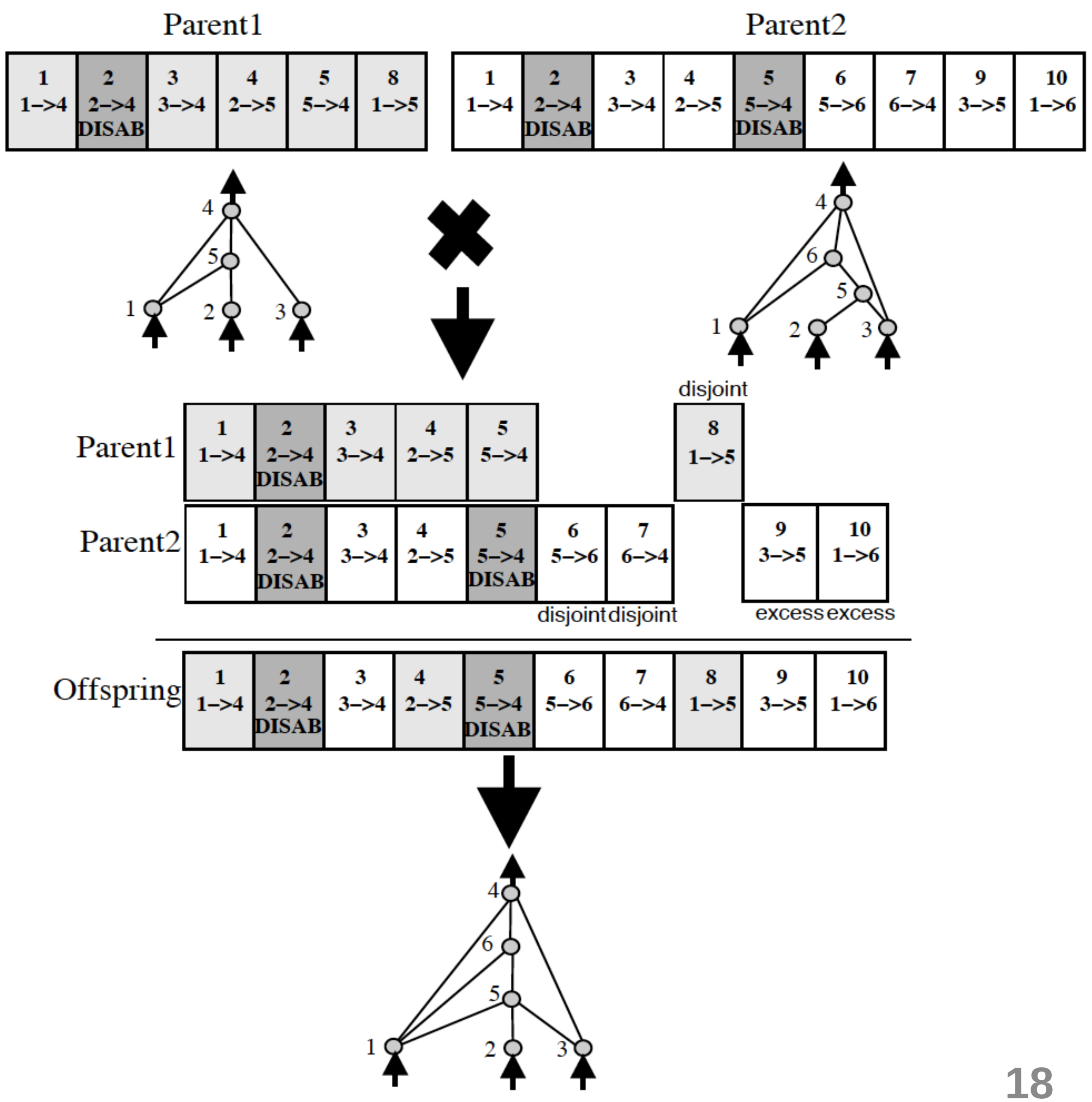
- Three main Types of Mutations
- **Randomly change a weight/bias** of the NN
- **Add a new gene (link) to the NN** - If a new connection is added between a start and end node, it is randomly assigned a weight.
- **Add a new node to the NN** - If a new node is added, it is placed between two nodes that are already connected. The previous connection is disabled. The previous start node is linked to the new node with the weight of the old connection and the new node is linked to the previous end node with a weight of 1. This was found to help mitigate issues with new structural additions.



Note: Input and output nodes are not evolved in the node gene list. Hidden nodes can be added or removed.

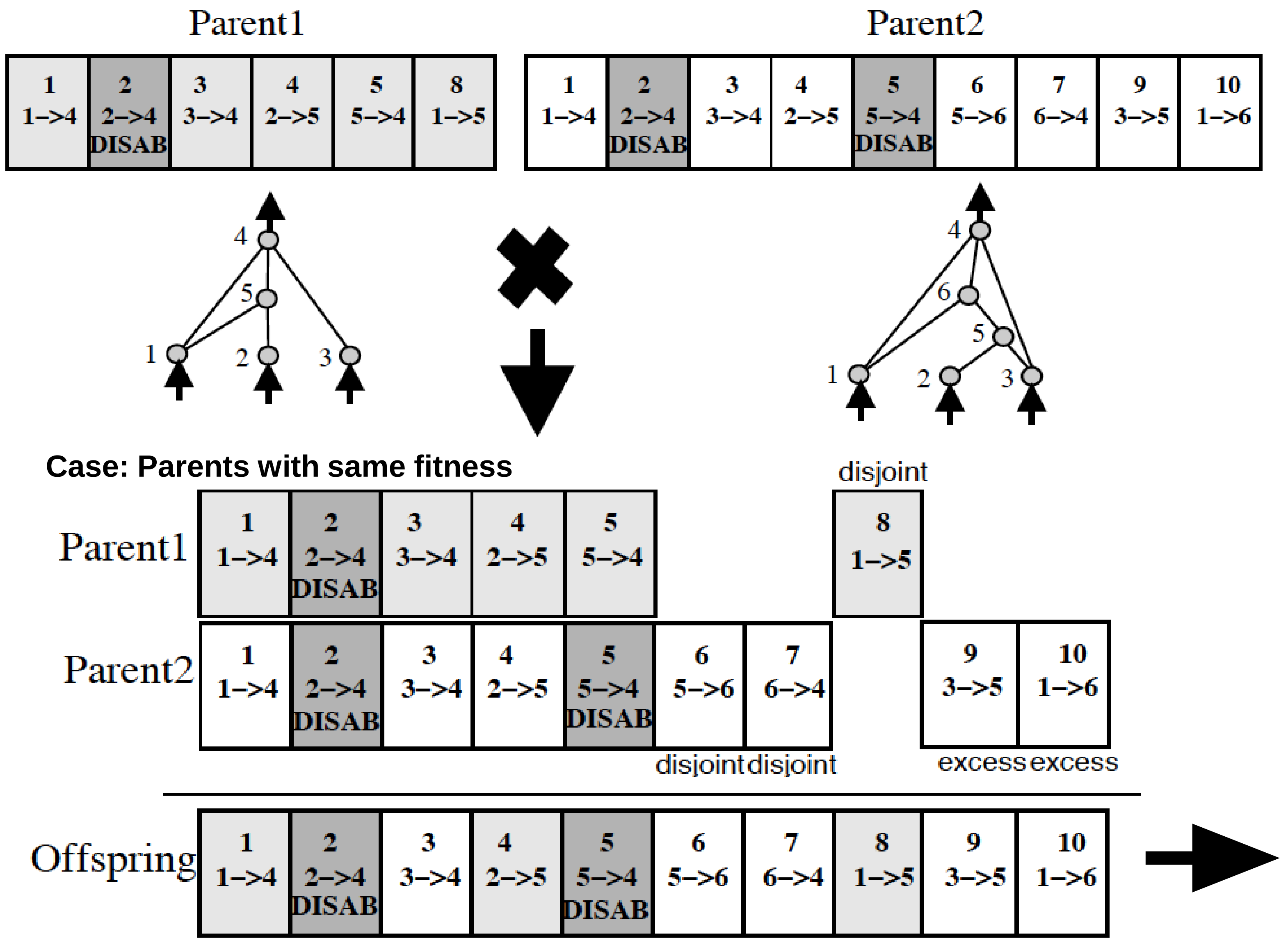
Historical Marking Crossover

- By marking new evolutions with a historical number, when it comes time to crossover two individuals, this can be done with much less chance of creating individuals that are non-functional
- We consider 3 types of genes (links) when crossing over two NN
  - **Matching** Genes
  - **Disjoint** Genes
  - **Excess** Genes



Parent1
1 1->4 | 2 2->4 DISAB | 3 3->4 | 4 2->5 | 5 5->4 | 8 1->5
Parent2
1 1->4 | 2 2->4 DISAB | 3 3->4 | 4 2->5 | 5 5->4 DISAB | 6 5->6 | 7 6->4 | 9 3->5 | 10 1->6
1 2 3 4 5
1 2 3 4 5 6
disjoint
Parent1
1 1->4 | 2 2->4 DISAB | 3 3->4 | 4 2->5 | 5 5->4 | 8 1->5
Parent2
1 1->4 | 2 2->4 DISAB | 3 3->4 | 4 2->5 | 5 5->4 DISAB | 6 5->6 | 7 6->4 | 9 3->5 | 10 1->6
disjoint disjoint
excess excess
Offspring
1 1->4 | 2 2->4 DISAB | 3 3->4 | 4 2->5 | 5 5->4 DISAB | 6 5->6 | 7 6->4 | 8 1->5 | 9 3->5 | 10 1->6
1 2 3 4 5 6

Historical Marking Crossover



Parent1

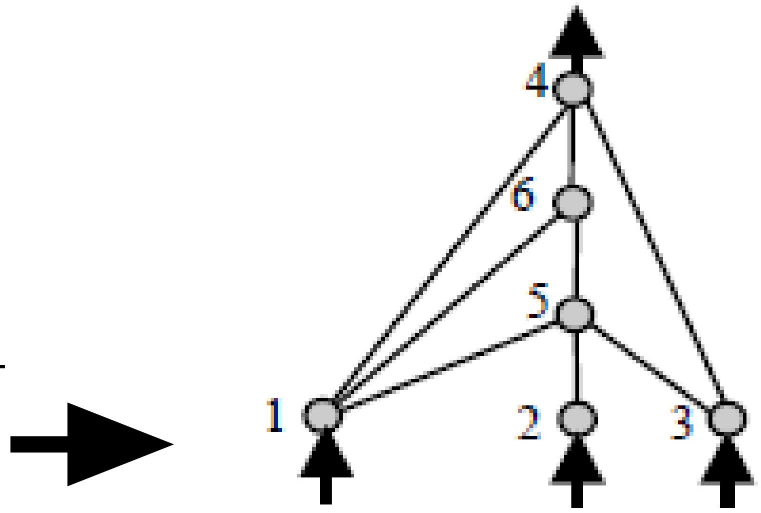
1	2	3	4	5	8
1->4	2->4 DISAB	3->4	2->5	5->4	1->5

Parent2

1	2	3	4	5	6	7	9	10
1->4	2->4 DISAB	3->4	2->5	5->4 DISAB	5->6	6->4	3->5	1->6

Case: Parents with same fitness

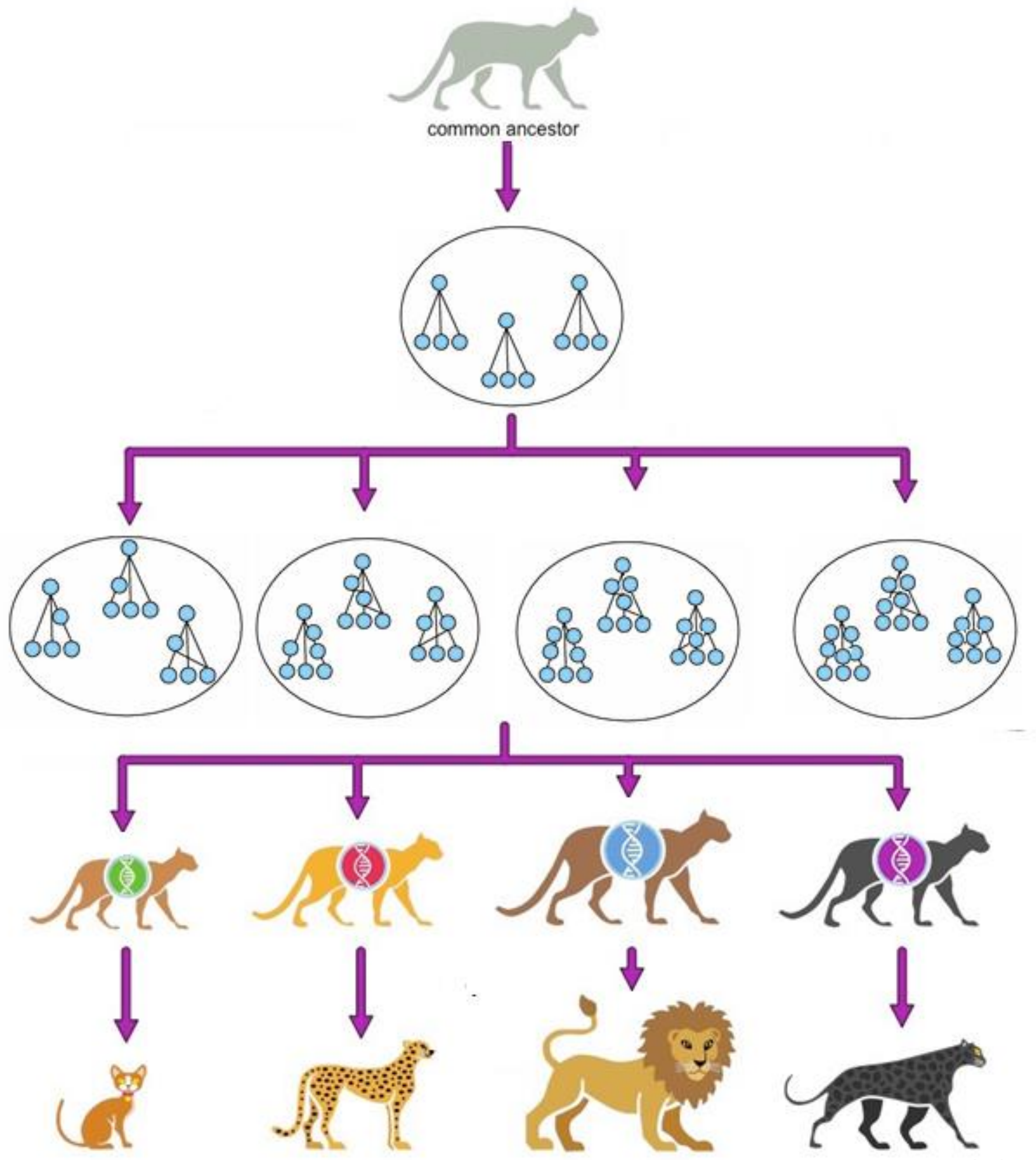
							disjoint		
Parent1	1 1->4	2 2->4 DISAB	3 3->4	4 2->5	5 5->4			8 1->5		
Parent2	1 1->4	2 2->4 DISAB	3 3->4	4 2->5	5 5->4 DISAB	6 5->6	7 6->4		9 3->5	10 1->6
						disjoint	disjoint		excess	excess
Offspring	1 1->4	2 2->4 DISAB	3 3->4	4 2->5	5 5->4 DISAB	6 5->6	7 6->4	8 1->5	9 3->5	10 1->6



- Matching genes are inherited randomly,
- Disjoint genes and Excess genes are inherited from the more fit parent.
- If both parents have the same fitness, then inherit all disjoint and excess genes.

Speciation

- Adding a new connection or node before any optimization of weights have occurred often leads to a lower performing individual.
- This puts new structures at a disadvantage.
- How can we **protect new structures and allow them to optimize** before we eliminate them from the population entirely?
- **Speciation simply splits up the population into several species** based on the similarity of topology and connections.
- The historical markings presented in the previous slides makes easier to split the the population into several species

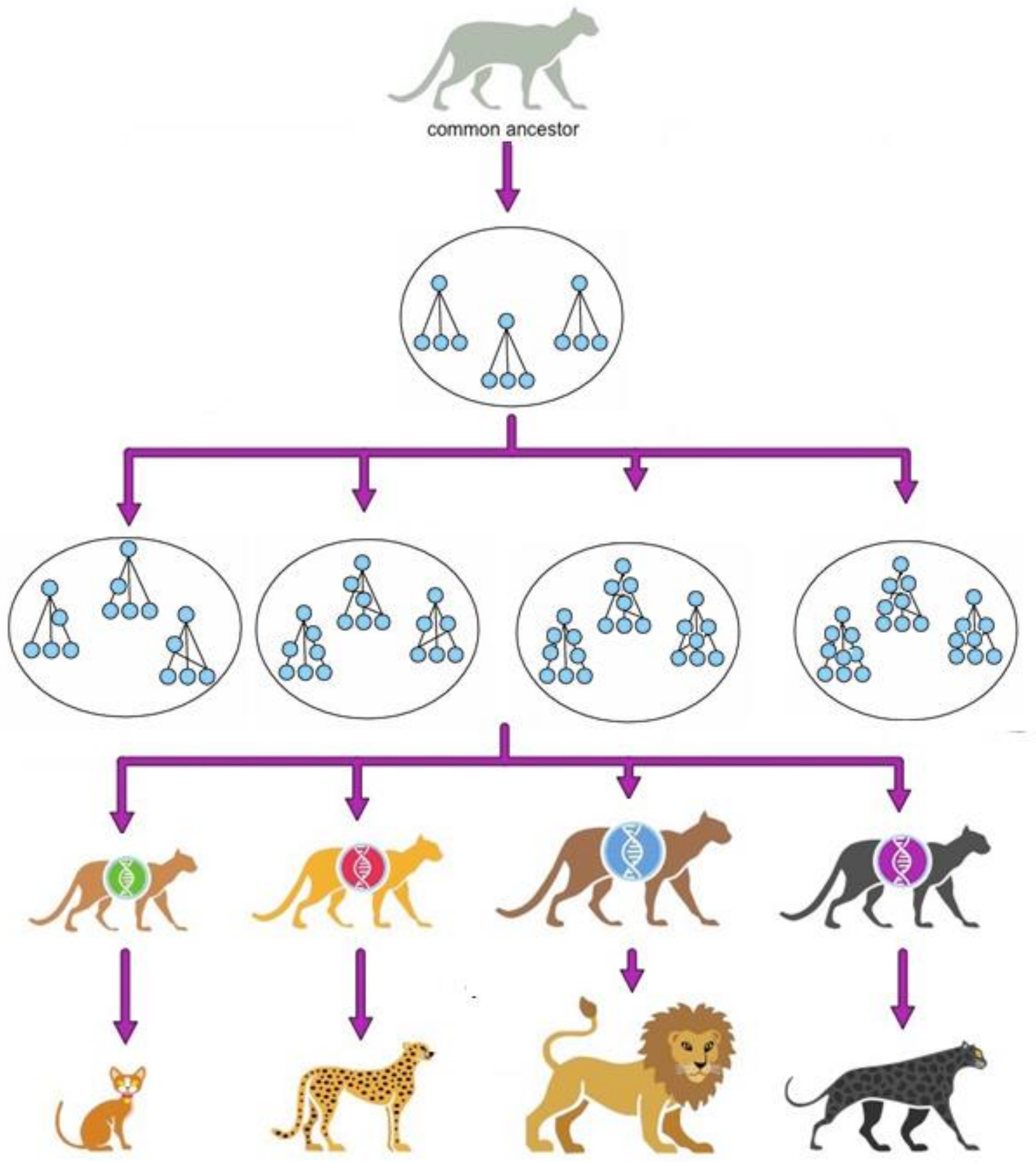


Speciation

- Species are determined by compatibility distance – calculated by measuring diversity of two individuals

$$\delta_{ij} = \frac{c_1 E_{ij}}{N_{ij}} + \frac{c_2 D_{ij}}{N_{ij}} + c_3 W_{ij}$$

- δ_{ij} is the measure of how compatible two NN are with each other
- c_1, c_2, c_3 are user parameters that dictates the importance given to E_{ij}, D_{ij}, W_{ij}
- E_{ij} represents the total number of excess genes
- D_{ij} represents the total number of disjoint genes
- E_{ij} represents the total weight difference between matching genes
- N_{ij} represents the number of genes of the NN with the larger number



Speciation

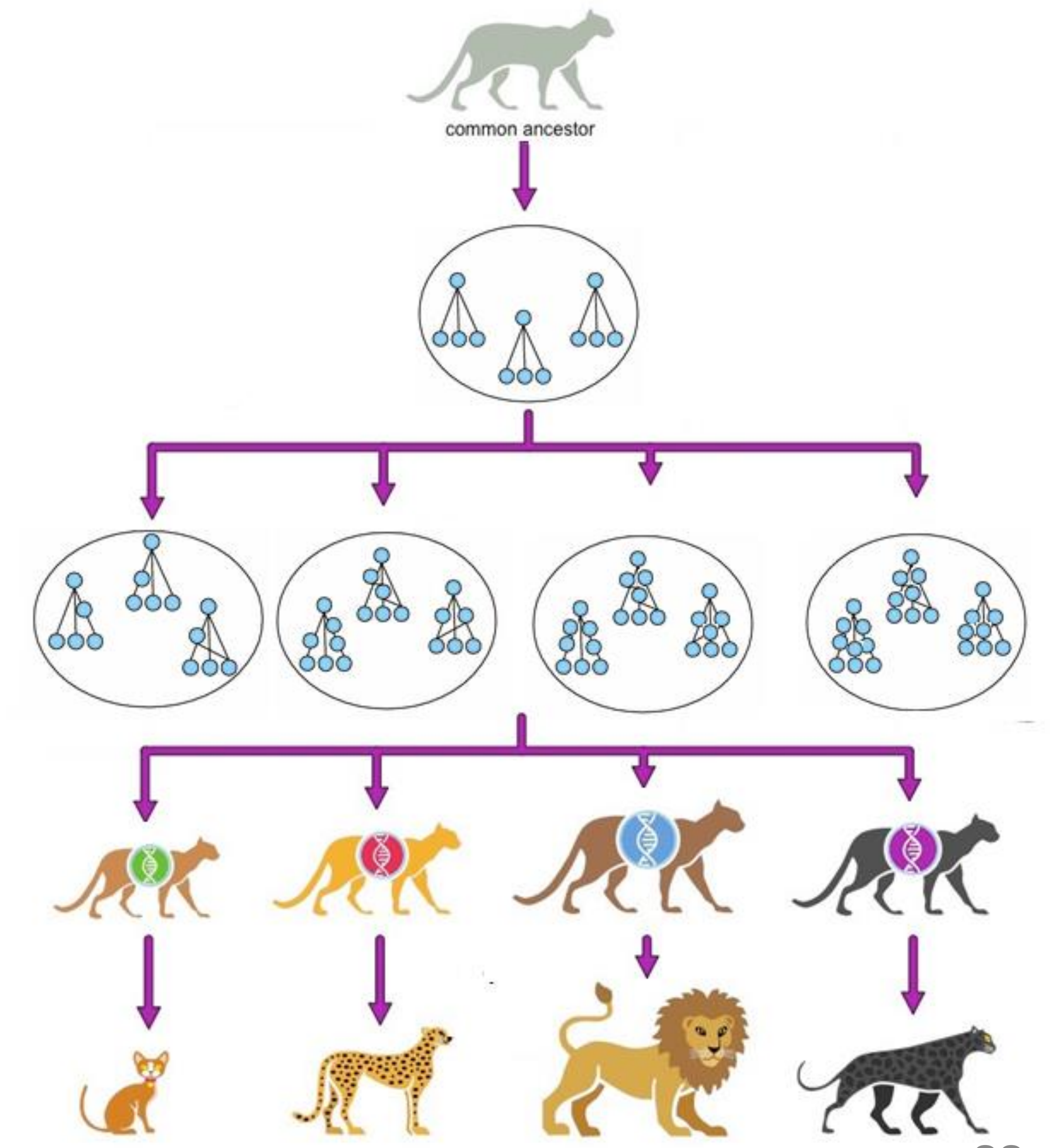
- Species are determined by compatibility distance – calculated by measuring diversity of two individuals

$$\delta_{ij} = \frac{c_1 E_{ij}}{N_{ij}} + \frac{c_2 D_{ij}}{N_{ij}} + c_3 W_{ij}$$

Example:

NN1	**ID=1 W=0.25 Matc.**	**ID=2 W=0.55 Disj.**		**ID=4 W=0.78 Disj.**		**ID=6 W=0.2 Excs.**
NN2	**ID=1 W=0.15 Matc.**		**ID=3 W=0.92 Disj.**		**ID=5 W=0.37 Disj.**	

$$\delta_{12} = \frac{c_1 \times 1}{4} + \frac{c_2 \times 4}{4} + c_3(0.25 - 0.15)$$



Fitness Sharing

- Species are determined by compatibility distance – calculated by measuring diversity of two individuals

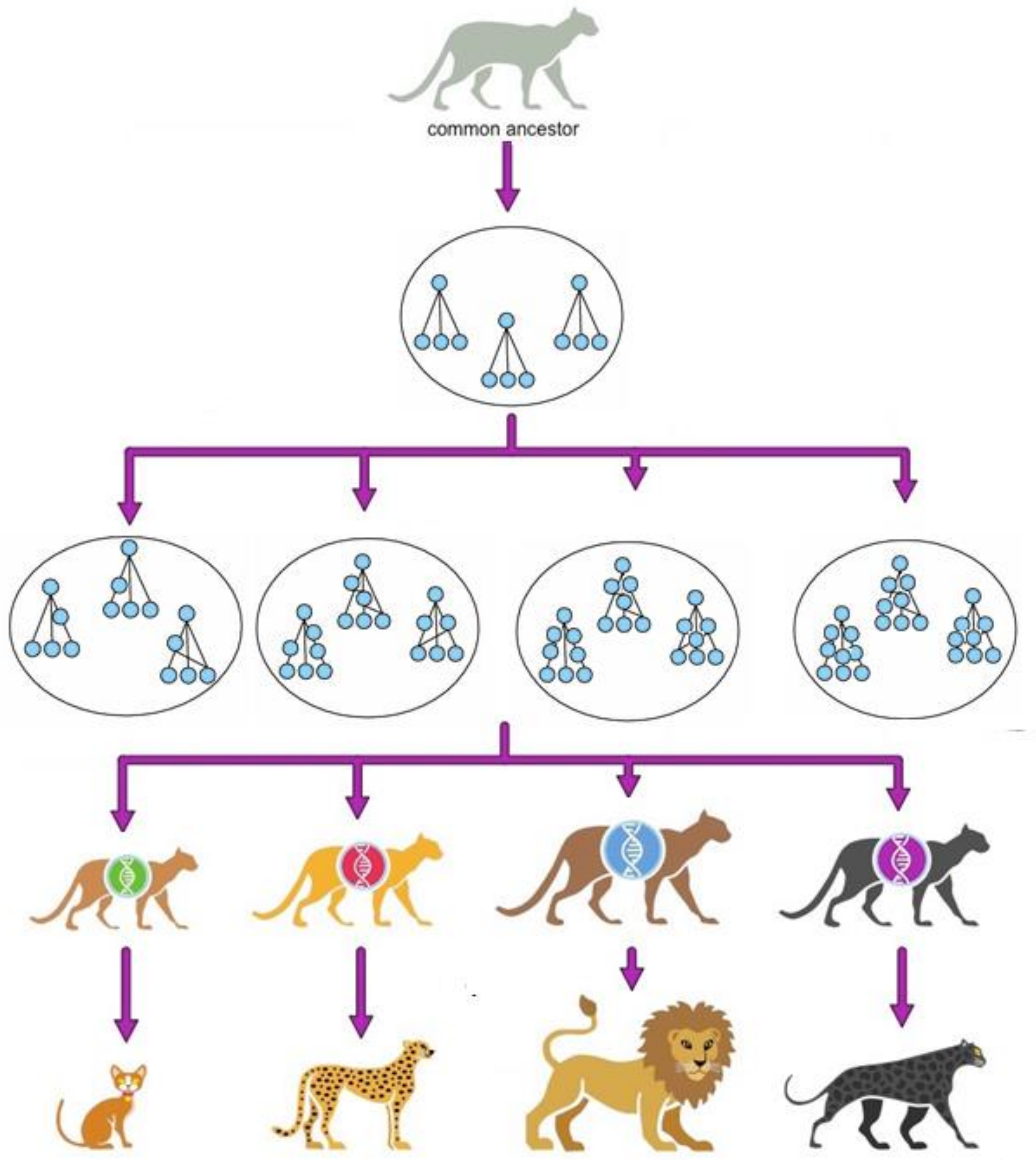
$$\delta_{ij} = \frac{c_1 E_{ij}}{N_{ij}} + \frac{c_2 D_{ij}}{N_{ij}} + c_3 W_{ij}$$

- Two NN belong to a same species if the compatibility distance is smaller than a given threshold σ.

- Fitness sharing transforms the raw fitness of an individual into the shared one (usually lower). NEAT will prioritize species that are more unique.

$$f_i' = \frac{f_i}{S_i}$$

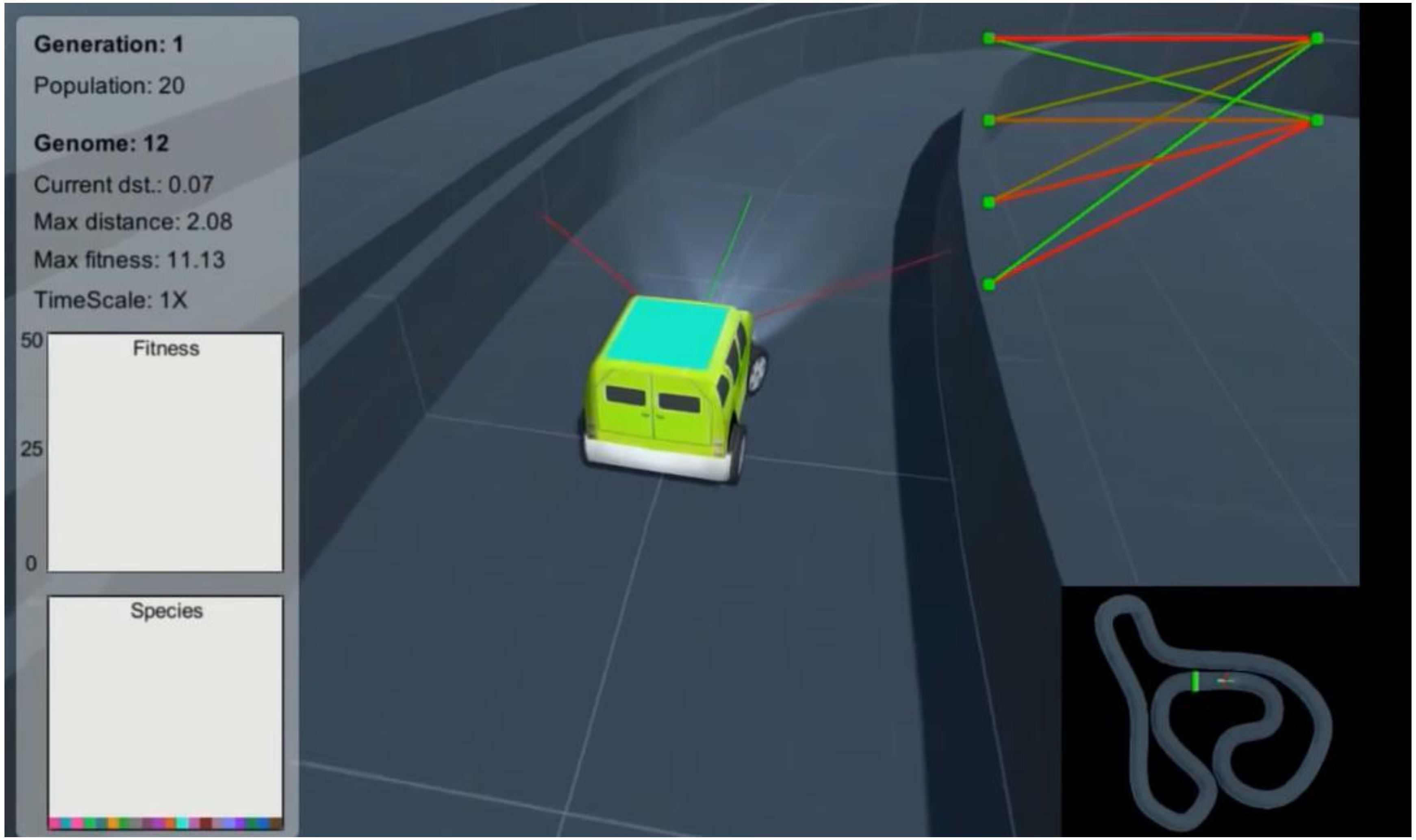
Where S_i is the number of NN in the same species as the NN i



Minimal Structure

- A large goal of NEAT is to create a framework for evolving networks that allowed for minimal networks to be evolved.
- The idea is to build an algorithm that started with the minimal amount of nodes and connections, evolving complexity as time goes on if and only if it is found to be useful and necessary.
- NEAT sets up their algorithm to evolve minimal networks by starting all networks with no hidden nodes. Each individual in the initial population is simply input nodes, output nodes, and a series of connection genes between them. By itself, this may not necessarily work, but when combined with the idea of speciation, this proves to be a powerful idea in evolving minimal, yet high-performing networks.
- Next Phase in NEAT research: combining NEAT with gradient-descent methods

Selfless Driving Car


Generation: 1
Population: 20
Genome: 12
Current dst.: 0.07
Max distance: 2.08
Max fitness: 11.13
TimeScale: 1X

https://www.youtube.com/watch?v=2bW9CdFcaUI

Multi-Objective Optimization

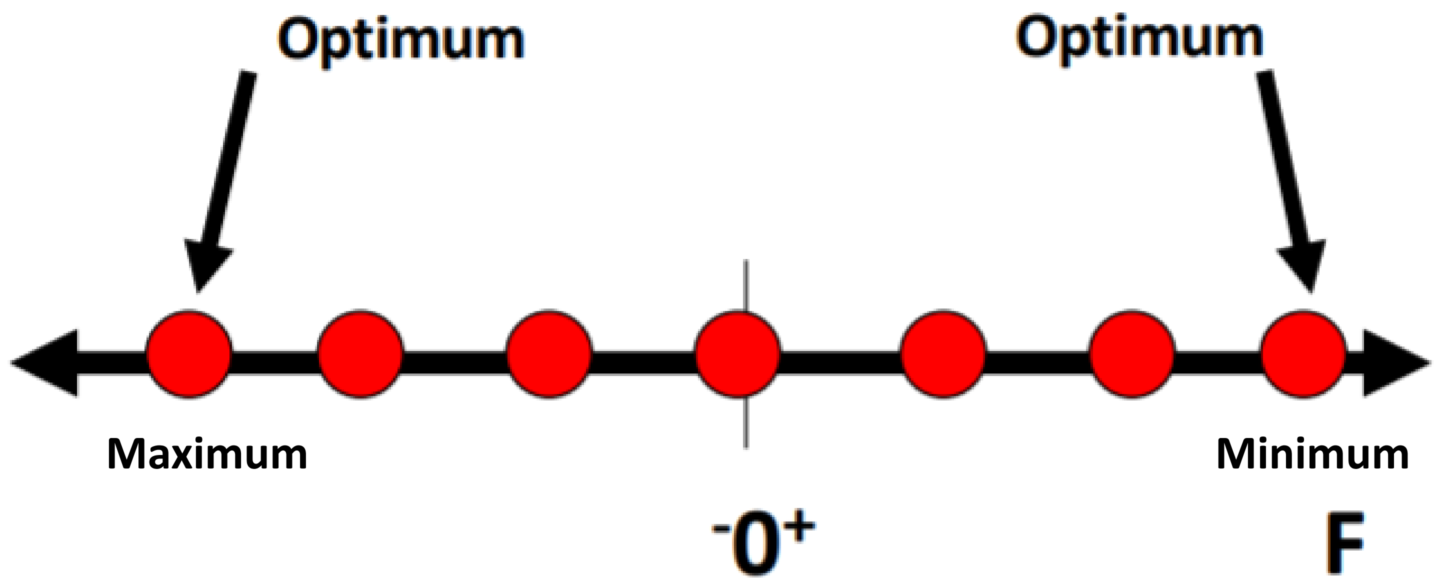
Nuno Antunes Ribeiro

Assistant Professor

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN | Engineering Systems and Design

Single Objective Optimization

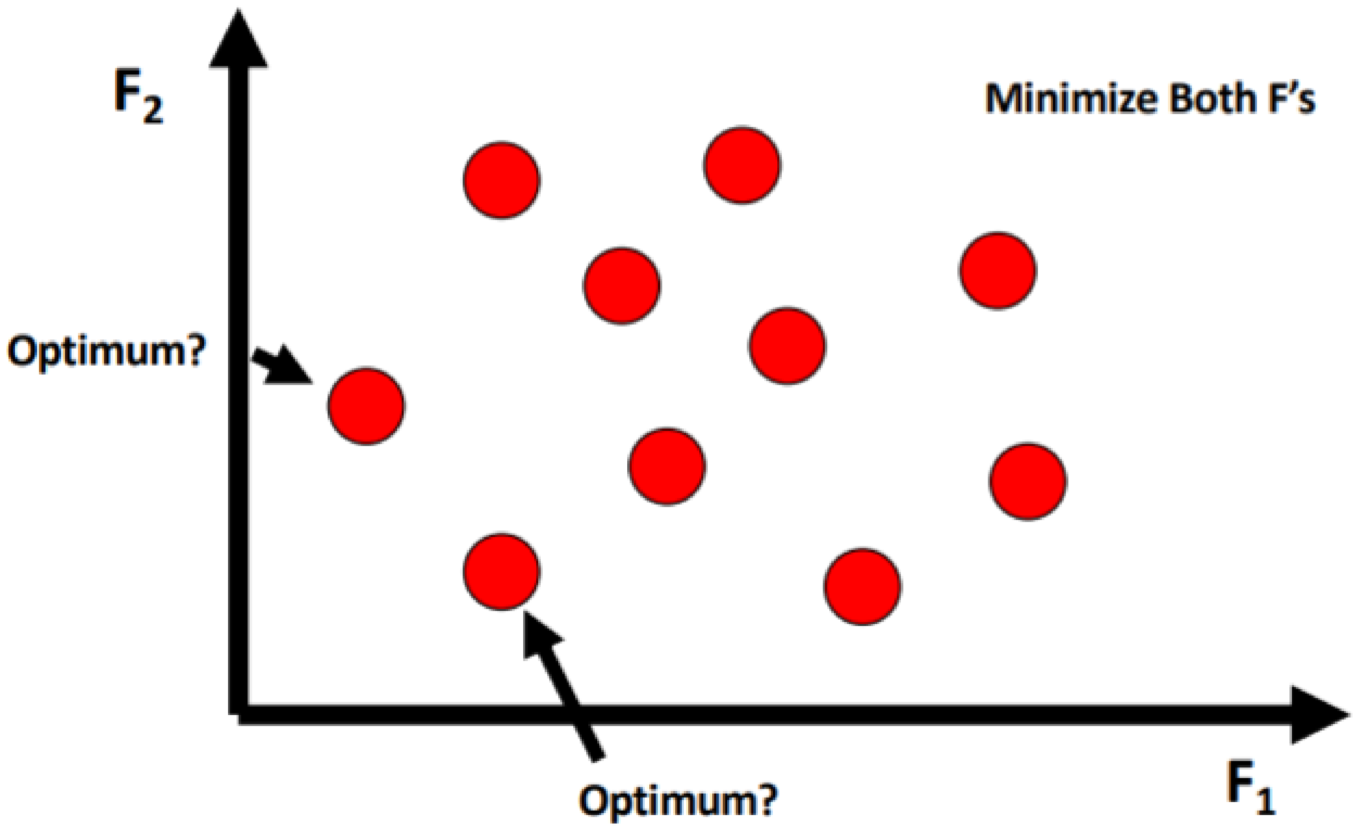
- The problem has a 1-dimensional performance space and the optimum
- point is the one that is the furthest toward the desired extreme.
- There is typically only a single solution that gives the best objective value.



Source: Håken Jevne, Norwegiam University of Science and Technology, Slides Multi-objective Evolutionary Algorithms

Multiple Objective Optimization (MOO)

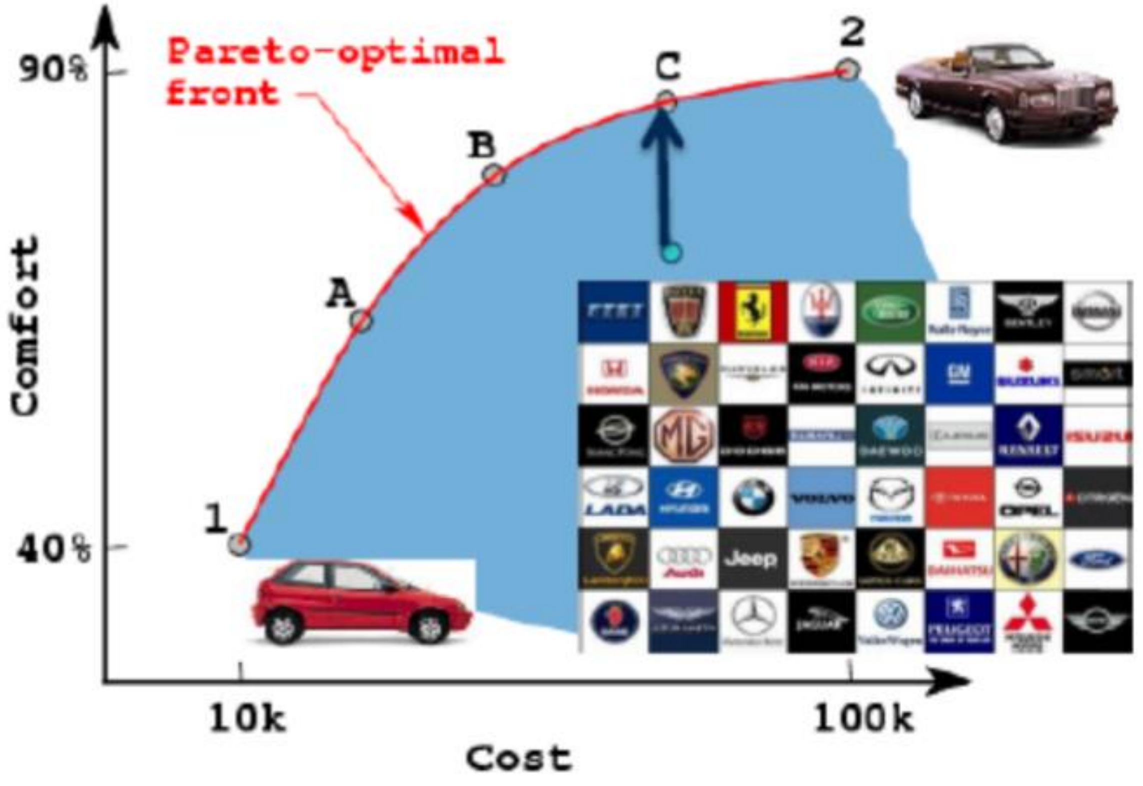
- But what happens in a case like this (conflicting)



Source: Håken Jevne, Norwegiam University of Science and Technology, Slides Multi-objective Evolutionary Algorithms

Why Multi-Objective Optimization (MOO)?

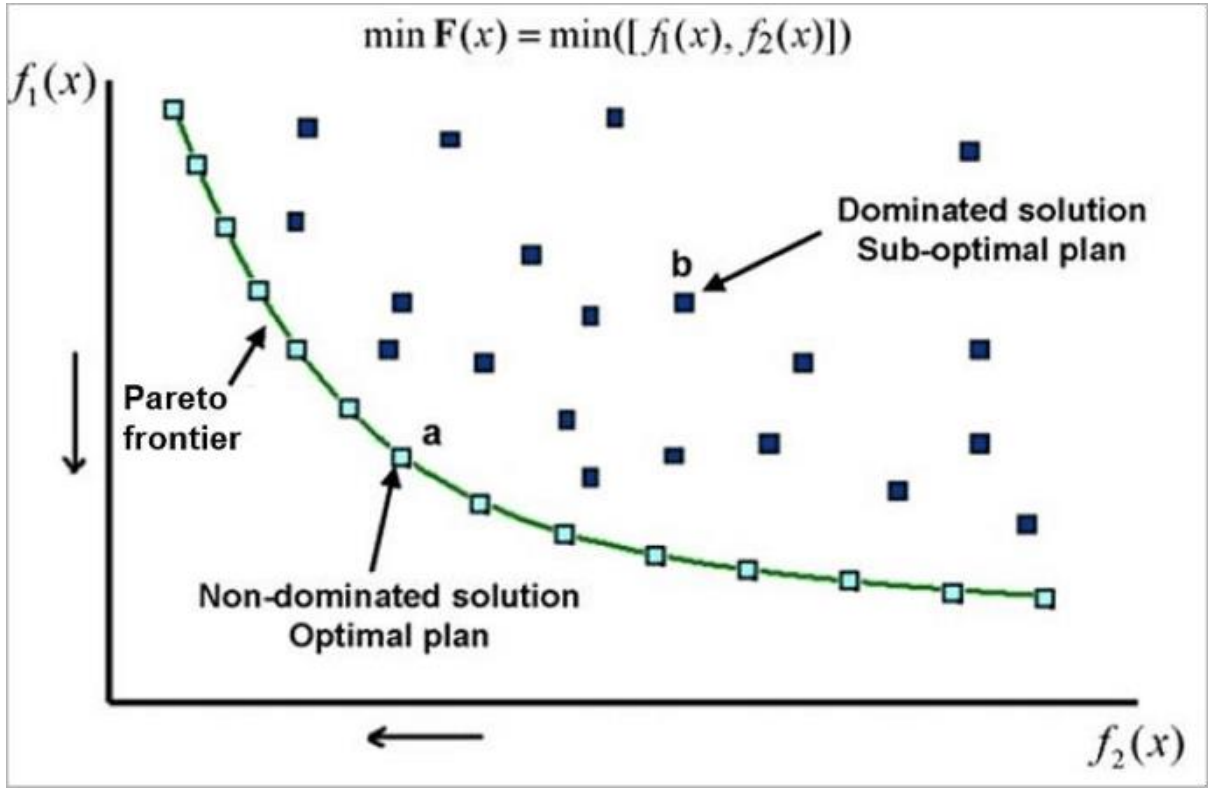
- Most industry optimization problems are multi-objective
- Example:
  - **Buying an Automobile**
  - Objective = reduce cost, while maximize comfort.
  - No solution from this set makes both objectives look better than any other solution from the set.
  - **No single optimal solution**.
  - **Trade off between conflicting objectives** - cost and comfort.



$$f(x) = Obj_1 + Obj_2$$

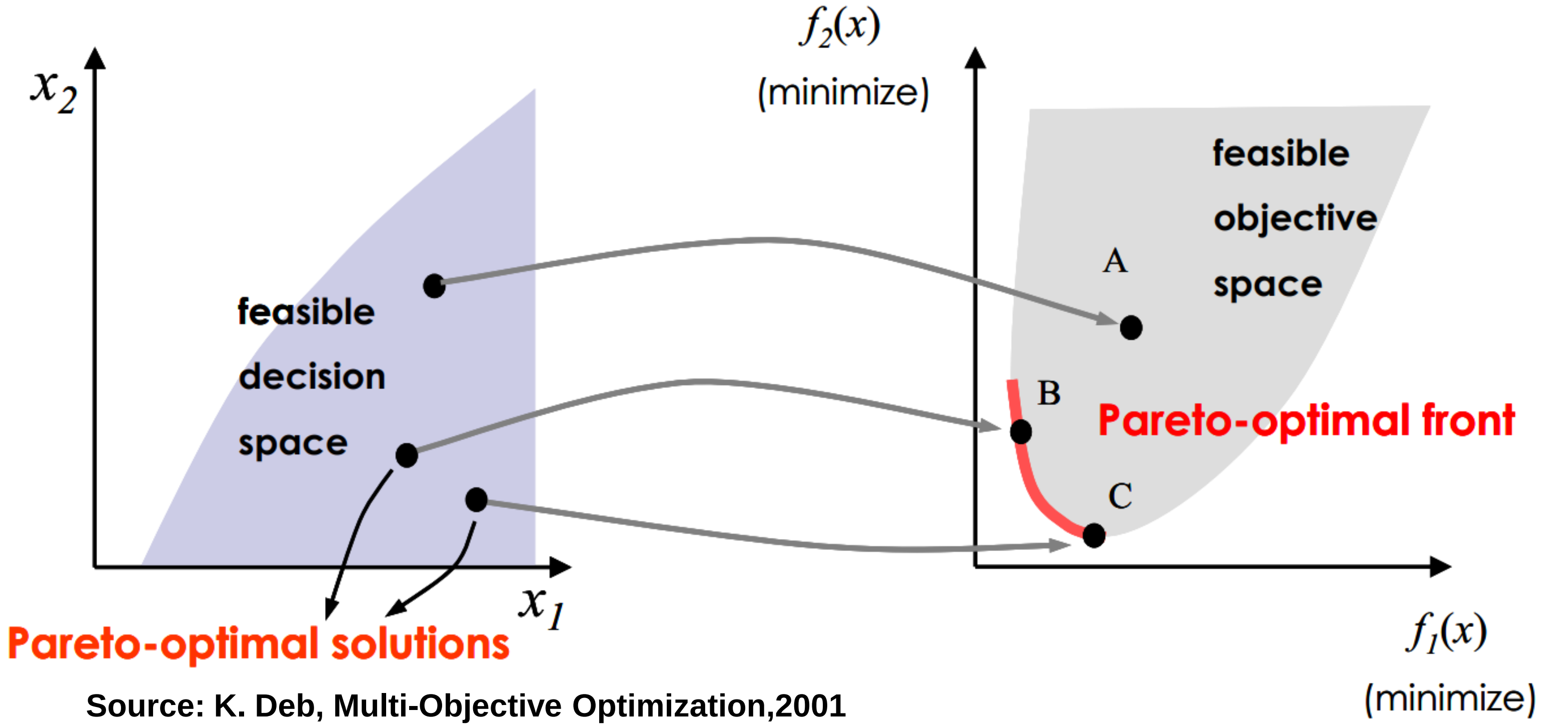
Pareto-Optimal Front

- Multi-objective optimization is extremely complex.
- We are not solving just a single optimization problem, but many optimizations at the same time.
- **It can be extremely computationally intensive.**
- Strategies to efficiently draw the pareto-optimal front of these problems is an important avenue for research that has been gaining more and more attention over the years.



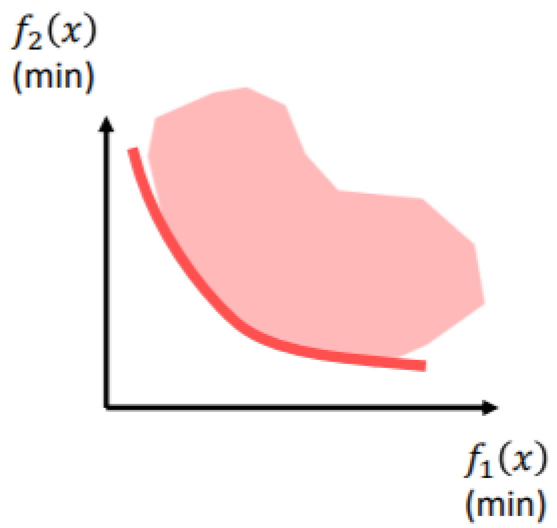
Pareto-Optimal Front

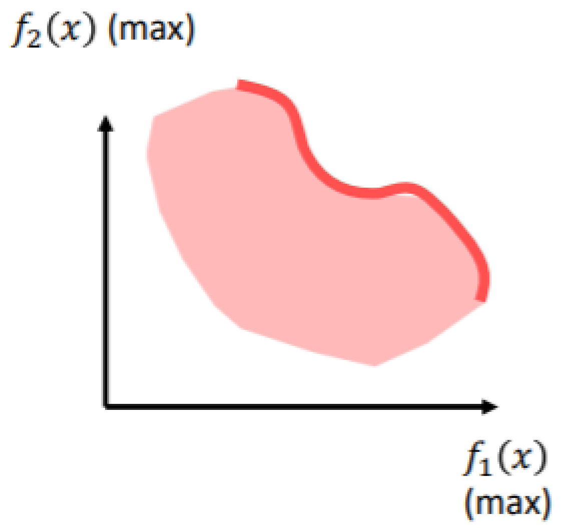
- The non-dominated set of the entire feasible decision space is called the **Pareto-optimal set**.
- The boundary defined by the set of all point mapped from the Pareto optimal set is called the **Pareto optimal front.**

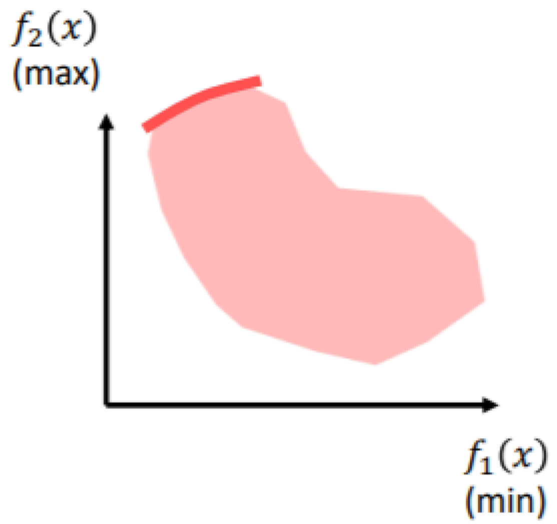


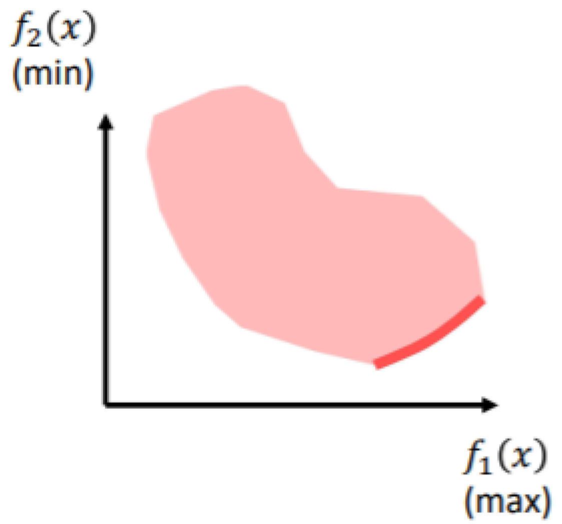
Source: K. Deb, Multi-Objective Optimization,2001

Pareto-Optimal Front









Dominance Test

- In the single-objective optimization problem, the superiority of a solution over other solutions is easily determined by comparing their objective function values
- In multi-objective optimization problem, the goodness of a solution is determined by the dominance
- Dominance Test:
  - x1 dominates x2, if
    - Solution x1 is no worse than x2 in all objectives
    - Solution x1 is strictly better than x2 in at least one objective

Solution	minimize		
	Obj 1	Obj 2	Obj 3
1	**20**	300	400
2	1000	**0**	1000
3	200	500	**10**
4	30	300	410
5	50	600	80

Dominated by solution 1

Non-dominated
How to find these solutions, they do not optimize any objective?

Classic MOO methods

- **Lexicographic method**
- **Weighted sum method**
- **ε-constraint method**
- Weighted metric method
- Rotated weighted metric method
- Dynamically changing the ideal solution
- Benson’s method
- Value function method

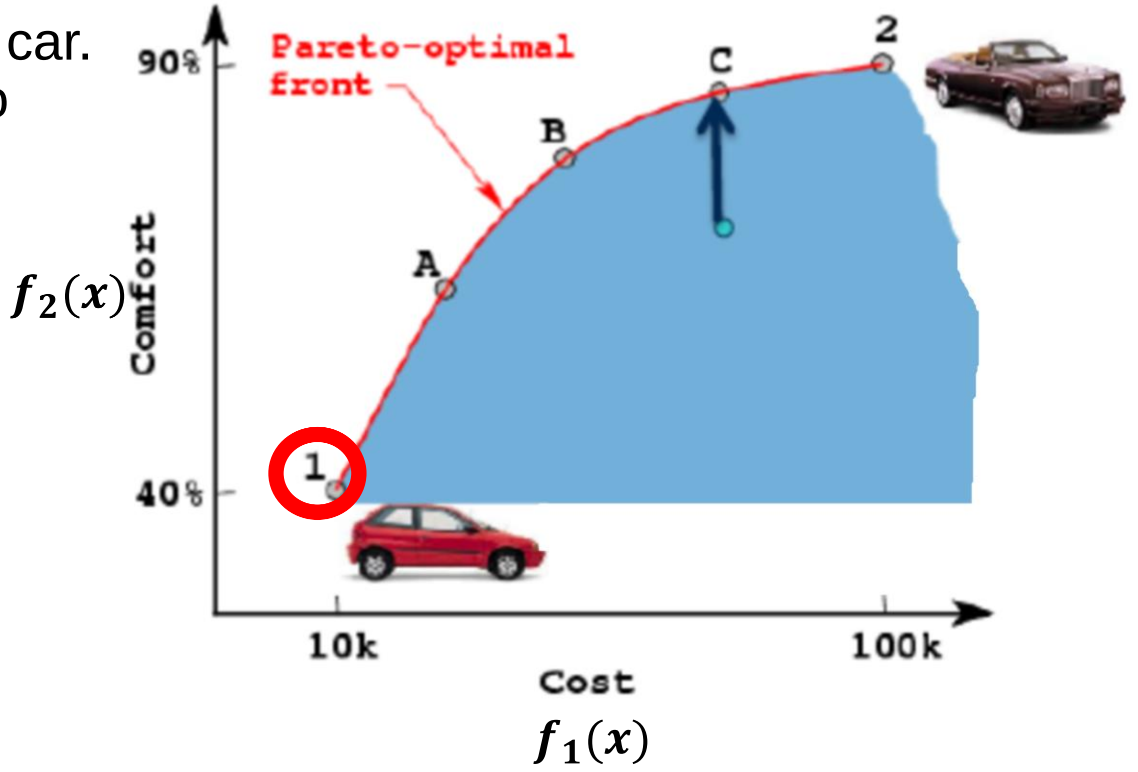
Lexicographic method

- With the **lexicographic method**, preferences are imposed by ordering the objective functions according to their importance or significance

- Example: your main goal is to find the cheapest car. Then, within the list of cheapest cars, you aim to purchase the most comfortable one.
- Solving the lexicographic method:
  - **Approach 1**

    Solve: $\boldsymbol{minimize\ f_1(x)}$
    Solve $\boldsymbol{maximize\ f_2(x); s.t. f_1(x) \leq \text{Optimum}}$
  - **Approach 2**

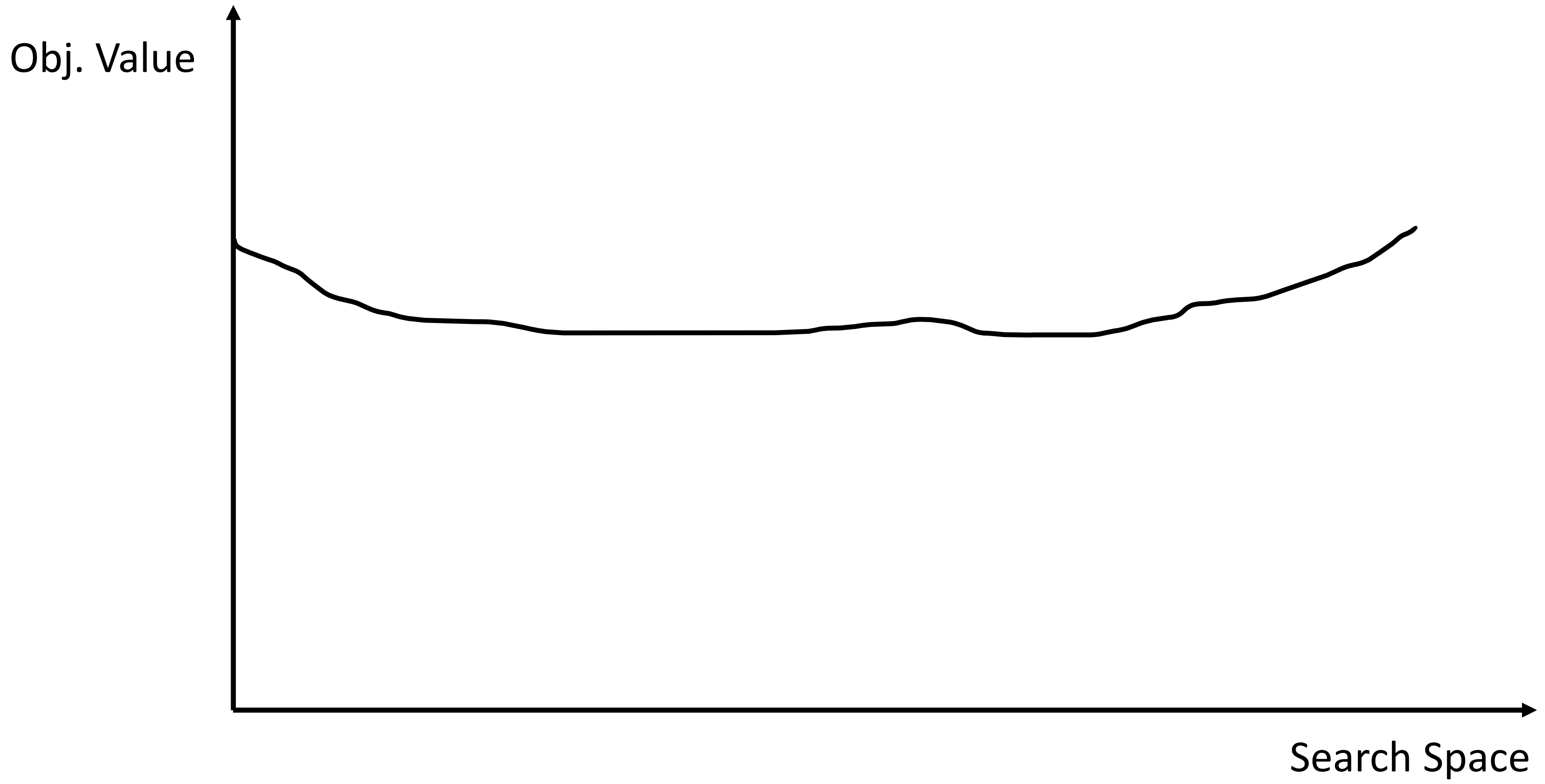
    Solve: $\boldsymbol{minimize\ Mf_1(x) - f_2(x)}$**, where M is a very large number**



- Main Issue: there may be a very good solution for objective 2, very near the optimal solution of objective 1

Different Problem – Different Landscapes

- Rugged Plain Landscape



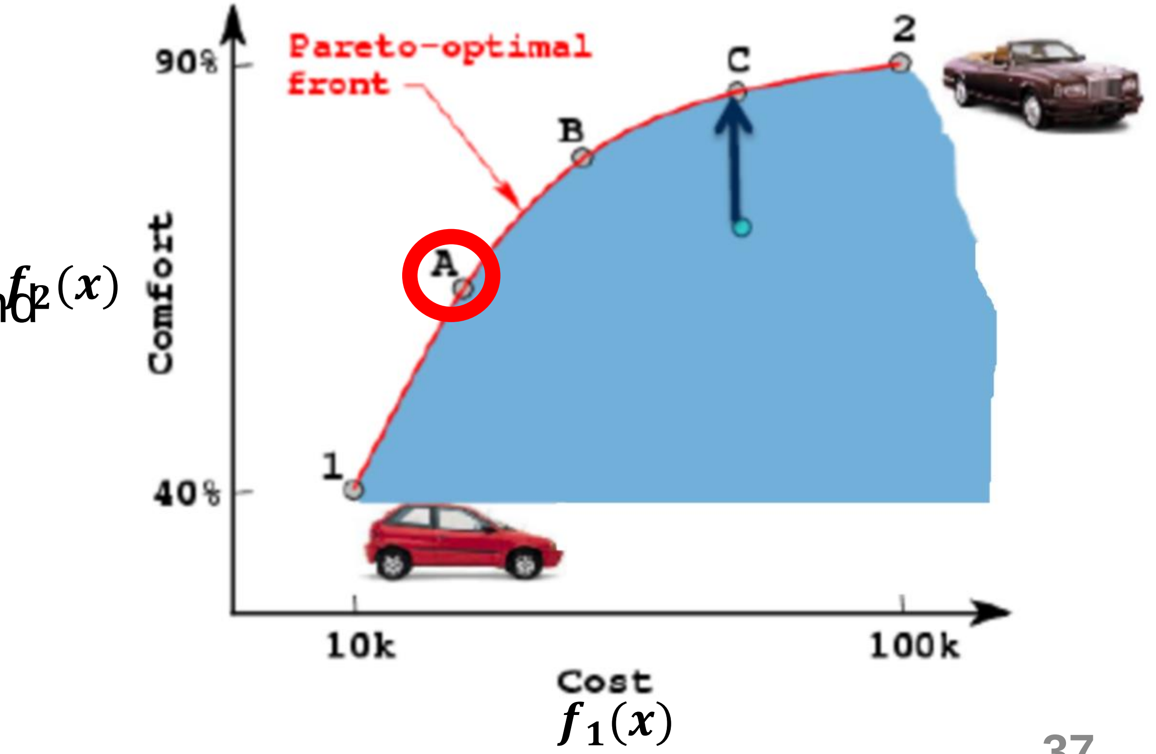
Problem with the objective function: A flat landscape indicates that regardless of the solution obtained the objective values are all going to be similar. This means that either there is nothing to optimize and therefore the problem is not relevant ; or the objective function is not well defined.
A flat landscape has nothing to do with the metaheuristic performance

Weighted method

- Scalarize a set of objectives into a single objective by adding each objective multiplied by a **user-specified weight**.
- Weight of an objective is chosen in proportion to the relative importance of the objective

Solve: $\boldsymbol{minimize\ \alpha\, f_1(x) + (1-\alpha) f_2(x)}$

- Main Issue: objectives may need to be normalized; e.g. it is difficult to compare cost and comfort.
- One of the simplest approaches is to optimize each of the objectives individually first. Then divide each objective by those optimum values and then sum up all normalized terms as one objective. The new objective will be dimensionless.

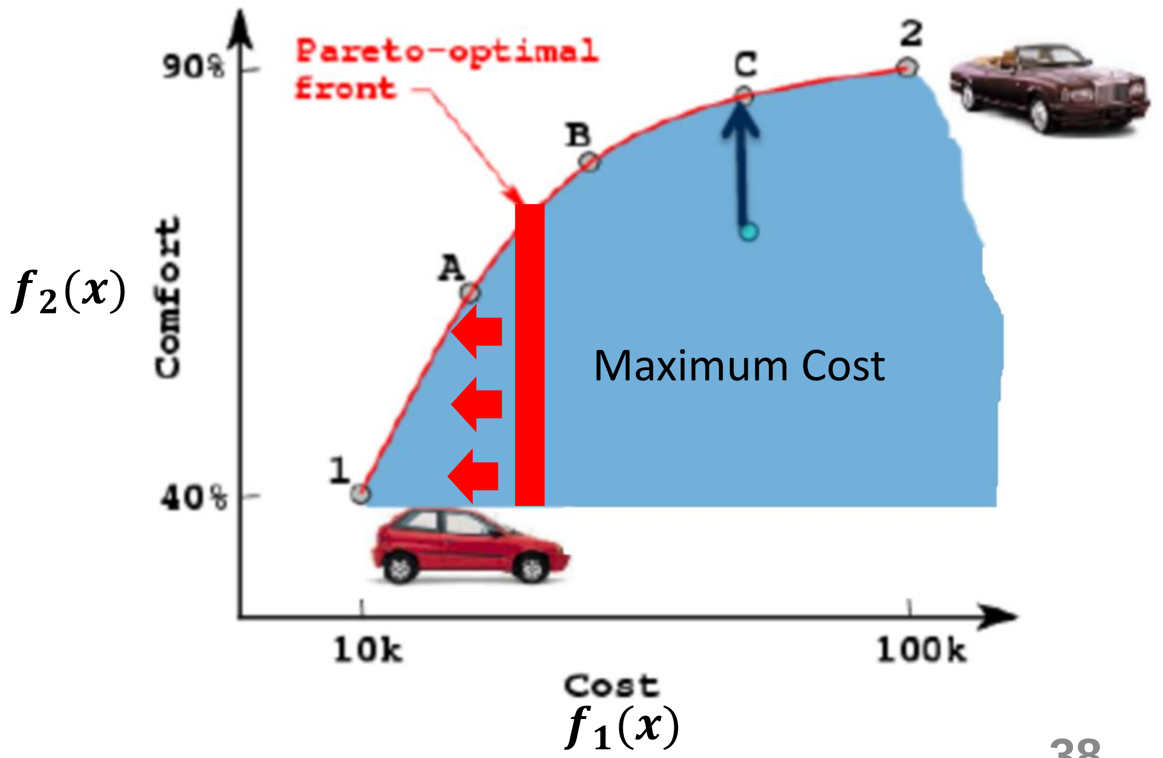


ε-constraint method

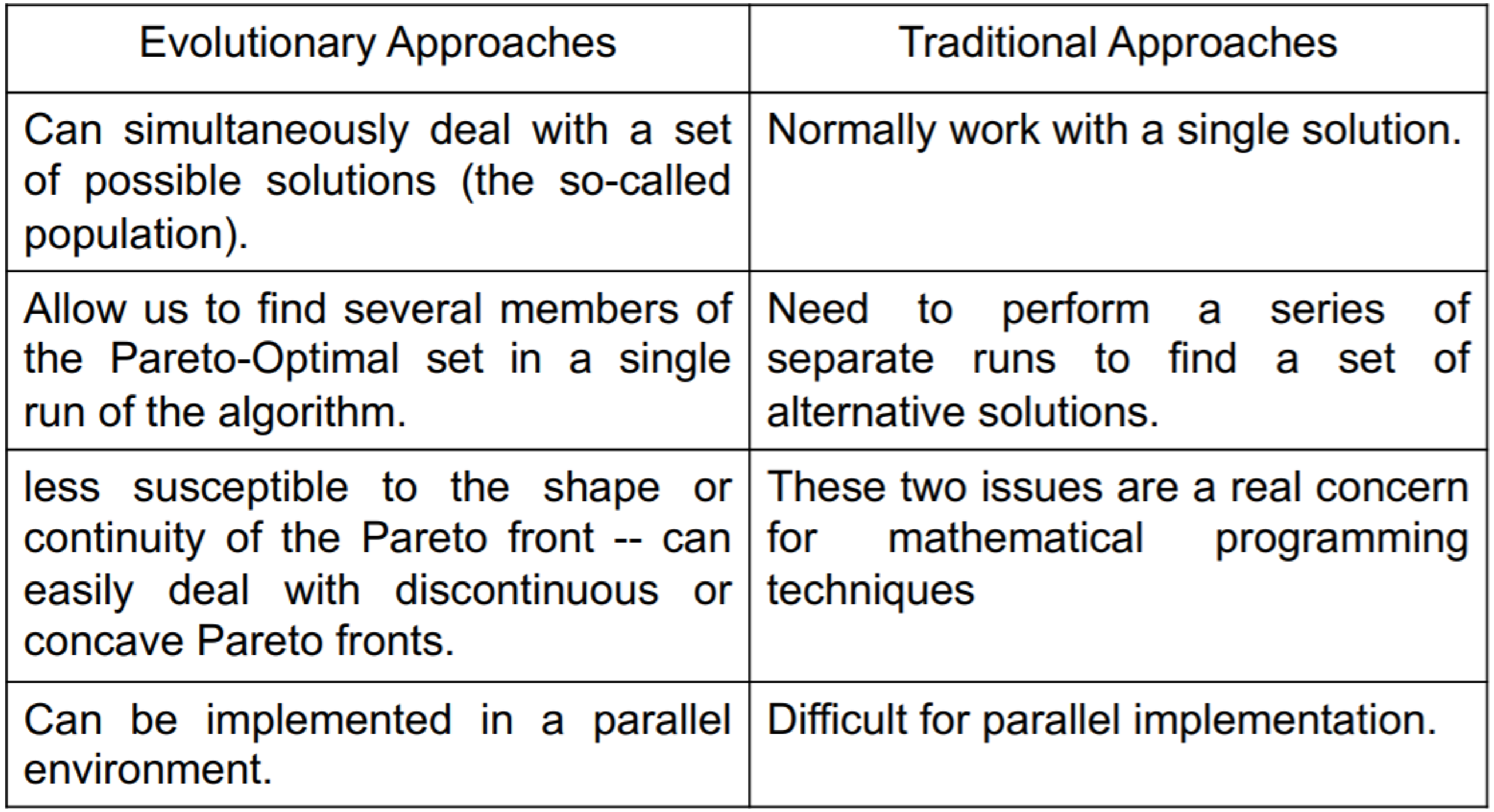
- Keep just one of the objectives
- Treat the rest as constraints and modify the constraint value.

Solve: $maximize\ f_2(x), s.t.\ f_1(x) \le \text{Target}$

- Main Issue: You may have no idea which targets to consider.



Evolutionary vs Traditional Approaches



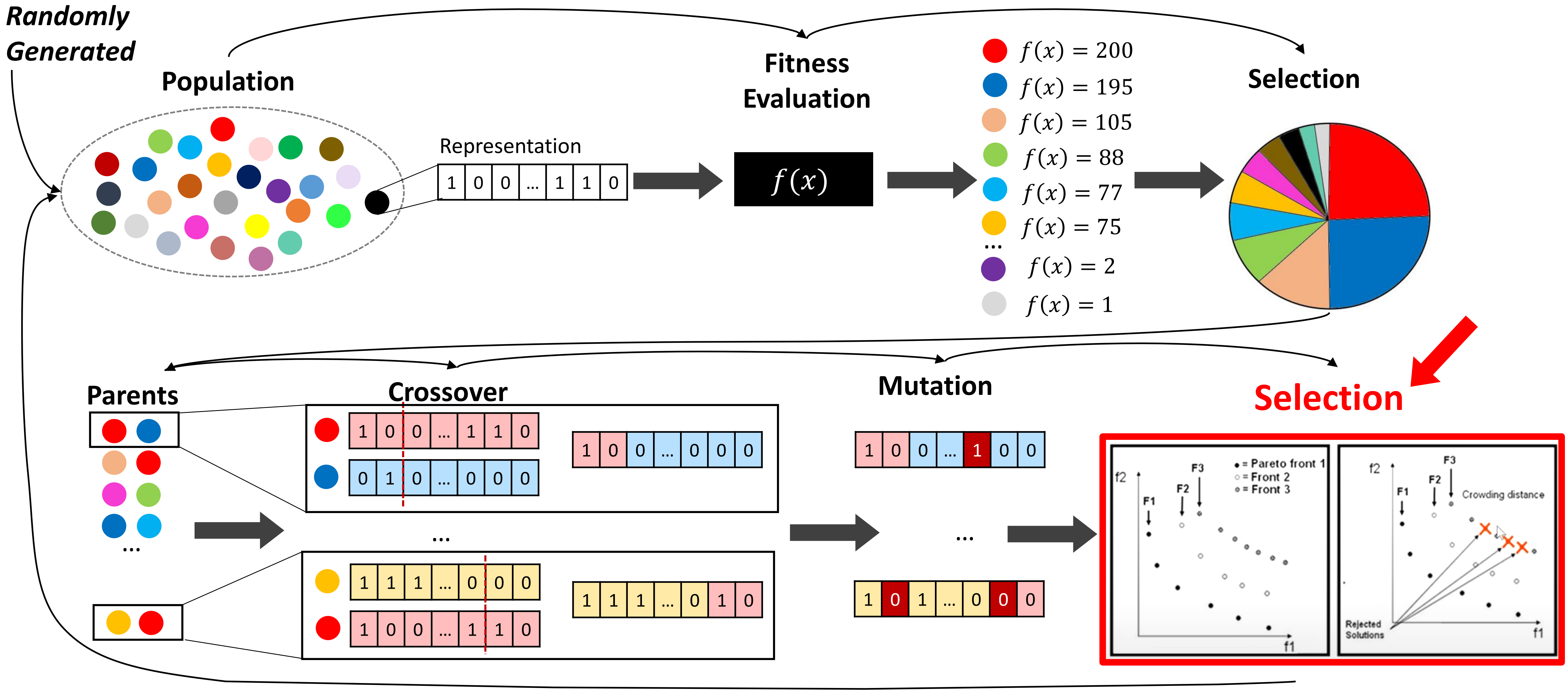
Evolutionary Approaches	Traditional Approaches
Can simultaneously deal with a set of possible solutions (the so-called population).	Normally work with a single solution.
Allow us to find several members of the Pareto-Optimal set in a single run of the algorithm.	Need to perform a series of separate runs to find a set of alternative solutions.
less susceptible to the shape or continuity of the Pareto front -- can easily deal with discontinuous or concave Pareto fronts.	These two issues are a real concern for mathematical programming techniques
Can be implemented in a parallel environment.	Difficult for parallel implementation.

Source: Håken Jevne, Norwegiam University of Science and Technology, Slides Multi-objective Evolutionary Algorithms

Elitist Non-Dominated Sorting GA (NSGA)

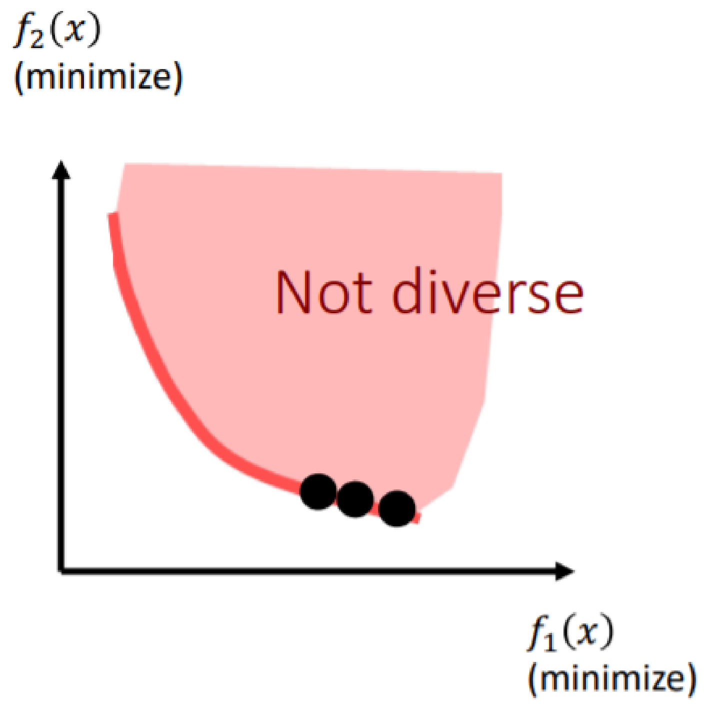
- **NSGA-II** is the most popular multi-objective metaheuristics used in MOO problems
- It was first proposed by Deb et. al. in 2002 in the paper “A fast and elitist multiobjective genetic algorithm: NSGA-II”– with more than 35000 citations.
- Three main features:
  - A **non-dominated sorting** where all the individuals are sorted according to the level of non-domination
  - An **elitism** selection to preserve solutions with best features
  - A **crowding distance** that emphasize on less crowded regions of the solution space to maintain diversity and spread of the solutions

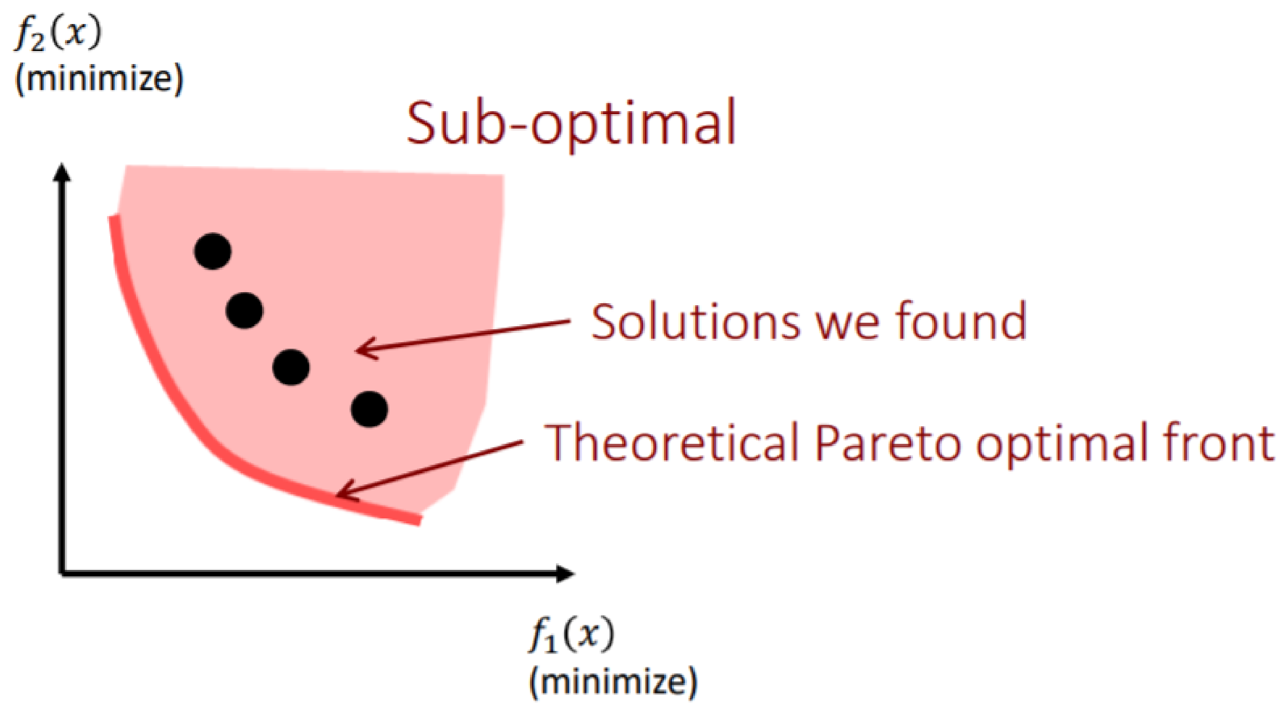
Evolutionary Algorithm (Review)



Goals of Multi-Objective Optimization (MOO)

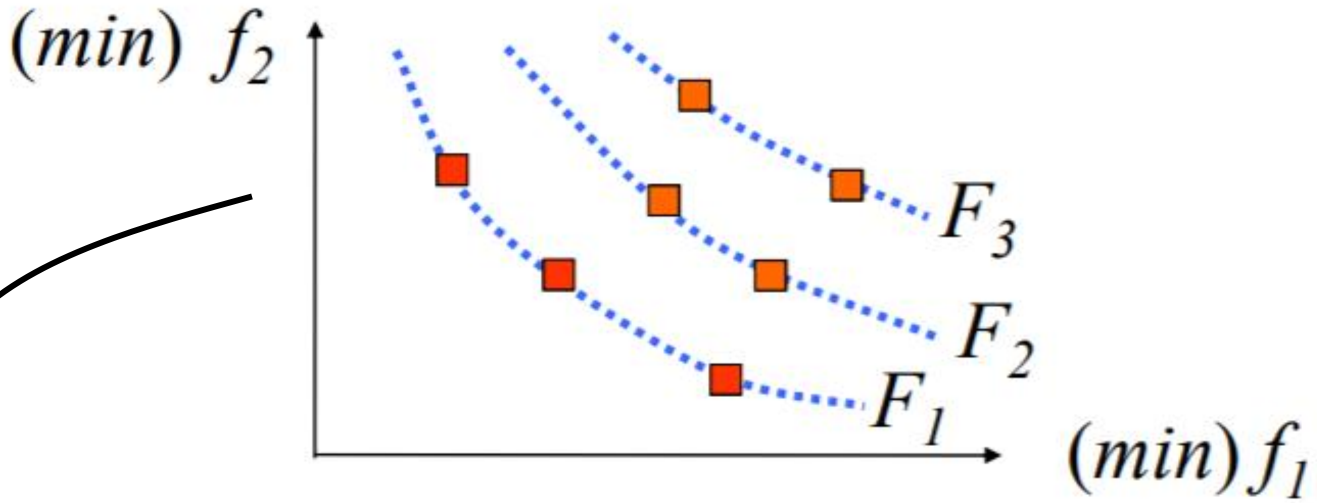
- Find a set of solutions as near as possible to Pareto-optimal front (**convergence**)
- Find a set of solutions as diverse as possible (**diversity**)

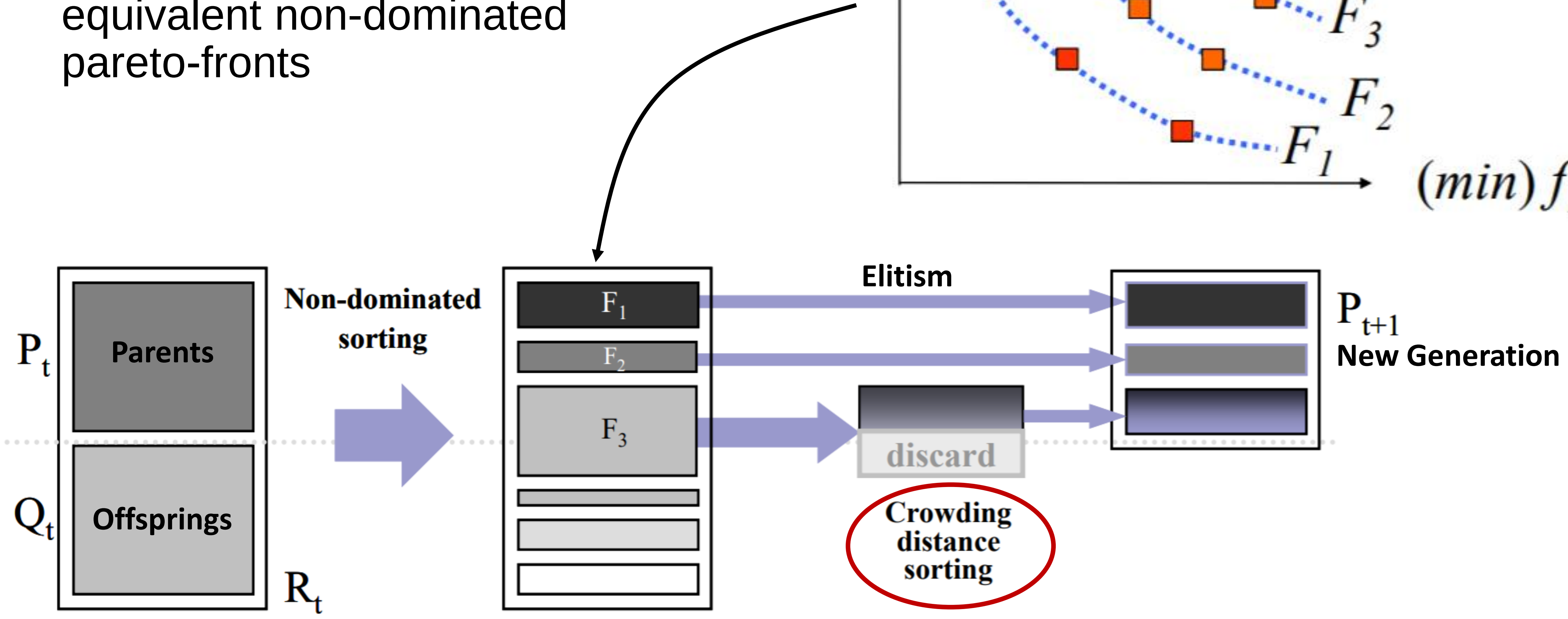




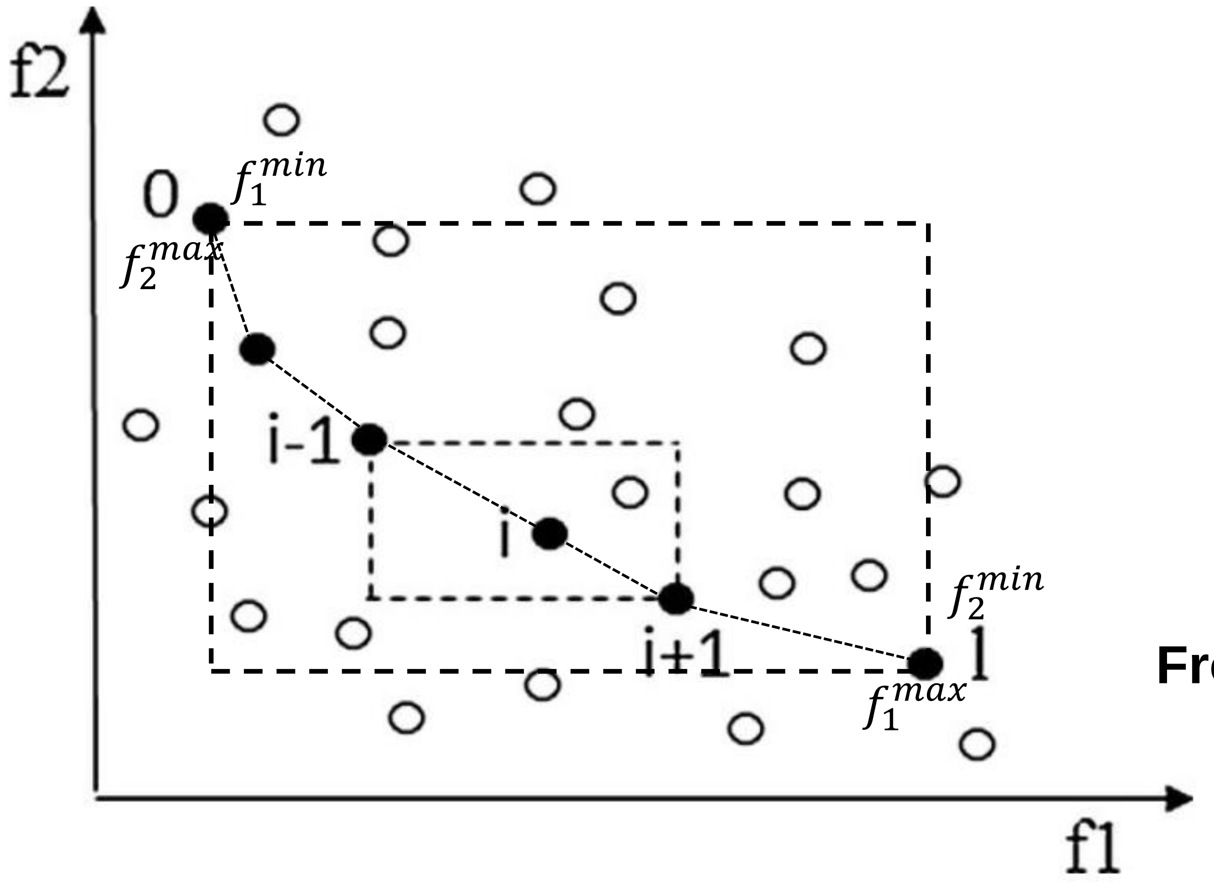
Non-Dominated Sorting

- Classify the solutions into a number of mutually exclusive equivalent non-dominated pareto-fronts





Crowding Distance



Front n

$$CD = \frac{f_1^{i+1} - f_1^{i-1}}{f_1^{max} - f_1^{min}} + \frac{f_2^{i+1} - f_2^{i-1}}{f_2^{max} - f_2^{min}}$$

$$CD = \sum_{M} \frac{f_m^{i+1} - f_m^{i-1}}{f_m^{max} - f_m^{min}}$$

M – Set of Objectives