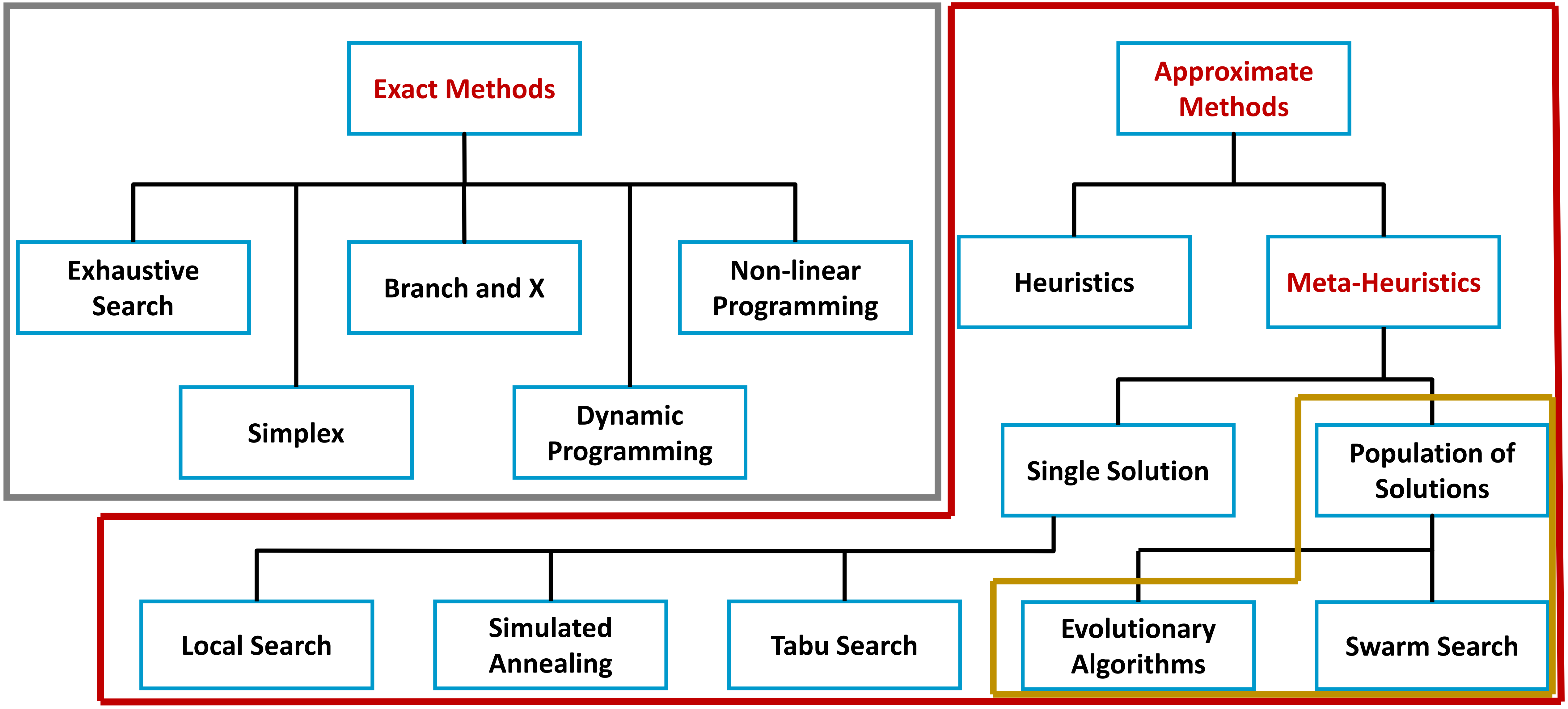
Common Concepts for Metaheuristics

Nuno Antunes Ribeiro

Assistant Professor

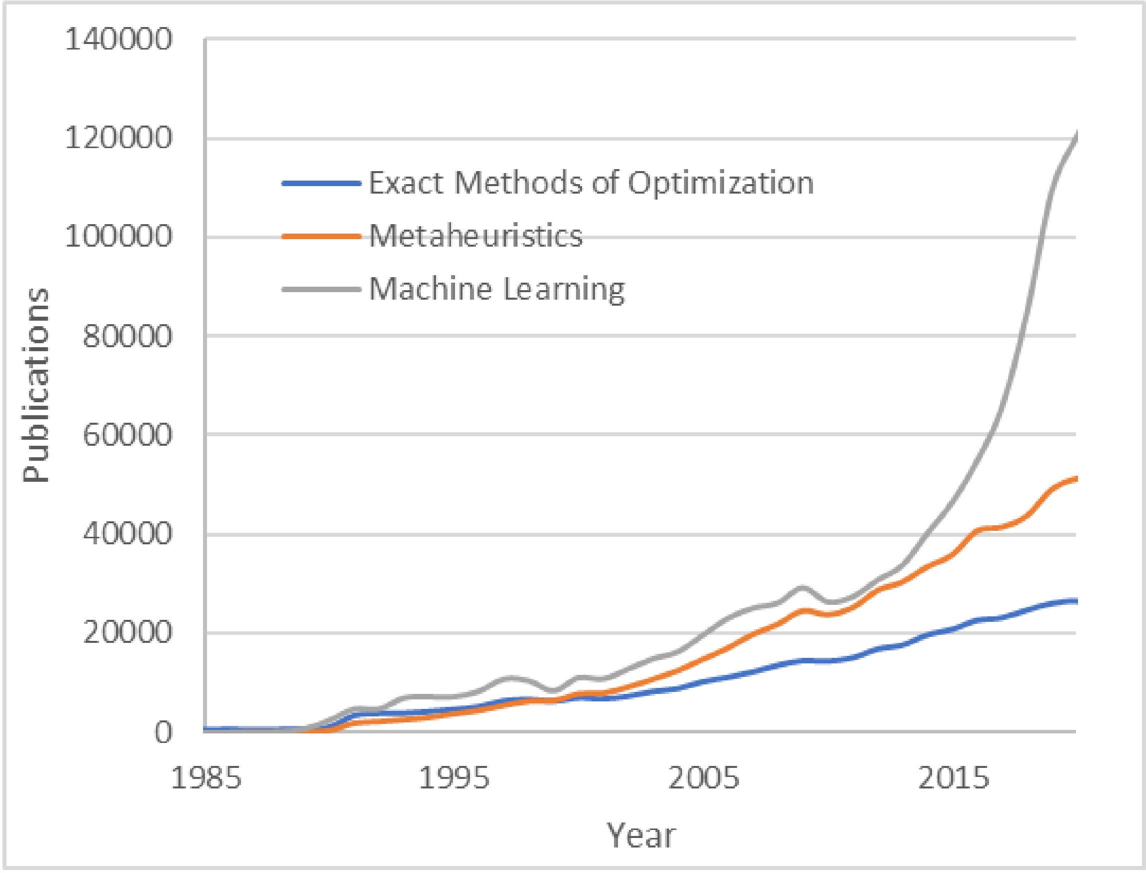
Singapore University of Technology and Design | Engineering Systems and Design

Optimization Methods

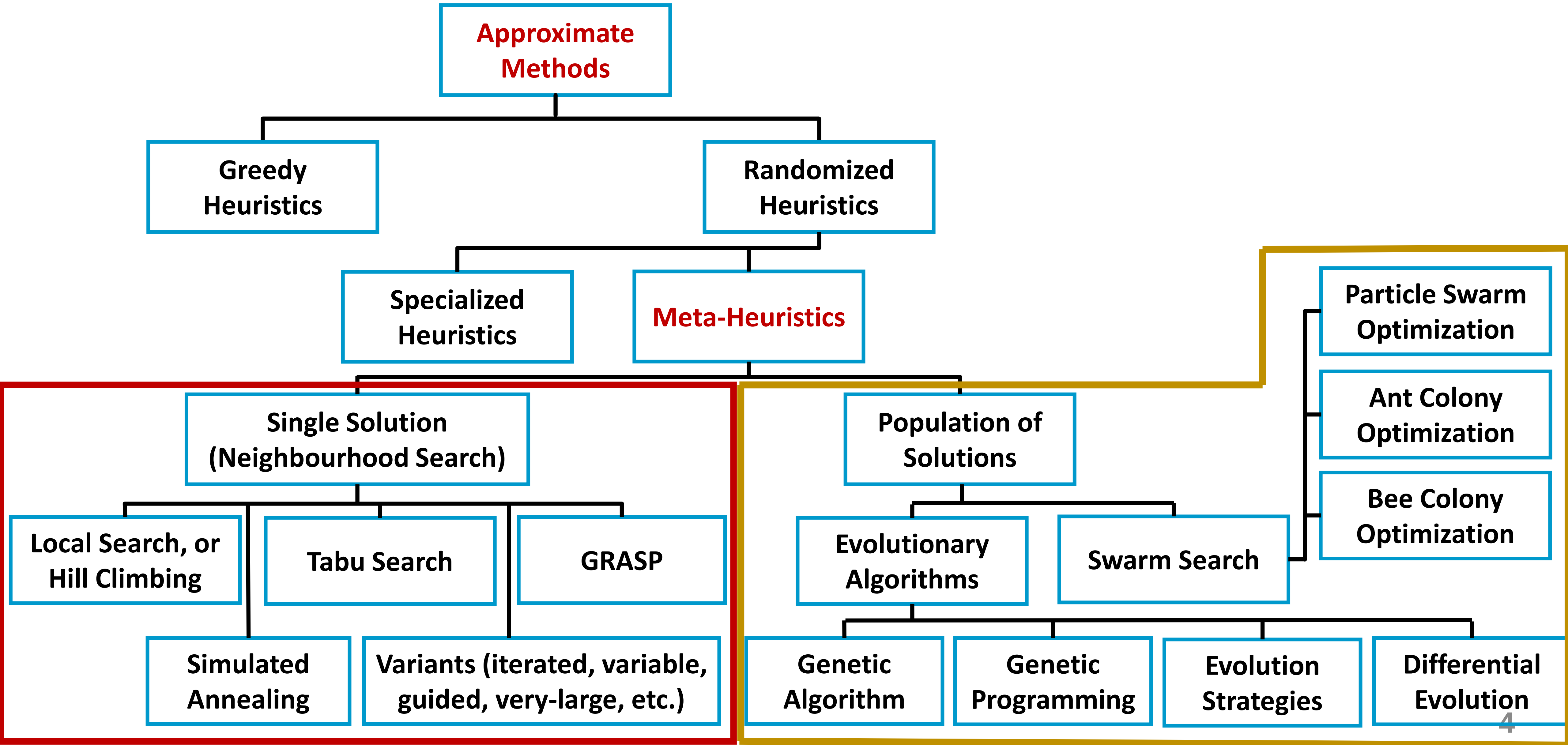


Optimization Methods

- **Exhaustive search** methods are ineffective when solving very large optimization problems
- **Exact methods of Optimization** solve complex problems without the need to exhaustively search for all possible solutions of a problem. These methos ensure that the solution obtained is the optimal one.
- In many instance, the solution space for solutions is so large that exact methods of optimization cannot even find feasible solutions for a problem. That is when we should use **approximate methods**

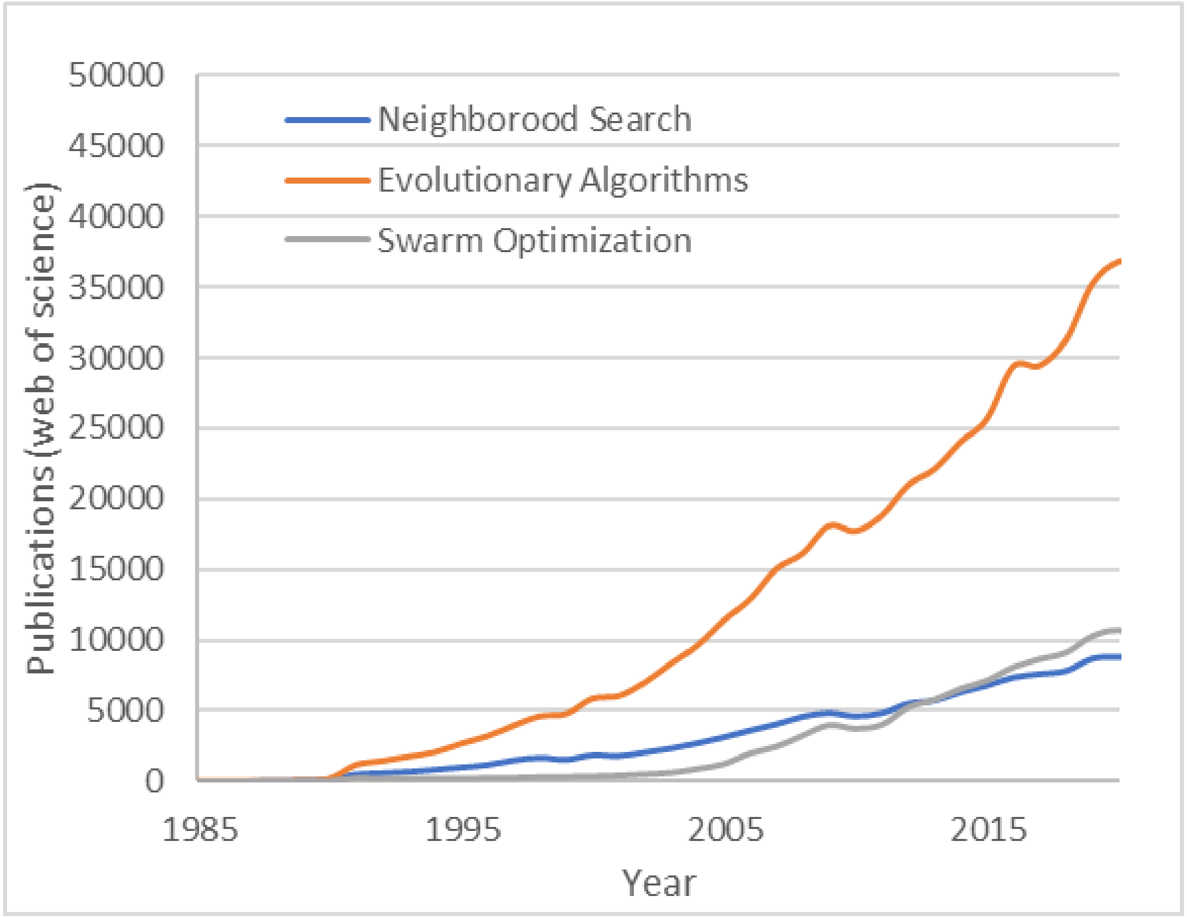


Approximate Methods



Approximate Solution Methods

- **Single solution approaches (or neighbourhood search)** focus on modifying and improving a single candidate solution; single solution metaheuristics include simulated annealing, tabu search, iterated local search, variable neighbourhood search, and guided local search.
- **Population-based approaches** maintain and improve multiple candidate solutions, often using population characteristics to guide the search; population based metaheuristics include evolutionary computation, genetic algorithms, and particle swarm optimization.



Single Solution Methods

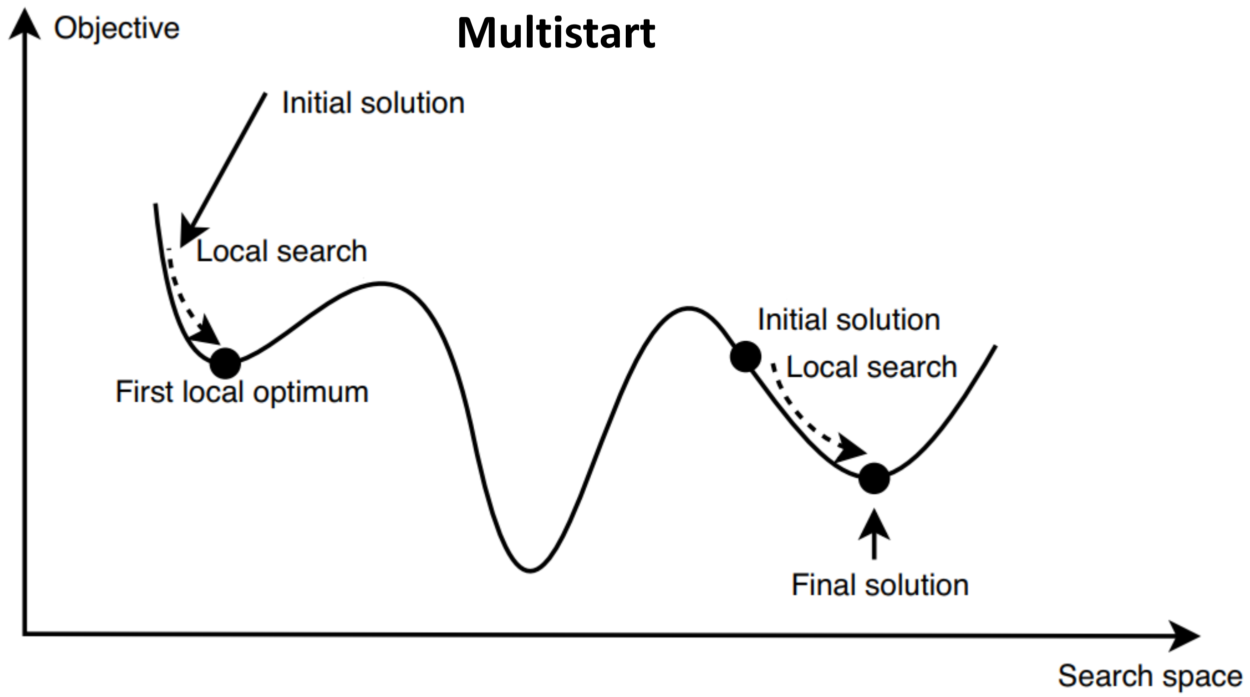
- **Local Search (or Hill Climbing):** oldest and simplest metaheuristic method. At each iteration, the heuristic replaces the current solution by a neighbour that improves the objective function. The search stops when all candidate neighbours are worse than the current solution, meaning a local optimum is reached.
- **Simulated Annealing:** Similar to local search, but worse solutions might be accepted. It uses a control parameter, called temperature, to determine the probability of accepting nonimproving solutions. The temperature is gradually decreased until the algorithm behaves as pure local search.
- **Tabu Search:** Similar to random walks, as it always accepts nonimproving solutions. In contrast to random walks, TS explores the whole neighborhood of a solution (best descent). To avoid cycles, TS discards the neighbours that have been recently visited by memorizing them in a tabu list.
- **GRASP:** Iterative greedy heuristic. At each iteration a feasible solution is generated using a randomized greedy algorithm. Afterwards a neighbourhood search algorithm is applied to improve the solution generated.

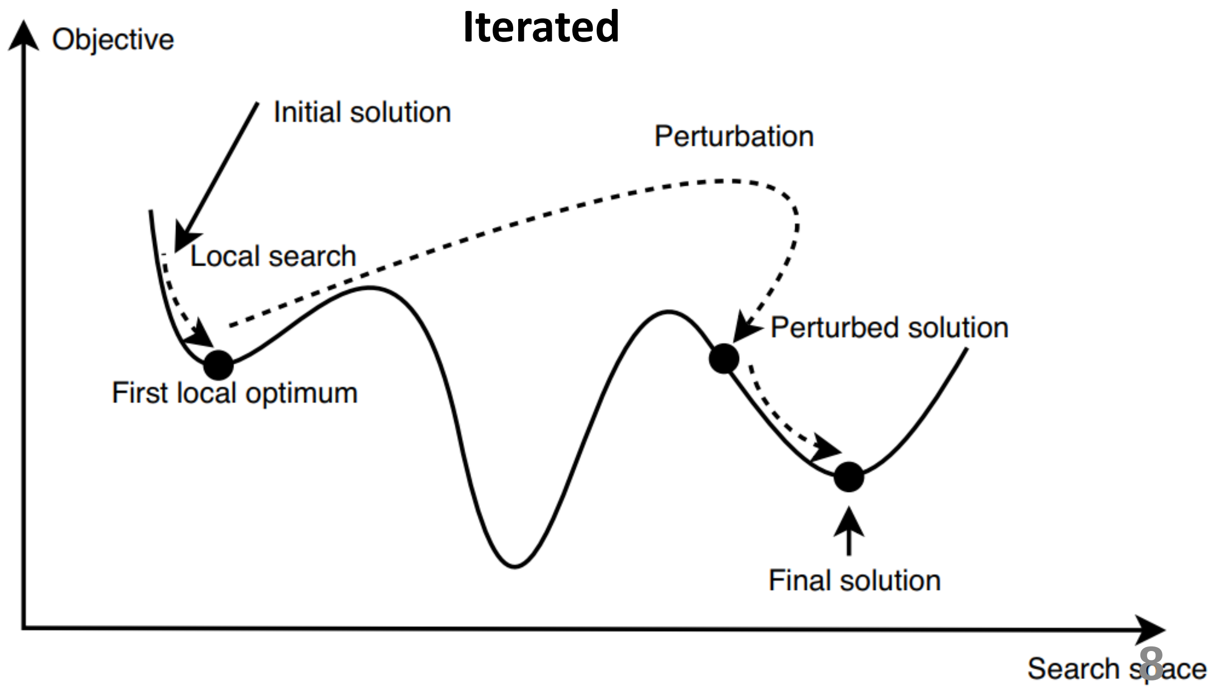
Variants - Single Solution Methods

- Multistart Local Search
- Iterated Local Search
- Multistage Local Search
- Variable Neighbourhood Search
- Guided Local Search
- Very-Large Neighbourhood Search

Variants - Single Solution Methods

- **Multistart vs Iterated Local Search**
  - **Multistart local search**, the initial solution is always chosen randomly and then is unrelated to the generated local optima.
  - **Iterated Local Search** improves the classical multistart local search by perturbing the local optima and reconsidering them as initial solutions





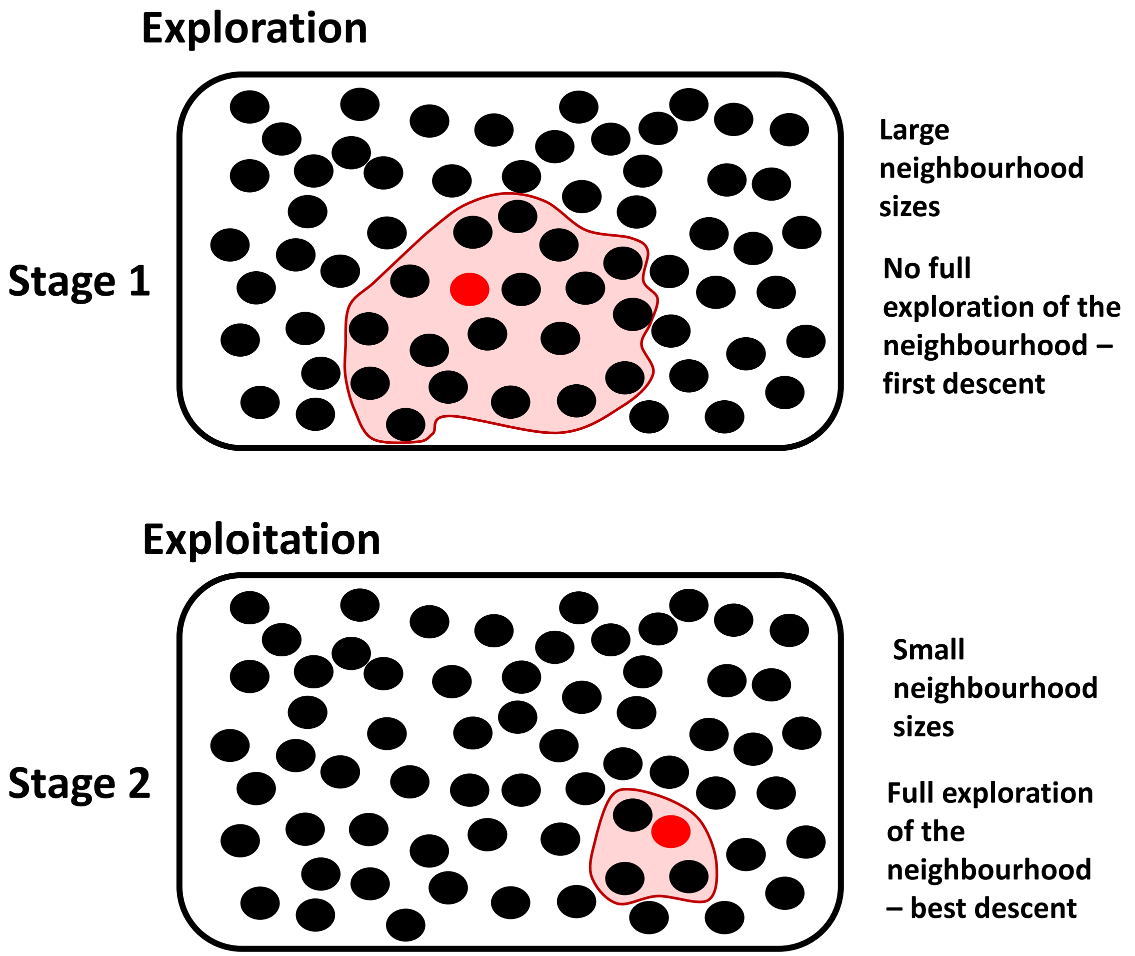
Variants - Single Solution Methods

- **Iterated Local Search - Example**

- **Restart diversification** in Tabu Search - this strategy consists in introducing in the current or best solution the least visited components. Then a **new search is restarted** from this new solution.
- A **perturbation** is applied to the current solution considering the frequency memory of the search procedure so far.
- The **frequency memory** stores for each component of the solution encoding the number of times the component is present in all visited solutions
- Example:
  - How often a variable had assumed a value 1 in a binary problem
  - How often a variable has assumed a certain value in a discrete problem
  - How often an edge have been selected in a permutation problem
  - Etc.

Variants - Single Solution Methods

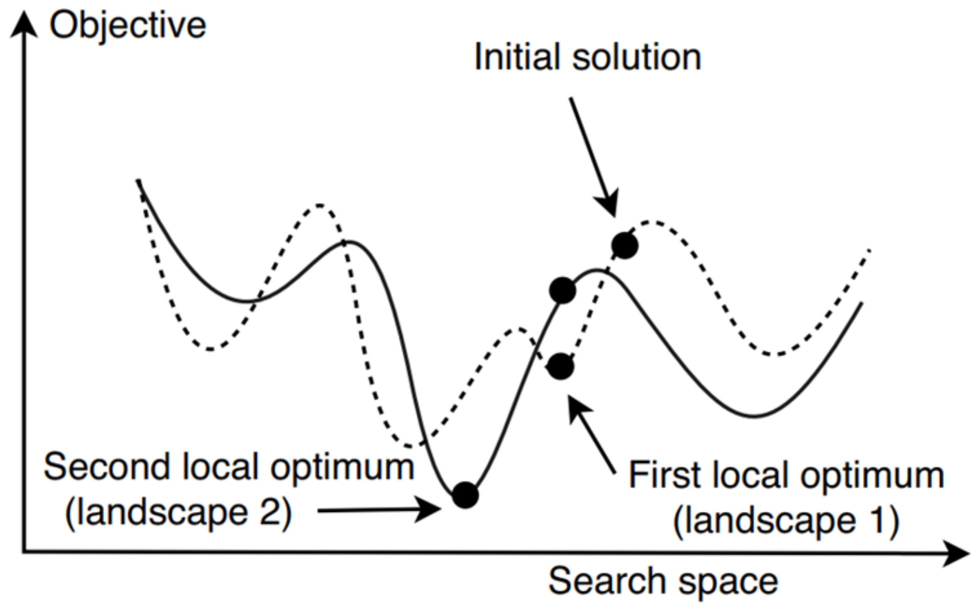
- **Multistage Local Search**
- The basic idea of Multistage Local Search is having successive stages where different metaheuristic strategies are applied.
- For instance:
  - Having different search operators (e.g. swap operator followed by 2-opt operator)
  - Having different **neighbourhood explorations** (e.g. first descent, followed by best descent)
  - Having different metaheuristic approaches (e.g. genetic algorithm followed by simulated annealing)


Exploration
Stage 1
Large neighbourhood sizes
No full exploration of the neighbourhood – first descent
Exploitation
Stage 2
Small neighbourhood sizes
Full exploration of the neighbourhood – best descent

Variants - Single Solution Methods

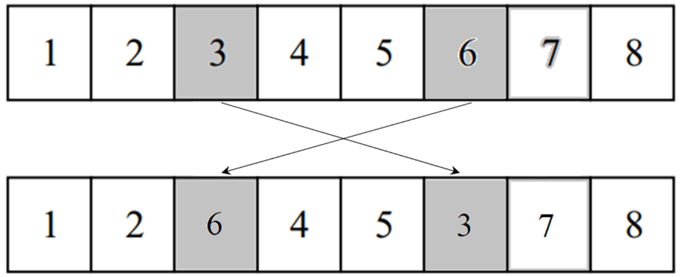
- **Variable Neighbourhood Search**
- The basic idea of Variable Neighbourhood Search VNS is to successively **explore a set of predefined neighbourhoods** to provide a better solution.
- It explores either at random or systematically a set of neighbourhoods to get different local optima and to escape from local optima.
- VNS exploits the fact that using various neighbourhoods in local search may generate different local optima and that the global optima is a local optima for a given neighbourhood.
- Different neighbourhoods generate different landscapes

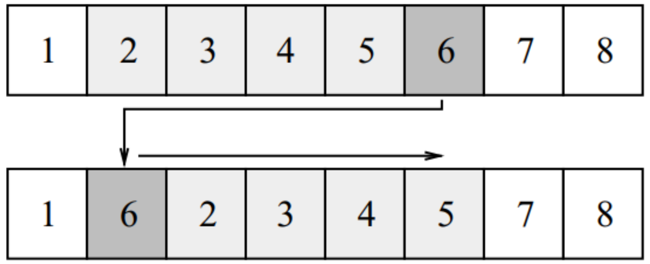
The neighbourhood of a solution is different depending on the move operator used


Objective
Initial solution
Second local optimum (landscape 2)
First local optimum (landscape 1)
Search space

Different Neighbourhoods

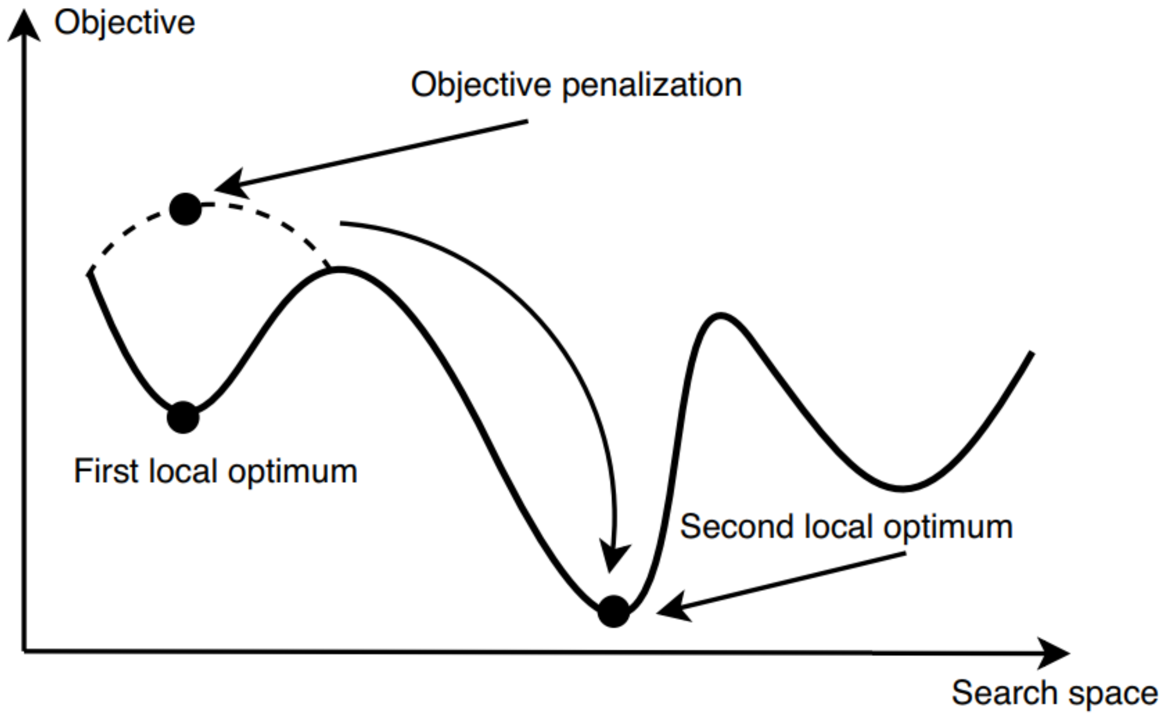
- **Variable Neighborood Search - Example**
- Current Solution: [1,2,3,4,5,6,7,8]
  - We randomly select 6 to be moved using an operator
  - Swap Operator Neighbourhood
    - [6,2,3,4,5,1,7,8]
    - [1,6,3,4,5,2,7,8]
    - [1,2,6,4,5,3,7,8]
    - [1,2,3,6,5,4,7,8]
    - [1,2,3,4,6,5,7,8]
    - [1,2,3,4,5,7,6,8]
    - [1,2,3,4,5,8,7,6]
  - Insertion Operator Neighbourhood
    - [6,1,2,3,4,5,7,8]
    - [1,6,2,3,4,5,7,8]
    - [1,2,6,3,4,5,7,8]
    - [1,2,3,6,4,5,7,8]
    - [1,2,3,4,6,5,7,8]
    - [1,2,3,4,5,7,6,8]
    - [1,2,3,4,5,7,8,6]





Single Solution Methods Variants

- **Guided Local Search**
- In GLS, a set of m features of a solution are defined.
- A solution feature defines a given characteristic of a solution regarding the optimization problem to solve.
- A cost is associated to each feature (updated during the search process).
- When trapped by a local optima, the algorithm will penalize solutions according to some selected features.

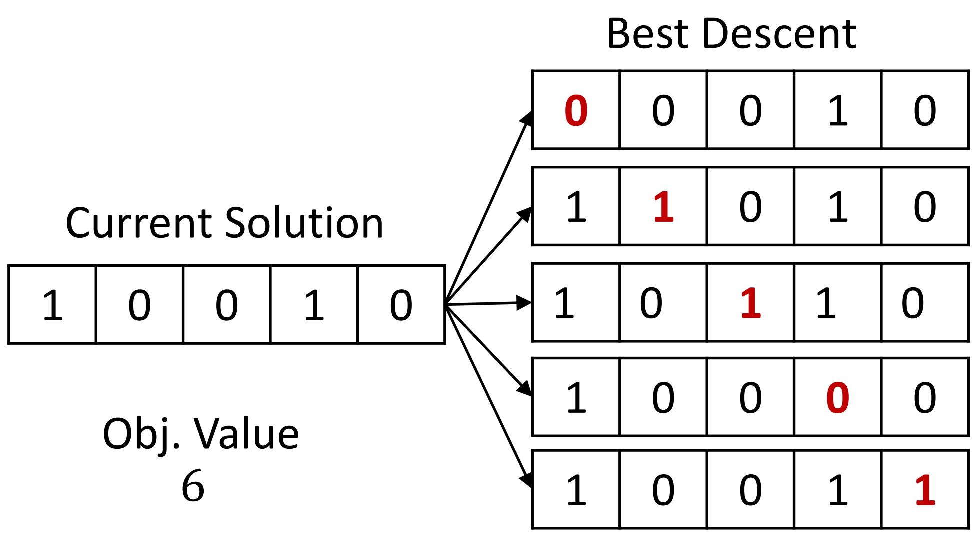


Single Solution Methods Variants

- **Guided Local Search - Example**
- Continuous Diversification - this strategy introduces during a search a bias to encourage diversification
  - v is the actual move value
  - v' is the penalized move value
  - w is a penalty factor
  - q is the frequency ratio

$$v' = \begin{cases} v & \textit{if solution improves} \\ v(1 \pm wq) & \textit{if solution does not improve} \end{cases}$$

Bit Move	Frequency Ratio
1	0.1
2	0.3
3	0.05
4	0.35
5	0.2



Current Solution

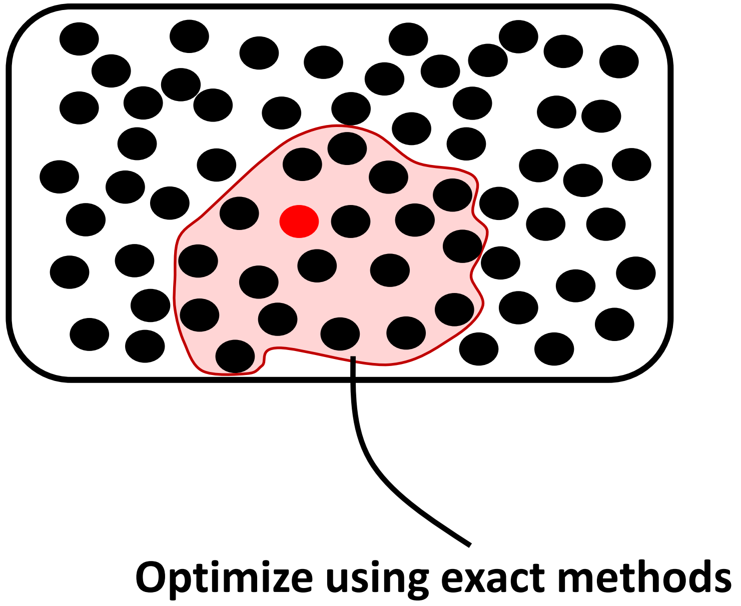
1	0	0	1	0

Obj. Value
6

Best Descent	Move Value	Penalized Move Value
0 0 0 1 0	10	10 (1+0.1)=11
1 **1** 0 1 0	8	8 (1+0.3)=10.4
1 0 **1** 1 0	10	10 (1+0.05)=10.5
1 0 0 **0** 0	7	7 (1+0.35)=9.45
1 0 0 1 **1**	8	**8 (1+0.2)=9.2**

Single Solution Methods Variants

- **Very-large Neighbourhood Search**
- Explore neighbourhoods that would be impossible to analyse using exhaustive search
- Integrates exact methods of optimization and local search
  - Start with an initial feasible solution
  - Select a very large neighbourhood
  - Optimize the neighbourhood using exact methods of optimization
  - Repeat



Topic for next class!

Exploration and Exploitation

Nuno Antunes Ribeiro

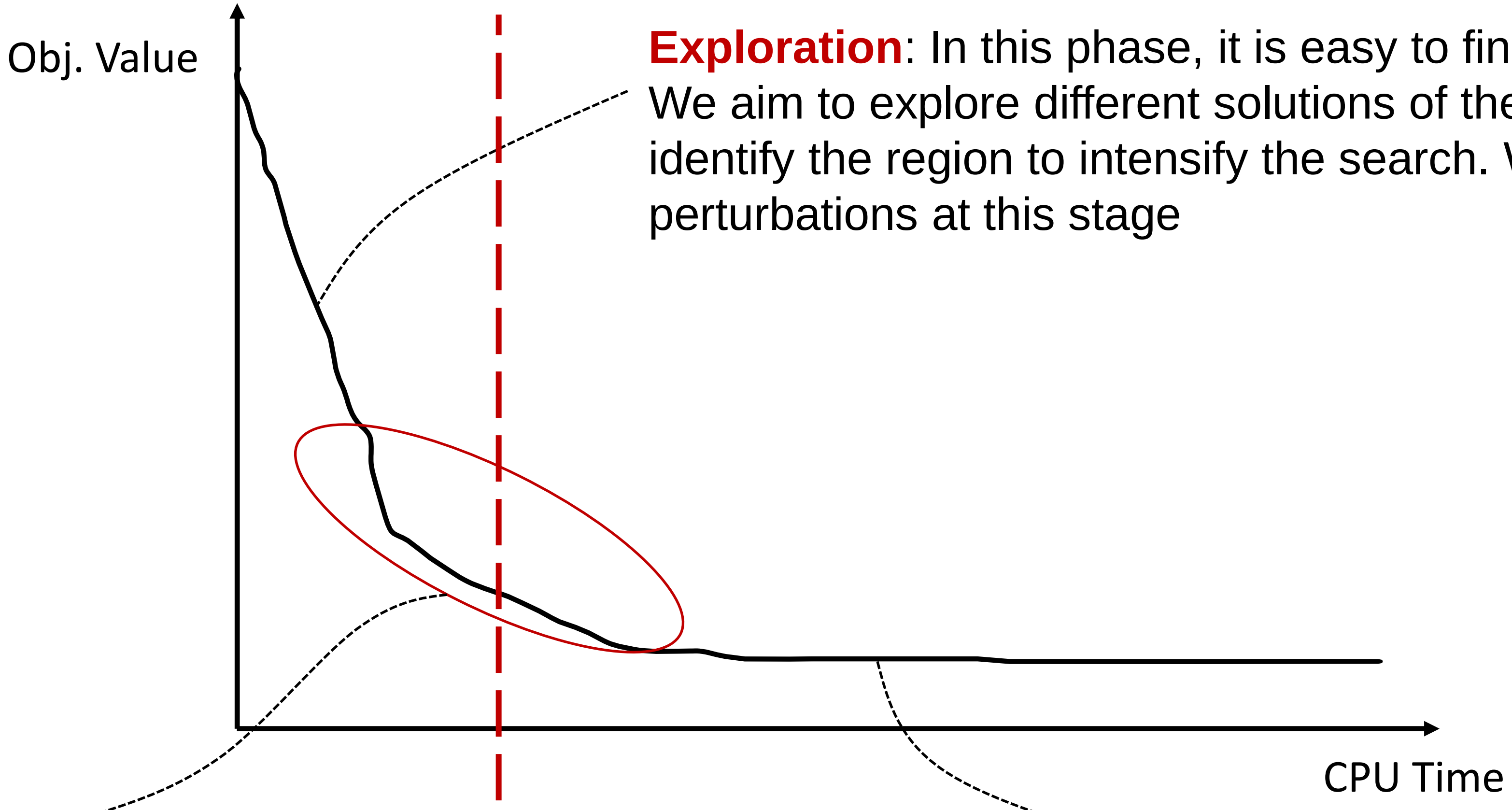
Assistant Professor

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN | Engineering Systems and Design

Convergence

- Sooner or later, every optimization process ends (convergence)
- An algorithm has converged if it cannot reach new candidate solutions anymore, or if it keeps on producing candidate solutions from a small subset of the solution space (i.e. the objective value is no longer improved)
- **Premature convergence** = convergence to local optimum
- What is the basic reason for premature convergence?
  - **Exploration** – search in distant areas of the search space, strong randomization, slow improvement
  - **Exploitation** – analyze neighborhood of current best solutions, fast improvement/ local convergence

Exploration and Exploitation

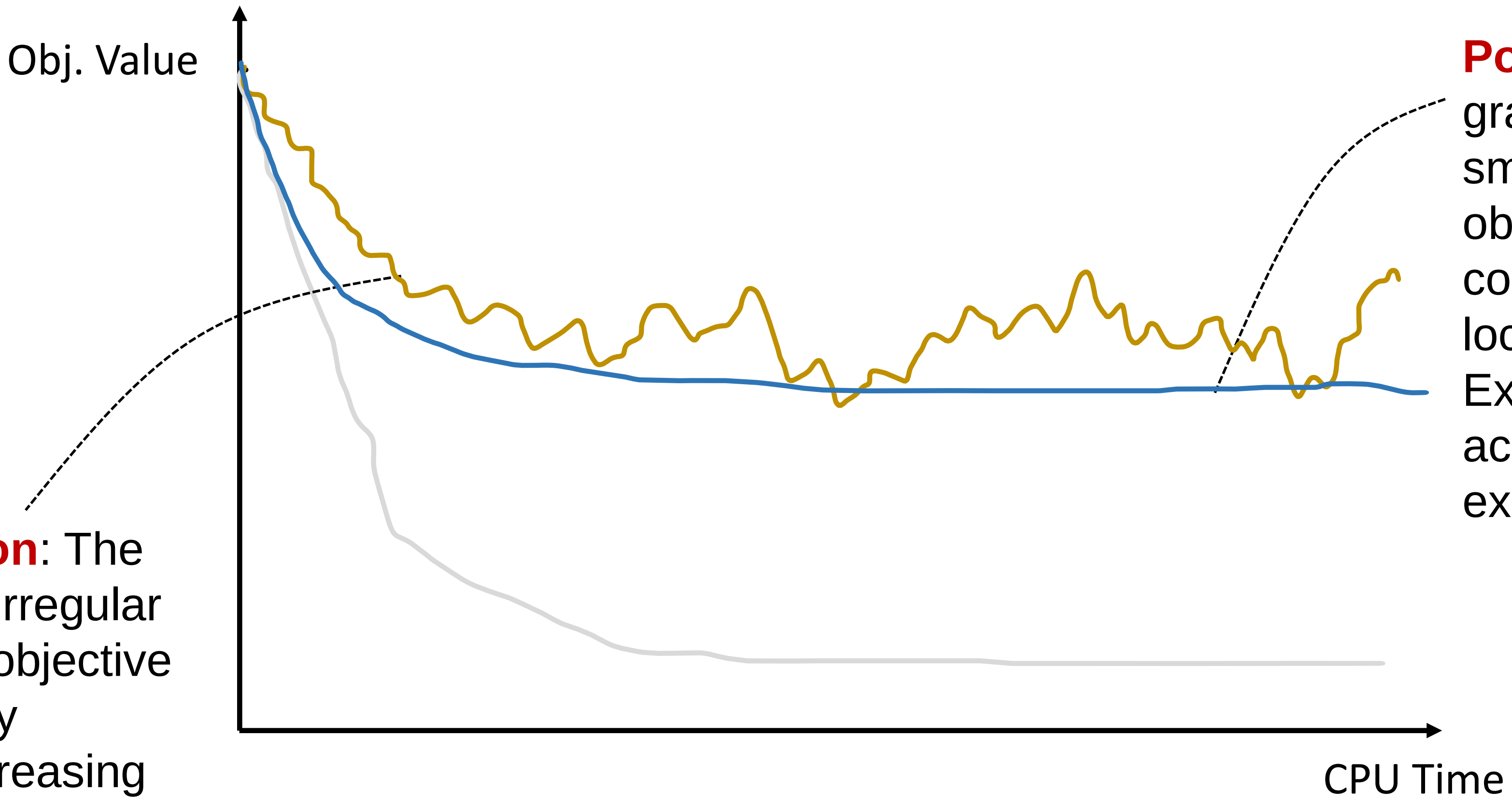


Exploration: In this phase, it is easy to find new solutions. We aim to explore different solutions of the search space to identify the region to intensify the search. We apply larger perturbations at this stage

Exploration + Exploitation: This is the critical phase of the metaheuristic search. Too much exploitation will lead the algorithm to converge to local optima; Too much exploration will lead the algorithm to behave as a random search algorithm, which will require a very large number of iterations to find the global optima

Exploitation: In this phase, hopefully we have found the region of the landscape where the global optimal is located. We aim to intensify the search and explore smaller perturbations.

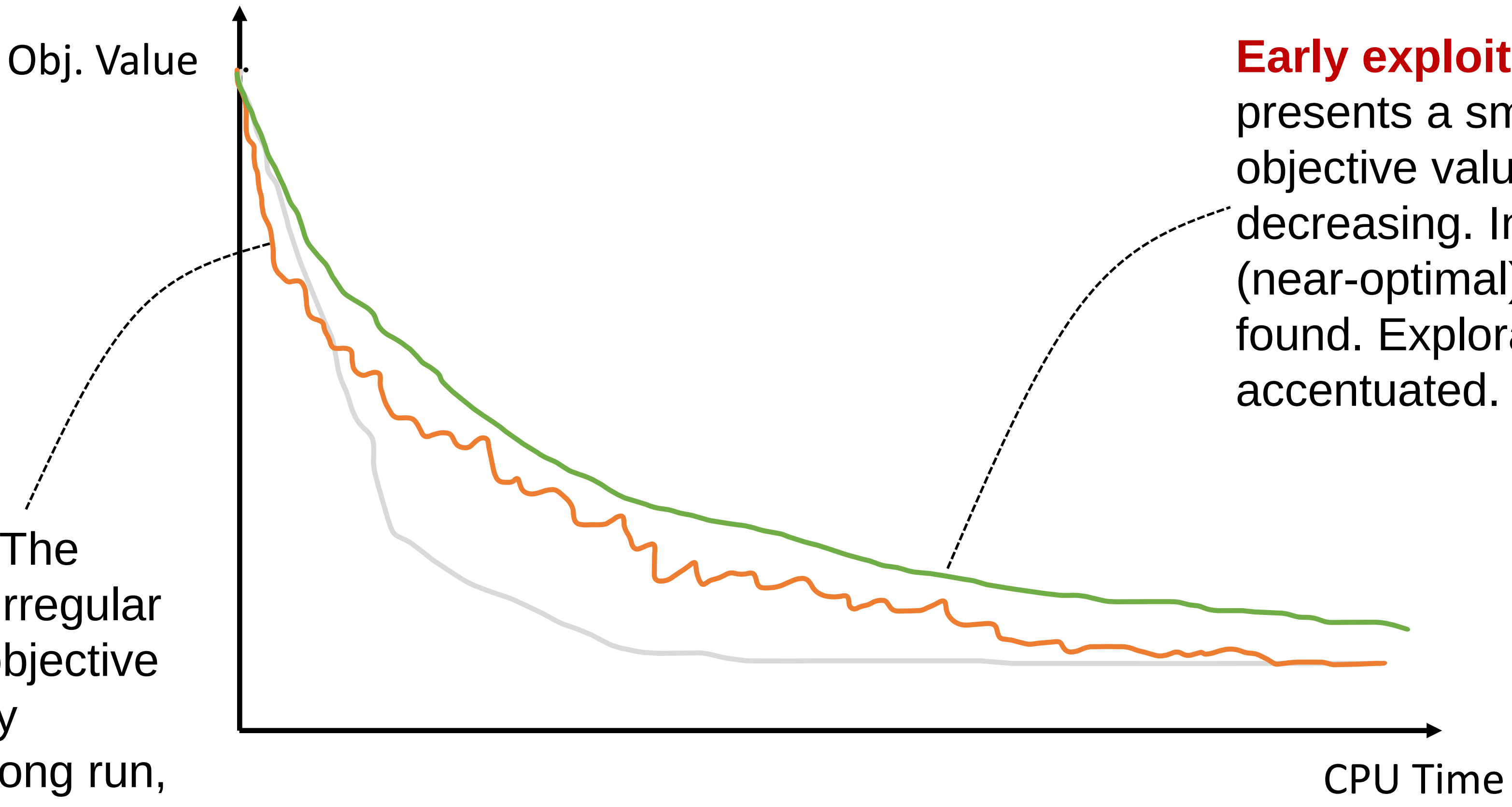
Exploration and Exploitation



Poor Exploration: The graph presents a smooth shape, and the objective value converges quickly to local optima solution. Exploration needs to be accentuated and exploitation eased

Poor intensification: The graph presents an irregular behaviour and the objective value is consistently increasing and decreasing without converging to a minimum value. Exploitation needs to be accentuated and exploration eased

Exploration and Exploitation



Early exploitation: The graph presents a smooth shape and the objective value is consistently decreasing. In the long run, good (near-optimal) solutions can be found. Exploration needs to be accentuated.

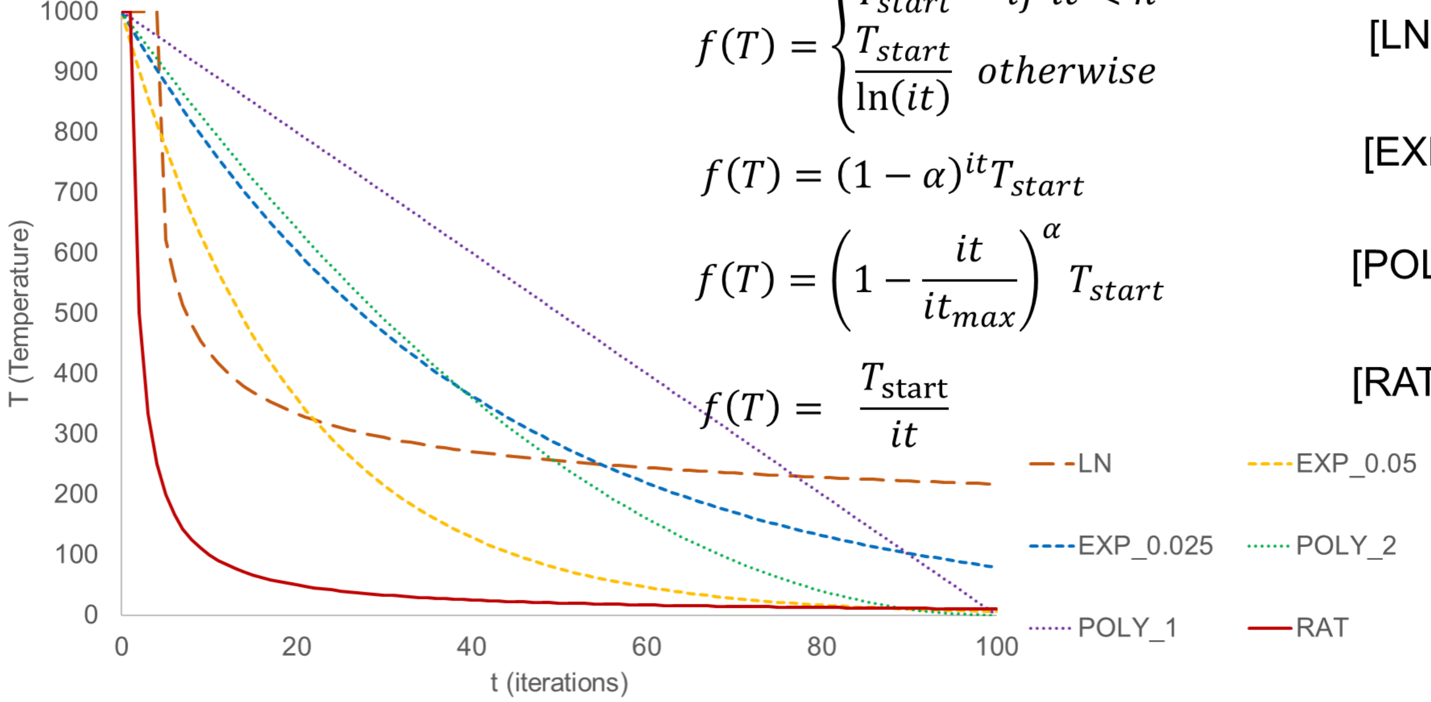
Late exploitation: The graph presents an irregular behaviour but the objective value is consistently decreasing. In the long run, good (near-optimal) solutions are found. Exploitation needs to be accentuated.

Exploration and Exploitation

- **Exploitation Methods**
  - Reduce probability of accepting worse solutions
  - Decrease tabu list size (short-term memory)
  - Apply best descent algorithms
  - Use search operators that apply smaller perturbations
  - Analyze neighborhood of selected elite solutions (medium-term memory)
- **Exploration Methods**
  - Increase probability of accepting worse solutions
  - Increase tabu list size (short-term memory)
  - Apply first descent algorithms
  - Use search operators that apply larger perturbations
  - Apply Iterated/multistart local search
  - Introduce bias to encourage moving to unexplored search areas (long-term memory)

Probability of accepting worse solutions

- If the probability of accepting worse solutions decreases slowly, convergence to the global optimum is likely to occur, but the number of iterations required may be very large
- If the probability of accepting worse solutions decreases fast, convergence to a local optimum is likely and the algorithm will behave as a pure local search



$$f(T) = \begin{cases} T_{start} & if\ it < n \\ \frac{T_{start}}{\ln(it)} & otherwise \end{cases}$$ [LN]

$$f(T) = (1-\alpha)^{it} T_{start}$$ [EXP]

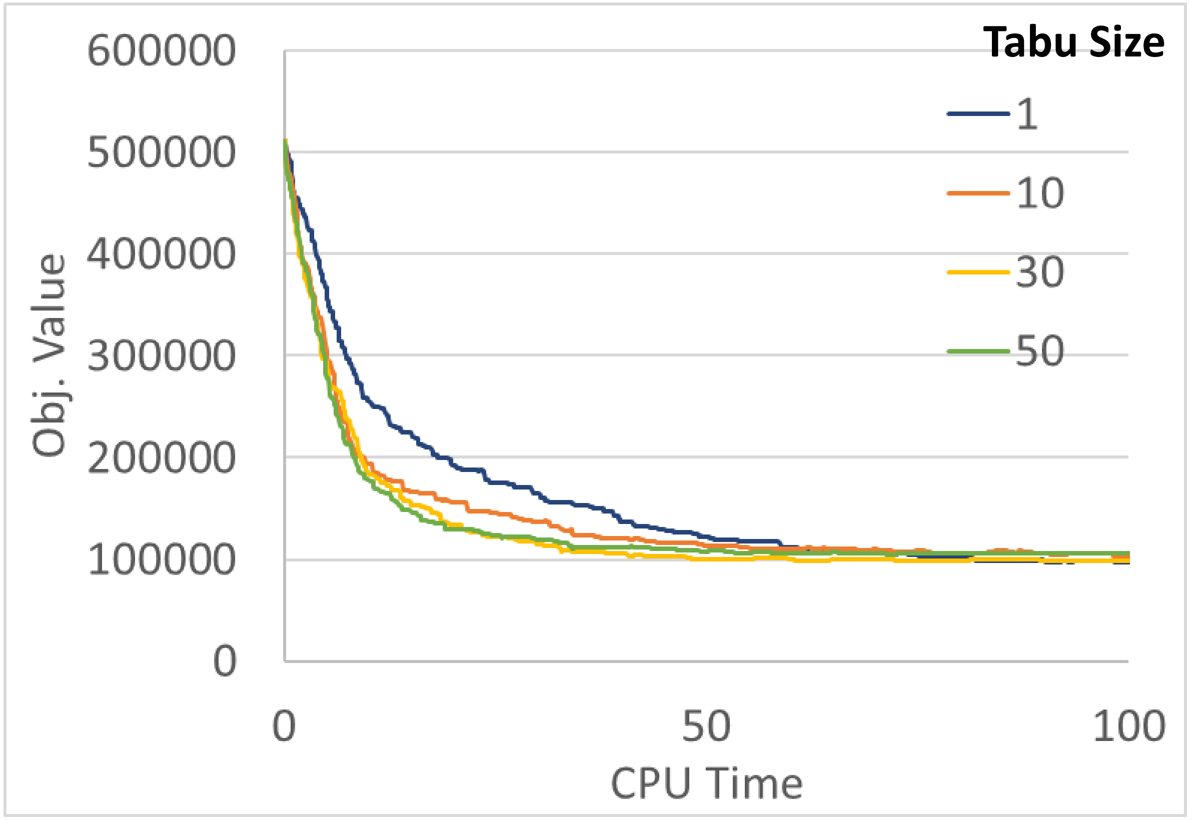
$$f(T) = \left(1 - \frac{it}{it_{max}}\right)^{\alpha} T_{start}$$ [POLY]

$$f(T) = \frac{T_{\text{start}}}{it}$$ [RAT]

24

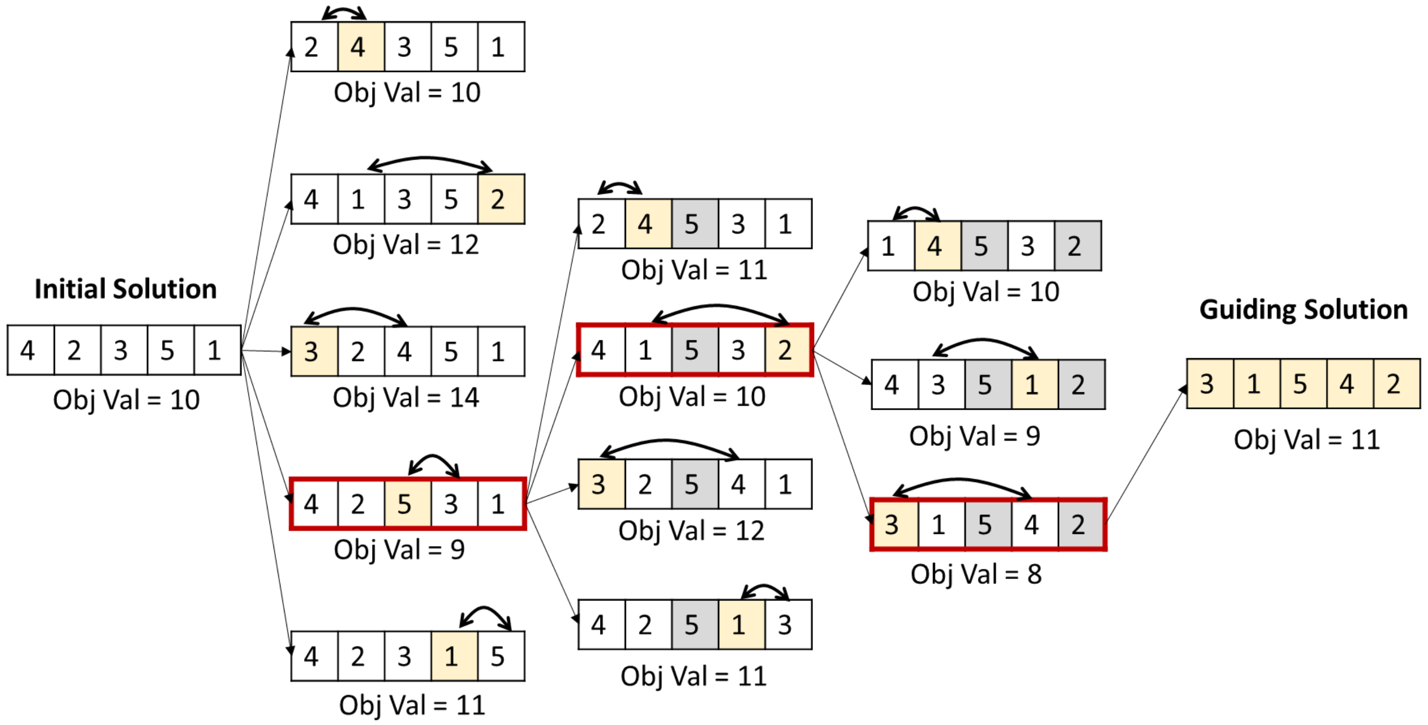
Short-Term Memory

- The role of the short-term memory is to store the recent history of the search to prevent cycling.
- Storing complete solutions generally consumes a massive amount of space and time. Instead, the search moves are typically stored.
- The number of iterations a move is in the short term memory (tabu list size) is critical for the metaheuristic performance
- The smaller is the size of the tabu list, the more likely is the probability of cycling. Larger sizes of the tabu list will provide many restrictions and encourage the exploration.



Medium-Term Memory

- Medium-term memory has been introduced in tabu search to encourage **exploitation** of the search.
- The role of medium-term memory is to exploit the information of the best-found solutions (**elite solutions**) to guide the search in promising regions of the search space.
- **Path-relinking** is a common strategy used for exploitation.

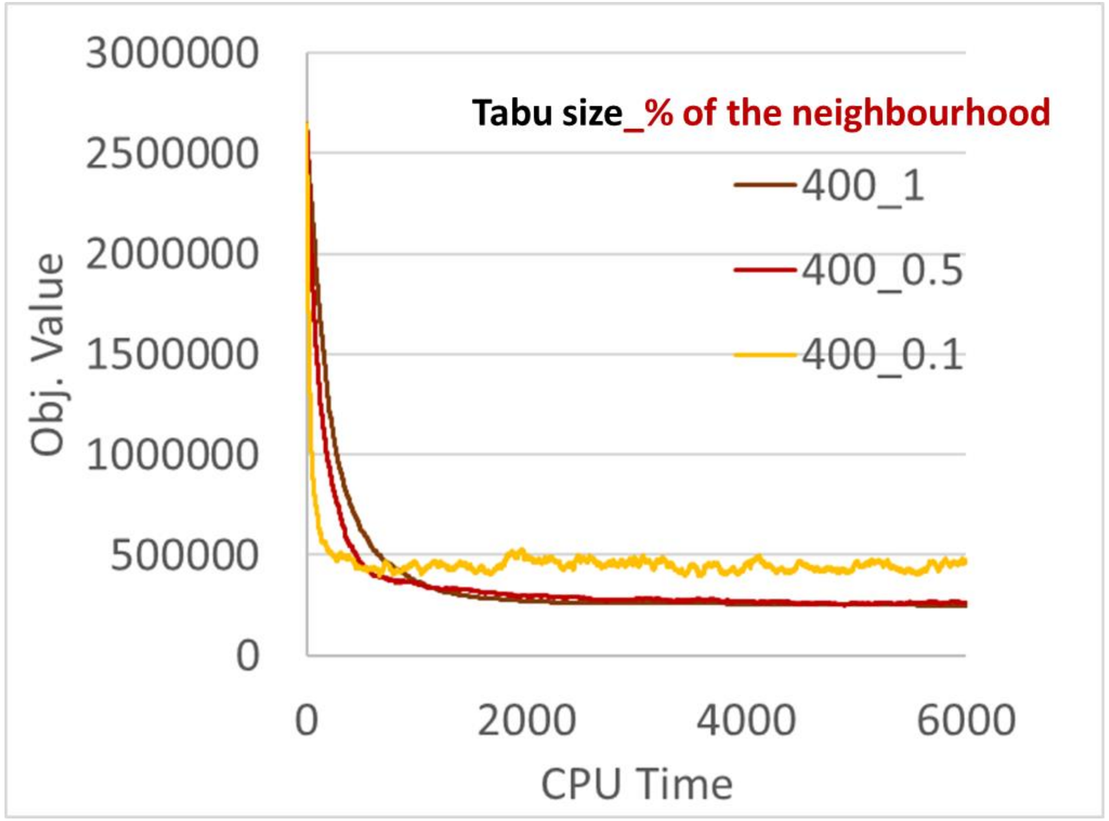


Long Term Memory

- Long-term memory has been introduced in tabu search to encourage the **exploration** of the search.
- The role of the long-term memory is to **force the search in non-explored regions** of the search space.
- The main representation used for the long-term memory is the **frequency memory**.
- Two popular diversification strategies may be applied:
  - **Continuous diversification (type of guided local search):** This strategy introduces during a search a bias to encourage diversification
  - **Restart diversification (type of iterated local search):** This strategy consists in introducing in the current or best solution the least visited components. Then a new search is restarted from this new solution.

First Descent vs Best Descent

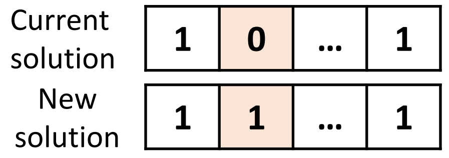
- **First Descent:** This strategy consists in choosing the first improving neighbour that is better than the current solution. Then, an improving neighbour is immediately selected to replace the current solution --- mostly useful during the exploration phase
- **Best Descent:** In this strategy, the best neighbour (i.e., neighbour that improves the most the cost function) is selected. The neighbourhood is evaluated in a fully deterministic manner --- mostly useful during the exploitation phase
- **N-Descent**: In this strategy, N random solutions in the neighbourhood are evaluated. The neighbour that improves the most the cost function) is selected --- useful if exploring the full neighbourhood is too costly



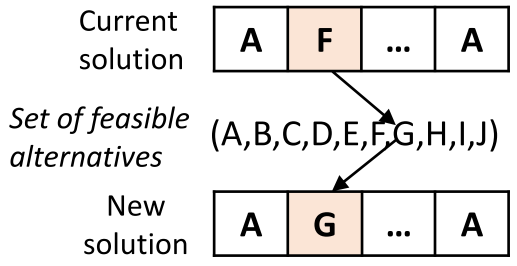
Search Operator

- The efficiency of a solution encoding is also related to the **search operator**.
- When defining a solution encoding, one has to bear in mind how the solution will be **perturbed**.
- More drastic perturbations (for instance flipping 2 bits instead of 1) encourage diversification

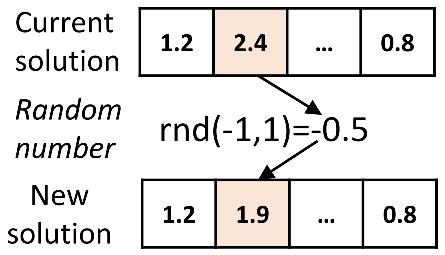
Binary encoding – flip **n** bits of the solution (typically 1 or 2 bits)



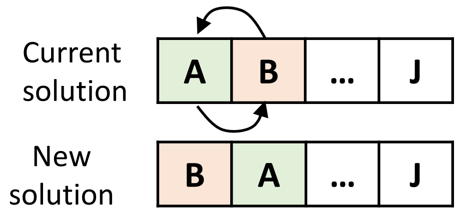
Discrete encoding – update **n** bits of the solution by randomly generating a new value (typically 1 or 2 bits)



Real encoding – update **n** bits of the solution by randomly generating a new value within a certain range (typically 2 elements)



Permutation encoding – swap the location of **n** elements (typically 2 elements)



Adaptive Metaheuristics

- Adaptive local search memorizes some information from the search iterative process with the intention to **guide the search for more promising solutions**.
- Mostly used to simplify the calibration of the metaheuristic parameters.
- Adaptive is a general term - it can be applied to different features of the metaheuristic algorithm,
  - **Constraint Handling** (e.g. decreasing the weight to penalize infeasible solutions when many feasible solutions are generated, and increasing otherwise.
  - **Selection of the Move Operator**: Change the move operator whenever a better solution is not found after n iterations.
  - **Simulated Annealing Parameter Tunning**: Most of the cooling schedules are static. In an adaptive cooling schedule, the decreasing rate is dynamic and depends on some information obtained during the search
  - **Tabu Search Parameter Tunning**: The size of the tabu list can be updated according to the search memory. For instance, the size is updated upon the performance of the search in the last iterations
  - **Iterated Local Search**: A perturbation is generated whenever a better solution is not found after *n* iterations.

Trade-off between Complexity and Simplicity

- Adaptive local search is an advanced feature in metaheuristic optimization. If successfully implemented it can:
  - Make **easier the calibration** of the parameters
  - Ensure **convergence to better solutions faster**

- However, careful analysis need to be done to understand the value of the adaptative approach
  - **Overfitting:** Many different instances must be evaluated to understand if the benefits from the adaptative approach can be extrapolated.
  - **More parameters to calibrate:** Although the adaptive approach aims to simplify the parameter calibration, it may in fact add new parameters to the algorithm. The key is to evaluate the robustness of these parameters (i.e. the algorithm should not be very sensitive to the new parameters --- an example will be provided in the next class)

Hybridization

- Combinations of algorithms such as different metaheuristics, exact methods of optimization, and machine learning techniques have provided very powerful search algorithms.
- Four different types of combinations are typically considered
  - Combining metaheuristics with (complementary) metaheuristics.
  - Combining metaheuristics with exact methods of optimization.
  - Combining metaheuristics with machine learning and data mining techniques.

Fitness Landscape Analysis

Nuno Antunes Ribeiro

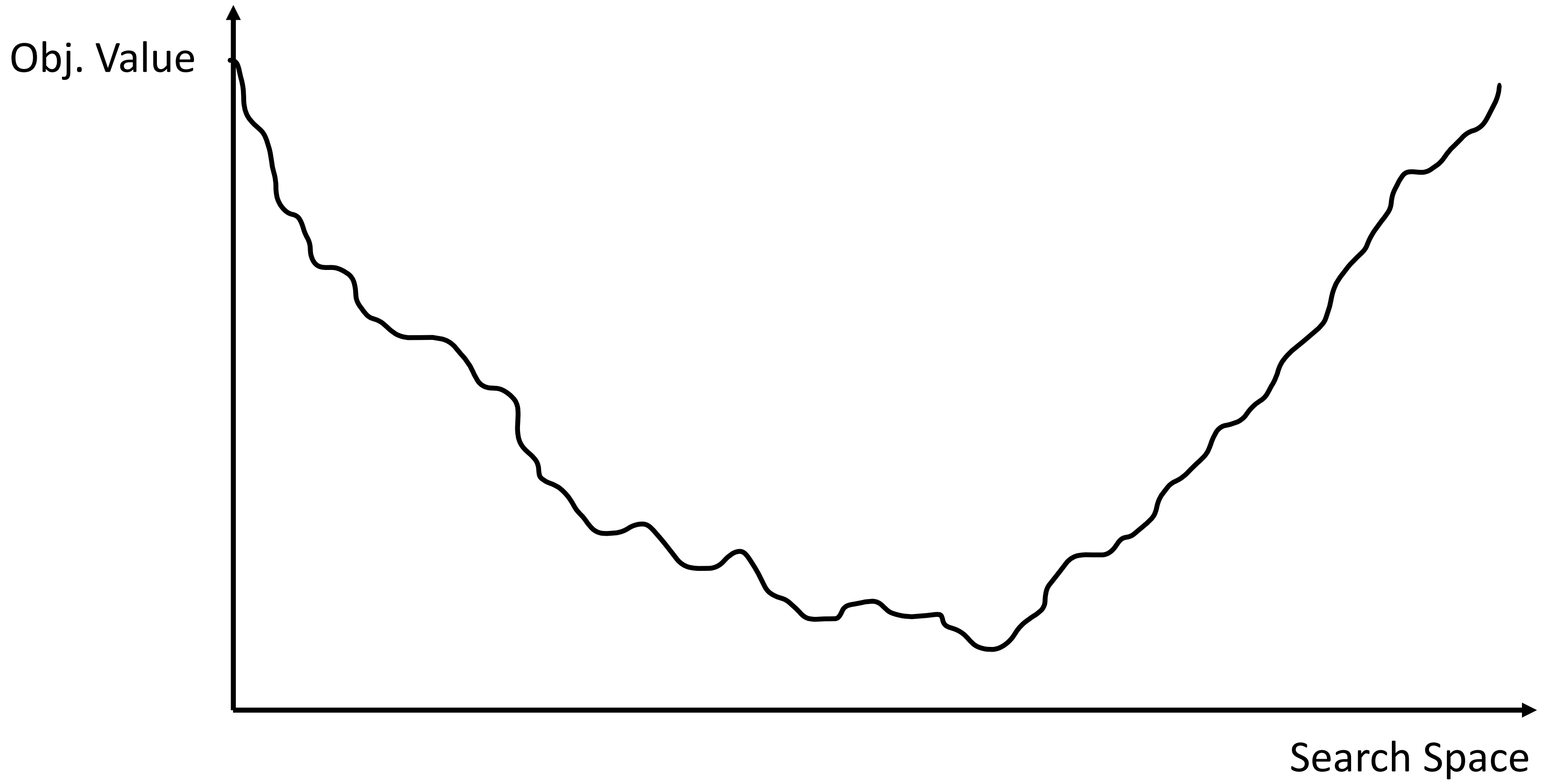
Assistant Professor

Engineering Systems and Design

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Different Problems – Different Landscapes

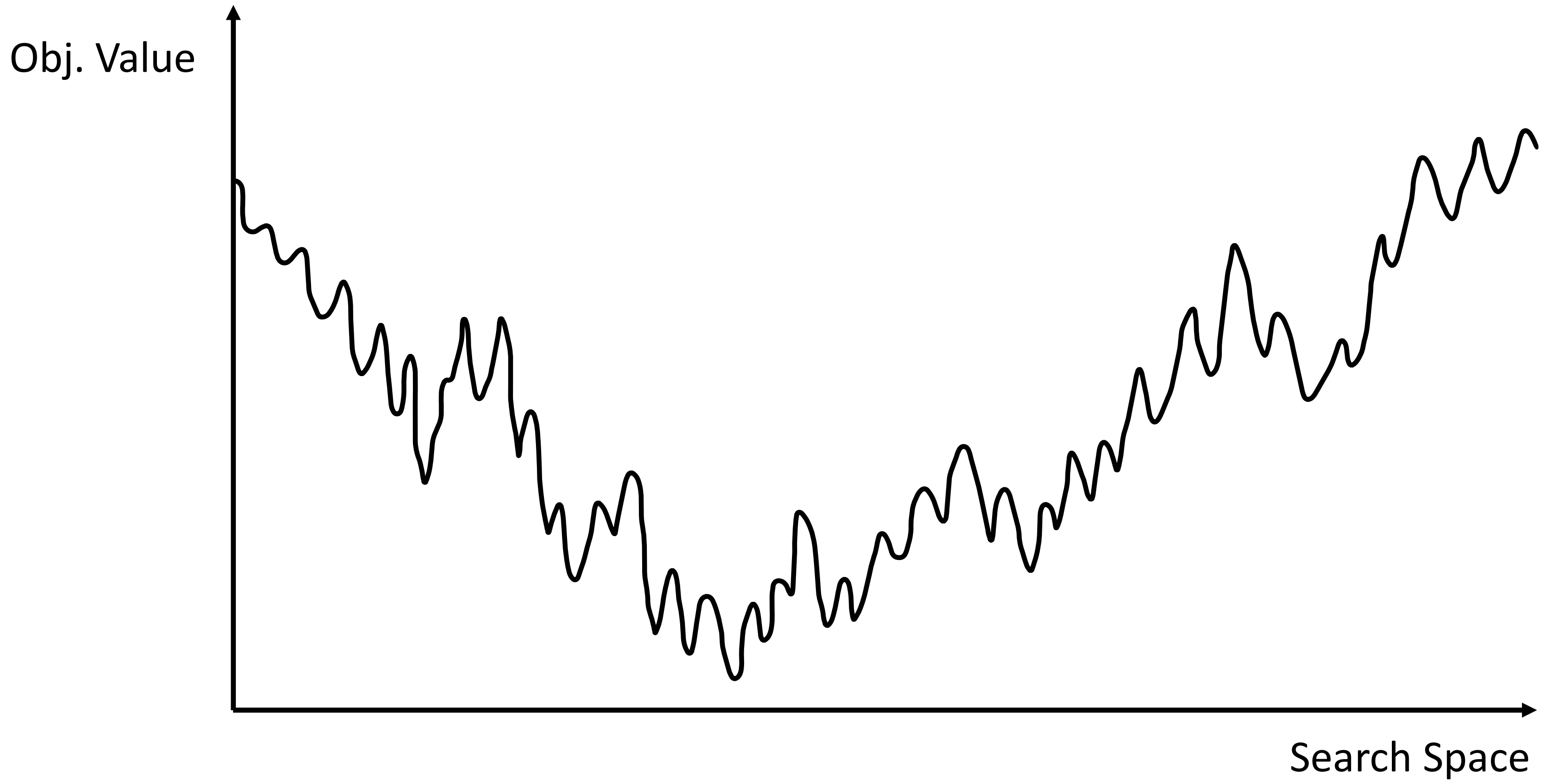
- Unimodal Landscape



Easy to solve: Hill climbing + multistart local search will be enough to find a the global optima

Different Problems – Different Landscapes

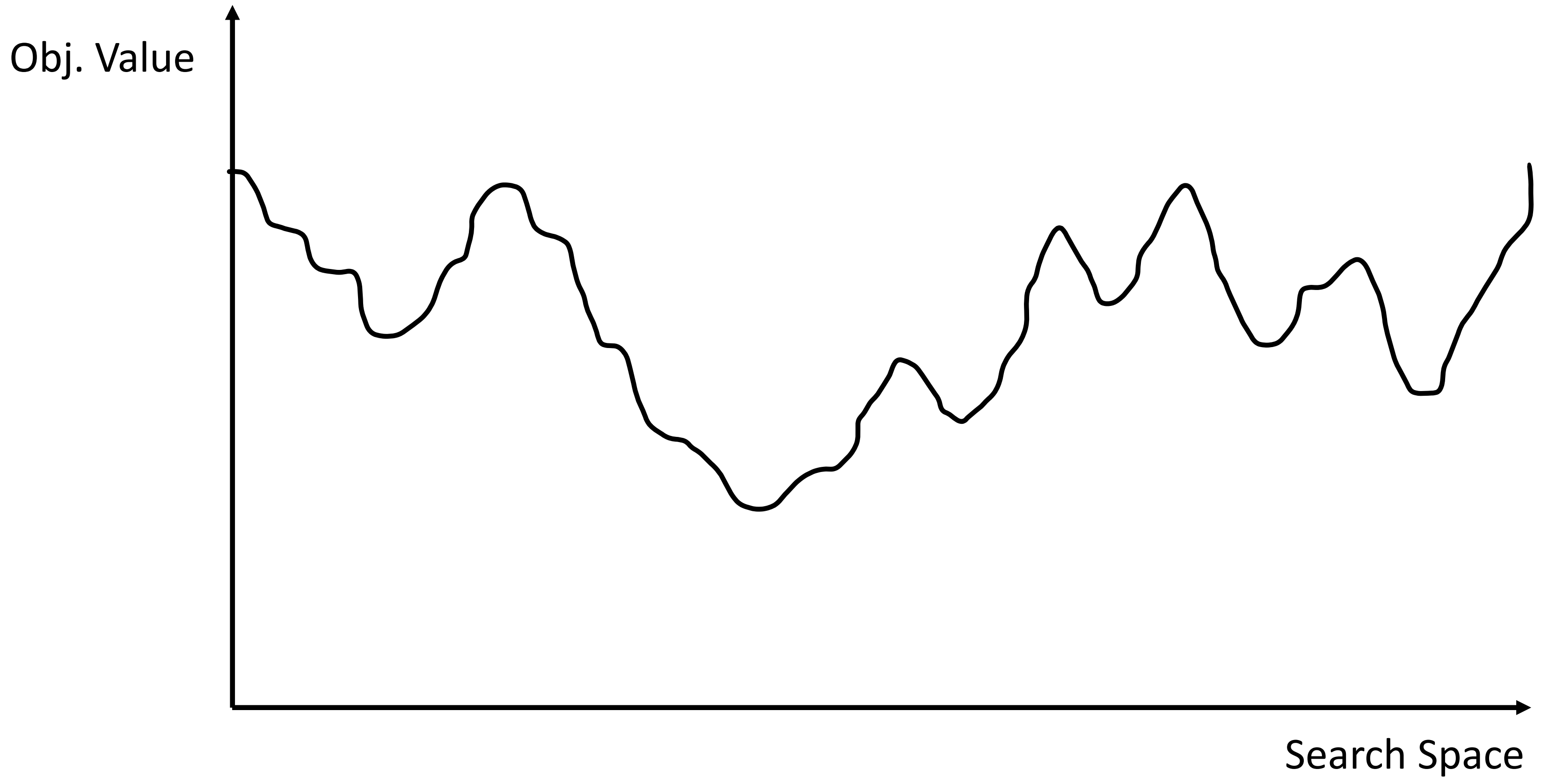
- Unimodal Rugged Landscape



Hard to solve: Simulated Annealing and Tabu Search are effective in these cases but exploration + exploitation methods need to be well calibrated.

Different Problems – Different Landscapes

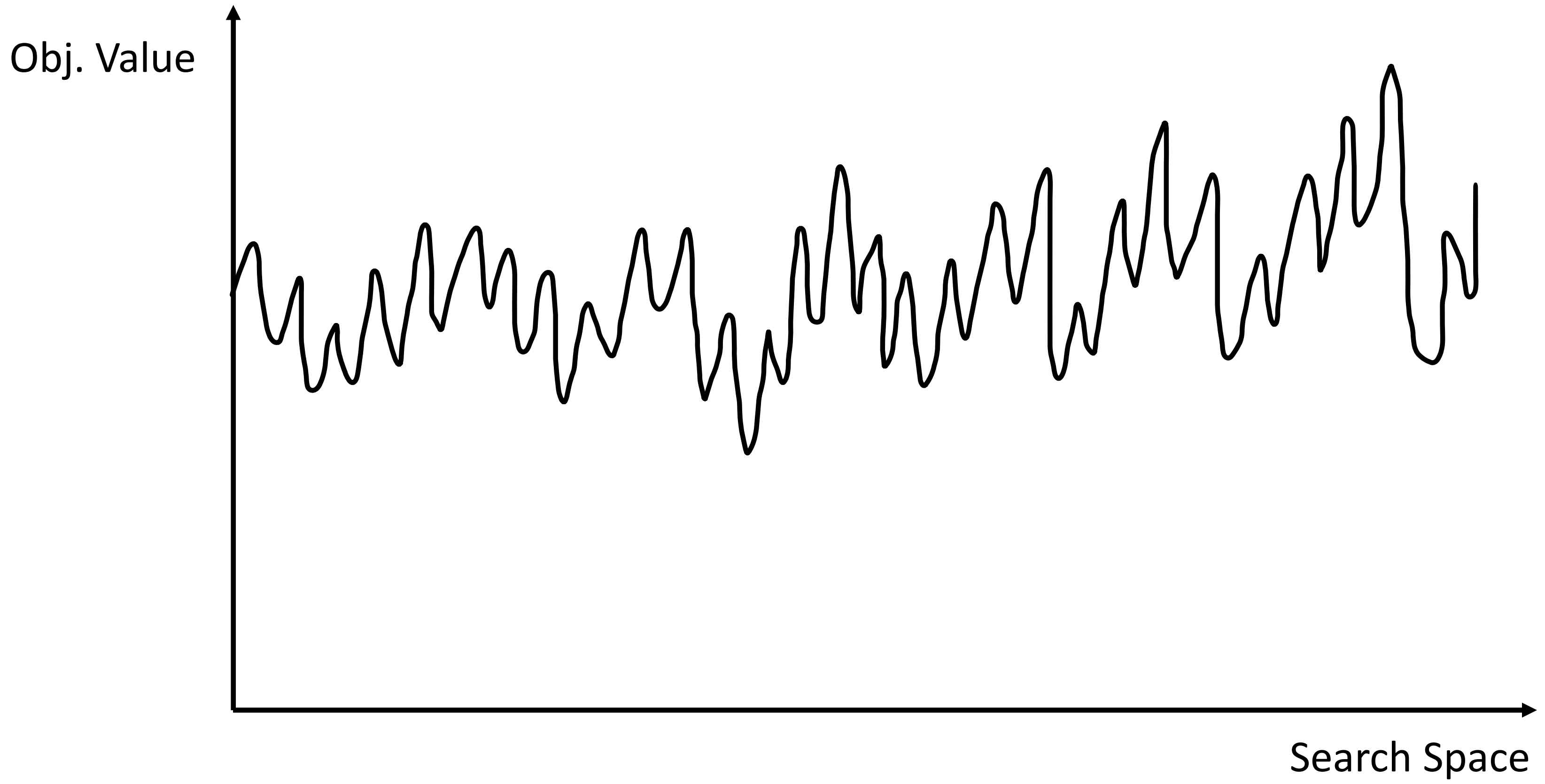
- Multimodal Landscape



Hard to solve: Simulated Annealing and Tabu Search are effective in these cases but exploration + exploitation methods need to be well calibrated.
Iterated Local Search can be useful in these cases
Population-based metaheuristics can be a good alternative to local search methods (2nd half of the course)

Different Problems – Different Landscapes

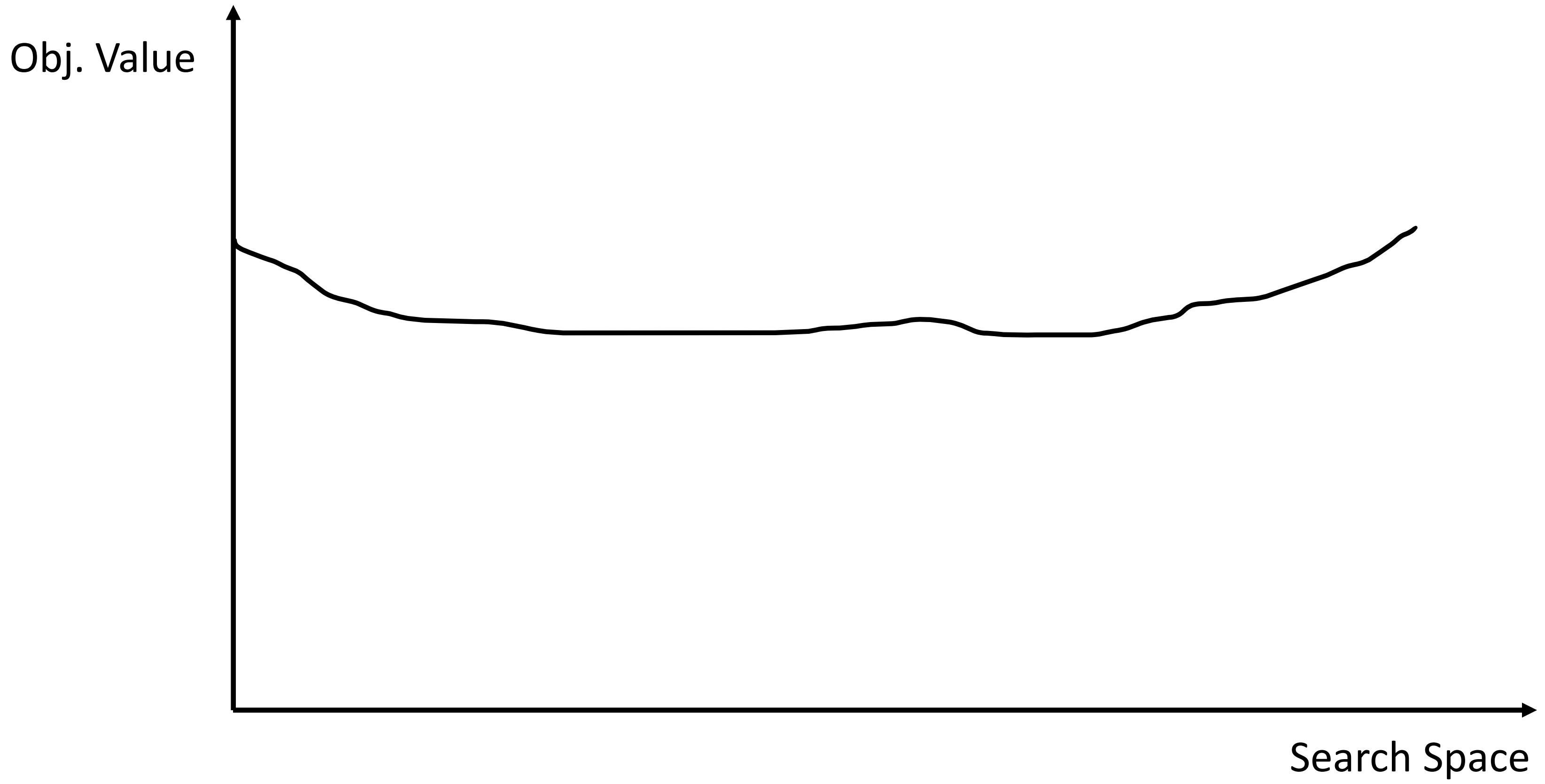
- Multimodal Rugged Landscape



Very Hard to solve: Simulated Annealing and Tabu Search may not be the most effecting metaheuristics to use. Population-based metaheuristics may be a better alternative (2nd half of the course)

Different Problems – Different Landscapes

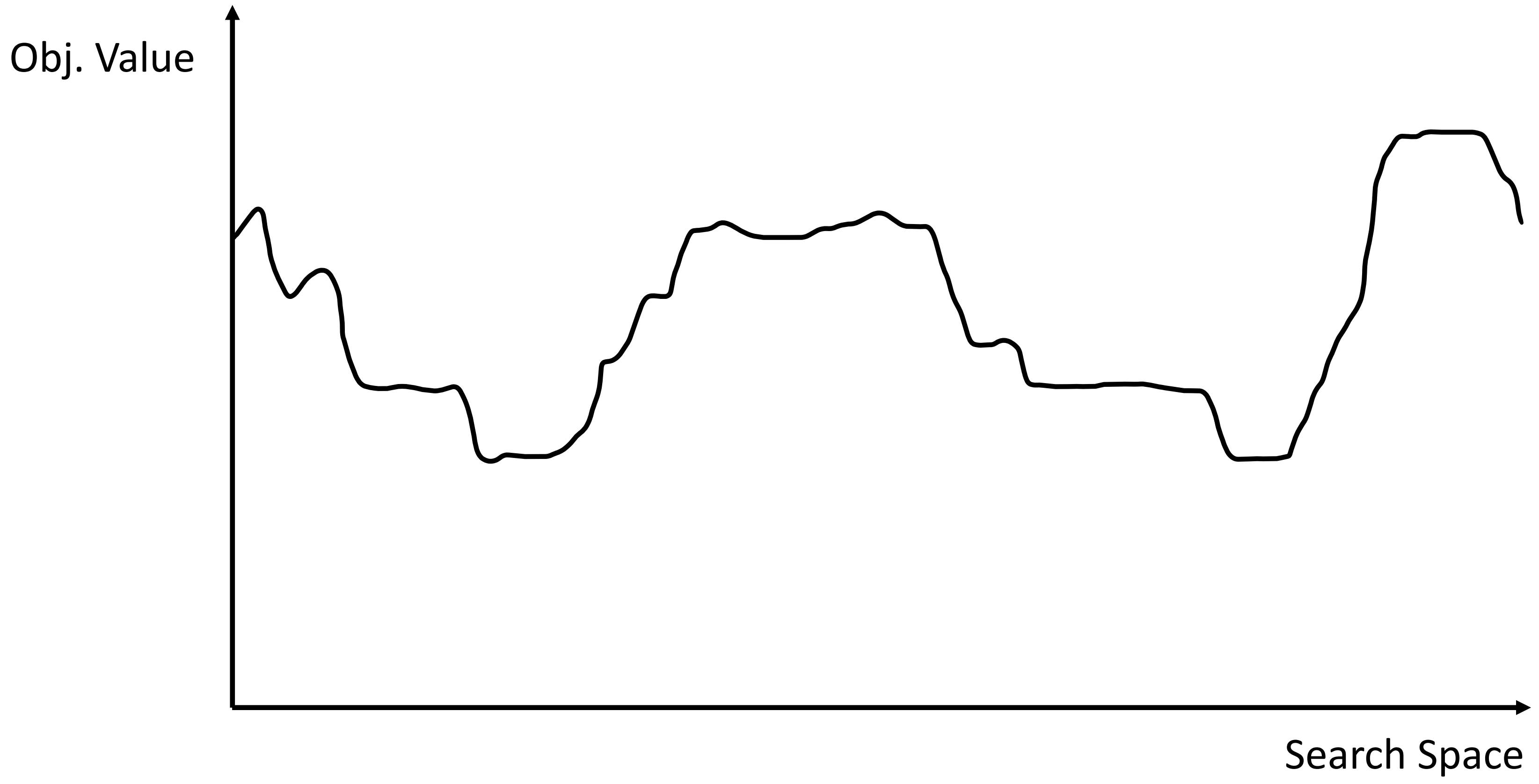
- Plain Landscape



Problem with the objective function: A flat landscape indicates that regardless of the solution obtained the objective values are all going to be similar. This means that either there is nothing to optimize and therefore the problem is not relevant ; or the objective function is not well defined.
A flat landscape has nothing to do with the metaheuristic performance

Different Problems – Different Landscapes

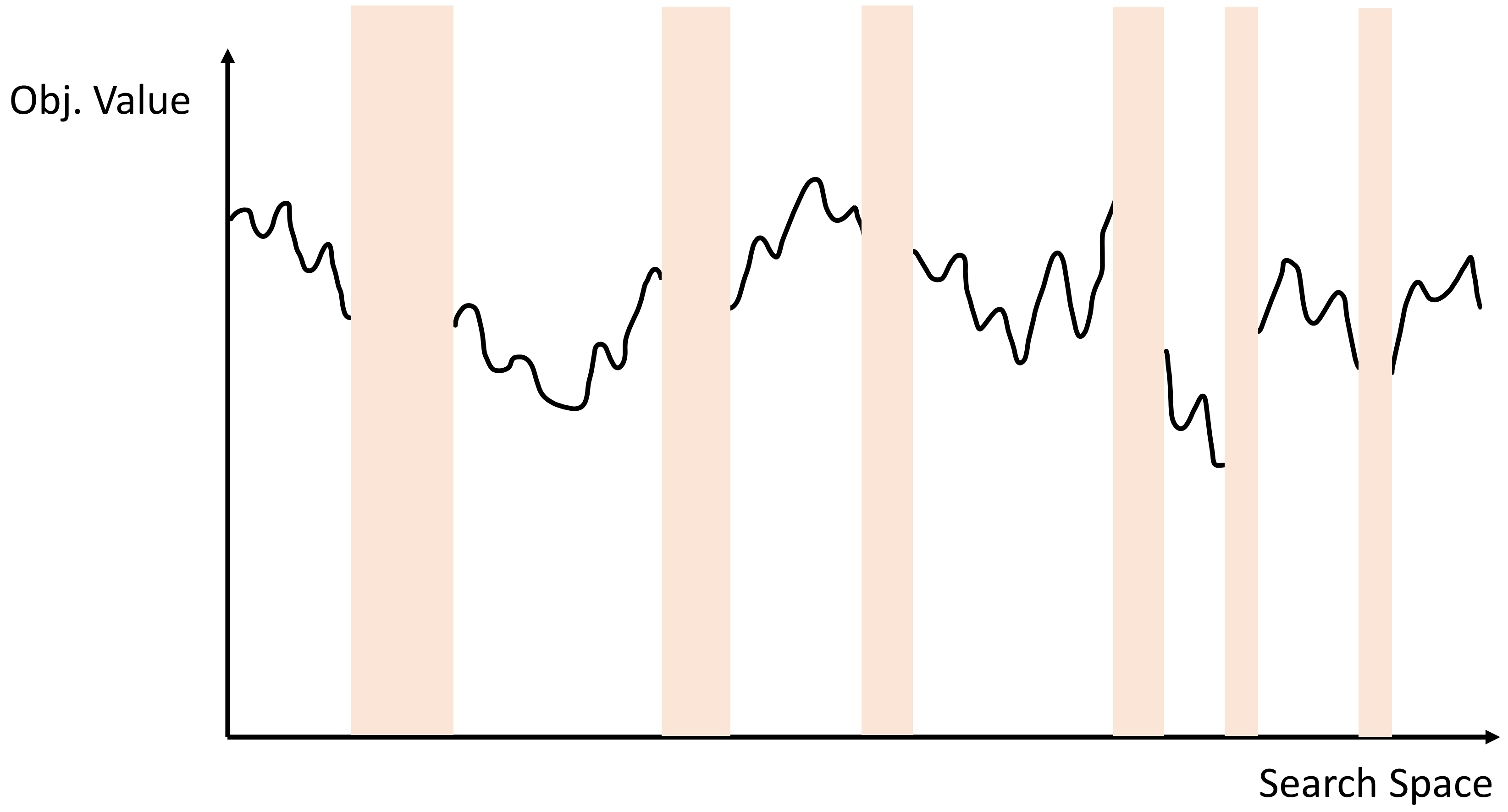
- Multimodal with plateaus



Hard to solve: Plateaus are tediously crossed by metaheuristics. Indeed, no information will guide the search toward better regions. Eventually the metaheuristic will converge to local optima solutions
To cope with these cases, larger neighbourhoods may be considered (very large neighbourhood search – next class).
Adding a secondary objective to the problem will help to shape the landscape eliminating plateaus

Different Problems – Different Landscapes

- Constrained Landscape



Depend on the type of constraints: A landscape with a small number of constraints is easy to optimize using local search methods (depending also on how rugged the landscape is). Highly constrained landscapes are harder to solve, they may require the use of constraint handling techniques, and/or larger neighbourhoods to avoid getting stuck in a constraint region

Constraint Handling Techniques

- **Reject Strategies:** 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.
- **Repairing strategies:** A **repairing procedure** is applied to infeasible solutions to generate feasible ones (e.g. extracting from the knapsack some elements to satisfy the capacity constraint in the knapsack problem)
- **Penalizing Strategies:** reject strategies do not exploit infeasible solutions. Indeed, it would be interesting to use some information on infeasible solutions to guide the search. In penalizing strategies, infeasible solutions are considered during the search process. The unconstrained objective function is extended by a **penalty function that will penalize 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

Penalizing Strategies

- The objective function *f* may be penalized in a linear manner, where *c(s)* represents the cost of the constraint violation and *λ* the weights given to infeasibilities.

$$f'(s) = f(s) + \lambda c(s)$$

- Different penalty functions may be use:
  - **Violated constraints**: A straightforward function is to **count the number of violated constraints**. No information is used on how close the solution is to the feasible region of the search space. (e.g. number of bins with capacity violated in the bin-packing problem)
  - **Amount of infeasibility**: Information on **how close a solution is to a feasible region** is taken into account (e.g. how much the capacity of a bin is exceeded in the bin-packing problem).

Comparing Optimization Algorithms

Nuno Antunes Ribeiro

Assistant Professor

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN | Engineering Systems and Design

Randomized Algorithms

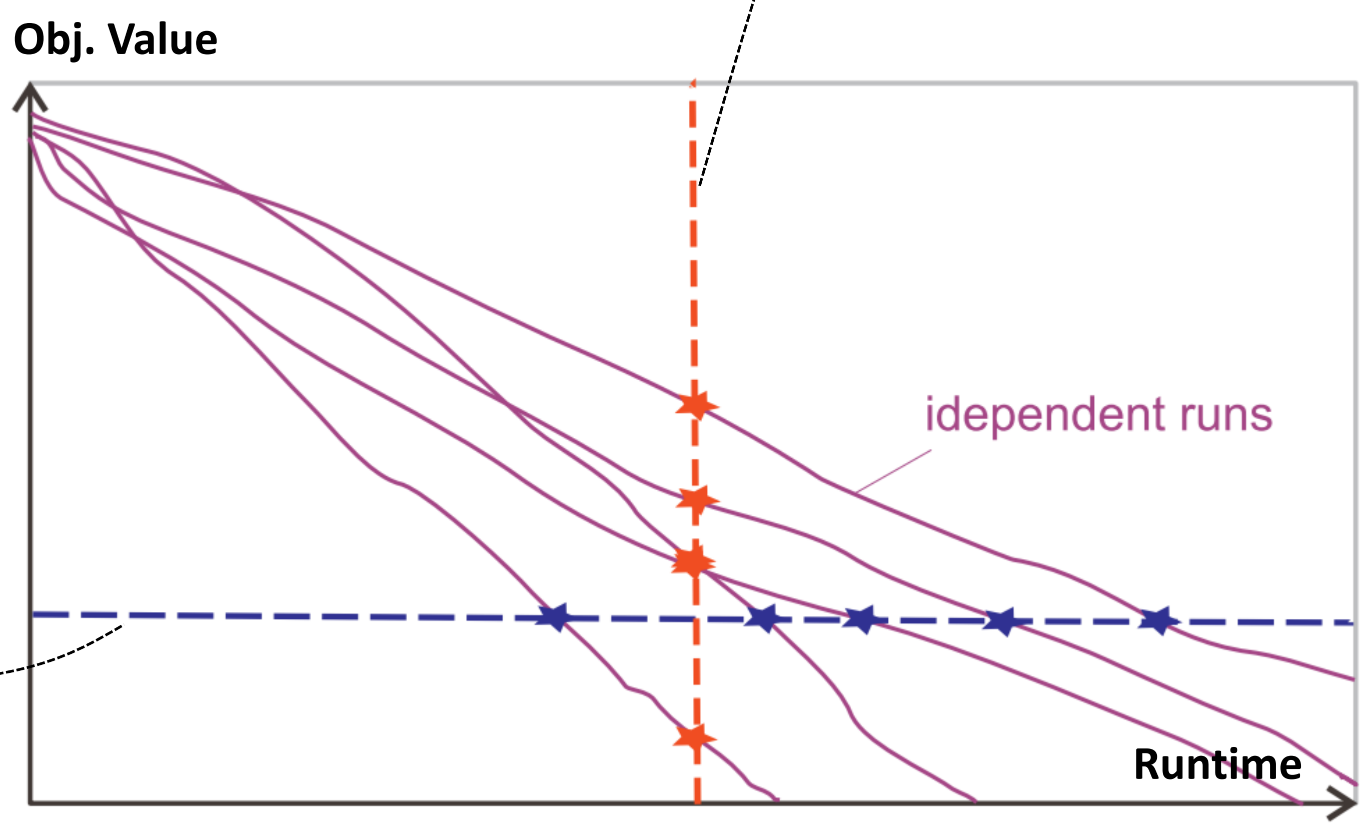
- Metaheuristics are randomized algorithms
- Performance values cannot be given absolute!
- 1 run = 1 application of an optimization algorithm to a problem, runs are independent from all prior runs
- Results can be different for each run!
- Executing algorithm one time does not give reliable information
- Statistical evaluation over a set of runs necessary

Key Parameters

No official consensus on which view is "better". This also strongly depends on the situation. Best approach: Evaluate algorithm to both methods

- Two key parameter:
  - Solution quality reached after a certain runtime
  - Runtime to reach a certain solution quality
- What is runtime?
  - **CPU Time** – easy to interpret, but machine dependent measure
  - **No. of Iterations** – machine independent measure, but no clear relation to real time (different iterations may take longer to be evaluated

Vertical Cut: Best objective value found after a certain runtime



Horizontal Cut: Runtime needed to reach certain objective value

Statistical Analysis

- Many trials must be carried out to derive significant statistical results. From this set of trials, many measures may be computed: **mean, median, minimum, maximum, standard deviation, the success rate** that the reference solution (e.g., global optimum, best known, given goal) has been attained

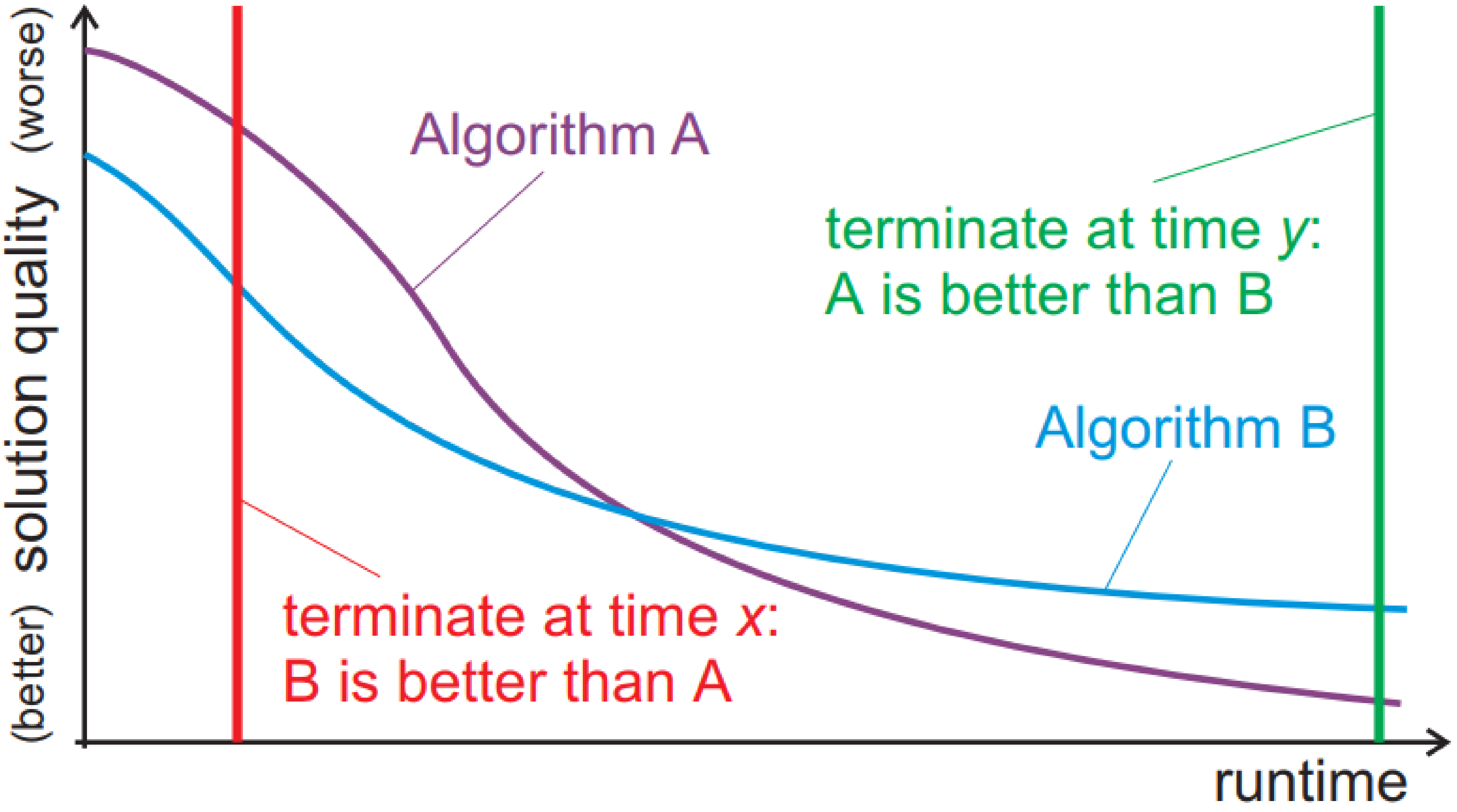
$$\text{success rate} = \frac{\text{number of successful runs}}{\text{total number of runs}}$$

- The random variable associated with the average of the results often follow a Gaussian with average m and standard deviation σ.
- **Confidence intervals** (CI) can be used to indicate the reliability of the experiments. In practice, most confidence intervals are stated at the 95% level. It represents the probability that the experimental value is located in the interval $m - 1.96\sigma/\sqrt{n}$ and $m + 1.96\sigma/\sqrt{n}$, where n is the number of evaluations .
- A result with small CI is more reliable than results with a large CI.

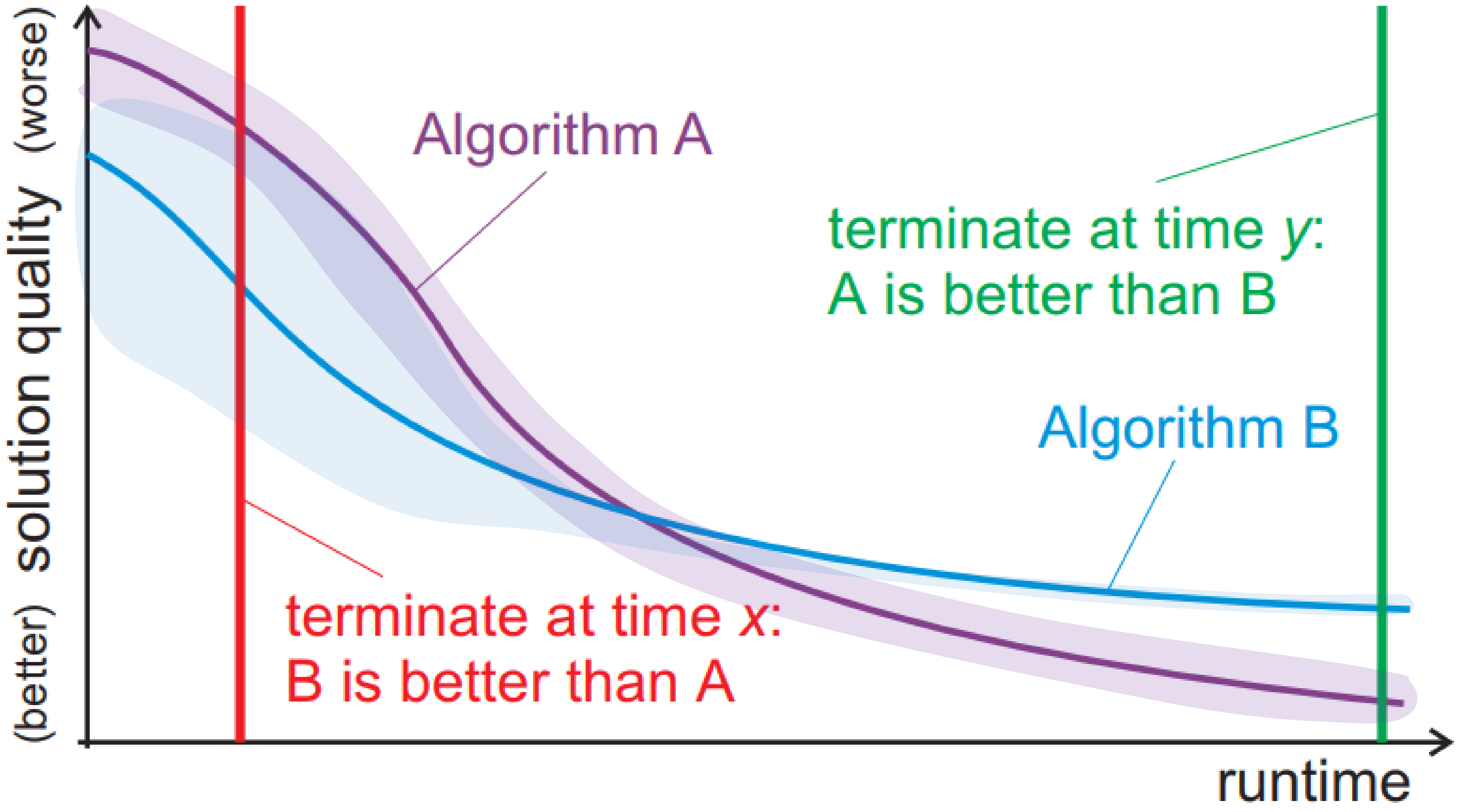
Robustness

- Robustness measures the performance of the algorithms according to different types of input instances and/or problems.
- Robustness may also be related to the average/deviation behaviour of the algorithm over different runs of the algorithm on the same instance
- The metaheuristic should be able to perform well on a large variety of instances and/or problems using the same parameters.
- The parameters of the metaheuristic may be overfitted using the training set of instances and less efficient for other instances/random seeds.

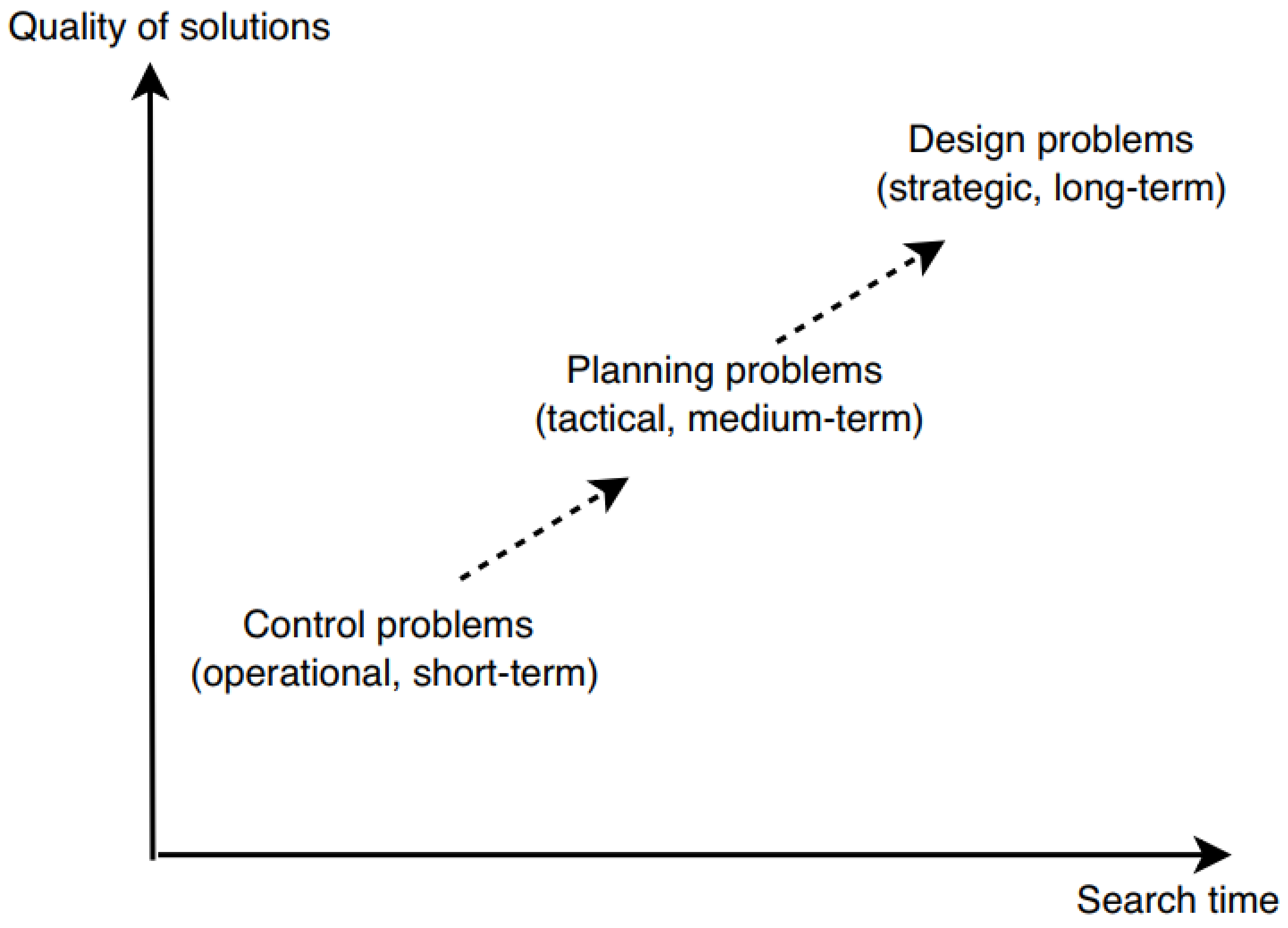
Design versus Control Problems



Design versus Control Problems



Design versus Control Problems



Quality of solutions
Design problems (strategic, long-term)
Planning problems (tactical, medium-term)
Control problems (operational, short-term)
Search time

Design versus Control Problems

- **Design problems:** Design problems are generally solved once. They need a very good quality of solutions whereas the time available to solve the problem is important. These problems involve an important financial investment; (e.g. telecommunication network design and processor design, etc.)

- **Control problems:** Control problems represent the other extreme where the problem must be solved frequently in real time. These problems require very fast heuristics are needed; the quality of the solutions is less critical (e.g. routing messages in a computer network and traffic management in a city.

- **Planning problems:** Between these extremes, one can find an intermediate class of problems represented by planning problems. In this class of problems, a trade-off between the quality of solution and the search time must be optimized; (e.g. scheduling of operations ; task assignment, etc.)