Part1:Evolutionary Systems

Professeur Abdellatif El Afia

- All biological systems went throw an evolutionary process.

- Because of the robustness and adaptability of biological systems, the mechanisms of this systems become an inspiration for many scientists.

- Computer scientists were able to create artificial evolutionary systems

Artificial evolutionary systems were used to solve problems in different fields:

- Scheduling:
  - Scheduling Solving Production Scheduling Problem of Automotive Parts Workshop Based on Improved Genetic Algorithm (2020)
- Path planning:
  - Path planning of cooperating industrial robots using evolutionary algorithms (2020)
- ANN Design:
  - Determine an optimal neural network topology Using Genetic Algorithms to Evolve Artificial Neural Networks (2016)
- SVM classifier improvement:
  - SVGPM: evolving SVM decision function by using genetic programming to solve imbalanced classification problem(2021)

I. EVOLUTIONARY COMPUTATION

1. Introduction
2. Metaheuristics
3. Exploration / Exploitation
4. Metaheuristics types

II. SINGLE SOLUTION BASED

1. Introduction
2. Hill Climbing
3. Simulated annealing
4. Tabu search
5. Variable neighborhood search
6. Others Metaheuristics

III. POPULATION BASED

1. Introduction
2. The Biology of Evolutionary Systems
3. Overview of EA

IV. PERFORMANCE IMPROVEMENT

1. Introduction
2. Parameter control techniques
3. Parameter control using Machine learning
4. Hybridization
5. Novel genetic operators

V. MULTIOBJECTIVE EVOLUTIONARY ALGORITHM

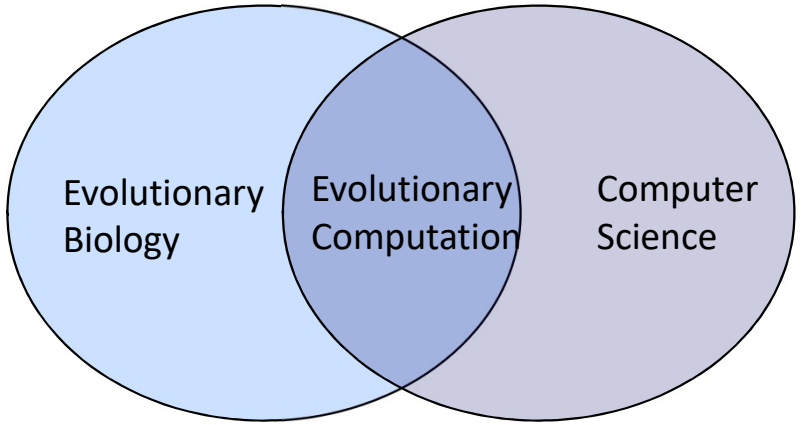
1. Introduction
2. Optimization Theory
3. Constraint handling
4. MOEA basic structure
5. Non-dominated Sorting Genetic Algorithm (NSGA-II)
6. Strength Pareto Evolutionary Algorithm 2 (SPEA2)
7. Non-dominated Sorting Genetic Algorithm (NSGA-III)
8. Multi-objective evolutionary algorithm based on decomposition (MOEA/D)

I. EVOLUTIONARY COMPUTATION

1. Introduction
2. Metaheuristics
3. Exploration / Exploitation
4. Metaheuristics types

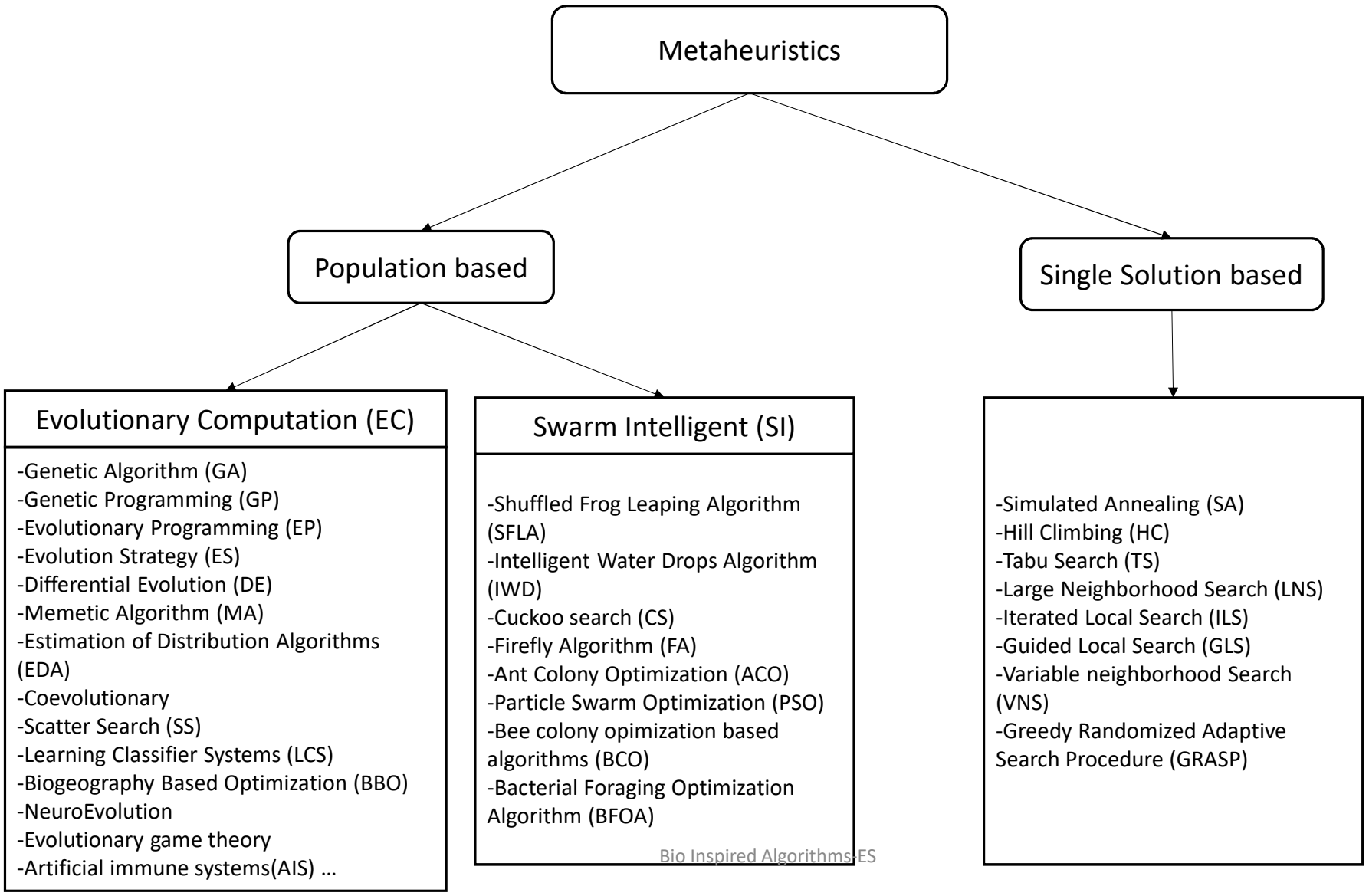
1.Introduction

- Techniques inspired by natural systems.
- Evolutionary computation is all about search and navigating the search space.
- EC is used for "optimization" ,which just means "finding the best" .
- EC are types of metaheuristics.



2. Metaheuristics

- High-level problem-solving strategies used to find approximate solutions for difficult optimization problems.
- Usually iterative and incorporate a random element to explore the search space effectively.
- Do not guarantee optimal solutions, but can be used to find near-optimal solutions in a reasonable amount of time.


Metaheuristics
Population based
Single Solution based
Evolutionary Computation (EC)
-Genetic Algorithm (GA)
-Genetic Programming (GP)
-Evolutionary Programming (EP)
-Evolution Strategy (ES)
-Differential Evolution (DE)
-Memetic Algorithm (MA)
-Estimation of Distribution Algorithms (EDA)
-Coevolutionary
-Scatter Search (SS)
-Learning Classifier Systems (LCS)
-Biogeography Based Optimization (BBO)
-NeuroEvolution
-Evolutionary game theory
-Artificial immune systems(AIS) …
Swarm Intelligent (SI)
-Shuffled Frog Leaping Algorithm (SFLA)
-Intelligent Water Drops Algorithm (IWD)
-Cuckoo search (CS)
-Firefly Algorithm (FA)
-Ant Colony Optimization (ACO)
-Particle Swarm Optimization (PSO)
-Bee colony opimization based algorithms (BCO)
-Bacterial Foraging Optimization Algorithm (BFOA)
-Simulated Annealing (SA)
-Hill Climbing (HC)
-Tabu Search (TS)
-Large Neighborhood Search (LNS)
-Iterated Local Search (ILS)
-Guided Local Search (GLS)
-Variable neighborhood Search (VNS)
-Greedy Randomized Adaptive Search Procedure (GRASP)

3. Exploration and Exploitation

In optimization and search algorithms:

- Exploration: searching the search space to find new solutions that have not been previously visited.
- Exploitation: improving the current solution by searching the immediate neighborhood.

3. Metaheuristics types

- Two main groups of metaheuristics:
- Single-solution metaheuristics:
  - Preform local search, operate on a single candidate solution and iteratively modify it to explore the search space.
  - Exploitation oriented.
- Population-based/trajectory-based metaheuristics:
  - Maintain a population of candidate solutions throughout the search process.
  - Exploration oriented.

II. SINGLE SOLUTION BASED

1. Introduction
2. Hill Climbing
3. Simulated annealing
4. Tabu search
5. Variable neighborhood search
6. Others Metaheuristics

1. Introduction

- Optimization algorithms that maintain a single solution in memory and iteratively improve it over time.
- Often used for solving continuous optimization problems where the objective function is computationally expensive or non-differentiable.

- In single solution-based metaheuristics:
- Exploration: moving to new solutions that are not necessarily better than the current solution.
  - Allows the algorithm to search a wider range of solutions
  - Avoid getting stuck in local optima.
- Exploitation: achieved by moving to the best solution in the current neighborhood.
  - Allows the algorithm to focus on improving the current solution
  - Converge towards the global optimum.

Single solution-based metaheuristics uses:

- Assembly line balancing problem:
  - A Fuzzy generalized simulated annealing for a simple assembly line balancing problem (2018)
- CNN optimization:
  - Metaheuristic Algorithms for Convolution Neural Network (2016)
- ANN design:
  - Metaheuristic design of feedforward neural networks: a review of two decades of research (2017)
- SVM parameter tunning:
  - Variable Neighborhood Search for parameter tuning in Support Vector Machines (2012)

1. Introduction

- In this part we are going to present some of the single solution based metaheuristics and their working principles

2. Hill Climbing (HC)

1) Work principle
2) Parameters
3) Pseudocode
4) Exploration and Exploitation
5) Variants

2. Hill Climbing (HC)

1) Work principle:

- Simple but effective optimization algorithm.
- Starts with an initial solution and iteratively improves it by making small adjustments to the solution until no further improvement can be made.
- Called hill climbing because it can be visualized as climbing up a hill, where the objective is to reach the highest peak.

2. Hill Climbing (HC)

2) Parameters:

- Initial solution: the starting point of the algorithm,
- Search direction: the direction in which the algorithm makes adjustments to the solution.
- Step size: The step size is the size of the adjustment made to the solution in each iteration.
- Termination criterion
- Neighborhood size: The number of neighboring solutions that the algorithm considers in each iteration.

2. Hill Climbing (HC)

3) Pseudocode:

```
Initialize current_solution
Initialize best_solution = current_solution
while termination criterion is not met do
    Initialize neighbors of current_solution
    for each neighbor in neighbors do
        if neighbor is better than current_solution then
            set current_solution = neighbor
            if current_solution is better than best_solution then
                set best_solution = current_solution
    end for
 end while
return best_solution
```

2. Hill Climbing (HC)

3) Exploration and Exploitation:

- Exploration is performed by generating neighboring solutions and evaluating their quality
- Exploitation is achieved by iteratively moving to the best neighboring solution found so far, until no better solution can be found in the immediate neighborhood.
- The amount of exploitation in hill climbing is controlled by the stopping criterion, which determines when the algorithm should stop searching for better solutions.

2. Hill Climbing (HC)

3) Exploration and Exploitation:

- To balance exploration and exploitation, the choice of neighborhood function and stopping criterion must be carefully chosen.
- The neighborhood function should generate a diverse set of neighboring solutions that have the potential to improve the quality of the current solution, while
- the stopping criterion should be flexible enough to allow for some exploration even after several iterations without improvement.

2. Hill Climbing (HC)

4) Variants:

Steepest-Ascent Hill Climbing

This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state.

This algorithm consumes more time as it searches for multiple neighbors

2. Hill Climbing (HC)

4) Variants:

Stochastic hill climbing.

does not examine for all its neighbor before moving.

this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state based on the heuristic of that node.

2. Hill Climbing (HC)

4) Variants:

Random-restart hill climbing:

restarts the search from a random starting point if the current search does not lead to the global maximum.

2. Hill Climbing (HC)

4) Variants:

Hill climbing with restarts:

Performs multiple iterations of hill climbing with different starting points,

Then returns the best result found among all the iterations.

This approach helps the algorithm to avoid getting stuck in local optima.

2. Hill Climbing (HC)

4) Variants:

Stochastic hill climbing

While basic hill climbing always chooses the steepest uphill move, "stochastic hill climbing chooses at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move.

3. Simulated annealing (SA)

1) Work principle
2) Parameters
3) Pseudocode
4) Exploration and Exploitation
5) Variants

3. Simulated annealing (SA)

1) Work principle

SA is a probabilistic approach that imitates the cooling strategy (annealing process) of a metallurgy industry

- Heat the solid state metal to a high temperature
- Cool it down very slowly according to a specific schedule.
- If the heating temperature is sufficiently high to ensure random state and the cooling process is slow enough to ensure thermal equilibrium, then the atoms will place themselves in a pattern that corresponds to the global energy minimum of a perfect crystal.

3. Simulated annealing (SA)

1) Work principle

Step 1: Initialize – Start with a random initial placement. Initialize a very high "temperature".

Step 2: Move – Perturb the placement through a defined move.

Step 3: Calculate score – calculate the change in the score due to the move made.

Step 4: Choose – Depending on the change in score, accept or reject the move. The prob of acceptance depending on the current "temperature".

Step 5: Update and repeat– Update the temperature value by lowering the temperature. Go back to Step 2.

The process is done until "Freezing Point" is reached.

3. Simulated annealing (SA)

2) Pseudocode

3. Simulated annealing (SA)

Data: the objective function f
Initialization: T_0: initial temperature, T_f: final temperature, $x \leftarrow x_0$: starting point, n ←0:temperature stage, $L \leftarrow L_0$: initialize the length of Metropolis chain
Repeat
  For i=1 to $\boldsymbol{L_n}$
    Generate_solution(x_i, x_{i-1}) {u is a Random vector from the uniform distribution over [0,1)}
    If $f(x_{new}) - f(x) \leq 0$ **then** $x \leftarrow x_{new}$
      else Generate a pseudo-random number u^* {$u^* \in$ [0,1)}
        If $u^* <$ exp($-\frac{f(x_{new})-f(x)}{T_n}$) **then** $x \leftarrow x_{new}$ **End**
    End
  End
  L_{n+1} ← Calculate_Length(L_n)
  T_{n+1} ← Calculate_Temperature(T_n)
Until $T_{n+1} \leq T_f$ indicating that the system is frozen

3. Simulated annealing (SA)

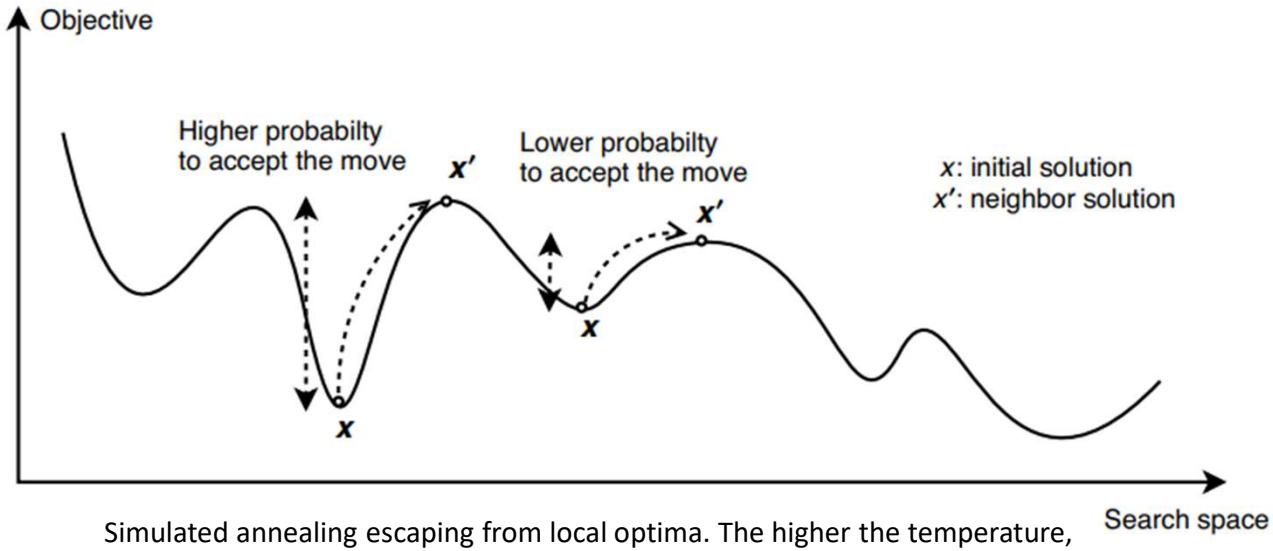
3) Parameters

- Initial temperature : The starting temperature for the algorithm.
- Cooling schedule: The way in which the temperature is gradually reduced during the algorithm.
- Acceptance probability function: Determines the probability of accepting a new solution that has a worse objective function value than the current solution. (Ex Boltzmann distribution)
- Neighborhood structure: The set of possible moves or changes that can be made to the current solution to generate a new solution.
- Stopping criterion.
- Number of iterations.

3. Simulated annealing (SA)

4) Exploration and exploitation

- Temperature is a knob that trade-offs between ‘Exploration’ and ‘Exploitation’.
- Basic idea is to start at high temperature and jump around a lot hopefully seeking the vicinity of the global minimum.
- Then gradually decrease temperature and try to settle on the locally optimal solution



Simulated annealing escaping from local optima. The higher the temperature, the more significant the probability of accepting a worst move. At a given temperature, the lower the increase of the objective function, the more significant the probability of accepting the move. A better move is always accepted.[15]

3. Simulated annealing (SA)

4) Exploration and exploitation

- The temperature schedule starts with a high temperature to allow for exploration
- and gradually decreases to focus the search on the best solution found so far.
- The acceptance probability function determines the probability of accepting a worse solution,
- and thus controls the amount of exploration that occurs during the search.

3. Simulated annealing (SA)

5) Variants

There are many variants and improvements made to the standard SA. Three approaches are applied to improve SA:

- variant based on the cooling schedule,
- variants based on the neighborhood selection, and
- variants based on the learning mechanism.

3. Simulated annealing (SA)

5) Variants

Boltzmann annealing

The standard simulated annealing needs a lot of time to converge to the optimal solution.

It is well-known that the standard SA requires a cooling schedule in which the temperature T must satisfy the following equation:

$$T(k) = \frac{T_0}{\ln(1+k)}$$

The equation characterizes the simplest cooling scheme for SA which is also called Boltzmann annealing.

Boltzmann annealing is called the standard SA and has been used widely in many applications due to its simplicity.

3. Simulated annealing (SA)

5) Variants

Fast annealing

As k becomes very large over the iterations T(k) approaching zero and the Boltzmann cooling scheme becomes slow.

A faster cooling scheme can be achieved by using a Cauchy distribution.

Cauchy distribution is a probability obeying function which is inversely linear in time

$$g(T,k) = \frac{T(k)}{(T(k)^2 + |\Delta x|^2)^{\frac{(D+1)}{2}}}$$

where D denotes the dimension of x, Δx denotes an increment of the state x and Δx denotes the Euclidean norm.

3. Simulated annealing (SA)

5) Variants

Fast annealing

Cauchy distribution has some advantages, e.g. it permits easier access to test local minima.

This method is dubbed fast annealing because it provides an annealing schedule exponentially faster than the Boltzmann annealing.

3. Simulated annealing (SA)

5) Variants

Very fast simulated re-annealing(VFSA)

In VFSA algorithm, the neighborhood function comprises all variables of the entire neighborhood.

An optimization problem with D decision variables has D dimensions.

a cooling schedule for VFSA where the temperature T decreases exponentially in time over the annealing process.

3. Simulated annealing (SA)

5) Variants

Very fast simulated re-annealing(VFSA)

For dimension i (i = 1,…,D), the cooling schedule is defined by following equation:

$$T^i(k) = T_0^i \exp[-c^i k^{\frac{1}{D}}]$$

where c^i is an empirical parameter.

It is apparent that the cooling schedule in VFSA is faster than the Boltzmann annealing due to the exponential term

3. Simulated annealing (SA)

Other Variants

Adaptive SA

Chaotic SA

Hybrid SA

4. Tabu Search (TS)

1) Work principle
2) Parameters
3) Pseudocode
4) Exploration and Exploitation
5) Variants

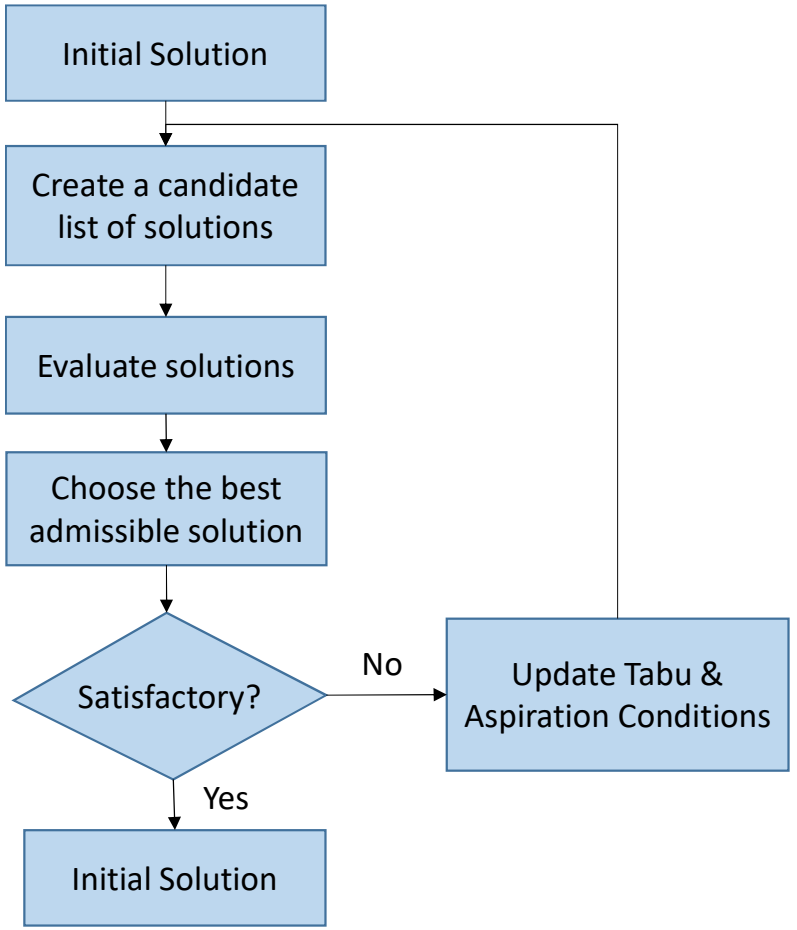
4. Tabu Search (TS)

1) Work principle

- It is a meta-heuristic that guides a local heuristic search procedure to explore the solution space beyond local optimum by use of Tabu list.
- Used to solve combinatorial (finite solution set) optimization problems.
- It uses a flexible memory to restrict the next solution choice to some subset of neighborhood of current solution.

4. Tabu Search (TS)

1) Work principle



Tabu search algorithm flow chart

4. Tabu Search (TS)

2) Parameters

- Tabu list: keeps track of recently visited solutions, and it prevents the algorithm from revisiting them again. The length of the tabu list is a parameter that needs to be set based on the problem size and complexity.
- Neighborhood size: determines the number of neighboring solutions that are considered when generating a new solution.
- Aspiration criterion: Rule that allows the algorithm to revisit a solution that is in the tabu list if it is deemed better than the current best solution. The aspiration criterion can be set based on the problem requirements.

4. Tabu Search (TS)

2) Parameters

- Termination criteria: These are the conditions that determine when the algorithm should stop searching for a solution.
- Initial solution: The quality of the initial solution can affect the speed and quality of the final solution. Different methods can be used to generate an initial solution, such as random initialization or constructive heuristics.

4. Tabu Search (TS)

3) Pseudocode

```
Begin
 t ← 0;
 Initialize tabu search
 While (t<t_max) do
    t ← t + 1;
    Search the neighborhood;
    Evalute candidate solutions;
    Update tabu list
  End
End
```

4. Tabu Search (TS)

4) Exploration and Exploitation

- In tabu search, exploration is achieved by generating new solutions that are not too similar to the current solution, while
- exploitation is achieved by searching in the vicinity of the current best solution to find a better solution.

4. Tabu Search (TS)

4) Exploration and Exploitation

- To balance exploration and exploitation, tabu search uses a neighborhood structure to define the set of neighboring solutions that can be explored.
- The size of the neighborhood is a parameter that can be adjusted to control the balance between exploration and exploitation.

4. Tabu Search (TS)

4) Exploration and Exploitation

At the beginning of the search, the algorithm may focus more on exploration to ensure that a wide range of solutions is explored.

As the search progresses, the algorithm may shift its focus to exploitation to refine the best solutions found so far.

4. Tabu Search (TS)

4) Exploration and Exploitation

- Tabu search also uses a tabu list to prevent the algorithm from revisiting recently explored solutions.
- The tabu list allows the algorithm to explore new solutions while preventing it from getting stuck in local optima.

4. Tabu Search (TS)

5) Variants

Reactive Tabu Search:

This variant adjusts the length of the tabu list during the search based on the quality of the solutions found.

A longer tabu list is used when the algorithm is stuck in a local optima, while a shorter tabu list is used when the algorithm is exploring new regions of the solution space.

4. Tabu Search (TS)

5) Variants

Variable Neighborhood Tabu Search:

Uses multiple neighborhood structures to generate candidate solutions.

The algorithm switches between different neighborhood structures during the search to balance exploration and exploitation.

4. Tabu Search (TS)

5) Variants

Parallel Tabu Search:

uses multiple search threads to explore different regions of the solution space concurrently.

The search threads communicate with each other to exchange information about the best solutions found.

4. Tabu Search (TS)

5) Variants

Hybrid Tabu Search:

This variant combines tabu search with other optimization techniques, such as simulated annealing or genetic algorithms.

The goal is to combine the strengths of different algorithms to improve the overall performance.

4. Tabu Search (TS)

5) Variants

Tabu Search with Intensification and Diversification:

This variant uses different intensification and diversification strategies to balance exploration and exploitation.

For example, the algorithm may use perturbation or randomization to explore new regions of the solution space, or intensify the search in promising regions.

5. Variable neighborhood search (VNS)

1) Work principle
2) Parameters
3) Pseudocode
4) Exploration and Exploitation
5) Variants

5. Variable neighborhood search (VNS)

1) Work principle

Metaheuristic algorithm used for optimization problems that were first introduced by Mladenović and Hansen in 1997.

Uses a combination of local search and randomization to explore the search space efficiently.

The search is carried out in a set of neighborhoods of a current solution.

Each neighborhood defines a specific way of modifying the current solution.

5. Variable neighborhood search (VNS)

1) Work principle

- The algorithm starts with an initial solution and a set of neighborhoods.
- At each iteration, the algorithm generates a new solution by applying a random perturbation to the current solution within the current neighborhood.
- If the new solution is better than the current solution, the algorithm accepts it as the new solution.
- Otherwise, the algorithm moves to a different neighborhood and repeats the process.
- The algorithm continues until a stopping criterion is met,

5. Variable neighborhood search (VNS)

2) Parameters

- Neighborhoods: a set of neighborhoods to explore the search space.
- Initial solution
- Perturbation: The perturbation operator determines how much the solution changes when moving to a new neighborhood.
- Local search: Determines how the algorithm explores each neighborhood. The local search algorithm can be based on any appropriate heuristic algorithm.
- Stopping criteria
- Acceptance criterion: Determines whether to accept a new solution or not.

5. Variable neighborhood search (VNS)

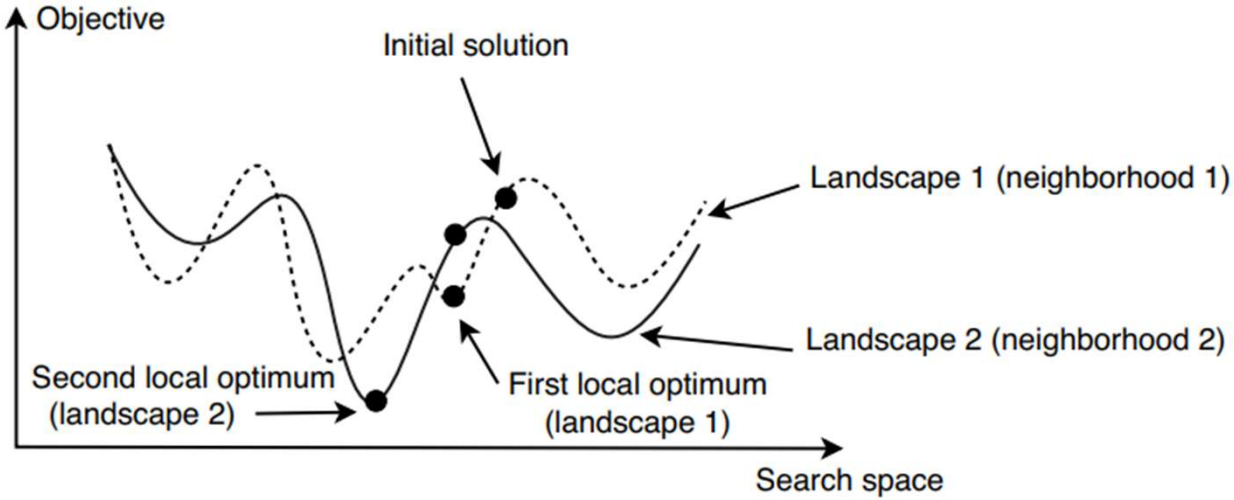
3) Pseudocode

Input: a set of neighborhood structures N_k for $k = 1, \dots, k_{max}$ for shaking,
Generate initial solution $x = x_0$
Repeat
  $k = 1;$
  Repeat
    Shaking: pick a random solution x' from the k^{th} neighborhood $N_k(x)$ of x;
    $x'' =$local search(x');
    If $f(x'') < f(x)$ **then**
      $x = x'';$
      Continue to search with N_1; $k = 1;$
    Otherwize $k = k + 1;$
  Until $k = k_{max}$
Until Stopping criteria
Output: Best found solution

5. Variable neighborhood search (VNS)

4) Exploration and Exploitation

- In VNS, exploration is achieved by randomly selecting a new neighborhood at each iteration and applying the perturbation operator to generate a new solution.
- By exploring new neighborhoods, the algorithm can discover new and potentially better solutions.



Variable neighborhood search using two neighborhoods. The first local optimum is obtained according to the neighborhood 1. According to the neighborhood 2, the second local optimum is obtained from the first local optimum. [15]

5. Variable neighborhood search (VNS)

4) Exploration and Exploitation

- Exploitation is achieved by applying a local search algorithm to each neighborhood.
- The local search algorithm tries to improve the solution within the current neighborhood by iteratively making small improvements until no further improvement is possible.

5. Variable neighborhood search (VNS)

5) Variants

Generalized Variable Neighborhood Search (GVNS):

Extends the algorithm by allowing the neighborhoods to change during the search.

GVNS adds an extra level of iteration, where the algorithm selects a new set of neighborhoods at each iteration.

This allows the algorithm to dynamically adjust the set of neighborhoods during the search and improve its ability to explore the search space.

5. Variable neighborhood search (VNS)

5) Variants

Variable Depth Neighborhood Search (VDNS):

The algorithm tries to improve the solution by exploring neighborhoods of different depths.

The algorithm starts by exploring neighborhoods that are close to the current solution and gradually moves to neighborhoods that are further away.

This approach can help the algorithm avoid getting stuck in local optima by allowing it to explore neighborhoods that are further from the current solution.

5. Variable neighborhood search (VNS)

5) Variants

Adaptive Variable Neighborhood Search (AVNS):

This variant of VNS dynamically adjusts the neighborhood selection during the search based on the algorithm's performance.

The algorithm uses a learning mechanism to adapt the neighborhood selection based on the quality of the solutions found so far.

This allows the algorithm to adjust the search strategy and focus on promising neighborhoods.

5. Variable neighborhood search (VNS)

5) Variants

Variable Neighborhood Decomposition Search (VNDS):

In this variant of VNS, the algorithm decomposes the problem into smaller sub-problems that can be solved independently.

The algorithm applies VNS to each sub-problem separately and combines the solutions to obtain the final solution.

This approach can improve the algorithm's performance by exploiting the problem's structure and reducing the search space.

5. Variable neighborhood search (VNS)

5) Variants

Large Neighborhood Search (LNS):

This variant of VNS focuses on the exploration phase by generating large perturbations to the current solution.

LNS generates large perturbations by removing a subset of the solution's elements and rebuilding a new solution from scratch.

This approach can help the algorithm explore new regions of the search space and escape from local optima.

6. Others Metaheuristics

Iterated local search

Guided local search

GRASP

III. POPULATION BASED

1. Introduction
2. The Biology of Evolutionary Systems
3. Overview of EA

1. Introduction

In this part we are going to present a famous population based which are evolutionary algorithms

We are going to present the working principes, and the algorithm parameters

2. The Biology of Evolutionary Computation

Evolutionary computation is inspired by the mechanisms of biological evolution

Evolution can be described as a process by which individuals become ‘fitter’ in different environments through adaptation, natural selection, and selective breeding.

2. The Biology of Evolutionary Computation

In evolutionary computation, we try to model these principles to find the best solution to a problem.

Each possible solution to a problem is represented as an individual in a pool of a population, where we perform adaptation, natural selection, and selective breeding on those possible solutions to ultimately find the best solution for the problem.

2. The Biology of Evolutionary Computation

Natural evolution does not have a predefined goal and is essentially an open-ended adaptation process,

Artificial evolution is an optimization process that attempts to find solutions to predefined problems.

Therefore, while in natural evolution the fitness of an individual is defined by its reproductive success (number of offspring),

In artificial evolution the fitness of an individual is a function that measures how well that individual solves a predefined problem.

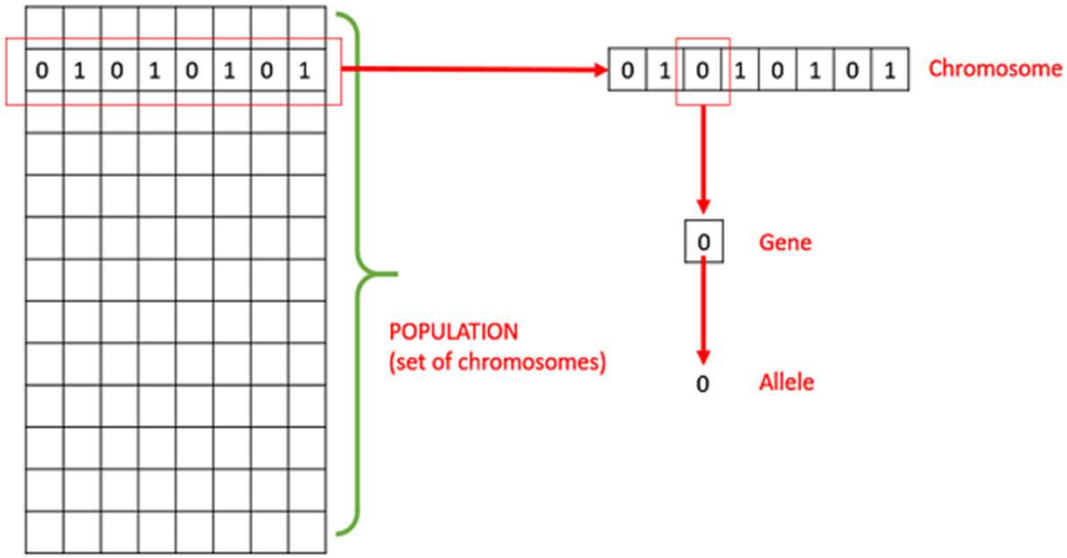
3. Overview of EA

1) Basic Terminology
2) EA Steps
3) Chromosome representation
4) Population
5) Fitness Function
6) Selection Techniques
7) Reproduction
8) Stoping Criterion
9) Pseudocode
10) Evolutionay Algorithms Timeline
11) Standard Evolutionay Algorithms

3. Overview of EA

1) Basic Terminology

- **Population** :subset of all the possible (encoded) solutions to the given problem.
- **Chromosome** :is one such solution to the given problem.
- **Gene :**one element position of a chromosome.
- **Allele :** the value a gene takes for a particular chromosome



3. Overview of EA

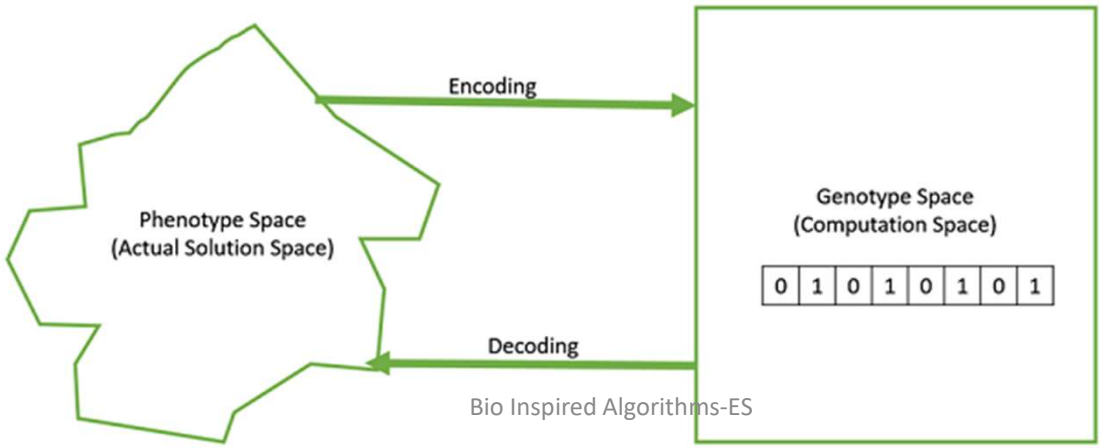
1) Basic Terminology

- **Genotype :**the population in the computation space. Represented in a way which can be easily understood and manipulated using a computing system.
- **Phenotype :** the population in the actual real world solution space, represented in a way they are represented in real world situations.

3. Overview of EA

1) Basic Terminology

- **Decoding** is a process of transforming a solution from the genotype to the phenotype space,
- **Encoding** is a process of transforming from the phenotype to genotype space.



3. Overview of EA

1) Basic Terminology

- **Fitness Function:** a function which takes the solution as input and produces the suitability of the solution as the output.
- **Genetic Operators :** Include crossover, mutation,cloning, selection, etc.

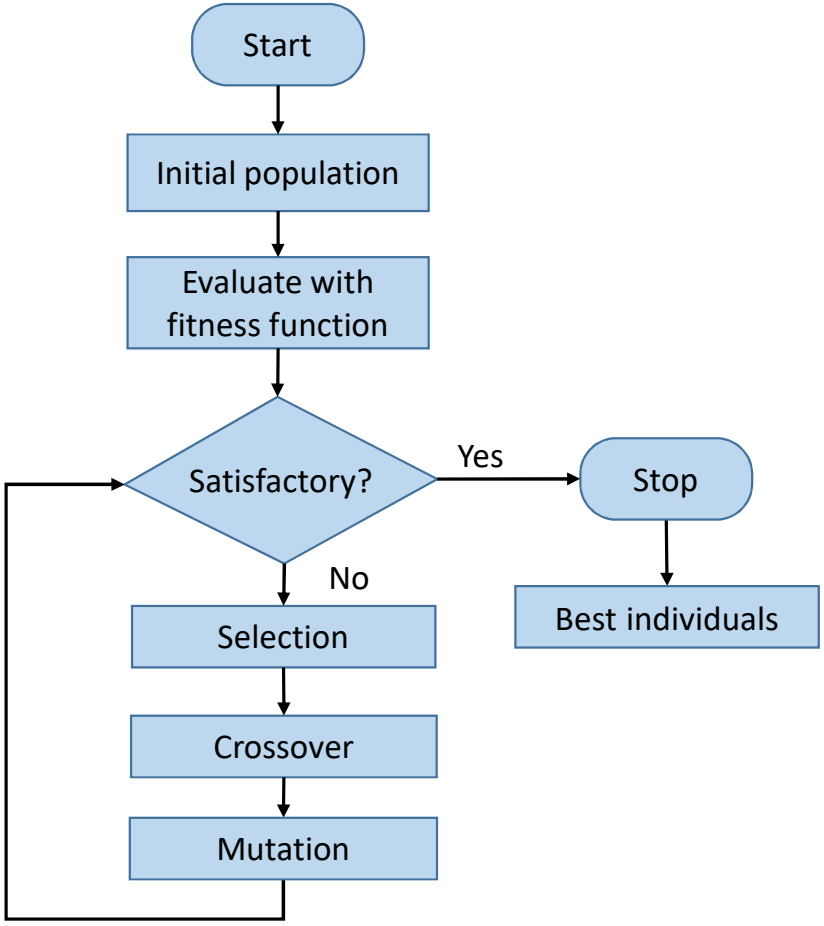
3. Overview of EA

2) EA Steps

Step 1: Generate the initial **population** of **individuals** randomly. (First generation)

Step 2: Repeat the following steps until termination:

- Evaluate the **fitness** of each individual in the population
- Select the fittest individuals for **reproduction**. (Parents)
- **Generate** new individuals through **crossover** and **mutation** operations to give birth to **offspring**.
- Replace the least-fit individuals of the population with new individuals.



Evolutionary Algorithm steps

3. Overview of EA

3) Chromosome Representation

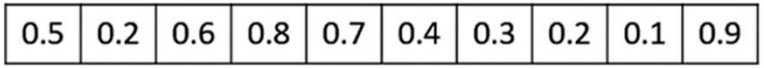
- The representation that we will use to represent our solutions.
- It has been observed that improper representation can lead to poor performance of EA

3. Overview of EA

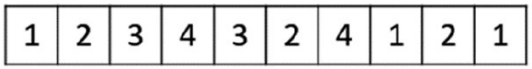
3) Chromosome Representation

- The most commonly used representations
  - Binary Representation

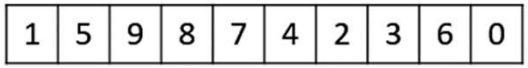
  - Real Valued Representation



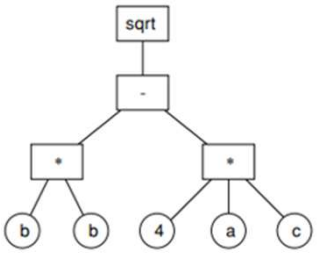
  - Integer Representation



  - Permutation Representation

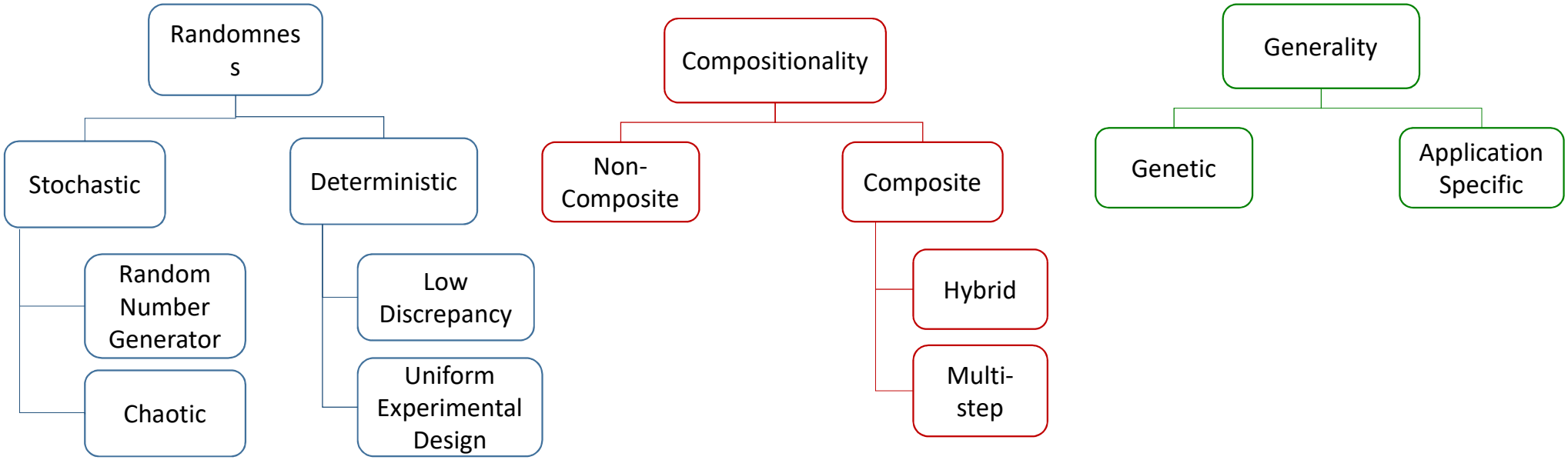


  - Tree



3. Overview of EA

4) Population



Three categorizations of population initialization techniques

3. Overview of EA

5) Fitness Function

- Function which takes a candidate solution to the problem as input and produces as output how “fit” or “good” the solution is with respect to the problem.
- Calculation of fitness value is done repeatedly in a EA, it should be sufficiently fast.
- In most cases the fitness function and the objective function are the same as the objective is to either maximize or minimize the given objective function.

3. Overview of EA

6) Selection Techniques

1. Introduction
2. Fitness Proportionate Selection
3. Tournament Selection
4. Rank Selection
5. Random Selection

3. Overview of EA

6) Selection Techniques

1. Introduction

- Selection is one of the main operators in EAs,
- relates directly to the natural selection concept of survival of the fittest.
- determines whether the particular individual will participate in the reproduction process or not.
- The main objective of selection operators is to emphasize better solutions

3. Overview of EA

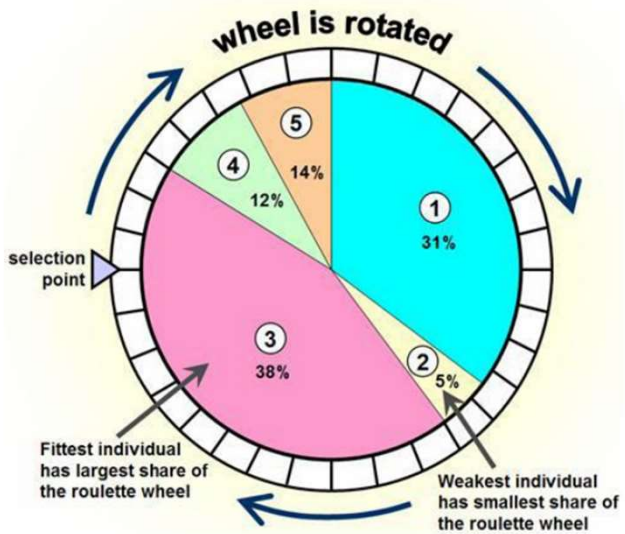
6) Selection Techniques

2. Roulette Wheel Selection (RWS)

In a population with $\boldsymbol{n}$ individuals, for each chromosome $\boldsymbol{x}$ with a corresponding fitness value $\boldsymbol{f_x}$

its probability of being selected is :

$$\boldsymbol{p_x} = \frac{\boldsymbol{f_x}}{\sum_{i=1}^{n} f_i}$$



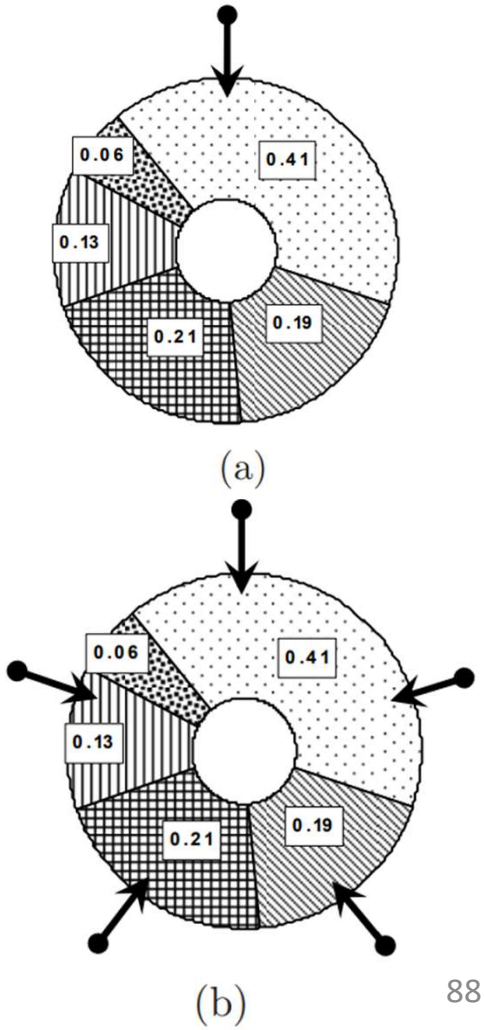
Roulette wheel selection method

3. Overview of EA

6) Selection Techniques

3. Stochastic Universal Sampling (SUS)

- Based on RWS.
- Instead of having one fixed point, we have multiple fixed points.
- All the parents are chosen in just one spin of the wheel.



Fitness Proportionate Selection.
(a) Roulette Wheel Sampling.
(b) Stochastic Universal Sampling

3. Overview of EA

6) Selection Techniques

4. Tournament Selection

- Selects a group of individuals randomly from the population.
- The performance of the selected individuals is compared
- the best individual from this group is selected and returned by the operator.

3. Overview of EA

6) Selection Techniques

5. Random Selection

We randomly select parents from the existing population. There is no selection pressure towards fitter individuals.

3. Overview of EA

6) Selection Techniques

6. Rank-based Selection

- Individuals in the population are ranked according to their fitness.
- The selection of the parents depends on the rank of each individual and not the fitness.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

3. Overview of EA

6) Selection Techniques

7. Elitism

Ensure that the best individuals of the current population survive to the next generation.

The best individuals are copied to the new population without applying reproduction.

3. Overview of EA

6) Selection Techniques

8. Other:

- Boltzmann selection
- Steady State Selection
- Hall of Fame
- Truncation selection
- Disruptive selection
- Survivor/replacement selection

3. Overview of EA

7) Reproduction

1. Introduction
2. Crossover operators
3. Mutation operators
4. Clone operators

3. Overview of EA

7) Reproduction

1. Introduction:

Reproduction is controlled by mutation and crossover operators.

Crossover defines the procedure for generating a child from two parents. Before the actual crossover is performed, the parents need to be selected.

3. Overview of EA

7) Reproduction

1. Introduction:

The mutation operation on the other hand introduces (totally new) parameter values into the genome which have never been used before.

The reproduction process is to allow the genetic information, stored in the good fitness for survive the next generation of the artificial strings, whereas the population's string has assigned a value and its aptitude in the object function.

3. Overview of EA

7) Reproduction

2. Crossover operators

a) Introduction
b) One Point Crossover
c) Multi Point Crossover
d) Uniform Crossover
e) Other crossover operators

3. Overview of EA

7) Reproduction

2. Crossover operators

a) Introduction

- Crossover use several parents (often two) to create one or more offspring using the genetic material of the parents
- Crossover usually applied in EA with a probability (p_c)
- The type of the operator depends on the type of encoding

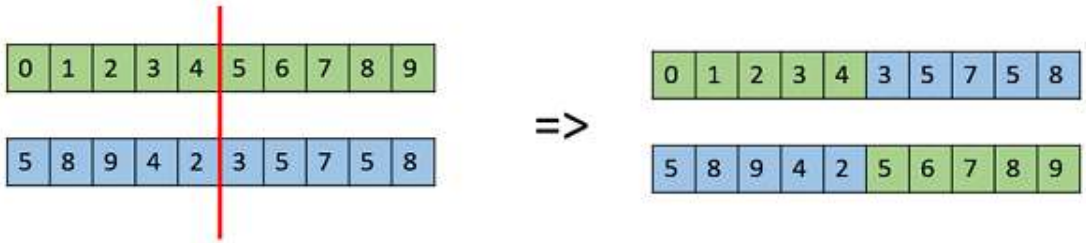
3. Overview of EA

7) Reproduction

2. Crossover operators

b) One Point Crossover

- a random crossover point is selected and
- the tails of the 2 parents are swapped to get new off-springs



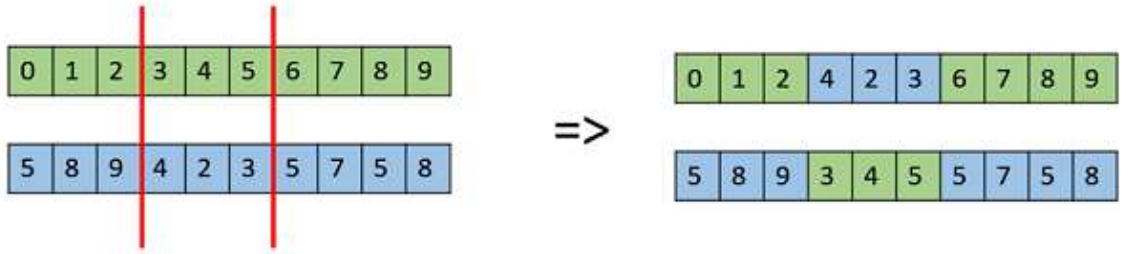
3. Overview of EA

7) Reproduction

2. Crossover operators

c) Multi Point Crossover

- consists of randomly selecting n crossover points on the two strings
- exchanging genetic material that falls between these points.



3. Overview of EA

7) Reproduction

2. Crossover operators

d) Uniform Crossover

consists of exchanging the genetic content at n randomly chosen positions

3. Overview of EA

7) Reproduction

2. Crossover operators

e) Other crossover operators

- Half-Uniform Crossover (HUX)
- Uniform with mask Crossover
- Partially mapped Crossover (PMX)
- Matrix Crossover
- Order Crossover (OX)
- Precedence preserving Crossover (PPX)
- Shuffle Crossover
- Reduced Surrogate Crossover (RCX)
- Three-Parent Crossover
- …

3. Overview of EA

7) Reproduction

3. Mutation operators

a) Introduction
b) Bit Flip Mutation
c) Swap Mutation
d) Scramble Mutation
e) Inversion Mutation
f) Other Mutation operators

3. Overview of EA

7) Reproduction

3. Mutation operators

a) Introduction

- Mutation is a small random tweak in the chromosome, to get a new solution
- used to maintain and introduce diversity in the genetic population
- applied with a low probability ($\boldsymbol{p_m}$=0.01 to 0.1) called mutation rate

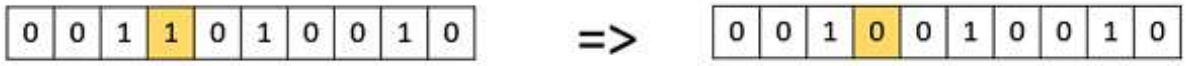
3. Overview of EA

7) Reproduction

3. Mutation operators

b) Bit Flip Mutation

- Select one or more random bits and flip them.
- Used for binary encoded GAs.



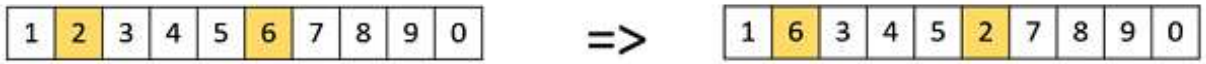
3. Overview of EA

7) Reproduction

3. Mutation operators

c) Swap Mutation

- Select two positions on the chromosome at random, and interchange the values.
- Common in permutation based encodings.



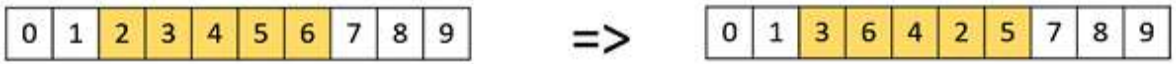
3. Overview of EA

7) Reproduction

3. Mutation operators

d) Scramble Mutation

- A subset of genes is chosen from the entire chromosome
- Their values are scrambled or shuffled randomly.



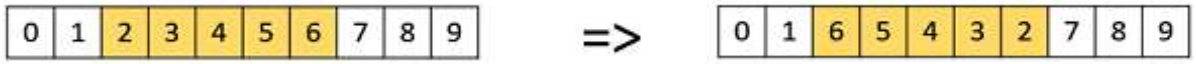
3. Overview of EA

7) Reproduction

3. Mutation operators

e) Inversion Mutation

Instead of shuffling the subset, we invert the entire string in the subset



3. Overview of EA

7) Reproduction

3. Mutation operators

f) Other Mutation operators

- Displacement Mutation
- Scramble Mutation
- Interchanging Mutation
- Creep Mutation
- Simple Sum Coding Mutation
- Cycle Sum Coding Mutation
- …

3. Overview of EA

7) Reproduction

4. Clone operator

Produce genetically identical copies of a biological entity.

The copied material, which has the same genetic makeup as the original, is referred to as a clone.

3. Overview of EA

8) Stopping Criterion

Usually, we keep one of the following termination conditions

- When there has been no improvement in the population for X iterations.
- When we reach an absolute number of generations.
- When the objective function value has reached a certain pre-defined value.

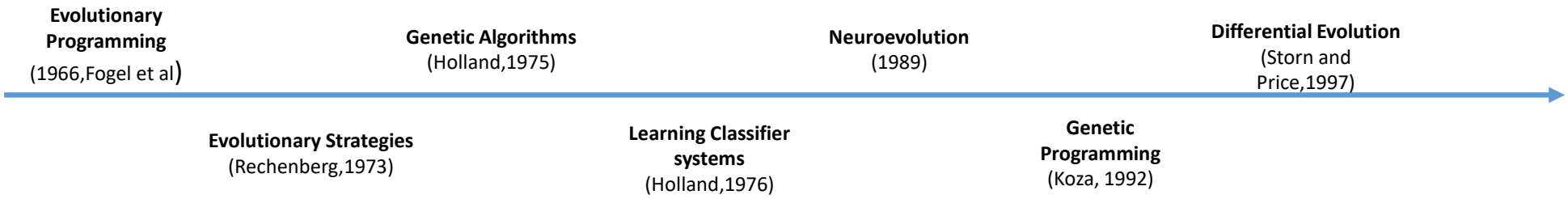
3. Overview of EA

9) Pseudocode

```
t=0;
initialize(P(t=0));
evaluate(P(t=0));
while not stop_condition do
    P'(t) = P(t).selectParents();
    Pc(t) = crossover(P'(t));
    mutate(Pc(t));
    evaluate(Pc(t));
    P(t+1)=selectNextGenerationFrom(Pc(t),P(t));
    t=t+1;
end
```

3. Overview of EA

10)Evolutionay Algorithms Timeline



3. Overview of EA

11)Standard Evolutionay Algorithms

- **Genetic algorithm** :This is the most popular type of EA. The solution of a problem in the form of strings of numbers ,often used in optimization problems.
- **Genetic programming**: Solutions are in the form of computer programs, and their fitness is determined by their ability to solve a computational problem.
- **Evolutionary programming**: Similar to genetic programming, but the structure of the program is fixed and its numerical parameters are allowed to evolve.
- **Evolution strategy**: Works with vectors of real numbers as representations of solutions, and typically uses self-adaptive mutation rates.

3. Overview of EA

11) Standard EA

	Evolutionary Programming(EP)	Evolution Strategy(ES)	Genetic Algorithm(GA)	Genetic Programming(GP)
Representation	Real valued vectors	Real valued vectors	Bit-strings	Tree structures
Crossover	None	Discrete or intermediary	1 point crossover	Exchange of subtrees
Mutation	Gaussian perturbation	Gaussian perturbation	Bit flip	Random change in trees
Parent selection	Deterministic(each parent creates one offpring via mutation)	Uniform random	Fitness proportional(Roulette Wheel)	Fitness proportional
Survival selection	Probabilistic	Deterministic elitist replacement	Generational	Generational replacement

IV. PERFORMANCE IMPROVEMENT

1. Introduction
2. Exploration and exploitation
3. Parameter control techniques
4. Parameter control using Machine learning
5. Hybridization
6. Novel genetic operators

1. Introduction

In this paragrapher we are going to define one of the most important terms in EA

Some techniques used to improve the performance of those algorithms

2. Exploration and Exploitation

An evolutionary search is categorized by two terms: exploration and exploitation.

- The former term is connected with a discovering of the new solutions,
- The later with a search in the neighborhood of knowing good solutions.

Both terms, interweave each other in the evolutionary search.

2. Exploration and Exploitation

The evolutionary search acts correctly when a sufficient diversity of population is present.

The population diversity can be measured differently:

- the number of different fitness values,
- the number of different genotypes,
- the number of different phenotypes, entropy, etc.

2. Exploration and Exploitation

- The higher the population diversity, the better exploration can be expected.
- Losing of population diversity can lead to the premature convergence.

2. Exploration and Exploitation

Exploration and exploitation of evolutionary algorithms are controlled by the control parameters, for instance

- the population size,
- the probability of mutation pm,
- the probability of crossover pc, and
- the tournament size, etc

To avoid a wrong setting of these, different control techniques can be used.

3. Parameter control techniques

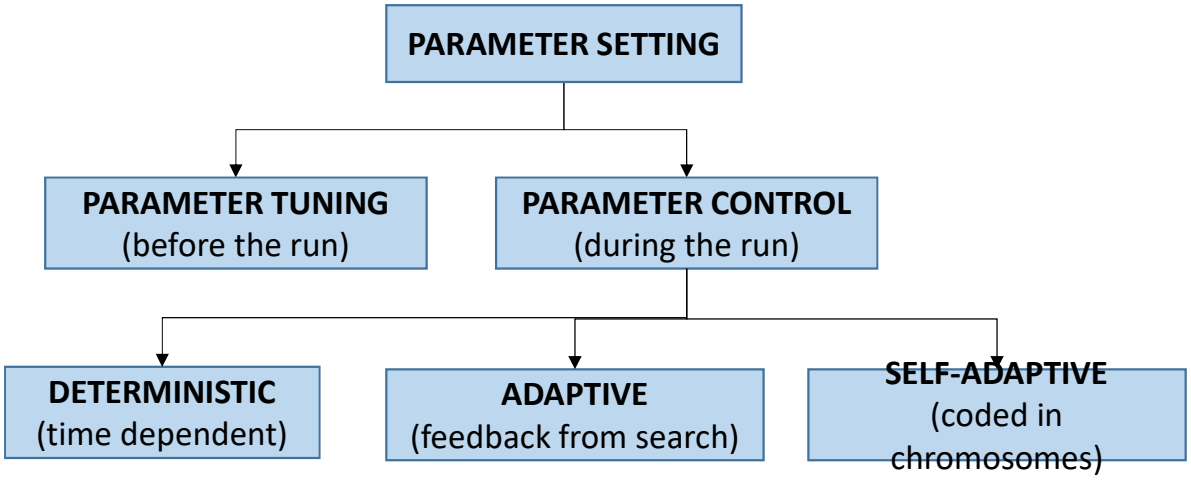
1) Introduction
2) Deterministic parameter control
3) Self-Adaptive parameter control
4) Adaptive parameter control

3. Parameter control techniques

1) Introduction

The distinguishing feature of different parameter control methods is the way that parameters are adapted during the search process.

Rules for changing parameter values can be classified into three categories:



3. Parameter control techniques

1) Introduction

Parameter adjustment approaches can be divided into 2 groups:

Tuning methods:

- The values of the parameters are adjusted before the execution of the algorithm,
- The chosen values are kept constant during the whole execution of the optimization process.

3. Parameter control techniques

1) Introduction

Control methods:

- The values of the parameters are adjusted during the execution of the algorithm.
- Usually, in these techniques, the control method chooses a set of values for the optimization algorithm, so that it will run for a given amount of time and, then, return a performance measure.
- The control method is then able to know how good that choice was.
- These steps are repeated iteratively, always trying to maximize the chances of success of the algorithm by making the best choice for each moment of the optimization process.

3. Parameter control techniques

2) Deterministic/uniformed parameter control

the parameters are adjusted according to a fixed time scheme, explicitly depending on the number of generations t.

without using feedback from the search similar to the way that the cooling schedule is applied in simulating annealing.

It may be useful to reduce the mutation strengths during the evolutionary search, in order to allow convergence of the population.

3. Parameter control techniques

2) Deterministic/uniformed parameter control

The problem then lies in devising such a schedule based on the total number of iterations that the optimization algorithm is likely to take before it converges, which is hard to predict before the run

3. Parameter control techniques

3) Self-Adaptive parameter control

Encodes parameters into individuals and undergoes mutation and recombination (simultaneous evolution of evolution).

High-fitness solutions survive to the next generation and propagate their superior traits along with the parameter values assumed responsible for high quality.

3. Parameter control techniques

3) Self-Adaptive parameter control

- Including the parameter space in the search space extends the solution size and increases the complexity of the optimisation problem.
- the search for optimal parameter values may be subject to premature convergence, which is a common problem faced by the search for optimal solutions.

3. Parameter control techniques

3) Self-Adaptive parameter control

There are different strategy parameters in self-adaptive EAs, e.g.,

- probability of mutation pm,
- probability of crossover pc,
- population size Np, etc.

3. Parameter control techniques

4) Adaptive parameter control

1) Introduction
2) Feedback Collection
3) Effect Assessment
4) Quality Attribution
5) Parameter Update

3. Parameter control techniques

4) Adaptive parameter control

1. Introduction

- Utilizes the feedback of an evolution process to control the directions and magnitudes of the parameters
- Separates the search for optimal solutions from the search for ideal parameter values
- These methods monitor properties of an EA run, such as the quality of the solutions.

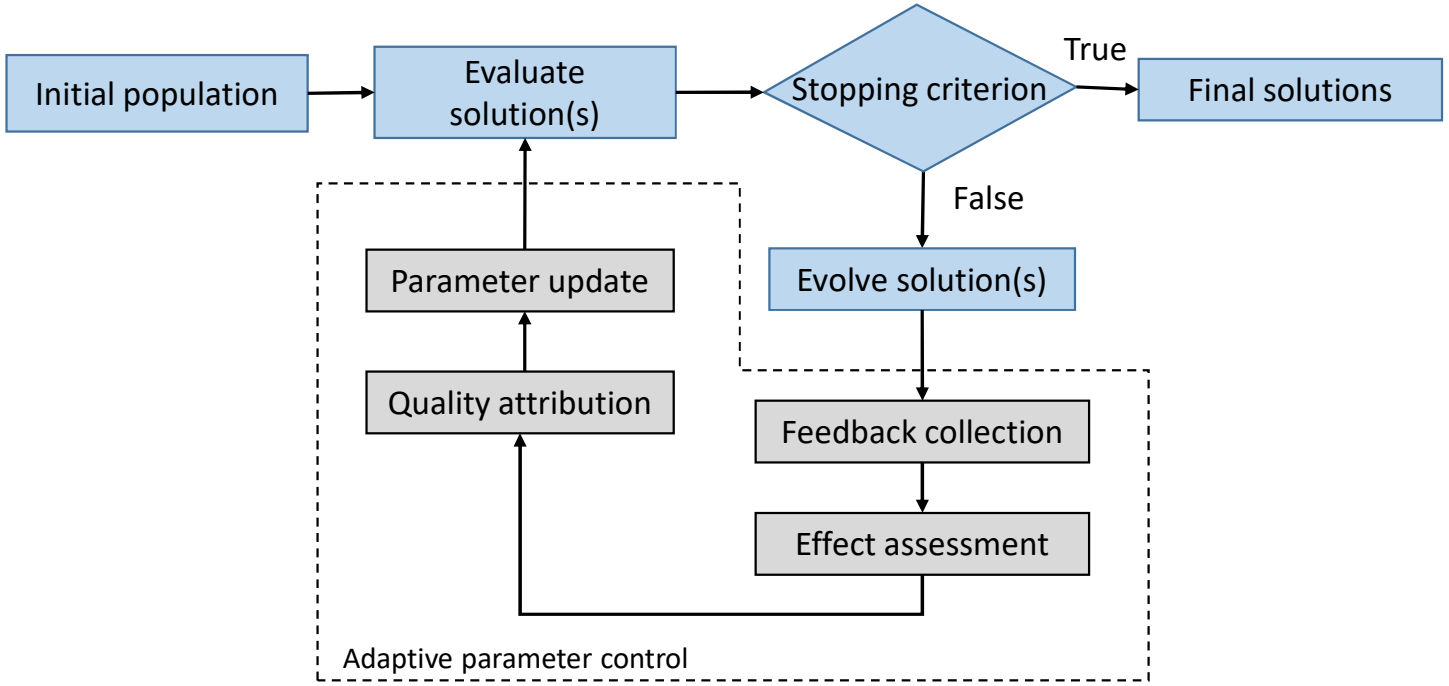
3. Parameter control techniques

4) Adaptive parameter control

- The change in algorithm properties is used as a guide to adjust parameter values in the following iterations.
- Unlike in self-adaptive parameter control, the update mechanism and the selection of the ideal parameter values are not part of the optimisation cycle.

3. Parameter control techniques

4) Adaptive parameter control



A conceptual model of adaptive parameter control in EAs

3. Parameter control techniques

4) Adaptive parameter control

2. Feedback Collection

- a distinguishing feature of adaptive parameter control,
- The feedback is a measure of certain properties of an EA, which are an indication of the algorithm's behaviour. (fitness, diversity, violation of constraints, quality of solutions…)
- The interpretation of the quality of the parameter values depends on the information provided by the feedback collection strategy, which makes it a very important step of adaptive parameter control.

3. Parameter control techniques

4) Adaptive parameter control

3. Effect Assessment

- The parameter effect assessment strategy derives the effect of parameter values based on the output from the feedback collection strategy.
- At each time step, the effect of each parameter value on the performance of the algorithm is estimated using the parameter effect assessment strategy.

3. Parameter control techniques

4) Adaptive parameter control

3. Effect Assessment

- The main difference between the available parameter effect assessment methods is how they define the ‘success’ of parameter values.
- Most of these methods calculate the effect of parameter values using the quality difference of the generated solutions compared to their parents, the overall best solution, a specific quantile (e.g., the median), or outlier solutions.

3. Parameter control techniques

4) Adaptive parameter control

4. Quality Attribution

- To make a better judgement on what would be a successful parameter value for the next iterations, current adaptive parameter control methods define a quality measure, which is
- calculated based on predefined rules that use the effect measured in the previous iterations.
- Whenever a parameter value is used, its current quality estimate is updated by the parameter quality attribution strategy.

3. Parameter control techniques

4) Adaptive parameter control

4. Quality Attribution

- For instance, if a mutation rate of 0.01 produces a solution that changes the fitness of its parent by Δf, the fitness improvement Δf is considered as the quality of this mutation rate.
- If more than one solution is created, the sum or average of the change in the properties of all solutions is attributed as the overall quality
- Other methods learn the quality of a parameter value as the average effect , or as the best value in the last W iterations , with the goal of reducing noise and uncertainty.

3. Parameter control techniques

4) Adaptive parameter control

5. Parameter Update

- The final stage of adaptive parameter control is the update of parameter values to use in the next iteration.
- The update is based on the trade-off between using parameter values with high quality and exploring new values.

3. Parameter control techniques

4) Adaptive parameter control

5. Parameter Update

- The update mechanism used in adaptive parameter control methods can be classified into the following four categories:
- Quality proportionate update: A probability vector is kept for each parameter value, which is updated based on estimated quality, and determines how frequently a parameter value is used in future iterations.
- Quality proportionate with minimum probability update: similar to the quality proportionate update mechanism, with the difference being that a minimum probability is assigned to each parameter value. This is to ensure that parameter values that do not perform well are not lost, since they may prove successful in future iterations.
- Greedy update: Greedy mechanisms select the best parameter value to use in the next iteration.
- Deterministic update: Methods that are part of this category use a deterministic rule for updating parameter values to use in the next iteration. Quality of parameter values is a direct indicator of what the next value should be.

4. Parameter control using Machine Learning

- Most methods used to create a general parameter control are
- Reinforcement learning or
- Time-series prediction.

4. Parameter control using Machine Learning

- The main disadvantages of the reinforcement learning-based approaches are
- the large set of hyperparameters, to which the performance of the whole process is quite sensitive,
- using a tree-based approximation method for state definition, the tree can get very large,

4. Parameter control using Machine Learning

- The time-series prediction-based approaches provide smaller sets of hyperparameters, but
- use only time-related variables, hence ignoring any information that could be extracted from the population.
- This characteristic gives less approximation power to the models used in these controllers when compared to the reinforcement learning-based ones.

4. Parameter control using Machine Learning

Other models used are:

- Dynamic programming,
- Fuzzy logic,
- Agent-based genetic algorithms,
- Surrogate models and
- Bayesian networks.

5. Hybridization

Evolutionary algorithms are too problem-independent.

They are hybridized with several techniques and heuristics that are capable to incorporate problem-specific knowledge.

5. Hybridization

- hybridization between two evolutionary algorithms,
- neural network assisted evolutionary algorithm,
- fuzzy logic assisted evolutionary algorithm,
- particle swarm optimization assisted evolutionary algorithm,
- ant colony optimization assisted evolutionary algorithm,
- bacterial foraging optimization assisted evolutionary algorithm,
- other heuristics, like local search, tabu search, simulated annealing, hill climbing, dynamic programming, etc.

5. Hybridization

In general, successfully implementation of evolutionary algorithms for solving a given problem depends on incorporated problem-specific knowledge.

All elements of evolutionary algorithms can be hybridized.

5. Hybridization

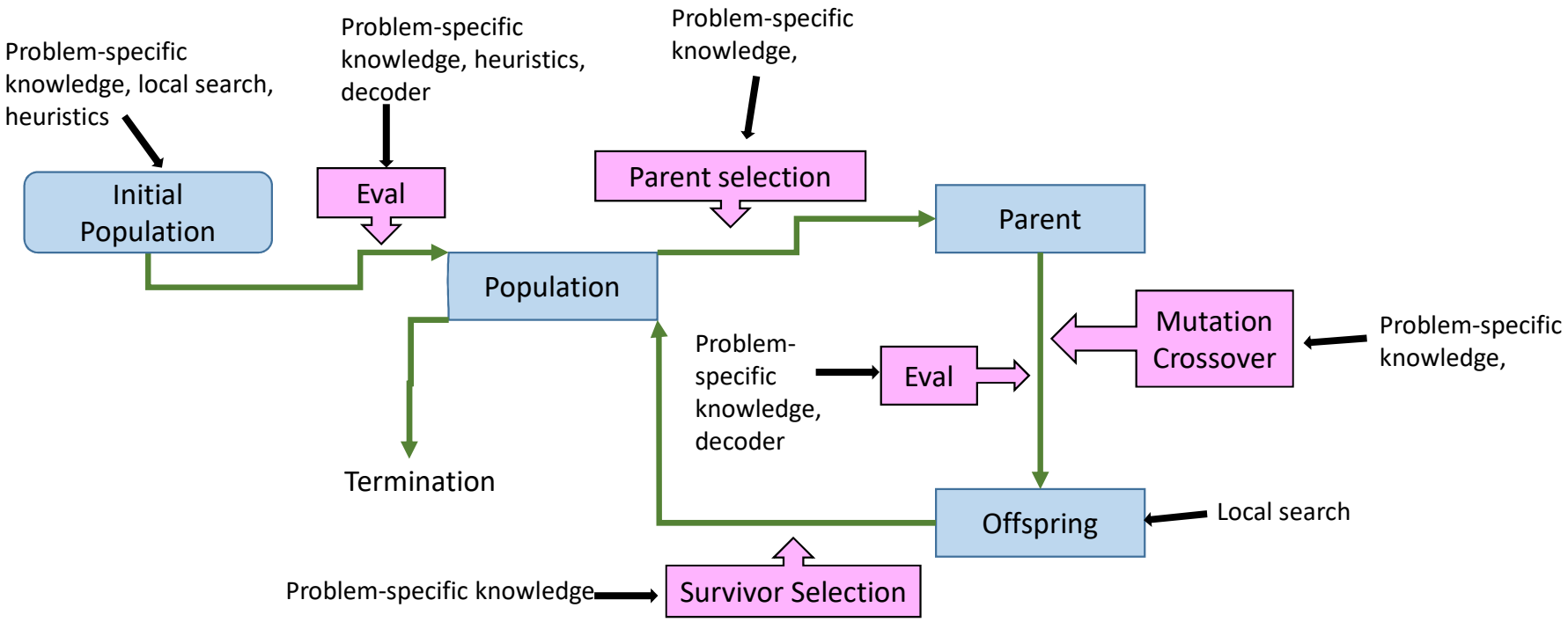
Mostly, a hybridization addresses the following elements of evolutionary algorithms :

- initial population,
- genotype-phenotype mapping,
- evaluation function, and
- variation and selection operators.

5. Hybridization

- The initial population can be generated by incorporating solutions of existing algorithms or by using heuristics, local search, etc.
- The local search can be applied to the population of offsprings.
- Evolutionary operators (mutation, crossover, parent and survivor selection) can incorporate problem-specific knowledge or apply the operators from other algorithms.
- Fitness function offers the most possibilities for a hybridization because it can be used as decoder that decodes the indirect represented genotype into feasible solution.

5. Hybridization



6. Novel genetic operators

Many genetic operators were proposed to improve the balance between exploration and exploitation

For example:

A new split ranked selection operator was presented which is a great trade-off between exploration and exploitation.

6. Novel genetic operators

In the proposed procedure, the individuals are ranked according to their fitness scores from worst to best, thus overcoming the fitness scaling issue.

After this, split the whole population into two portions and assigning them probabilities for selection based on their ranks.

Computational results proved the superior performance of the new selection scheme in comparison with the traditional GA approaches

VII. OPTIMIZATION THEORY

1. Introduction
2. Unconstrained Problems
3. Constrained Problems
4. Multi-Solution Problems
5. Multi-Objective Problems

1. Introduction

- Optimization theory is a branch of mathematics that deals with finding the optimal solution to a problem , given a set of constraints
- To solve an optimization problem we need to know:
  - The Objective Function
  - The Set of Variables
  - The Set of Constraints

2. Unconstrained Problems

- Involves finding the maximum or minimum of a function given a set of variables whose domain is either continuous, discrete, or both.

$$min\ f(x), x = (x_1, x_2, \dots, x_n)$$
$$subject\ to\ \ x_j \in domain(x_j)$$

$Where\ n\ is\ the\ number\ of\ variables$

3. Constrained Problems

Problems for which a function is to be minimized or maximized subject to constraints

$$
\begin{array}{rlll}
min & f(x) & & \\
subject\ to & g_i(x) = c_i & for\ i = 1, \dots, n & Equality\ Constraint \\
 & h_j(x) \geq d_j & for\ j = 1, \dots, m & Inequality\ Constraint
\end{array}
$$

4. Multi-Solution Problems

- Functions where there are no unique global extrema.
- Our goal is to return these multiple solutions so that we have the ability to choose which point or vector of variable values best fits our interest.

5. Multi-Objective Problems

Problems where we have many different objectives, or optimization problems, that we need to solve simultaneously.

$$\min_{x \in \Omega} F(x) = \left(f_1(x), f_2(x), \ldots, f_m(x)\right)^T$$

where Ω the decision space and R^m the objective space (m is the number of objectives).

The objective function $F: \Omega \rightarrow R^m$ consists of m real-valued objective functions.

V. CONSTRAINT HANDLING

1. Introduction
2. Constraint handling techniques

1. Introduction

To solve a constrained multiobjective optimization problem,

an algorithm should tackle the objective functions and the constraints simultaneously.

As a result, many constraint-handling techniques have been proposed

1. Constraint handling techniques

1) Penalty Function Approach
2) Repair Functions
3) Restricting Search to the Feasible Region
4) Decoder Functions
5) Jimenez-Verdegay-Gomez-Skarmeta's Method
6) Constrained Tournament Method
7) Ray-Tai-Seow's Method

2. Constraint handling techniques

1) Penalty Function Approach

- Penalty functions modify the original fitness function
- Minimization of all objective functions is assumed here.

2. Constraint handling techniques

1) Penalty Function Approach

First all constraints are normalized, the resulting constraint functions are $\underline{g_j}\left(x^{(i)}\right) \geq 0$ for $j = 1,2, \ldots, J,$

For each solution $x^{(i)}$, the constraint violation for each constraint is:

$$\omega_j\left(x^{(i)}\right) = \begin{cases} \left|\underline{g_j}(x^{(i)})\right|, & if\ \underline{g_j}\left(x^{(i)}\right) < 0; \\ 0, & otherwise. \end{cases}$$

2. Constraint handling techniques

1) Penalty Function Approach

All the constraint are added together to get the overall constraint:

$$\Omega(x^{(i)}) = \sum_{j=1}^{J} \omega_j\left(x^{(i)}\right),$$

This constraint is then multiplied with a penalty parameter Rm and the product is added to each of the objective function values

$$F_m\left(x^{(i)}\right) = f_m\left(x^{(i)}\right) + R_m\Omega\left(x^{(i)}\right)$$

The functions F_m takes into account the constraint violations.

2. Constraint handling techniques

2) Repair Functions

The use of repair algorithms with EAs can be seen as a special case of adding local search to the EA.

In this case the aim of the local search is to reduce (or remove) the constraint violation, rather than to improve the value of the fitness function,

2. Constraint handling techniques

2) Repair Functions

The use of local search has been intensively researched, with attention focusing on the benefits of Baldwinian versus Lamarckian learning.

In either case, the repair algorithm works by taking an infeasible point and generating a feasible solution based on it.

2. Constraint handling techniques

2) Repair Functions

In the Baldwinian case, the fitness of the repaired solution is allocated to the infeasible point, which is kept.

In the Lamarckian learning, the infeasible solution is overwritten with the new feasible point.

2. Constraint handling techniques

2) Repair Functions

In general defining a repair function may be as complex as solving the problem itself.

2. Constraint handling techniques

3) Restricting Search to the Feasible Region

It may be possible to construct a representation and operators so that the search is confined to the feasible region of the search space.

In constructing such an algorithm, care must be taken in order to ensure that all of the feasible region is capable of being represented.

It is equally desirable that any feasible solution can be reached from any other by (possibly repeated) applications of the mutation operator.

The classic example of this is permutation problems.

2. Constraint handling techniques

3) Restricting Search to the Feasible Region

In many cases it is difficult to find an existing or design a new operator that guarantees that the offspring are feasible.

2. Constraint handling techniques

4) Decoder Functions

Decoder functions are a class of mappings from the genotype space S ′ to the feasible regions F of the solution space S such as:

- Every $z \in S'$ must map to a single solution $s \in F$.
- Every solution $s \in F$ must have at least one representation $s' \in S'$.
- Every $s \in F$ must have the same number of representations in S ′ .

2. Constraint handling techniques

4) Decoder Functions

Such decoder functions provide a relatively simple way of using EAs for this type of problem, but they are not without drawbacks.

These are centred around the fact that decoder functions generally introduce a lot of redundancy into the original genotype space.

This arises when the new mapping is manyto-one, meaning that a number of potentially radically different genotypes may be mapped onto the same phenotype, and only a subset of the phenotype space can be reached.

2. Constraint handling techniques

5) Jimenez-Verdegay-Gomez-Skarmeta's Method

A systematic constraint handling procedure for the multi-objective optimization.

consideration of feasible and infeasible solutions and the use of niching to maintain diversity in the obtained Pareto-optimal solutions.

The algorithm uses the binary tournament selection in its core.

three different cases arise in a two-player tournament.

Both solutions may be feasible, or

both may be infeasible, or

one is feasible and the other is infeasible.

The investigators carefully considered each case and suggested the following strategies.

2. Constraint handling techniques

5) Jimenez-Verdegay-Gomez-Skarmeta's Method

Case 1: Both solutions are feasible.

- The investigators suggested following a procedure similar to the niched Pareto GA
- First, a random set of feasible solutions called a comparison set is picked from the current population.
- Secondly, two solutions chosen for a tournament are compared with this comparison set.
- If one solution is non-dominated in the comparison set and the other is dominated by at least one solution from the comparison set, the former solution is chosen as the winner of the tournament.

2. Constraint handling techniques

5) Jimenez-Verdegay-Gomez-Skarmeta's Method

- Otherwise, if both solutions are non-dominated or both solutions are dominated in the comparison set, both are equally good or bad in terms of their domination with respect to the comparison set.
- Thus, the neighborhood of each solution is checked to find its niche count.
- The solution with a smaller niche count means that the solution is situated in a least crowded area and is chosen as the winner of the tournament.
- The niche count nc is calculated by using the phenotypic distance measure, that is, problem variables are used to compute the Euclidean distance d between two solutions.
- Then, this distance is compared with a given threshold σ_{share}

2. Constraint handling techniques

5) Jimenez-Verdegay-Gomez-Skarmeta's Method

If the distance is smaller than σ $_{share}$, a sharing function value is computed by using equation

$$Sh(d) = \begin{cases} 1 - \left(\dfrac{d}{\sigma_{share}}\right)^{\alpha}, if\ d \leq \sigma_{share}; \\ 0, \qquad otherwise \end{cases}$$

with an exponent value equal to α = 2.

Then, the niche count is calculated by summing the sharing function values calculated with each population member.

2. Constraint handling techniques

5) Jimenez-Verdegay-Gomez-Skarmeta's Method

Case 2: One solution is feasible and other is not.

The feasible solution is declared as the winner.

2. Constraint handling techniques

5) Jimenez-Verdegay-Gomez-Skarmeta's Method

Case 3: Both solutions are infeasible.

- As in the first case, a random set of infeasible solutions is first chosen from the population.
- Secondly, both solutions participating in the tournament are compared with this comparison set.
- If one is better than the best infeasible solution and the other is worse than the best infeasible solution present in the comparison set, the former solution is chosen.
- A suitable criterion can be used for this comparison. For example, the constraint violation or nearness to a constraint boundary can be used.

2. Constraint handling techniques

5) Jimenez-Verdegay-Gomez-Skarmeta's Method

- If both solutions are either better or worse than the best infeasible solution, both of them are considered as being equally good or bad.
- A decision with the help of a niche count calculation is made, as before.
- The niche count is calculated with the phenotypic distance measure (with Xi values) and with respect to the entire population, as described in the first case.
- The one with the smaller niche count wins the tournament.

2. Constraint handling techniques

6) Constrained Tournament Method

Uses the binary tournament selection, where two solutions are picked from the population and the better solution is chosen.

In the presence of constraints, each solution can be either feasible or infeasible.

Thus, there may be at most three situations:

1. both solutions are feasible,
2. one is feasible and other is not
3. both are infeasible.

2. Constraint handling techniques

6) Constrained Tournament Method

For single-objective optimization, we used a simple rule for each case:

Case2 Choose the feasible solution.

Case3 Choose the solution with smaller overall constraint violation.

Case1: We can use the concept of dominance

2. Constraint handling techniques

6) Constrained Tournament Method

When both solutions are feasible, we can check if they belong to separate non-dominated fronts.

In such an event, choose the one that belongs to the better non-dominated front.

If they belong to the same non-dominated front, we can use the diversity preservation task to resolve the tie.

Since maintaining diversity is another goal in multi-objective optimization, we can choose the one which belongs to the least crowded region in that non-dominated set

2. Constraint handling techniques

6) Constrained Tournament Method

Definition(constrain-domination condition):

A solution $x^{(i)}$ is said to 'constrain-dominate' a solution $x^{(j)}$ (or $x^{(i)} \preceq_c x^{(j)}$), if any of the following conditions are true:

1. Solution $x^{(i)}$ is feasible and solution $x^{(j)}$ is not.
2. Solutions $x^{(i)}$ and $x^{(j)}$ are both infeasible, but solution $x^{(i)}$ has a smaller constraint violation.
3. Solutions $x^{(i)}$ and $x^{(j)}$ are feasible and solution $x^{(i)}$ dominates solution $x^{(j)}$ in the usual sense.

2. Constraint handling techniques

7) Ray-Tai-Seow's Method

the constraint violations of all constraints are not simply added together, instead, a non-domination check of the constraint violations is made.

Three different non-dominated rankings of the population are first performed.

The first ranking is performed by using M objective function values, and the resulting ranking is stored in an N-dimensional vector R_{obj}

The second ranking (R_{con}) is performed by using only the constraint violation values of all constraints and no objective function information is used.

2. Constraint handling techniques

7) Ray-Tai-Seow's Method

The constraint violation of each constraint is used as a criterion and a non-domination classification of the population is performed with the constraint violation values.

all feasible solutions have a rank 1 in R_{con}.

Such a ranking procedure will allow solutions violating only one independent constraint to lie in the same non-dominated front.

2. Constraint handling techniques

7) Ray-Tai-Seow's Method

The third ranking is performed by using a combined objective function and constraint violation values ,This produces the ranking R_{com}.

Although objective function values and constraint violations are used together, interestingly there is no need of any penalty parameter.

In the domination check, the criteria are individually compared, thereby eliminating the need of any penalty parameter.

Once these rankings are complete, the following algorithm is used for handling the constraints.

2. Constraint handling techniques

7) Ray-Tai-Seow's Method

Step 1

Using R_{com}, select all feasible solutions having a rank 1. This can be achieved by observing the feasibility of all rank 1 solutions from R_{con} . That is,

$p_i = \{q: R_{com}(q) = 1$ and constraint violation $w_j(q) = 0$, j $= 1,2,\ldots,$ J$\}$.

If $|P'|$ < N, fill population P' by using Step 2.

2. Constraint handling techniques

7) Ray-Tai-Seow's Method

Step 2

Choose a solution A with R_{obj} by assigning more preference to low ranked solutions. Choose its mate by first selecting two solutions Band C by using R_{con}. Then, use the following cases to choose either B or C:

2. Constraint handling techniques

7) Ray-Tai-Seow's Method

Case 1:

If B and C are both feasible, choose the one with better rank in R_{obj} .

If both ranks are the same, a head-count metric is used to decide which of them resides in a less-crowded region in the population.

2. Constraint handling techniques

7) Ray-Tai-Seow's Method

Case 2:

If B and C are both infeasible, choose the one with a better rank in R_{con} .

If both ranks are the same, then use a common-constraint satisfaction metric to choose one of these.

This metric measures the number of constraints that each of B or C satisfies along with A.

The one satisfying the least number of common constraints is chosen.

2. Constraint handling techniques

7) Ray-Tai-Seow's Method

Case 3:

If B is feasible and C is not, choose B; otherwise, choose C.

On the other hand, if B is infeasible and C is feasible, then choose C; otherwise, choose B.

VI. MULTIOBJECTIVE EVOLUTIONARY ALGORITHM

1. Introduction
2. MOEA basic structure
3. Non-dominated Sorting Genetic Algorithm (NSGA-II)
4. Strength Pareto Evolutionary Algorithm 2 (SPEA2)
5. Non-dominated Sorting Genetic Algorithm (NSGA-III)
6. Multi-objective evolutionary algorithm based on decomposition (MOEA/D)

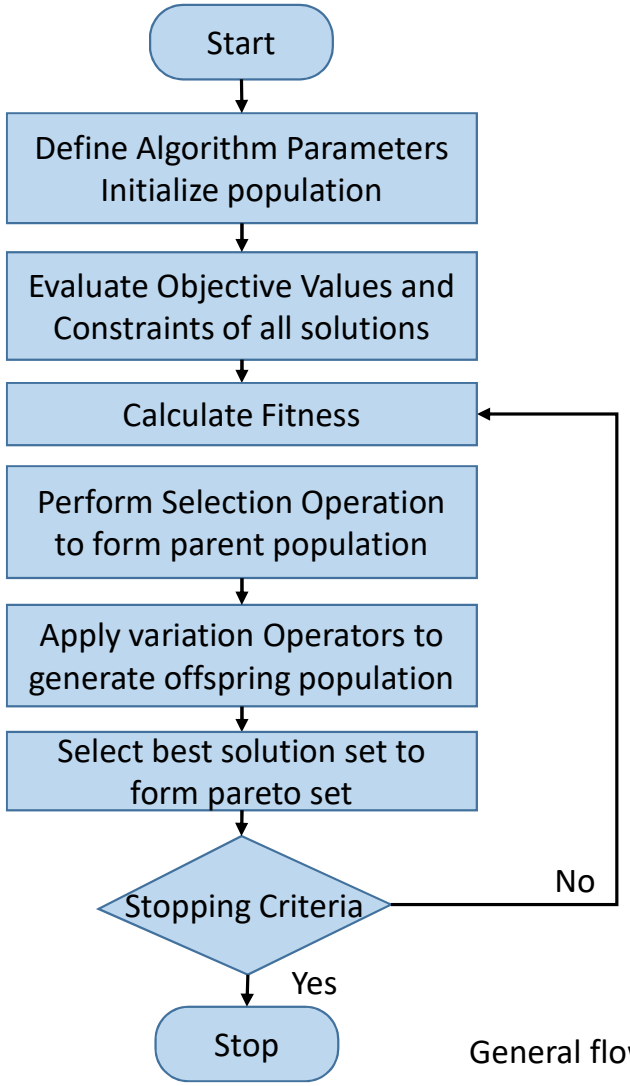
1. Introduction

MOEA used in different areas:

- For SVM construction
  - Using a multi-objective genetic algorithm for SVM construction(2006)
- Jobshop scheduling
  - An Effective Hybrid Multiobjective Flexible Job Shop Scheduling Problem Based on Improved Genetic Algorithm(2022)
- Assembly line balancing
  - Hybrid Multiobjective Evolutionary Algorithm for Assembly Line Balancing Problem with Stochastic Processing Time(2014)
- Neural network optimization
  - Multi-Objective Hyperparameter Optimization for Spiking Neural Network Neuroevolution(2021)

2. MOEA basic structure

There are many different evolutionary algorithms, but they all have the same basic structure



General flow diagram of MOEA[10]

3. Non-dominated Sorting Genetic Algorithm (NSGA-II)

One of the most popular optimization algorithms.

commonly used as a base algorithm for comparing new algorithms.

NSGA-II is a non-dominated sorting approach based algorithm and has low computation time.

It was initially proposed for 2–3 objective problems. But, a number of improvements and extensions have been proposed to make the algorithm efficient for solving multi-objective (MOO) problems with a larger number of objectives.

It is an elitist algorithm.

3. Non-dominated Sorting Genetic Algorithm (NSGA-II)

Algorithm(NSGA-II)
Require: Parameters, Initial Population of size N, Maximum Generation
Set the generation count $i = 0$
Assess the fitness values of the P_0 population and rank them accordingly
while i $\leq$ Maximum number of Generations **do**
    Use crossover and mutation to produce an offspring population Q_i of size N
    Merge parent and offspring population $R_i \leftarrow P_i \cup Q_i$
    Sort R_i and identify different pareto fronts
    Keep elite members in S_i with size N
    Calculate crowding distance between the solutions of S_i
    Perform selection on $S_i$and update population P_{i+1}
    Increment the generation count, $i++$
end while
return Population P_{i+1}

4. Strength Pareto Evolutionary Algorithm 2 (SPEA2)

Proposed to deal with some of the shortcomings of SPEA

The shortcomings are: diversity in problems with more than two objectives, ineffcient fitness assignment, and loss of individual's boundary.

SPEA is the basic algorithm which forms the basis of SPEA2.

It uses non-dominance sorting for selection and clustering to maintain diversity of solutions stored in archive. n-Pareto solutions are stored in an external archive (EA) and if better non-dominated solutions are found, then EA is updated.

The maximum size of an EA is pre-assigned.

If the number of individuals in EA reaches its maximum size, then the set members are decreased by clustering.

4. Strength Pareto Evolutionary Algorithm 2 (SPEA2)

For the successful functioning of SPEA, it is necessary to provide a balance between the size of the population and the external population.

The large EA increases the selection chances for elite individuals, and the convergence of the solutions to the Pareto-optimal front decreases.

Contrastingly, if a small EA is used, the influence of elitism will be suppressed.

Tournament selection is done by merging the population and EA members.

Fitness values are assigned to a combined population, and individuals with the lowest fitness values are selected. So the best individuals get selected for the next iteration

4. Strength Pareto Evolutionary Algorithm 2 (SPEA2)

SPEA2 makes use of neighborhood-based density estimation for the ranking of the individuals.

Also, a better fitness function is used that considers the number of individuals a population member dominates as well as the number of archive members that dominate it.

Moreover, the EA is updated such that the individual count in the EA remains constant, and the boundary solutions of the approximation set are preserved.

Algorithm(SPEA2)

Require: Algorithm Parameters, Initial Population P_0 of size N, Maximum Generation
Set the generation count $i = 0$ and create an empty External Archive EA
while i $\leq$ Maximum number of Generations **do**
  Assess the fitness values of the P_i population and EA_i, and rank them accordingly.
  Combine all non-dominated solutions in P_i and EA_i, to make EA_{i+1}
  if size of $EA_{i+1} > N$ **then**
    Decrease individuals from EA, using truncation operation
  else
    Select dominated individuals from P_i and EA_i, and assign them to EA_{i+1}.
  end if
  if i $\leq$ Maximum number of Generation **then**
    Set EA to EA_i and
    Return Population P_{i+1}
  else
    Perform Selection with a replacement operation on EA_i, to fill the parent population
    Apply crossover and mutation operations to the parent population and set P_i as the Offspring population
  endif
  Increment the generation count, i++
end while

5. Non-dominated Sorting Genetic Algorithm (NSGA-III)

Makes use of reference point based methodology for optimization of multi-objective problems.

NSGA-III uses the basic framework of NSGA-II.

It uses a well-spread reference point mechanism to maintain diversity.

NSGA-III was developed to solve optimization problems with more than four objectives.

5. Non-dominated Sorting Genetic Algorithm (NSGA-III)

The set of reference points is updated dynamically in the process of its decomposition.

The number of reference points corresponds to the number of individuals in the population.

Each individual is associated with a reference point.

The reference points are uniformly dispersed across the normalized hyper-plane and are adaptively updated.

5. Non-dominated Sorting Genetic Algorithm (NSGA-III)

Algorithm(NSGA-III)

Require: Initial Population P_0 of size N, Set of Reference Points H, Maximum Generations

Set the generation count gen = 0

Perform non-dominated sorting on the population and select the parent population P_{gen}

while gen $\leq$ Maximum Generations **do**

- Apply crossover and mutation to population P_{gen} to generate an offspring population Q_{gen}
- Combine the Parent P_{gen} and offspring populations Q_{gen} as $R_{gen} \leftarrow P_{gen} \cup Q_{gen}$
- Use a non-dominated sort to describe various fronts$(F_1, \ldots, F_n)$ in the population R_{gen}
- Preserve top N elite members in S_{gen}
- Adaptively Normalize S_{gen} members
- Assign reference points to the population members S_{gen}
- Apply the niche preservation for S_{gen}
- Assign the niche obtained solutions for the next generation P_{gen+1}
- Increment the generation count, gen$++$

end while

return Population P_{gen}

6. Multi-objective evolutionary algorithm based on decomposition (MOEA/D)

Decomposition-based method

Decomposition maintains the diversity of individuals.

Subproblems can be solved easily compared to the whole problem

Optimization of each sub-problem makes use of neighborhood information.

The principles of neighborhood help in the convergence of individuals towards Pareto front

.

6. Multi-objective evolutionary algorithm based on decomposition (MOEA/D)

To solve a problem using MOEA/D, we decomposed it into sub-problems that are associated with weight vectors.

The neighborhood links between the sub-problems is determined as per the Euclidean distance in their weight vectors.

Weight vectors, initial population, the neighborhood of each sub-problem, and associated reference points are provided as input to the algorithm.

Algorithm(MOEA/D)

Require: Set of Reference Points Z^*, #Subproblems, N
weight Vectors $W = w_1, \dots, w_N$
Generate initial population $X = x_1, \dots, x_N$
Set the generation count gen = 0 and External Population EP =0
Compute C nearest weight vectors for each weight vector
for i=1 to N **do**
    Set NB(i)=i_1 to i_C
    Here w_{i1} to w_{ic} are C nearest weight vectors to w_i
end for
Initialize reference points associated with individuals of population $z = (z_1, \dots, z_m)^C$, where z contains the best value for objective f_i
while gen ≤ Maximum Generations **do**
    for i=1 to N **do**
        Select two indexes a, b randomly from NB(i)
        Perform crossover and mutation on $x_a$and $x_b$and store in y_i
        y_i←crossover(x_a, x_b), mutation(x_a, x_b)
    end for
    y ←sub-problem specific, heuristic improvement of y
    Update reference points z^*, if one of its individuals is larger than one of the corresponding individual of $f(y_i)$
    for $j = 1, \dots, m$ **do**
        if $Z_j \leq f_j(y_i)$ **then** set $z_j = f_j(y_i)$
        end if
    end for
    Update neighboring solutions
    if $g(y|w_i, z^*)$<$g(x_i|w_i, z^*)$ **then** $x_i \leftarrow y$
    end if
    if no vector in EP dominate f(y) **then**
        Add f(y) to EP
    else
        Remove all vectors dominated by f(y) from EP
    end if
    Increment the generation count, gen+ +
end while
return EP