Chapter 2: Main Common Concepts for Metaheuristics

CS-616: Optimization Algorithms

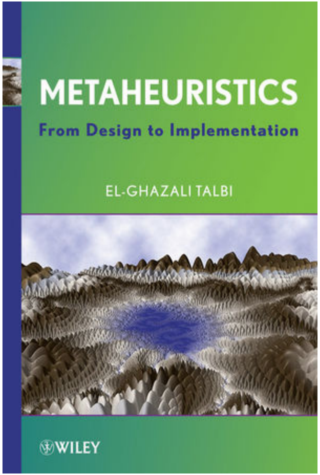
Dr. Mahdi A. Khemakhem & Dr. Esraa M. Eldesouky

Department of Computer Science
College of Computer Engineering and Science
Prince Sattam bin Abdulaziz University

AY: 2025/2026

Materials

- **Textbook:** **"Metaheuristics: From Design to Implementation"** by *El-Ghazali Talbi*
- ➥Read Chapter 1 (Sections 1.4, 1.5, 1.6, and 1.7), Chapter 2 (Section 2.1), and Chapter 3 (Section 3.1) for this chapter's material

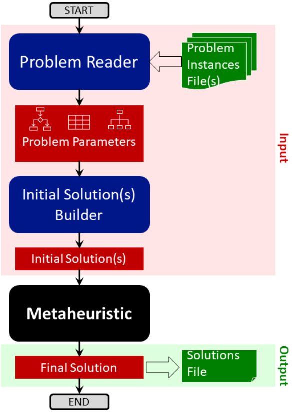


Outline

Outline

General Schema of Metaheuristics

- For now, we consider that a Metaheuristic is a $\boxed{\text{black box!}}$
- Since a Metaheuristic is an algorithm so it should have Input and Output.
- Two modules[a] are responsible to build the Input:
  - Problem reader: a module that reads the problem instance (example) then provides the problem parameters stored in some data structures.
  - Initial solution(s) builder: a module that, from the problem parameters, build an initial solution(s) (feasible solution(s)) considered as an initial starting point to the $\boxed{\text{Metaheuristic}}$.
- Starting from an initial solution(s), the $\boxed{\text{Metaheuristic}}$ provides a final solution (not necessarily optimal) considered as an Output to be stored in a solution file.



[a]functions, methods, procedures, etc.

Initial Solution(s)

- **The majority of Metaheuristics start their searching process from a single or multiple initial solution.**
- **Initial solution(s) should be built from the problem parameters and respecting all problem constraints.**
- **Generally, the quality/qualities (objective function value(s)) of initial solution(s) are relatively ignored.**
- **A good Metaheuristic should be insensitive to the quality of the initial solution used as starting point.**
- **Many strategies used in the literature to build initial solution(s) for Metaheuristics.**
  - **The well known named "Greedy Algorithms"**

Greedy Algorithms (1/2)

In greedy or constructive algorithms[1], we start from scratch (empty solution) and construct a solution by assigning values to one decision variable at a time, until a complete solution is generated.

- Given an optimization problem, where:
  - a solution can be defined by the presence/absence of a finite set of elements $E = \{e_1, e_2, \ldots, e_n\}$,
  - the objective function may be defined as $f : 2^E \to \mathbb{R}$,
  - the search space is defined as $F \subset 2^E$.
- A partial solution s may be seen as a subset $\{e_1, e_2, \ldots, e_k\}$ of elements e_i from the set of all elements E. Initially s is empty.
- At each step, a local heuristic is used to select the new element to be included in the set s.
- Once an element e_i is selected to be part of the solution s, it is never replaced by another element.
- There is no backtracking of the already taken decisions.

[1] Also referred to as successive augmentation algorithms.

Greedy Algorithms (2/2)

Algorithm 1 shows the template of a greedy algorithm.

Algorithm 1: Template of a greedy algorithm

Input: Problem Parameters, $s = \{\}$; `/* Initial solution (null) */`

Output: Initial solution s ; `/* Feasible solution (full) */`

repeat

  $e_i = \textbf{Local-Heuristic}(E - \{e/e \in s\})$; `/* next element selected from` E `minus already selected elements */`

  if $s \cup e_i \in F$ **then**

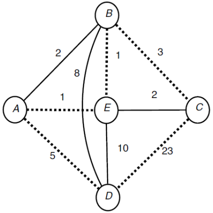
    $s = s \cup e_i$;

  end

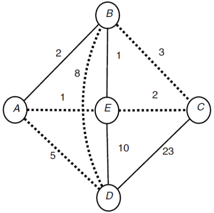
until *Complete solution found*;

Greedy Algorithm for the Traveling Salesman Problem

- In the **Traveling Salesman Problem (TSP)**, the set E is defined by the **set of edges**.
- The set F of **feasible solutions** is defined by the subsets of 2^E that forms **Hamiltonian cycles**.
- A **solution** can be considered as a set of edges.
- A heuristic that can be used to select the next edge may be based on the distance. **One possible greedy heuristic** is to **select the nearest neighbor**.
- Figure at the right illustrates the application of the **nearest-neighbor greedy heuristic** on the graph beginning from the node A.
- Figure at the bottom illustrates the **optimal solution** that can be provided after applying a Metaheuristic on the **initial solution** on the top.



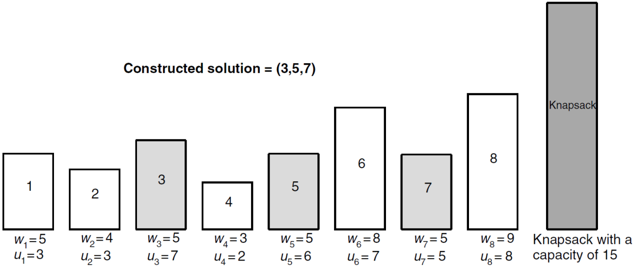
Greedy final solution : A – E – B – C – D – A
with cost = 33



Better solution : A – E – C – B – D – A
with cost = 19

Greedy Algorithm for the Knapsack Problem

- In the Knapsack Problem (KP), the set E is defined by the set of objects to be packed.
- The set F represents all subsets of E that are feasible solutions.
- A greedy algorithm that can be used to solve the KP consists in choosing the object minimizing the ratio $\frac{w_i}{u_i}$ where w_i (resp. u_i) represents the weight (resp. profit) of the object i.
- The figure below illustrates this greedy heuristic for a KP instance.



Outline

Solution Encoding (Representation)

- Designing any iterative $\boxed{\text{Metaheuristic}}$ needs an encoding (representation) of a solution[2].
  - It is a fundamental design question in the development of Metaheuristics.
- The encoding plays a major role in the efficiency and effectiveness of any Metaheuristic and constitutes an essential step in designing a Metaheuristic.
- The encoding must be suitable and relevant to the tackled optimization problem.
- The efficiency of a representation is also related to the search operators applied on this representation (neighborhood, recombination, etc.).
- When defining a representation, we should to bear in mind how the solution will be evaluated and how the search operators will operate.

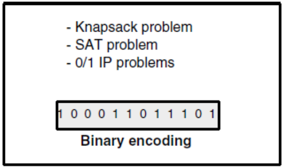
[2] In the evolutionary computation community, the **genotype** defines the representation of a solution. A solution is defined as the **phenotype**.

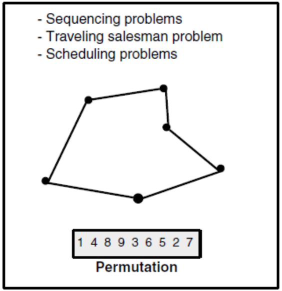
Solution Encoding - Characteristics

Many alternative encoding/representations may exist for a given problem.

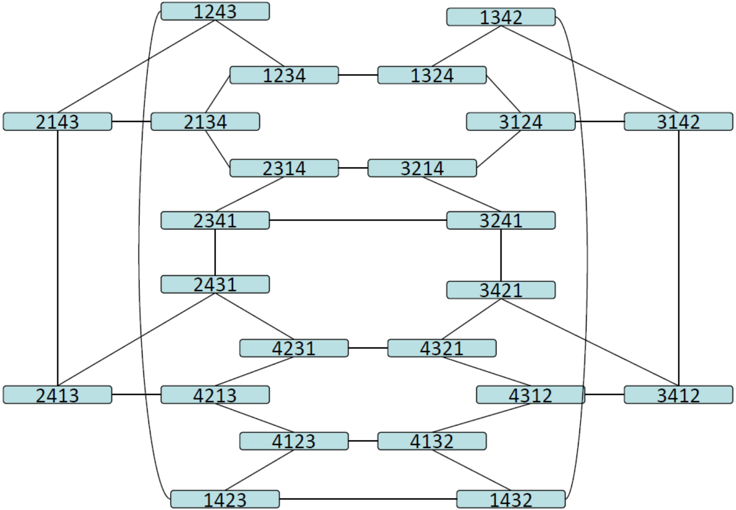
A representation must have the following characteristics:

- **Completeness:** all solutions associated with the problem must be represented.
- **Connexity:** a **search path must exist between any two solutions of the search space**. Any solution of the search space, especially the global optimum solution, can be attained.
- **Efficiency:** **encoding must be easy to manipulate by the search operators**. The time and space complexities of the operators dealing with the representation must be reduced.





Example of TSP Solution Encoding (Representation space)



Outline

Objective Function

- The objective function[3] f formulates the goal to achieve.
  - It associates, with each solution $s \in S$ of the search space S, a value that describes the quality or the fitness of the solution, $f : S \to \mathbb{R}$.
  - Then, it represents an absolute value and allows a complete ordering of all solutions of the search space.
  - From the representation space of the solutions R, some decoding functions d may be applied, $d : R \to S$, to generate a solution that can be evaluated by the function f.
- The objective function is an important element in designing a Metaheuristic. It will guide the search toward "good" solutions of the search space. If the objective function is improperly defined, it can lead to non-acceptable solutions whatever Metaheuristic is used.

[3] Also defined as the **cost function**, **evaluation function**, and **utility function**.

Outline

Constraint Handling

- Dealing with constraints in optimization problems is another important topic for the efficient design of Metaheuristics. Many continuous and discrete optimization problems are constrained, and it is not trivial to deal with those constraints.
- The constraints may be of any kind: linear or nonlinear and equality or inequality.
- Constraint handling strategies, which mainly act on the representation of solutions or the objective function, can be classified as:
  - Reject strategies
  - Penalizing strategies
  - Repairing strategies
  - Preserving strategies

Reject Strategies

- **Reject strategies[4] represent a simple approach, where only feasible solutions are kept during the search and then infeasible solutions are automatically discarded.**
- **This kind of strategies are conceivable if the portion of infeasible solutions of the search space is very small.**
- **Reject strategies do not exploit infeasible solutions. Indeed, it would be interesting to use some information on infeasible solutions to guide the search toward global optimum solutions that are in general on the boundary between feasible and infeasible solutions.**

[4] Also named **"death penalty"**

Penalizing Strategies

- In penalizing strategies, infeasible solutions are considered during the search process.
- The objective function is extended by a penalty function that will penalize infeasible solutions.
- This is the most popular approach.
- Many alternatives may be used to define the penalties.
  - For instance, the objective function f may be penalized in a linear manner: $f'(s) = f(s) + \lambda c(s)$ where $c(s)$ represents the cost of the constraint violation and λ the aggregation weights.
  - The search enables sequences of the type (s_t, s_{t+1}, s_{t+2}) where s_t and s_{t+2} represent feasible solutions, s_{t+1} is an infeasible solution, and s_{t+2} is better than s_t.

Repairing Strategies

- Repairing strategies consist in heuristic algorithms transforming an infeasible solution into a feasible one.
- A repairing procedure is applied to infeasible solutions to generate feasible ones.
- For instance, those strategies are applied in the case where the search operators used by the optimization algorithms may generate infeasible solutions.

Preserving Strategies

- In preserving strategies for constraint handling, a specific representation and operators will ensure the generation of feasible solutions.
- They incorporate problem-specific knowledge into the representation and search operators to generate only feasible solutions and then preserve the feasibility of solutions.

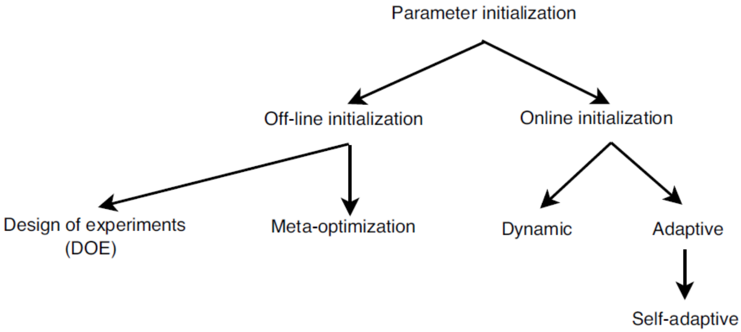
Outline

Parameter Tuning

- Many parameters have to be tuned for any $\boxed{\text{Metaheuristic}}$.
- Parameter tuning may allow a larger flexibility and robustness, but requires a careful initialization.
- Those parameters may have a great influence on the efficiency and effectiveness of the search.
- It is not obvious to define a priori which parameter setting should be used.
- The optimal values for the parameters depend mainly on the problem and even the instance to deal with and on the search time that the user wants to spend in solving the problem.
- A universally optimal parameter values set for a given Metaheuristic does not exist.

Parameter Tuning Strategies

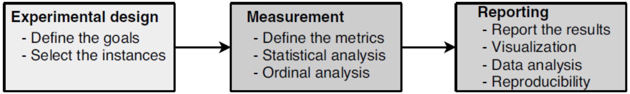
- There are two different strategies for parameter tuning:
  - the $\boxed{\text{off-line}}$[5] parameter initialization (or meta-optimization): the values of different parameters are fixed before the execution of the Metaheuristic.
  - the $\boxed{\text{online}}$[6] parameter tuning strategy:the parameters are controlled and updated dynamically or adaptively during the execution of the Metaheuristic.



Outline

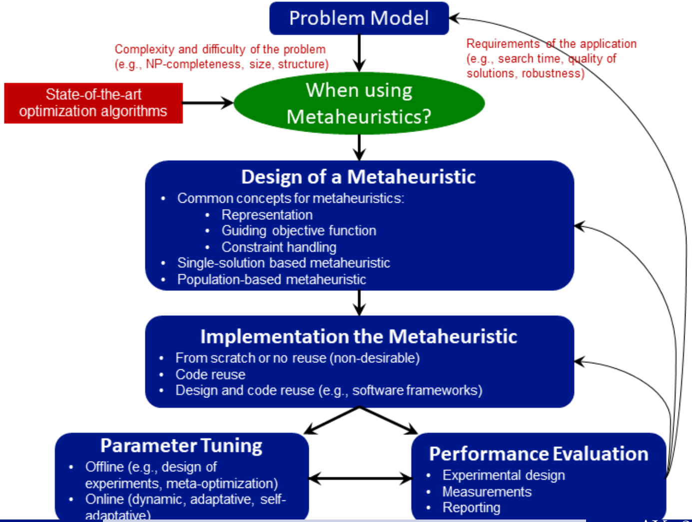
Performance Analysis (3 steps)

- Performance analysis of Metaheuristics is a necessary task to perform and must be done on a fair basis.
- A theoretical approach is generally not sufficient to evaluate a Metaheuristic.
- To evaluate the performance of a Metaheuristic in a rigorous manner, the following three steps must be considered:
  - Experimental design: in the first step, the goals of the experiments, the selected instances, and factors should be defined.
  - Measurement: in the second step, the measures to compute are selected. After executing the different experiments, statistical analysis should be applied to the obtained results.
  - Reporting: finally, the results should be presented in a comprehensive way, and an analysis should be carried out following the defined goals.



Outline

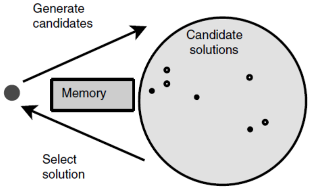
Guidelines for Solving Optimization Problem



Outline

General Schema of S-Metaheuristics (1/3)

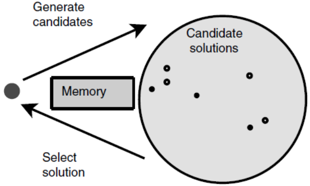
- Single-Solution Based Metaheuristics (S-Metaheuristics) iteratively apply the generation and replacement procedures from the current single solution.
- In the generation phase, a set $C(s)$ of candidate solutions are generated from the current solution s. This set $C(s)$ is generally obtained by local transformations of the solution.
- In the replacement phase[a], a selection is performed from the candidate solution set $C(s)$ to replace the current solution; that is, a solution $s' \in C(s)$ is selected to be the new solution.
- This process iterates until a given stopping criteria.



[a] Also named **transition rule**, **pivoting rule**, and **selection strategy**.

General Schema of S-Metaheuristics (2/3)

- The generation and the replacement phases may be memoryless. In this case, the two procedures are based only on the current solution.
- Otherwise, some history of the search stored in a memory can be used in the generation of the candidate list of solutions and the selection of the new solution.
- Popular examples of such S-Metaheuristics are Local Search (LS), Simulated Annealing (SA), and Tabu Search (TS).



General Schema of S-Metaheuristics (3/3)

The following Algorithm illustrates the high-level template (General Schema) of S-Metaheuristics.

Algorithm 2: High-level Template of S-Metaheuristics

Input: Initial solution s_0.
Output: Best Solution found

```
t = 0;
s_t = s_0;
repeat
    Generate(C(s_t)) ; /* Generate candidate solutions (partial or complete neighborhood)
     from s_t                                                                          */
    s_{t+1} = Select(C(s_t)); /* Select a solution from C(s_t) to replace the current solution s_t
     */
    t = t + 1;
until Stopping Criteria Staisfied;
```

The common search concepts for all S-Metaheuristics are the definition of the neighborhood structure and the determination of the initial solution.

Neighborhood Structure

- The definition of the neighborhood is a required common step for the design of any $\boxed{\text{S-Metaheuristic}}$.
- The neighborhood structure plays a crucial role in the performance of an S-Metaheuristic.
- If the neighborhood structure is not adequate to the problem, any S-Metaheuristic will fail to solve the problem.

Definition of Neighborhood (1/2)

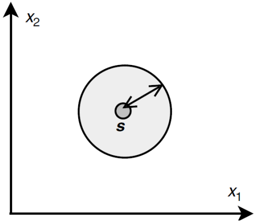
Definition of Neighborhood - A neighborhood function N is a mapping $N : S \to 2^S$ that assigns to each solution $s \in S$ a set of solutions $N(s) \subset S$.

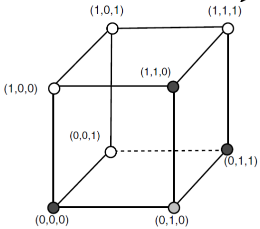
- **A solution s' in the neighborhood of s ($s' \in N(S)$) is called a neighbor of s.**
- **A neighbor is generated by the application of a move operator m that performs a small perturbation to the solution s.**
- **The main property that must characterize a neighborhood is locality. Locality is the effect on the solution when performing the move (perturbation) in the representation.**
  - **Neighborhood have a strong locality when small changes are made in the representation then the solution must reveal small changes. Hence, a S-Metaheuristic will perform a meaningful search in the landscape of the problem.**
  - **Neighborhood have a weak locality when small changes are made in the representation then the solution must reveal large changes. Hence, a S-Metaheuristic will converge toward a random search in the search space.**
- **The structure of the neighborhood depends on the target optimization problem.**

Definition of Neighborhood (2/2)

Definition of Continuous Neighborhood- The neighborhood $N(s)$ of a solution s in a continuous space is the ball with center s and radius equal to ε with $\varepsilon > 0$.

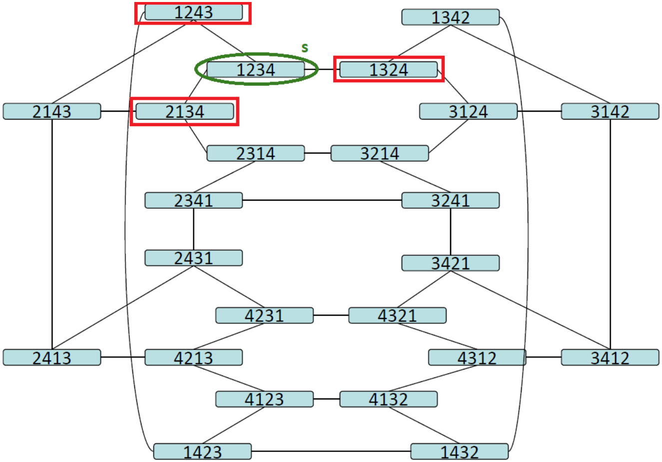
Definition of Discrete Neighborhood- In a discrete optimization problem, the neighborhood $N(s)$ of a solution s is represented by the set $\{s'/d(s', s) \leq \varepsilon\}$, where d represents a given distance that is related to the move operator.





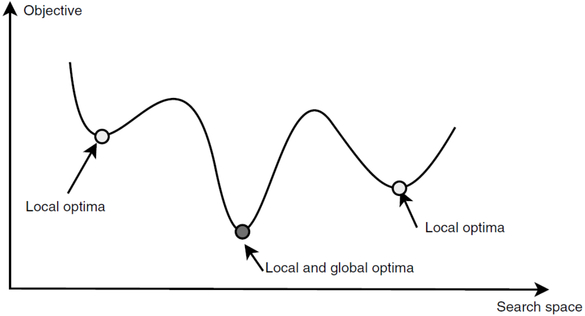
Example of TSP Neighborhood

- An example of neighborhood for The TSP of size 4.
- The move operator consists to make a permutation (switch) between two adjacent vertices.
- For instance, the neighbors of the solution (1, 2, 3, 4) are (2, 1, 3, 4), (1, 3, 2, 4), and (1, 2, 4, 3).



Local Optimum

Definition of Local Optimum- Relatively to a given neighboring function N, a solution $s \in S$ is a local optimum if it has a better quality than all its neighbors; that is, $f(s) \leq f(s') \forall s' \in N(s)$.[7]



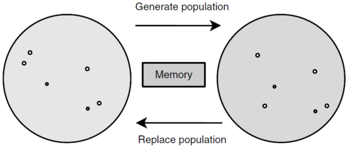
- For the same optimization problem, a local optimum for a neighborhood N_1 may not be a local optimum for a different neighborhood N_2.

[7] for a minimization problem.

Outline

General Schema of P-Metaheuristics (1/3)

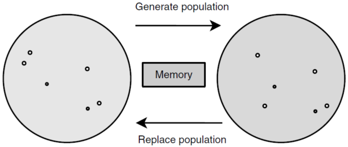
- Population-based Metaheuristics (P-Metaheuristics) start from an initial population of solutions[a]. Then, they iteratively apply the generation of a new population and the replacement of the current population.
- In the generation phase, a new population of solutions is created.
- In the replacement phase, a selection is carried out from the current and the new populations.
- This process iterates until a given stopping criteria.



[a]Some P-Metaheuristics such as **Ant Colony Optimization** start from partial or empty solutions.

General Schema of P-Metaheuristics (2/3)

- The generation and the replacement phases may be memoryless. In this case, the two procedures are based only on the current population.
- Otherwise, some history of the search stored in a memory can be used in the generation of the new population and the replacement of the old population.
- Most of the P-Metaheuristics are nature-inspired algorithms.
- Popular examples of such P-Metaheuristics are Genetic Algorithms (GA), Ant Colony Optimization (ACO), Scatter Search (SS), Particle Swarm Optimization (PSO), Bee Colony (BC), and Artificial Immune Systems (AIS).



General Schema of P-Metaheuristics (3/3)

The following Algorithm illustrates the **high-level template** (General Schema) of **P-Metaheuristics**.

Algorithm 3: High-level template of P-Metaheuristics.

Input: Initial population P_0. ; /* Generation of the initial population */
Output: Best Solution found

$t = 0$;
$P_t = P_0$;
repeat
  Generate(P'_t) ; /* Generation a new population */
  P_{t+1} = **Select-Population**($P_t \cup P'_t$) ; /* Select new population */
  $t = t + 1$
until *Stopping Criteria Staisfied*;

P-Metaheuristics differ in the way they perform the generation and the selection procedures and the search memory they are using during the search.

Initial Population

- Due to the large diversity of initial populations, $\boxed{\text{P-Metaheuristics}}$ are naturally more exploration search algorithms whereas $\boxed{\text{S-Metaheuristics}}$ are more exploitation search algorithms.
- The determination of the initial population is often disregarded in the design of a $\boxed{\text{P-Metaheuristic}}$. However, this step plays a crucial role in the effectiveness of the algorithm and its efficiency. Hence, one should pay more attention to this step.
- In the generation of the initial population, the main criterion to deal with is diversification.
- If the initial population is not well diversified, a premature convergence can occur for any $\boxed{\text{P-Metaheuristic}}$.
  - For instance, this may happen if the initial population is generated using a greedy heuristic or a S-Metaheuristic (e.g., Local Search, Tabu Search) for each solution of the population.

Different Initialization Strategies

Strategies dealing with the initialization of the population may be classified into four categories:

- **Random Generation:** The initial population is generated randomly with respect of the problem constraints.
- **Sequential Diversification:** The initial population is uniformly sampled in the decision space. The solutions are generated in sequence in such a way that the diversity is optimized.
- **Parallel Diversification:** The solutions of a population are generated in a parallel independent way.
- **Heuristic Initialization:** Any heuristic (e.g., local search) can be used to initialize the population. This strategy may be more effective and/or efficient than a random initialization. It is obvious to "randomize" the greedy procedure to obtain different solutions from the greedy procedure. The main drawback of this approach is that the initial population may lose its diversity, which will generate a premature convergence and stagnation of the population.

Analysis of Initialization Strategies

Strategies dealing with the initialization of the population may be analyzed according to $\boxed{\text{diversity}}$, $\boxed{\text{computational cost}}$, **and** $\boxed{\text{quality of the solutions}}$

Strategy	Diversity	Computational Cost	Quality of Initial Solutions
Pseudo-random	++	+++	+
Quasi-random	+++	+++	+
Sequential diversification	++++	++	+
Parallel diversification	++++	+++	+
Heuristic	+	+	+++

Stopping Criteria

Many stopping criteria based on the evolution of a population may be used. Some of them are similar to those designed for S-metaheuristics.

- **Static procedure:** The end of the search may be known a priori. For instance, one can use a fixed number of iterations (generations), a limit on CPU resources, or a maximum number of objective function evaluations.
- **Adaptive procedure:** The end of the search cannot be known a priori. One can use a fixed number of iterations (generations) without improvement, when an optimum or a satisfactory solution is reached (e.g., a given error to the optimum or an approximation to it when a lower bound is known beforehand).

The End