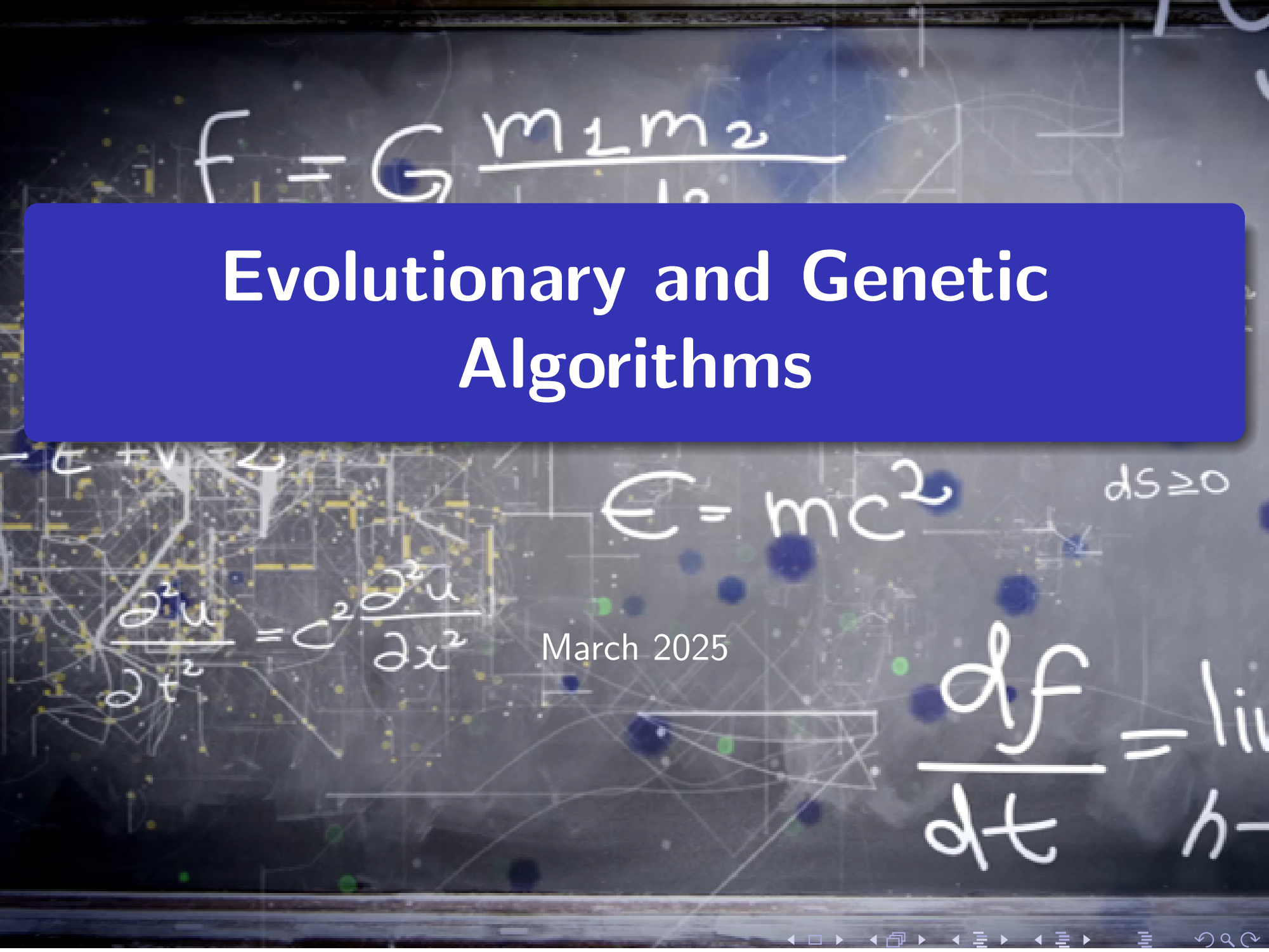


Evolutionary and Genetic Algorithms

March 2025

Topics to be discussed

Part I. Introduction

1. Introduction to Artificial intelligence

Part II. Search and optimisation.

2. Search - basic approaches
3. Search - optimisation
4. Two-player deterministic games
5. Evolutionary and genetic algorithms

Part III. Machine learning and data analysis.

6. Regression, classification and clustering (Part I & II)
8. Artificial neural networks
9. Bayesian models
10. Reinforcement Learning

Part IV. Logic, Inference, Knowledge Representation

11. Propositional logic and predicate logic
12. Knowledge Representation

Part V. AI in Action: Language, Vision

13. AI in Natural language processing
14. AI in Vision and Graphics

Part VI. Summary

15. AI engineering, Explainable AI, Ethics,

Overview

- Introduction to evolutionary and genetic algorithms
- Key concepts in evolutionary algorithms
- Basic genetic algorithm operations
- Fitness functions and selection methods
- Examples of applications of evolutionary and genetic algorithms

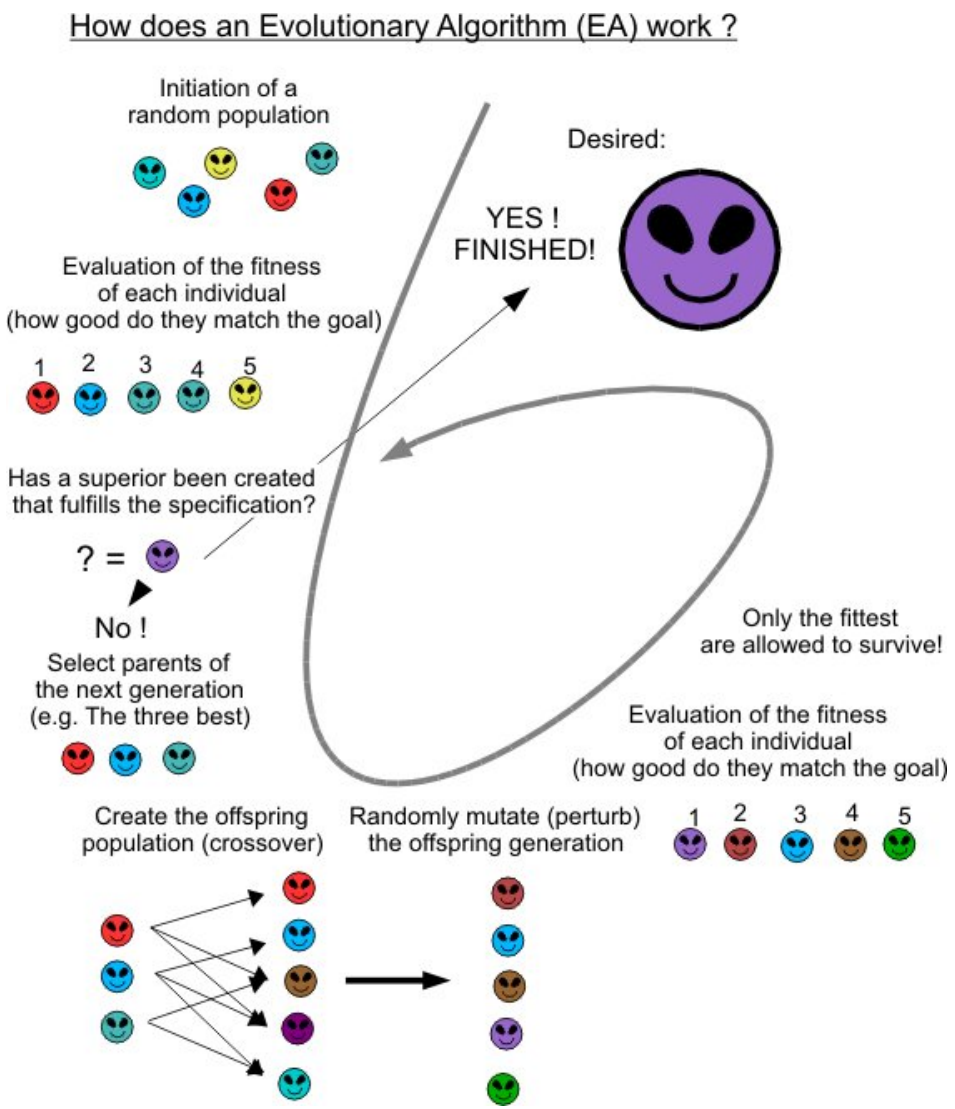
Introduction

- Evolutionary and genetic algorithms are a class of optimization algorithms that are based on the principles of natural selection and genetics.
- The origins of these algorithms can be traced back to the work of Charles Darwin, who proposed the theory of evolution by natural selection in the 1850s.
- In the 1960s and 1970s, researchers began to develop algorithms that were inspired by Darwin's theory, including evolutionary algorithms and genetic algorithms.
- Today, these algorithms are widely used in a variety of fields, including engineering, economics, and computer science.

History of Genetic Algorithms (selected dates)

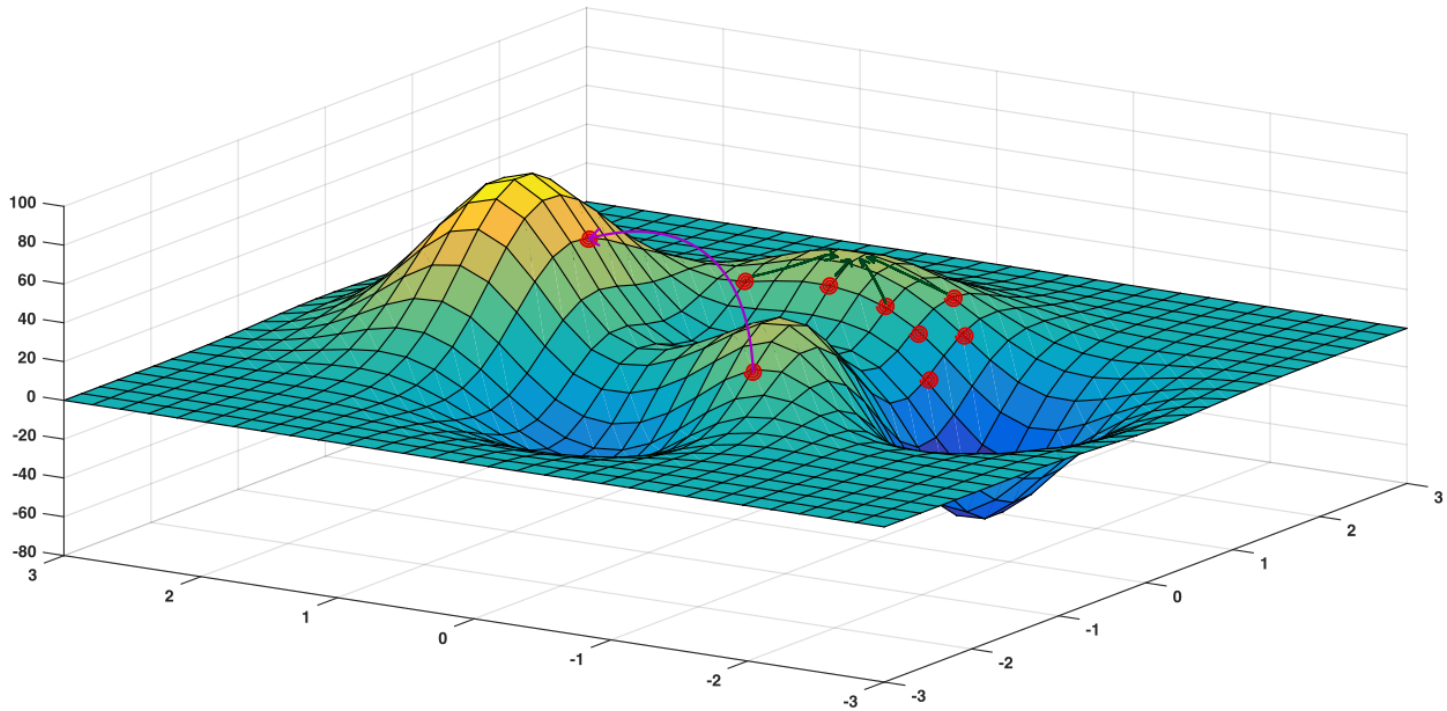
- **1950**, Alan Turing proposed an idea of simulating the evolution process to solve complex problems.
- **mid-1960s**, John Henry Holland introduced Genetic Algorithms (GA) as a method for optimization inspired by biological evolution.
- **1975**, John Koza introduced Genetic Programming (GP), which uses GAs to evolve computer programs.
- **1980s**, David E. Goldberg introduced the concept of niching in GAs, allowing for multiple solutions to coexist.
- **1992**, Zbigniew Michalewicz introduced the concept of constraint handling in GAs.
- **2002**, Carlos Coello Coello introduced the concept of coevolutionary Genetic Algorithms (cGAs).
- **2014**, the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) was proposed, which has become a popular method for continuous optimization.

Evolutionary Algorithms Key Idea



https://www.kip.uni-heidelberg.de/vision/previous-projects/evolvable-hardware/evolutionary-algorithms/

Example: Optimization with Genetic Algorithm



- Fitness function - the function to find the maximum of
- Individual - a point in space
- Genome - every respective point in space, the genome will just be a collection of real numbers (one for each dimension)

Source: https://ai.stackexchange.com/questions/8168/how-do-i-optimize-a-specific-function-using-a-genetic-algorithm

Diverse Applications of Genetic Algorithms (I)

https://www.turing.com/kb/genetic-algorithm-applications-in-ml

Diverse Applications of Genetic Algorithms (II)

- **Neural Networks:** GAs optimize neural network structures for tasks like parameter selection, enhancing AI's capability in complex problem-solving and decision-making processes.
- **Image Processing:** In tasks such as image segmentation and feature extraction, GAs offer solutions to traditionally intractable optimization challenges, benefiting medical imaging and satellite imagery analysis.
- **Wireless Sensor Networks (WSN):** GAs optimize sensor placements and network protocols, crucial for environmental monitoring and enhancing smart city infrastructures.
- **Mechanical Engineering Design:** From optimizing aircraft wing designs to enhancing manufacturing processes, GAs drive innovation in engineering by finding solutions that balance multiple design criteria.
- **Financial Markets:** GAs assist in discovering optimal trading strategies by evaluating a vast array of potential market scenarios.
- **Robotics:** GAs are instrumental in developing efficient navigation strategies, allowing robots to adapt to dynamic environments in industrial automation and service robotics.

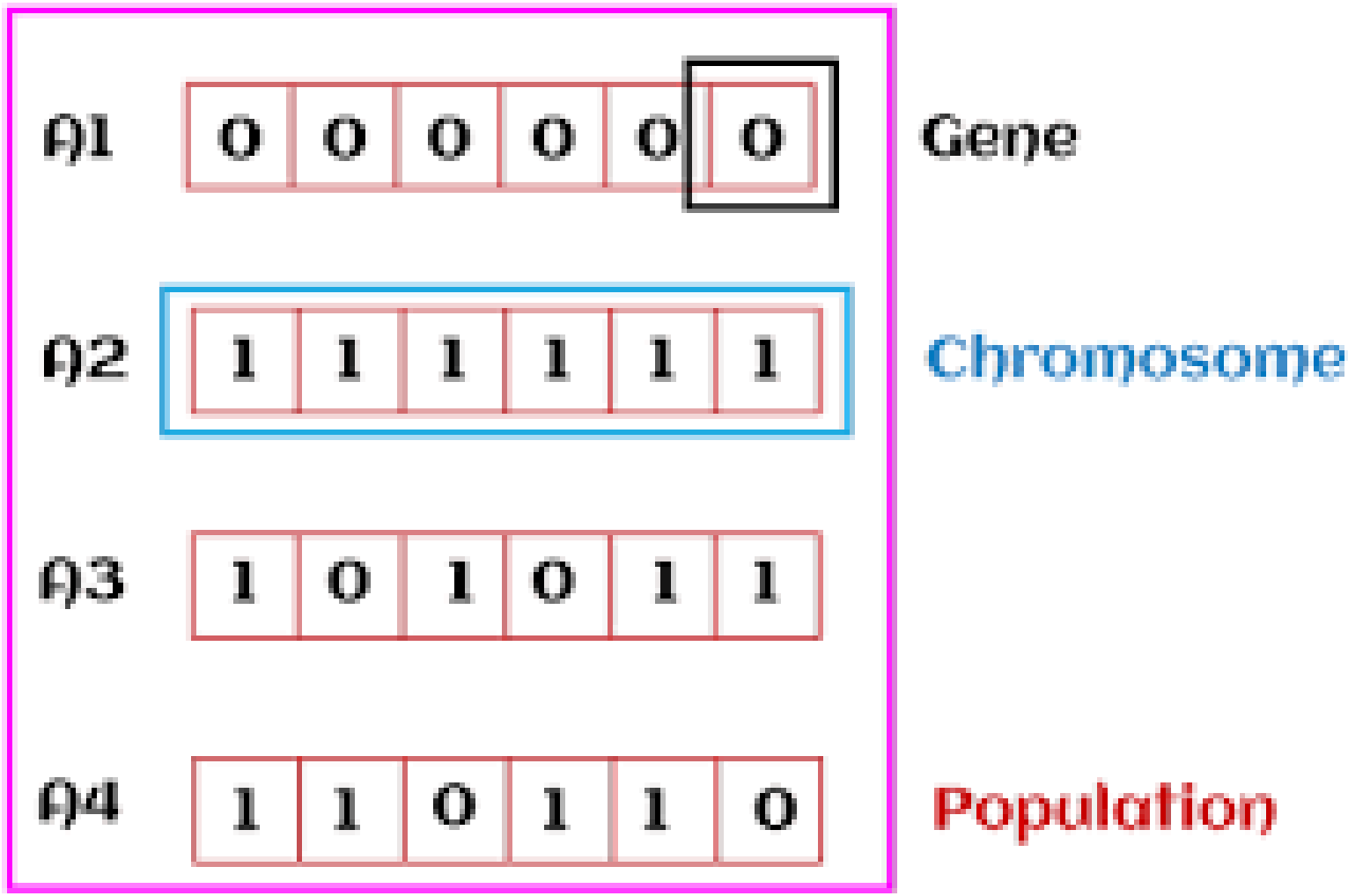
Diverse Applications of Genetic Algorithms (III)

- **Economics:** GAs model complex economic dynamics, aiding in the understanding of market behaviors and the optimization of resource distribution strategies.
- **Manufacturing Systems:** By optimizing production plans and operational parameters, GAs contribute to reducing costs and improving efficiency in manufacturing environments.
- **Supply Chain Management:** GAs optimize logistics, from inventory levels to distribution routes, ensuring timely delivery of goods while minimizing costs.
- **Environmental Planning:** In urban and environmental planning, GAs help balance developmental needs with sustainability goals, optimizing land use and resource management.
- **Energy Systems / Intelligent Grids:** GAs are used to optimize the operation and distribution of renewable energy sources, enhancing the efficiency of grids and reducing environmental impact.
- **Bioinformatics:** Through the analysis of genetic data and the modeling of evolutionary processes, GAs support advancements in drug discovery and genetic research.

Why Evolutionary and Genetic Algorithms

- Traditional optimization algorithms, such as gradient descent and linear programming, are often not effective for complex, nonlinear optimization problems.
- Evolutionary and genetic algorithms were developed as a way to address these types of problems by mimicking the process of natural selection.
- These algorithms are particularly effective for problems that involve large search spaces or have multiple, conflicting objectives.
- Additionally, evolutionary and genetic algorithms can be used to find approximate solutions to problems that are difficult to solve exactly.
- These algorithms were developed as a way to address complex optimization problems that are difficult to solve using traditional optimization algorithms.

Genes, chromosomes and populations

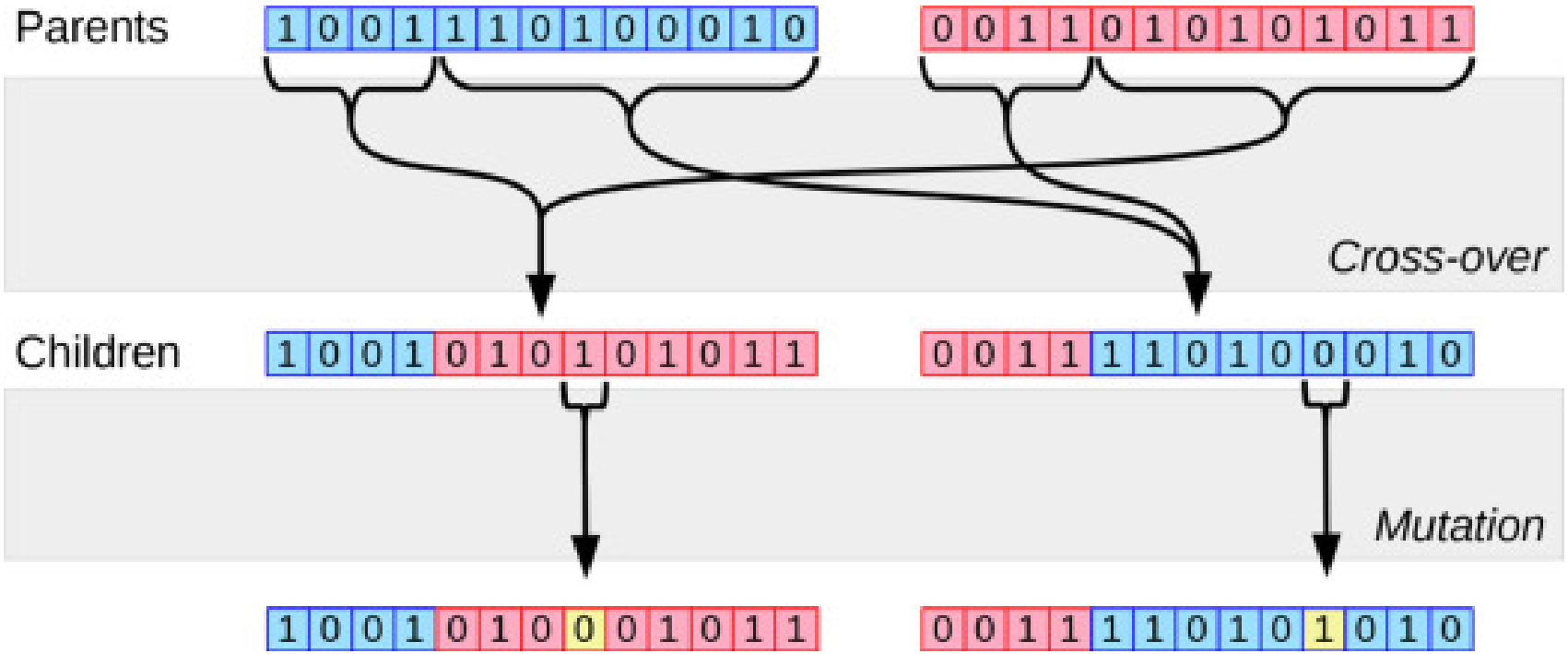


https://www.javatpoint.com/genetic-algorithm-in-machine-learning

Key Concepts of Evolutionary and Genetic Algorithms

- **Evolutionary Algorithms:** A set of computational methods inspired by the principles of biological evolution to solve optimization problems.
- **Genetic Algorithms:** A subset of evolutionary algorithms that use genetic operators (e.g., mutation, crossover) to simulate the evolution of populations of solutions.
- **Fitness Function:** A function that measures the quality of a candidate solution. The fitness function is used to select the fittest solutions for reproduction and to evaluate the overall performance of the algorithm.
- **Selection Operator:** A mechanism that chooses which individuals in a population will be selected for reproduction based on their fitness values.
- **Crossover Operator:** A genetic operator that combines the genetic material of two parent solutions to create new offspring solutions.
- **Mutation Operator:** A genetic operator that introduces small random changes into the genetic material of a single solution to create a new solution.
- **Population:** A collection of candidate solutions that evolves over time through the application of genetic operators and selection mechanisms.

Genetic Operators: crossover and mutation



https://www.sciencedirect.com/topics/medicine-and-dentistry/genetic-operator

Genetic Operators: crossover and mutation (cont.)

- **Crossover Operator:** A crossover operator is a genetic operator that combines genetic material from two parent solutions to create one or more offspring solutions. In GAs, crossover is typically performed by selecting a crossover point in the two parent strings and exchanging the genetic material to the left and right of that point. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be the two parent solutions, c be the crossover point, and $\mathbf{x}_3$ and $\mathbf{x}_4$ be the two offspring solutions, given by:

$$\mathbf{x}_3 = \mathbf{x}_1[1 : c] + \mathbf{x}_2[c + 1 : |\mathbf{x}_2|]$$

$$\mathbf{x}_4 = \mathbf{x}_2[1 : c] + \mathbf{x}_1[c + 1 : |\mathbf{x}_1|]$$

  where $|\mathbf{x}_i|$ is the length of the string representation of candidate solution $\mathbf{x}_i$.

- **Mutation Operator:** A mutation operator is a genetic operator that introduces small random changes into the genetic material of a single solution to create a new solution. In GAs, mutation is typically performed by selecting one or more positions in the solution string and replacing the symbol at that position with a randomly chosen symbol from the set of possible symbols. Let $\mathbf{x}$ be the parent solution, m be the mutation rate, and $\mathbf{x}'$ be the offspring solution, given by:

$$\mathbf{x}'_i = \begin{cases} \mathbf{x}_i & \text{with probability } 1 - m \\ \text{random symbol} & \text{with probability } m \end{cases}$$

  where $\mathbf{x}_i$ is the ith symbol in the solution string.

Algorithm Genetic operators

Crossover Operator:
for $i = 1$ *to* N *by* 2 **do**
Randomly select two parents $x_{parent1}$ and $x_{parent2}$ from P
With probability p_c, generate two offspring $x_{offspring1}$ and $x_{offspring2}$ using the following equation:

$$x_{offspring1,j} = \begin{cases} x_{parent1,j}, & \text{if } rand(0,1) < 0.5 \\ x_{parent2,j}, & \text{otherwise} \end{cases}$$

$$x_{offspring2,j} = \begin{cases} x_{parent2,j}, & \text{if } rand(0,1) < 0.5 \\ x_{parent1,j}, & \text{otherwise} \end{cases}$$

where j is the gene index
Otherwise, copy $x_{parent1}$ and $x_{parent2}$ to $x_{offspring1}$ and $x_{offspring2}$, respectively
end
Mutation Operator:
for $i = 1$ *to* N **do**
For each gene j in x_i, with probability p_m, mutate the gene as follows:

$$x_{i,j} = x_{i,j} + \delta_j$$

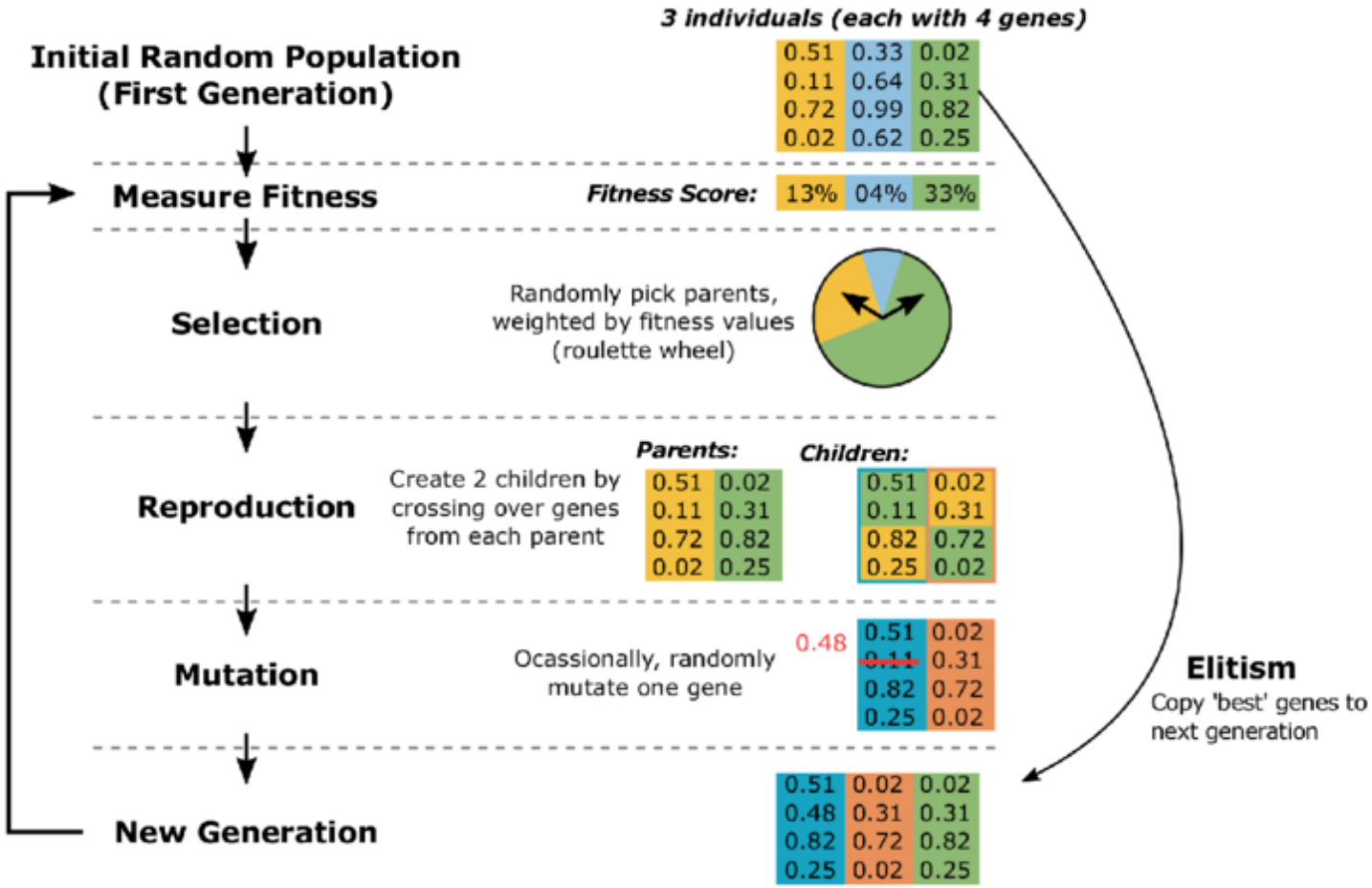
where δ_j is a random value sampled from a Gaussian distribution with mean 0 and standard deviation σ_m
end

Genetic Algorithm - Pseudocode

Algorithm Basic Genetic Algorithm

Input : population size N, fitness function f,
mutation probability p_m, crossover probability p_c,
maximum number of generations G
Output: Optimal solution x^*
Initialize population P with N random solutions;
$t \leftarrow 0$;
while $t < G$ **do**
    Evaluate the fitness of each individual in P;
    Select parents for reproduction based on fitness;
    foreach *pair of parents* **do**
        if $rand() < p_c$ **then**
            Apply crossover to generate offspring
        end
        if $rand() < p_m$ **then**
            Apply mutation to the offspring
        end
    end
    Replace the least fit individuals in P with the new offspring
    $t \leftarrow t + 1$
end
$x^* \leftarrow$ the solution in P with the highest fitness;

Genetic Operators Illustrated



hhttps://www.researchgate.net/publication/305440735

Example: Genetic Algorithm version of TSP

Algorithm Genetic Algorithm version of TSP

Input : Population P of candidate solutions, number of cities n, distance matrix D
Output: New population P' of candidate solutions
$P' \leftarrow \emptyset$;
  Sbest $\leftarrow$ the solution in P with the best fitness;
  $P' \leftarrow$ Elitism(P, Sbest);

while $|P'| < |P|$ **do**
    S $\leftarrow$ a solution randomly selected from P
    Cx $\leftarrow$ a random subset of cities from S
    Offspring $\leftarrow$ Pmx(S, Cx, Sbest)
    Offspring $\leftarrow$ Swap(Offspring, p_m)
    $P' \leftarrow P' \cup \{$Offspring$\}$

return P';

Example: Genetic Algorithm version of TSP (cont.)

Suppose we have the following distance matrix for the 5 cities:

$$\begin{pmatrix} 0 & 5 & 3 & 8 & 4 \\ 5 & 0 & 6 & 7 & 3 \\ 3 & 6 & 0 & 4 & 6 \\ 8 & 7 & 4 & 0 & 3 \\ 4 & 3 & 6 & 3 & 0 \end{pmatrix}$$

We start with an initial population of candidate solutions, each representing a possible tour of the cities. Let's say our initial population is:

$$P = \{\langle 1, 2, 3, 4, 5\rangle, \langle 1, 3, 5, 4, 2\rangle, \langle 2, 4, 1, 5, 3\rangle, \langle 3, 2, 5, 1, 4\rangle, \langle 4, 5, 3, 1, 2\rangle\}$$

We apply the genetic algorithm coding phase to generate a new population of candidate solutions. Let's assume we use the following parameter values:

- Population size $|P| = 5$
- Crossover probability $p_c = 0.9$
- Mutation probability $p_m = 0.1$

We randomly select one solution S from P, and select a random subset C_x of cities from S to be used in the crossover operation. Let's say we select $S = \langle 3, 2, 5, 1, 4\rangle$ and $C_x = \{2, 5\}$.
We also find the best solution S_{best} in P, which in this case is $S_{best} = \langle 1, 2, 3, 4, 5\rangle$.
We then apply the Partially Mapped Crossover (Pmx) operator with C_x and S_{best} to create a new offspring solution *Offspring*:

$$\mathit{Offspring} = \text{Pmx}(S, C_x, S_{best}) = \langle 3, 2, 1, 4, 5\rangle$$

We then apply the swap mutation operator with probability p_m to *Offspring*, resulting in:

$$\mathit{Offspring} = \text{Swap}(\mathit{Offspring}, p_m) = \langle 3, 5, 1, 4, 2\rangle$$

We add *Offspring* to our new population P'. We repeat this process until we have generated a new population of size $|P'| = 5$.
Note that the ELITISM function preserves the best solution S_{best} in P without modification, so it is automatically included in the new population P'.
We then repeat this process for multiple generations until convergence.

Fitness functions and selection methods

In evolutionary algorithms, fitness functions and selection methods play crucial roles.

- Fitness functions evaluate the quality of each individual in the population
- Selection methods determine which individuals should be used for the next generation

Some common fitness functions are:

- Sum of Squares
- Sphere Function
- Rastrigin Function
- Rosenbrock Function
- Ackley Function

Selection methods include:

- Roulette Wheel Selection
- Tournament Selection
- Rank-Based Selection

Choosing an appropriate fitness function and selection method can have a significant impact on an evolutionary algorithm's performance, affecting both its convergence and diversity.

Fitness functions used in optimisation - definitions

- **Sum of Squares:** a simple optimization problem that involves finding the minimum value of a sum of squares function. :

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$$

where $\mathbf{x}$ is a vector of n real-valued inputs.

- **Sphere Function:** a simple optimization problem that involves finding the minimum value of a sum of squares function, but with a specific structure.

$$f(\mathbf{x}) = \sum_{i=1}^{n} (x_i - c)^2$$

where $\mathbf{x}$ is a vector of n real-valued inputs and c is a constant.

- **Rastrigin Function:** a multi-modal optimization problem that involves finding the minimum value of a specific function.

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} (x_i^2 - A\cos(2\pi x_i))$$

where $\mathbf{x}$ is a vector of n real-valued inputs, A is a constant, and n is the dimensionality of the problem.

- **Rosenbrock Function:** a non-convex optimization problem that involves finding the minimum value of a specific function.

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

where $\mathbf{x}$ is a vector of n real-valued inputs.

- **Ackley Function:** a non-convex optimization problem that involves finding the minimum value of a specific function.

$$f(\mathbf{x}) = -a\exp\left(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + e$$

where $\mathbf{x}$ is a vector of n real-valued inputs, and a, b, c, and e are constants.

Selection methods

- **Roulette Wheel Selection:** Each individual in the population is assigned a fitness-proportional probability of selection, such that individuals with higher fitness have a higher probability of being selected. The selection process involves spinning a roulette wheel, where the size of each slice is proportional to the fitness of the corresponding individual.
- **Tournament Selection:** A small subset of individuals is selected randomly from the population, and the individual with the highest fitness in that subset is chosen as a parent. This process is repeated multiple times to select the desired number of parents for reproduction.
- **Rank-Based Selection:** Individuals are ranked according to their fitness, with the fittest individual having rank 1 and the least fit individual having rank n. Each individual is then assigned a selection probability based on their rank, such that individuals with higher ranks have a higher probability of being selected. The selection process involves choosing individuals at random with probabilities proportional to their ranks.

Selection methods (cont.)

- **Fitness Proportional Selection (Roulette Wheel Selection):** Probability of selection of an individual is proportional to its fitness value.
  Probability of selection of an individual i with fitness f_i is given by:
  $P(i) = \frac{f_i}{\sum_{j=1}^{N} f_j}$, where N is the population size.
- **Rank Selection:** Probability of selection of an individual is proportional to its rank in the population.
  Probability of selection of an individual i with rank r_i is given by:
  $P(i) = \frac{1}{\sum_{j=1}^{N} r_j}$, where N is the population size.
- **Tournament Selection:** Selects k individuals randomly from the population and chooses the best individual among them as the winner.
- **Truncation Selection:** Selects the top k individuals in the population based on their fitness values and allows them to mate.
- **Boltzmann Selection:** Probability of selection of an individual is based on its fitness relative to the fitness of the entire population and a temperature parameter T.
  Probability of selection of an individual i with fitness f_i is given by:
  $P(i) = \frac{e^{\frac{f_i}{T}}}{\sum_{j=1}^{N} e^{\frac{f_j}{T}}}$, where N is the population size and T controls the degree of selection pressure.
- **Stochastic Universal Sampling:** A variation of fitness proportional selection that selects k individuals by dividing the fitness range into k equal parts and choosing one individual from each part at random.
- **Softmax Selection:** Probability of selection of an individual is based on its fitness relative to the fitness of the entire population and a temperature parameter T.
  Probability of selection of an individual i with fitness f_i is given by:
  $P(i) = \frac{e^{f_i/T}}{\sum_{j=1}^{N} e^{f_j/T}}$, where N is the population size and T controls the degree of randomness in the selection process.

Algorithm Generational Evolutionary Algorithm

Input : Population size N, maximum number of generations G, fitness function f, selection method S, crossover operator C, mutation operator M with strength s_m and probability p_m, termination criteria T
Output: Best individual found in the final generation

Initialization: Initialize population P_0 of size N randomly
Evaluate fitness of each individual in P_0 using f
Set current generation counter g to 0
Set best fitness found so far f_{best} to $-\infty$
Set best individual found so far x_{best} to null

while $g < G$ **and** *termination criteria* T *not satisfied* **do**
  $g \leftarrow g + 1$
  Selection: Select parents from P_{g-1} using selection method S
  Recombination: Generate offspring by applying crossover operator C to the selected parents
  Mutation: Apply mutation operator M to the offspring with strength s_m and probability p_m
  Evaluation: Evaluate fitness of each offspring using f
  Survival: Form the next generation by replacing the N worst individuals in P_{g-1} with the offspring
  Update best individual: Find the best individual $x_{best,g}$ and its fitness $f_{best,g}$ in P_g
  if $f_{best,g} > f_{best}$ **then**
    $f_{best} \leftarrow f_{best,g}$
    $x_{best} \leftarrow x_{best,g}$
  end
end
return x_{best}

Exploration vs Exploitation in Genetic Algorithms

- In Genetic Algorithms, the balance between exploration and exploitation is crucial for achieving expected good performance.
- **Exploration** involves searching the solution space broadly, in order to discover new areas that may contain better solutions.
- **Exploitation** involves focusing on the most promising areas of the solution space, in order to refine and improve the current best solution.
- The choice of exploration vs exploitation strategy depends on the stage of the search and the properties of the optimization problem.
- In the early stages of the search, more emphasis is typically placed on exploration, in order to quickly discover diverse solutions and avoid getting stuck in local optima.
- As the search progresses, more emphasis is typically placed on exploitation, in order to refine the best solution and improve its quality.
- Various techniques can be used to balance exploration and exploitation in GA, including population diversity measures, adaptive mutation rates, and multi-objective optimization.

Evolutionary Strategies in Genetic Algorithms

- Evolutionary strategies (ES) are a type of GA that focus on continuous optimization problems.
- They differ from traditional GAs in that they use self-adaptive mutation operators that adjust their parameters during the optimization process.
- ES algorithms typically use a Gaussian mutation operator, which adds a random value sampled from a Gaussian distribution to each gene in the individual.
- ES algorithms also use a "plus" selection mechanism, in which the parent and offspring populations are combined and the top individuals are selected for the next generation.
- ES algorithms can be classified as $(\mu/\mu, \lambda)$, (μ/λ), or $(\mu + \lambda)$ depending on how many parents and offspring are generated per generation.
- One of the most popular ES algorithms is the $(1 + 1)$ ES, which generates only one offspring per generation and adapts the mutation step size.
- ES algorithms have been successfully applied in various fields, such as engineering, finance, and machine learning.

Generational Evolutionary Algorithm with ES(1+1)

Generational Evolutionary Algorithm with ES(1+1) is a type of evolutionary algorithm that uses a population size of one and an elitist selection strategy. The algorithm works as follows:

- Step 1: Initialization - Generate an initial solution randomly from the search space.
- Step 2: Mutation - Apply a mutation operator to the current solution to generate a new candidate solution.
- Step 3: Selection - Select the best solution between the current solution and the candidate solution based on their fitness values.
- Step 4: Termination - Stop the algorithm when a stopping criterion is met, such as a maximum number of iterations or a desired fitness value.

The ES(1+1) variant of the algorithm uses a Gaussian mutation operator with a fixed step size and a 1/5th rule for adjusting the step size based on the success rate of the algorithm. This variant has been shown to be effective for optimizing continuous functions with noise and has been used in a wide range of applications.

Gaussian Mutation Operator

The Gaussian mutation operator is a widely used mutation operator in evolutionary algorithms that perturbs a current solution $\boldsymbol{x}$ by adding a random value $\boldsymbol{\Delta}$ sampled from a Gaussian distribution:

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{\Delta},$$

where $\boldsymbol{y}$ is the new candidate solution and $\boldsymbol{\Delta}$ is a vector of random values sampled from a Gaussian distribution with mean $\mathbf{0}$ and standard deviation σ, i.e., $\boldsymbol{\Delta} \sim N(\mathbf{0}, \sigma^2 \boldsymbol{I})$.
The magnitude of the perturbation is controlled by the standard deviation σ of the Gaussian distribution, which determines the spread of the generated values. Larger values of σ result in larger perturbations and vice versa.

Algorithm Generational Evolutionary Algorithm with ES(1+1)

Input : Population size N, maximum number of generations G, fitness function f, selection method S, crossover operator C, mutation operator M with strength s_m and probability p_m, termination criteria T

Output: Best individual found in the final generation

Initialization: Initialize population P_0 of size N randomly, Evaluate fitness of each individual in P_0 using f, Set current generation counter g to 0, Set best fitness found so far f_{best} to $-\infty$, Set best individual found so far x_{best} to null

while $g < G$ **and** *termination criteria* T *not satisfied* **do**

- $g \leftarrow g + 1$
- **Selection:** Select parents from P_{g-1} using selection method S
- **Recombination:** Generate offspring by applying crossover operator C to the selected parents
- **Mutation:**
- **if** *ES(1+1) strategy* **then**
  - $x_{new} \leftarrow x_{old} + \mathcal{N}(0, s_m)$
  - **if** $f(x_{new}) > f(x_{old})$ **then**
    - $x_{old} \leftarrow x_{new}$, $s_m \leftarrow s_m \cdot e^{1/\sqrt{2d}}$
  - **else**
    - $s_m \leftarrow s_m \cdot e^{-1/\sqrt{2d}}$
- **else**
  - Apply mutation operator M to the offspring with strength s_m and probability p_m
- **Evaluation:** Evaluate fitness of each offspring using f
- **Survival:** Form the next generation by replacing the N worst individuals in P_{g-1} with the offspring
- **Update best individual:** Find the best individual $x_{best,g}$ and its fitness $f_{best,g}$ in P_g
- **if** $f_{best,g} > f_{best}$ **then**
  - $f_{best} \leftarrow f_{best,g}$, $x_{best} \leftarrow x_{best,g}$

return x_{best}

Next Four Types of Genetic Algorithms

- Generational GA (GGA):
  - All parents are replaced by their offspring in the next generation.
  - New population is generated using selection, crossover, and mutation operators.
- Steady-State $(\mu + 1)$-GA (SSGA):
  - Only one offspring is generated at a time and replaces one parent in the population.
  - Selection, crossover, and mutation operators are applied to select the parent and generate the offspring.
- Steady-Generational (μ, μ)-GA (SGGA):
  - Only μ parents are selected to generate μ offspring.
  - The best μ individuals are selected from the combined parent and offspring population to form the next generation.
- $(\mu + \mu)$-GA:
  - Two populations of μ individuals each are evolved independently.
  - The best individuals from each population are periodically swapped.

Other Variants of Genetic Algorithms

- $(\mu + \lambda)$-GA:
  - Parent population size: μ
  - Offspring population size: λ
  - Generate λ offspring solutions by applying selection, crossover, and mutation to the parent population.
  - Combine the μ parent solutions with the λ offspring solutions to form a combined population of $\mu + \lambda$ individuals.
  - Select the best μ individuals from the combined population as the parent population for the next generation.
- (μ, λ)-GA:
  - Parent population size: μ
  - Offspring population size: λ
  - Generate λ offspring solutions by applying selection, crossover, and mutation to the μ parent population.
  - Select the best μ individuals from the λ combined population as the parent population for the next generation.

Other Variants of Genetic Algorithms

- Elitist GA:
  - Preserves the best individuals in the population across generations.
  - Ensures that the fittest solutions are not lost.
- Niching GA:
  - Seeks multiple solutions to the problem.
  - Structures the population to maintain diversity among the solutions.
- Island GA:
  - Divides the population into multiple subpopulations.
  - Each subpopulation evolves independently, with individuals periodically migrating between subpopulations to maintain diversity.
- Multi-objective GA:
  - Optimizes multiple objectives simultaneously.
  - Produces a set of solutions that represent trade-offs between the objectives.
- Adaptive GA:
  - Modifies its parameters, such as crossover and mutation rates, over time to improve performance.
- Hybrid GA:
  - Combines Genetic Algorithms with other optimization techniques, such as simulated annealing, to leverage the strengths of each method.

Replacement Approaches in GA

- In GA, the population size is typically fixed, so new offspring must replace some of the existing individuals in the population.
- The choice of which individuals to replace can have a significant impact on the performance of the algorithm.
- Common replacement strategies include:
  - **FIFO (First In, First Out)**: The oldest individuals in the population are replaced by the new offspring.
  - **Elitist**: The best individuals from the current population are preserved, and the worst individuals are replaced by the new offspring.
  - **Steady-State**: A small number of individuals are replaced at each iteration, typically through a combination of selection and mutation.
- FIFO can be useful for maintaining diversity in the population, as it ensures that the population does not become dominated by a single solution.
- Elitist replacement can be useful for preserving the best solutions found so far and accelerating convergence towards a high-quality solution.
- Steady-state replacement can be useful for exploring the search space more thoroughly and can be particularly effective for problems with rugged fitness landscapes.
- Other replacement approaches include tournament selection, age-based replacement, and fitness-based replacement.
- The choice of replacement approach depends on the specific problem being solved and the goals of the optimization.

Evaluation of Evolutionary Algorithms

- Evaluating the performance of evolutionary algorithms (EAs) is a critical task in order to determine their effectiveness in solving optimization problems.
- One commonly used approach is to test EAs on a set of benchmark functions, which are well-known functions with known optima and known levels of difficulty.
- The results of benchmark tests are typically reported using measures such as mean value, best value, standard deviation, and success rate.
- Another approach is to compare the performance of different EAs on the same set of problems, using statistical tests such as the Wilcoxon rank-sum test or the Friedman test.
- In addition to benchmark tests, it is important to evaluate EAs on real-world problems to assess their practical usefulness.
- Other factors to consider when evaluating EAs include computational efficiency, robustness to noise and uncertainty, and scalability to high-dimensional problems.

Example: CEC2005 Benchmark and GA Comparisons

- The CEC2005 benchmark was developed for the Congress of Evolutionary Computation to evaluate the performance of optimization algorithms on a set of 25 real-world optimization problems.
- The benchmark includes a mix of unimodal, multimodal, separable, and non-separable problems with varying levels of difficulty.
- The CEC2005 benchmark has become a widely used benchmark for evaluating the performance of evolutionary algorithms, including genetic algorithms (GAs).
- Comparing the performance of different GAs on the CEC2005 benchmark can provide insights into the strengths and weaknesses of different GA variants.
- However, it is important to note that the CEC2005 benchmark is just one of many possible benchmark sets, and the results may not generalize to other problem domains.
- Furthermore, the CEC2005 benchmark does not include constraints or other complexities that are present in many real-world problems, so it is important to evaluate GAs on such problems as well.

Other Evolutionary Algorithms Beyond Genetic Algorithms

- **Differential Evolution (DE):** A population-based optimization method that relies on differential mutations as a mechanism for generating new candidate solutions. DE is particularly effective for multi-modal optimization problems.
- **Particle Swarm Optimization (PSO):** Inspired by social behavior patterns of organisms like fish schooling or bird flocking. PSO optimizes a problem by iteratively improving a candidate solution with regard to a given measure of quality. It adjusts the trajectories of individual particles through the solution space by following the current optimum particles.
- **Ant Colony Optimization (ACO):** Mimics the behavior of ants searching for food. ACO is used to find optimal paths through graphs and is effective in solving combinatorial optimization problems, such as traveling salesman and vehicle routing problems.

Differential Evolution (DE) Explained

- **Mutation:** DE's mutation strategy is distinctive, utilizing the weighted difference between population vectors. This approach allows for the exploration of the solution space by generating diverse trial vectors, driving the population towards optimal solutions.
- **Crossover:** The crossover mechanism in DE integrates the mutated vector with the current generation's vector to produce trial vectors. This step enhances the algorithm's ability to explore and exploit the solution space, contributing to the diversity and quality of solutions.
- **Selection:** DE employs a straightforward selection process, where the trial vector competes with its corresponding target vector. The one with better fitness survives to the next generation, ensuring a gradual improvement in the population's quality.
- **Optimizing Complex Continuous Functions:** The combination of DE's unique mutation strategy, effective crossover, and selective pressure makes it particularly powerful for solving optimization problems in continuous domains. Its ability to balance exploration and exploitation efficiently helps in finding global optima in complex landscapes.

Differential Evolution - Pseudocode

Algorithm Basic Differential Evolution Algorithm

Input : population size N, fitness function f,
mutation factor F, crossover rate CR,
maximum number of generations G
Output: Optimal solution x^*
Initialize population P with N random individuals;
Evaluate the fitness of each individual in P;
$t \leftarrow 0$;
while $t < G$ **do**
  foreach *individual* x_i *in* P **do**
    Select three distinct individuals a, b, c from P excluding x_i;
    Generate a trial vector v_i by mutating a with $F(b - c)$;
    Perform crossover between x_i and v_i to form trial individual u_i based on CR;
    Evaluate the fitness of u_i;
    if *fitness of* u_i *is better than fitness of* x_i **then**
      Replace x_i in P with u_i;
    end
  end
  $t \leftarrow t + 1$;
end
$x^* \leftarrow$ the individual in P with the highest fitness;

Particle Swarm Optimization (PSO) (I)

- **Initialization:** Start with a swarm of particles, each having a randomly assigned position and velocity within the search space. Each particle represents a potential solution to the optimization problem.
- **Velocity and Position Update:** For each particle, update its velocity and position based on its own best known position and the swarm's best known position. This step incorporates both personal and social knowledge to guide the search process.
  - *Velocity Update:* Influenced by the particle's historical best position and the global best among all particles.
  - *Position Update:* Moves the particle through the solution space based on its updated velocity.
- **Iteration:** Repeat the process of updating velocity and position for each particle in the swarm across a set number of iterations, or until a convergence criterion is met. This iterative process allows the swarm to explore and exploit the search space efficiently.
- **Optimization and Solution Identification:** Through this iterative process, the swarm converges towards the optimal solution, with the global best position being continuously updated as particles find better positions.

Particle Swarm Optimization (PSO) (II)

Key Concepts:

- **Particle**: An individual solution in the search space.
- **Position** (x_i): Current solution state of a particle.
- **Velocity** (v_i): Determines the movement of a particle.
- **Personal Best** ($pbest_i$): Best position achieved by the particle.
- **Global Best** ($gbest$): Best position found by the swarm.

Update Formulas:

$$\text{Velocity Update: } v_i^{(t+1)} = w \cdot v_i^{(t)} + c_1 \cdot r_1 \cdot (pbest_i - x_i^{(t)}) + c_2 \cdot r_2 \cdot (gbest - x_i^{(t)})$$

$$\text{Position Update: } x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}$$

Where:

- $v_i^{(t)}$ and $x_i^{(t)}$ are the velocity and position of particle i at time t.
- w is the inertia weight, c_1 and c_2 are cognitive and social coefficients, respectively.
- r_1 and r_2 are random numbers between 0 and 1.
- $pbest_i$ and $gbest$ represent the best positions found by particle i and any particle in the swarm, respectively.

Particle Swarm Optimization - Pseudocode

Algorithm Basic Particle Swarm Optimization Algorithm

input : number of particles N, objective function $f(\cdot)$,
maximum number of iterations T,
inertia weight ω, cognitive coefficient c_1,
social coefficient c_2
output: Best found solution g^*
Initialize swarm of N particles with random positions and velocities;
Evaluate the fitness of each particle;
$g^* \leftarrow$ position of the best performing particle;
$t \leftarrow 0$;
while $t < T$ **do**
  foreach *particle* i **do**
    Update particle i's velocity using ω, c_1, c_2, g^*, and i's best known position;
    Update particle i's position based on its velocity;
    Evaluate the new fitness of particle i;
    if *fitness of particle* i *is better than* i*'s best known position* **then**
      Update i's best known position to its current position;
    end
    if *fitness of particle* i *is better than* g^* **then**
      Update g^* to i's current position;
    end
  end
  $t \leftarrow t + 1$;
end

Ant Colony Optimization (ACO)

- **Pheromone Initialization:** The algorithm starts with the initialization of pheromone levels on all paths. This simulates the natural behavior of ants leaving pheromones on their trails to communicate with other ants.
- **Solution Construction:** Ants construct solutions iteratively based on pheromone and heuristic information. The balance between these two factors (α for pheromone influence and β for heuristic influence) guides the ants in exploring the solution space.
- **Solution Evaluation:** Each solution constructed by an ant is evaluated using a predefined objective function. This step determines the quality of the solutions found by the ants.
- **Best Solution Update:** The algorithm keeps track of the best solution found across all iterations. If an ant finds a better solution than the current best, the best solution is updated.
- **Pheromone Update:** After all ants have constructed their solutions, the pheromone levels are updated. This includes evaporating pheromones on all paths to simulate the natural decay of pheromone trails, and ants depositing new pheromones on the paths they used, proportional to the quality of their solutions.

The Traveling Salesman Problem (TSP) and ACO

In the context of the TSP:

- Cities are represented by graph nodes.
- Distances between cities are represented by the weights of the edges connecting the nodes.
- Pheromone levels (τ_{ij}) on edges symbolize the accumulated experience of previous solutions, guiding ants towards more promising paths.
- Visibility (η_{ij}), often the inverse of the distance between two cities, represents the desirability of moving from one city to another, encouraging shorter paths.

How ACO Tackles the TSP:

- **Probabilistic Path Construction:** Ants probabilistically choose their next city based on the pheromone level and visibility, favoring paths that are both popular (high pheromone) and short (high visibility).
- **Pheromone Update:** After all ants complete their tours, pheromones on the paths are updated to reinforce the trails of shorter tours and let the pheromone on longer tours evaporate, gradually leading the colony to converge on the best solution found.
- **Solution Improvement:** Through iterations, the process of exploration (via stochastic decisions) and exploitation (via pheromone reinforcement) allows the algorithm to explore the solution space effectively and hone in on the optimal or near-optimal solutions to the TSP.

Mathematical Formulas in Ant Colony Optimization

The probability of an ant k choosing to move from city i to city j is given by:

$$P_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta} & \text{if } j \in N_i^k \\ 0 & \text{otherwise} \end{cases}$$

Where:

- $\tau_{ij}(t)$ is the amount of pheromone on edge (i, j) at time t.
- η_{ij} is the visibility or desirability of edge (i, j), typically the inverse of the distance between cities i and j.
- N_i^k is the set of cities not yet visited by ant k located in city i.
- α and β are parameters that control the influence of $\tau_{ij}(t)$ and η_{ij}, respectively.
- $P_{ij}^k(t)$ is the probability that ant k will move from city i to city j at time t.

Pheromone update rule:

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^k$$

Where:

- ρ is the pheromone evaporation rate.
- m is the number of ants.
- $\Delta\tau_{ij}^k$ is the amount of pheromone deposited on edge (i, j) by ant k, typically related to the quality of the solution found by ant k.

Ant Colony Optimization - Pseudocode

Algorithm Basic Ant Colony Optimization Algorithm

input : number of ants M, number of iterations T,
evaporation rate ρ, pheromone influence α,
heuristic influence β, objective function $f(\cdot)$
output: Best found solution s^*
Initialize pheromones on all paths;
$t \leftarrow 0$;
while $t < T$ **do**
  foreach *ant* k **do**
    Construct a solution s_k using α and β to balance between pheromone strength and heuristic information; Evaluate the quality of s_k using $f(\cdot)$;
  end
  Update s^* if any ant has found a better solution than the current best solution s^*; **foreach** *path* **do**
    Evaporate pheromones using rate ρ; **foreach** *ant* k **do**
      Deposit pheromones on the path used by k proportional to the quality of s_k;
    end
  end
  $t \leftarrow t + 1$;
end

Summary

- Introduction to evolutionary and genetic algorithms
- Key concepts in evolutionary algorithms
- Basic genetic algorithm operations
- Fitness functions and selection methods
- Examples of applications of evolutionary and genetic algorithms

References

1. S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach", Financial Times Prentice Hall, 2019.
2. M. Flasiński, Introduction to Artificial Intelligence", Springer Verlang, 2016
3. M. Muraszkiewicz, R. Nowak (ed.), Sztuczna Inteligencja dla inżynierów", Oficyna Wydawnicza PW, 2022
4. J. Prateek , Artificial Intelligence with Python", Packt 2017