CS616 - Optimization Algorithms

Chapter 3 - Local Search-Based Algorithms

Dr. Mahdi Khemakhem

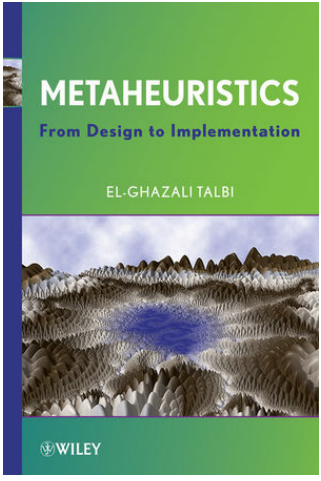
Department of Computer Science
College of Computer Engineering and Science
Prince Sattam bin Abdulaziz University

AY - 2025/2026

Outline

Materials

- **Textbook:** "Metaheuristics: From Design to Implementation" by *El-Ghazali Talbi*
- $\Rightarrow$ **Read Chapter 2 (Sections 2.3, 2.6, 2.7, and 2.8) for this chapter's material**

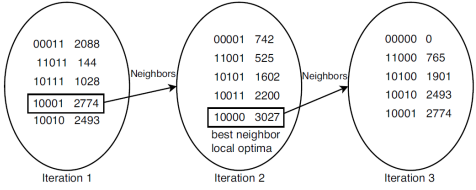


Outline

Local Search - Principle

Local Search (LS)[1] is likely the **oldest** and **simplest S-Metaheuristic** method. It is based on the following principles:

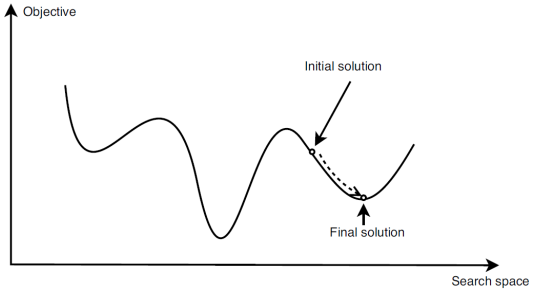
- It **starts** at a given **initial solution**.
- **At each iteration**, the heuristic **replaces** the **current solution** by a **neighbor** that **improves the objective function**.
- The search **stops** when **all candidate neighbors** are **worse than** the **current solution**, meaning **a local optimum** is **reached**.



Local search process using a **binary representation of solutions**, a **flip move operator**, and the **best neighbor selection strategy**. The objective function is Max $Z = x^3 - 60x^2 + 900x$. The **global optimal solution** is $f(01010) = f(10) = 4000$, while the **final local optima found** is $s = (10000)$, **starting from the solution** $s_0 = (10001)$.

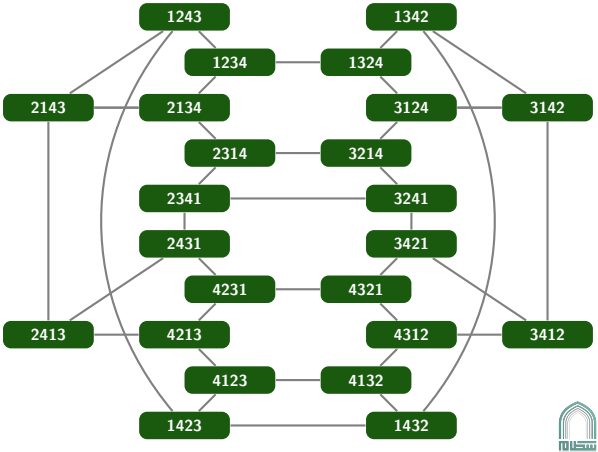
Local Search - Behavior

- For **large neighborhoods**, the **candidate solutions may be a subset of the neighborhood**.
  - The **main objective** of this **restricted neighborhood strategy** is to **speed up the search**.
- **Variants of LS** may be distinguished **according** to the **order in which the neighboring solutions are generated** (*deterministic/stochastic*) and the **selection strategy** (*selection of the neighboring solution*).



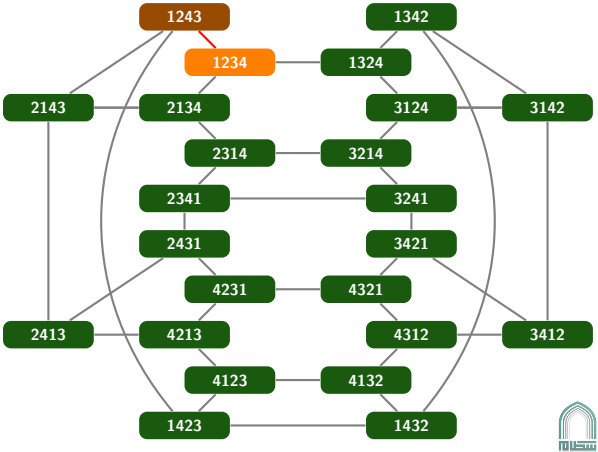
Local Search - General Schema (1/3)

- **Local Search** may be seen as a **descent walk** in the **graph representing the search space**.
- This **graph** may be defined by $G = (S, V)$, where S **represents the set of all feasible solutions of the search space** and V **represents the neighborhood relation**.



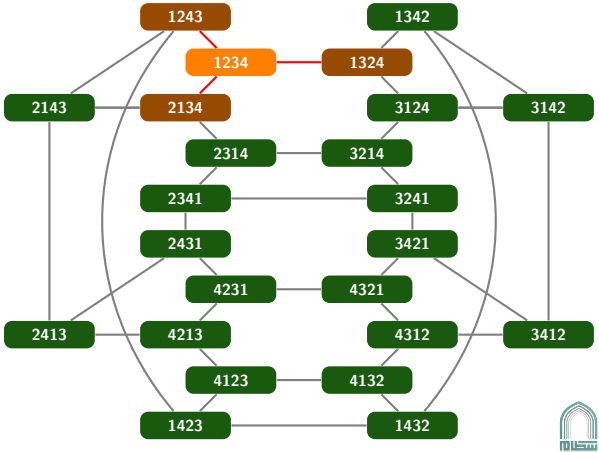
Local Search - General Schema (1/3)

- **Local Search** may be seen as a **descent walk** in the **graph representing the search space**.
- This **graph** may be defined by $G = (S, V)$, where S **represents the set of all feasible solutions of the search space** and V **represents the neighborhood relation**.
- In the graph G, an *edge* (i, j) will connect to any **neighboring solutions** s_i and s_j.



Local Search - General Schema (1/3)

- **Local Search** may be seen as a **descent walk** in the **graph representing the search space**.
- This **graph** may be defined by $G = (S, V)$, where S **represents the set of all feasible solutions of the search space** and V **represents the neighborhood relation**.
- In the graph G, an *edge* (i, j) will connect to any **neighboring solutions** s_i and s_j.
- For a **given solution** s, the number of *associated edges* will be $|N(s)|$ (number of **neighbors**).
- For example: solution s **(1,2,3,4)** is **connected** to all its **neighbors** $s' \in N(s) = \{$ **(2,1,3,4), (1,3,2,4),(1,2,4,3)** $\}$ by *edges*.



Local Search - General Schema (2/3)

Algorithm 1 shows the general schema of a Local Search Metaheuristic.

Algorithm 1: Template of a Local Search Algorithm

Input: Initial solution s_0; /* Initial solution generated by a Greedy Algorithm */
Output: Final solution found s; /* Local optima */
$s = s_0$; /* s is the current solution that represents always the best-known solution */
while not termination_Criterion **do**
  $N(s) =$ **Generate_Neighbors(**s**)** ; /* Generation of candidate neighbors */
  $s' =$ **Select_Best(**$N(s)$**)** ; /* Select the best neighbor in $N(s)$ */
  if $f(s')$ is **not better** that $f(s)$ **then** ; /* Stop the search if a local optima is found */
    Stop;
  end
  $s = s'$; /* Update the current solution by the better neighbor */
end

Local Search - General Schema (3/3)

Local Search Behavior

From an **initial solution** s_0, the algorithm will **generate a sequence** $s_1, s_2, \ldots, s_k$ of solutions **with the following characteristics:**

- The size of the sequence k is unknown a priori.
- $s_{i+1} \in N(s_i), \forall i \in [0, k-1]$.
- $f(s_{i+1}) < f(s_i), \forall i \in [0, k-1]$.[a]
- s_k is a local optimum: $f(s_k) \leq f(s), \forall s \in N(s_k)$.

[a] *The problem is supposed to be a minimization problem.*

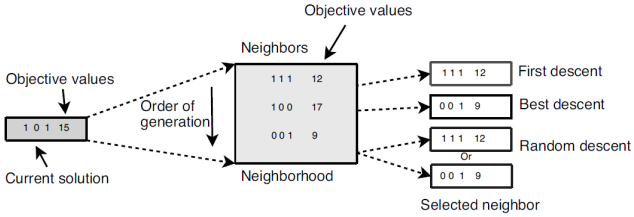
Selection Strategy of the Neighbor

In addition to the definition of the **initial solution** and the **neighborhood structure**, designing a local search algorithm has to **address** the *selection strategy of the neighbor* that will determine the next current solution.

Selection of the Neighbor (1/3)

Many strategies can be applied in the **selection** of a *better neighbor*:

1. **Best improvement (steepest descent):** *the best neighbor is selected.*
   - The **neighborhood** is evaluated in a **fully deterministic** manner.
   - The **exploration of the neighborhood** is **exhaustive**, and **all possible moves are tried** for a solution to select the **best neighboring solution**.
   - This type of exploration may be **time-consuming for large neighborhoods**.

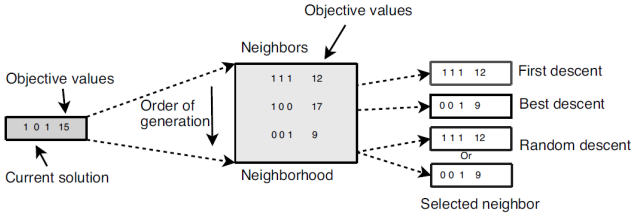


Selection of the Neighbor (2/3)

2 **First improvement:** *chooses the first improving neighbor that is better than the current solution.*

- Involves a **partial evaluation** of **the neighborhood**.
- In a cyclic exploration, the **neighborhood** is **evaluated** in a **deterministic way** following a **given order of generating the neighbors**.
- **In the worst case** (*i.e., when no improvement is found*), **a complete evaluation of the neighborhood is performed**.

3 **Random selection:** *a random selection is applied to those neighbors improving the current solution.*



Selection of the Neighbor (3/3)

Compromise between Quality and Time

A **compromise** in terms of **quality of solutions** and **search time** may consist in using:

- *First improvement strategy* *when the initial solution is randomly generated*,
- *Best improvement strategy* *when the initial solution is generated using a greedy procedure*.

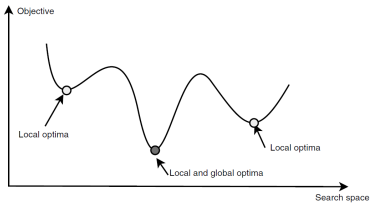
First improvement in practice

In practice, on many applications, it has been observed that the **first improving strategy** **leads to the same quality of solutions** as the **best improving strategy** **while using a smaller computational time**.

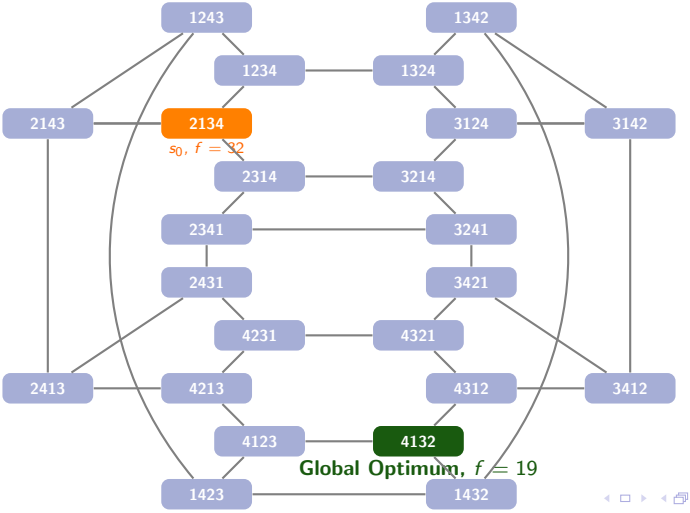
- Moreover, the **probability of premature convergence to a local optima** is **less important** in the **first improvement strategy**.

Local Optima Disadvantage (1/2)

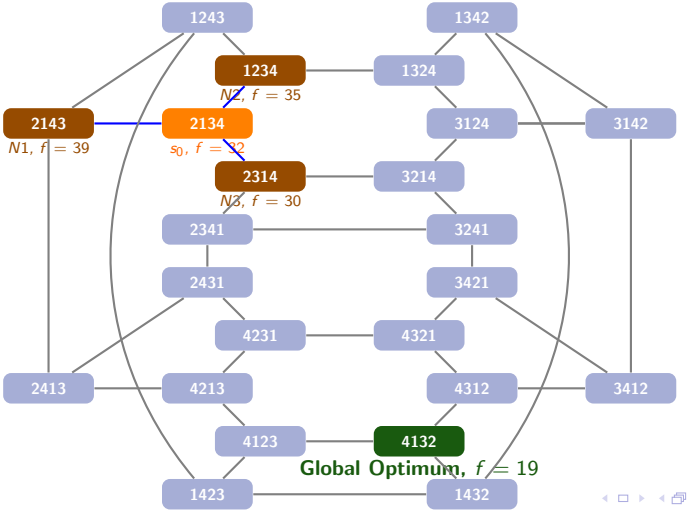
- In general, **local search** is a **very easy method to design and implement and gives fairly good solutions very quickly** $\Rightarrow$*widely used optimization method in practice*.
- One of the **main disadvantages** of LS is that it **converges** toward *local optima*. $\Rightarrow$ *no means to estimate the relative error from the global optimum and the number of iterations performed may not be known in advance*.
- Local search **works well if there are not too many local optima in the search space** or **the quality of the different local optima is more or less similar**.



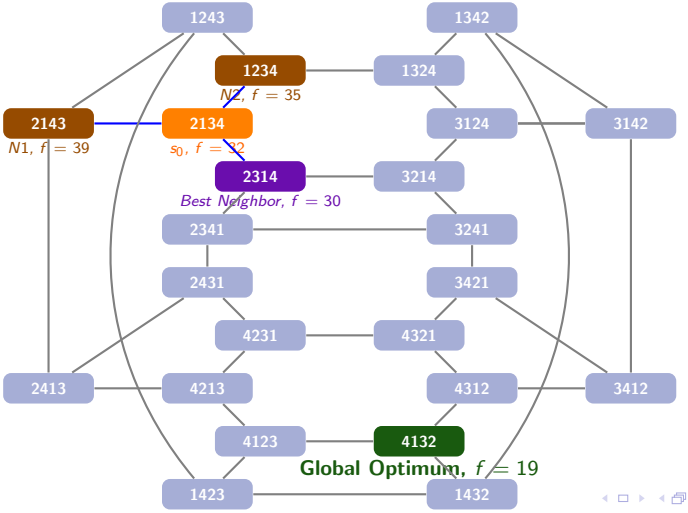
Local Optima Disadvantage (2/2)



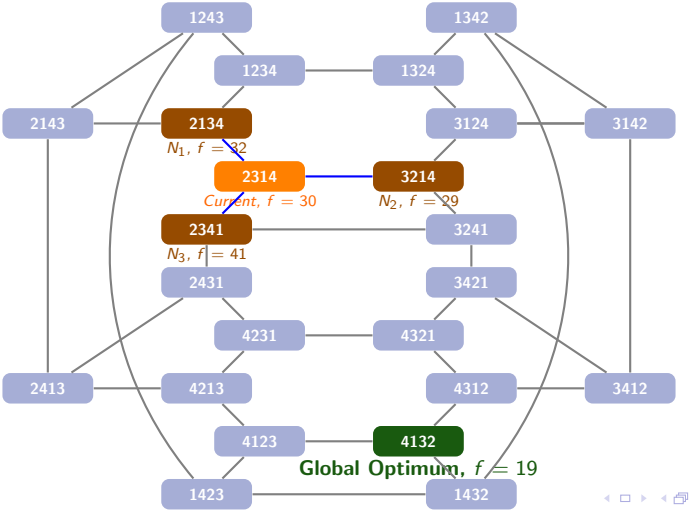
Local Optima Disadvantage (2/2)



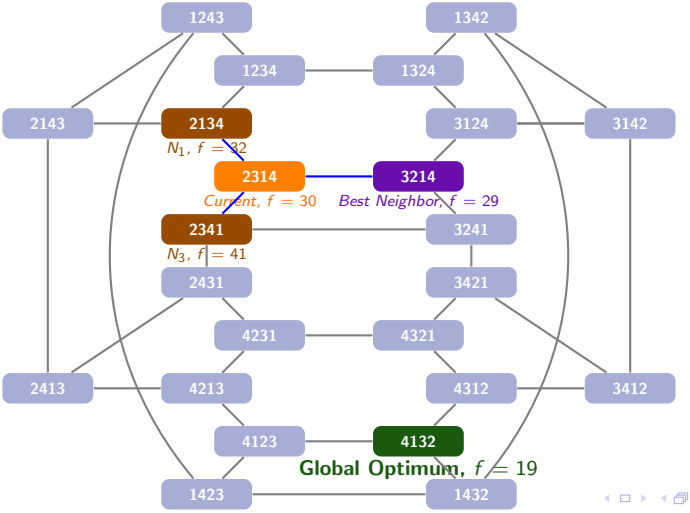
Local Optima Disadvantage (2/2)



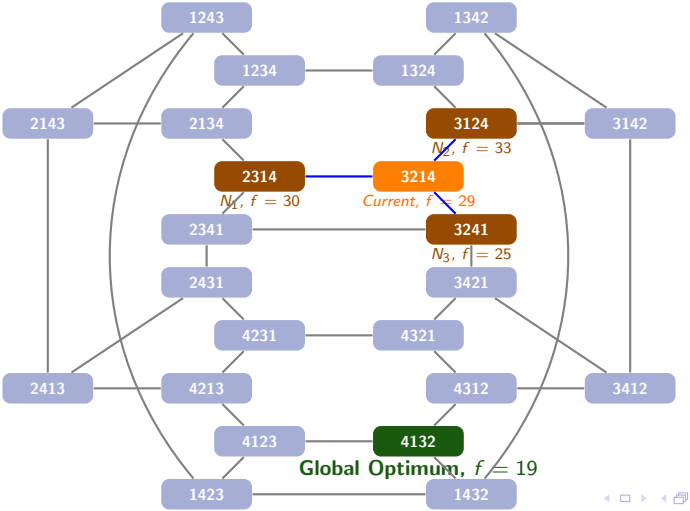
Local Optima Disadvantage (2/2)



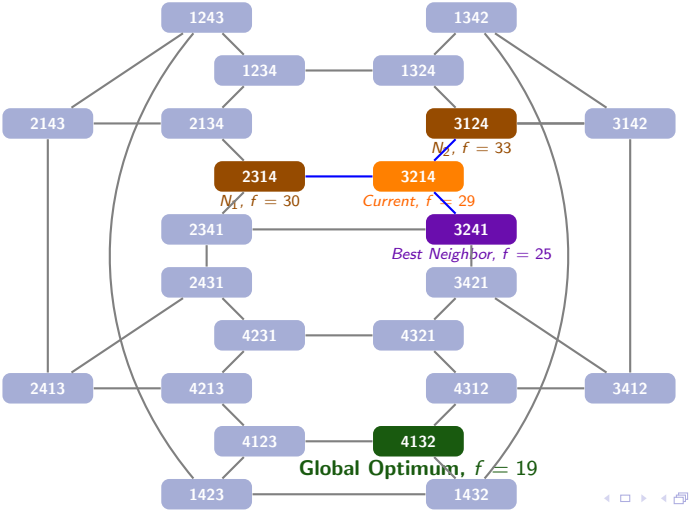
Local Optima Disadvantage (2/2)



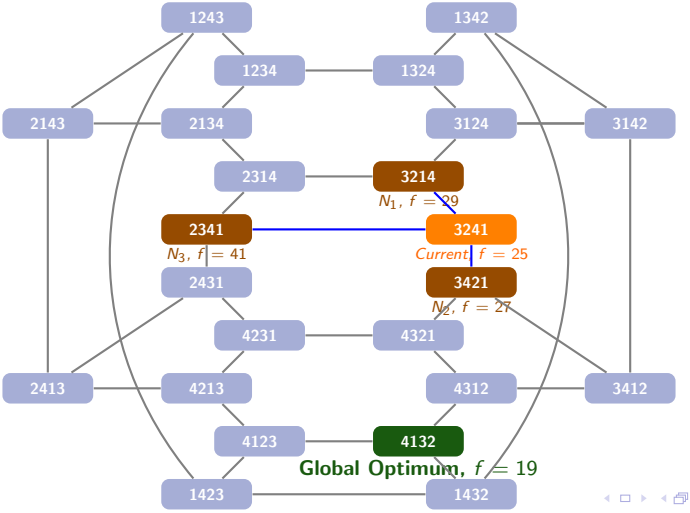
Local Optima Disadvantage (2/2)



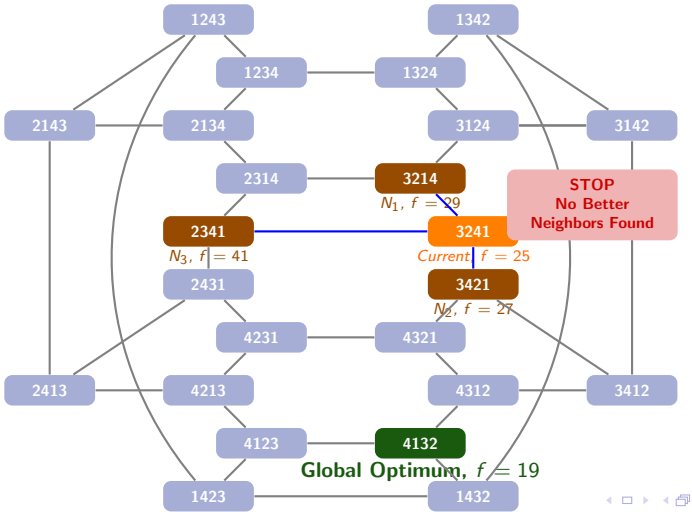
Local Optima Disadvantage (2/2)



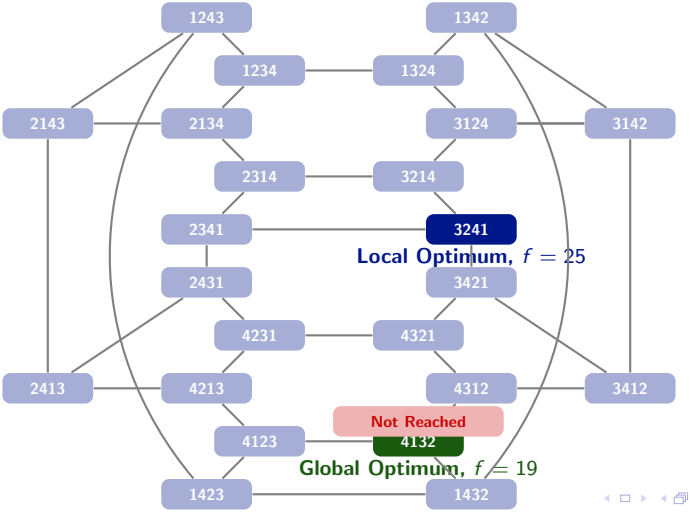
Local Optima Disadvantage (2/2)



Local Optima Disadvantage (2/2)



Local Optima Disadvantage (2/2)

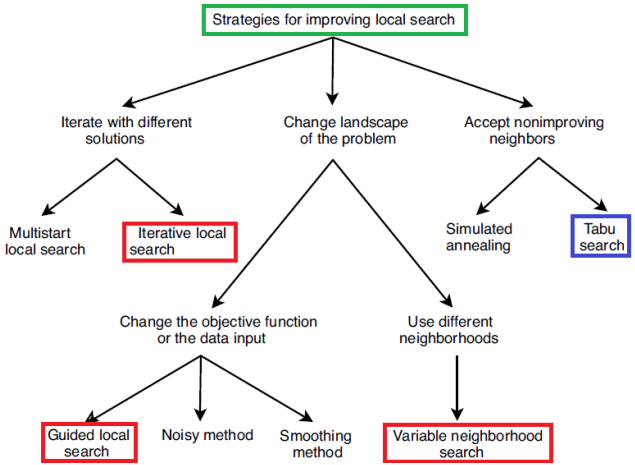


Escaping from Local Optima (1/2)

As the **main disadvantage** of **local search algorithms** is the **convergence toward local optima**, many **alternatives algorithms** have been proposed to avoid becoming stuck at local optima.
Four different families of approaches can be used to avoid local optima:

1. *Iterating from different initial solutions:* applied in **Multi-start Local Search**, **Iterated Local Search** (see Section 2), **GRASP**, and so forth.
2. *Accepting non-improving neighbors:* enable moves that degrade the current solution (it is possible to move out the basin of attraction of a given local optimum). **Simulated Annealing** and **Tabu Search** (see Chapter 3) are popular representative of this class of algorithms.
3. *Changing the neighborhood:* it consists in changing the neighborhood structure during the search. For instance **Variable Neighborhood Search** (see Section 3) uses these strategies.
4. *Changing the objective function or the input data of the problem:* perturb the input-data or the objective function or the constraints of the problem in order to diversify search space. For instance **Guided Local Search** (see Section 4).

Escaping from Local Optima (2/2)



Key Takeaways

Key Takeaways

- **Local Search** is a **simple and effective metaheuristic** for solving **combinatorial optimization problems**.
- The **performance** of a **local search algorithm depends on the neighborhood structure** used.
- The msin **disadvantage** of **local search algorithms** is the **convergence toward local optima**.
- Many **strategies** can be used to **escape from local optima**, such as **Iterated Local Search**, **Tabu Search**, **Variable Neighborhood Search**, and so forth.

Outline

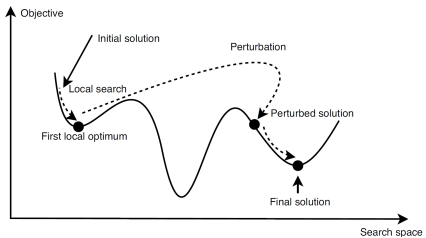
Iterated Local Search - Principle (1/2)

- The quality of the **local optima** obtained by a **Local Search** method **depends on the initial solution**.
- As we **can generate local optima** with **high variability**, *Iterated Local Search (ILS)*[2] may be used **to improve the quality of successive local optima**.
- In **multi-start Local Search**, the **initial solution** is always **chosen randomly** and then is **unrelated to the generated local optima**.
- **ILS improves** the **classical multi-start Local Search** by **perturbing the local optima** and **reconsidering them as initial solutions**.

Iterated Local Search - Principle (2/2)

ILS is based on a simple principle described as follows:

- First, a **Local Search** is **applied** to an **initial solution**.
- Then, **at each iteration**, a *perturbation* **of the obtained local optima is carried out**.
- Finally, a **Local Search is applied to the perturbed solution**.
- The *generated solution* **is accepted as the new current solution under some conditions**.
- This process **iterates until a given stopping criterion**.



Iterated Local Search - General Schema

Algorithm 2 shows the general schema of a Iterated Local Search Metaheuristic.

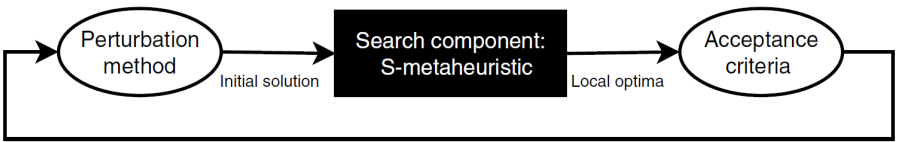
Algorithm 2: Template of the Iterated Local Search Algorithm

Input: Initial solution s_0; `/* Initial solution generated by a Greedy Algorithm */`
Output: Best solution found s^*; `/* Local optima */`
1 $s^* =$ **Local_Search**(s_0) ; `/* Apply a given local search algorithm */`
2 **repeat**
3 $\quad s' =$ **Perturb**(s^*, *search history*) ; `/* Perturb the obtained local optima */`
4 $\quad s'_* =$ **Local_Search**(s') ; `/* Apply local search on the perturbed solution */`
5 $\quad$ **Accept**(s^*, s'_*, *search memory*) ; `/* Accepting criteria */`
6 **until** Stopping criteria satisfied;

Iterated Local Search - Basic Elements

Three basic elements compose an Iterated Local Search:

- *Local_Search:* any **S-metaheuristic** (deterministic or stochastic) **can be used in the ILS framework** such as a simple *Descent Algorithm*, *Tabu Search*, or *Simulated Annealing*. **The search procedure is treated as a** black box.
- *Perturb:* **perturbation operator** may be seen as a **large random move of the current solution**. *It should keep some part of the solution and perturb strongly another part of the solution.*
- *Accept:* **acceptance criterion** defines the **conditions the new local optima must satisfy to replace the current solution**.



Perturbation Method (1/2)

- The first **motivation** of the **ILS algorithm** is based on the fact that the **perturbation method** must be **more effective than** a **random restart approach**.
- The **length of a perturbation** may be related to the **neighborhood associated with the encoding** or with the **number of modified components**.
  - **Too small perturbation** *may generate cycles in the search and no gain is obtained.*
  - **Too large perturbation** *will erase the information about the search memory, and then the good properties of the local optima are skipped.*
  - **The optimal length** *depends mainly on the landscape structure of the optimization problem.*
- The **move operator** used in the **perturbation may be of different nature from the neighborhood relation used in the local search procedure**.

Perturbation Method (2/2)

Many **biased perturbation methods** can be designed according to the following criteria:

- *Fixed or variable perturbations:* The **length of the perturbations applied to a local optima** may be defined as follows:
  - **Static:** *The length is fixed a priori before the beginning of the ILS search.*
  - **Dynamic:** *The length of the perturbation is defined dynamically without taking into account the search memory.*
  - **Adaptive:** *The length of the perturbation is adapted during the search according to some information about the search memory*. Indeed, the *optimal length will depend on the input instance and its structure*. More information about the characteristics of the landscape may be extracted during the search.
- *Random or semi-deterministic perturbation:* The **perturbation** carried out on a solution may be a **random one** (memoryless) in which each move is generated randomly in the neighborhood. In **semi-deterministic** perturbations, the move is biased according to the memory of the search.

Acceptance Criteria

- The role of the **acceptance criterion** combined with the **perturbation method** is to **control** the classical trade-off between the **intensification** and the **diversification** tasks.
  - The **extreme solution in terms of intensification** is to **accept only improving solutions in terms of the objective function** (*strong selection*).
  - The **extreme solution in terms of diversification** is to **accept any solution without any regard to its quality** (*weak selection*).
- Many **acceptance criteria** that **balance the two goals** may be applied namely:
  - **Probabilistic acceptance criteria**
  - **Deterministic acceptance criteria**

Key Takeaways

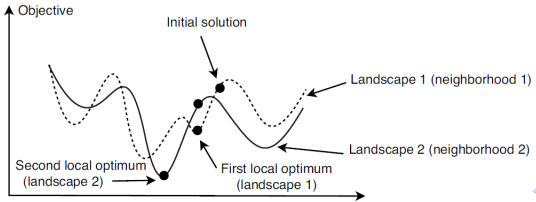
Key Takeaways

- **Iterated Local Search (ILS)** is a **metaheuristic** that **iterates calls to a local search algorithm** from **perturbed local optima**.
- The **perturbation method** is a **key element** of the **ILS algorithm** that **enables escaping from local optima**.
- The **acceptance criterion** is another **key element** of the **ILS algorithm** that **balances the intensification and diversification tasks**.

Outline

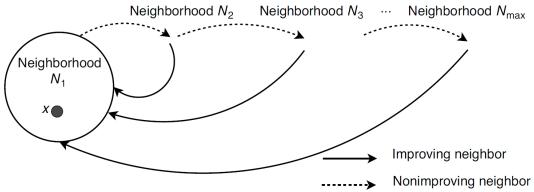
Variable Neighborhood Search - Principle

- The basic idea of the *Variable Neighborhood Search (VNS)* is to **successively explore a set of predefined neighborhoods** to **provide a better solution**.
- **VNS explores a set of neighborhoods** to **get different local optima** to militarybfescape from local optima.
- **VNS exploits** the fact that **using various neighborhoods in local search** may **generate different local optima** and that **the global optima is a local optima for a given neighborhood structure**.
  - *Indeed, different neighborhoods generate different landscapes.*
- The **VNS algorithm** is based on the *Variable Neighborhood Descent (VND)*, which is a **deterministic version of VNS**.



Variable Neighborhood Descent

- **VND uses successive neighborhoods** in descent to a **local optimum**.
  - First, one has to define a **set of neighborhood structures** $N_\ell(\ell = 1, \ldots, \ell_{max})$.
  - Let N_1 be the **first neighborhood** to use and x the **initial solution**.
    - *If an* **improvement of the solution** x *in its* **current neighborhood** $N_\ell(x)$ *is* **not possible**, *the* **neighborhood structure is changed** *from* N_ℓ *to* $N_{\ell+1}$.
    - *Otherwise, the* **neighborhood structure returns to the first one** $N_1(x)$ *to restart the search.*
- This **strategy will be effective** if the **different neighborhoods** used are **complementary** in the sense that a local optima for a neighborhood N_i will not be a local optima in the neighborhood N_j.



Variable Neighborhood Descent - General Schema

Algorithm 3 shows the general schema of the VND algorithm.

Algorithm 3: Template of the Variable Neighborhood Descent Algorithm

Input: A set of neighborhood structures N_ℓ for $\ell = 1, \ldots, \ell_{max}$.
Output: Best solution found

```
1  x = x_0 ; /* Generate the initial solution using Greedy Algorithm */
2  ℓ = 1;
3  while ℓ ≤ ℓ_max do
4  |   Find the best neighbor x' of x in N_ℓ(x);
5  |   if f(x') < f(x) then
6  |   |   x = x';
7  |   |   ℓ = 1;
8  |   else
9  |   |   ℓ = ℓ + 1;
10 |   end
11 end
```

Variable Neighborhood Descent - Features

- The **design** of the **VND algorithm** is **mainly related** to the **selection of neighborhoods** and **the order of their application**.
- The **complexity of the neighborhoods** in terms of their **exploration** and **evaluation must be taken into account**.
- The **larger are the neighborhoods**, the **more time consuming is the VND algorithm**.
- Concerning the **application order**, the **most popular strategy is to rank the neighborhoods following the increasing order of their complexity** (e.g., the size of the neighborhoods $|Nl(x)|$).

General Variable Neighborhood Search

VNS is a stochastic algorithm in which:

- First, a set of **neighborhood structures** $N_k (k = 1, \ldots, n)$ is defined.
- Then, each iteration of the algorithm is composed of **three steps**: *shaking*, *local search*, and *move*:
  1. At each iteration, an initial solution is **shaked** from the current neighborhood N_k. *For instance, a solution x' is selected/generated randomly in the current neighborhood $N_k(x)$.*
  2. A **local search** procedure is **applied to the solution x' to generate the solution x''.**
  3. The **move** step consists to:
     - **replace** *the* **current solution** *is* **replaced** *by the* **new local optima** x'' *if and only if a better solution has been found (i.e., $f(x'') < f(x)$).* The same search procedure is thus restarted from the solution x'' in the first neighborhood N_1.
     - **move** *to the* **next neighborhood** N_{k+1}, *randomly* **generates a new solution in this neighborhood**, *and* **attempts to improve it if no better solution is found** *(i.e., $f(x'') \geq f(x)$).*

Basic Variable Neighborhood Search - General Schema

Algorithm 4 presents the template of the Basic VNS algorithm

Algorithm 4: Template of the Basic Variable Neighborhood Search Algorithm.

Input: A set of neighborhood structures N_k for $k = 1, \ldots, k_{max}$ for **Shaking**.
Output: Best solution found

```
1  x = x_0 ; /* Generate the initial solution using Greedy Algorithm */
2  repeat
3      k = 1;
4      repeat
5          Shaking: pick a random solution x' from the k^th neighborhood N_k(x) of x;
6          x'' = Local_Search(x');
7          if f(x'') < f(x) then
8              x = x'';
9              Continue to search with N_1, k = 1;
10         else
11             k = k + 1;
12         end
13     until k = k_max;
14 until Stopping criteria statisfied;
```

General Variable Neighborhood Search - General Schema

A more General VNS algorithm (simple LS replaced by VND) can be found in Algorithm 5.

Algorithm 5: Template of the General Variable Neighborhood Search Algorithm.

Input: A set of neighborhood structures N_k for $k = 1, \ldots, k_{max}$ for **shaking**, A set of neighborhood structures N_ℓ for $\ell = 1, \ldots, \ell_{max}$ for **Local_Search**.
Output: Best solution found

$x = x_0$; /* Generate the initial solution using Greedy Algorithm */
repeat
  for $k = 1$ **to** k_{max} **do**
    Shaking: pick a random solution x' from the k^{th} neighborhood $N_k(x)$ of x;
    for $\ell = 1$ **to** ℓ_{max} **do** ; /* Local search by VND */

      Find the best neighbor x'' of x' in $N_\ell(x')$;
      if $f(x'') < f(x')$ **then**
        $x' = x''$;
        Continue to search with N_1, $\ell = 1$;
      else
        $\ell = \ell + 1$;
    if local optimum is better than x, $f(x'') < f(x)$ **then** ; /* Move or not */

      $x = x''$;
      Continue to search with N_1, $k = 1$;
    else
      $k = k + 1$;
until Stopping criteria satisfied;

Variable Neighborhood Search - Features

- In addition to the design of a simple **LS** or a **VND** algorithm, the **design of the VNS algorithm** is **mainly related to** the **selection of neighborhoods** for the **shaking phase**.
- Usually, **nested neighborhoods** are used, where **each neighborhood** $N_k(x)$ contains the previous one $N_{k-1}(x)$: $N_1(x) \subset N_2(x) \subset \cdots \subset N_k(x), \forall x \in S$.
- A **compromise** must be found between **intensification** of the search and its **diversification** through the **distribution of work between the local search phase and the shaking phase**.
  - *An* **increased work in the local search phase will generate better local optima (more intensification)**, *whereas an* **increased work in the shaking phase will lead to potentially better regions of the search space (more diversification)**.
- As in **LS**, **VNS requires** a **small number of parameters**.
  - *For the* **shaking phase**, *the* **single parameter** *is the* **number of neighborhood structures** k_{max}. *If* **large values** *of* k_{max} *are used (i.e., very large neighborhoods are considered),* **VNS will be similar to a multi-start LS**. *For* **small values** *of* k_{max}, **VNS will degenerate to a simple LS algorithm**.

Key Takeaways

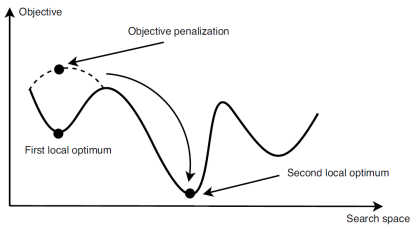
Key Takeaways

- **Variable Neighborhood Search (VNS)** is a **metaheuristic** that **uses multiple neighborhoods** to **escape from local optima**.
- The **Variable Neighborhood Descent (VND)** is a **deterministic version of VNS** that **uses multiple neighborhoods in descent**.
- The **General VNS algorithm** combines the **shaking phase** and a **local search phase** (simple LS or VND).

Outline

Guided Local Search - Principle (1/2)

- *Guided Local Search (GLS)* is a deterministic **S-Metaheuristic** that has been mainly applied to combinatorial optimization problems.
- The **basic principle** of **GLS** is the **dynamic changing of the objective function according to the already generated local optima**.
- The **features of the obtained local optima** are **used to transform the objective function**.
- It **allows the modification of the landscape structure to be explored by an S-metaheuristic to escape from the obtained local optima**.



Guided Local Search - Principle (2/2)

- In **GLS**, a set of m features $ft_i (i = 1, \ldots, m)$ of a solution are defined.
- A **solution feature defines a given characteristic of a solution regarding the optimization problem to solve**. A cost c_i is associated to each feature i.
- When trapped by a **local optima**, the algorithm will **penalize solutions** according to some selected features.
  - To each feature i is associated a penalty p_i **that represents the importance of the feature**. The **objective function f associated with a solution s is then penalized as follows**:

$$f'(s) = f(s) + \lambda \sum_{i=1}^{m} p_i \times I_i(s)$$

where λ **represents the weights associated with different penalties** and, $I_i(s)$ an **indicator function that determines whether the feature ft_i is present in the solution s**:

$$I_i(s) = \begin{cases} 1 & \text{if the feature } ft_i \in s \\ 0 & \text{otherwise.} \end{cases}$$

All penalties p_i are initialized to 0.

Guided Local Search - General Schema

Algorithm 6 describes the template of the GLS algorithm.

Algorithm 6: Template of the Guided Local Search algorithm.

Input: S-Metaheuristic **LS**, λ, **Features** $ft_i(i = 1, \ldots, m)$, **Costs** $c_i(i = 1, \ldots, m)$.
Output: Best solution found

1 $s = s_0$; /* Generate the initial solution using Greedy Algorithm */
2 $p_i = 0, \forall i \in \{1, 2, \ldots, m\}$; /* Penalties initialization */
3 **repeat**
4 **Apply** a S-metaheuristic **LS** ; /* Let $s*$ the final solution obtained */
5 **for** each feature i of $s*$ **do**
6 $u_i = \frac{c_i}{1+p_i}$; /* Compute its utility */
7 **end**
8 $j = \max_{i=1,\ldots,m}(u_i)$; /* Compute the maximum utilities */
9 $p_j = p_{j+1}$; /* Change the objective function by penalizing the feature j */
10 **until** Stopping criteria satisified;

Suitable Features Identification

The **application of GLS to a given optimization problem requires the identification of suitable features of the solutions**.
The **choice of features depends mainly on the model associated with the optimization problem**.
For example:

- **Routing problems:** In routing problems such as the Traveling Salesman Problem and the Vehicle Routing Problem, a feature may be associated with the presence of an edge (a, b) in the solution. The cost corresponds to the distance (or travel time) associated with the edge.
- **Assignment problems:** In assignment problems, such as the generalized assignment problem and location facility problems, a solution represents an assignment of a given number of objects to a set of locations. A feature may be represented by the pair (i, k), where i represents an object and k represents a specific location.

Key Takeaways

Key Takeaways

- **Guided Local Search (GLS)** is a **metaheuristic** that **dynamically modifies the objective function** to **escape from local optima**.
- The **features of the obtained local optima** are **used to modify the objective function**.
- The **application of GLS to a given optimization problem requires the identification of suitable features of the solutions**.

End of Chapter 3