UiO : University of Oslo

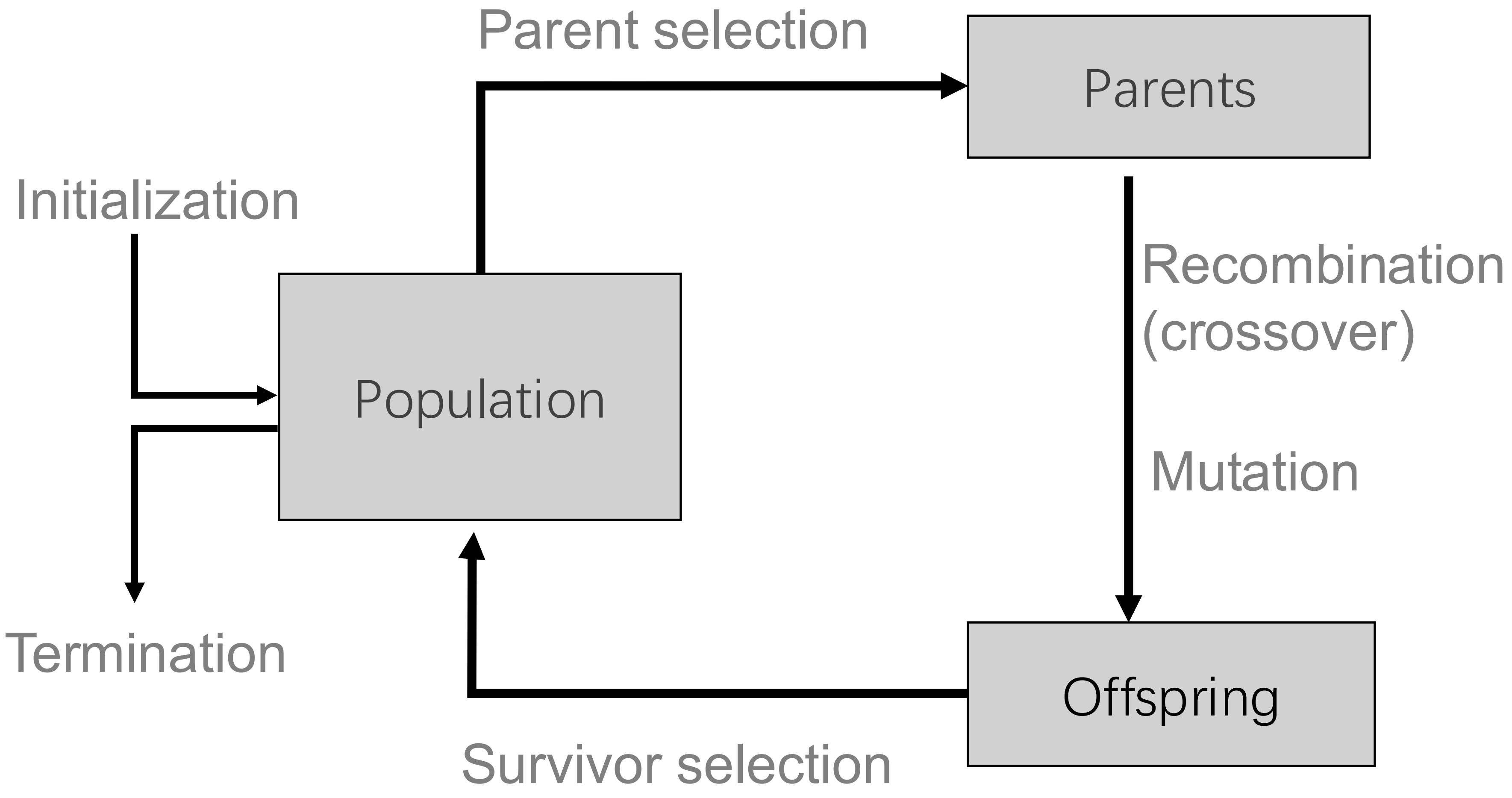
institutt for informatikk

IN3050/IN4050 - Introduction to Artificial Intelligence and Machine Learning

Lecture 6: *Evolutionary Algorithms 2 –Population Management and More*

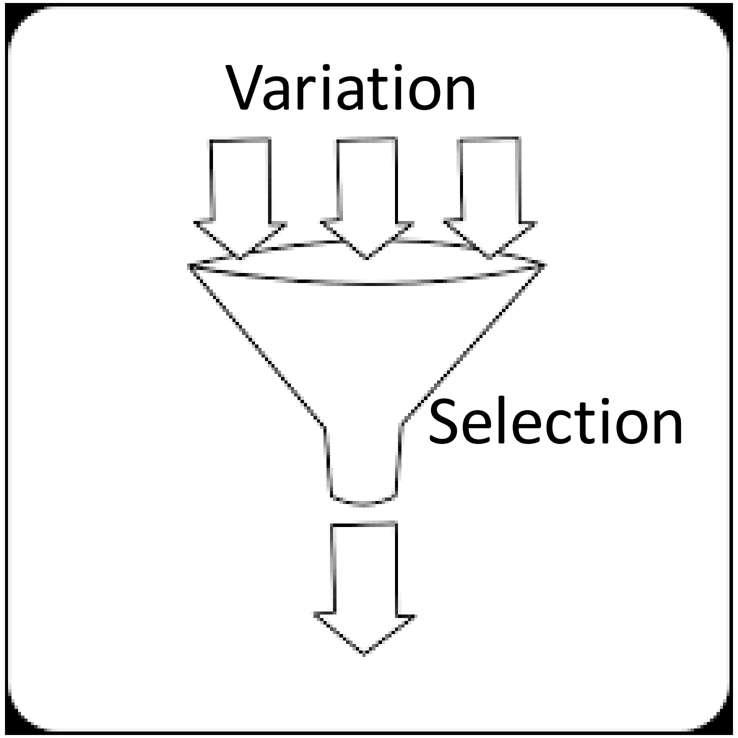
Pooya Zakeri Fall 2025

Repetition: General scheme of EAs

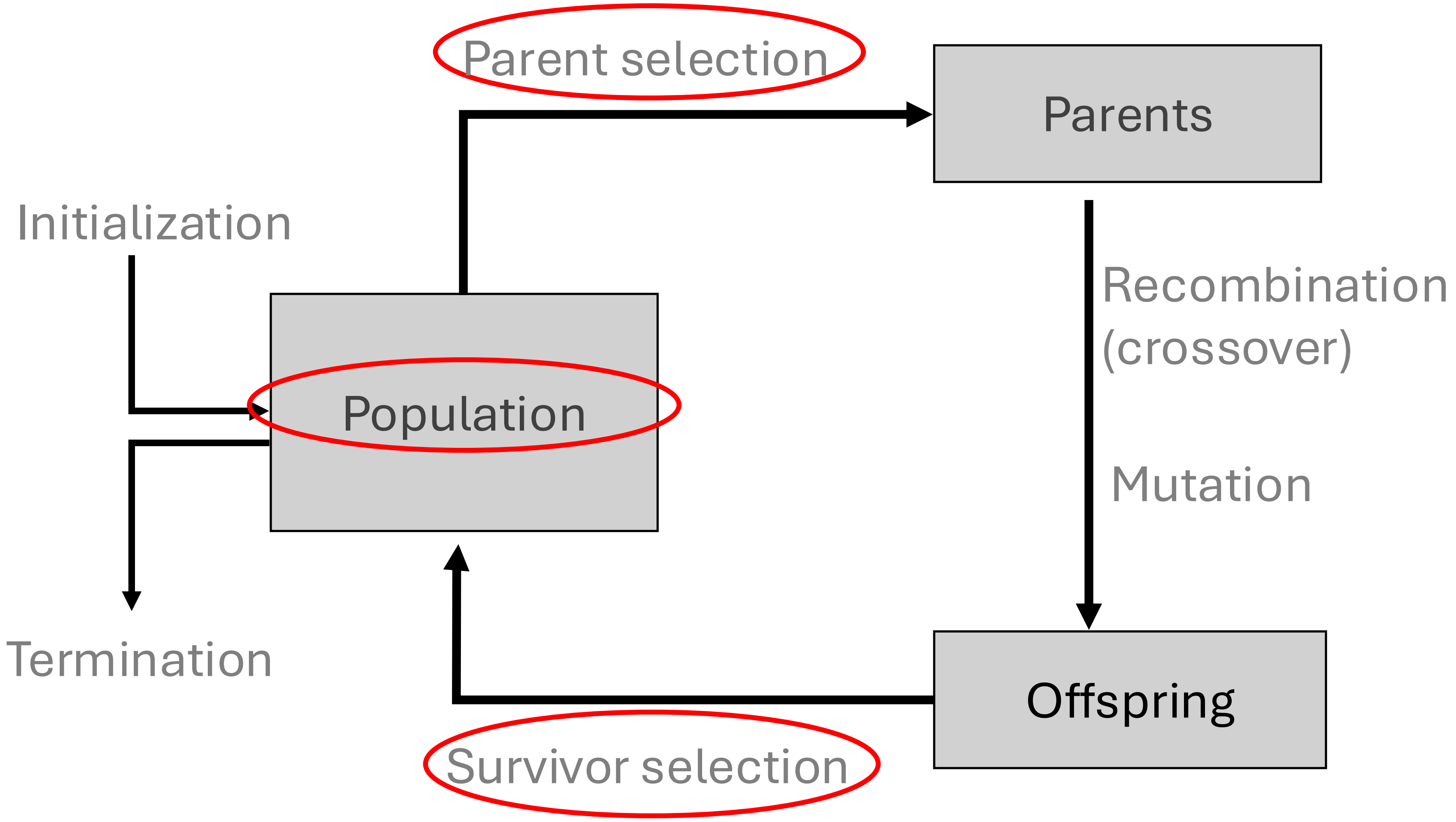


Chapter 5: Fitness, Selection and Population Management

- **Selection** is the second fundamental force for evolutionary systems
- Topics include:
  - Selection operators
  - Preserving diversity



Scheme of an EA: General scheme of EAs



Parent selection

Parents

Initialization

Recombination (crossover)

Population

Mutation

Termination

Offspring

Survivor selection

Population Management Models: Introduction

- Two different population management models exist:
  - **Generational model**
    - Each individual survives for exactly one generation
    - λ offspring are generated
    - The entire set of μ parents is replaced by μ offspring
  - **Steady-state model**
    - λ (< μ) parents are replaced by λ offspring
    - Generation Gap
      - The proportion of the population replaced
      - Parameter = 1.0 for G-GA, =λ/pop_size for SS-GA

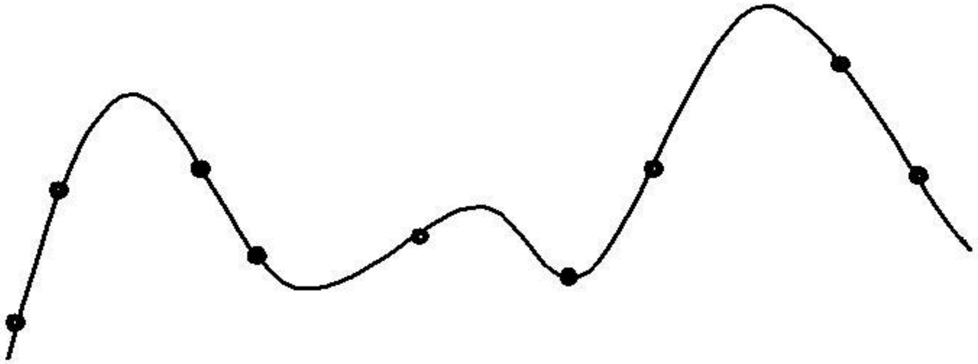
Selection

- Selection can occur in two places:
  - **Parent selection** (selects mating pairs)
  - **Survivor selection** (replaces population)
- Selection works on the population

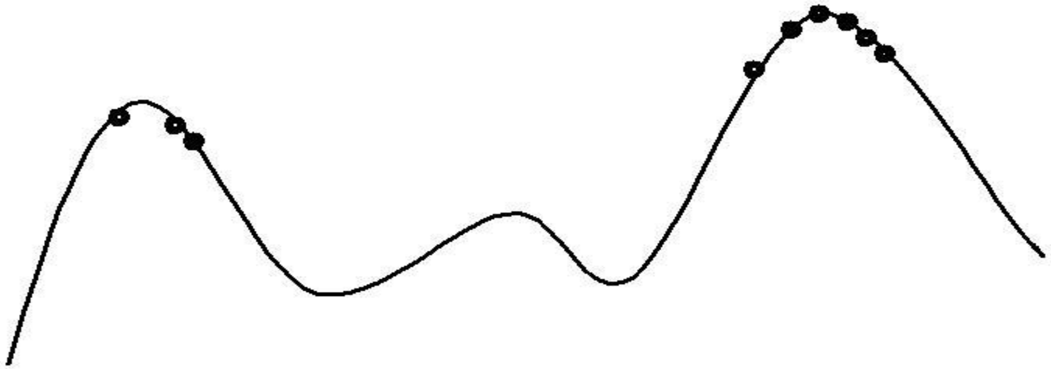
  -> Selection operators are **representation-independent** because they work on the fitness value
- **Selection pressure**: As selection pressure increases, fitter solutions are more likely to survive, or be chosen as parents

Why Not Always High Selection Pressure?

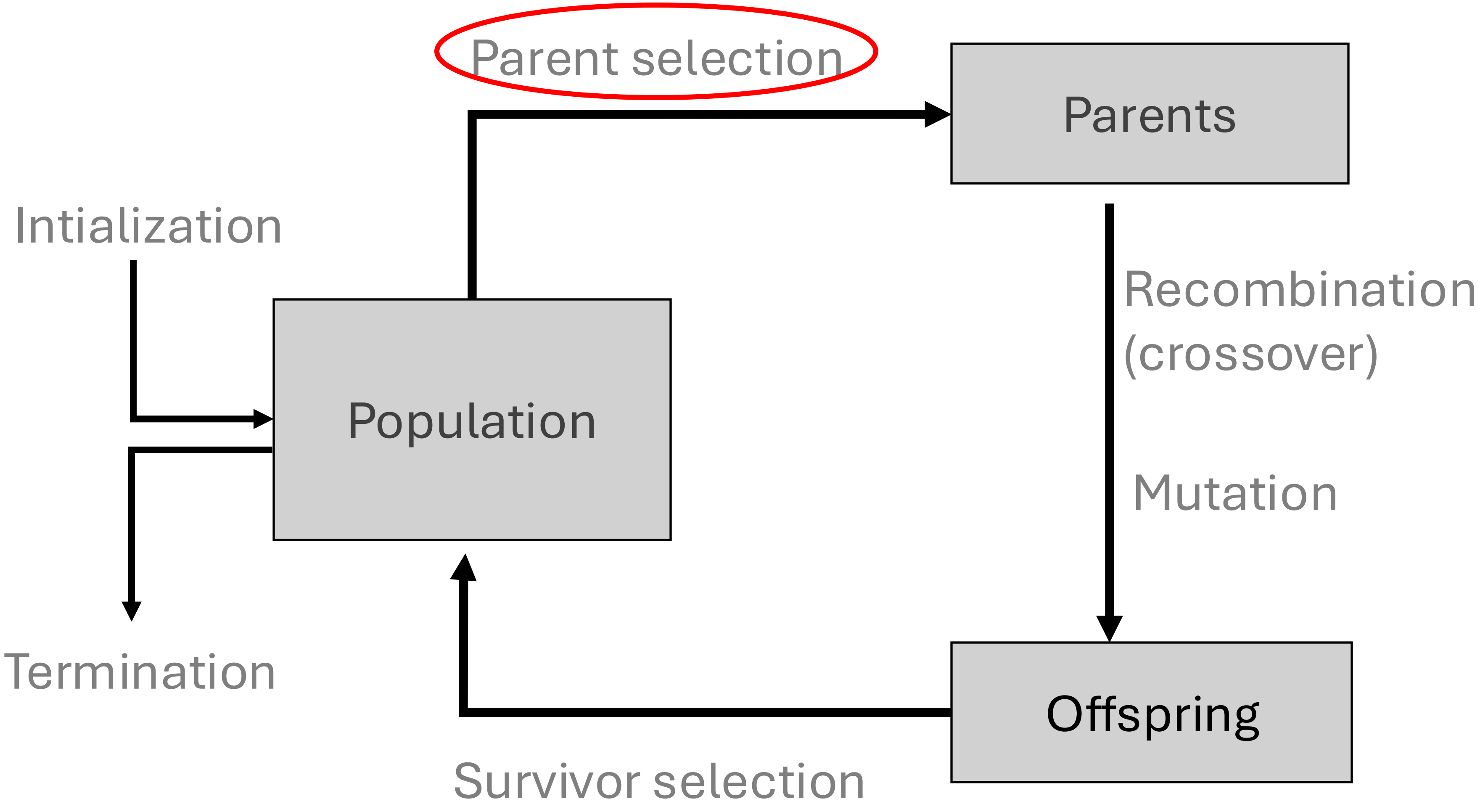
Exploration



Exploitation



Scheme of an EA: General scheme of EAs



Parent Selection:
Fitness-Proportionate Selection

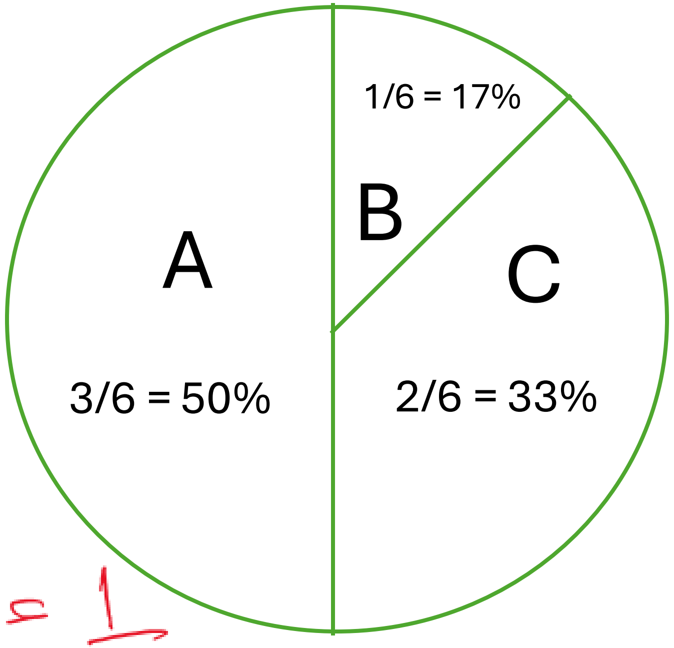
Example: roulette wheel selection

fitness(A) = 3

fitness(B) = 1

fitness(C) = 2

→



$$P(A) = \frac{3}{3+1+2} = \frac{3}{6} = \frac{1}{2}$$

Parent Selection:
Fitness-Proportionate Selection (FPS)

- Probability for individual *i* to be selected for mating in a population size μ with FPS is

$$P_{FPS}(i) = f_i \Big/ \overset{\mu}{\underset{j=1}{\square}} f_j$$

- Problems include
  - One highly fit member can rapidly take over if rest of population is much less fit: **Premature Convergence**
  - At end of runs when finesses are similar, **loss of selection pressure**

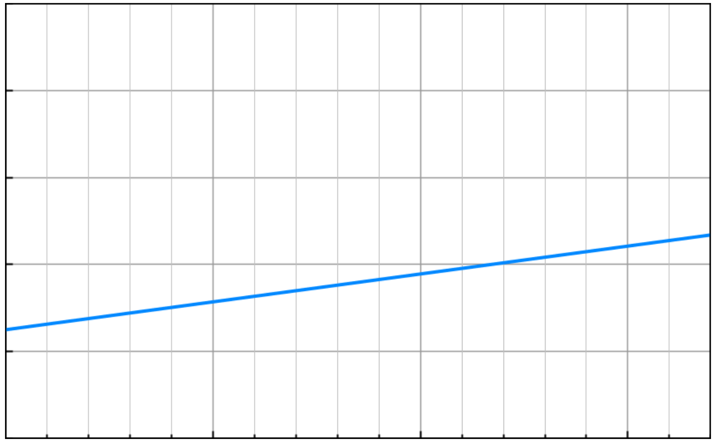
Parent Selection: Rank-based Selection

	1	2	3
fitness	100	90	110
Rank	1	2	0

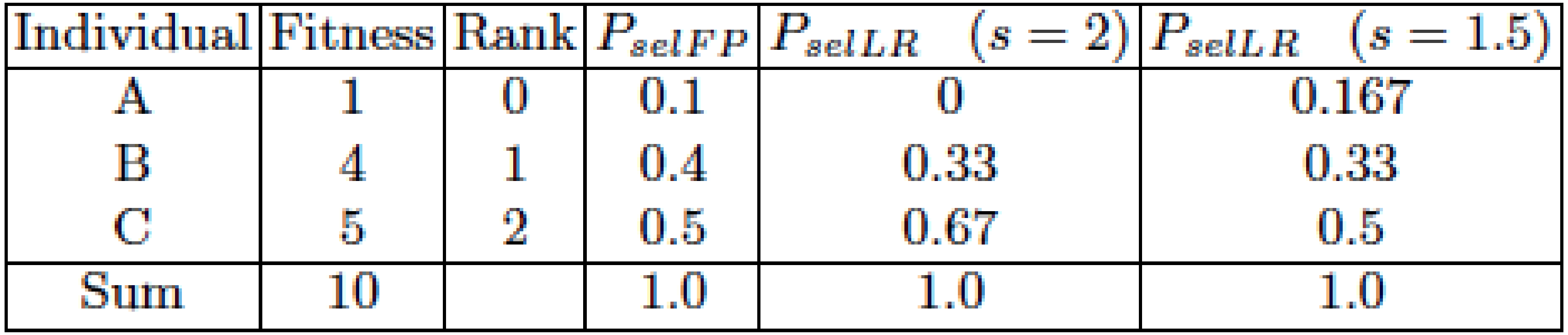
- Attempt to remove problems of FPS by basing selection probabilities on ***relative* rather than *absolute* fitness**
- **Rank population** according to fitness and then base selection probabilities on rank (fittest has rank *μ-1* and worst rank 0)
- This imposes a sorting overhead on the algorithm

Rank-based Selection: Linear Ranking

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$



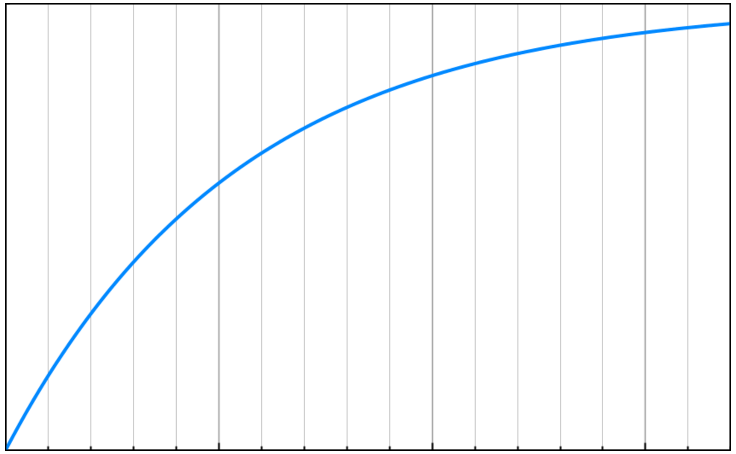
- Parameterized by factor *s:* 1 < *s* ≤ 2
    - Tunes selection pressure
- Simple 3 - member example



Individual	Fitness	Rank	P_{selFP}	$P_{selLR}\quad(s=2)$	$P_{selLR}\quad(s=1.5)$
A	1	0	0.1	0	0.167
B	4	1	0.4	0.33	0.33
C	5	2	0.5	0.67	0.5
Sum	10		1.0	1.0	1.0

Rank-based selection: Exponential Ranking

$$P_{\exp-rank}(i) = \frac{1-e^{-i}}{c}$$



- Linear Ranking is limited in selection pressure
- Exponential Ranking can allocate more than 2 copies to the fittest individual
- Normalize constant factor c according to population size

Parent Selection:
Tournament Selection (1/3)

- The methods above rely on **global population statistics**
  - This could be a **bottleneck, especially on parallel machines**, very large population
  - Relies on the presence of external fitness functions that might not exist, e.g. evolving game players

Parent Selection:
Tournament Selection (2/3)

The idea for a procedure using only local fitness information:

- Pick ***k*** **members at random,** then select the best of these
- **Repeat to select more** individuals

Parent Selection:
Tournament Selection (3/3)

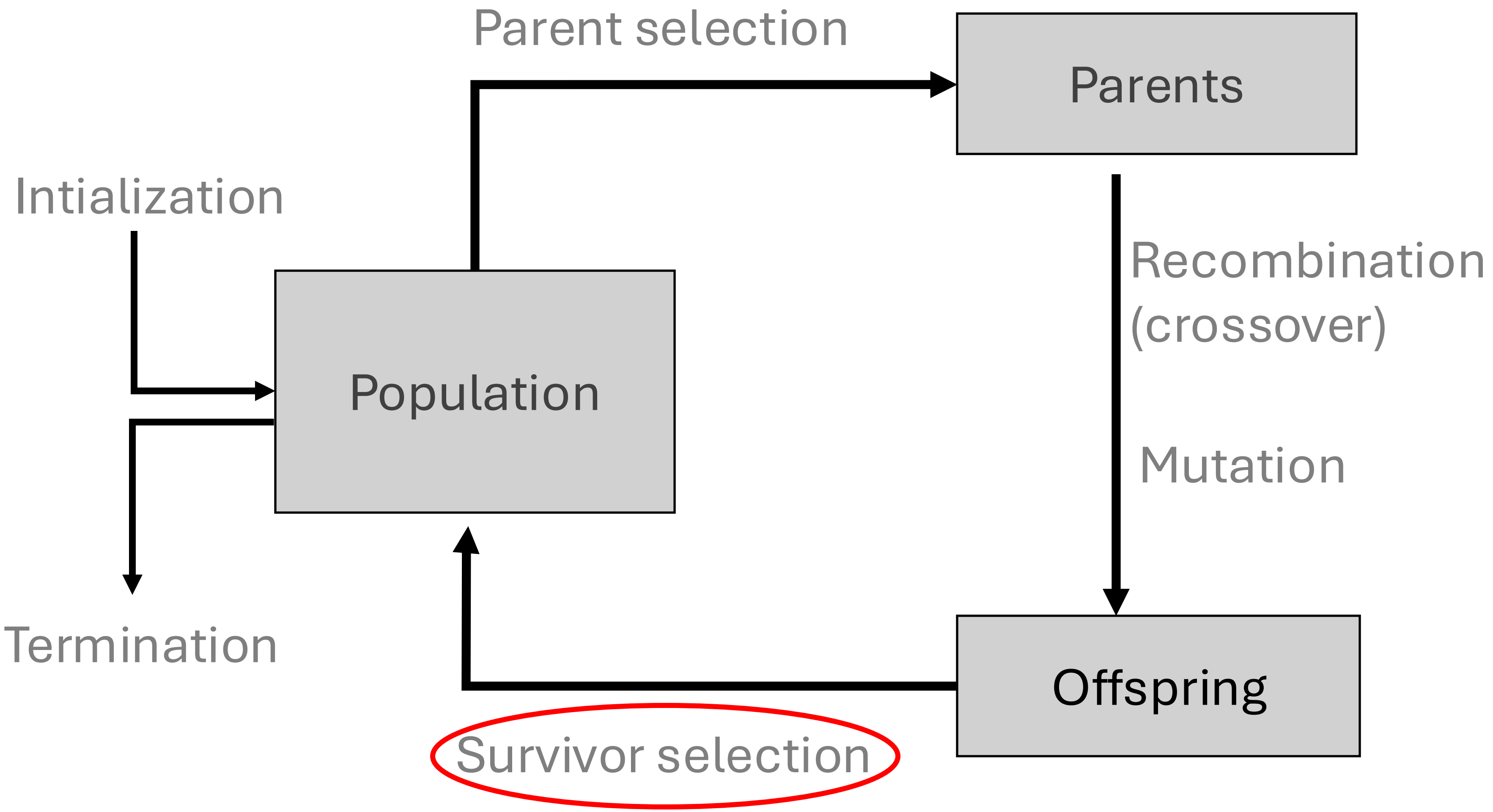
- Probability of selecting *i* will depend on:
  - Rank of *i*
  - Size of sample *k*
    - higher *k* increases selection pressure
  - Whether contestants are picked with replacement
    - Picking without replacement increases selection pressure
  - Whether fittest contestant always wins (deterministic) or this happens with probability *p*

Parent Selection: Uniform

$$P_{uniform}(i) = \frac{1}{\mu}$$

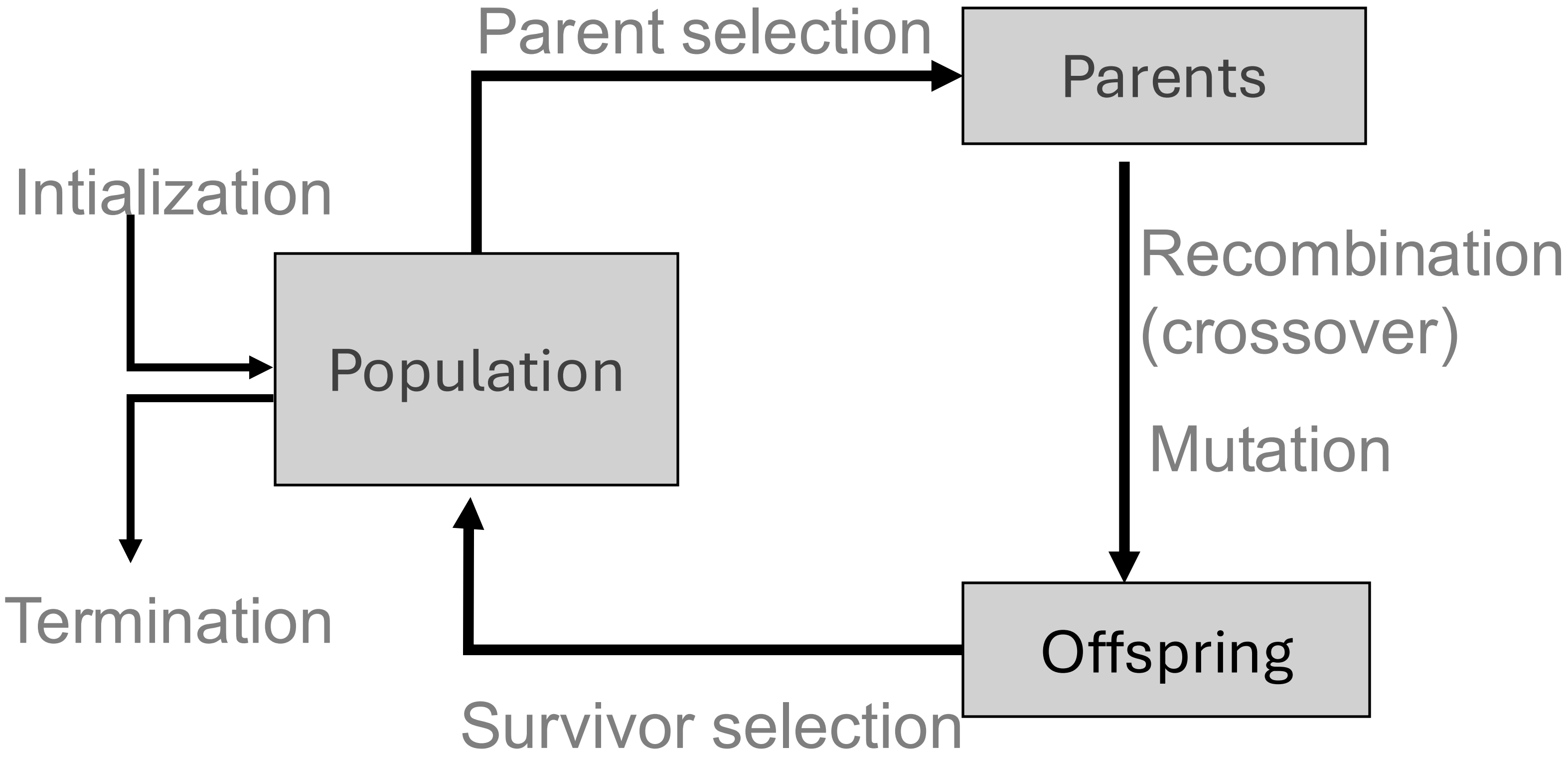
- Parents are selected by uniform random distribution whenever an operator needs one/some
- Uniform parent selection is unbiased - every individual has the **same probability** to be selected

Scheme of an EA:
General scheme of EAs



Survivor Selection (Replacement)

- From a set of μ old solutions and λ offspring: Select a set of μ individuals **forming the next generation**

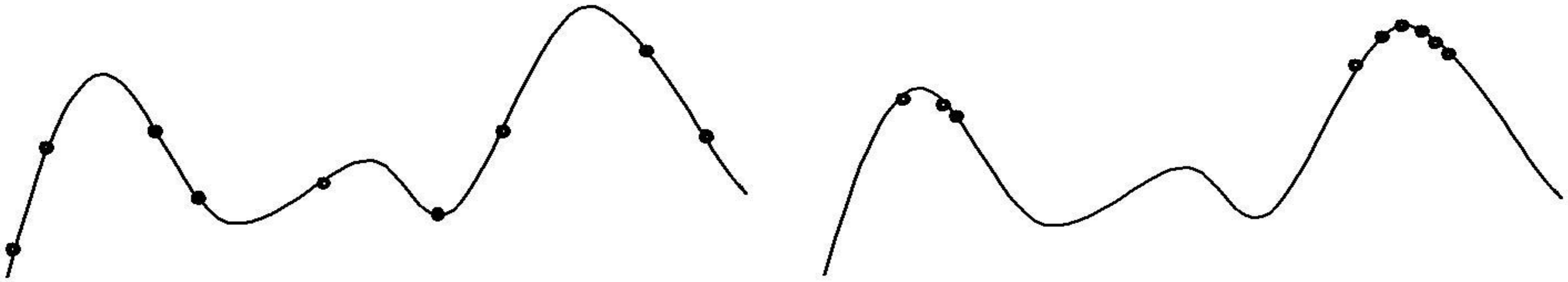


Fitness-based replacement – examples

- Elitism
  - Always **keep** at least one copy of **the N fittest solution(s)** so far
  - Widely used in most EA-variants
- **(μ,λ)-selection** (best candidates can be lost)
  - based on the set of **children only** ($\lambda > \mu$)
  - Choose the **best** μ offspring for the next generation
- **(μ+λ)-selection** (elitist strategy)
  - based on the set of **parents and children**
  - Choose the **best** μ individuals for the next generation
- (μ,λ)-selection may lose the best solution, but is better at leaving local optima

Multimodality

- Often, you might want to identify several possible peaks
- Different peaks may be different good ways to solve the problem.
- We therefore need methods to **preserve diversity** (instead of converging to one peak)



Approaches for Preserving Diversity: Introduction

- Explicit vs implicit:
- **Explicit** approaches
  - Make **similar individuals compete** for resources (**fitness**)
  - Make **similar individuals compete** with each other for **survival**

- **Implicit** approaches:
  - Impose an equivalent of **geographical separation**
  - Impose an equivalent of **speciation**

Explicit Approaches for Preserving Diversity: Fitness Sharing (1/2)

- Restricts the number of individuals within a given niche by "sharing" their fitness
- Need to set the size of the niche σ_{share} in either genotype or phenotype space
- run EA as normal but after each generation set

$$f'(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i,j))}$$

$$sh(d) = \begin{cases} 1 - d/\sigma & d \le \sigma \\ 0 & otherwise \end{cases}$$

Explicit Approaches for Preserving Diversity: Fitness Sharing (2/2)

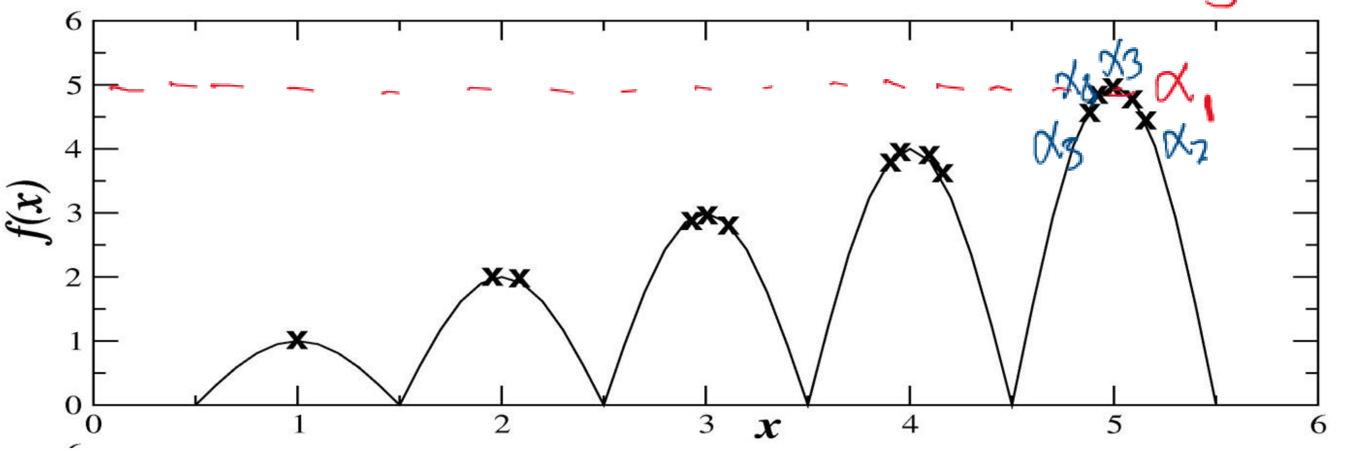
$$f'(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i,j))}$$

$$sh(d) = \begin{cases} 1 - d/\sigma & d \le \sigma \\ 0 & otherwise \end{cases}$$

$\sigma = 1$

$$f'(x) = \frac{5}{\sum_{j=1}^{\mu} 0 + 1 + 1 + 1 + 1 + 1}$$

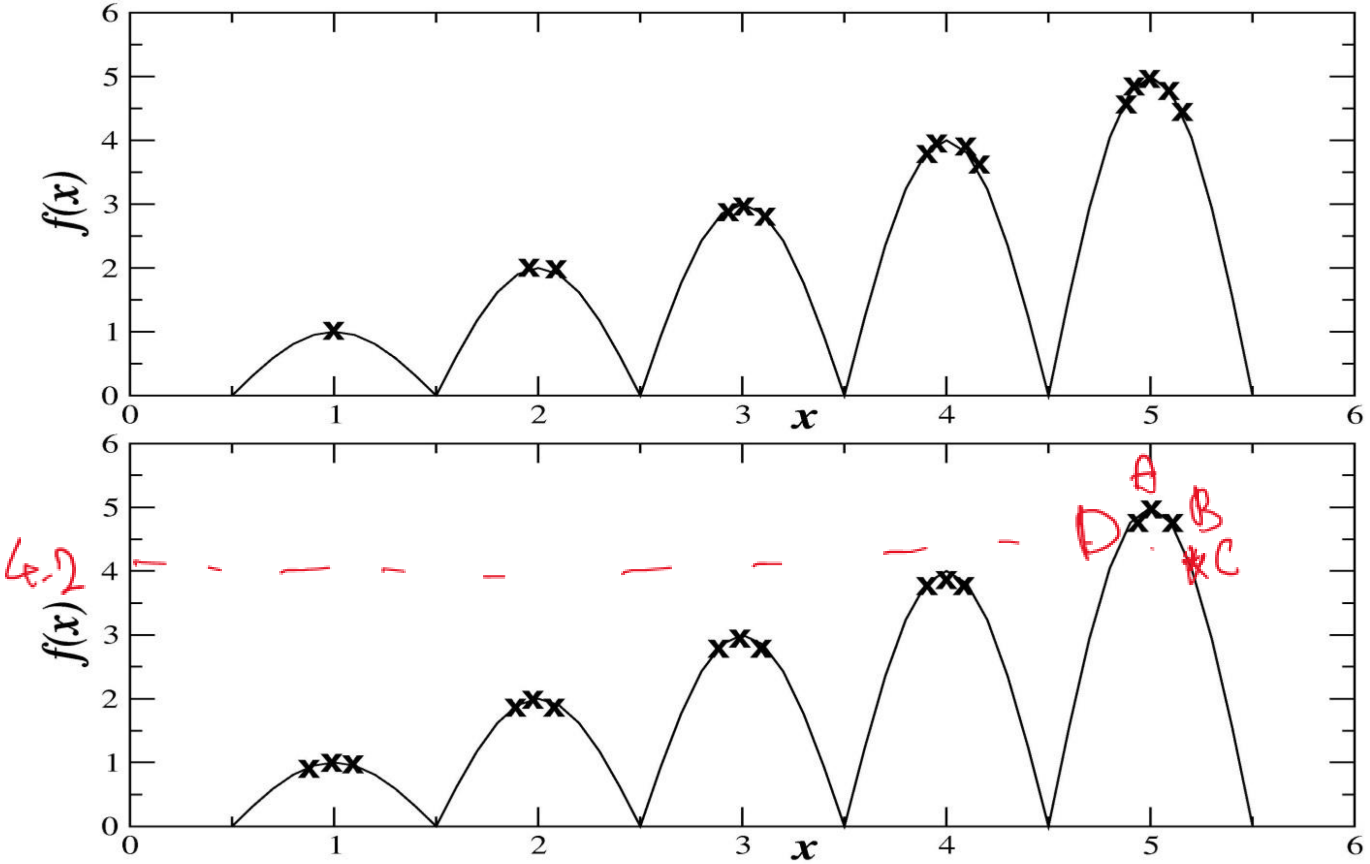
$$= \frac{5}{5} = 1$$



Explicit Approaches for Preserving Diversity: Crowding

- Idea: New individuals replace *similar* individuals
- Randomly shuffle and pair parents, produce 2 offspring
- Each offspring competes with their **nearest** parent for survival (using a distance measure)
- Result: Even distribution among niches.

Explicit Approaches for Preserving Diversity: Crowding vs Fitness sharing



Fitness Sharing

✓

$f(B) > \cancel{f(C)}$

✓ $f(A) > \cancel{f(D)}$

Crowding

Observe the number of individuals per niche

Implicit Approaches for Preserving Diversity: Automatic Speciation

tag

[1, 2, 3, 4] A

[1, 3, 2, 4] A

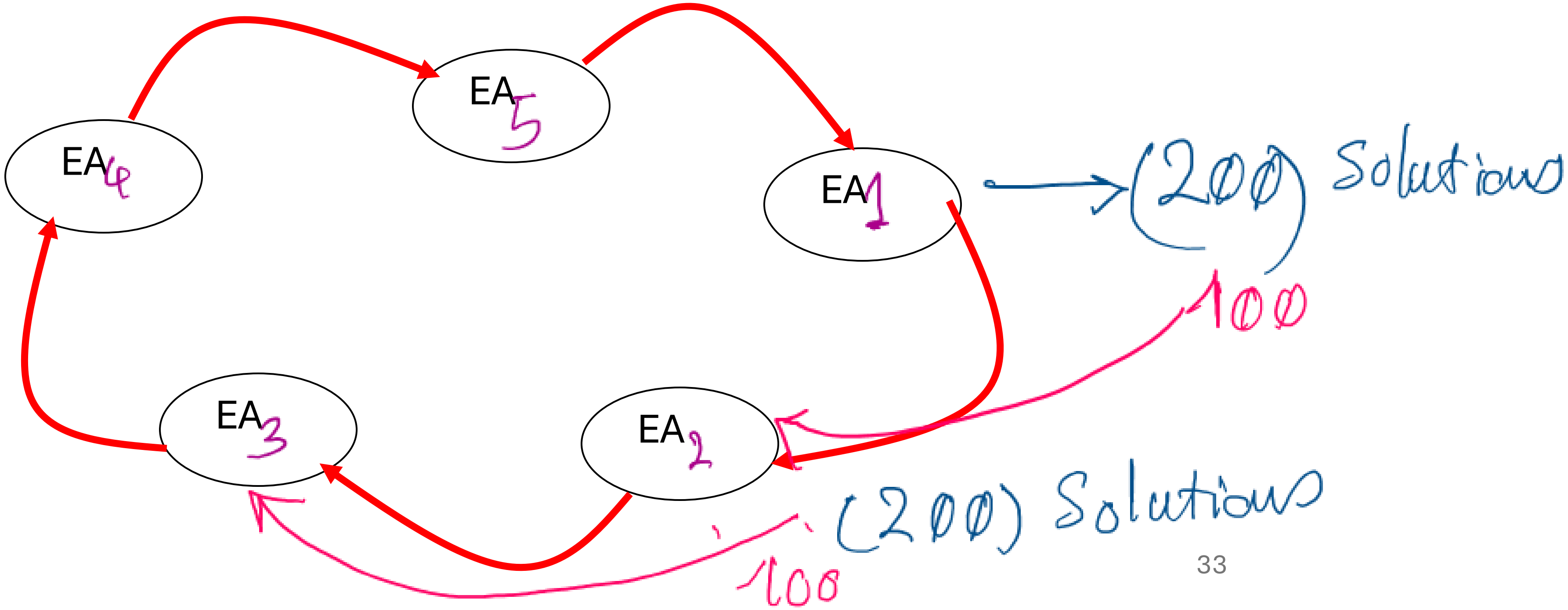
[2, 4, 1, 3] B

[2, 1, 4, 3] B

- Either only mate with genotypically / phenotypically similar members or
- Add species-tags to the genotype
  - initially randomly set
  - When selecting a partner for recombination, only pick members with a good match

Implicit Approaches for Preserving Diversity: Geographical Separation

- "Island" Model Parallel EA
- Periodic migration of individual solutions between populations



Implicit Approaches for Preserving Diversity: “Island” Model Parallel EAs

- Run multiple populations in parallel
- After a (usually fixed) number of generations (an ***Epoch***), exchange individuals with neighbours
- Repeat until ending criteria met
- Partially inspired by parallel/clustered systems

Island Model: Parameters

- How often to exchange individuals ?
  - too quick and all sub-populations converge to same solution
  - too slow and waste time
  - can do it adaptively (stop each pop when no improvement for (say) 25 generations)
- Operators can differ between the sub-populations

Real-world applications of GAs

- Engineering Design
  - Structural optimization; Control system design; Robotics
- Scheduling & Planning
  - Job shop scheduling; Timetabling; Vehicle routing problems (VRP)
- Finance & Economics
  - Portfolio optimization; Algorithmic trading; Economic modeling
- Game Development & Procedural Content
  - AI opponents; Level generation; Strategy evolution
- Creativity
  - Generative design; Evolving music; Visual art
- Industrial Design & Manufacturing
  - 3D printing; Tool path planning
- Cryptography and Security
  - Breaking codes or optimizing encryption algorithms; Evolving rule sets
- Telecommunications
  - Antenna design; Network routing

Real-world applications of GAs

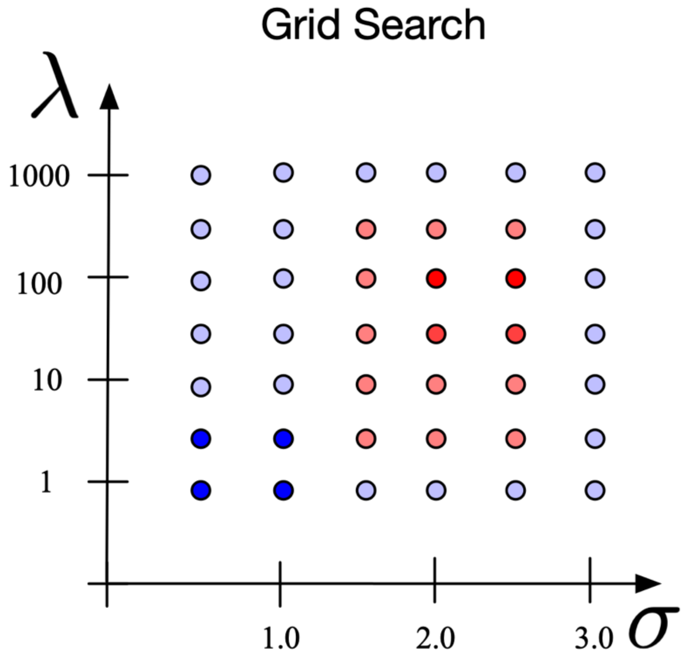
- Bioinformatics
    - Gene sequencing; Protein structure prediction; Drug discovery
- Machine Learning & AI
    - **Hyperparameter tuning**: GAs optimize settings (e.g., learning rate, architecture) for models like neural networks.
    - **Feature selection**: GAs find the best subset of features from large datasets.
    - **Evolving neural networks** (neuroevolution): GAs evolve architectures or weights (e.g., NEAT, genetic CNNs).

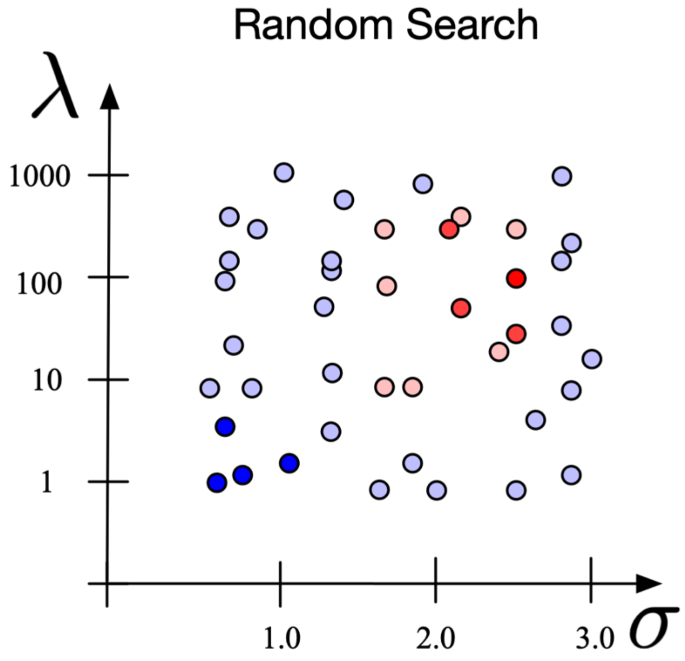
Genetic Algorithms for Hyperparameter Optimization

- GAs are commonly used in various ML methods to **tune hyperparameters**
- Hyperparameters govern the model's performance.
- Manual tuning can be inefficient and time-consuming.
  - Such as *grid search* or *random search*
- GAs provide an effective way to automate this process by searching for optimal hyperparameter combinations—e.g.,applying **Differential Evolution**
- Popular ML methods and techniques where GAs are employed for hyperparameter optimization:
  - Neural Networks/Deep Learning, Support Vector Machines (SVM), 3. Decision Trees / Random Forest, Kernel Ridge Regression, k- Nearest Neighbors (k-NN), Clustering Algorithms (e.g., K-Means, DBSCAN)

Hyperparameter tuning techniques?

- Always find the best-performing combination in the grid, but not the overall best
- Can be computationally more expensive


Grid Search


Random Search

- It can lead to good solutions, but it’s not guaranteed
- Less computationally demanding

- When training relatively small models
- Small number of Hyperparameters
- With narrow range of values

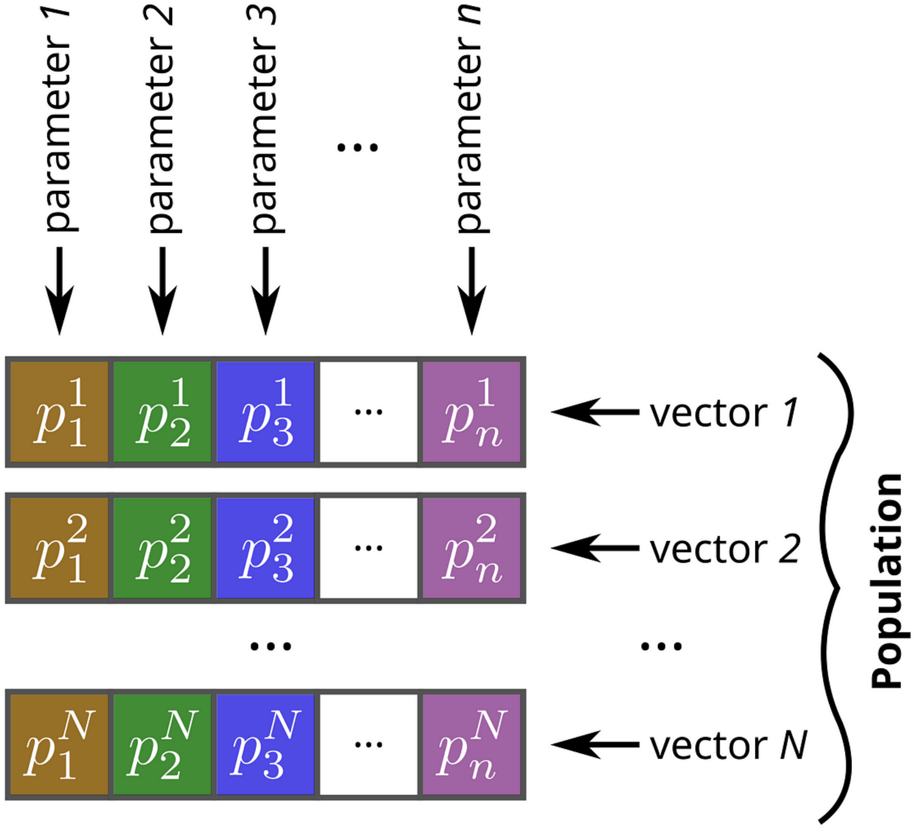
- When training relatively complex models
- Many Hyperparameters
- With wider range of values

Why GAs for Hyperparameter Tuning?

- **Exploration and Exploitation Balance**: GAs maintain a good balance between exploring new solutions and exploiting known good solutions, avoiding the risk of getting stuck in local optima.
- **Flexibility**: GAs can handle various types of hyperparameters, including discrete, continuous, and categorical variables.
- **Global Search**: Compared to grid search or random search, GAs offer a more global exploration of the hyperparameter space, making them suitable for complex or non-convex optimization problems.
- **Parallelizable**: GAs are inherently parallelizable, meaning they can be easily distributed across multiple processors, speeding up the optimization process.
- **Efficient**: They reduce the computational expense of grid or random search.

Overview of Differential Evolution Algorithm for Hyperparameter Tuning (1/2)

- **Differential Evolution** is a type of genetic algorithm that uses a population of solutions (vectors) to evolve the best parameters and iteratively optimizes a function by evolving a population of candidate solutions.
- Each vector contains **parameters** that represent the hyperparameters of the model.


parameter 1
parameter 2
parameter 3
...
parameter *n*
p_1^1 p_2^1 p_3^1 ... p_n^1
vector *1*
p_1^2 p_2^2 p_3^2 ... p_n^2
vector *2*
...
...
p_1^N p_2^N p_3^N ... p_n^N
vector *N*
Population

Overview of Differential Evolution Algorithm for Hyperparameter Tuning (2/2)

- **Initialization**: Create an initial population of vectors with random parameter values within predefined boundaries. The size of the population is NP (number of vectors).
- **Evaluation**: Evaluate the fitness of each vector in the population by calculating its function value. (e.g., mean squared errors on a validation set)
- For each vector in the population, Iterate until convergence is achieved (**repeat**)

1. **Mutation**: Build a new vector by mutating the parameters of existing vectors.
   - The **best1bin strategy** is commonly used:
     - The mutant parameter is a variation of the best vector plus a mutation rate (F) times the difference between two other random vectors.

$$p_i^{mut} = p_i^{best} + F \cdot (p_i^{r_1} - p_i^{r_2})$$

2. **Recombination**: Combine parameters from the current vector and mutant vector to create a trial vector.
   - For each parameter, a random uniform number R is generated.
   - If R < recombination rate, the mutant parameter is selected; otherwise, the current parameter is retained.
3. **Replacement**:
   - Evaluate the fitness of the trial vector.
   - If the trial vector has a better fitness than the current vector, it replaces the current vector in the population.

Further Reading

Differential Evolution Algorithm for Hyperparameter Tuning

$0 \le P_1 \le 5$

$0 \le P_2 \le 2$

	P_1	P_2	f
V_1	0	2	0.8
best $\underline{\underline{V_2}}$	3	1	0.9
V_3	4	1	0.87
V_4	0	1	0.75
V_5	5	0	0.82

Randomly chosen $0 < R < 1$

$v_1^{mut} = ?$ $\quad F = 0.1$

$$P_1^{mut} = P_1^{best} + F(P_1^{3} - P_1^{5})$$

$$= 3 + 0.1(\overset{-0.1}{4 \cancel{-5}}) = 2.9$$

$$P_2^{mut} = 1 + 0.1(1 - 0) = 1.1$$

P_1^{mut}	P_2^{mut}

0	1.1

trial vecto

Randomly Recomb. Rate = 0.2

Randomly

0.1	0.5

$0.1 < 0.2$

$0.5 > 0.2$

0	2

✗ $f_{v_1} = 0.85$ ⇒ ✓

⇒

0	1.1

$f_t = 0.85$

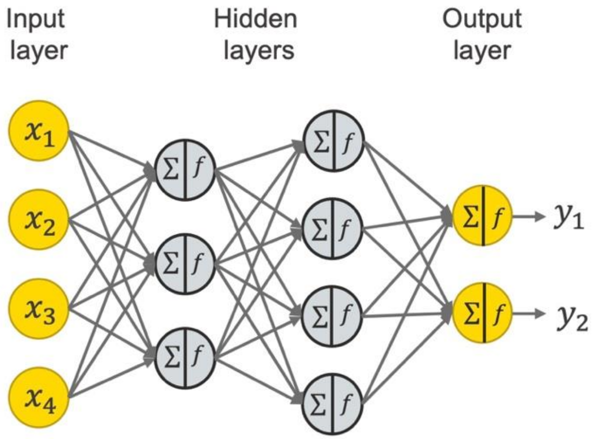
Using Gas for Weight Optimization in NN (1/3)

- Neural networks are traditionally trained using **gradient descent**, which adjusts weights based on error.
- Genetic algorithms can be used to **encode neural network weights** as a set of strings.
- **Fitness Function**: Measures performance using **sum-of-squares error**, similar to how gradient descent minimizes error.
- **Drawbacks**:
  - Local information at each node is discarded and reduced to a single fitness value.
  - GA-based optimization ignores **gradient information**, losing a valuable source of guidance.
  - Results can be good, but this approach loses some valuable information compared to gradient descent.

Machine Learning, by Stephen Marsland, Chapter 10

Evolving Neural Network Topology with GAs (2/3)

- **Topology Optimization**: GAs are more effectively applied to evolve the structure or topology of the neural network, such as:
    - Adding or deleting neurons.
    - Adding or deleting weight connections.
- **Mutation Operators**:
    - **Delete a neuron**: Simplifies the network.
    - **Delete a weight connection**: Reduces complexity.
    - **Add a neuron**: Increases complexity.
    - **Add a connection**: Enhances inter-neuron communication.
- Deletion operations bias the learning toward **simpler networks**. GAs provide an automated way to explore different network architectures instead of manually trying different structures.



Machine Learning, by Stephen Marsland, Chapter 10

Neuroevolution (3/3)

- ***Neuroevolution*** merges genetic algorithms with neural networks.
- Iterative process of improving neural networks through generations.
- NEAT (Neuroevolution of Augmented Topologies) is a specific algorithm that evolves both the architecture and weights of neural networks.
  - It starts with simple networks and gradually increases complexity, allowing the emergence of efficient architectures.
  - Particularly useful for tasks requiring complex decision-making and adaptation.

Further Reading

Feature Selection Using Genetic Algorithm

- The feature selection methods”
  - **Filter methods:** Rank features using model-agnostic stats, then pick top ones. Examples: correlation. Fast, scalable, ignores the downstream model.
  - **Wrapper methods:** Search subsets by training a model and using its validation score as the objective. Examples: genetic algorithm wrappers. More accurate, but slower and prone to overfitting.
  - **Embedded methods:** The model selects features during training via its own regularization/structure.
    Examples: L1/Lasso/Elastic Net (weights shrink to zero). Efficient and usually robust.

GA for Feature Selection — Wrapper Approach

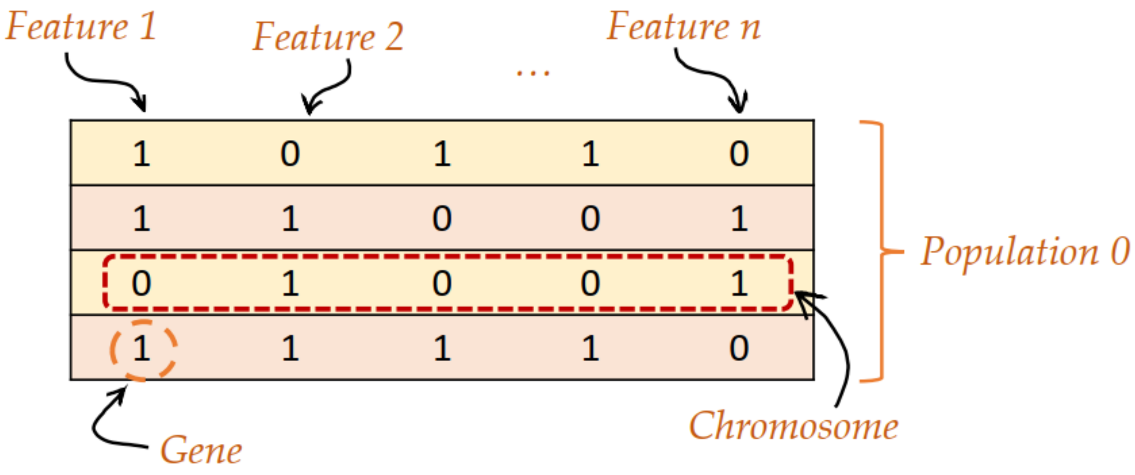
- Use GA to search subsets of features
- GA is integrated with a classifier/Regressor/estimator (e.g., the random forest classifier) to optimize the feature subset selection.
- Fitness = validation score of the classifier using only the selected features. (e.g., accuracy — but pick the metric that fits the problem)

Further Reading

Encoding & Population

- Binary chromosome (1 = include, 0 = exclude).
- Initialize random population under size constraints.
- Keep an elite fraction unchanged for the next generation.
- Constrain subset size with min/max #features.

1. Create initial population

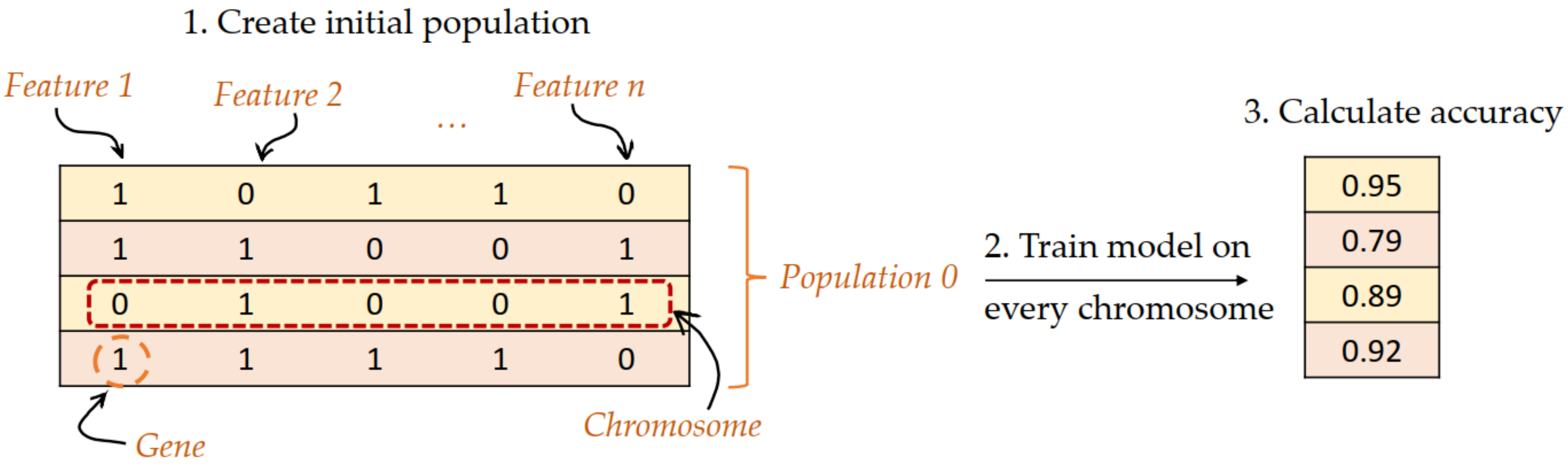


1	0	1	1	0
1	1	0	0	1
0	1	0	0	1
1	1	1	1	0

Further Reading/source

Evaluate & Select

- Train estimator per chromosome; compute fitness.
- Roulette-wheel selection picks parents by fitness.
- Elitism preserves top performers



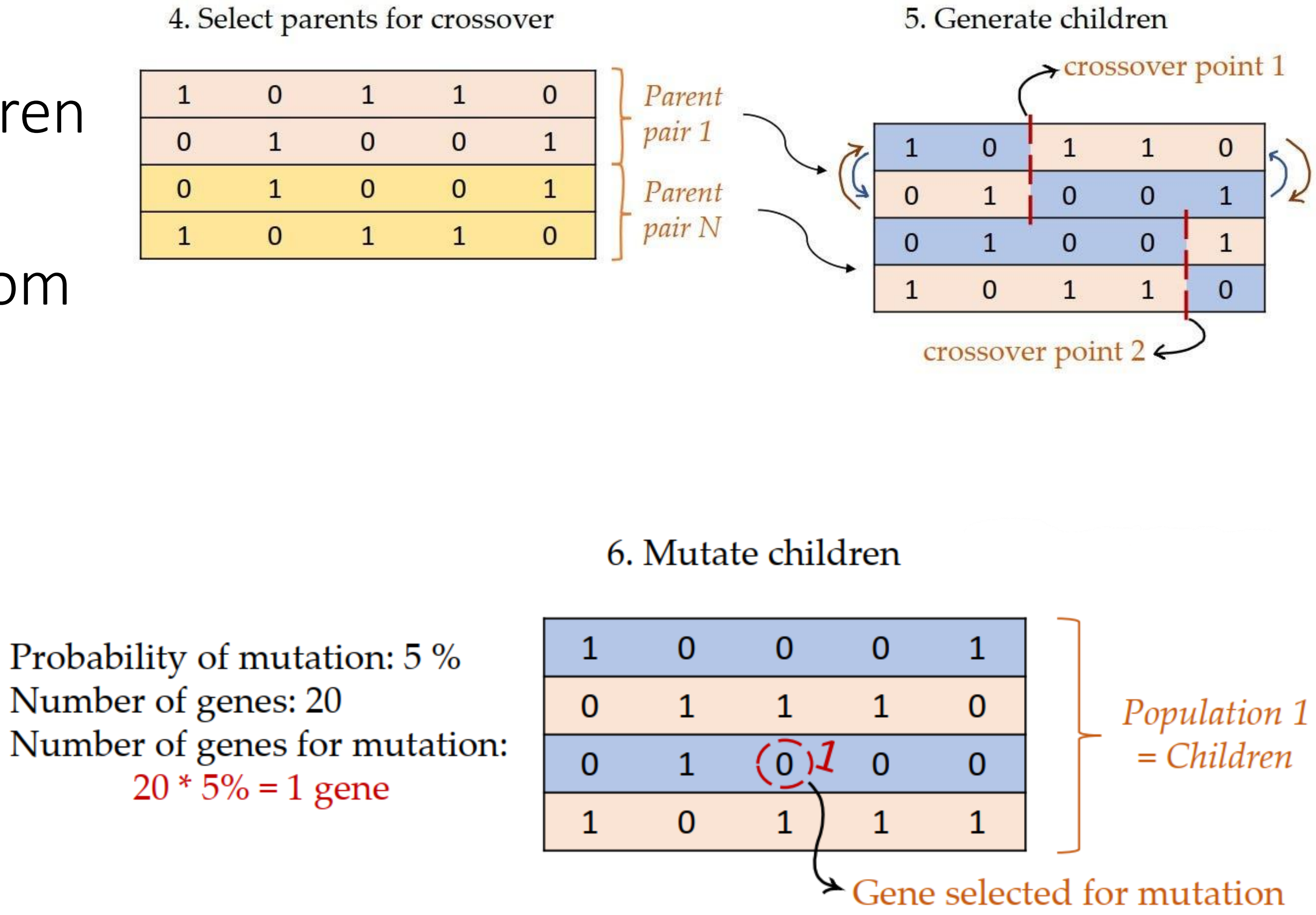
1	0	1	1	0
1	1	0	0	1
0	1	0	0	1
1	1	1	1	0

0.95
0.79
0.89
0.92

Further Reading/source

Recombine & Mutate

- One-point crossover creates children from parents.
- Mutation (e.g., p=0.05) flips random genes for diversity.
- Enforce min/max features after changes.



4. Select parents for crossover

1	0	1	1	0
0	1	0	0	1
0	1	0	0	1
1	0	1	1	0

5. Generate children

1	0	1	1	0
0	1	0	0	1
0	1	0	0	1
1	0	1	1	0

6. Mutate children

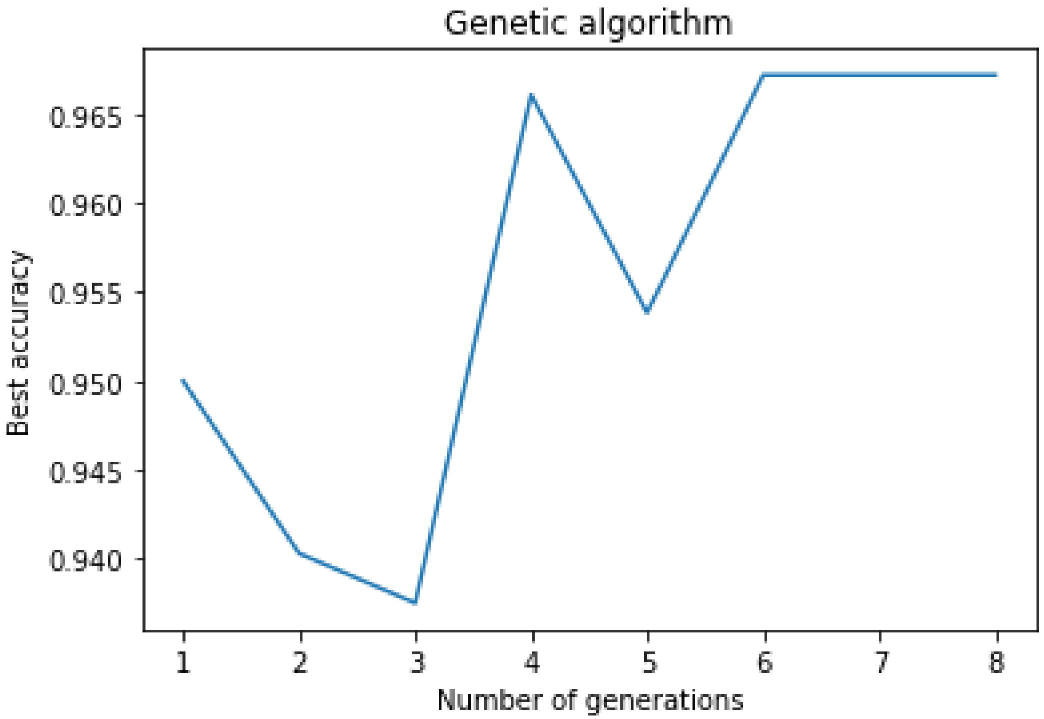
Probability of mutation: 5 %
Number of genes: 20
Number of genes for mutation:
20 * 5% = 1 gene

1	0	0	0	1
0	1	1	1	0
0	1	0 1	0	0
1	0	1	1	1

Further Reading/source

Iterate & Stop

- Next generation = elites + children; re-evaluate.
- Track best fitness; stop at max generations or the desired level of solution quality .
- Return the best chromosome and feature names.



Further Reading/source

Limitations of Evolutionary Algorithms (1/2)

- **Slow Convergence/Computational Cost** :

  GAs can be **slow**, especially after reaching a local maximum. It may take a long time to escape and find a better solution.

- **Fitness Landscape**

  Without knowing the **fitness landscape**, it's difficult to gauge how well the GA is performing.

- **Difficult to Analyze**

  The behavior of GAs is hard to analyze and predict. we cannot guarantee that the algorithm will converge at all

  - It's hard to prove that the GA will converge to the optimal solution.

- **Black Box Approach**

  GAs are often treated as a black box, which makes it difficult to improve or interpret the results.

Limitations of Evolutionary Algorithms (1/2)

- **Difficulties in Parameter Tuning**
  - EAs have several hyperparameters (e.g., *population size*, *mutation rate*, *crossover rate*) that significantly impact their performance.
  - Incorrect hyperparameter choices can lead to poor convergence, premature convergence, or excessively slow search.
- **Brittle Representation**
  - Finding a suitable representation for complex problems can be challenging and can make or break the performance of the EA.
- **Fitness Function Design**
  - Designing a good fitness function is often non-trivial and problem-specific, making EAs difficult to apply in certain cases.
- **Not Applicable everywhere?**
  - Particularly when the **fitness landscape** is not continuous

Concluding Insights: Evolutionary Algorithms

- How unrealistic are Evolutionary Algorithms as representations of biological evolution?

- Are computer scientists truly inspired by evolutionary theory?