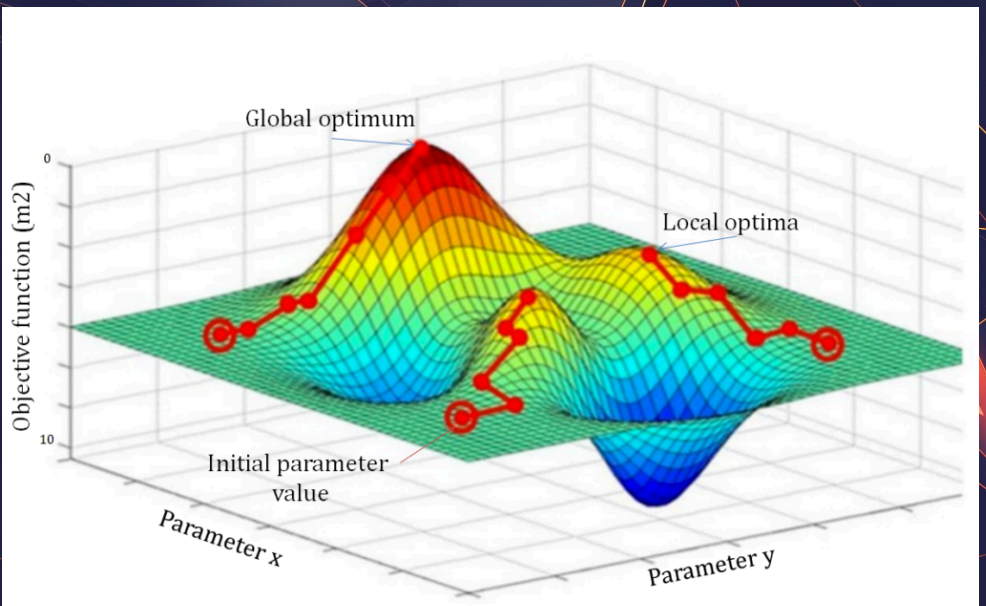



# SINGLE SOLUTION METAHEURISTICS

## ESCAPING FROM LOCAL OPTIMA

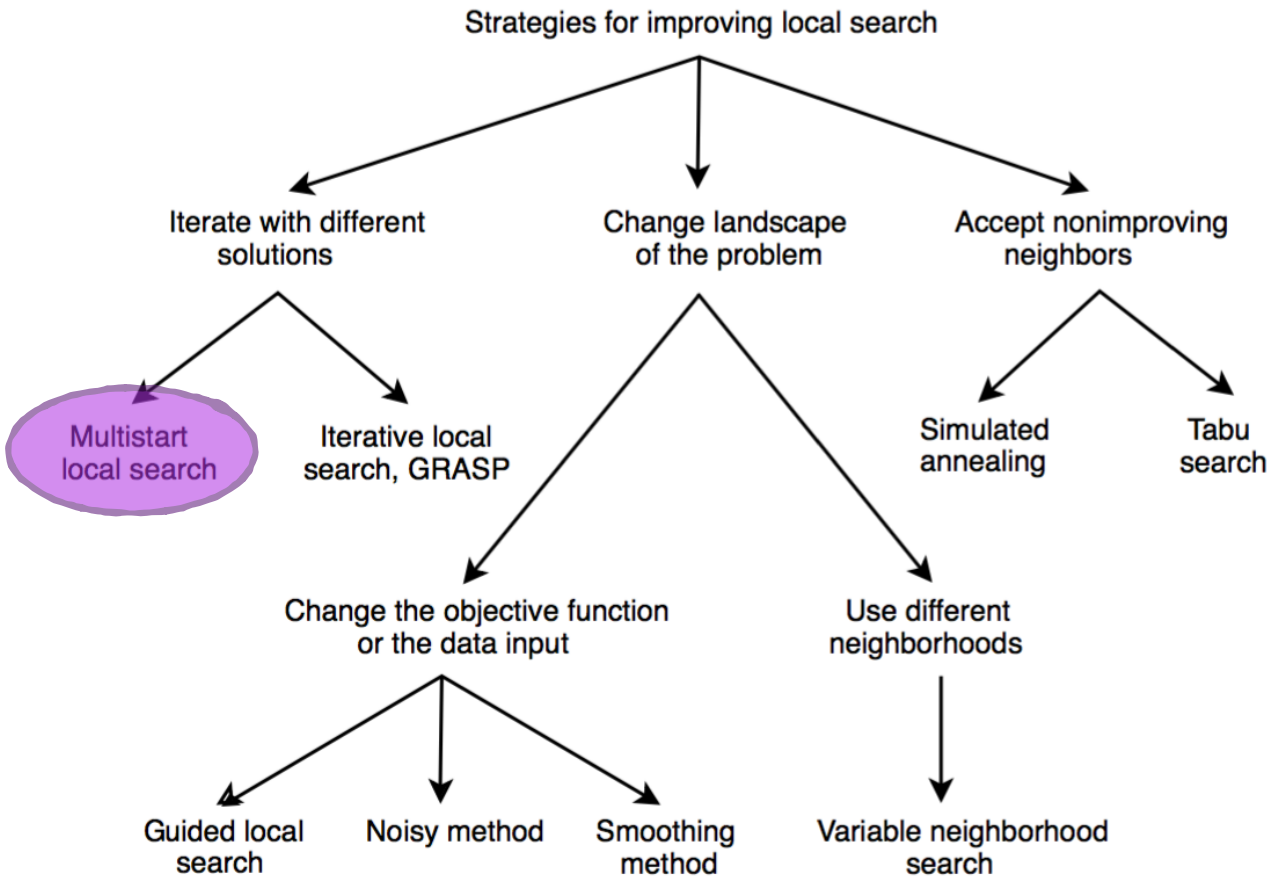
*Prof. Eduardo Pécora*

# MULTI START LOCAL SEARCH

## ESCAPING FROM LOCAL OPTIMA

*Prof. Eduardo Pécora*

# ESCAPING FROM LOCAL OPTIMA




(c) Eduardo Pecora - Combinatorial Optimization and Metaheuristic Course - GTAO UFPR

# MULTI-START LOCAL SEARCH

{

Construct a Solution - S

Apply a Local Search - LS(S)

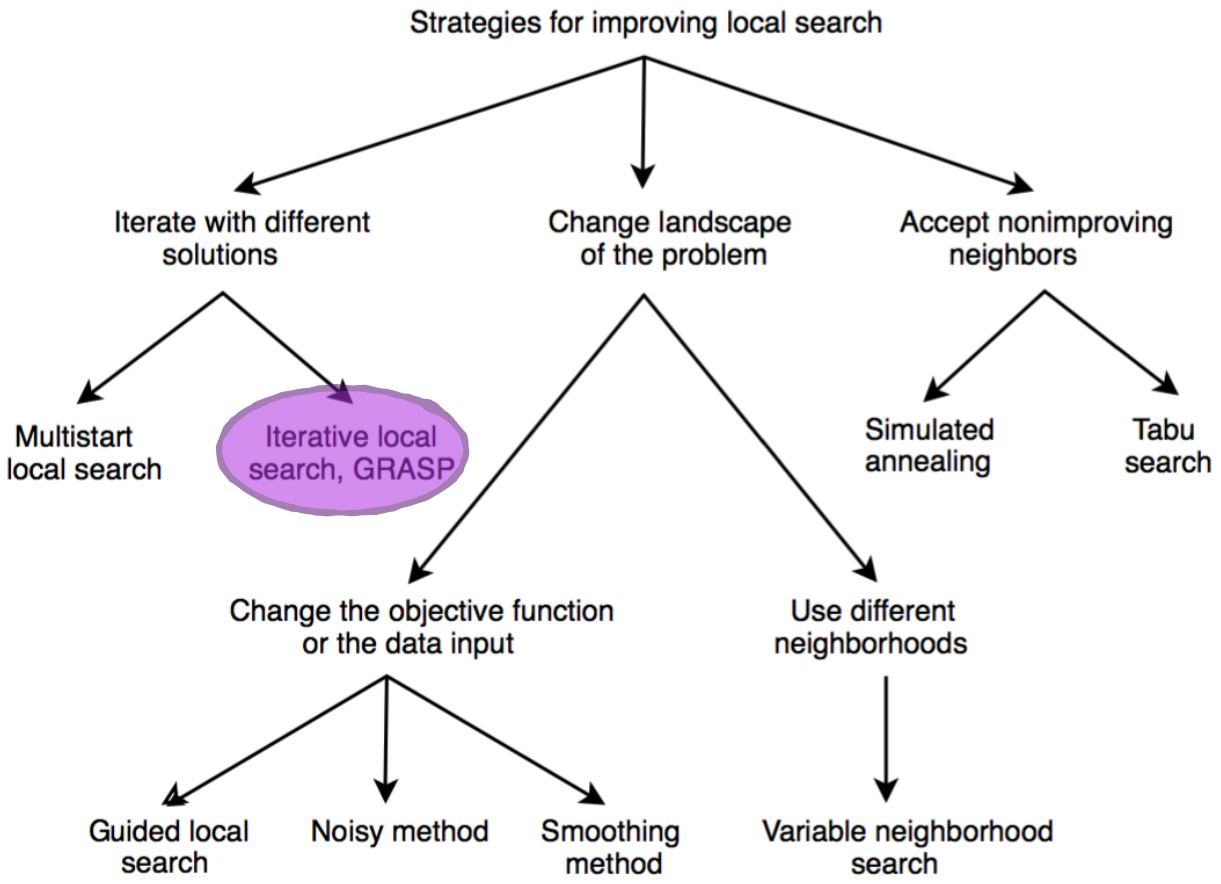
}Repeat until stop criteria

# ITERATED LOCAL SEARCH

## ESCAPING FROM LOCAL OPTIMA

*Prof. Eduardo Pécora*

# ESCAPING FROM LOCAL OPTIMA



# ILS

**Algorithm 2.10** Template of the iterated local search algorithm.

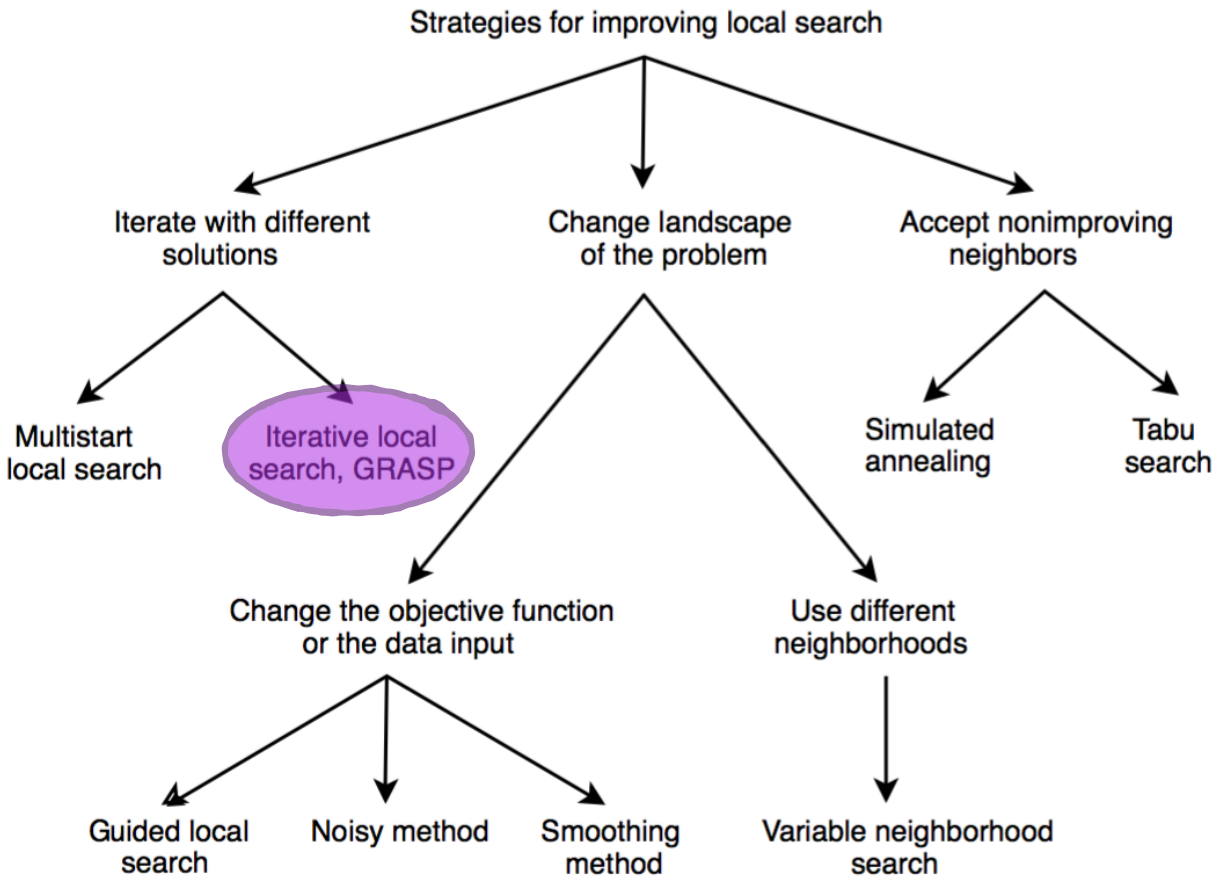
$s_* =$ local search$(s_0)$ ; /* Apply a given local search algorithm */
**Repeat**
  $s' =$ Perturb $(s_*,$ search history$)$ ;  /* Perturb the obtained local optima */
  $s'_* =$ Local search $(s')$ ; /* Apply local search on the perturbed solution */
  $s_* =$ Accept $(s_*, s'_*,$ search memory$)$ ; /* Accepting  criteria */
**Until** Stopping criteria
**Output:** Best solution found.

# GREED RANDOMIZED ADAPTATIVE SEARCH PROCEDURE

## ESCAPING FROM LOCAL OPTIMA

*Prof. Eduardo Pécora*

# GRASP – GREED RANDOMIZED ADAPTATIVE SEARCH PROCEDURE



# GRASP – GREED RANDOMIZED ADAPTATIVE SEARCH PROCEDURE

- The Greedy Randomized Adaptative Search Procedure was proposed by Feo et Resende (1995);
- Each iteration of GRASP consists of two phases: constructive and local search.
  - RCL - Restrict Candidate List;
    - The size of the RCL is defined by the value of $\alpha \in [0,1]$. Where, $\alpha = \begin{cases} 1, & \text{totally random ;} \\ 0, & \text{totally greedy.} \end{cases}$
  - Each element in the list is an element to be inserted on the partial solution. I.e. an object to be inserted into the knapsack.
  - Elemento is chosen by a random selection based on the insertion cost or a greedy function.

# GRASP – GREED RANDOMIZED ADAPTATIVE SEARCH PROCEDURE

**Algorithm 2.17** Template of the greedy randomized adaptive search procedure.

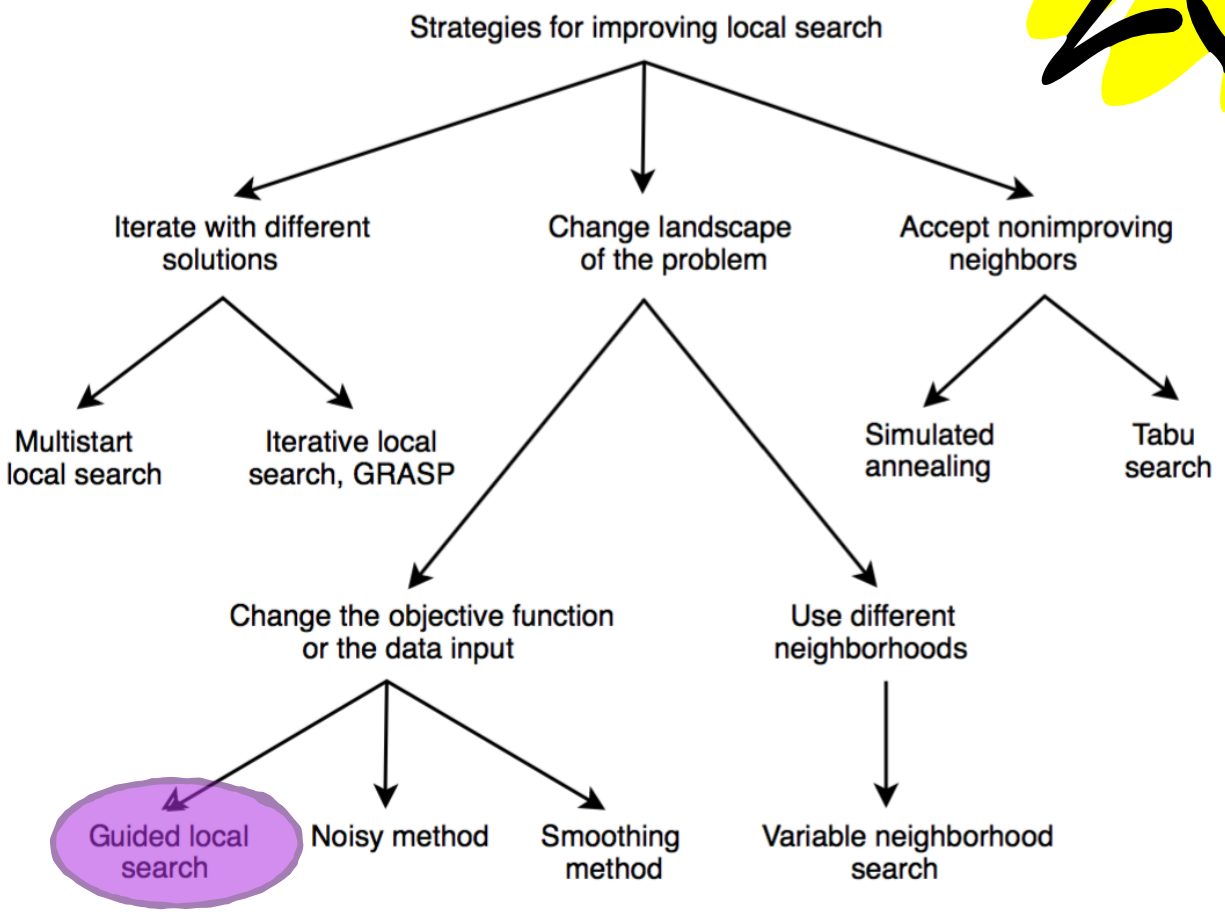
**Input:** Number of iterations.
**Repeat**
  $s$ = Random-Greedy(seed) ; /* apply a randomized greedy heuristic */
  $s' = Local - Search(s)$ ; /* apply a local search algorithm to the solution */
**Until** Stopping criteria /* e.g. a given number of iterations */
**Output:** Best solution found.

# GRASP – GREED RANDOMIZED ADAPTATIVE SEARCH PROCEDURE

**Algorithm 2.18** Template of the greedy randomized algorithm.

$s = \{\}$ ; /* Initial solution (null) */
Evaluate the incremental costs of all candidate elements ;
**Repeat**
  Build the restricted candidate list $RCL$ ;
  /* select a random element from the list $RCL$ */
  $e_i$ = Random-Selection($RCL$) ;
  **If** $s \cup e_i \in F$ **Then** /* Test the feasibility of the solution */
  $s = s \cup e_i$ ;
  Reevaluate the incremental costs of candidate elements ;
**Until** Complete solution found.

# ESCAPING FROM LOCAL OPTIMA



# GUIDED LOCAL SEARCH

**Algorithm 2.14** Template of the guided local search algorithm.

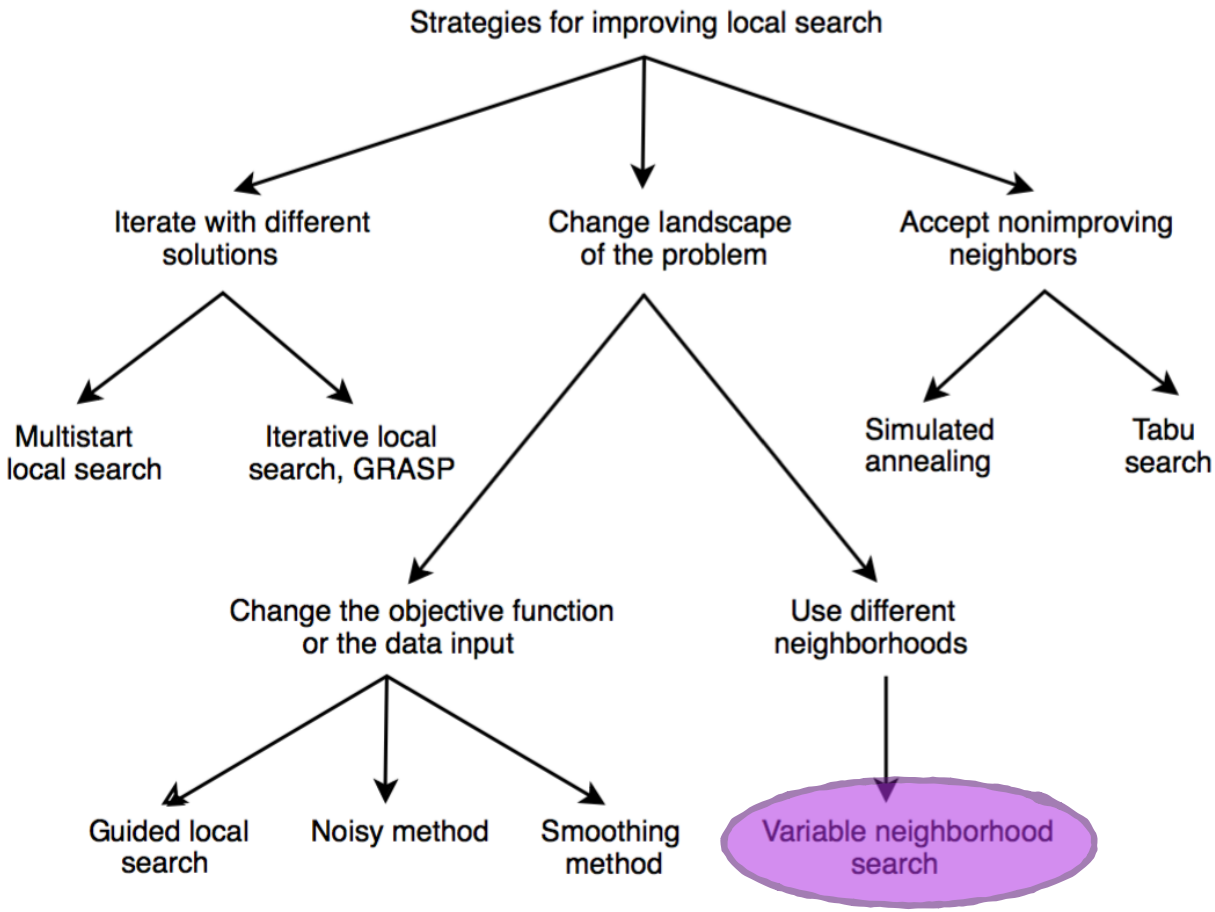
**Input:** S-metaheuristic $LS$, $\lambda$, Features $I$, Costs $c$.
$s = s_0$ /* Generation of the initial solution */
$p_i = 0$ /* Penalties initialization */
**Repeat**
  Apply a S-metaheuristic $LS$ ; /* Let $s^*$ the final solution obtained */
  **For** each feature $i$ of $s^*$ **Do**
    $u_i = \frac{c_i}{1+p_i}$ ; /* Compute its utility */
  $u_j = max_{i=1,..m}(u_i)$ ; /* Compute the maximum utilities */
  $p_j = p_j + 1$ ; /* Change the objective function by penalizing the feature $j$ */
**Until** Stopping criteria /* e.g. max number of iterations or time limit */
**Output:** Best solution found.

# VND AND VNS

## ESCAPING FROM LOCAL OPTIMA

*Prof. Eduardo Pécora*

# ESCAPING FROM LOCAL OPTIMA



# VARIABLE NEIGHBOURHOOD DESCENT

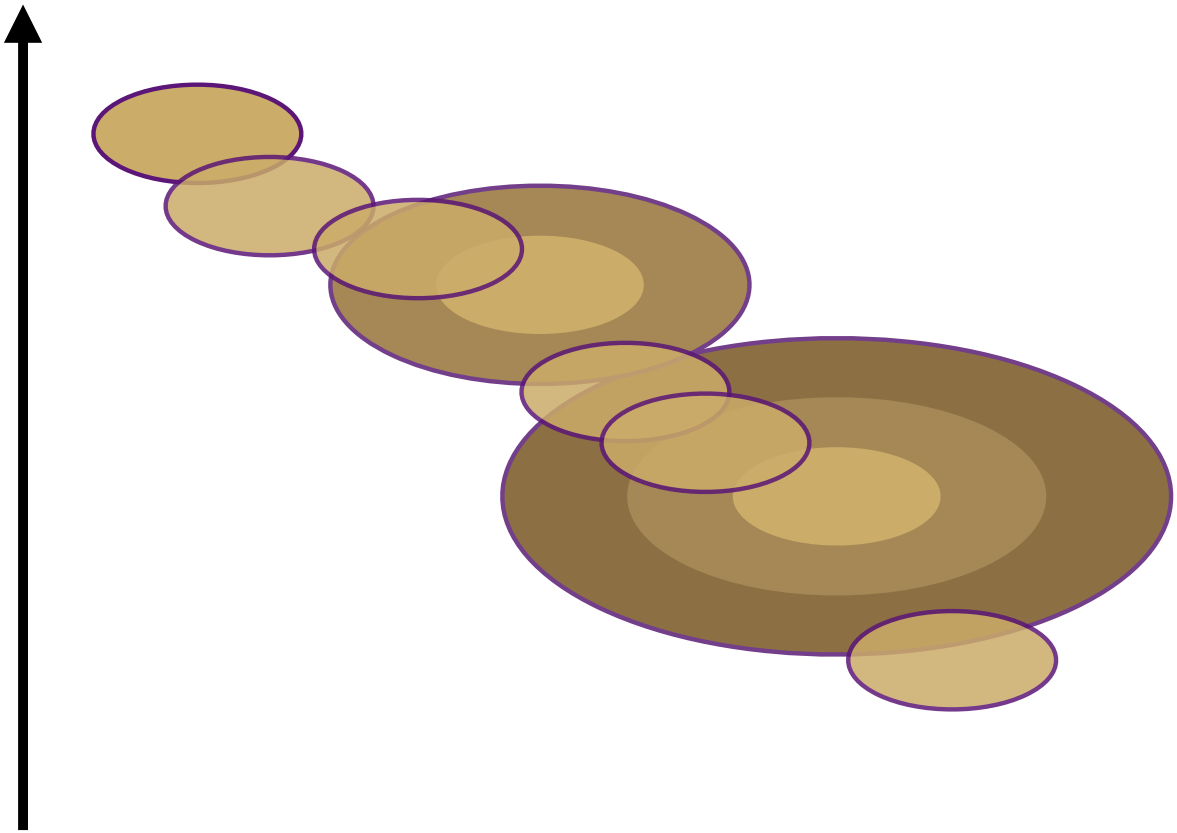
**Algorithm 2.11** Template of the variable neighborhood descent algorithm.

**Input**: a set of neighborhood structures $N_l$ for $l = 1, \ldots, l_{max}$.
$x = x_0$ ; /* Generate the initial solution */
$l = 1$ ;
**While** $l \leq l_{max}$ **Do**
  Find the best neighbor $x'$ of $x$ in $N_l(x)$ ;
  **If** $f(x') < f(x)$ **Then** $x = x'$ ; $l = 1$ ;
  **Otherwise** $l = l + 1$ ;
**Output**: Best found solution.

*The design of the VND algorithm is mainly related to the selection of neighbourhoods and the order of their application. The complexity of t*
*in terms of their exploration and evaluation must be taken into account*

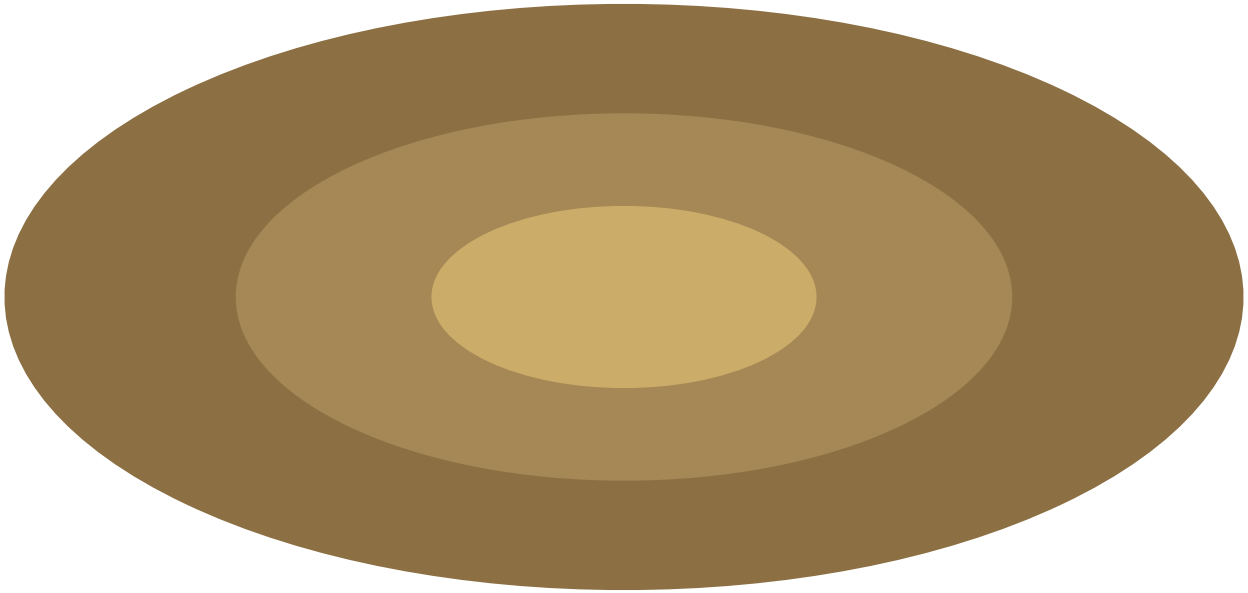
*The most popular strategy is to rank the neighbourhoods following the increasing order of their complexity (e.g., the size of the neighbourh*

# VND



# VND – NESTED NEIGHBOURHOODS

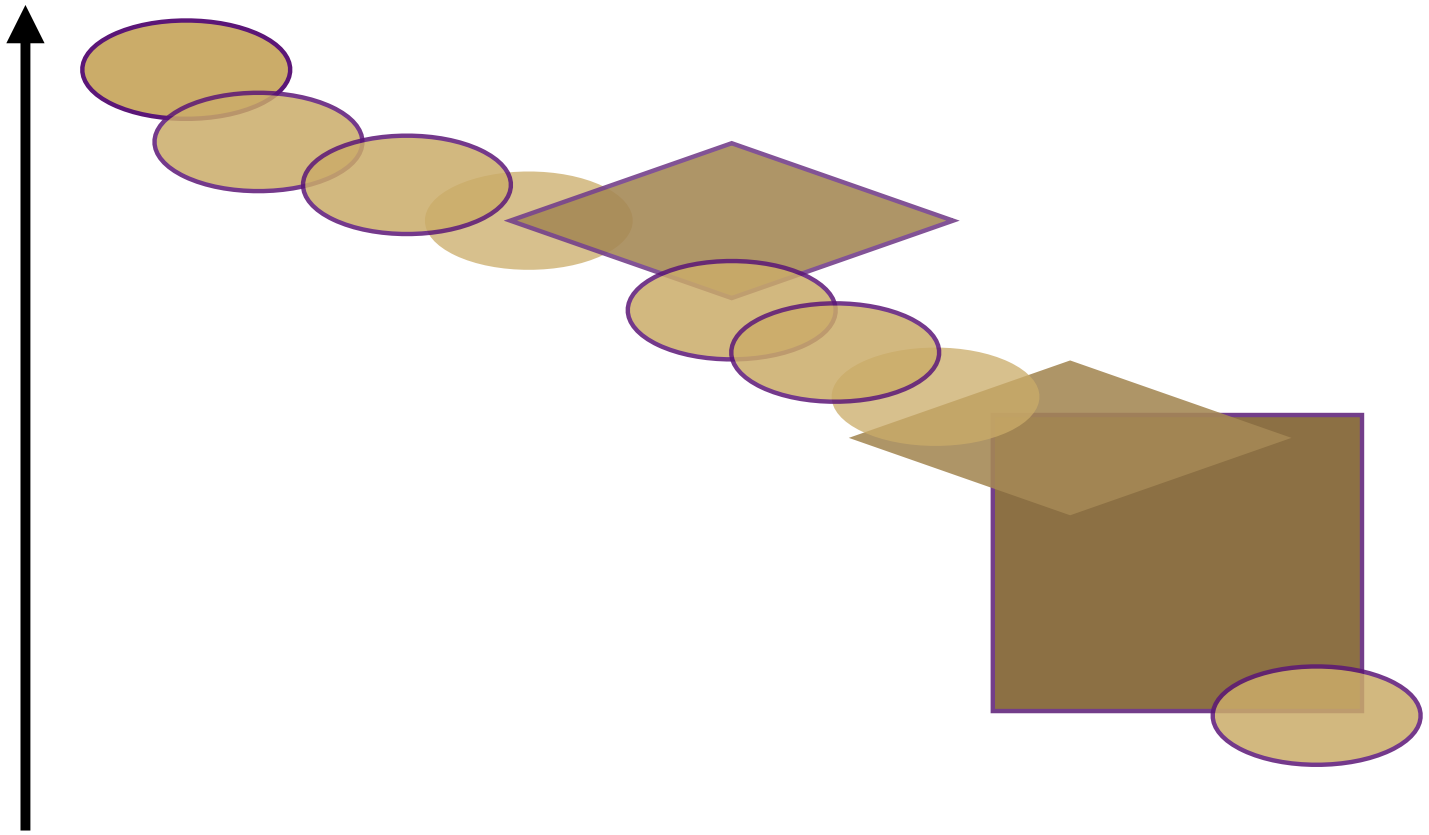
$$N_1(x) \subset N_2(x) \subset \cdots \subset N_k(x), \forall x \in S$$



# VND – NON NESTED NEIGHBOURHOODS

# VND – NON NESTED NEIGHBOURHOODS

# VND



## FIRST DO PROPOSE

*BOCTOR, F. (1993), "Discrete optimization and multi-neighbourhood, local improvement heuristic", Doc. de travail 93-35, FSA, Université Laval.*

## SECOND TO PROPOSE

*M. Mladenovic and P. Hansen. Variable neighborhood search. Computers and Operations Research, 24:1097–1100, 1997.*

# VARIABLE NEIGHBOURHOOD SEARCH

**Algorithm 2.12** Template of the basic variable neighborhood search algorithm.

**Input**: a set of neighborhood structures $N_k$ for $k = 1, \ldots, k_{max}$ for shaking.
$x = x_0$ ; /* Generate the initial solution */
**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 $N_1$ ; $k = 1$ ;
    **Otherwise** k=k+1 ;
  **Until** $k = k_{max}$
**Until** Stopping criteria
**Output**: Best found solution.

# VARIABLE NEIGHBOURHOOD SEARCH

**Algorithm 2.13** Template of the general variable neighborhood search algorithm.

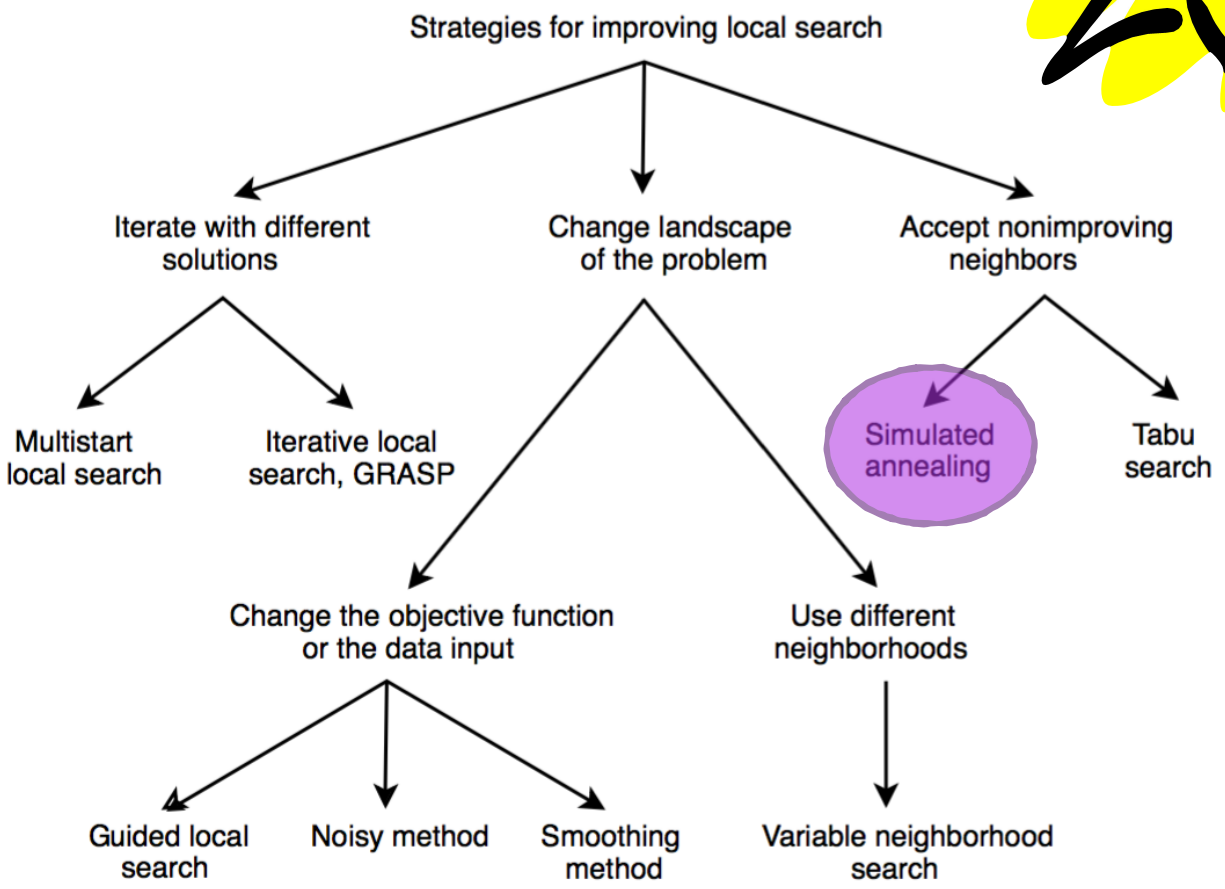
**Input**: a set of neighborhood structures $N_k$ for $k = 1, \ldots, k_{max}$ for shaking.
a set of neighborhood structures $N_l$ for $k = 1, \ldots, l_{max}$ for local search.
$x = x_0$ ; /* Generate the initial solution */
**Repeat**
**For** k=1 **To** $k_{max}$ **Do**
Shaking: pick a random solution $x^{'}$ from the $k^{th}$ neighborhood $N_k(x)$ of $x$ ;
Local search by VND ;
**For** l=1 **To** $l_{max}$ **Do**
Find the best neighbor $x^{''}$ of $x^{'}$ in $N_l(x^{'})$ ;
**If** $f(x^{''}) < f(x^{'})$ **Then** $x^{'} = x^{''}$ ; l=1 ;
**Otherwise** l=l+1 ;
Move or not:
**If** local optimum is better than $x$ **Then**
$x = x^{''}$ ;
Continue to search with $N_1$ $(k = 1)$ ;
**Otherwise** k=k+1 ;
**Until** Stopping criteria
**Output**: Best found solution.

# SIMULATED ANNEALING

## ESCAPING FROM LOCAL OPTIMA

*Prof. Eduardo Pécora*

# ESCAPING FROM LOCAL OPTIMA

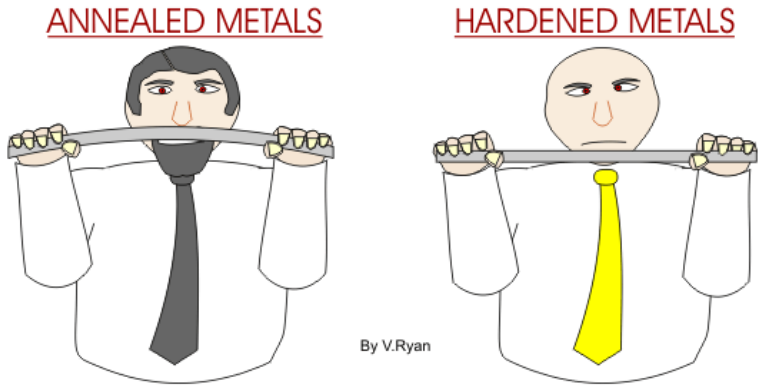


# SIMULATED ANNEALING

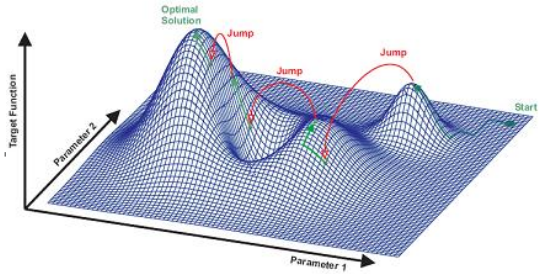
- SA is based on the principles of **statistical** mechanics whereby the annealing process requires heating and then slowly **cooling** a substance to obtain a strong crystalline structure. The strength of the structure depends on the rate of cooling metals. If the initial temperature is not sufficiently high or a fast cooling is applied, imperfections (metastable states) are obtained. In this case, the cooling solid will not attain thermal equilibrium at each temperature. Strong crystals are grown from careful and slow cooling. The SA algorithm simulates the energy changes in a system subjected to a cooling process until it converges to an equilibrium state (steady frozen state). This scheme was developed in 1953.


- S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, 1983.

- V. Cerny. A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications, 45:41–51, 1985.

# SIMULATED ANNEALING




- At each iteration, a **random neighbour** is generated. Moves that improve the cost function are always accepted. Otherwise, the neighbour is selected with a given **probability** that depends on the **current temperature** and the **amount of degradation** $\Delta E$ of the objective function. $\Delta E$ represents the difference in the objective value (energy) between the current solution and the generated neighbouring solution. As the algorithm progresses, the probability that such moves are **accepted decreases**.

# SA ALGORITHM

**Algorithm 2.3** Template of simulated annealing algorithm.

**Input:** Cooling schedule.
$s = s_0$ ; /* Generation of the initial solution */
$T = T_{max}$ ; /* Starting temperature */
**Repeat**
  **Repeat** /* At a fixed temperature */
    Generate a random neighbor $s'$ ;
    $\Delta E = f(s') - f(s)$ ;
    **If** $\Delta E \leq 0$ **Then** $s = s'$ /* Accept the neighbor solution */
    **Else** Accept $s'$ with a probability $e^{\frac{-\Delta E}{T}}$ ;
  **Until** Equilibrium condition
  /* e.g. a given number of iterations executed at each temperature $T$ */
  $T = g(T)$ ; /* Temperature update */
**Until** Stopping criteria satisfied /* e.g. $T < T_{min}$ */
**Output:** Best solution found.

# PARAMETERS: INITIAL TEMPERATURE

- If the starting temperature is very high, the search will be more or less a random local search.

- If the initial temperature is very low, the search will be more or less a first improving local search algorithm.

# PARAMETERS: COOLING FUNCTION

- Linear, Geometric, Logarithmic, Monotonic[*], Very Slow Decrease
- Adaptive: Most of the cooling schedules are static in the sense that the cooling schedule is defined completely *a priori*. In this case, the cooling schedule is "blind" to the characteristics of the search landscape. In an adaptive cooling schedule, the decreasing rate is dynamic and depends on some information obtained during the search [**]. A dynamic cooling schedule may be used where a small number of iterations are carried out at high temperatures and a large number of iterations at low temperatures.

**(non-monotonic) T. C. Hu, A. B. Kahng, and C.-W. A. Tsao. Old bachelor acceptance: A new class of non-monotone threshold accepting methods. ORSA Journal on Computing, 7(4):417–425, 1995.*

***L. Ingber. Adaptive simulated annealing. Control and Cybernetics, 25(1):33–54, 1996.*

# SSS

# SIMULATED ANNEALING – EXERCISES

- Using the Excel Worksheet KnapsackLocalSearch develop a Simulated Annealing Algorithm
  - Define your Neighbourhood you may use (swap or exchange one)
  - Define your cooling function and Equilibrium
  - Use the any Feasible Solution as S0
  - Chose between random neighbour or Best Neighbour
- Apply a Simulated Annealing
- Compare with your colleagues

# WHICH IMPORTANT CONCEPT WE STUDIED HERE?