Course on Metaheuristics and Hybrid Methods for Combinatorial Optimization (Vienna 2022)

Christian Blum

Artificial Intelligence Research Institute (IIIA-CSIC)

Personal introduction

CSIC: Spanish National Research Council

- Largest public institution dedicated to research in Spain (created in 1939)
- Third-largest in Europe
- 6% of all research staff in Spain work for the CSIC
- 20% of the scientific production in Spain is from the CSIC

Artificial Intelligence Research Institute (IIIA)

- 21 tenured scientists (of three different ranks)
- Around 50 additional staff member (post-docs, PhD students, technicians, administration)
- Research lines: machine learning, optimization, logic and reasoning, multi-agent systems

Personal introduction: group

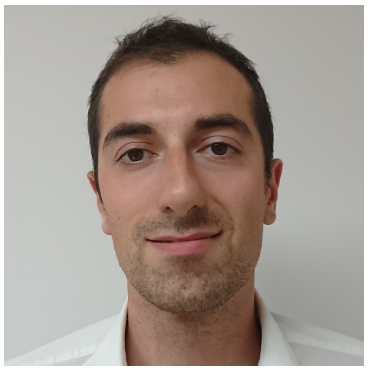
Mehmet Akbay (PhD stud.)

Camilo Chacón (PhD stud.)

Albert López (Master stud.)

Teddy Nurcahyadi (PhD stud.)

Guillem Rodríguez (Techn.)



Roberto Rosati (PhD stud.)

The two parts of the course

1. **Theory classes**: approx. 2 hours daily
   - Optimization: intruduction
   - Metaheuristics
   - Hybrid metaheuristics
   - Algorithm evaluation and comparison

2. **Practical part**: mainly on your own
   - Choice of working environment (Windows, Linux)
   - Understanding the given optimization problem
   - Developing algorithms for this problem
   - Evaluation of the implemented algorithms

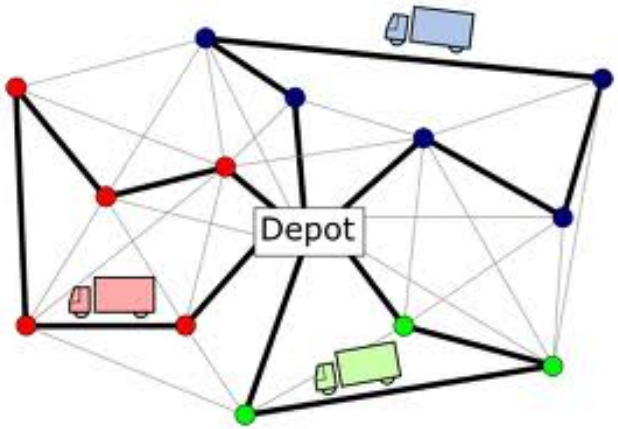
3. **Course evaluation**: practical work

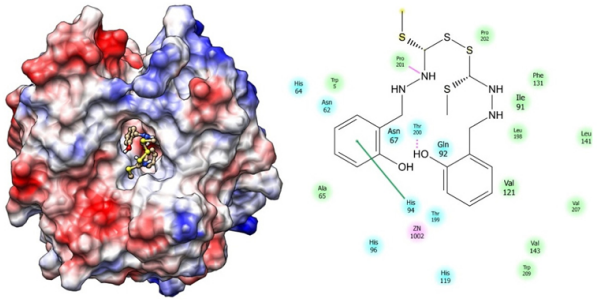
© Alex Wild

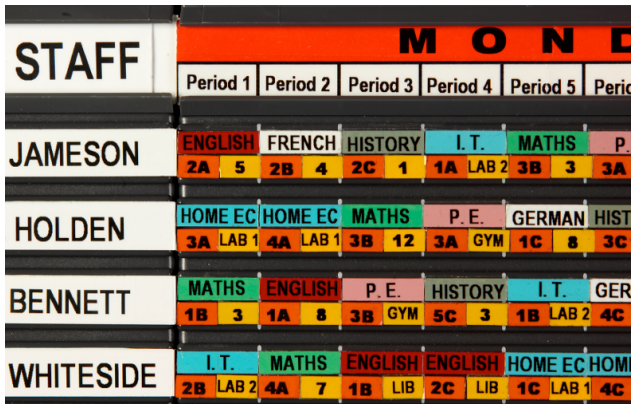
(`http://www.myrmecos.net`)

Optimization: introduction

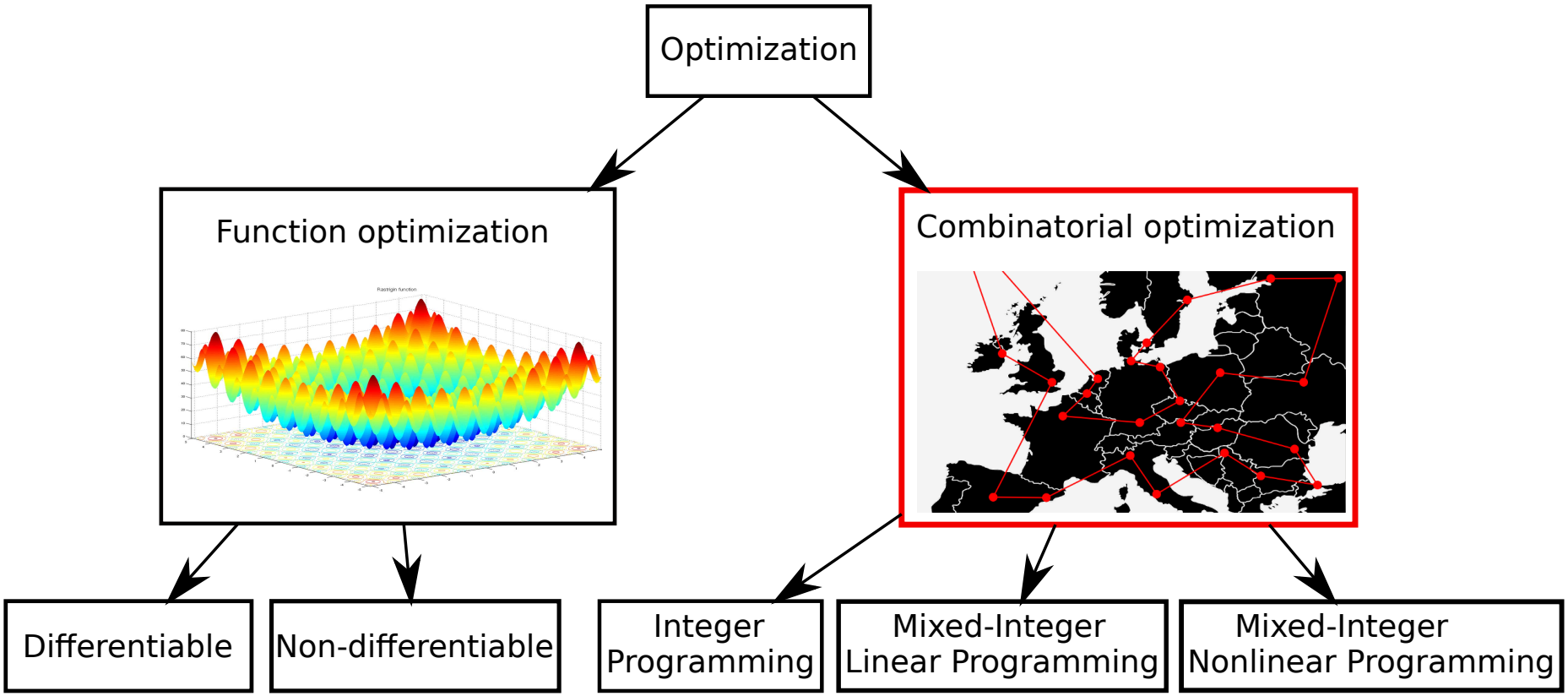
- Optimization problems arise in many industrial settings, but also, e.g., in the adacemic world.
- Optimization techniques are important tools for the industry in the context of process optimization and for reducing costs.
- Well-working optimization techniques are also crucial in many research fields.



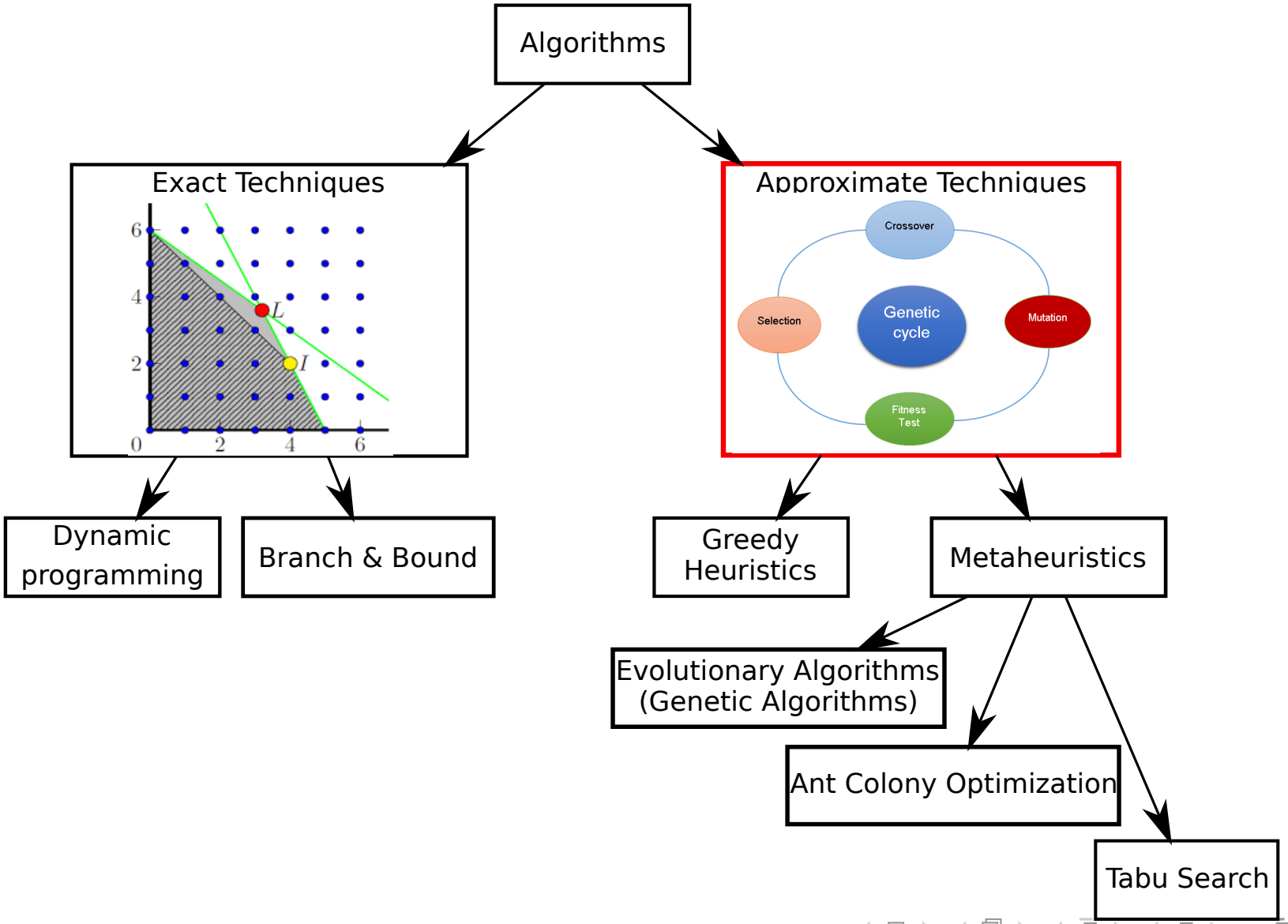




Rough taxonomy of optimization problems



Optimization techniques: overview



Optimization techniques (1)

1. **Basic methods:**
   - Greedy algorithms
   - Local search (hill climbing)
2. **Metaheuristics based on solution construction**
   - Iterated greedy (IG) algorithm
   - Greedy Randomized Adaptive Search Procedure (GRASP)
3. **Metaheuristics based on local search**
   - Tabu search (TA)
   - Simulated annealing (SA)
   - Iterated local search (ILS)
   - Variable Neighborhood Search (VNS)
   - Guided local search (GLS)

Optimization techniques (2)

3 **Bio-inspired metaheuristics**

- Evolutionary algorithms (EAs)
- Ant colony optimization (ACO)
- Particle swarm optimization (PSO)
- Artificial Bee Colony (ABC) algorithm
- Problems with bio-inspired techniques

4 **Hybrid Metaheuristics**

- Reason of being
- Popular exact methods (B&B, dynamic programming)
- Large Neighborhood Search (LNS)
- Construct, Merge, Solve & Adapt (CMSA)
- ACO based on negative learning

Practical part: optimization problem

Far From Most String (FFMS) Problem: input

1. A set $\Omega = \{s^1, \ldots, s^n\}$ of n sequences of length m over a finite alphabet Σ
2. A fixed threshold value $0 \leq t \leq m$

FFMS Problem: solutions and objective function

- Any sequence s of length m over Σ is a valid solution
- Given Ω, t, and a solution s,

$$P^s := \{s^i \in \Omega \mid d_H(s^i, s) \geq t\} \ ,$$

where $d_H(.,.)$ is the Hamming distance

- **Objective function:** Find a valid solution s s.t. $f(s) := |P^s|$ is maximal

FFMS problem: example

Example instance and solutions

- $n = 3$, $m = 4$, $\Sigma = \{0, 1\}$, $t = 3$
- $s^1 = 0101$, $s^2 = 0111$, $s^3 = 0011$
- **Possible solution:**
  - $s = 1100$
  - $P^s = \{s^2, s^3\}$
  - Objective function value: 2
- **Optimal solution:**
  - $s^* = 1000$
  - $P^{s^*} = \{s^1, s^2, s^3\}$
  - Objective function value: 3

Course evaluation: practical work

Tasks concerning the FFMS problems

1. Implementation of the ILP model for solving the FFMS problem with CPLEX
2. Development and implementation of a greedy heuristic
3. Development and implementation of a local search
4. Development and implementation of a metaheuristic
5. Development and implementation of a hybrid metaheuristic
6. Computational evaluation of the implemented techniques
7. Preparation of a report about the obtained results

Important

Choose at least 3 of the first 5 tasks. The remaining two tasks must be completed by everyone.

Practical work: choice of environment

What I offer

- Software package in C++
- Implemented for compilation with GCC under Linux
- Nevertheless: also compiles under Visual Studio in Windows
- Course material: in Google Drive folder
  `https://drive.google.com/drive/folders/1LNg-laRoYaHiywu5u2Dye6do1x7ABUW2?usp=sharing`
- Folder access: send me an email to `christian.blum@csic.es`

What you can do

- Option 1: Make use of my software package
- Option 2: Use the operation system and programming language of your choice

Tool for improving C++ skills

Important to know:

- Programming without classes
- Loops (for, while)
- Use of strings
- Standard Template Library (STL)
  1. Vectors
  2. Sets
  3. Pairs
  4. Maps

Jutge.org

The Virtual Learning Environment for Computer Programming

Tools: R package irace

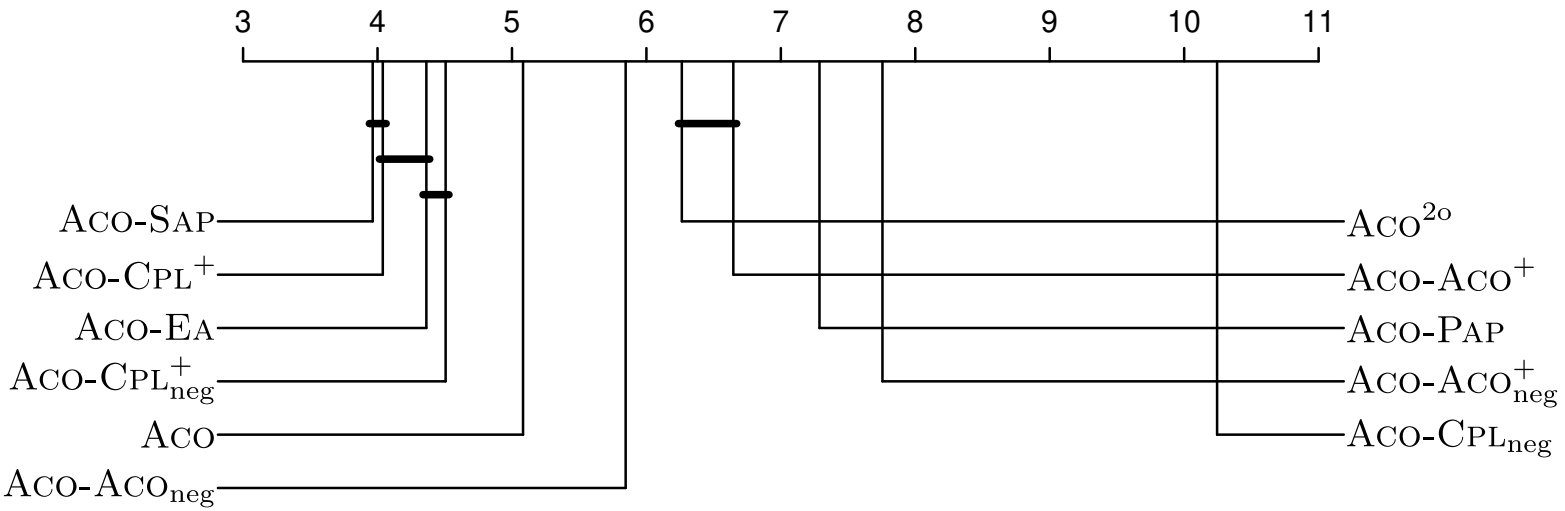
Description: irace is a R package for tuning the parameters of metaheuristics

Main scientific article about irace:

Tools: R package scmamp

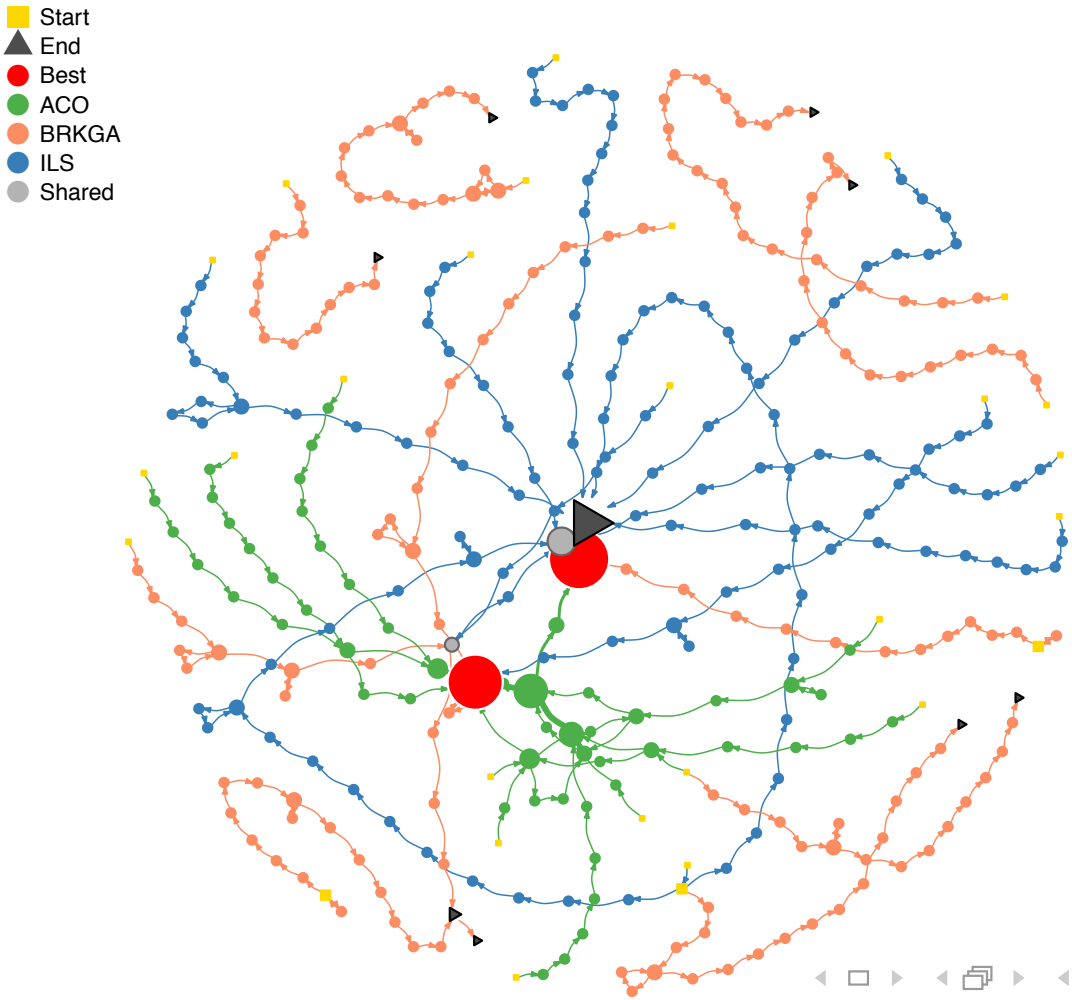
Description: scmamp is a R package for the statistical comparison of two (or more) metaheuristics.

***Criticial difference* (CD) plot:**



Tools: Search Trajectory Networks (STNs)

Description: STNs is a R package for the visual comparison of two (or more) metaheuristics



Additional literature

To be found on Google Drive:

- Gendreau, M. and Potvin, J.-Y., eds. (2010). **Handbook of Metaheuristics**. Vol. 2. Springer Verlag.
- Blum, C. and Roli, A. (2003). **Metaheuristics in combinatorial optimization: Overview and conceptual comparison**. ACM Computing Surveys (CSUR), 35(3), 268-308.
- Blum, C. and Raidl, G.R. (2016). **Hybrid Metaheuristics – Powerful Tools for Optimization**. Springer Verlag.

Questions?

Components of an optimization problem

- **Solutions:** Normally there are many
- **Restrictions:** they determine if a possible solution is valid, or invalid
- **Objectives:**
  1. There is at least one objective (which is minimized or maximized)
  2. Sometimes there are several conflicting objectives (e.g.: minimizing costs vs. maximizing security)
- **Problem data:**
  1. Do we have all the information at the start, or is there a constant stream of new information? (static vs. online/dynamic)
  2. Is all the problem information known centrally, or do problem components only have a limited view? (centralized vs. distributed)
  3. Is there uncertainty about certain problem data?

Formalizing an optimization problem

An optimization problem

An optimization problem (in the context of this course) consists of:

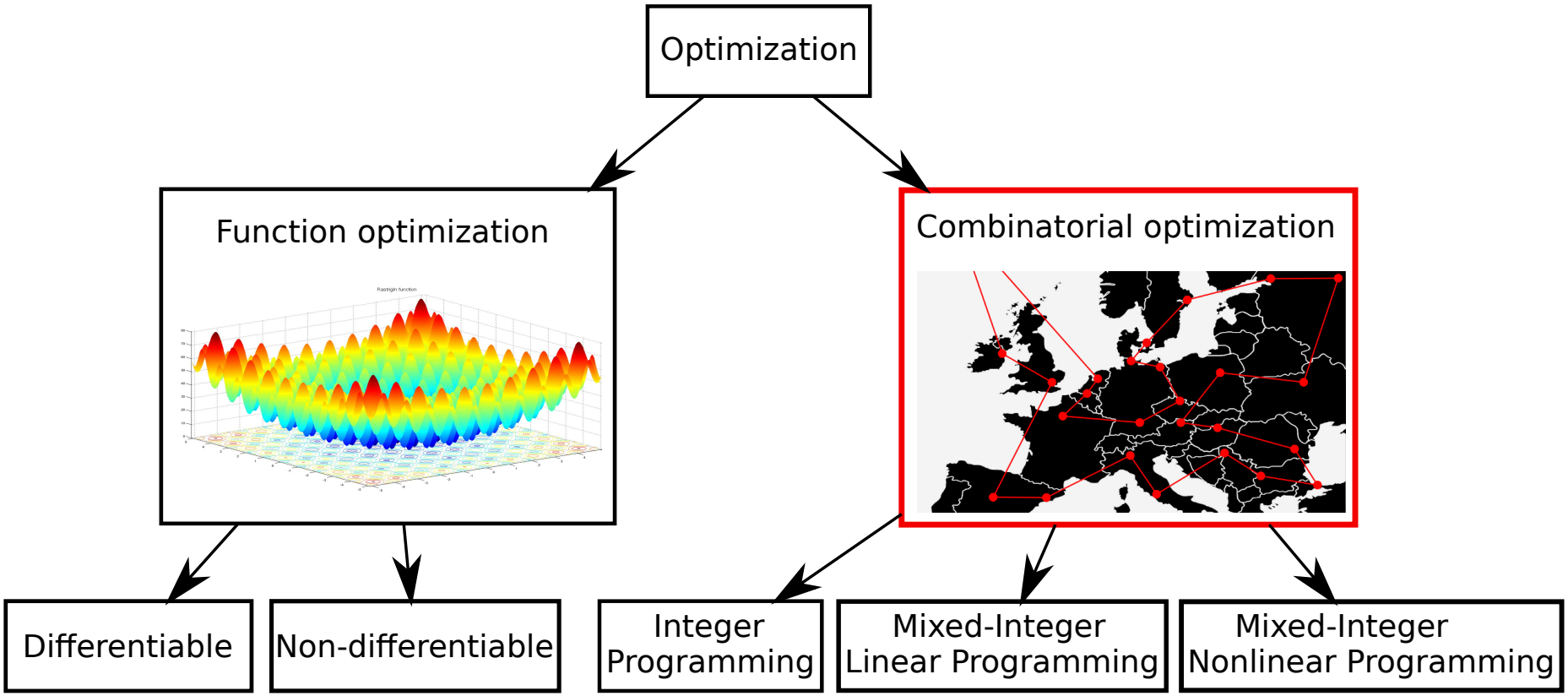
1. A set $\mathcal{S}$ of possible solutions (**search space**)
2. A function $D : \mathcal{S} \mapsto \{\texttt{valid}, \texttt{invalid}\}$ that determines whether a possible solution is valid
3. An objective function $f : \mathcal{S} \mapsto \mathbb{R}$
4. An optimization objective: minimization or maximization

Modelling an optimization problem

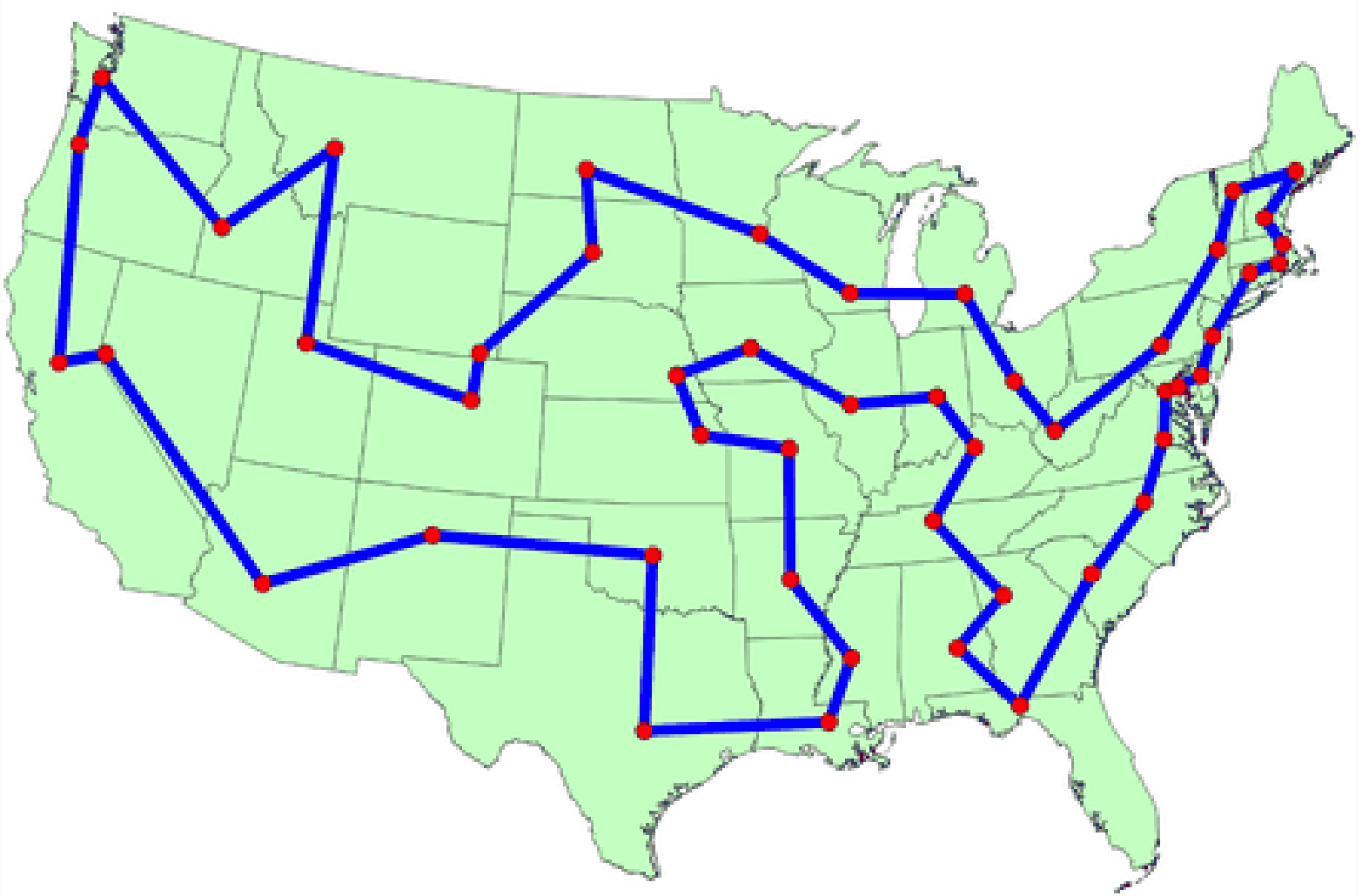
An optimization problem is modelled via

- A set of variables
- Restrictions between the variables
- An objective function

Optimization problem taxonomy



Travelling Salesman Problem (TSP)



Number of valid solutions: $\frac{(n-1)!}{2}$

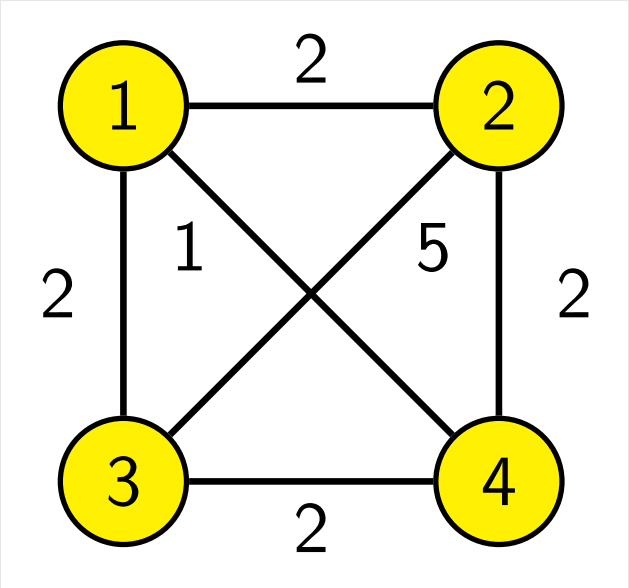
Travelling Salesman Problem (TSP)

Possible application: optimization of a drilling machine

TSP in technical terms

Lawler et al., 1985

In the **Travelling Salesman Problem (TSP)** we are given a **complete graph** $G = (V, E)$ with edge weights.



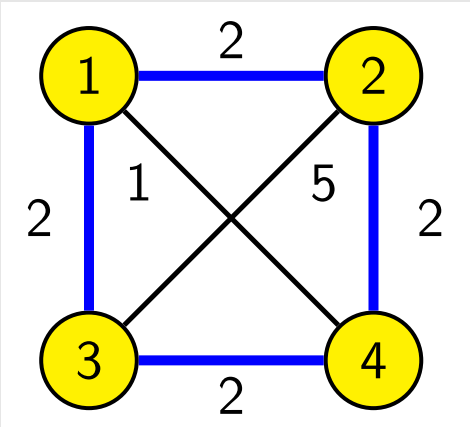
Objective: Find a Hamiltonian cycle in G whose sum of the edge weights is minimal.

Classical example: TSP

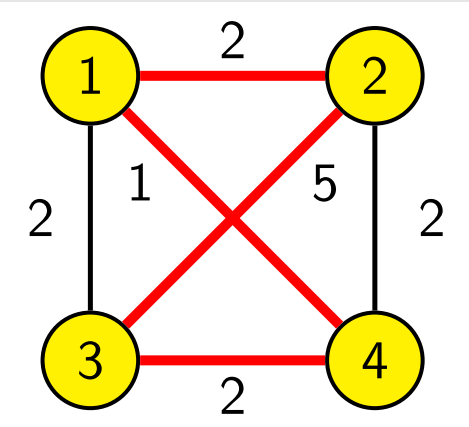
TSP expressed in terms of $\mathcal{P} = (\mathcal{S}, f)$

- The search space $\mathcal{S}$ contains all Hamiltonian cycles of G.
- The objective function $f : \mathcal{S} \mapsto \mathbb{R}^+$: $s \in \mathcal{S}$ is defined as the sum the weights of all edges of s.

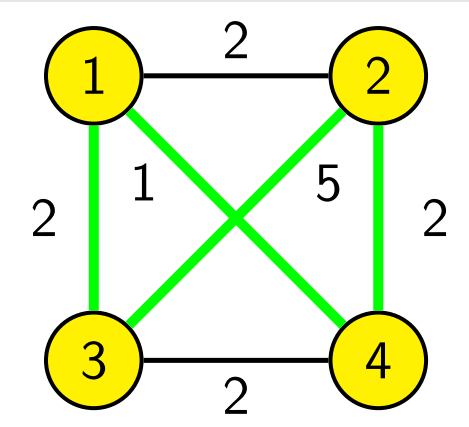
All valid solutions (search space)



value: 8



value: 10



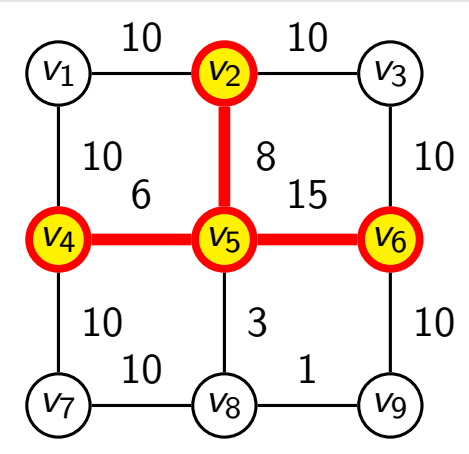
func. value: 10

Example: k-Cardinality Tree Problem

Technical definition of the KCT problem

- **Given:** An undirected graph $G = (V, E)$ with edge weights
- **Search space:** All trees in G with exactly k edges
- **Objective function:** sum of the weights of the edges of a tree

Example instance



Applications

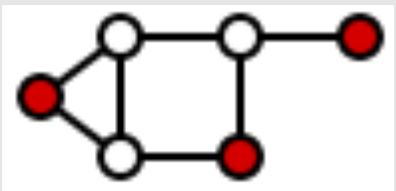
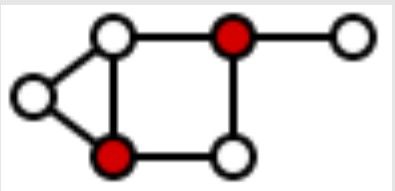
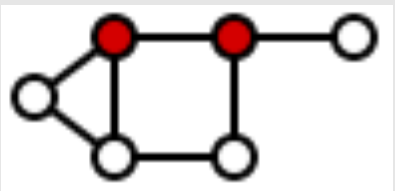
Open pit mining, telecommunication network design

Problem: Minimum Dominating Set (MDS)

Definition: MDS problem

- **Given:** An undirected graph $G = (V, E)$.
- **Search space:** Every set $S \subseteq V$ such that $N[v] \cap S \neq \emptyset$ for each $v \in V$
- **Objective function:** $F(S) := |S|$ (cardinality of set S)
- **Optimization objective:** minimization

Examples of solutions

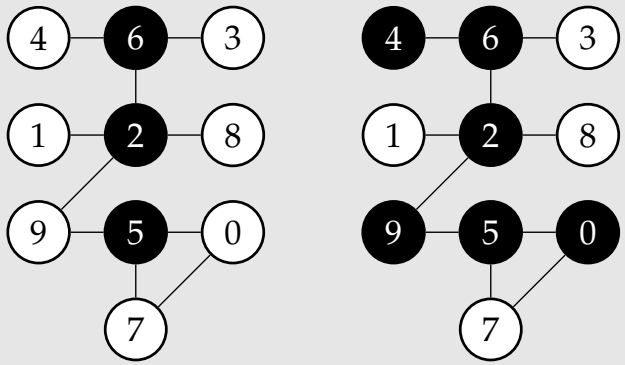




Minimum Pos. Influence Dominating Set

Definition: MPIDS problem

- **Given:** An undirected graph $G = (V, E)$.
- **Search space:** Every set $S \subseteq V$ such that at least $\left\lceil \frac{|N(v)|}{2} \right\rceil$ form part of S for each $v \in V$
- **Objective function:** $F(S) := |S|$ (cardinality of set S)
- **Optimization objective:** minimization

Examples



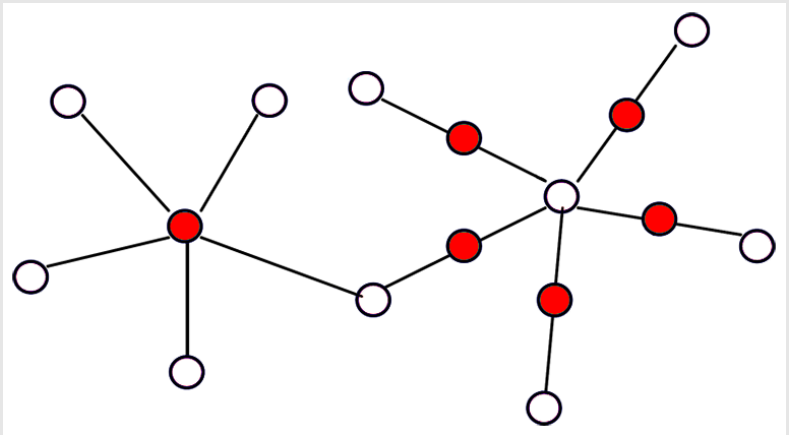
invalid solution valid solution

Problem: Min. Weight Vertex Cover

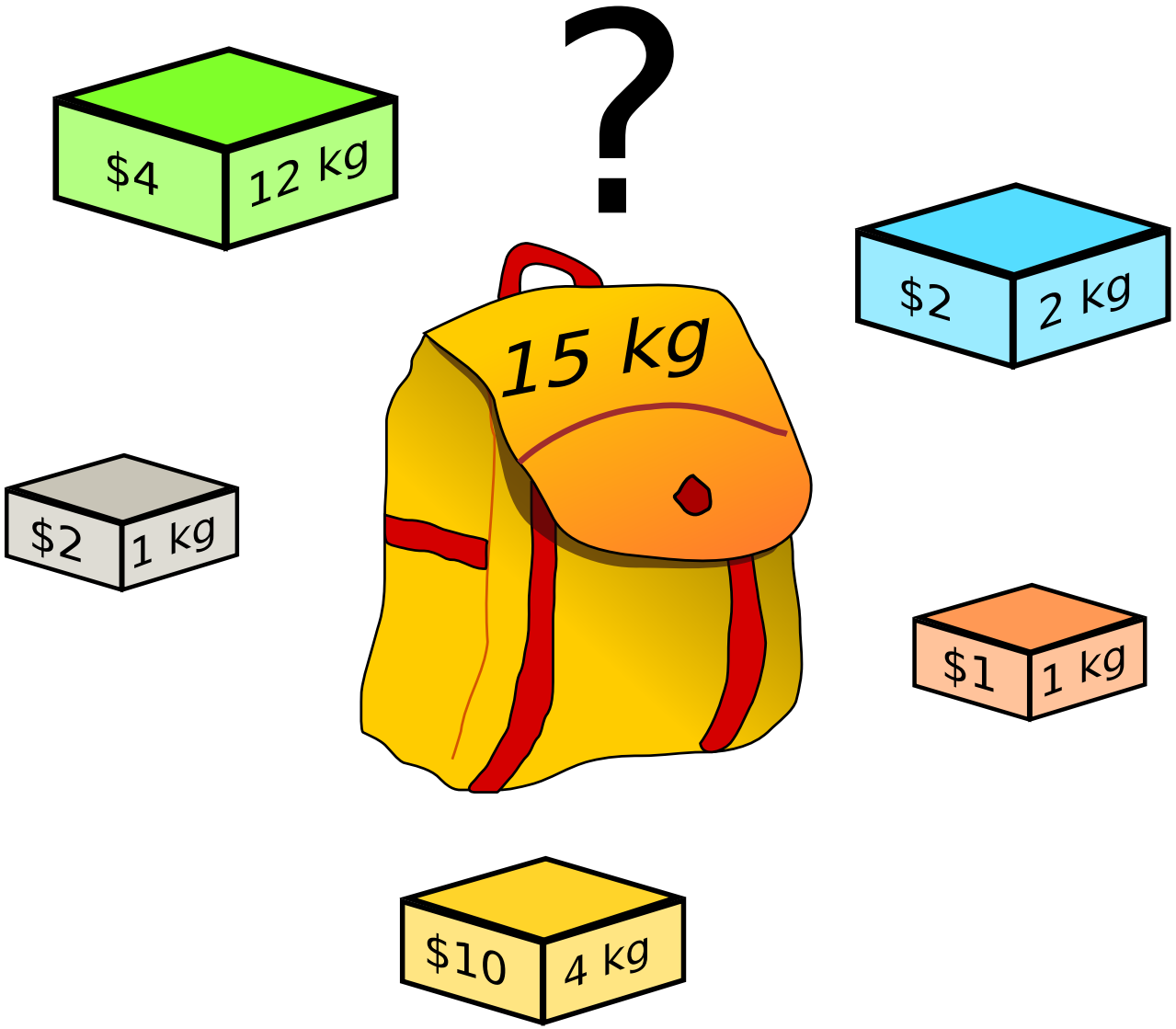
Definition: MWVC problem

- **Given:** An undirected graph $G = (V, E)$. Each $v \in V$ has a node weight $w(v) \geq 0$
- **Search space:** Every set $S \subseteq V$ such that $\{u, v\} \cap S \neq \emptyset$ for each $e = (v, u) \in E$
- **Objective function:** $F(S) := \sum_{v \in S} w(v)$
- **Optimization objective:** minimization

Example with unit weights on the nodes



Knapsack problem



Definition: Multi Dimensional Knapsack Problem (MDKP)

1. Given is a set $C=\{c_1, \ldots, c_n\}$ of n objects:
   - Each object $c_i \in C$ has a value $p_i > 0$
   - Each object $c_i \in C$ requires an amount $r_{i,k} \geq 0$ of resource $k = 1, \ldots, m$
2. Resources are limited: $\text{cap}_k > 0, k = 1, \ldots, m$
3. **Search space:** every subset S of C such that $\sum_{c_i \in S} r_{i,k} \leq \text{cap}_k$, $k = 1, \ldots, m$.
4. **Objective function:** $f(S) := \sum_{c_i \in S} p_i$ for each valid solution S
5. **Optimizatio objective** maximization

Example: Group shop scheduling (GSS)

Inspiration: organization of a parent-teacher day

	Teacher A	Teacher B	Teacher C	Teacher D
Parent 1	√	√		
Parent 2		√	√	√
Parent 3	√			√

Parents can choose the following:

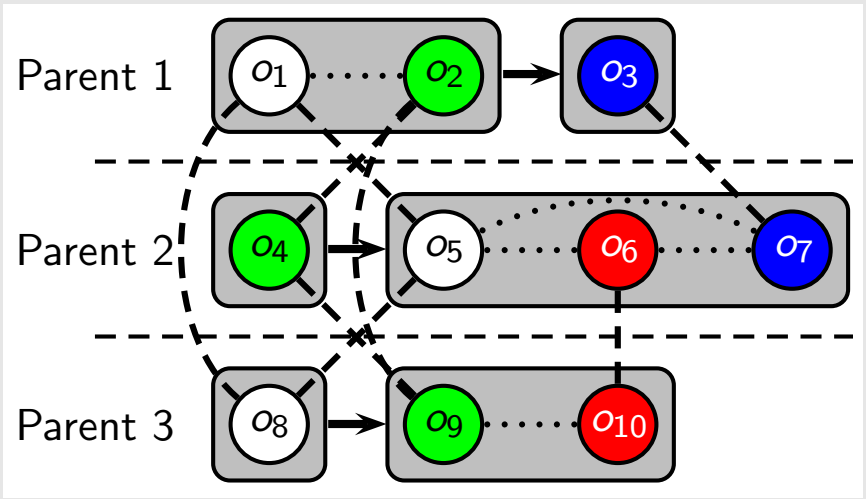
- The duration of a meeting
- The (partial) order between their meetings

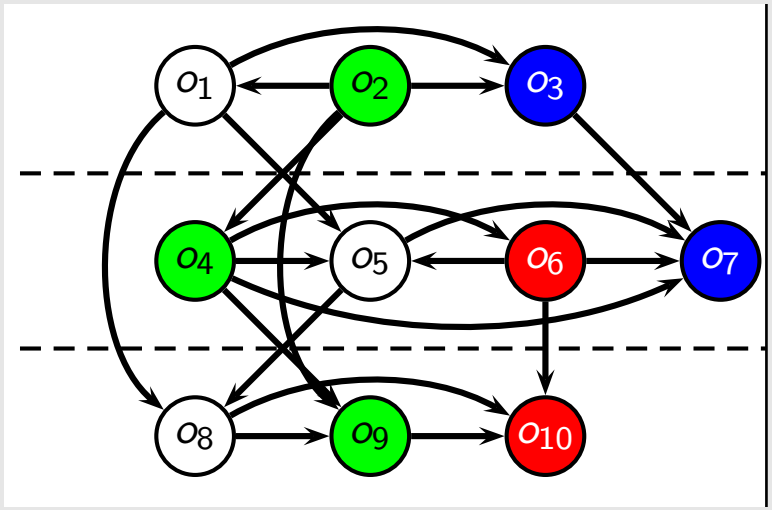
Optimization objective

Order all meetings such that the parent-teacher day finishes as soon as possible

Example: Group shop scheduling (GSS)

Graphical illustration





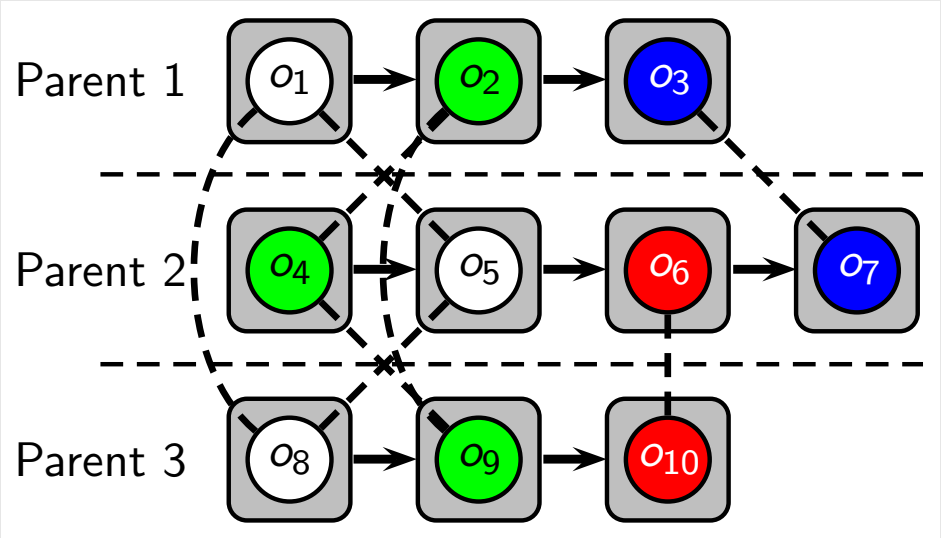
Search space: assign a direction to each un-directed edge such that the resulting graph does not contain directed cycles

Objective function: makespan (length of the longest directed path)

Other applications: industrial production environments

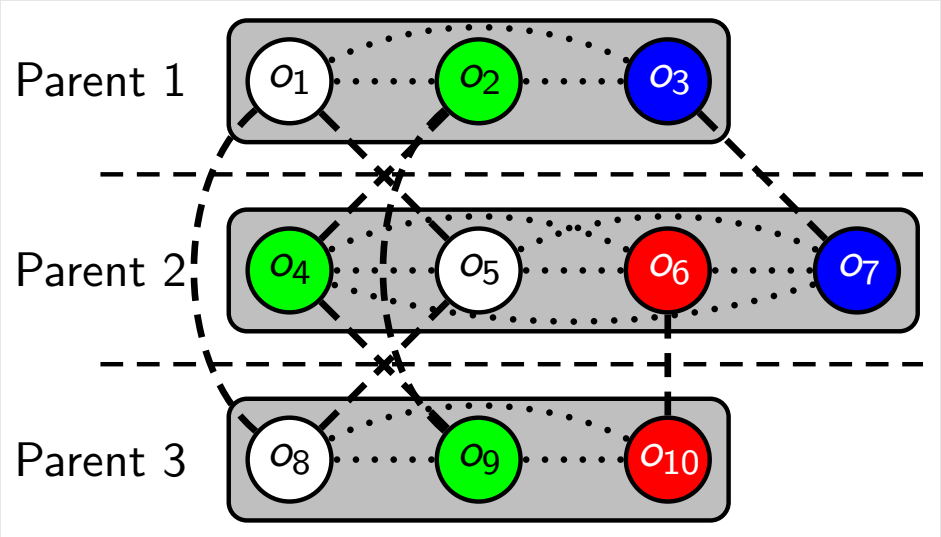
Example: Group shop scheduling (GSS)

GSS special case: Job shop scheduling (JSS)



Example: Group shop scheduling (GSS)

GSS special case: Open shop scheduling (OSS)



What is a subsequence of a string?

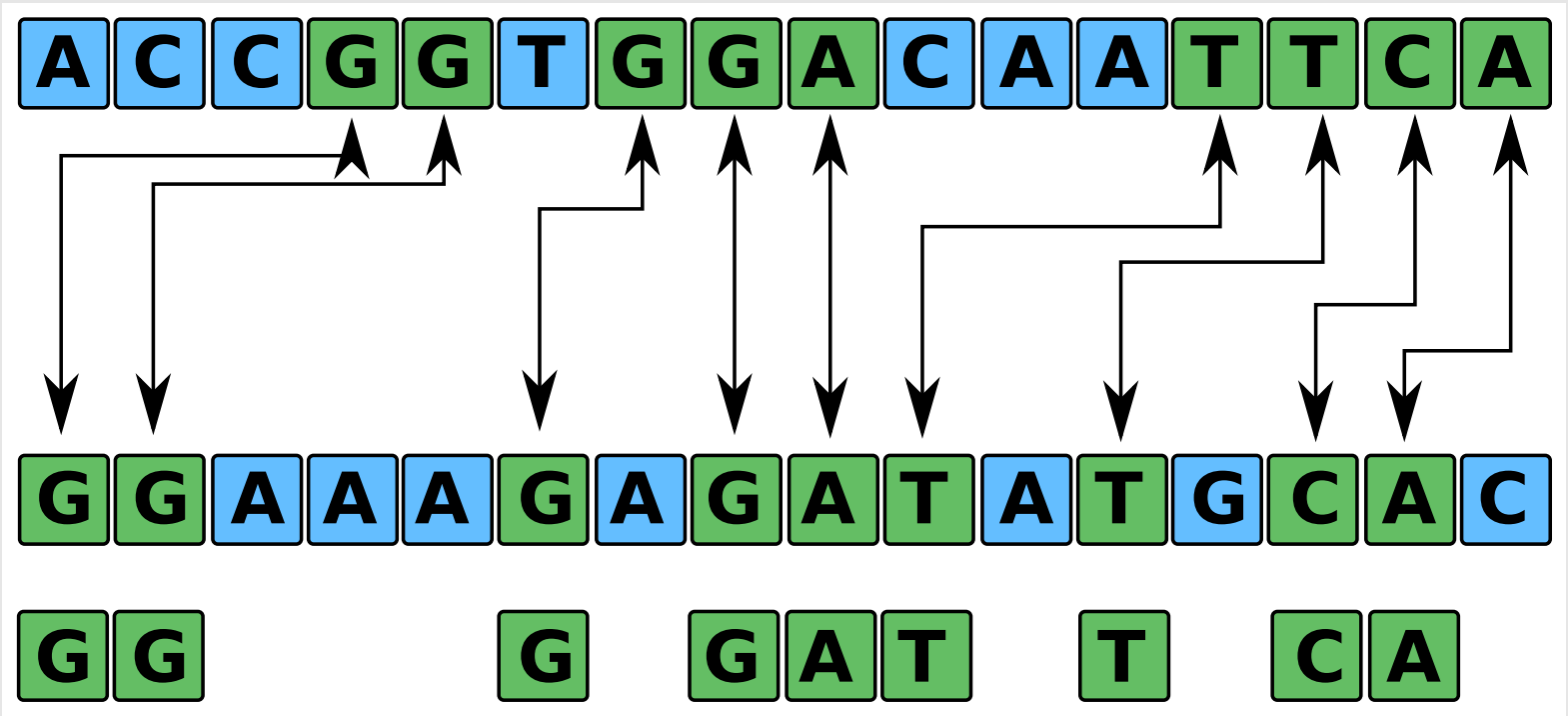
- A string t is a subsequence of a string x, if and only if t is obtained from x by deleting $c \geq 0$ characters
- **Example:** Is **AAC** a subsequence of **ACAGTC**? **YES**

Definition: longest common subsequence (LCS) problem

- **Given:** S, a set of n strings over an alphabet Σ
- **Search space:** contains each t that is a subsequence of all strings in S
- **Objective function:** $f(t) := |t|$ (length of t)
- **Optimization objective:** maximization

Example: LCS

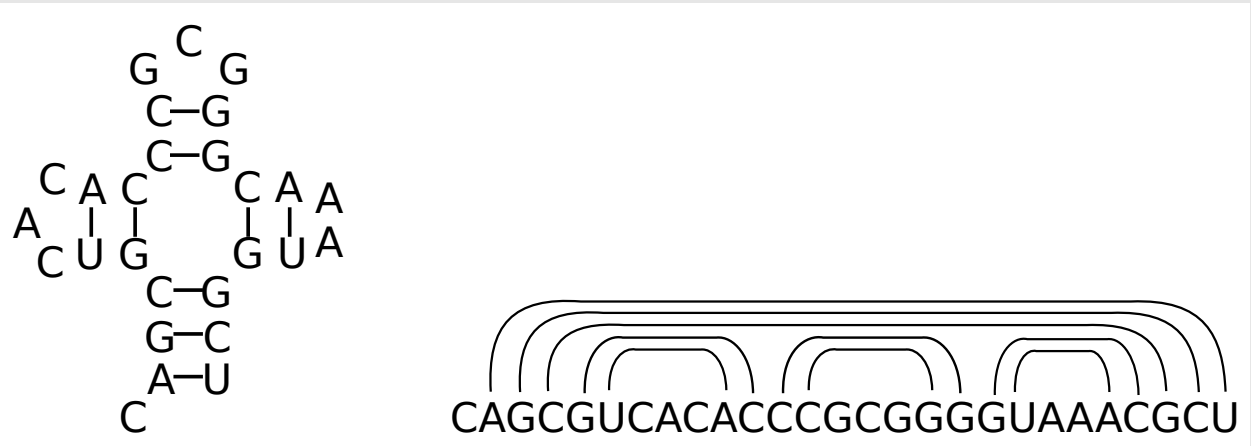
Example with 2 input strings



LCS: problem variations

LCS problem versions with additional constraints

- **Repetition-free LCS (RFLCS)**: each character of Σ may appear at most once in a solution
- **Longest arc-preserving common subsequence (LAPCS)**: solutions must respect the secondary RNA structure



Discrete optimization

Definition: discrete optimization problem

A *discrete optimization problem* $\mathcal{P} = (\mathcal{S}, \Omega, f)$ consists of the following components:

- A **set of valid solutions** $\mathcal{S}$ defined over
  - a finite set of n discrete decision variables X_i $(i = 1, \ldots, n)$;
  - a set Ω of *constraints* between the variables;
- An *objective function* $f : \mathcal{S} \rightarrow \mathbb{R}^+$ that must be minimized/maximized.

Note

If set Ω is not empty, $\mathcal{P}$ is called a constrained problem; an unconstrained problem otherwise.

Observe

Many optimization problems can be modelled as discrete optimization problems

Discrete model: MWVC

Discrete model of Minimum Weight Vertex Cover

$$\begin{aligned} \text{min} \quad & \sum_{v \in V} x_v w(v) & \\ \text{subject to} \quad & x_v + x_u \geq 1 & \forall\ e = (u,v) \in E \\ & x_v \in \{0,1\} & \forall\ v \in V \end{aligned}$$

Note

In the case of a linear objective function and linear constraints, discrete models are also called **integer linear programming (ILP)** models.

Discrete model: MDS

ILP model of Minimum Dominating Set

$$\begin{aligned} \text{min} \quad & \sum_{v \in V} x_v & \\ \text{subject to} \quad & x_v + \sum_{u \in N(v)} x_u \geq 1 & \forall\, v \in V \\ & x_v \in \{0,1\} & \forall\, v \in V \end{aligned}$$

ILP model of the TSP

$$\min \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{i,j} x_{i,j}$$

$$\text{subject to} \quad \sum_{i=1}^{j-1} x_{i,j} + \sum_{i=j+1}^{n} x_{j,i} = 2 \qquad j = 1, \ldots, n$$

$$\sum_{i,j \in L, 1 \leq i < j \leq n} x_{i,j} \leq |L| - 1 \qquad \forall\, L \subset \{1, \ldots, n\}$$

such that $3 \leq |L| \leq n - 3$

$$x_{i,j} \in \{0, 1\} \qquad \forall\, 1 \leq i < j \leq n$$

Remember: i and j are vertex indices. In this way, a variable $x_{i,j}$ represents and edge between i and j

Alternative TSP model (1)

First part of the model ...

$$\begin{aligned}
\text{min} \quad & \sum_{e \in E} d_e y_e \\
\text{subject to} \quad & x_{1,1} = 1 \\
& \sum_{k=1}^{n} x_{i,k} = 1 \qquad i = 1, \ldots, n \\
& \sum_{i=1}^{n} x_{i,k} = 1 \qquad k = 1, \ldots, n \\
& \sum_{e \in E} y_e = n
\end{aligned}$$

Alternative TSP model (2)

... continuation

$$x_{i,k-1} + x_{j,k} - y_e \leq 1 \qquad \forall\ e = (i,j) \in E, k \geq 2$$
$$x_{i,n} + x_{1,1} - y_e \leq 1 \qquad \forall\ e = (i,1) \in E$$
$$x_{i,k} \in \{0,1\} \qquad \forall\ i, k = 1, \ldots, n$$
$$y_e \in \{0,1\} \qquad \forall\ e \in E$$

Meaning of the variables: variable $x_{i,k}$ is assigned value 1 if i is the k-th visited city

Questions?

Definition: continuous optimization problem

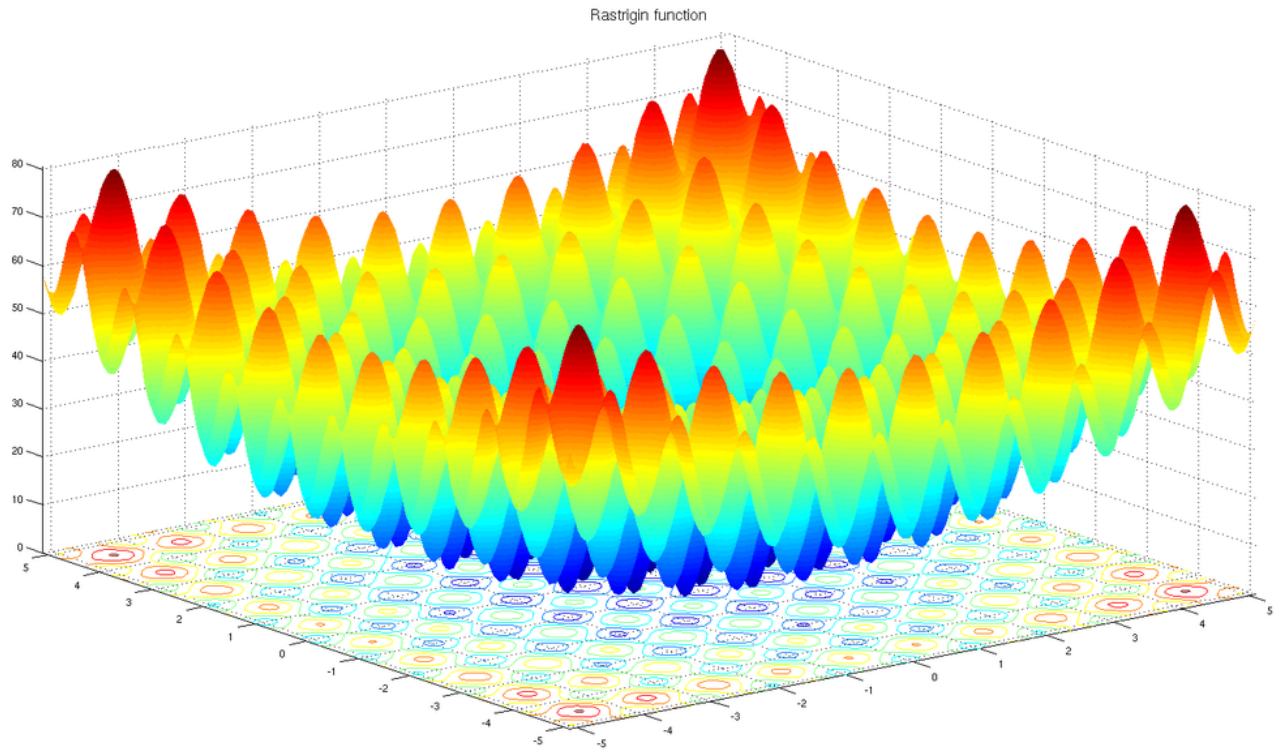
A *continuous optimization problem* $\mathcal{P} = (\mathcal{S}, \Omega, f)$ consists of the following components:

- A **set of valid solutions** $\mathcal{S} \subseteq \mathbb{R}^n$ defined over
  - a finite set of n continuous decision variables X_i $(i = 1, \ldots, n)$;
  - a set Ω of *constraints* among the variables;
- An *objective function* $f : \mathcal{S} \to \mathbb{R}$ that must be minimized/maximized.

Note

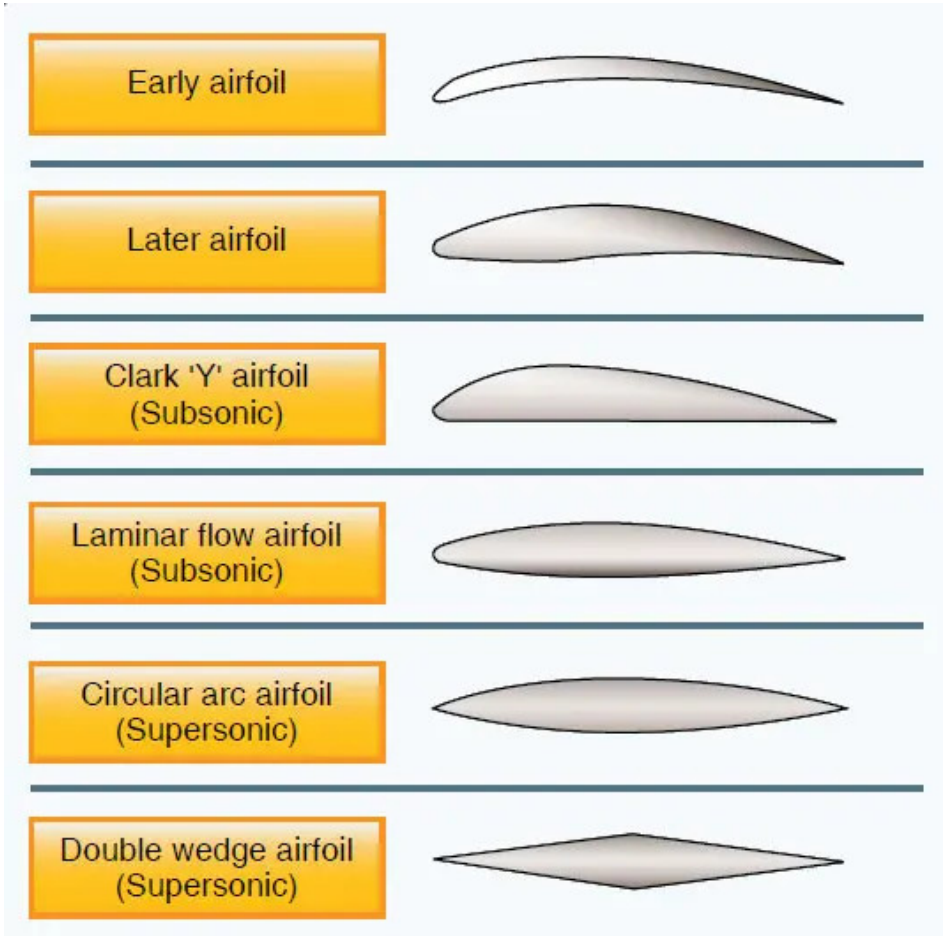
If Ω is empty, $\mathcal{P}$ is called a constrained problem, an unconstrained problem otherwise.

Example: Analytical Functions

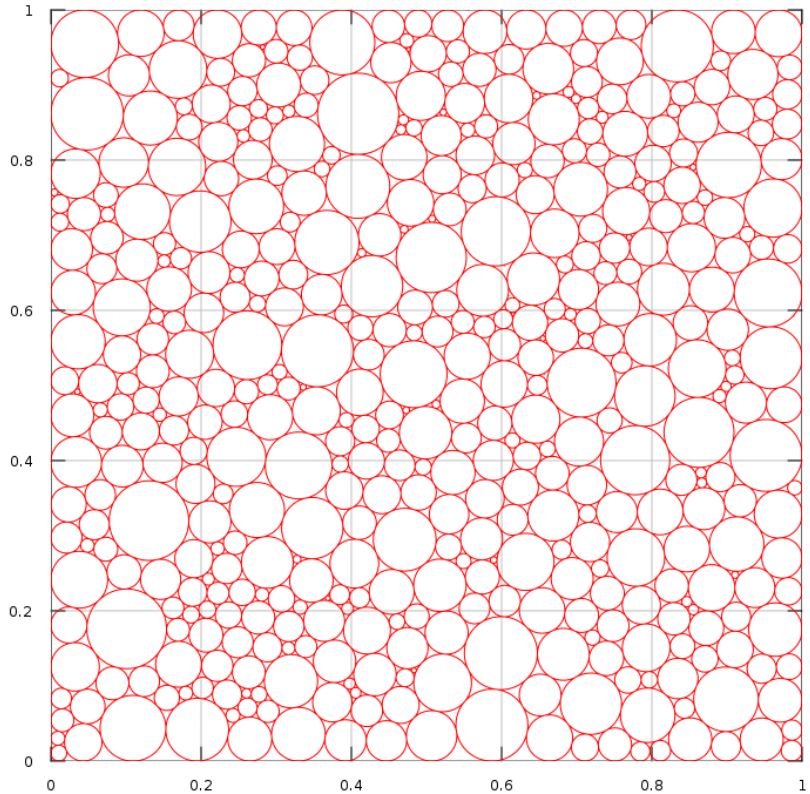


- **Function expression:** $f(x_1, x_2) = 20 + \sum_{i=1}^{2}(x_i^2 - 10\cos(2\pi x_i))$, $x_i \in [-5.12, 5.12]$
- **Optimization objective:** minimization

Example: Aerodynamic Design

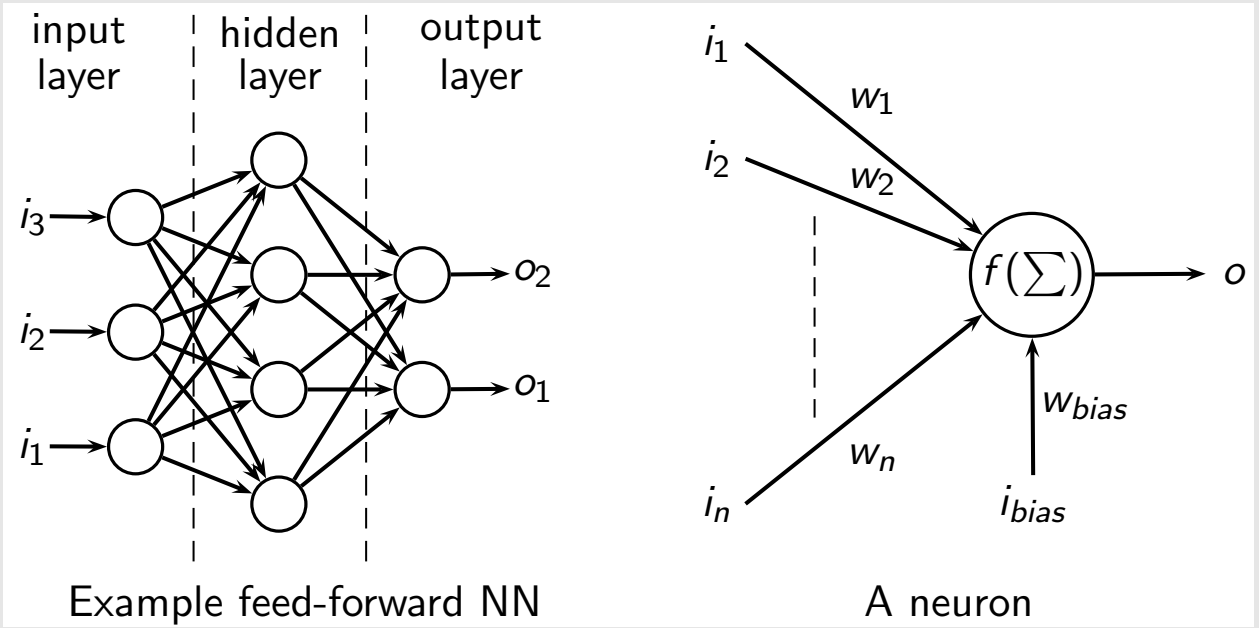


Example: Packing Problems



Example: Neural Network Training

Feed-forward neural network



Example: Neural Network Training

Pattern Classification

- Given is a set S of pattern
- Each pattern p consists of a number of n measurements
- Each pattern p belongs to a class c from a finite set C of classes

Objective

Generate a classifier that classifies the pattern correctly (and that generalizes to unknown pattern)

Solving optimization problems

We need an algorithm!!!

General distinction:

1. **Complete/exact methods.**

   They guarantee to find for every finite size problem instance an optimal solution in bounded time.
2. **Approximate methods.**

   No guarantee of finding an optimal solution.

Question

Why not always using a complete/exact method?

Observation

Many relevant combinatorial optimization problems are *NP*-hard

Problem Complexity

There are different classes of problems

- *P*: A complete method exists that requires a **polynomial execution time**.

  Example: Minimum Spanning Tree (MST) problem

- *NP*-hard: **No polynomial time algorithm exists**

  Example: Traveling Salesman Problem (TSP)

Implication

- Complete methods might need exponential computation time in the worst-case.
- This often leads to computation times too high for practical purposes

Problem Complexity

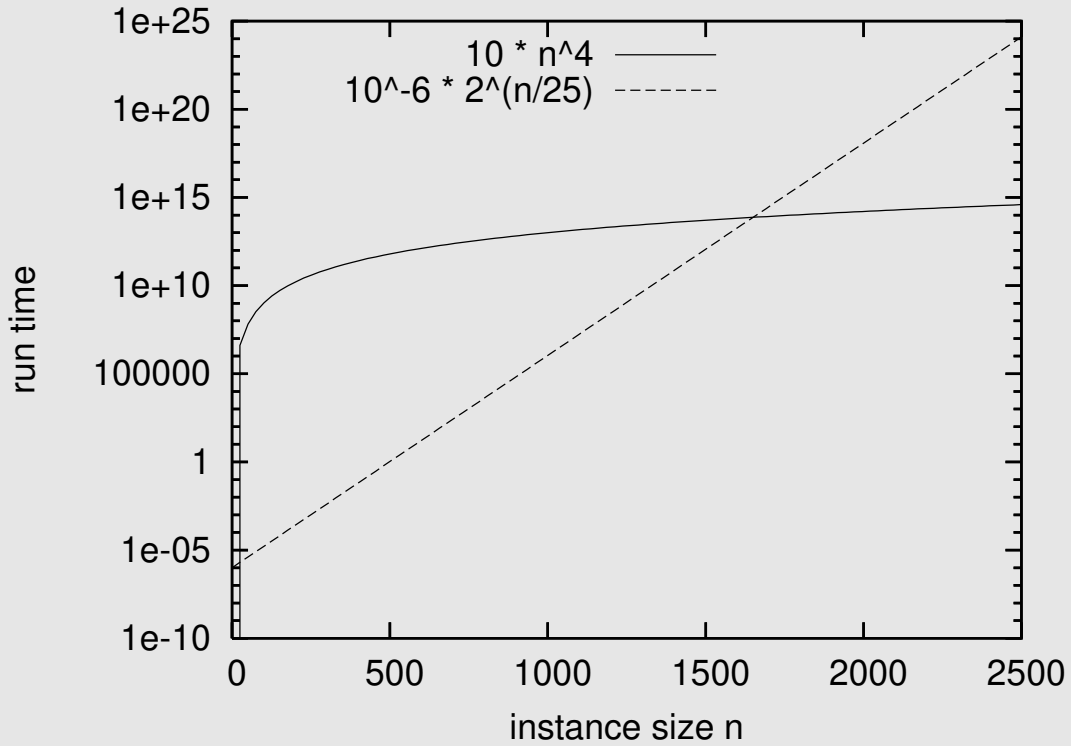
Considerations

A complete method might need hundreds of years on a computer to find an optimal solution to an open shop scheduling instance with 200 operations

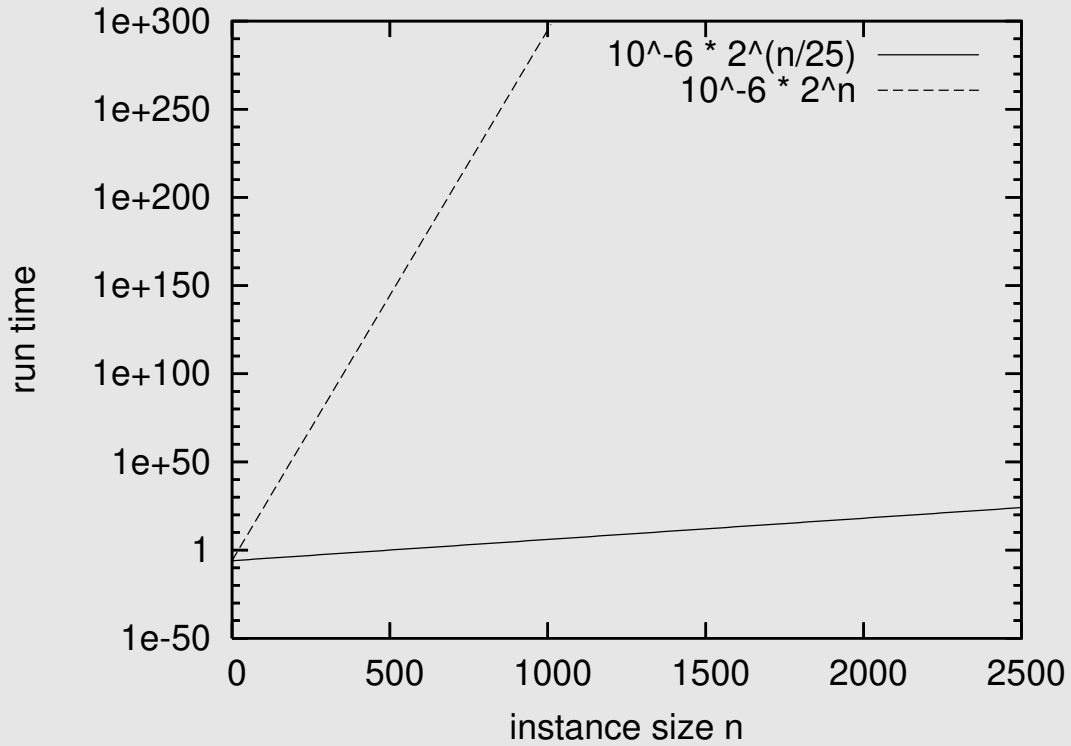
Consequence

For large-scale instances of *NP*-hard problems approximate methods are often the only alternative

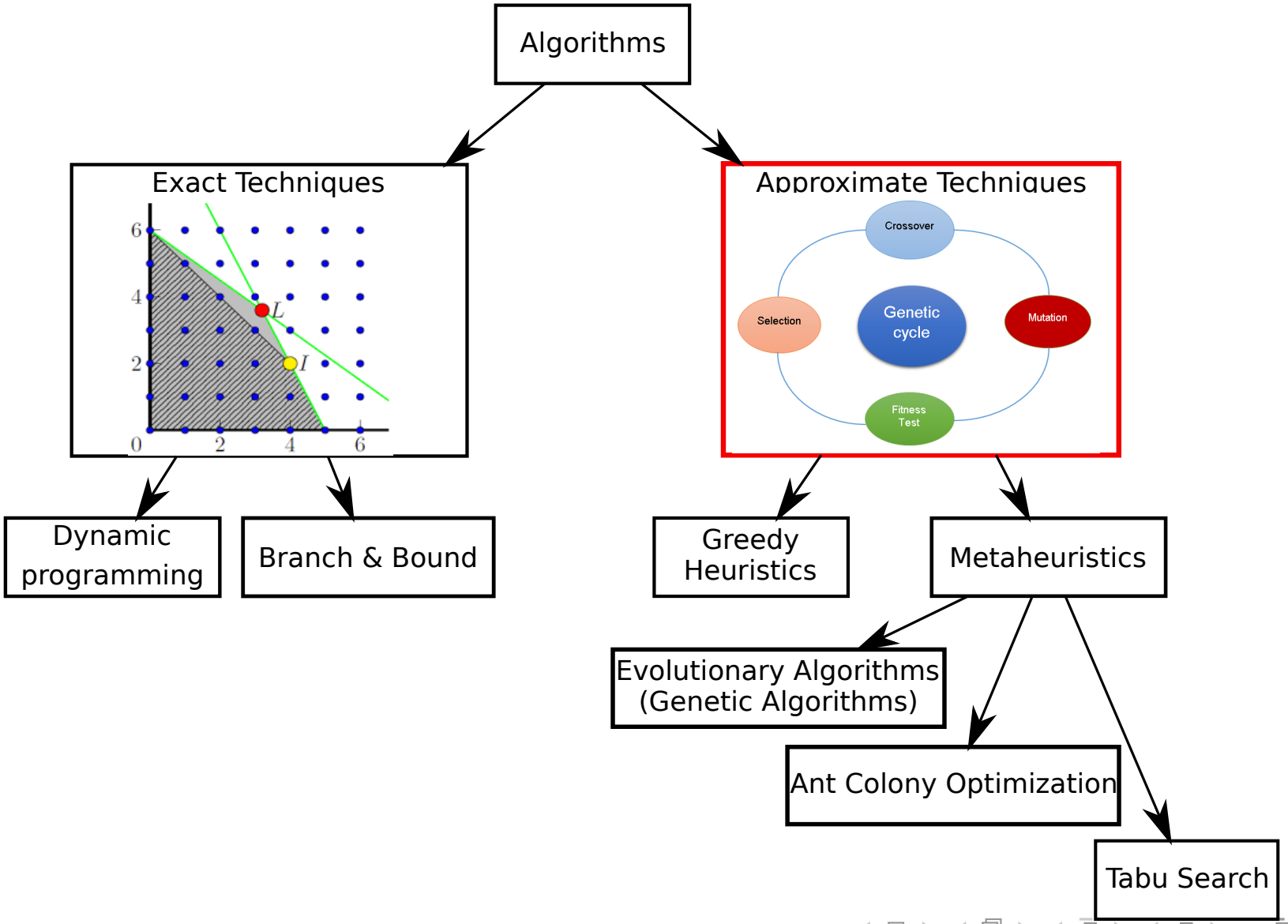
Example: polynomial vs. exponential growth



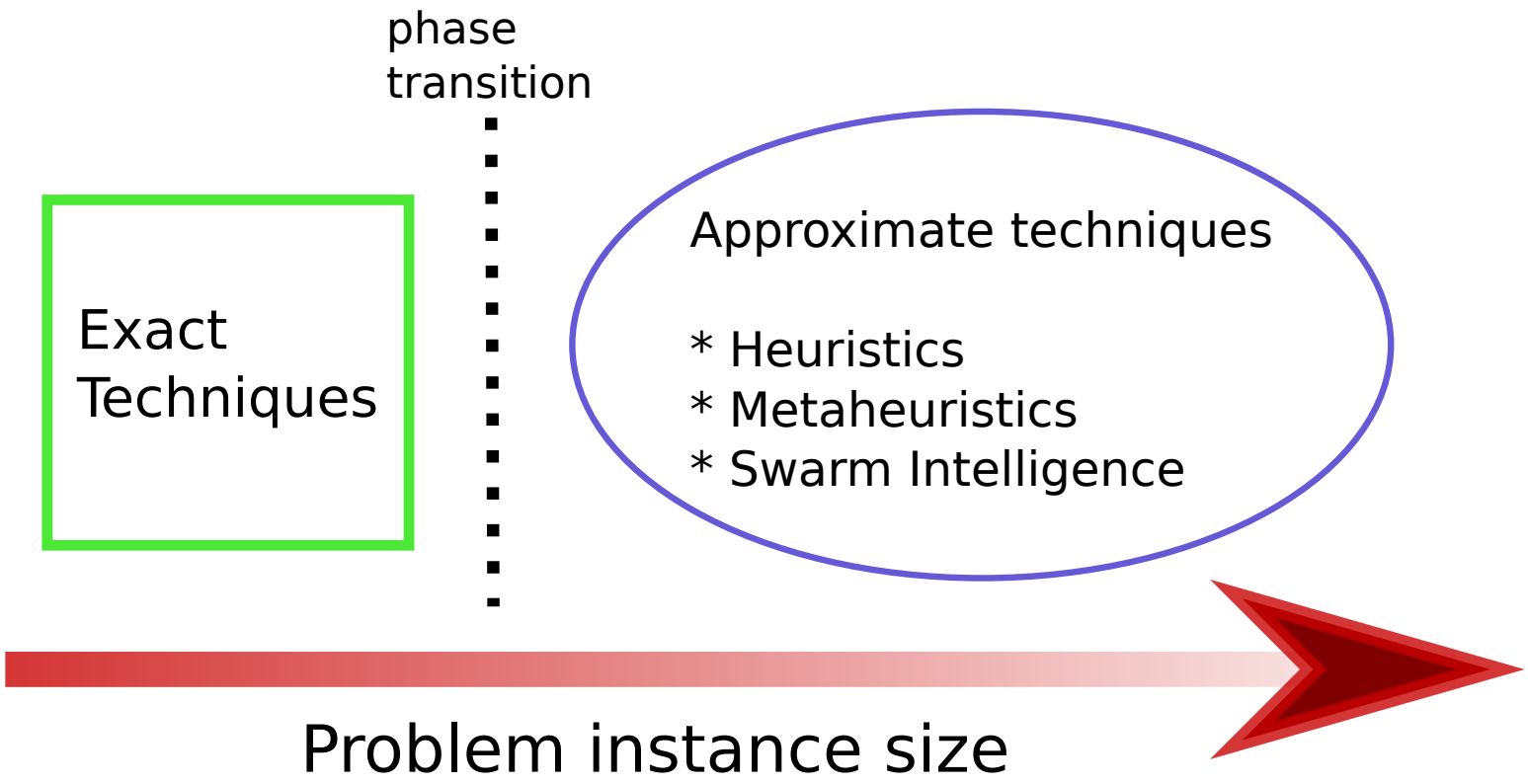
Example: impact of constants



Optimization Techniques



Approximate Techniques: Advantages



Remember: Approximate techniques excel for large instances, problems with uncertainty, multi-objective problems

Constructive heuristics

Historical note

Constructive heuristics are mong the earliest approximate methods!

How do they work?

- They generate solutions from scratch, starting from an empty solution.
- At each step they add a solution component to the current partial solution until the solution is complete.
- Solutions (s) and partial solutions (s^p) are sequences $\langle c_i, ..., c_k \rangle$ composed of solution components from a finite set of solution components C.

Basic ingredient

A construction mechanism which specifies for each feasible partial solution s^p the set of solution components $N(s^p) \subset C$ that can be added.

Constructive heuristics

Pseudo-code

1: $s^p = \langle\rangle$
2: Generate $N(s^p)$
3: **while** $N(s^p) \neq \emptyset$ **do**
4: $\quad c \leftarrow$ Choose$(N(s^p))$
5: $\quad s^p \leftarrow$ extend s^p by appending c
6: $\quad$ Generate $N(s^p)$
7: **end while**

Special case: Greedy heuristics

Characteristics of Greedy heuristics

- In Greedy heuristics function Choose($N(s^p)$) is implemented using a weighting function:

  $\eta : N(s^p) \mapsto \mathbb{R}$

- Greedy heuristics **select at each construction step** the component with the highest/lowest value from $N(s^p)$.

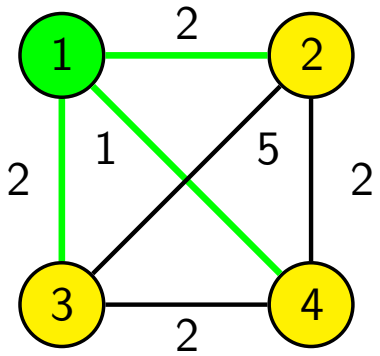
Important

The solution construction mechanism and the implementation of Choose($N(s^p)$) are of crucial importance for the algorithms' performance.

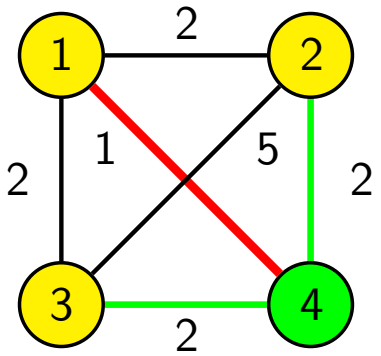
Advantage

Implementation is generally simple and the computation time is often low.

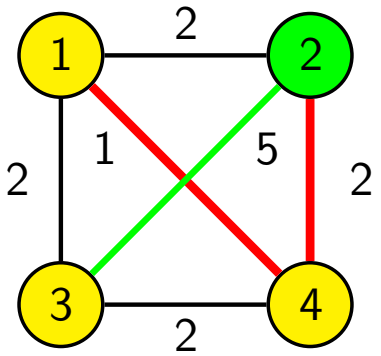
Example TSP: *Nearest-neighbor*



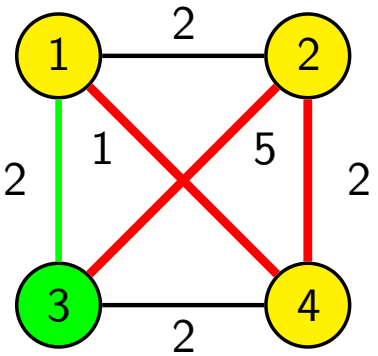
First step



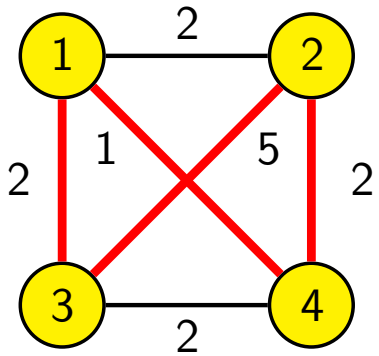
Second step



Third step



Fourth step

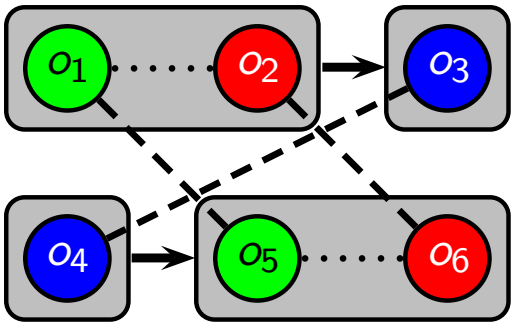


Finish

Additional examples: on the white board!!

- Construction of trees for the k-cardinality tree (KCT) problem
- *List scheduler algorithm* for group shop scheduling (GSS)

Problem data for the GSS example



Processing times:

- $o_1 = 6,\ o_2 = 4,\ o_3 = 3$
- $o_4 = 2,\ o_5 = 10,\ o_6 = 1$

Straight-forward extensions of constructive heuristics: white board!

- Look-ahead strategies
- Roll-out method

List Scheduler Algorithm (GSS)

Characteristics

- Solutions to the GSS problem can be represented by means of permutations of all operations. Example:

$$(o_1, o_2, o_3, o_4, o_5, o_6)$$

- Each solution induces a *disjunctive graph*
- **But:** not every possible permutation results in a valid solution
- Valid solutions may be visualized by means of a **Gantt chart**

List scheduler: way of working

1. Constructs a permutation of all operations from left to right.
2. At each iteration it chooses from those operations whose predecessors are already scheduled.

Possible greedy functions

1. η_1: choose always an operation with shortest processing time
2. η_2: choose always an operation with the earliest possible starting time
   Tie breaking: use η_1

Search trees (1)

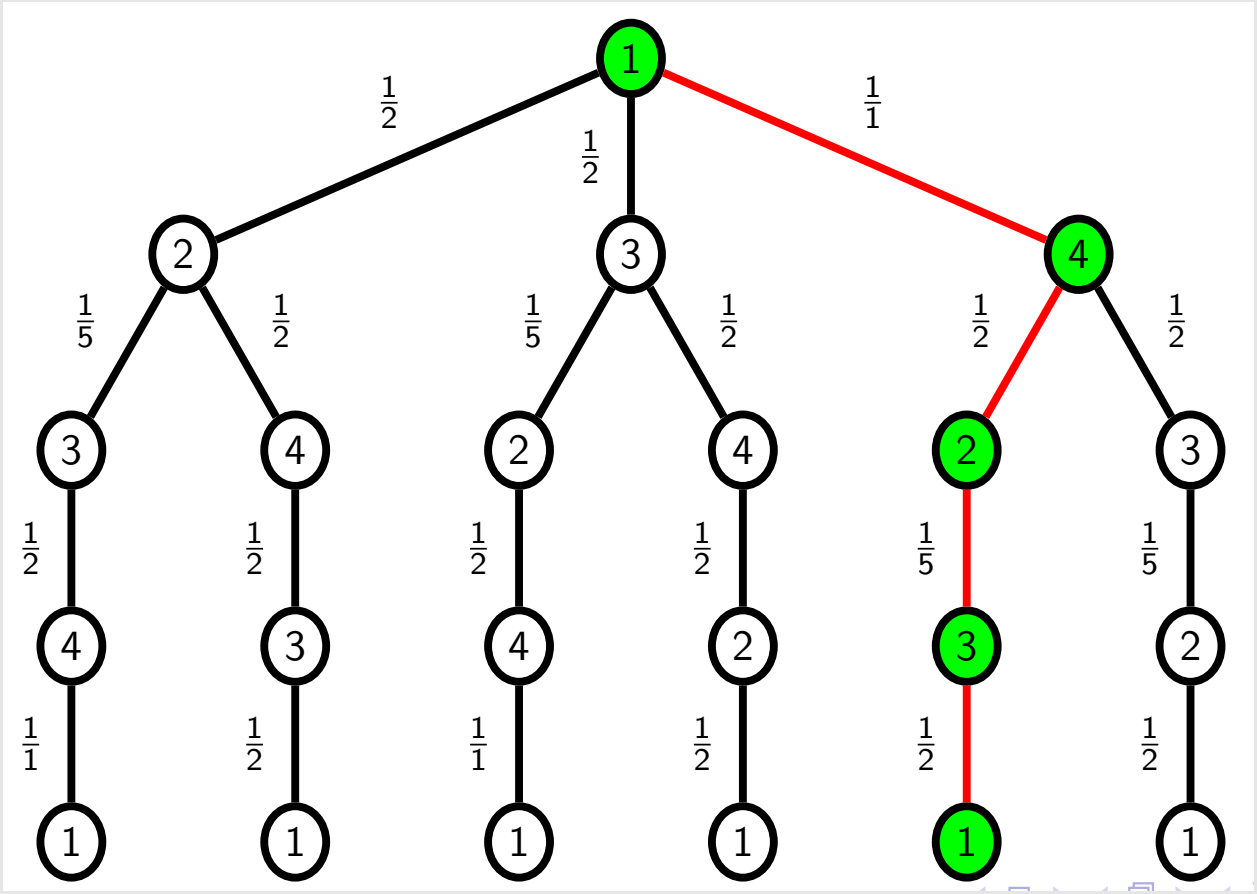
Search trees

A search tree is the depiction of all possible solution constructions of a constructive heuristic in form of a tree.

Observe

Search trees are a way of depicting the search space of a problem when solved by constructive heuristics.

Example: *Nearest-neighbor heuristic* for the TSP



Questions?

Local search

Main idea

Change an existing solution rather than creating solutions from scratch.

How does basic local search work?

- Takes a solution as input. This is the first incumbent solution.
- At each iteration it chooses a solution better than the incumbent solution from a **neighborhood** of the incumbent solution.
- If there is no better solution, the local search stops.

Observe

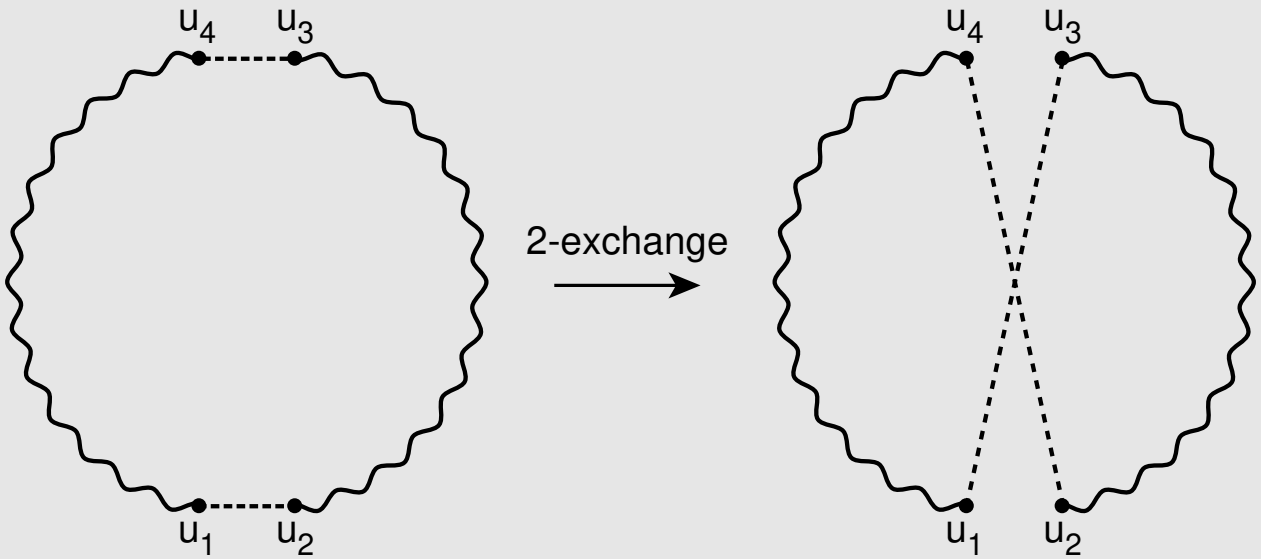
Neighborhoods are the basic ingredients of local search methods.

Neighborhood structures/operators (1)

Definition

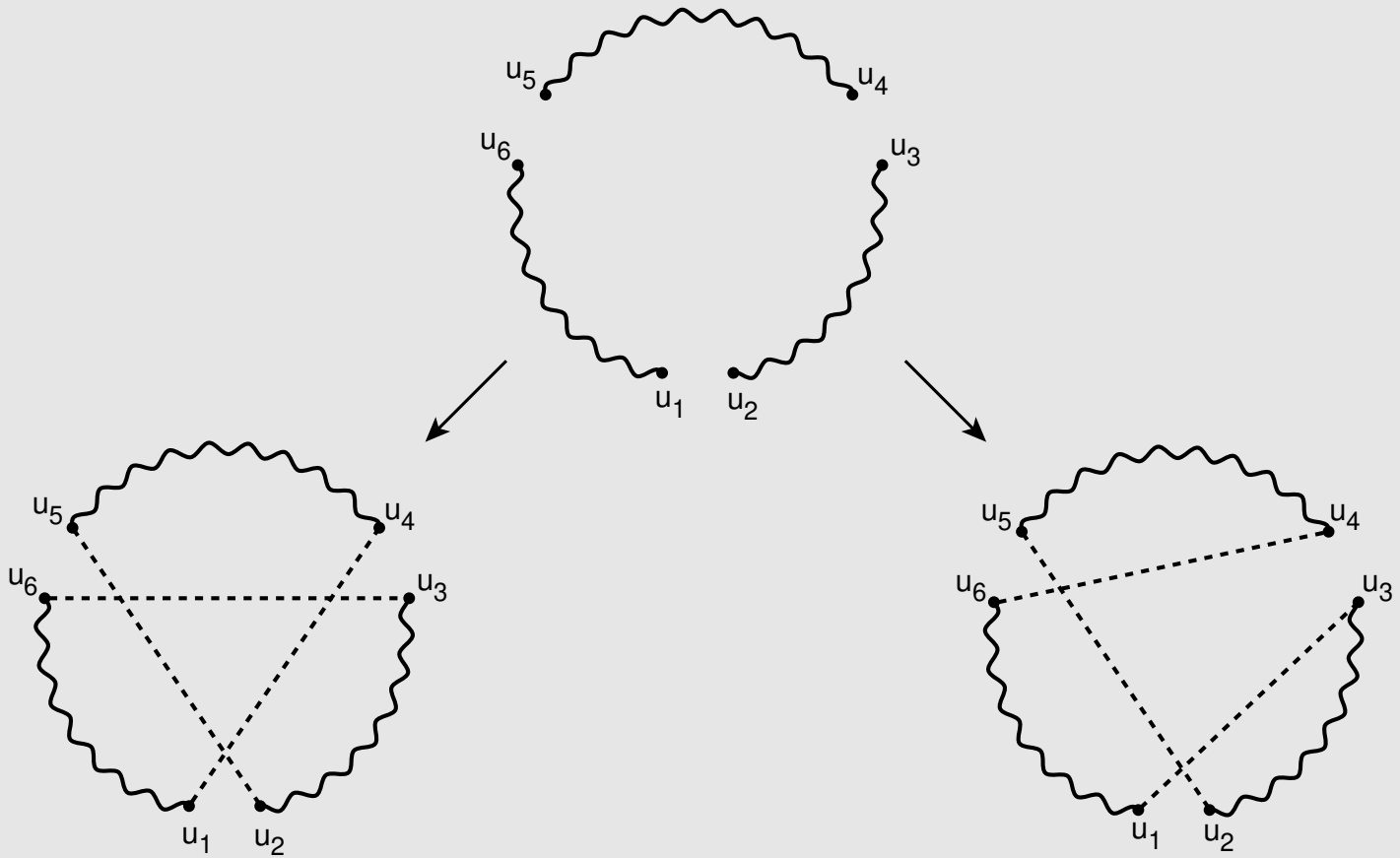
A neighborhood structure/operator is a function $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ that assigns to each $s \in \mathcal{S}$ a set of neighbors $\mathcal{N}(s) \subseteq \mathcal{S}$. $\mathcal{N}(s)$ is called the neighborhood of s.

Example: 2-opt neighborhood for the TSP



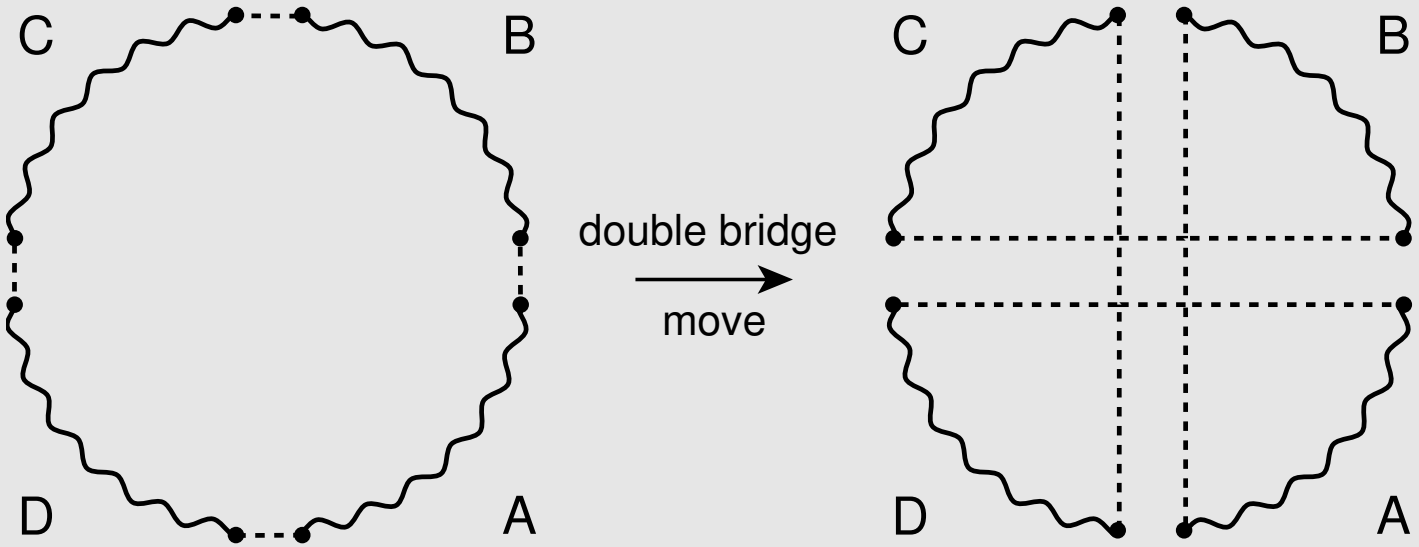
Neighborhood structures/operators (2)

Example: 3-opt neighborhood for the TSP



Neighborhood structures/operators (3)

Example: *double-bridge* neighborhood for the TSP

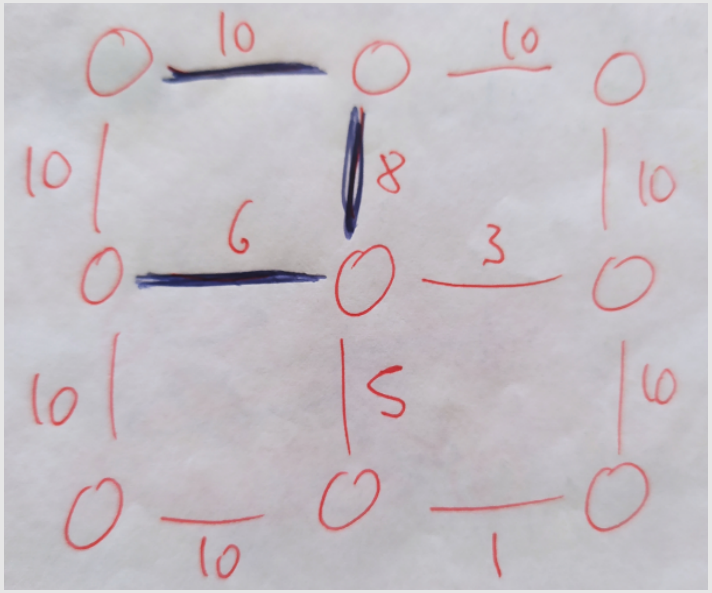


Example: k-cardinality tree (KCT) probl.

1-exchange neighborhood for the KCT problem

Given a tree T, the 1-exchange neighborhood $N(T)$ of T consists of the following trees: $T' \in N(T)$ if and only if T' is obtained from T by removing exactly one leaf edge e from T and by adding an edge $e' \neq e$ that is connected with exactly one of its end-nodes to $T \setminus \{e\}$.

Example: 3-KCT



Example: group shop scheduling (GSS)

Neighborhood: the intuitive option

- **Definition:** the neighborhood of a solution consists of those solutions that are obtained by changing the direction of exactly one arc (of those arcs that are initially undirected).
- **Disadvantage:** Many solutions obtained in this way are invalid

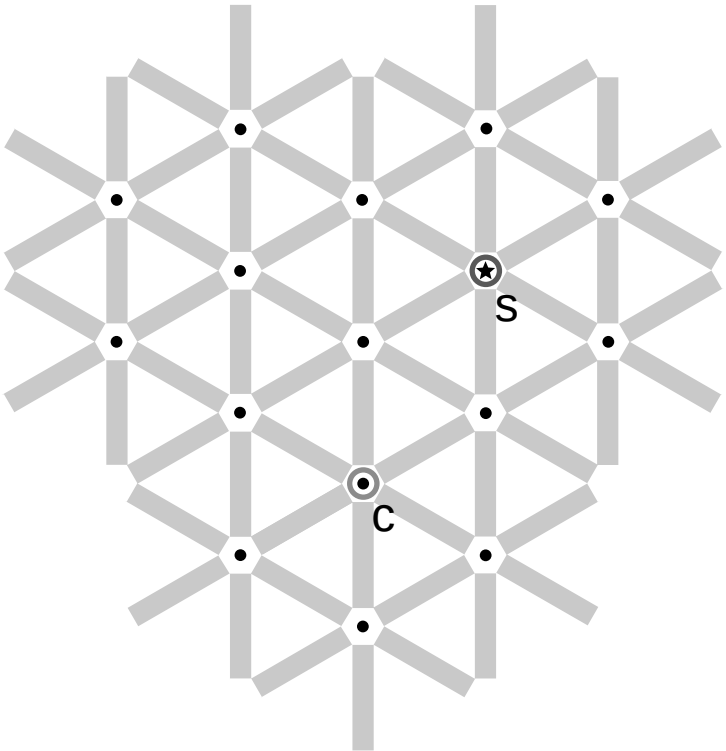
Observations

- Changing the direction of arcs that do not belong to a critical path cannot improve a solution.
- **Literature result:** Changing the direction of an arc of a critical path always leads to a feasible solution.
- Each critical path can be decomposed into sequences of operations from the same group (*group block*), respectivamente the same machine (*machine block*).
- **Literature result:** only changing the direction of the first or the last arc of a block may improve a solution.

Neighborhood graphs

Neighborhood graph

A neighborhood structure together with a problem instance define the so-called neighborhood graph



- **Nodes:** solutions
- **Edges:** connect neighboring solutions
- s: optimal solution
- c: current solution

Local and global minima

Definition: global minimum

A solution $s^* \in \mathcal{S}$ is called a globally minimal solution (or global minimum) if for all $s \in \mathcal{S}$ it holds that $f(s^*) \leq f(s)$. The set of all globally minimal solutions is henceforth denoted by $\mathcal{S}^*$.

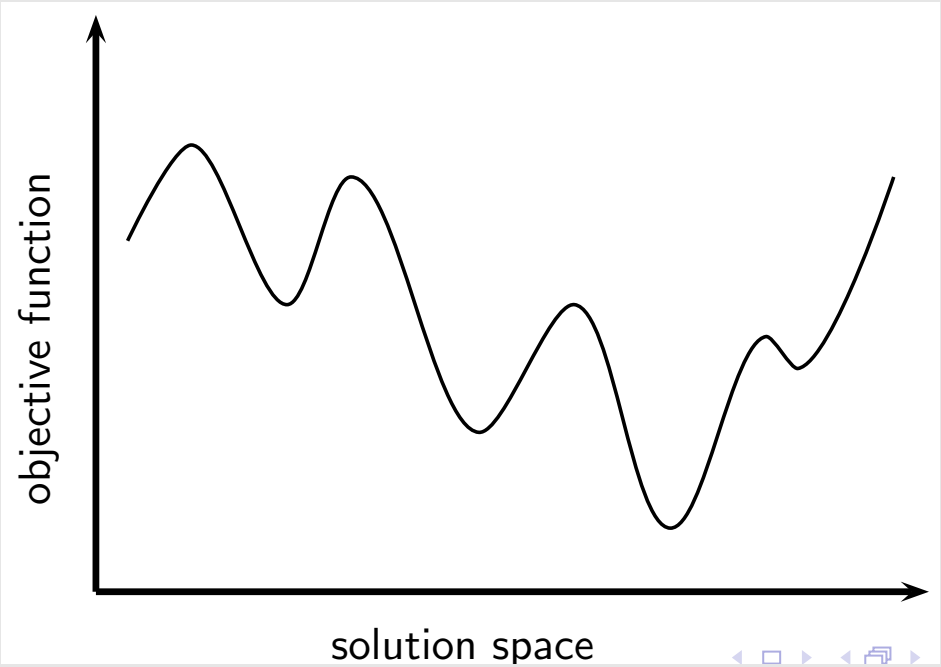
Definition: local minimum

A locally minimal solution (or local minimum) with respect to a neighborhood structure $\mathcal{N}$ is a solution $\hat{s} \in \mathcal{S}$ such that $f(\hat{s}) \leq f(s)$ for all $s \in \mathcal{N}(\hat{s})$. We call $\hat{s}$ a strict local minimmum of $f(\hat{s}) < f(s)$ for all $s \in \mathcal{N}(\hat{s})$.

Search landscape

Neighborhood graphs can be graphically shown as search landscapes.

Graphical example (2D -> not realistic)



Local search: pseudo code

Pseudo code

$s \leftarrow$ GenerateInitialSolution()
while $\exists\, s' \in \mathcal{N}(s)$ such that $f(s') < f(s)$ **do**
  $s \leftarrow$ ChooseImprovingNeighbor($\mathcal{N}(s)$)
end while

Implementations of ChooseImprovingNeighbor($\mathcal{N}(s)$)

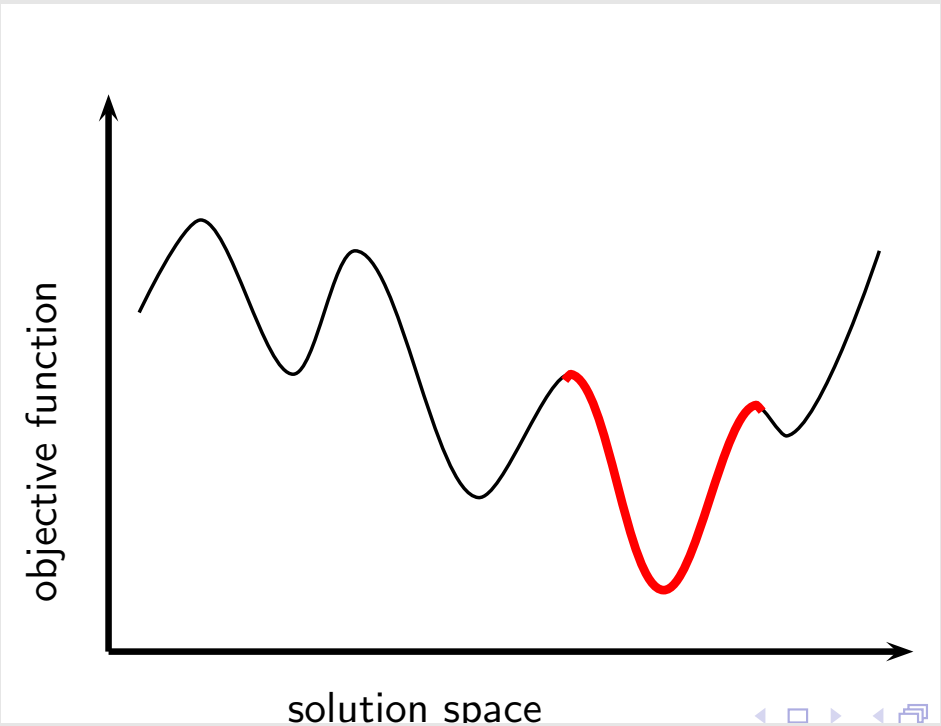
- **First improvement:** Scans the neighborhood an returns the first improving neighbor
- **Best improvement:** Returns the best neighbor of the neighborhood

Basins of attraction

Definition: basin of attraction

The basin of attraction of a local minimum $\hat{s} \in \mathcal{S}$ is defined as the set of all solutions s for which a local search stops in $\hat{s}$ when started in s.

Graphical example



Important considerations (1)

Smoothness/ruggedness of the *search landscape*

- The smaller the average size of the neighborhood of a solution, the more rugged is the search landscape.
- The bigger the average size of the neighborhood of a solution, the more smooth is the search landscape.

Implications for local search

- The more rugged the search landscape, the worse are the the local optima on average.
- The more smooth the search landscape, the better are the local optima on average.

Important considerations (2)

However ...

- The smaller the average size of the neighborhood of a solution, the faster is the iterative improvement procedure.
- The bigger the average size of the neighborhood of a solution, the slower is the iterative improvement procedure.

Conclusion

A good trade-off has to be found between the the average size of the neighborhood of a solution, and the average quality of the local minimum that is found by the iterative improvement procedure.

Questions?

Question

We have constructive methods and we have local search. Why do we need something better, or more intelligent?

Possible answers

- Constructive method and greedy functions might be such that an optimal solution can not be found.
- The probability to find by chance a good starting point for local search such that an optimal solution is found might be very low.

Conclusion

Methods for exploring a search space must to be used!

Historical note

The term metaheuristic, first introduced in [Glover, 1986],[a] derives from the composition of two Greek words. The term heuristic derives from the verb *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$) which means to find, while the suffix meta means beyond, in an upper level.

[a] [Glover, 1986] F. Glover, 1986. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*, 533–549.

Properties of metaheuristics (1)

- Metaheuristics are strategies that guide the search process.
- The goal is to efficiently explore the search space in order to find (near)optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.

Properties of metaheuristics (2)

- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Todays more advanced metaheuristics use search experience (memory).

Intensification vs. Diversification

Important concepts

- **Intensification:** Exploitation of the accumulated search experience.
- **Diversification:** Identification of areas in the search space that are not yet explored.

General requirement

A well-working metaheuristic provides a balance between intensification and diversification (alternating phases)

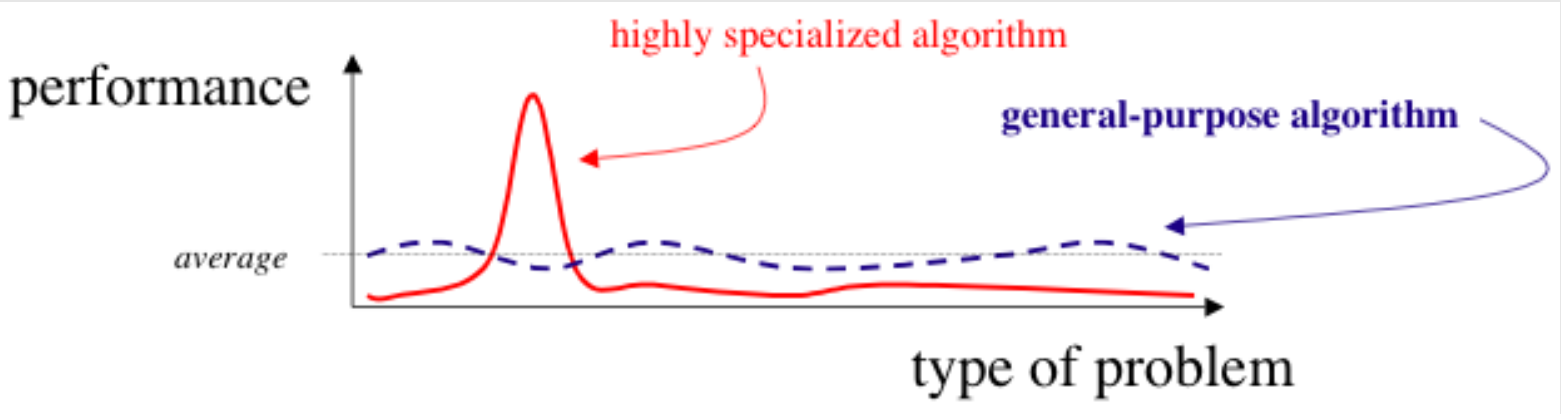
No free lunch

David H. Wolpert and William G. Macready (1995)

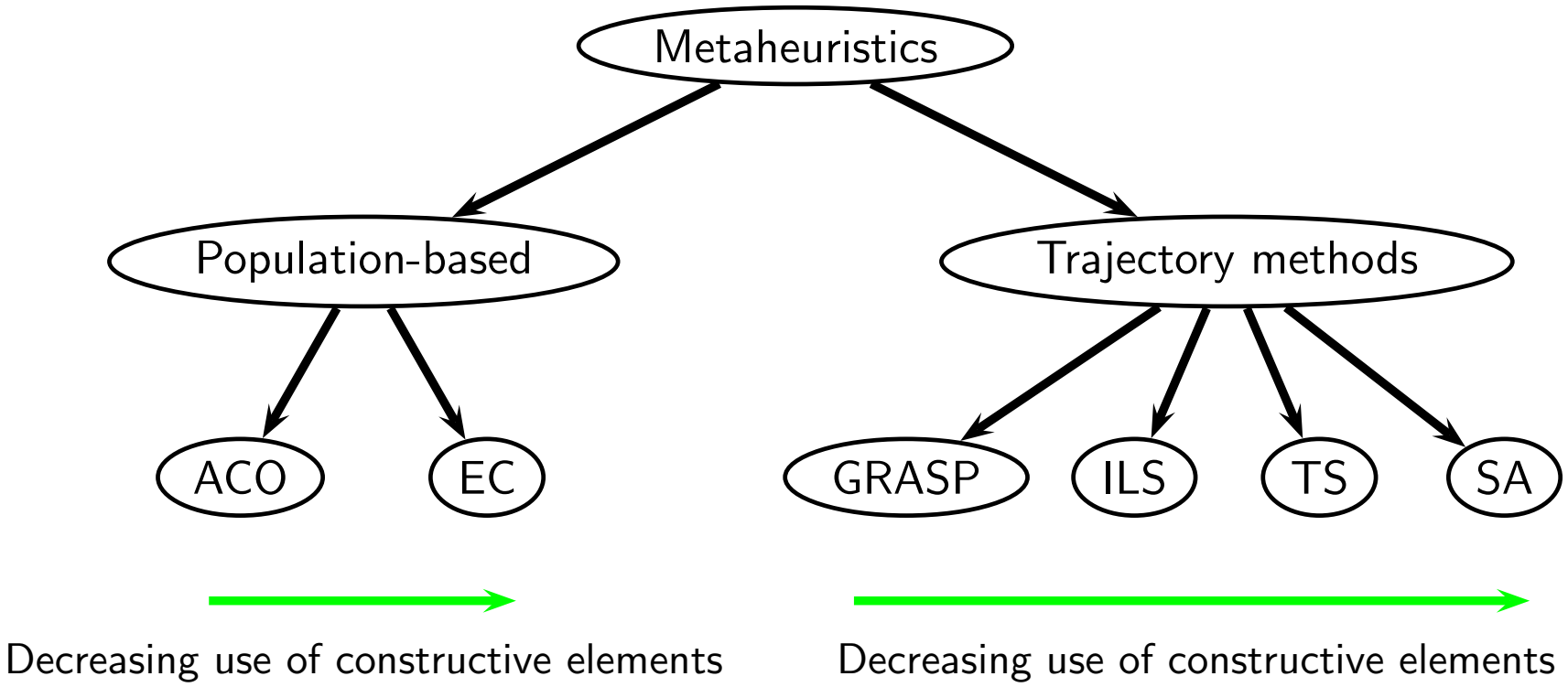
... proved that

"[...] all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions."

No free lunch: graphical illustration



Metaheuristics: classification



Metaheuristics: time line

Year of introduction

- Simulated Annealing (SA) [Kirkpatrick, 1983]
- Tabu Search (TS) [Glover, 1986]
- Genetic and Evolutionary Computation (EC) [Goldberg, 1989]
- Ant Colony Optimization (ACO) [Dorigo, 1992]
- Greedy Randomized Adaptive Search Procedure (GRASP) [Resende, 1995]
- Particle Swarm Optimization (PSO) [Kennedy, 1995]
- Guided Local Search (GLS) [Voudouris, 1997]
- Iterated Local Search (ILS) [Stützle, 1999]
- Variable Neighborhood Search (VNS) [Mladenović, 1999]

Questions?