# Chapter 9: Advance Optimization Algorithms

## 9.1. Genetic algorithms

Genetic algorithms (GAs) are optimization algorithms inspired by the process of natural selection and genetics. They are commonly used to solve optimization and search problems by mimicking the process of natural selection to evolve solutions.

Here's a basic explanation of how genetic algorithms work:

1. **Initialization**: A population of individuals, often represented as strings of binary digits or other data structures, is randomly generated. Each individual represents a potential solution to the problem.
2. **Evaluation**: Each individual in the population is evaluated according to a predefined fitness function, which quantifies how well the individual solves the problem. The fitness function typically measures how close the solution is to the desired outcome.
3. **Selection**: Individuals are selected from the current population to serve as parents for the next generation. Selection is usually based on the individuals' fitness scores, with fitter individuals being more likely to be selected. Common selection methods include roulette wheel selection, tournament selection, and rank-based selection.
4. **Crossover**: Selected individuals undergo crossover (also known as recombination or mating), where parts of their genetic information are exchanged to create offspring. This process is analogous to genetic recombination in biological reproduction. Crossover helps combine the characteristics of different individuals and potentially produce better solutions.
5. **Mutation**: After crossover, some individuals in the population undergo mutation, where random changes are introduced to their genetic information. Mutation helps introduce diversity into the population, preventing premature convergence to suboptimal solutions.
6. **Replacement**: The new offspring, along with some individuals from the previous generation, form the next generation population. The individuals from the previous generation may undergo some form of elitism, where the best individuals are directly transferred to the next generation without undergoing crossover or mutation.
7. **Termination**: The algorithm iterates through generations until a termination condition is met, such as reaching a maximum number of generations, finding a satisfactory solution, or reaching a predefined computational limit.

By iteratively applying selection, crossover, and mutation operators, genetic algorithms explore the search space and converge towards solutions that are increasingly fit for the problem at hand. They are particularly useful for problems where the search space is large, complex, and poorly understood, as they can efficiently explore and exploit the solution space to find near-optimal solutions.

**Different crossover Techniques:**

Crossover, also known as recombination or mating, is a key operator in genetic algorithms (GAs) used to generate new offspring by combining genetic information from parent individuals. Crossover helps maintain diversity in the population and allows for the exchange of useful genetic material between individuals. There are several crossover techniques used in GAs, each with its own characteristics and advantages. Here are some common crossover techniques:

1. **Single-Point Crossover**:
   * In single-point crossover, a single crossover point is randomly selected along the length of the parent chromosomes.
   * Offspring are created by exchanging the genetic material between the parents' chromosomes at the chosen crossover point.
   * This technique is straightforward and easy to implement but may not effectively explore the solution space when the optimal solution requires more complex combinations.
2. **Two-Point Crossover**:
   * Two-point crossover is similar to single-point crossover, but instead of a single crossover point, two crossover points are randomly selected along the length of the parent chromosomes.
   * Genetic material between the two crossover points is exchanged between the parents' chromosomes to create offspring.
   * Two-point crossover can provide additional diversity compared to single-point crossover and may be more effective for certain types of problems.

Similarly Multi-Point crossover is defined:

**genetic algorithms to solve 0/1 knapsack**

Genetic algorithms (GAs) can be applied to solve the 0/1 knapsack problem, a classic optimization problem in which items of different weights and values must be selected to maximize the value while keeping the total weight within a certain limit (the knapsack capacity).

Let's consider the following scenario:

* We have a knapsack with a maximum capacity of 15 units.
* There are 6 items with their respective values and weights:
  1. Item 1: Value = 10, Weight = 5
  2. Item 2: Value = 8, Weight = 7
  3. Item 3: Value = 15, Weight = 8
  4. Item 4: Value = 7, Weight = 4
  5. Item 5: Value = 6, Weight = 3
  6. Item 6: Value = 18, Weight = 9

We want to find the combination of items that maximizes the total value while keeping the total weight within the capacity of the knapsack.

Here's how the genetic algorithm can be applied step by step:

1. **Initialization**: Generate an initial population of individuals, each representing a possible combination of items. For example, a binary string can represent whether each item is included (1) or excluded (0) from the knapsack. For instance, an initial population might include individuals like [1, 0, 1, 0, 1, 0], [0, 1, 0, 1, 0, 1], etc.
2. **Fitness Evaluation**: Evaluate the fitness of each individual by calculating the total value of the items included in the knapsack and penalizing if the total weight exceeds the knapsack's capacity.
3. **Selection**: Use a selection method (e.g., tournament selection) to choose individuals from the current population to serve as parents for the next generation. Fitter individuals (i.e., those with higher total value and within the weight constraint) are more likely to be selected.
4. **Crossover**: Apply crossover (e.g., one-point crossover) to pairs of selected parents to create offspring. For example, if we select two parents [1, 0, 1, 0, 1, 0] and [0, 1, 0, 1, 0, 1], a crossover might produce offspring like [1, 0, 0, 1, 0, 1] and [0, 1, 1, 0, 1, 0].
5. **Mutation**: Introduce random changes (mutations) to the genetic information of some individuals. For example, randomly flip some bits (change inclusion/exclusion status) in the binary strings representing individuals.
6. **Replacement**: Form the next generation population by combining the offspring generated through crossover and mutation with some individuals from the previous generation, possibly using elitism to transfer the best individuals directly to the next generation.
7. **Termination**: Repeat the process for a predefined number of generations or until a termination condition is met (e.g., finding a satisfactory solution, reaching a computational limit).

By iteratively applying these steps, the genetic algorithm explores the solution space. It converges towards solutions that maximize the total value of items included in the knapsack while respecting its weight constraint.

Videa link: <https://youtu.be/Q530u_g2CWE?feature=shared>

## 9.2 Particle Swarm Algorithm:

Particle Swarm Optimization (PSO) is a population-based optimization algorithm inspired by the social behavior of birds flocking or fish schooling. It was originally proposed by Kennedy and Eberhart in 1995. PSO is commonly used to solve optimization problems, especially in continuous spaces, where the objective is to find the optimal solution within a search space.

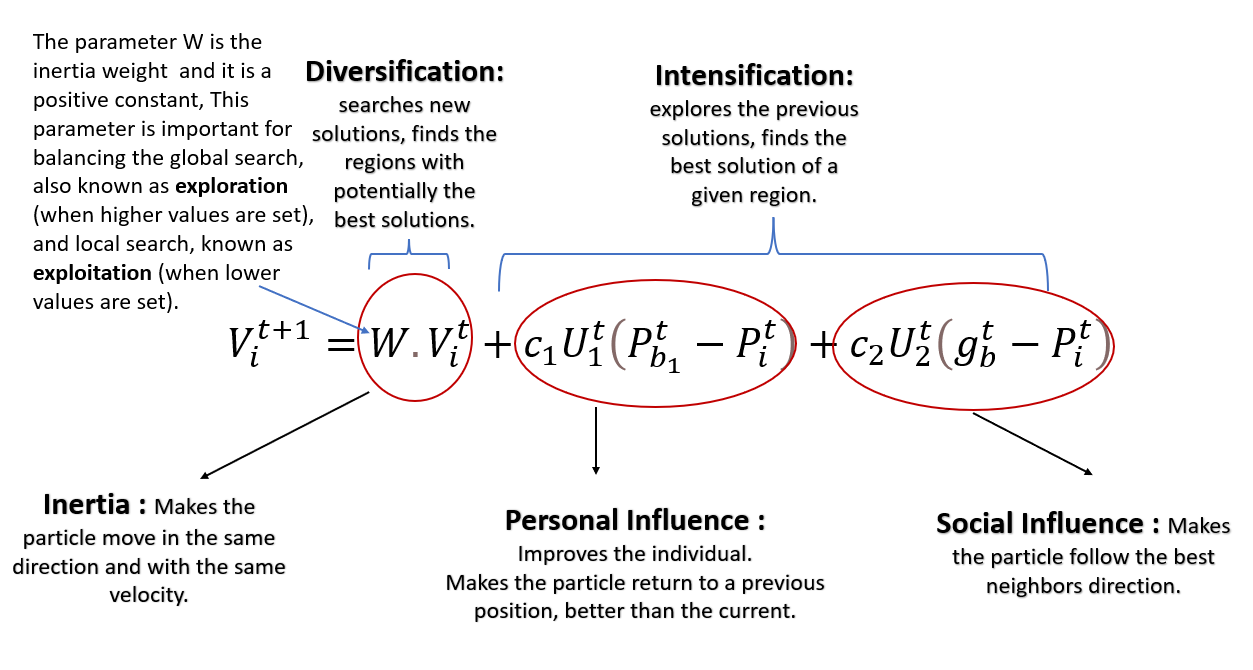
Here's an explanation of how PSO works:

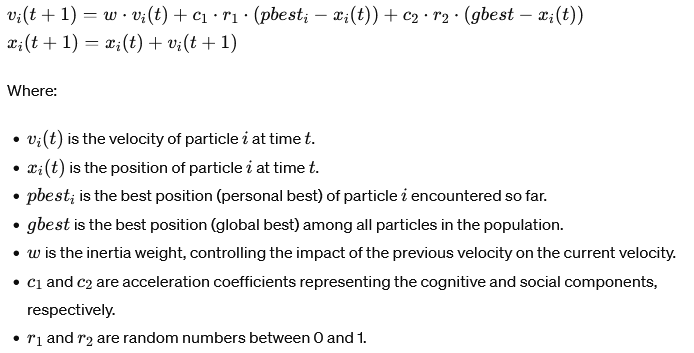
**Initialization**: PSO starts by initializing a population of particles. Each particle represents a potential solution to the optimization problem. The particles are randomly distributed across the search space.

**Velocity and Position Update**: Each particle adjusts its velocity and position iteratively based on its own experience and the experiences of its neighbors. The velocity update equation for each particle is influenced by two components:

* + **Cognitive Component**: Represents the particle's memory of its own best position (best solution it has encountered so far).
  + **Social Component**: Represents the influence of the particle's neighbors' best positions (best solutions found by particles in its neighborhood).

The velocity and position of each particle are updated according to the following equations:

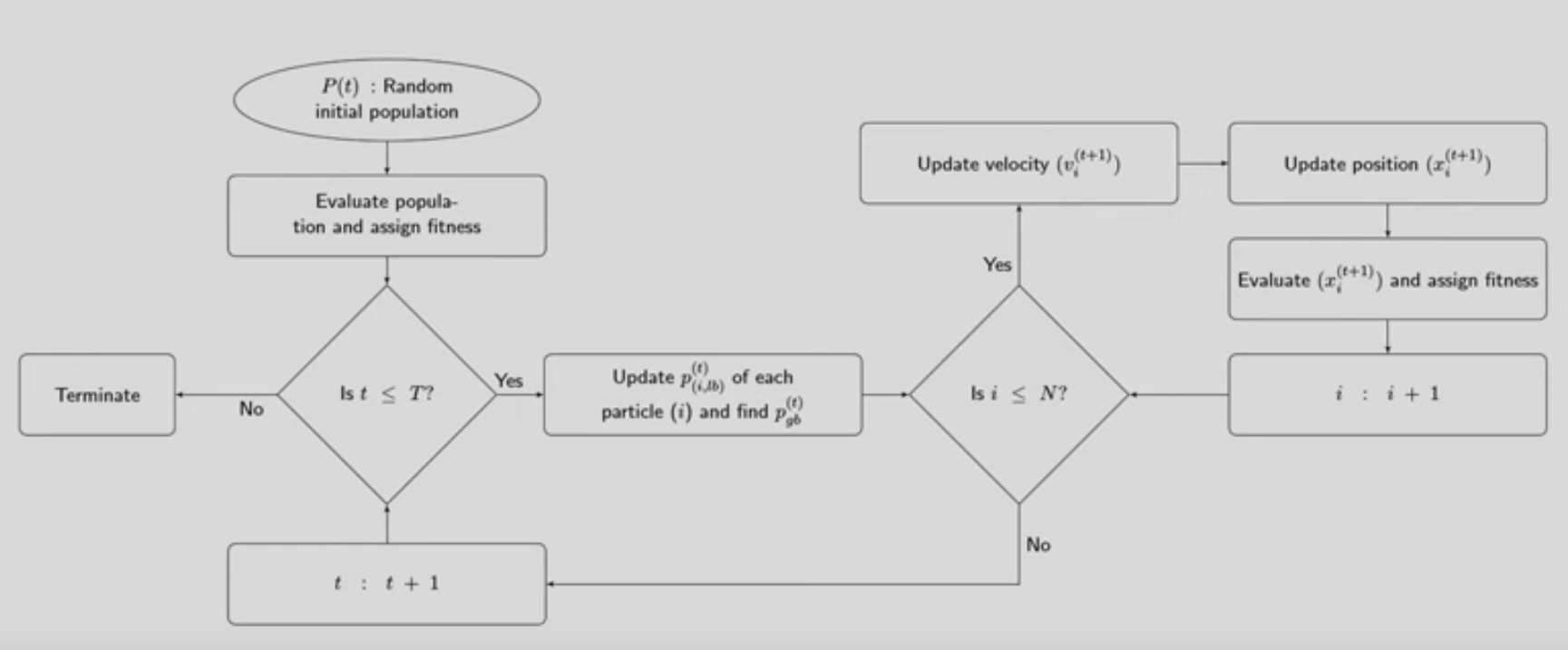
****



**Evaluation**: After updating their positions, each particle evaluates its fitness (objective function value) based on its new position in the search space.

1. **Update Personal Best and Global Best**: Each particle compares its current position's fitness value with its personal best fitness value. If the current position is better, the particle updates its personal best position. Additionally, the population's global best position is updated based on the best position found by any particle in the population.
2. **Termination**: PSO iterates through a predefined number of iterations or until a termination condition is met, such as finding a satisfactory solution or reaching a computational limit.

PSO relies on collaboration and information exchange among particles to guide the search toward promising regions of the search space. It effectively balances exploration (searching different areas of space) and exploitation (focusing on promising regions) to converge towards the optimal solution. PSO's simplicity and effectiveness make it popular for solving various optimization problems.



<https://youtu.be/8dpYjgRskgQ?feature=shared> upto 32minutes

## **9.3 Ant Colony Optimization (ACO)**

Ant Colony Optimization (ACO) is a metaheuristic optimization algorithm inspired by the foraging behavior of ants. It was introduced by Marco Dorigo in the early 1990s. ACO is particularly useful for solving combinatorial optimization problems, such as the Traveling Salesman Problem (TSP) and the Vehicle Routing Problem (VRP).

Here's an explanation of how ACO works:

1. **Initialization**:
   * Initialize a population of artificial ants. Each ant represents a potential solution to the optimization problem.
   * Initially, ants are placed randomly in the search space or at predefined locations, depending on the problem.
2. **Construction of Solutions**:
   * Each ant constructs a solution iteratively by probabilistically choosing the next component (e.g., city in the TSP) based on pheromone trails and heuristic information.
   * Pheromone trails represent the amount of pheromone deposited by ants on the edges (or components) of the solution.
   * Heuristic information guides ants towards promising components based on domain-specific knowledge. For example, in the TSP, the distance between cities can serve as heuristic information.
3. **Pheromone Update**:
   * After all ants construct their solutions, pheromone trails are updated based on the quality of the solutions found.
   * Ants deposit pheromone on the solution's edges (or components) based on the quality (e.g., length of tour in TSP) of the solution they constructed.
   * Pheromone evaporation is also applied to prevent pheromone buildup and encourage search space exploration.
4. **Global Update**:
   * In addition to local pheromone updates by individual ants, a global pheromone update is performed to further influence the search process.
   * This global update reinforces pheromone trails of good solutions found by ants, encouraging other ants to explore similar paths in subsequent iterations.
5. **Termination**:
   * Repeat the process for a predefined number of iterations or until a termination condition is met, such as finding a satisfactory solution or reaching a computational limit.

ACO relies on the collective intelligence of the ant colony to efficiently explore the solution space and converge towards good solutions. Pheromone trails guide ants towards promising regions of the search space, while the iterative process of construction, evaluation, and pheromone update helps refine the solutions over time. ACO is known for its ability to find high-quality solutions for combinatorial optimization problems and its flexibility in adapting to different problem domains.

**Pheromone Update:**

The pheromone update mechanism in Ant Colony Optimization (ACO) plays a crucial role in guiding the search process towards promising regions of the solution space. It involves updating the amount of pheromone deposited on edges (or components) of the solution based on the quality of solutions found by ants. The pheromone update is typically performed after all ants have completed their construction of solutions.

Here's a detailed explanation of the pheromone update process:

1. **Local Pheromone Update**:
   * After each ant constructs its solution, it deposits pheromone on the edges (or components) of the solution it traversed.
   * The amount of pheromone deposited is typically proportional to the quality of the solution. For example, in the Traveling Salesman Problem (TSP), ants deposit more pheromone on shorter edges (or tours).
   * The local pheromone update is performed to bias future ant behavior towards edges that are part of good solutions.
2. **Evaporation**:
   * To prevent pheromone buildup and encourage search space exploration, a certain amount of pheromone is evaporated from all edges (or components) between iterations.
   * Pheromone evaporation reduces the influence of old pheromone trails, allowing ants to explore new paths and preventing premature convergence to suboptimal solutions.
   * The evaporation rate determines how quickly the pheromone dissipates over time. A higher evaporation rate encourages exploration, while a lower evaporation rate promotes exploitation of known good paths.
3. **Global Pheromone Update**:
   * In addition to local pheromone updates by individual ants, a global pheromone update is performed to influence the search process further.
   * The global pheromone update reinforces pheromone trails of good solutions found by ants, making them more attractive to future ants.
   * The amount of pheromone deposited globally depends on the quality of the solutions found by the entire ant colony. Higher-quality solutions receive more pheromone reinforcement.
   * The global update helps guide ants towards promising regions of the search space discovered by other ants, facilitating cooperation and information sharing among the colony.

## **9.4 Optimizing the Fuzzy Systems:**

To optimize a fuzzy system for a bank loan example using credit score as a single parameter, we'll design a fuzzy logic system that evaluates loan eligibility based on an applicant's credit score. Fuzzy logic is well-suited for modeling complex systems with uncertain or imprecise inputs, making it applicable in various domains, including finance.

Here's how we can approach the optimization process:

1. **Fuzzy System Design**:
   * Define linguistic variables: In this case, the linguistic variable is "credit score," which can be described by fuzzy sets such as "low," "medium," and "high."
   * Define membership functions: Membership functions characterize the degree of membership of a value in each fuzzy set. Triangular or trapezoidal membership functions can be used to represent "low," "medium," and "high" credit scores.
   * Define rules: Rules map input variables (credit score) to output variables (loan eligibility). For example, "If credit score is low, then loan eligibility is low."
   * Define inference mechanism: Use fuzzy inference (e.g., Mamdani or Sugeno) to determine the degree of support for each output based on the input variables and rules.
   * Define the defuzzification method: Convert the fuzzy output into a crisp value. Common defuzzification methods include centroid, weighted average, and max membership.
2. **Optimization**:
   * Fine-tune the membership functions: Adjust the shape and parameters of the membership functions better to capture the relationship between credit score and loan eligibility. This optimization process may involve expert knowledge or data-driven approaches.
   * Optimize rule base: Evaluate and refine the fuzzy rules to improve the accuracy of loan eligibility assessment. Consider adding or modifying rules to handle different credit score ranges better.
   * Validate and test: Validate the optimized fuzzy system using historical loan data or simulated scenarios to ensure its effectiveness in predicting loan eligibility accurately.
3. **Example**:
   * Let's say we have a fuzzy system with three linguistic variables for credit score: "low," "medium," and "high." We define triangular membership functions for each linguistic variable.
   * We also define rules such as:
     + If the credit score is low, then loan eligibility is low.
     + If the credit score is medium, then loan eligibility is medium.
     + If the credit score is high, then loan eligibility is high.
   * We use Mamdani fuzzy inference and centroid defuzzification to determine loan eligibility based on the credit score input.
4. **Evaluation**:
   * Evaluate the optimized fuzzy system using real-world credit score data and loan application scenarios.
   * Measure the accuracy and performance of the fuzzy system in predicting loan eligibility compared to traditional methods or expert judgment.
   * Iterate on the optimization process if necessary to further improve the performance of the fuzzy system.

**The Centroid Method:**

By optimizing the fuzzy system for loan eligibility assessment based on credit score, banks can make more informed lending decisions, reduce risk, and improve customer satisfaction. The flexibility of fuzzy logic allows for robust modeling of complex relationships in financial decision-making processes.

The Centroid Method is a common defuzzification technique used in fuzzy logic systems to convert the fuzzy output into a crisp (non-fuzzy) value. It calculates the center of mass or centroid of the fuzzy output's membership function to determine the final output value.

Here's how the centroid method works:

1. **Fuzzy Inference**:
   * After the fuzzy inference process, each linguistic variable (output) typically has a membership function representing its fuzzy output. These membership functions may have different shapes and heights, reflecting the degree of membership of the output variable in different linguistic terms (e.g., "low," "medium," "high").
2. **Defuzzification**:
   * Defuzzification is the process of converting the fuzzy output into a single crisp value. The centroid method is one of several defuzzification techniques used for this purpose.
   * To apply the centroid method, first, determine the area under the fuzzy output's membership function. This area represents the degree of support for each linguistic term.
   * Then, calculate the centroid or center of mass of this area. The centroid is calculated as the weighted average of the values in the universe of discourse (the range of possible output values), where the weights are determined by the degree of membership at each point.
3. **Example**:
   * Suppose we have a fuzzy output variable "temperature" with linguistic terms "low," "medium," and "high," each represented by triangular membership functions.
   * After fuzzy inference, we calculate the area under each membership function to determine the degree of support for each linguistic term.
   * Using the centroid method, we find this area's center of mass, representing the crisp output temperature value.
   * If the centroid value is, for example, 75 degrees Fahrenheit, this would be the final output of the fuzzy system.

Video Link: <https://youtu.be/__0nZuG4sTw?feature=shared>

MCQ Links:   
<https://gtu-mcq.com/BE/Mechanical-Engineering/Semester-7/2171901/1114/MCQs?q=9aZHDjblmRk=#google_vignette>

<https://testbook.com/objective-questions/mcq-on-transportation-model--5eea6a0e39140f30f369e4eb>