Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
IAMP Institut für Angewandte
Mathematik und Physik

MSE Algorithms 2023 – Part 2

L01: Basic Problems and Solutions

Samuel Beer

About

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

Samuel Beer

Email: beer@zhaw.ch

Phone: 058 934 73 20

Office: TN O3.46, Technikumstr. 9, 8401 Winterthur

El. Eng. HTL in Burgdorf

Master in Math at University of Bern

PhD in Algebra at University of Bern

Current:

Lecturer in Mathematics at ZHAW

Head of Inst. of Applied Math and Physics

Course Organization

- Slides and exercises on Moodle:
  https://moodle.msengineering.ch/course/view.php?id=2740
- Lecture on Zoom:
  https://zhaw.zoom.us/j/69998123580
- Starting time: 13:10
- Breaks upon request ☺

Literature

- **J. Dréo, A. Pétrowski, P. Siarry, É. Taillard**, *Metaheuristic for hard combinatorial optimization*, Springer, 2006

- **S. Skiena**, The Algorithm Design Manual, Springer, 2008

- **T. Cormen, C. Leiserson, R. Rivest, C. Stein,** *Introduction to algorithms*, MIT Press, 2009
- **E. Aarts, J. K. Lenstra** (éd.), *Local search in combinatorial optimization*, Princeton Univ. Press, Princeton, 2003 (2nd edition).
- **T. Baeck, D. B. Fogel, Z. Michalewicz,** *Evolutionary Computation*, Institute of Physics Publishing, 2000
- **M. Dorigo, T. Stützle**, *Ant Colony Optimization*, MIT Press, 2004.
- **F.Glover, M. Laguna,** *Tabu Search*, Kluwer, Boston/Dordrecht/London, 1997.
- **H. H. Hoos, T. Stützle**: *Stochastic Local Search: Foundations and Applications*, Morgan Kaufmann / Elsevier, 2004
- **E.-G. Talbi**: *Metaheuristics: From Design to Implementation*, Wiley, 2009

Introduction

Course Overview

PART 1: Fundamental Algorithms and Data Structures

- Sorting, Searching, Lists, Time Complexity, Sweep-Line, Graphs …

PART 2: Meta-Heuristics

- Introduction: Basic problems and algorithms
- Constructive methods: Random building, Greedy
- Local searches
- Randomized methods
- Threshold accepting, Simulated annealing
- Decomposition methods: Large neighbourhood search
- Learning methods for solution building: Artificial ant systems
- Learning methods for solution improvement: Tabu search
- Methods with a population of solutions: Genetic algorithms

Challenge: Help Santa distribute his presents!

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

Starts in second week…

Lecture 1:
Introduction - Basic Problems and Algorithms

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

Goal:

Understand what kind of problems we are going to tackle within this module, and why they are difficult.

Topics:

- Recap: Runtime of Algorithms
- Complexity Theory in a Nutshell
- Sample Problems and Solution Approaches

Showcase 1: Sorting

Input: *n* numbers $a_1, \ldots, a_n$

Problem: Sort the numbers in ascending order

Problem Input	Solution
27, 13, 4, 8, 15, 3, 5	3, 4, 5, 8, 13, 15, 27

Showcase 2: Traveling Salesperson Problem TSP

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

Input: n cities, $D = (d_{ij})$ distances matrix between cities i and j.

Problem: Find the shortest tour passing exactly once in each city.

Problem Input

One "Good" Solution

Runtime of Algorithms

Excercise: Asymptotic Running Time

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

Write down the formal definition of the O-Notation

Solution: O-Notation

Definition from Part 1 of this Lecture:

$f(x) = O(g(x))$ as $x \to \infty$ if there exists a positive number M and a real number x_0 such that

$$f(x) \leq M \cdot g(x) \quad \forall x \geq x_0$$

Alternative definition:

$f(n) = O(g(n)) \quad \Leftrightarrow \quad \exists\ c, n_0$ constant: $\forall n \geq n_0$: $f(n) \leq c*g(n)$

$\Leftrightarrow$ there exist constants c and n_0, such that $f(n) \leq c*g(n)$ holds for all $n \geq n_0$.

O-Notation in a Nutshell

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

- Runtime of algorithms is measured in «O-Notation»:
- Formal definition: $f(n) \in O\big(g(n)\big) \Leftrightarrow \exists\, c, n_0 \text{ constant, such that } \forall\, n \geq n_0 \colon f(n) < c \cdot g(n)$.
- Example: Sorting n numbers with algorithm «Insertion Sort» has runtime «$O(n^2)$».
- This basically means: There is a positive constant *c*, such that for any input of (sufficiently large) size *n* (here: the *n* numbers to be sorted), the algorithm «Insertion Sort» takes at most $c*n^2$ operations. This is an *upper bound* on the real runtime.
- Typical runtimes are $\log(n)$, n, $n*\log(n)$, n^2, n^3, 2^n.
- «Good» runtimes are at most polynomial in *n* with a small exponent, otherwise too slow.
- For a more precise introduction: See Part 1 of this course and/or Skiena, Chapter 2.

Runtime of Algorithms

$f(n)$	$n = 2$	$2^4 = 16$	$2^8 = 256$	$2^{10} = 1024$	$2^{20} = 1048576$
log *n*	1	4	8	10	20
n	2	16	256	1024	1048576
*n**log n	2	64	2048	10240	20971520
n^2	4	256	65536	1048576	$\approx 10^{12}$
n^3	8	4096	16777200	$\approx 10^9$	$\approx 10^{18}$
2^n	4	65536	$\approx 10^{77}$	$\approx 10^{308}$	$\approx 10^{315653}$

Question

Given a TSP with n cities $c_1, \ldots, c_n$.

How many different tours exist that start in the first city c_1?

How many different tours?

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

Solution: Number of tours for TSP

Given a TSP with n cities $c_1, \ldots, c_n$.

If a tour always starts and ends in city c_1, then there exist $(n\text{-}1)!$ different tours.

Question

What is the **runtime** of the naive algorithm for TSP that just tries all potential tours?

Solution: Runtime of TSP

- *(n-1)!* different tours
- *n* additions per tour to compute its cost

==> Runtime is $O(n!)$

- Estimation of n! :

$n! = n * (n\text{-}1) * \dots * (\frac{n}{2}+1) * (\frac{n}{2}) * \dots * 2 * 1$

==> $(\frac{n}{2})^{\frac{n}{2}} \leq n! \leq n^n$

The Traveling Salesperson Problem

- Is one of the most-famous combinatorial optimization problems, and a showcase for many algorithmic approaches
- Is hard to solve («NP-hard»)
- Can be solved exactly, but run time is huge (super exponential in number of cities)
- No exact polynomial-time algorithm is known
- Can be be solved reasonably well with heuristic approaches

Complexity Theory

Complexity Theory in a Nutshell

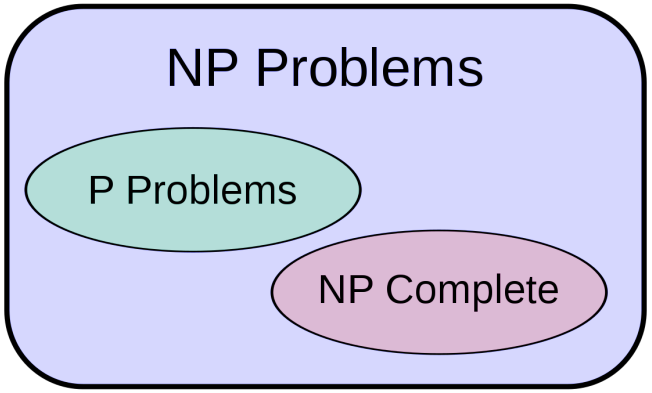
Class P: Problems that can be solved in polynomial time

Example: Sorting, Shortest Path, etc.

Class NP: Problems for which a given solution can be verified in polynomial time

Example: Decision problem of TSP: “Is there a solution for this TSP instance with total travel distance at most 2038?"

-> **P is a subclass of NP**



NP-Completeness

A Problem π is ***NP-complete*** if it is in NP and all other problems in NP can be "reduced" to π in polynomial time.

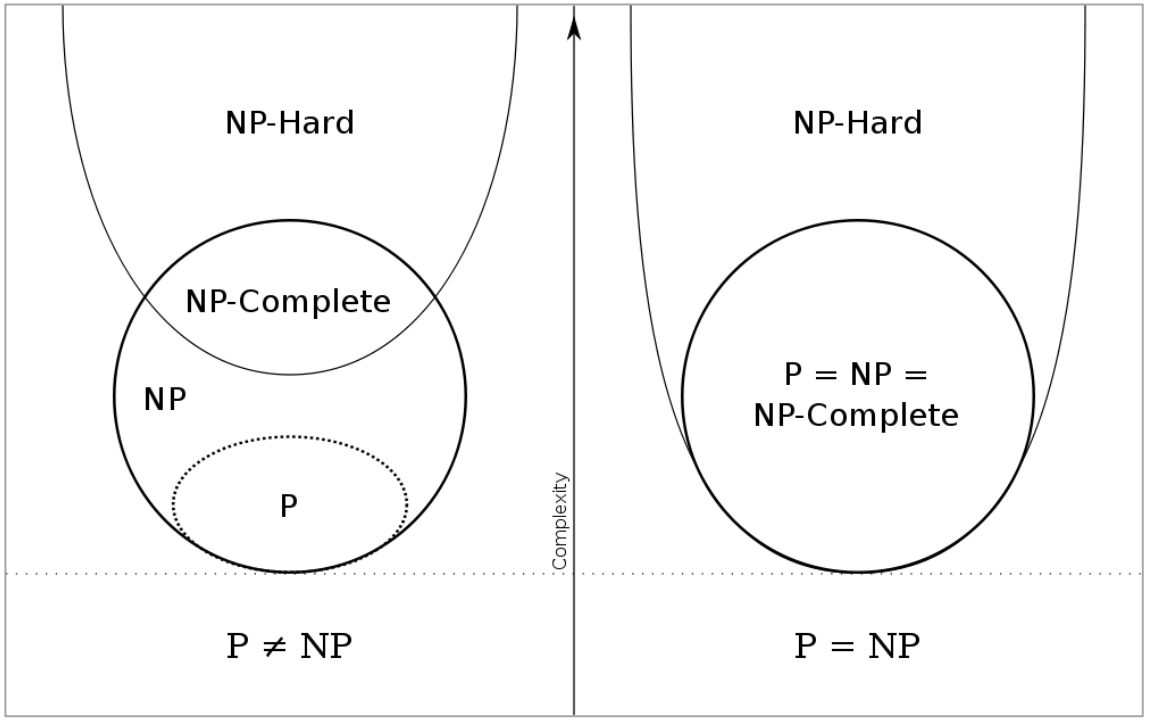
Remarks:

- This means, that an NP-complete problem is one of the hardest problems in NP
- Decision Problem of TSP is NP-complete
- More than 2000 NP-complete problems are known, e.g. 3-SAT, Knapsack, Subset Sum, Graph Coloring…
- No exact polynomial-time algorithm is known for any of these problems

P ≠ NP Conjecture

- **OPEN PROBLEM**: is P = NP?
- ***If* P ≠ NP, then no exact polynomial time algorithms exist for TSP and all other NP-complete problems**

→ Therefore, we need to find "other" approaches to solve these problems!



Heuristic

A heuristic method is based on **knowledge acquired by experience** on a given problem.

Heuristic methods do not necessarily provide the best possible solution. They are opposed to **exact algorithms**, that guarantee an optimal solution.

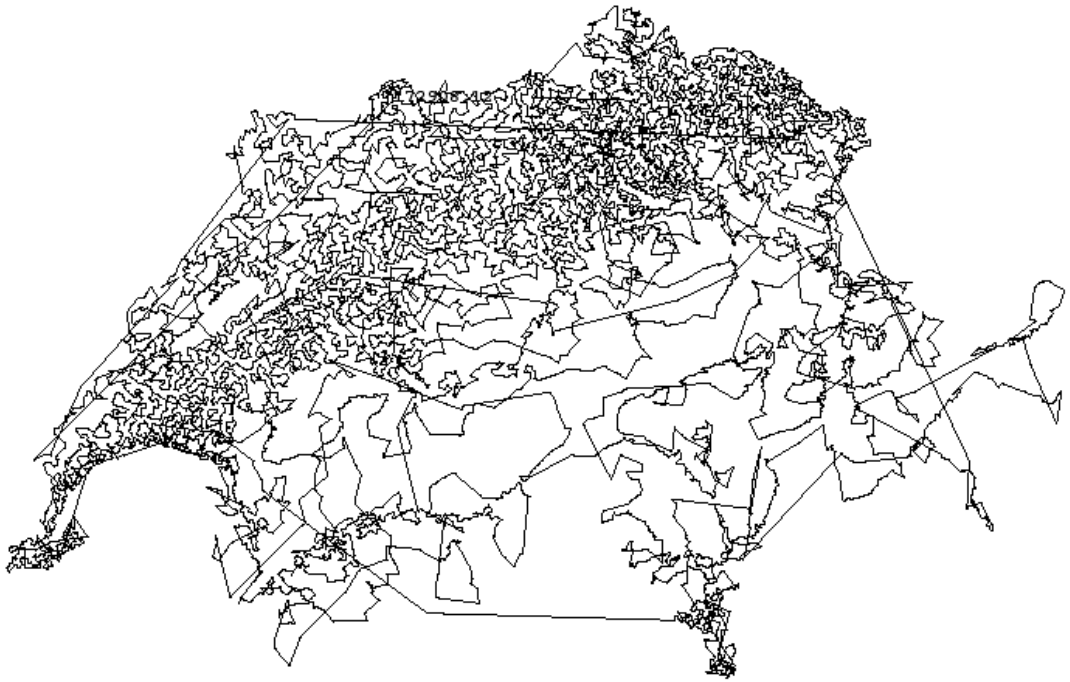
Example : The nearest neighbour heuristic method for the travelling salesperson problem

Start form city 1

Repeat

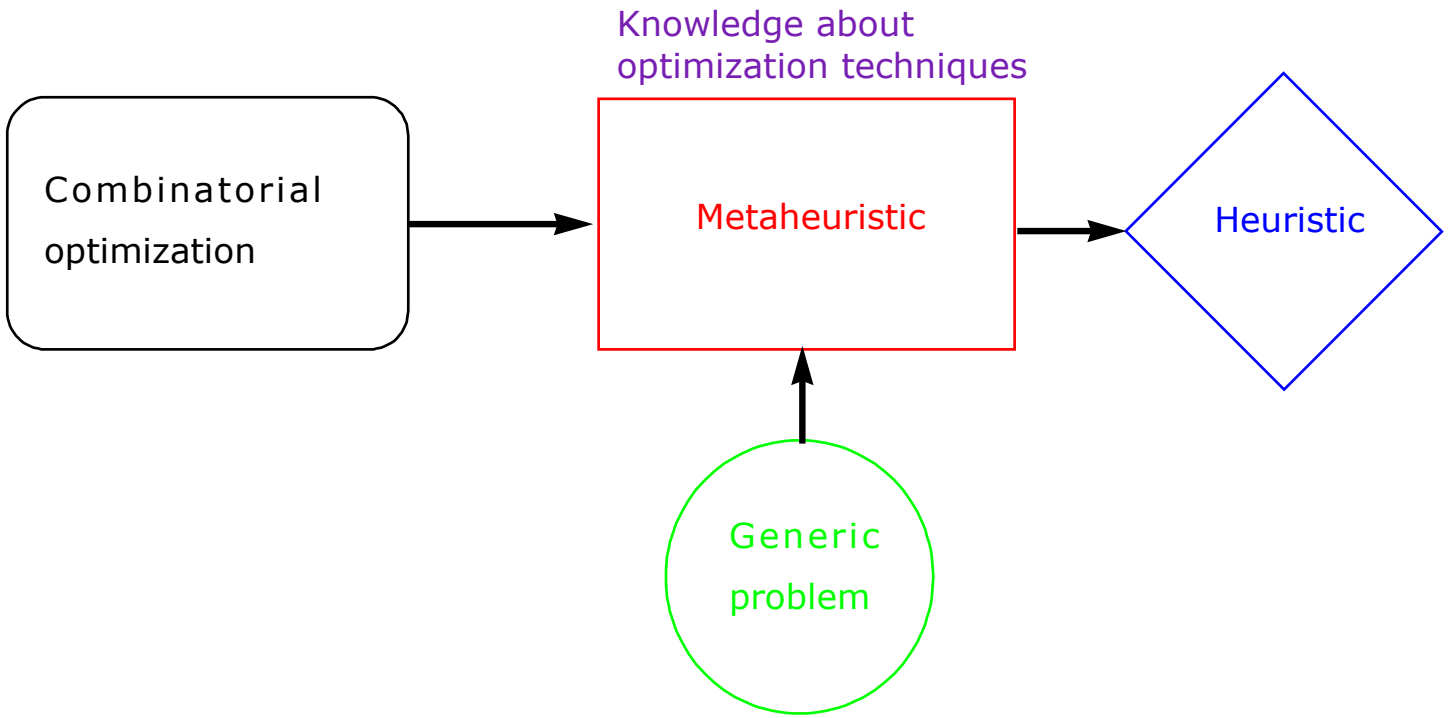
Go to the nearest city not yet visited

Until all cities are visited



Meta-Heuristic

A **meta-heuristic** is a conceptual procedure that can be adapted to a large set of combinatorial optimization problems thereby yielding heuristic methods.



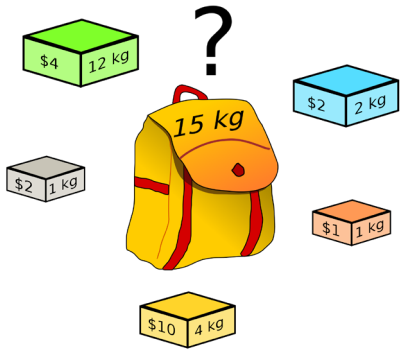
In this course, we will…

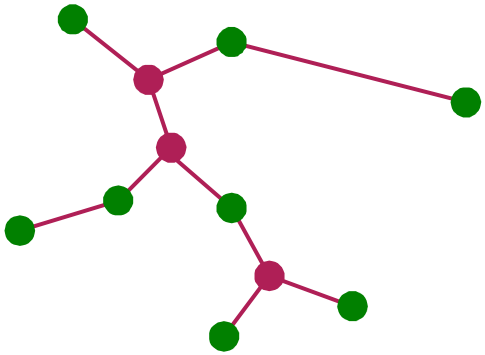
- … learn various approaches to tackle hard problems
- … implement solution approaches for several problems
- … learn how to compare the efficiency of such approaches

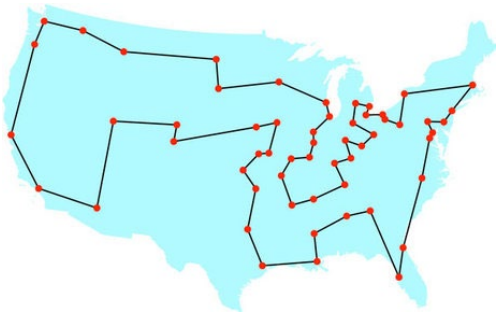
Sample Problems in this Course

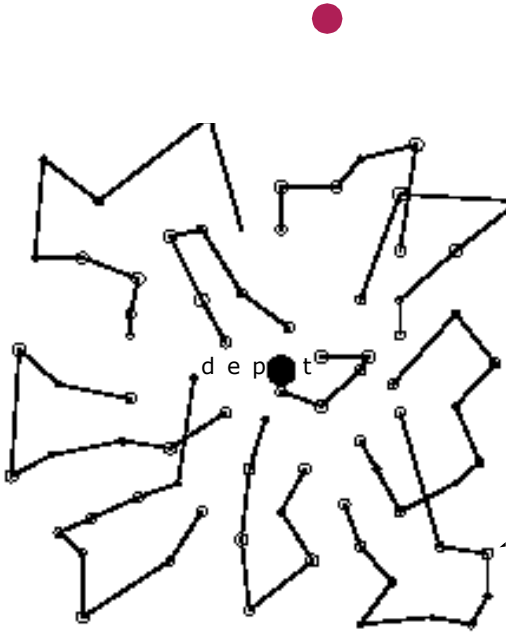
Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

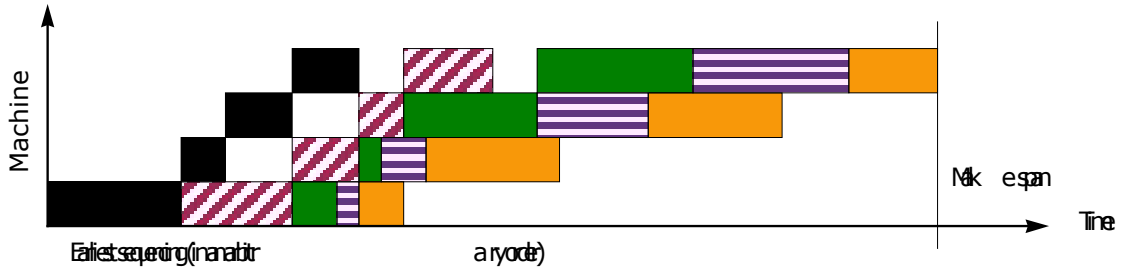
- Traveling Salesperson Problem
- Capacitated Vehicle Routing Problem
- Steiner Tree Problem
- Scheduling Problems
- Knapsack Packing Problem
- Santa Claus Problem





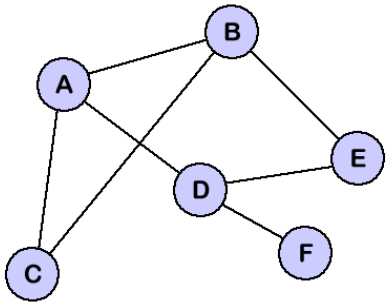


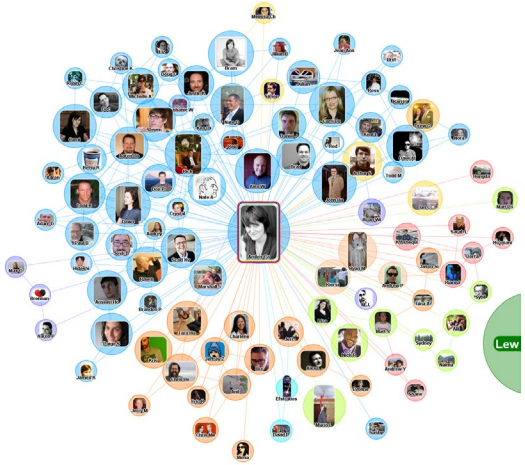


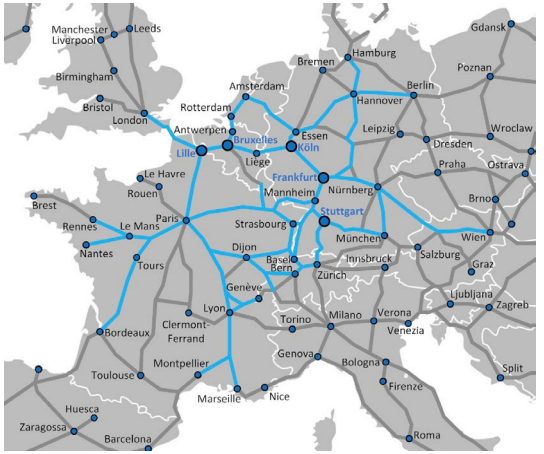


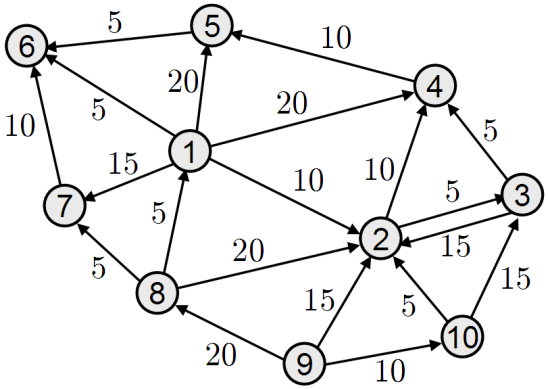
Graph Problems

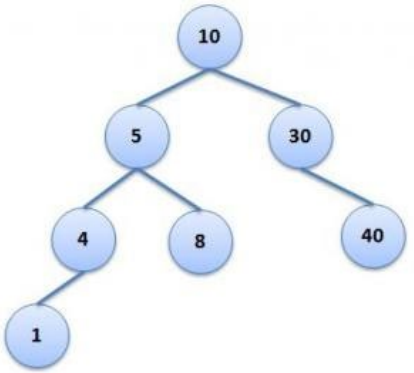
What is a Graph?







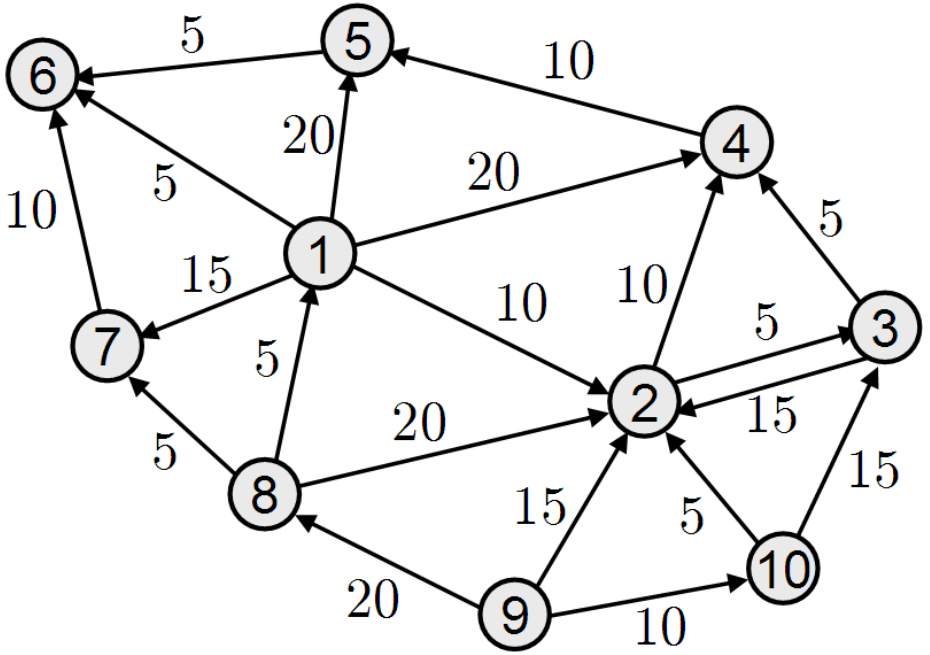




What is a Graph?

- A Graph is a tuple (V, E) with Vertices V and Edges E

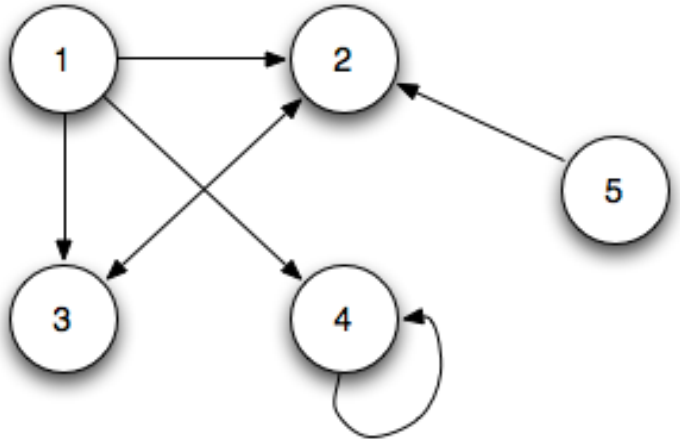
- Properties:
  - Directed or undirected
  - Weighted
  - Cyclic
  - Connected



Exercise

Given the following graph, denote its represention as

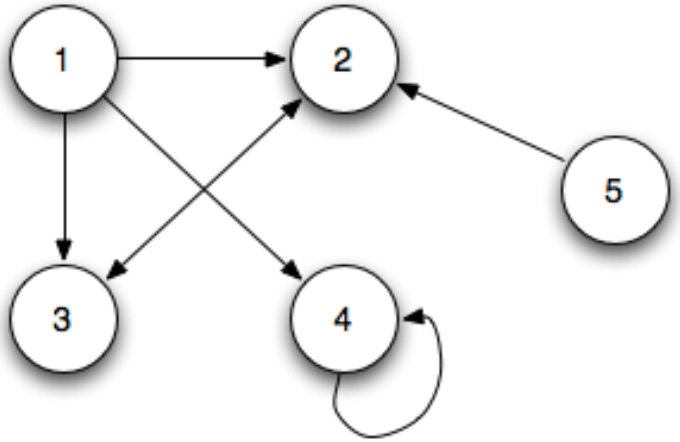
1. an adjacency matrix
2. an adjacency list
3. a vertex list
4. an edge list



Solution: Graph-Representations

1. Adjacency Matrix
2. Adjacency List: for each vertex the list of its neighbors
   - 1: 2,3,4
   - 2: 3
   - 3: 2
   - 4: 4
   - 5: 2
3. Vertex List:
   Number of vertices; Number of edges;
   For each vertex: number and list of neighbors.
   Example: 5,7,3,2,3,4,1,3,1,2,1,4,1,2
4. Edge List:
   Number of vertices; Number of edges;
   For each edge: start and end vertex
   Example: 5,7,1,2,1,3,1,4,2,3,3,2,4,4,5,2

from\to	1	2	3	4	5
1	0	1	1	1	0
2	0	0	1	0	0
3	0	1	0	0	0
4	0	0	0	1	0
5	0	1	0	0	0



Shortest Path Problem

Given

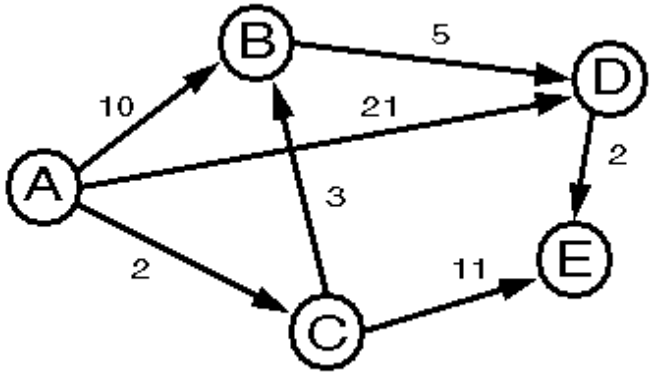
G = (V, E, C) a Graph with

Vertices V, |V| = n

Edges E, |E| = m

cost $c_{ij} \geq 0$ for each edge $e \in E$,

and a start vertex $s \in V$



Goal

Find the length of the shortest path from

s to j for ***each*** vertex $j \in V$

Shortest Path Problem: Dijkstra's Algorithm (1959)

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

Data:
$G = (V, E, C)$ with directed edge costs $c_{ij} \geq 0$
Vertex s (being a root of G)

Result:
λ_j : length of the shortest path from s to j
p_j: predecessor of j on the shortest path from s to j

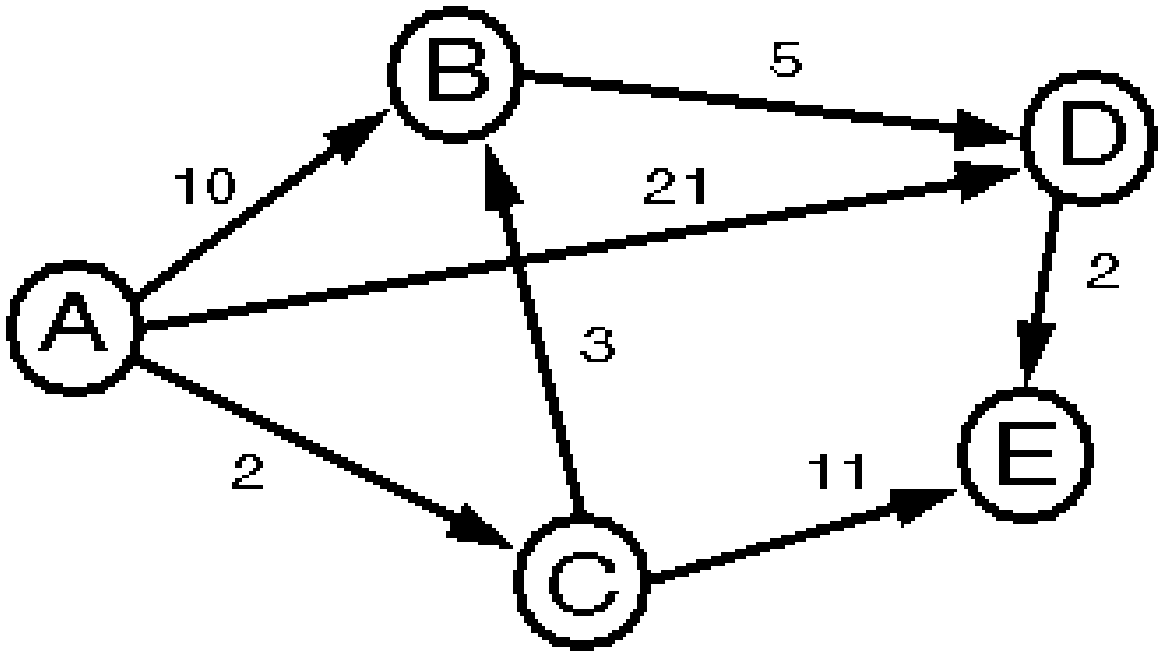
For all $j \in V$ **do** $\lambda_j = \infty$; $p_j = \emptyset$
$\lambda_s = 0$; $L = V$

While $L \neq \emptyset$
Find i such that $\lambda_i = \min(\lambda_k \mid k \in L)$
$L = L \setminus \{i\}$
For all $j \in succ[i]$ **do**
If $j \in L$ **and** $\lambda_j > \lambda_i + c_{ij}$
$\lambda_j = \lambda_i + c_{ij}$;
$p_j = i$
Return Lengths λ and predecessors p

Complexity

$O(|E| + |V| \log |V|)$ if L is managed with a Brodal queue

Example: Dijkstra's Algorithm



Problem: Minimum Spanning Tree (MST)

Data

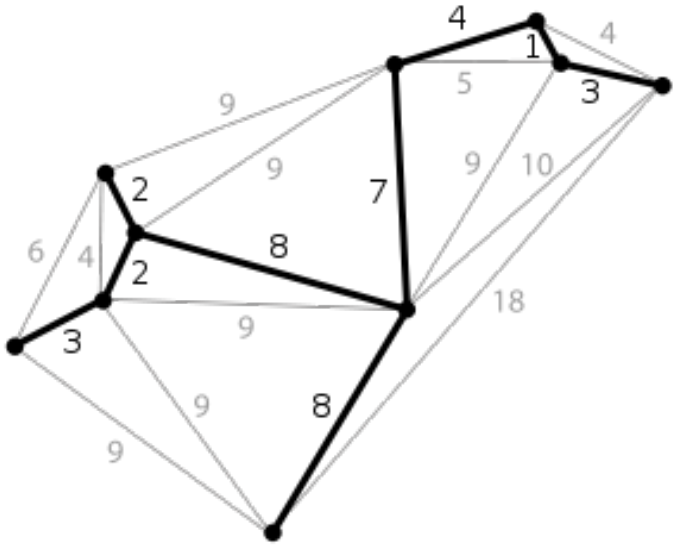
$G = (V, E, W)$ connected, vertices set V, undirected edges set E with weights given by function W,

$|V| = n$,

$|E| = m$

Result

Spanning tree $T = (V, E_T)$ of minimum weight



Minimum Spanning Tree: Prim's Algorithm (1957)

Data : $G = (V, E, W)$ connected
Result : Spanning tree $T = (V, E_T)$ of minimum weight

For each vertex $u \in V$ **do**
    $\lambda[u] = \infty$ //Weight for inserting u in T
    $p[u] = \emptyset$ //Predecessor of u in T
Chose a vertex s and set $\lambda[s] := 0$
$E_T = \emptyset$
$L = V$ //List of vertices not yet in T

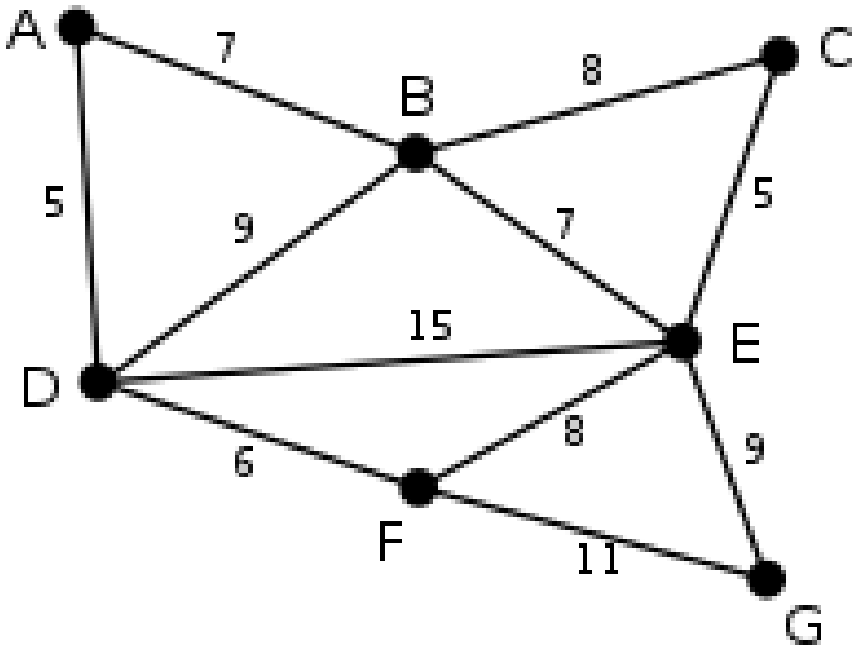
While $L \neq \emptyset$ **do**
    Remove from L the vertex u with lower $\lambda[u]$
    If $u \neq s$ **then**
        $E_T := E_T \cup \{p[u], u\}$
    For each vertex $v \in adj[u]$ **do**
        If $v \in L$ **and** $\lambda[v] > c_{uv}$ **then**
            $\lambda[v] = c_{uv}$
            $p[v] = u$

Return $T = (V, E_T)$

Complexity

$O(|E| + |V| \log |V|)$ if L is managed with a Brodal queue

Example: Prim's Algorithm



Minimum Spanning Tree: Kruskal's Algorithm (1956)

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
IAMP Institut für Angewandte Mathematik und Physik

Input: $G = (V, E, W)$ connected, vertices set V, undirected edges set E with weights given by function W, $|V| = n$, $|E| = m$

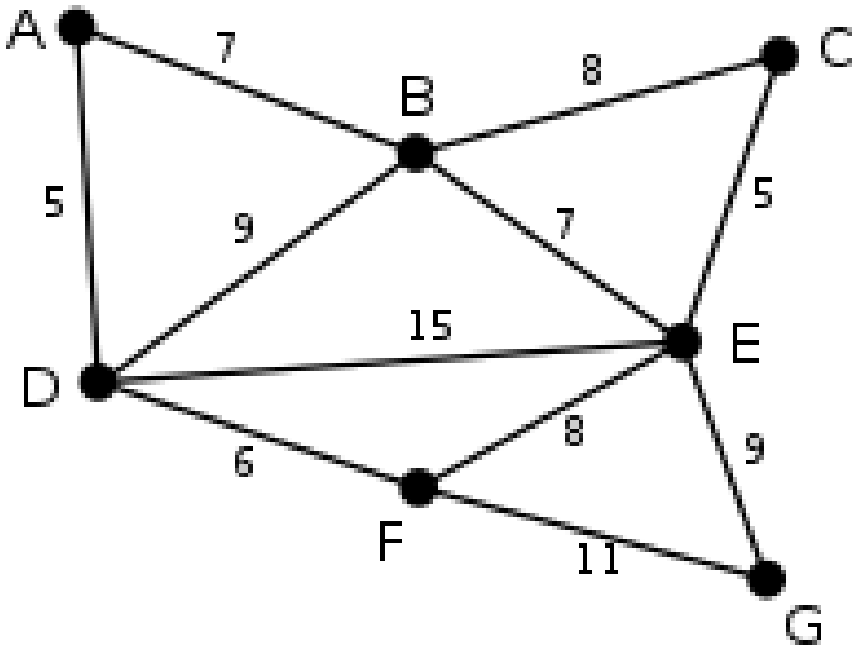
Algorithm:

Sort the edges of G by non-decreasing weight $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_m)$
Set $E_T = \varnothing$
For $k = 1 \ldots m$, **do**:
    If $E_T \cup \{e_k\}$ has no cycle, **Set** $E_T = E_T \cup \{e_k\}$
Return $T = (V, E_T)$

Remarks

- The algorithm can be stopped as soon as $| E_T | = n - 1$
- Complexity : $O(| E | \log | E |)$ with "Union-Find-Datastructure

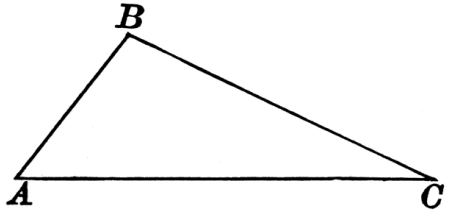
Example: Kruskal's Algorithm



Question

Given a graph G with ***triangle inequality***.

Triangle inequality: Going from A to B to C is not shorter than going directly from A to C

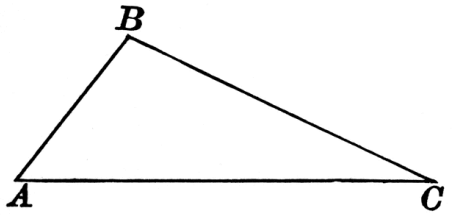


Which is shorter (sum of all edge lengths): An *optimal tour for TSP* or a *Minimum Spanning Tree*?

Which is shorter?

Question

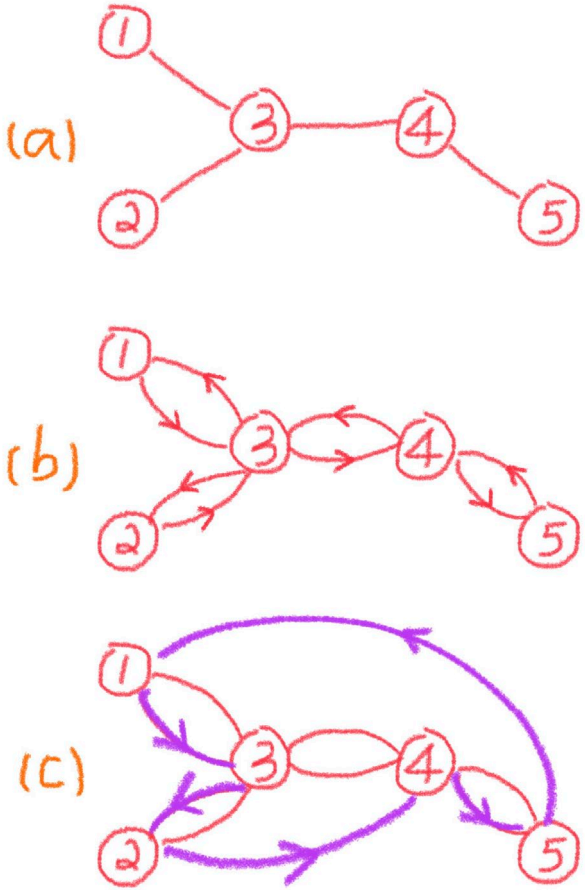
Given a graph G with ***triangle inequality***.



You are given an MST with weight w for the graph. How can you construct a TSP tour of length at most 2*w from this?

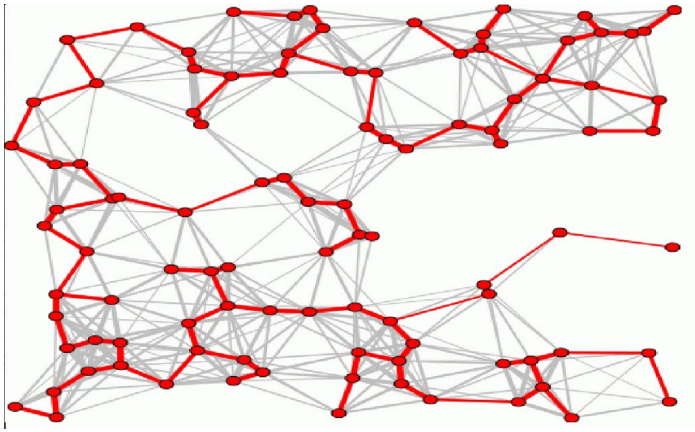
Solution: TSP Tour from MST

Solution Sketch:



Application of MST

1. Network Design
   Water pipes, electricity cables,
   chip design etc.



2. Approximation Algorithm for TSP
   A "duplicated" MST is a 2-Approximation for TSP with triangle inequality

Generic Definition of Optimization Problems

Minimize $f(s)$

subject to $s \in S$

Where :

f	Objective function
s	Solution
S	Set of feasible solutions

Examples:

- Minimum Spanning Tree (MST)

S	Sub-graph connected and without cycle (set of trees)
s	A given tree
$f(s)$	Sum of the weight of the edges of tree s

- The Travelling Salesperson Problem (TSP)

S	Set of all tours (cycles passing exactly once through each node)
s	A given tour (given, e.g. by a permutation of the cities)
$f(s)$	Length of a given tour

Scheduling Problems

Scheduling Problem

Given:

n jobs {1..n} and *m* machines {1..m}

A job *i* consists of *m* tasks

The j^{th} task of each job must be processed on machine *j*

A job can start on machine *j* only if

It is completed on machine *j* – 1 and

If machine *j* is free

Each task *i* has a known processing time p_{ij} on machine *j*

Goal:

Find a "good" (the optimal) order in which the jobs should be processed

Scheduling: Optimization Objectives

Objectives

- **Makespan**: Minimize the completion time of the last job
- **Sum of Completion Times**: (weighted) sum of completion times of all jobs
- **Minimum Tardiness**: given that each job has an individual due date d_i, minimize the (weighted) sum of tardinesses of all jobs
- …and many more

Scheduling Work Sequences

Permutation Flow Shop Problem

The ordering in which the tasks of each job are processed is given, AND on each machine the jobs have to be processed in the same ordering (jobs cannot pass other jobs).

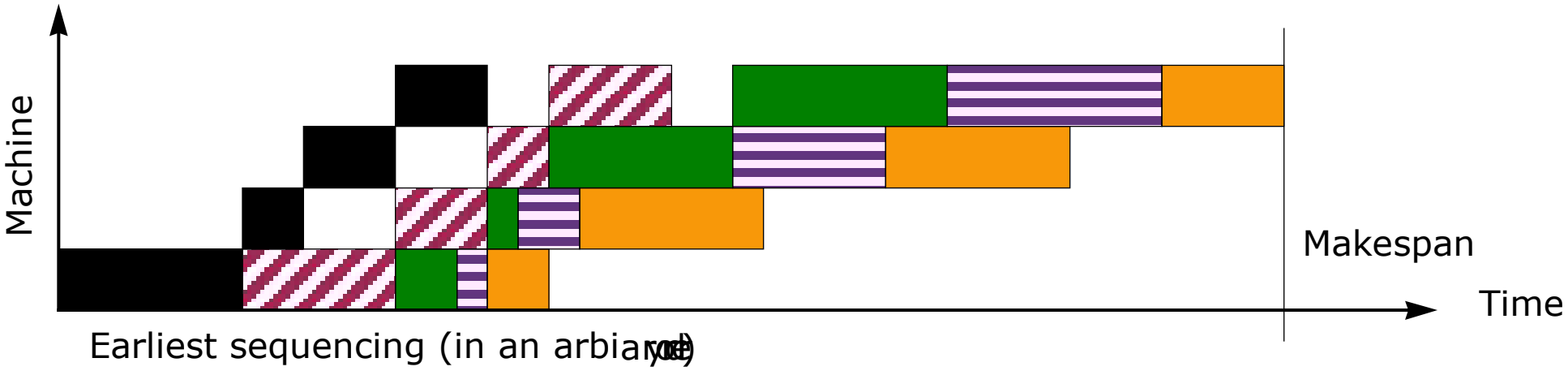
Flow Shop Problem

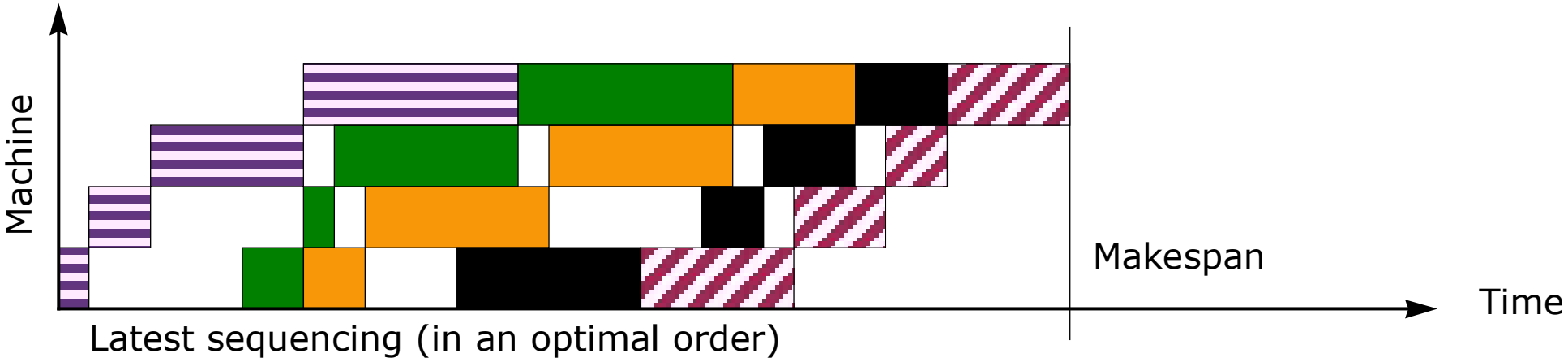
The ordering in which the tasks of each job are processed is given, but jobs can have different orderings on different machines.

Job Shop Problem (most flexible)

Each job can be processed in an individual ordering of its tasks.

Gant Chart


Machine
Makespan
Time
Earliest sequencing (in an arbiarde)


Machine
Makespan
Time
Latest sequencing (in an optimal order)

Solving Scheduling Problems

- A majority of Scheduling Problems are NP-hard
- Trivial approach: compute all allowed job placements on different machines
  -> huge computation time
- There are heuristics for constructing good solutions

Lecture 1: Summary

- Asymptotic runtime is used to measure performance of algorithms
- Many problems are NP-complete, which means that there is no known polynomial-time algorithm to solve the problems exactly
- Heuristics can create "good" approximations to solutions for such problems
- Meta-heuristics allow to create new heuristics for a problem