

Course Overview
Preliminaries
Algorithms
Metaheuristics

Course: Metaheuristics and Hybrid Methods for Combinatorial Optimization

Christian Blum

Artificial Intelligence Research Institute (IIIA-CSIC)
Bellaterra, Spain

March, 2017

Christian Blum
Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview
Preliminaries
Algorithms
Metaheuristics

Schedule, Topics and Requirements

Schedule

Schedule

- ▶ **Week 1:** Theory classes, (10:00-12:00 a.m.)
- ▶ **Week 2:** Theory classes, (10:00-12:00 a.m.)
- ▶ **Consultation:** every day from 15:00 - 17:00 as needed

Christian Blum
Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview
Preliminaries
Algorithms
Metaheuristics

Schedule, Topics and Requirements

Evaluation

Evaluation

- ▶ **Informal tasks** from one day to the other. Examples:
 - ▶ Develop a greedy heuristic for problem X
 - ▶ Develop an integer linear programming model for problem Y
- ▶ **Practical project**
 - ▶ Implementation of metaheuristics and hybrids for a given problem
 - ▶ Experimental evaluation
 - ▶ Written report (to be delivered by April 7, 2017).
 - ▶ Delivery by email to christian.blum@iiia.csi.es

Christian Blum
Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview
Preliminaries
Algorithms
Metaheuristics

Schedule, Topics and Requirements

Topics

Course content

- ▶ **Combinatorial optimization**
- ▶ **Simple heuristic methods**
- ▶ **Metaheuristics (MHs)**
 - ▶ Ant colony optimization
 - ▶ Particle swarm optimization
 - ▶ Evolutionary computation
 - ▶ Iterated local search, variable neighborhood search
 - ▶ Tabu search
- ▶ **Hybrid Metaheuristics**
 - ▶ Combinations of MHs with integer linear programming solvers
- ▶ **Algorithm tuning and comparison**

Christian Blum
Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview
Preliminaries
Algorithms
Metaheuristics

Schedule, Topics and Requirements

Requirements and Links

Requirements

- ▶ **Laptop** with a recent version of **Linux** as operation system
- ▶ **Software**
 - ▶ GNU C++ compiler (gcc). I have version 4.8.4 on my laptop
 - ▶ Statistics package R: <https://www.r-project.org/>
 - ▶ R library *ggplot2*: <http://ggplot2.org/>
 - ▶ R library *scmamp*: <https://cran.r-project.org/web/packages/scmamp/>
 - ▶ R library *reshape2*: <https://cran.r-project.org/web/packages/reshape2/index.html>
 - ▶ IBM ILOG CPLEX (general purpose ILP solver). To be found in Dropbox for 64-bit Linux systems.

Christian Blum
Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview
Preliminaries
Algorithms
Metaheuristics

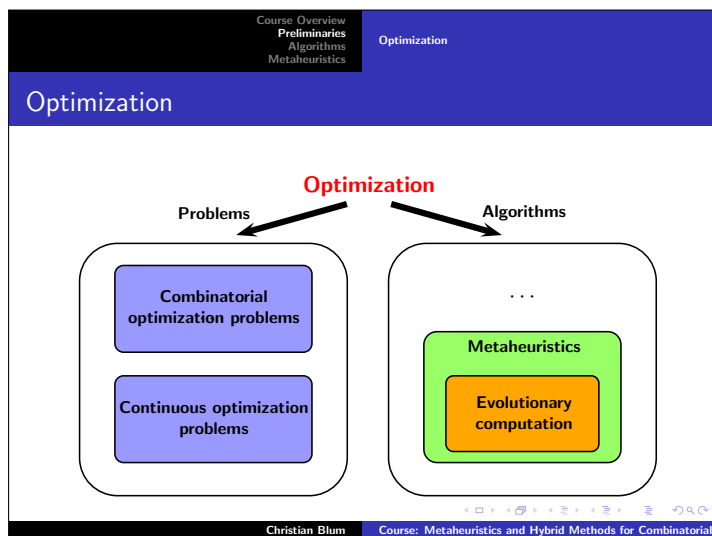
Schedule, Topics and Requirements

Reading List

Additional Literature (Dropbox)

- ▶ 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.

Christian Blum
Course: Metaheuristics and Hybrid Methods for Combinatorial



Course Overview
Preliminaries
Algorithms
Metaheuristics

Optimization

Combinatorial optimization

Papadimitriou and Steiglitz, 1982

Definition: CO problem

A combinatorial optimization (CO) problem $\mathcal{P} = (S, f)$ is an optimization problem in which is given:

- ▶ a **finite set of objects S** (typically integer numbers, subsets of a set of items, permutations of a set of items, or graph structures);
- ▶ an **objective function $f : S \rightarrow \mathbb{R}^+$** that assigns a positive cost value to each of the objects $s \in S$.

Goal: Find an object of minimal cost value.

Note: Minimizing f is the same as maximizing $-f$.

Christian Blum Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview
Preliminaries
Algorithms
Metaheuristics

Optimization

Importance of Combinatorial Optimization

Examples of CO problems in real-life

- ▶ Flight/train scheduling
- ▶ Timetabling
- ▶ Vehicle Routing
- ▶ Layout problems
- ▶ Assignment problems

Goal: Saving money and time!!

Christian Blum Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview
Preliminaries
Algorithms
Metaheuristics

Optimization

Classical Problem: Minimum Weight Vertex Cover

Problem input

- ▶ An undirected graph $G = (V, E)$
- ▶ Each $v \in V$ has an integer weight $w(v) \geq 0$

Solutions to the problem

Any $S \subseteq V$ such that

- ▶ for all $e = (v, u) \in E$ it holds that $\{u, v\} \cap S \neq \emptyset$

Optimization goal

Find $S^* \subseteq V$ that minimizes $f(S^*) := \sum_{v \in S^*} w(v)$

Christian Blum Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview
Preliminaries
Algorithms
Metaheuristics

Optimization

Classical Problem: Minimum Weight Vertex Cover

Example

Christian Blum Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview
Preliminaries
Algorithms
Metaheuristics

Optimization

Classical Problem: Minimum Dominating Set

Problem input

- ▶ An undirected graph $G = (V, E)$

Solutions to the problem

Any $S \subseteq V$ such that

- ▶ for all $v \in V$ it holds that $N[v] \cap S \neq \emptyset$

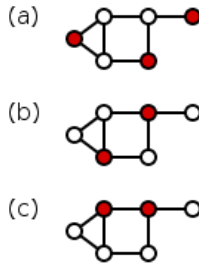
Optimization goal

Find $S^* \subseteq V$ that minimizes $f(S^*) := |S^*|$

Christian Blum Course: Metaheuristics and Hybrid Methods for Combinatorial

Classical Problem: Minimum Dominating Set

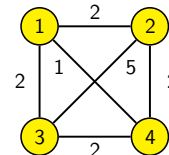
Example



Classical example: TSP

Lawler et al., 1985

In the **Travelling Salesman Problem (TSP)** is given a **completely connected, undirected graph** $G = (V, E)$ with edge-weights.



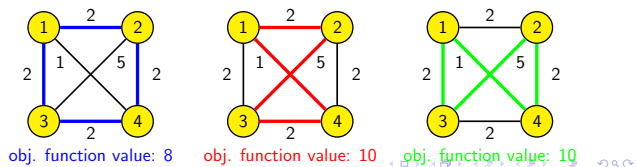
Goal: Find a tour (a Hamiltonian cycle) in G with minimal sum of edge weights.

Classical example: TSP

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

- \mathcal{S} consists of all possible Hamiltonian cycles in G .
- Objective function $f: \mathcal{S} \mapsto \mathbb{R}^+$: $s \in \mathcal{S}$ is defined as the sum of the edge-weights of the edges that are in s .

Example



Example: k -cardinality tree (KCT) problem

Motivation: Design of telecommunication networks



Example: k -cardinality tree (KCT) problem

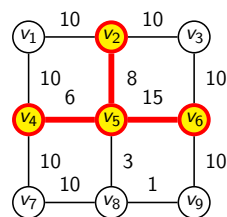
Technical definition

Given: An edge-weighted, undirected graph $G = (V, E)$

Search space: All trees in G with k edges

Obj. function: Sum of the edge-weights

Other real life applications: open pit mining, facility layout



Example: Group shop scheduling (GSS)

Motivation: Organizing a parents-teacher day at a school

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

Constraints:

- Parents can choose the **duration** of a meeting
- Parents can specify **partial orders** of their meetings with the teachers

Course Overview

Preliminaries

Algorithms

Metaheuristics

Optimization

Example: Group shop scheduling (GSS)

Graphically

example instance

possible solution

Parent 1

Parent 2

Parent 3

Obj. function: Makespan, which is the longest path in a solution

Other real life applications: industrial production environments

Christian Blum

Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview

Preliminaries

Algorithms

Metaheuristics

Optimization

Example: Group shop scheduling (GSS)

Special GSS case: Job shop scheduling (JSS)

Parent 1

Parent 2

Parent 3

Christian Blum

Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview

Preliminaries

Algorithms

Metaheuristics

Optimization

Example: Group shop scheduling (GSS)

Special GSS case: Open shop scheduling (OSS)

Parent 1

Parent 2

Parent 3

Christian Blum

Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview

Preliminaries

Algorithms

Metaheuristics

Optimization

Example: Longest Common Subsequence (LCS) Problem

Notation: What is a subsequence of a string?

A string t is called a subsequence of a string x ,
iff t can be produced from x by deleting characters

Example:

Is AAC a subsequence of ACAGTC? YES

ACAGTC

Christian Blum

Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview

Preliminaries

Algorithms

Metaheuristics

Optimization

Example: Longest Common Subsequence (LCS) Problem

Problem definition

Given:

A problem instance (S, Σ) , where

►

S is a set of n input strings over the alphabet Σ

Optimization goal:

Find a longest string t^* that is a subsequence of all strings in $S \rightarrow$
a longest common subsequence

Christian Blum

Course: Metaheuristics and Hybrid Methods for Combinatorial

Course Overview

Preliminaries

Algorithms

Metaheuristics

Optimization

Example: Longest Common Subsequence (LCS) Problem

Example (on 2 input strings)

ACCGGTGGACAATTCA

GGAAAGAGATATGCAC

GGG GAT TCA

Christian Blum

Course: Metaheuristics and Hybrid Methods for Combinatorial

Example: Longest Common Subsequence (LCS) Problem

Restricted versions of the LCS problem

- ▶ **Repetition-free LCS problem**: Each letter may only occur once in the solution
- ▶ **Longest arc-preserving common subsequence problem**: respect RNA secondary structure



Discrete optimization

Definition: Discrete optimization problem

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

- ▶ a search (or solution) space \mathcal{S} defined over
 - ▶ a finite set of n discrete decision variables X_i ($i = 1, \dots, n$);
 - ▶ and a set Ω of constraints among the variables;
- ▶ an objective function $f : \mathcal{S} \rightarrow \mathbb{R}^+$ to be minimized.

If the set of constraints Ω is empty, \mathcal{P} is an **unconstrained problem model**, otherwise a **constrained problem model**.

Note: Each CO problem can be formulated as a discrete optimization problem

Example: Minimum Weight Vertex Cover

Discrete Model

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v w(v) \\ \text{s.t.} \quad & x_v + x_u \leq 1 \quad \text{for } e = (u, v) \in E \\ & x_v \in \{0, 1\} \quad \text{for } v \in V \end{aligned} \quad (1)$$

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

Example: Minimum Dominating Set

ILP model

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad & x_v + \sum_{u \in N(v)} x_u \geq 1 \quad \text{for } v \in V \\ & x_v \in \{0, 1\} \quad \text{for } v \in V \end{aligned} \quad (2)$$

Example: TSP

ILP model

$$\begin{aligned} \min \quad & \sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{i,j} x_{i,j} \\ \text{s.t.} \quad & \sum_{i=1}^{j-1} x_{i,j} + \sum_{i=j+1}^n x_{j,i} = 2 \quad \forall j = 1, \dots, n \\ & \sum_{i,j \in L} x_{i,j} \leq |L| - 1 \quad \forall L \subset \{1, \dots, n\} \\ & x_{i,j} \in \{0, 1\} \quad \text{s.t. } 3 \leq |L| \leq n-3 \\ & \quad \quad \quad \forall 1 \leq i < j \leq n \end{aligned}$$

Alternative ILP model (1)

$$\begin{aligned} \min \quad & \sum_{e \in E} d_e y_e \\ \text{s.t.} \quad & x_{1,1} = 1 \\ & \sum_{k=1}^n x_{i,k} = 1 \quad i = 1, \dots, n \\ & \sum_{i=1}^n x_{i,k} = 1 \quad k = 1, \dots, n \\ & \sum_{e \in E} y_e = n \end{aligned} \quad (4)$$

Alternative ILP model (2)

$$\begin{aligned} x_{i,k-1} + x_{j,k} - y_e &\leq 1 & \forall e = (i,j) \in E, k \geq 2 \\ x_{i,n} + x_{1,1} - y_e &\leq 1 & \forall e = (i,1) \in E \\ x_{i,k} &\in \{0,1\} & \forall i,k = 1, \dots, n \\ y_e &\in \{0,1\} & \forall e \in E \end{aligned}$$

Interpretation of variables: $x_{i,k}$ is set to one if node i is the k -th visited city

Continuous optimization

Definition: Continuous optimization problem

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

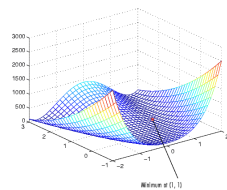
- ▶ a **search (or solution) space** $\mathcal{S} \subseteq \mathbb{R}^n$ defined over
 - ▶ a finite set of n continuous decision variables x_i ($i = 1, \dots, n$);
 - ▶ and a set Ω of *constraints* among the variables;
- ▶ an *objective function* $f : \mathcal{S} \rightarrow \mathbb{R}$ to be minimized.

If the set of constraints Ω is empty, \mathcal{P} is an **unconstrained problem model**, otherwise a **constrained problem model**.

Note: Minimizing over an objective function f is the same as maximizing over $-f$

Example: Function optimization

Rosenbrock function

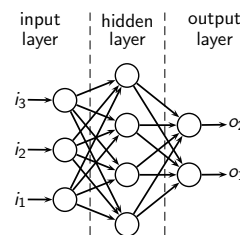


Mathematical description:

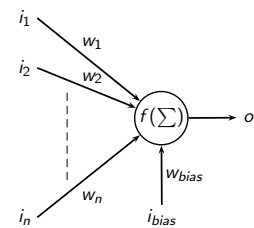
$$f(X_1, X_2) = 100 \cdot (X_2 - X_1^2)^2 + (1 - X_1)^2$$

Example: Neural network training

Feed-forward neural networks



Example feed-forward NN



A neuron

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

Goal

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

How to solve a CO problem?

We need an ALGORITHM !!!

General distinction:

- ▶ **Complete methods.**
They **guarantee to find** for every finite size instance of a CO problem **an optimal solution in bounded time.**
- ▶ **Approximate methods.**
No guarantee of finding an optimal solution.

Question: Why not always using a complete method?

Problem hardness

Many relevant CO problems are *NP*-hard

Different classes of problems:

- ▶ *P*: A complete method exists that needs an execution time that is polynomial.

Example: Minimum Spanning Tree (MST) problem

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

Example: Traveling Salesman Problem (TSP)

Therefore:

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

Problem hardness

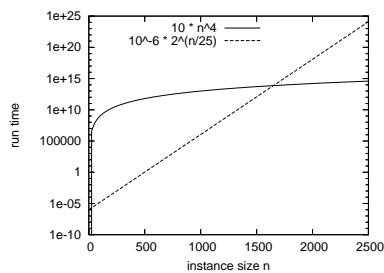
Example

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

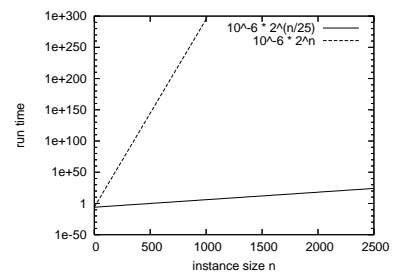
Problem hardness

Example: Polynomial vs exponential growth

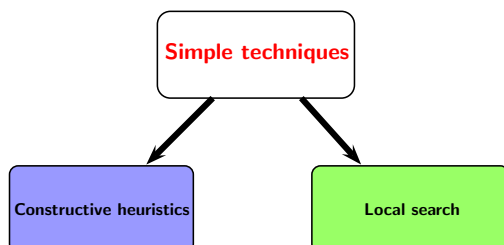


Problem hardness

Example: Impact of constants



Simple techniques



Constructive heuristics

Historical note: Among the earliest approximate methods!

How do they work? They generate solutions from scratch by adding opportunistically defined solution components to an initially empty partial solution, until a solution is complete.

This means: Solutions s and partial solutions s^p are sequences $\langle c_1, \dots, c_k \rangle$ 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 $N(s^p)$ that can be added.

Constructive heuristics

Pseudo-code

```

1:  $s^P = \langle \rangle$ 
2: Determine  $N(s^P)$ 
3: while  $N(s^P) \neq \emptyset$  do
4:    $c \leftarrow \text{ChooseFrom}(N(s^P))$ 
5:    $s^P \leftarrow \text{extend } s^P \text{ by adding solution component } c$ 
6:   Determine  $N(s^P)$ 
7: end while

```

Constructive heuristics

Example: Greedy heuristic

Implements procedure $\text{ChooseFrom}(N(s^P))$ by applying a **weighting function** to the allowed solution components. Greedy heuristics **choose at each step the component with the highest/lowest value.**

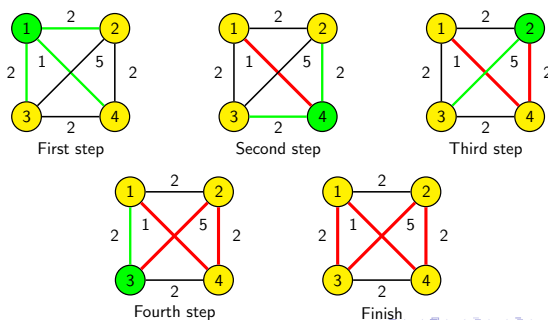
Weighting function: $\eta : N(s^P) \mapsto \mathbb{R}$

Crucial: Construction mechanism and implementation of $\text{ChooseFrom}(N(s^P))$.

Advantage: **Easy to implement and fast in execution.**

Constructive heuristics

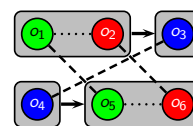
Example: Nearest-neighbor heuristic for TSP



Constructive heuristics

Additional examples

- ▶ Tree construction for the k -cardinality tree problem
- ▶ List scheduler algorithms for group shop scheduling:



Processing times:

- ▶ $\alpha_1 = 6, \alpha_2 = 4, \alpha_3 = 3$
- ▶ $\alpha_4 = 2, \alpha_5 = 10, \alpha_6 = 1$

On the board!!

Constructive heuristics

Extensions of simple greedy heuristics

- ▶ Look-ahead strategies
- ▶ Roll-out methods

On the board!!

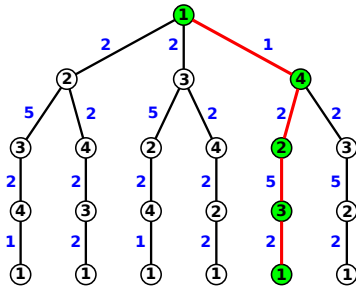
Constructive heuristics

Search trees

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

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

Example: Nearest-neighbor heuristic for TSP



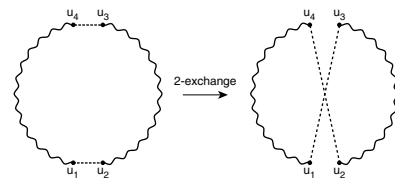
Principle

How does it work? A basic local search method **starts from a solution** and moves successively to a **better solution** in the **neighborhood** of the current solution, until a better neighbor can not be found.

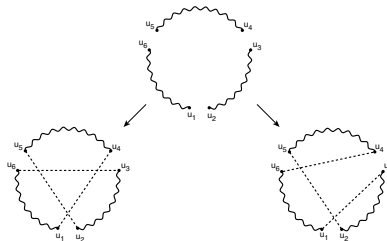
Neighborhood structures
... are the basic ingredient of local search methods.

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

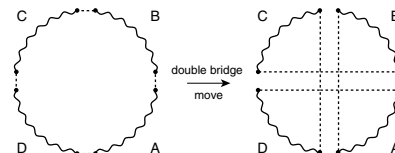
Example: 2-opt neighborhood for TSP



Example: 3-opt neighborhood for TSP



Example: double-bridge neighborhood for TSP



Local search

Additional examples

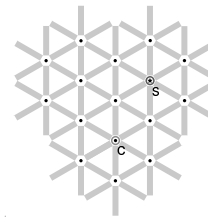
- ▶ 1-exchange neighborhood for the k -cardinality tree problem
- ▶ Different neighborhoods for group shop scheduling

On the board!

Local search

Neighborhood graphs

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



- ▶ **vertices**: candidate solutions
- ▶ **edges**: connect neighboring positions
- ▶ **s**: (optimal) solution
- ▶ **c**: current position

Local search

Local (and global) minima

Definition

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

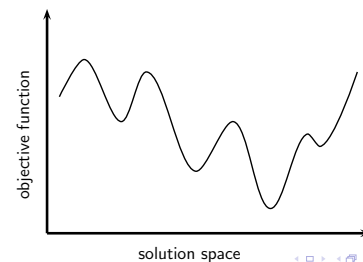
Definition

A **locally minimal solution (or local minimum)** with respect to a neighborhood structure \mathcal{N} is a solution \hat{s} such that $\forall s \in \mathcal{N}(\hat{s}) : f(\hat{s}) \leq f(s)$. We call \hat{s} a strict locally minimal solution if $f(\hat{s}) < f(s) \forall s \in \mathcal{N}(\hat{s})$.

Local search

Search landscape

Neighborhood graphs can be graphically shown as search landscapes



Local search

Pseudo code

```

s ← GenerateInitialSolution()
while ∃ s' ∈ N(s) such that f(s') < f(s) do
    s ← ChooseImprovingNeighbor(N(s))
end while
    
```

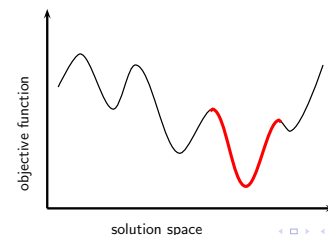
Implementations of ChooseImprovingNeighbor($\mathcal{N}(s)$)

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

Local search

Basins of attraction

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



Local search

Important considerations

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 the iterative improvement procedure:

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

Local search

Important considerations (continued)

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 average size of the neighborhood of a solution, and the average quality of the local minimum that is found by the iterative improvement procedure.

Metaheuristics

Question

We have **constructive methods** and we have **iterative improvement local search**!

Why do we need something more?

Metaheuristics

Some answers

- ▶ Constructive method and weighting function might be such that an optimal solution **can not be found**
- ▶ The **probability to find by chance a "good" starting point** for iterative improvement local search such that an optimal solution is found might be **very small**.

Conclusion: Methods for **exploring a search space** must to be used!

Metaheuristics

Historical note

The term **metaheuristic**, first introduced in [Glover, 1986], derives from the composition of two Greek words. **Heuristic** derives from the verb *heuriskein* (εὕρισκειν) which means "to find", while the suffix **meta** means "beyond, in an upper level".

Metaheuristics

What are metaheuristics?

- ▶ 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**.

Metaheuristics

What are metaheuristics? (continued)

- ▶ 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.
- ▶ Today's more advanced metaheuristics **use search experience (memory)**.

Metaheuristics

Important concepts

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

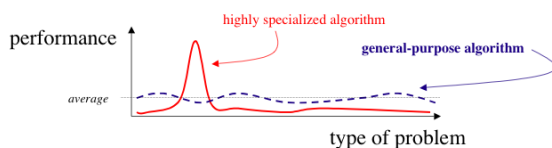
In general: A well-working metaheuristic provides a **balance** between **intensification** and **diversification** (alternating phases)

Metaheuristics

No free lunch

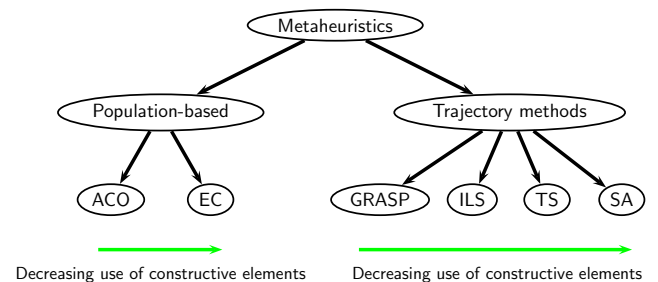
David H. Wolpert and William G. Macready (1995) proved that

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



Metaheuristics

Classification of metaheuristics



Metaheuristics

Metaheuristics timeline

- ▶ 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?

Questions?