

Table of Contents

Introduction to Combinatorial Optimization

1 Course organization

Zdeněk Hanzálek

zdenek.hanzalek@cvut.cz

Acknowledgments to: Přemysl Šúcha, Jan Kelbel
Jiří Trdlická, Zdeněk Baumelt, Roman Čapek

CTU in Prague

February 19, 2019

Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	1 / 49
Labs		Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization

Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	1 / 49
Lectures		Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization

1 Introduction of Basic Terms, Example Applications. Test 0.

2 Integer Linear Programming - Algorithms.

3 Problem Formulation by Integer Linear Programming.

4 Shortest Paths.

5 Problem Formulation by Shortest Paths.

6 Flows and Cuts - Algorithms and Problem Formulation. Test I.

7 Bipartite Matching. Multi-commodity Network Flows.

8 Knapsack Problem, Pseudo-polynomial and Approximation Algs.

9 Traveling Salesman Problem and Approximation Algorithms.

10 Mono-processor Scheduling.

11 Scheduling on Parallel Processors. Test II.

12 Project Scheduling with Time Windows.

13 Constraint Programming.

- 1 - motivation; 4,5,6,7 - mostly polynomial complexity
- 8,9,10,11,12 - NP-hard problems
- 2,3,13 - declarative programming techniques

Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	2 / 49
Labs		Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization

Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	3 / 49
Labs		Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization

Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	4 / 49
Labs		Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization

Course Organization

What is Combinatorial Optimization?



Optimization is a term for the mathematical discipline that is concerned with the minimization/maximization of some objective function subject to constraints or decision that no solution exists.

- **Combinatorics** is the mathematics of discretely structured problems.
- **Combinatorial optimization** is an optimization that deals with discrete variables.

It is very similar to **operation research** (a term used mainly by economists, originated during WW II in military logistics).



<https://cw.fel.cvut.cz/wiki/courses/ko/start>

- Courses
- Labs
- Project
- Classification
- Literature

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 5 / 49

Application Areas



Many real-life problems can be formulated as combinatorial optimization problems.

Typical application areas:

- Production (production speed up, cost reduction, efficient utilization of resources...)
- Transportation (fuel saving, reduction of delivery time...)
- Employees scheduling (reduction of human resources...)
- Hardware design (acceleration of computations...)
- Communication network design (end-to-end delay reduction...)
- ...

This course will help you to understand and solve such problems:

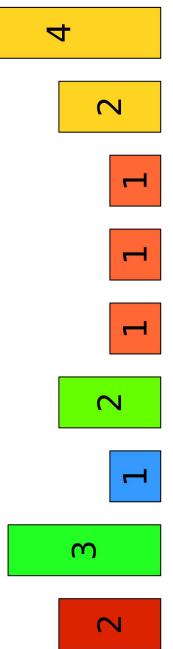
- Formalization - transformation to a known optimization problem
- Algorithms - the problem solution

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 6 / 49

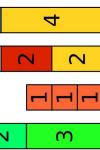
Bin Packing Problem

Indivisible objects of different size and bins of equal capacity.
Goal:

- Assign the objects to the bins, minimize the number of bins.



- Different versions of the problem:
- Bins of different sizes {3,5}
 - Minimize the number of bins for each color Result:

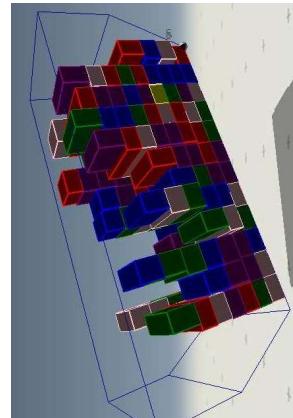


Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 8 / 49



Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 8 / 49

Container Loading



- Goal:
- To store as much boxes as possible in a container.

Constraints:

- size of the container
- sizes of the boxes
- loading process
- stability, orientation of the boxes
- requested order of the boxes when unloading the cargo

Similar problems: **2-D cutting**, **Guillotine 2-D cutting**.

Z. Hanzálek (CTU) February 19, 2019 9 / 49

Assignment of Shifts to Employees

Assignment of Shifts to Employees



Goal:

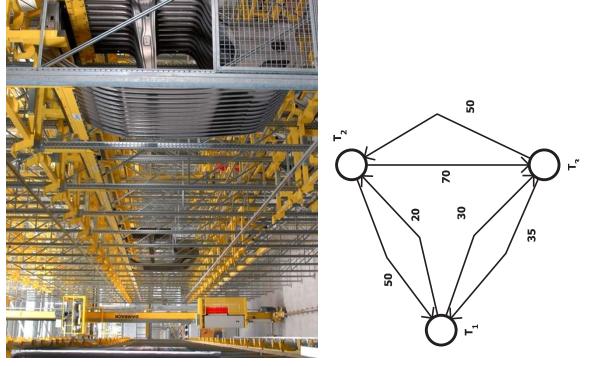
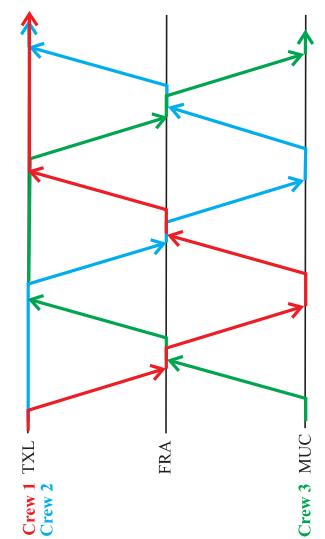
- create acceptable shift roster,
so that all required shifts are assigned
- Constraints:
- qualification of employees
 - labor code restrictions (e.g. at least 12 hours of rest during 24 hours)
 - collective agreement restrictions (e.g. maximum number of night shifts in a block)
 - employees demands (e.g. required day-off)
 - fair assignment of shifts (same number of weekend shifts)

Z. Hanzálek (CTU) February 19, 2019 10 / 49

Storage System in Automated Warehouse

Assignment of human resources is often formalized as a **matching in a bipartite graph**

The problem becomes harder when we consider the geographic position of the employees (e.g. stewards and pilots in an airline company).



Goal:

- reduce time of the vehicle trips

The problem is represented by the digraph. The nodes of the graph represent the picking tasks. Edge (i, j) means the possibility to perform task j right after i . The edge cost represents the duration of the trip from i to j .

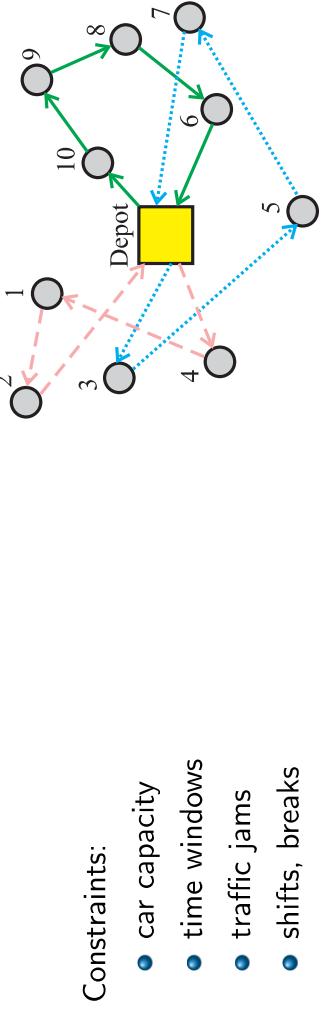
Can be formulated as
Asymmetric Traveling Salesman Problem.

Vehicle Routing



Goods, customers and fleet of cars.

- Goal:
- fulfill demands of customers
 - minimize transportation cost



Constraints:

- car capacity
- time windows
- traffic jams
- shifts, breaks

Problem can be divided into two subproblems.

Z. Hanzák (CTU) Introduction to Combinatorial Optimization February 19, 2019 13 / 49

Surface Mount Technology

Surface Mount Technology

The placement machines are scarce resource of the Printed Circuit Board (PCB) production, due to their high cost.

Goal:

- Maximize production speed

Constraints:

- Assembly line configuration
- Description of produced PCB

Problem can be divided into two subproblems.

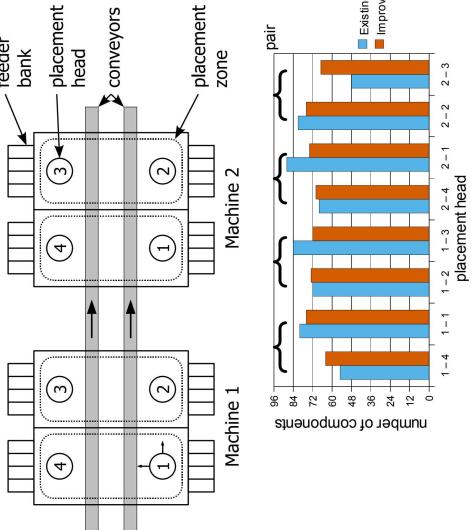
Z. Hanzák (CTU) Introduction to Combinatorial Optimization February 19, 2019 14 / 49

Surface Mount Technology

A) Allocation of the components to the placement heads

Can be formulated as an **Assignment Problem**

- Input:
- types of SMT components
 - number of components of a given type
 - precedence relations among some components
 - machine parameters

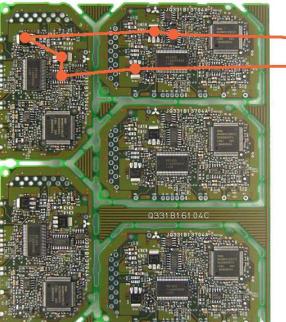


- Output:
- allocation of components to the placement heads

assembly sequence
operation time

B) Sequencing for a given head can be formulated as a (capacitated multi-trip) **Traveling Salesman Problem**

- Input:
- allocation of components to the assembly head
 - position of components on the PCB
 - capacity of the head



Output:

- assembly sequence
operation time

Z. Hanzák (CTU) Introduction to Combinatorial Optimization February 19, 2019 15 / 49

Z. Hanzák (CTU) Introduction to Combinatorial Optimization February 19, 2019 16 / 49

Routing in Wireless Sensor Network - Specification

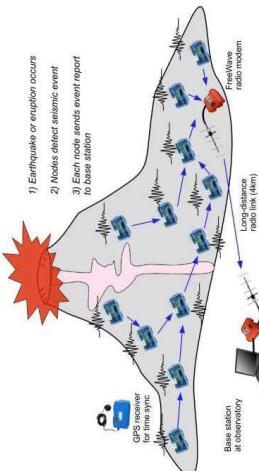
Routing in Wireless Sensor Network - Formalization

A volcano monitoring using autonomous devices equipped with:

- own power supply
- wireless short range communication
- temperature sensors

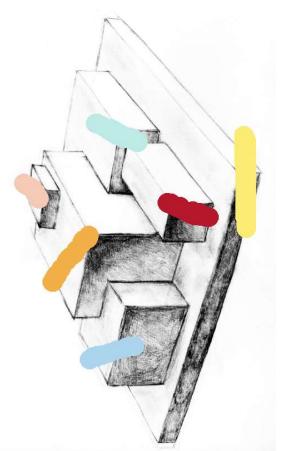
Goal:

- create routing tables
 - minimize energy consumption
- Constraints:
- capacity of each communication link
 - limited transmitter performance
 - maximal allowed end-to-end delay of communication
 - memory capacity of devices



Coloring Book - Specification

The aim is to color hand-drawn black-and-white image with minimal effort and still achieve accuracy comparable to precise manual pixel-wise coloring.



Problems:

- color leakage through small gaps in outlines
- too many small regions
- color flooding into outlines (zapít barvu do vnitřku kontury)
- robustness to inaccuracy (přetahy)

Can be formalized as a **Multi-Commodity Network Flows problem**.

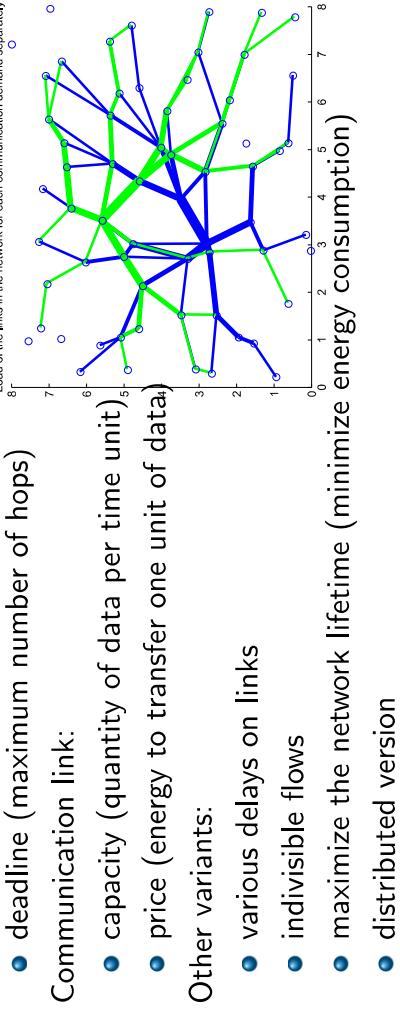
WSN represented by a graph (devices = vertices, links = edges)

Communication demand = commodity flow:

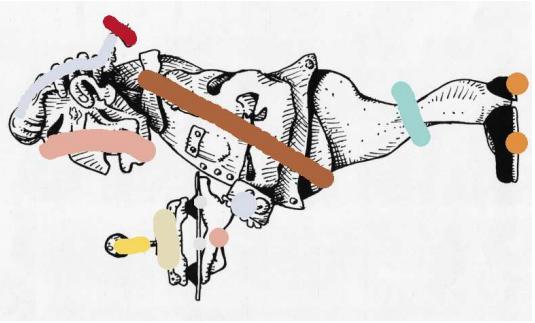
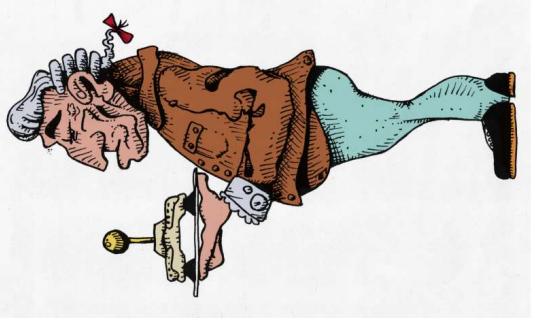
- source devices and destination devices

volume of demand (quantity of data per time unit)

deadline (maximum number of hops)



Used by illustrators and for production of cartoon animations like Rumcajs, Dr. Animo, and Husiti.



Coloring Book - Application

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 18 / 49

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 17 / 49

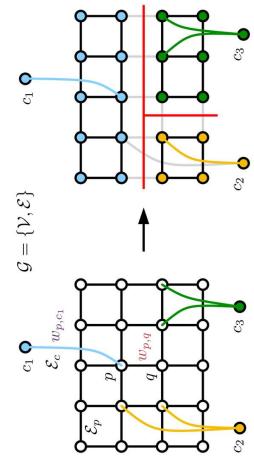
Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 18 / 49

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 19 / 49

Coloring Book Formulated as Multiway Cut Problem

Multiway cut (given a graph with a set of terminals, find the minimum-weight set of edges whose removal separates each pair of terminals) is NP-hard problem.

- terminal nodes with different colors are specified by the user
- edge capacity is high (two neighbor pixels are white) or low (two neighbor pixels are black)
- a cut is the boundary between two colors



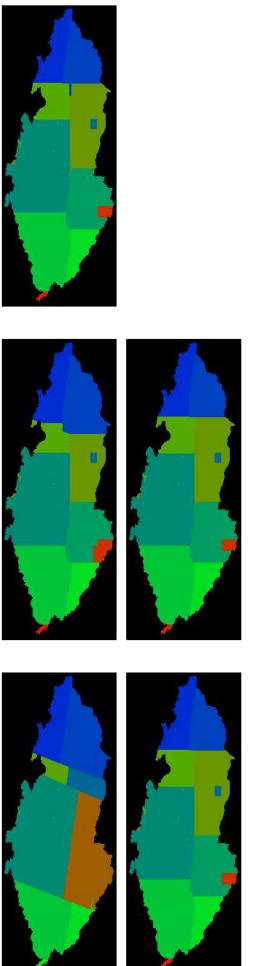
It can be approximated by multiple binary minimum cut problems which are polynomial. GridCut U.S. patent by Jamříška and Sýkora, DCGI.

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 21 / 49

Region Covering with Satellite Images

Generic ILP solvers are too slow. The problem was solved by an approximation algorithm, which starts from solution obtained by greedy set-cover algorithm (neglecting criterion b)

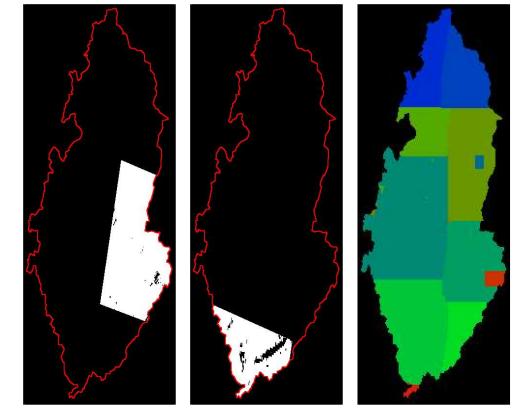
- iteratively improves the solution by solving a minimum cut problems in each iteration as shown below.



The problem was solved as an industrial project for the geo-information company GISAT s.r.o. by Boris Flach, CMP.

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 23 / 49

Region Covering with Satellite Images



Input: Large database with geo-referenced satellite images and visibility masks (clouds!) including time-stamps. Customer defined spatial region (ROI) & time interval.

Output: set of images s.t. each pixel of the ROI is visible in one image.

Objective function: weighted combination of the two criteria:

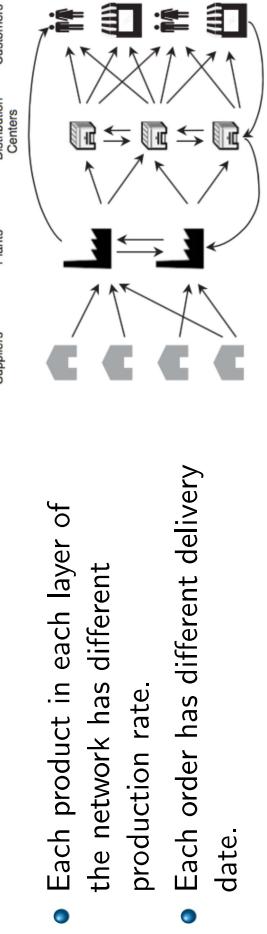
- minimize number of used images (i.e., set-cover)
- penalize time difference across image borders

Two visibility masks and final cover (similar colours encode similar dates)

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 22 / 49

Supply Network Management

Given a prediction of customers demands, the aim is to plan **production** and **transport** of products in order to maximize the profit.



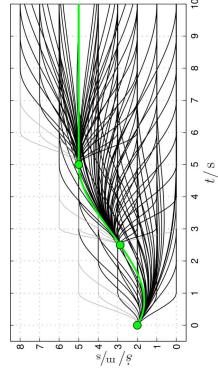
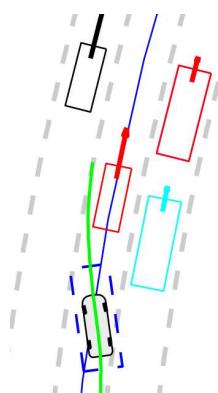
A solution is the amount of products in time, such that the profit is maximized and the risk of losses (unsold products) is mitigated.

Current project funded by an IT company: solve a supply network management problem given **historical data** of a customer.

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 24 / 49

Trajectory Panning for Autonomous Vehicle

Given a manoeuvre to be performed the aim is to find an optimal trajectory such that the constraints given by vehicle dynamics are respected.



Project work:

- reaction to dynamic changes
- reliability of perception
- data fusion
- computing power vs reliability



People have driving schools, robots have Combinatorial Optimization.

Z. Hanzálek (CTU)

February 19, 2019

25 / 49

Configurable HW for Specific DSP Application Design

Can be formalized as a cyclic extension of $PS|temp|C_{max}$

Used hardware features	unit	count [-]	computation time [c/k]
ADD	ADD1	2	1
MUL	ADD2	1	2



Explain the typical goals of optimization:

- automation of the design/decision process
- increase the volume of the production (shorter production-line cycle)
- cost reduction (fuel saving, less machines)
- risk reduction (error elimination due to automated creation of production schedule)
- lean manufacturing (supply and stores reduction, outgrowths reduction when delay in supply)
- increase of the flexibility (faster reaction to structure or constraint change)
- user-friendly solutions (balanced schedule for all employees)



Lattice Wave Digital Filter (LWDF)	X(k)	Y(k)
	X(k)	

Configurable HW for Specific DSP Application Design

Configurable HW for Specific DSP Application Design

Z. Hanzálek (CTU)

February 19, 2019

26 / 49

Convincing Arguments

Used hardware features	unit	count [-]	computation time [c/k]
ADD	ADD1	2	1
MUL	ADD2	1	2



Lattice Wave Digital Filter (LWDF)	X(k)	Y(k)
	X(k)	

How does this course cover the important skills?

An engineer is usually hired to systematically solve the problem.

Skill	Example
Business case Specification	Attract a customer Production scheduling
Formalization	Flow-shop
Algorithms	Johnson's algorithm
Prototype solution	Matlab OPL, ...
Implementation	C#, dB, ...

Z. Hanzák (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	29 / 49
<h2>How does this course cover the important skills?</h2>			

An engineer is usually hired to systematically solve the problem.

Skill	Example	Lectures	Seminars	Exam
Business case Specification	Attract a customer	-	Project?	-
Formalization	Production scheduling	Application examples	Project Exercises	Project
Algorithms	Flow-shop	Formulation of the opt. problem	Project Exercises	Project Exercises
Prototype solution	Pseudocode	Pseudocode	Project	Project
Implementation	Johnson's algorithm	iteration with data	Project Exercises	Project Exercises
	Matlab OPL, ...	-	Project	Project Prac. Test
	C#, dB, ...	-	Project?	-

How does this course cover the important skills?

An engineer is usually hired to systematically solve the problem.

Skill	Example
Business case Specification	Attract a customer Production scheduling
Formalization	Flow-shop
Algorithms	Johnson's algorithm
Prototype solution	Matlab OPL, ...
Implementation	C#, dB, ...

Z. Hanzák (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	29 / 49
<h2>How does this course cover the important skills?</h2>			

An engineer is usually hired to systematically solve the problem.

Skill	Example	Lectures	Seminars	Exam
Business case Specification	Attract a customer	-	Project?	-
Formalization	Production scheduling	Application examples	Project Exercises	Project
Algorithms	Flow-shop	Formulation of the opt. problem	Project Exercises	Project Exercises
Prototype solution	Pseudocode	Pseudocode	Project	Project
Implementation	Johnson's algorithm	iteration with data	Project Exercises	Project Exercises
	Matlab OPL, ...	-	Project	Project Prac. Test
	C#, dB, ...	-	Project?	-

Algorithms are used to solve the problems

Time Complexity of Algorithms

n	$100n \log n$	$10n^2$	$n^{3.5}$	$n^{\log n}$	2^n	$n!$
10	3 μ s	1 μ s	3 μ s	2 μ s	1 μ s	4 ms
20	9 μ s	4 μ s	36 μ s	420 μ s	1 ms	76 years
30	15 μ s	9 μ s	148 μ s	20 ms	1 s	$8 \cdot 10^{15}$ y.
40	21 μ s	16 μ s	404 μ s	340 ms	1100 s	
50	28 μ s	25 μ s	884 μ s	4 s	13 days	
60	35 μ s	36 μ s	2 ms	32 s	37 years	
80	50 μ s	64 μ s	5 ms	1075 s	$4 \cdot 10^7$ y.	
100	66 μ s	100 μ s	10 ms	5 hours	$4 \cdot 10^{13}$ y.	
200	153 μ s	400 μ s	113 ms	12 years		
500	448 μ s	2.5 ms	3 s	$5 \cdot 10^5$ y.		
1000	1 ms	10 ms	32 s	$3 \cdot 10^{13}$ y.		
10^4	13 ms	1 s	28 hours			
10^5	166 ms	100 s	10 years			
10^6	2 s	3 hours	3169 y.			
10^7	23 s	12 days	10^7 y.			
10^8	266 s	3 years	$3 \cdot 10^{10}$ y.			
10^{10}	9 hours	$3 \cdot 10^4$ y.				
10^{12}	46 days	$3 \cdot 10^8$ y.				

It is not always easy to find the optimal solution efficiently.
In the case of exhaustive search while enumerating all solutions, the computation time for bigger instances can be enormous.
For example, the permutation Flow-shop problem has complexity of $n!$, where n is the number of jobs.

Z. Hanzálek (CTU) February 19, 2019 30 / 49

Graph Theory Overview

Z. Hanzálek (CTU) February 19, 2019 31 / 49

Directed Graph

Graphs informally:

- A graph consists of nodes and edges.
- Each edge joins two nodes, it is directed or undirected.
- In a directed graph, the edge leaves one node and enters another one.
- In an undirected graph, an edge is a symmetric join of two nodes.

Directed graph (digraph) is a triplet (V, E, Ψ) :

- V is a finite set of nodes
- E is a finite set of directed edges
- Ψ is a mapping from the set of edges to the ordered pair of nodes, i.e.
- $\Psi : E \rightarrow \{(v, w) \in V \times V : v \neq w\}$

- Two edges e, e' are called **parallel edges** if $\Psi(e) = \Psi(e')$.
- A graph containing parallel edges is called a **multigraph** - we usually do not work with a multigraph.
- A graph that does not contain parallel edges is called **simple graph**.
- We usually denote simple graphs by pair $G = (V(G), E(G))$, where $V(G)$ is a set of nodes and $E(G)$ is set of (ordered) pairs of nodes that describes the edges (i.e. we identify an edge with its image $\Psi(e)$).
- Undirected edge $e = \{v, w\}$ or directed edge $e = (v, w)$ connects v and w . Nodes v and w are the endpoints of e or we can say they are incident with e . In case of a directed graph we say that e leaves v and enters w .

Comparison of Graphs

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 34 / 49

Comparison of Graphs

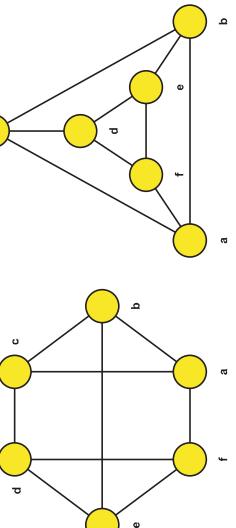
For directed graph G we can have an **underlying undirected graph** G' , with the same set of vertices and undirected edge $\{v, w\}$ for every directed edge (v, w) from G . We call graph H a **subgraph** (podgraf) of graph G , if we can create it by omitting the nodes (zero or more of them) or edges (when an edge is in the subgraph, its endpoints must be there as well). Special cases of subgraphs:

- Subgraph H of G is called **spanning** (faktor) if we omit some or zero edges and $V(H) = V(G)$.
- Graph H is called **subgraph induced by set of vertices** $V(H) \subseteq V(G)$ if we omit some (or zero) nodes and incident edges, i.e. H contains all edges from G whose both endpoints are in $V(H)$.

Two graphs G and H are called **isomorphic** if there are bijections: $\phi_V : V(G) \rightarrow V(H)$ and $\phi_E : E(G) \rightarrow E(H)$ such that:

- for directed graphs: $\phi_E((v, w)) = (\phi_V(v), \phi_V(w))$ for all $(v, w) \in E(G)$
- for undirected graphs: $\phi_E(\{v, w\}) = \{\phi_V(v), \phi_V(w)\}$ for all $\{v, w\} \in E(G)$

We usually don't deal with isomorphic graphs when talking about algorithms, but it is good to be aware of them during modelling.



Comparison of Graphs – Isomorphic Graphs

February 19, 2019 35 / 49

Comparison of Graphs – Isomorphic Graphs

February 19, 2019 35 / 49

Other Terms for Digraphs

Other Terms for Digraphs

For node v of digraph G we define:

- a set of **successors** of v is a set of nodes such that there is an edge from v to each of these nodes, i.e. $\{w \in V : (v, w) \in \Psi(E)\}$
- a set of **predecessors** v is a set of nodes such that there is an edge from each of these nodes to v , i.e. $\{w \in V : (w, v) \in \Psi(E)\}$
- $\Gamma(v)$, a set of **neighbors** of v , is set of nodes connected by an edge with v , i.e. a union of successors and predecessors
- $\delta^+(v)$, a set of edges leaving v
- $\delta^-(v)$, a set of edges entering v
- $\delta(v)$, a set of edges incident with v
- $|\delta^+(v)|$, **out-degree**
- $|\delta^-(v)|$, **in-degree**
- $|\delta(v)|$, **degree of node**

Notice: $\sum_{v \in V(G)} |\delta(v)| = 2|E(G)|$.

the number of nodes with an odd degree is even

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 36 / 49

Similar Terms for an Undirected Graph

These terms are used in undirected graphs:

- $\Gamma(v)$, a set of neighbors of node v
- $\delta(v)$, a set of edges incident with v
- $|\delta(v)|$, a degree of v
- $E(X, Y)$, a set of edges between X and Y , i.e.
- $E(X, Y) = \{(x, y) \in E(G) : x \in X \setminus Y, y \in Y \setminus X\}$
- $\delta(X)$, a set of "border" edges of set X , i.e. $\delta(X) := E(X, V(G) \setminus X)$
- $\Gamma(X)$, set of neighbours of set X , i.e.
- $\Gamma(X) := \{v \in V(G) \setminus X : E(X, v) \neq \emptyset\}$

Mostly, we will use digraphs (from an undirected graph we can make a digraph by replacing every undirected edge by a pair of inverse directed edges).

- For sets $X, Y \subseteq V(G)$ of directed graph G we define:
- $E^+(X, Y)$ a set of edges from X to Y , i.e.
 - $E^+(X, Y) = \{(x, y) \in E(G) : x \in X \setminus Y, y \in Y \setminus X\}$

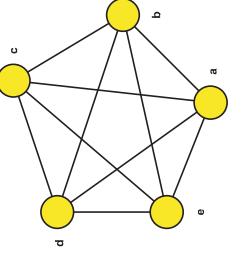
- $\delta^+(X)$, a set of edges leaving set X , i.e. $\delta^+(X) := E^+(X, V(G) \setminus X)$
- $\delta^-(X)$, a set of edges entering set X , i.e. $\delta^-(X) := \delta^+(V(G) \setminus X)$
- $\delta(X)$, a set of "border" edges of set X , i.e. $\delta(X) := \delta^+(X) \cup \delta^-(X)$
- $\Gamma(X)$, a set of neighbor nodes of set X , i.e. $\Gamma(X) := \{v \in V(G) \setminus X : \text{"border" edge } e \in \delta(X) \text{ exists and it is incident with node } v\}$

Z. Hanzálek (CTU) Introduction to Combinatorial Optimization February 19, 2019 39 / 49

Special Graphs

A **complete digraph** (úplný graf) is a simple graph $G = (V, E)$, where E is a set of all possible pairs of different nodes of V .

A **complete undirected graph** is a simple graph, in which every pair of vertices is connected by a unique edge. We denote it K_n , where n is the number of nodes.
A graph is called **regular** (regulární) if all its nodes have the same degree. If the degree of all nodes is k , the graph is called **k -regular**.

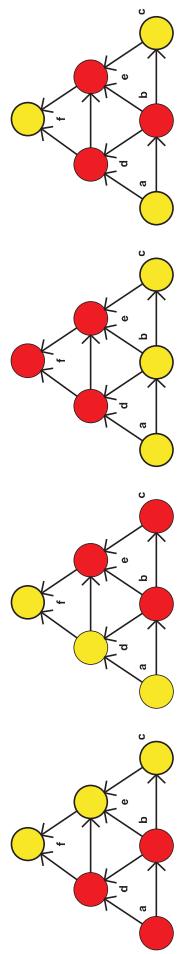


- **Edge progression (sled)** is a sequence $v_1, e_1, v_2, e_2, \dots, v_k, e_k, v_{k+1}$ such that $e_i = (v_i, v_{i+1}) \in E(G)$ or $e_i = \{v_i, v_{i+1}\} \in E(G)$ for all i .
- Edge progression is called **closed** if $v_1 = v_{k+1}$.
- Directed (undirected) **walk (tah)** is directed (undirected) edge progression, where no edge appears more than once, i.e. $e_i \neq e_j$ for all $1 \leq i < j \leq k$.
- Directed (undirected) **path (cesta)** is directed (undirected) walk, where no node appears more than once, i.e. $v_i \neq v_j$ for all $1 \leq i < j \leq k+1$.

- **The circuit (also called cycle)** is a subgraph $(v_1, \dots, v_k, e_1, \dots, e_k)$ such that the sequence $v_1, e_1, v_2, \dots, v_k, e_k, v_1$ is a closed undirected walk (tah) and $v_i \neq v_j$ for $1 \leq i < j \leq k$.
- **The cycle (also called circuit)** is a subgraph $(v_1, \dots, v_k, e_1, \dots, e_k)$ such that the sequence $v_1, e_1, v_2, \dots, v_k, e_k, v_1$ is a closed directed walk (tah) and $v_i \neq v_j$ for $1 \leq i < j \leq k$.

A complement of simple graph G is simple graph H such that $G + H$ is the complete graph. A pair of nodes in the complement is connected if it is not connected in G .

A clique is a subgraph that is complete. The number of nodes in the maximum (biggest) clique is called **the clique number**



Undirected Graph - Tree

Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	42 / 49
Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	43 / 49

- An undirected graph is called **connected**, if every pair of nodes is connected by an undirected path.
- The maximal connected subgraphs of G are its **connected components**.
 - Every node of the graph is included in exactly one connected component.
 - The connected component containing node x can be found as a complete subgraph induced by the set of all nodes which can be reached from x via the undirected path.
- **A forest** is an undirected graph G without a circuit.
- **A tree** is an undirected graph G without a circuit that is connected.
- For every connected graph there exists a spanning that is the tree and it is called the **spanning tree**.

Let G be an undirected graph with n nodes, then the following statements are equivalent:

- (a) G is a tree.
- (b) G has $n-1$ edges and no circuit.
- (c) G has $n-1$ edges and is connected.
- (d) G is connected and while removing any edge it will not be connected anymore.
- (e) G is a minimal graph which has $\delta(X) \neq \emptyset$ for all $\emptyset \neq X \subset V(G)$
- (f) G is circuit-free and the addition of any edge creates a circuit.
- (g) G contains a unique path between any pair of vertices.

Undirected Graph - Tree

Connectivity and Trees in Digraphs

- Proof:
- (a) \Rightarrow (g): follows from the fact that the union of two distinct paths with the same endpoints contains a circuit.
 - (f) \Rightarrow (b) \Rightarrow (c): follows from the fact that for a forest with n nodes, m edges and p connected components $n = m + p$. (The proof is a trivial induction on m).
 - (g) \Rightarrow (e) \Rightarrow (d): see [1] page 17 Proposition 2.3.
 - (d) \Rightarrow (f): trivial.
 - (c) \Rightarrow (a): G is connected with $n - 1$. As long as there are any circuits in G , we destroy them by deleting any edge of the circuit. Suppose we have deleted k circuits, the resulting graph G' is a tree (contains no circuit and is connected) and has $m = n - 1 - k$ edges. So $n = m + p = n - 1 - k + 1$, implying $k = 0$.

Any **directed acyclic graph** has at least one topological ordering.

Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	46 / 49
Z. Hanzálek (CTU)	Literature	February 19, 2019	47 / 49

Summary

A graph is:

- often used to formalize optimization problems
- very general
- easy to represent



B. H. Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms.
Springer, third edition, 2006.

Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	47 / 49
Z. Hanzálek (CTU)	Literature	February 19, 2019	48 / 49

Z. Hanzálek (CTU)	Introduction to Combinatorial Optimization	February 19, 2019	48 / 49
Z. Hanzálek (CTU)	Literature	February 19, 2019	49 / 49

Table of contents

Integer Linear Programming (ILP)

1 Introduction

- Problem Statement
- Comparison of ILP and LP
- Examples

Zdeněk Hanzálek, Přemysl Šúcha
zdenek.hanzalek@cvut.cz

CTU in Prague

April 10, 2018

Problem Statement

Z. Hanzálek (CTU) Integer Linear Programming (ILP)

April 10, 2018 1 / 42

3 Problem Formulation Using ILP

4 Conclusion

Comparison of ILP and LP

Z. Hanzálek (CTU) Integer Linear Programming (ILP)

April 10, 2018 2 / 42

- The LP problem solution space is convex, since $x \in \mathbb{R}^n$
- The ILP problem differs from the LP problem in allowing integer-valued variables. If some variables can contain real numbers, the problem is called Mixed Integer Programming - MIP. Often MIP is also called ILP, and we will use the term ILP when at least one variable has integer domain.
- If we solve the ILP problem by an LP algorithm and then just round the solution, we could not only get the suboptimal solution, we can also obtain a solution which is not feasible.
- While the LP is solvable in polynomial time, ILP is NP-hard, i.e. there is no known algorithm which can solve it in polynomial time.
- Since the ILP solution space is not a convex set, we cannot use convex optimization techniques.

Integer Linear Programming (ILP)

The ILP problem is given by matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and vectors $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^n$. The goal is to find a vector $\mathbf{x} \in \mathbb{Z}^n$ such that $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ and $\mathbf{c}^T \cdot \mathbf{x}$ is the maximum.

Usually, the problem is given as $\max \{ \mathbf{c}^T \cdot \mathbf{x} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^n \}$.

- A large number of practical optimization problems can be modeled and solved using Integer Linear Programming - ILP.

Z. Hanzálek (CTU) Integer Linear Programming (ILP)

April 10, 2018 3 / 42

Z. Hanzálek (CTU) Integer Linear Programming (ILP)

April 10, 2018 4 / 42

Z. Hanzálek (CTU) Integer Linear Programming (ILP)

April 10, 2018 2 / 42

Example ILP1a: 2-Partition Problem

Example ILP1b: Fractional Variant of the 2-Partition Prob.

We allow division of banknotes such that $x_{i \in 1..n} \in \langle 0, 1 \rangle$. The solution space is a convex set - the problem can be formulated by LP:

$$\begin{aligned} \min & \quad 0 \\ \text{subject to:} & \quad \sum_{i \in 1..n} x_i * p_i = 0.5 * \sum_{i \in 1..n} p_i \\ & \quad x_i \leq 1 \quad i \in 1..n \\ \text{parameters:} & \quad n \in \mathbb{Z}_0^+, \quad p_{i \in 1..n} \in \mathbb{Z}_0^+ \\ \text{variables:} & \quad x_{i \in 1..n} \in \mathbb{R}_0^+ \end{aligned}$$

- For example: $p = [100, 50, 50, 20, 20, 10, 10]$ the fractional variant allows for $x = [0, 0, 0.9, 1, 1, 1, 1]$ and thus divides the banknotes into equal halves $100 + 50 + 5 = 45 + 50 + 20 + 20 + 10 + 10$, but this instance does not have a non-fractional solution.
- For some non-fractional instances we can easily find that they cannot be partitioned (e.g. when the sum of all values divided by the greatest common divisor is not an even number), however we do not know any alg that can do it in polynomial time for any non-fractional instance.

April 10, 2018 Integer Linear Programming (ILP) 6 / 42

Example ILP1c: 2-Partition Prob. - Optimization Version

- The decision problem can be solved by an optimization algorithm while using a **threshold** value (here $0.5 * \sum_{i \in 1..n} p_i$) and comparing the optimal solution with the threshold.
- Moreover, when the decision problem has no solution, the optimization algorithm returns a value that is closest to the threshold.

$$\begin{aligned} \min & \quad C_{\max} \\ \text{subject to:} & \quad \sum_{i \in 1..n} x_i * p_i \leq C_{\max} \\ & \quad \sum_{i \in 1..n} (1 - x_i) * p_i \leq C_{\max} \\ \text{parameters:} & \quad n \in \mathbb{Z}_0^+, \quad p_{i \in 1..n} \in \mathbb{Z}_0^+ \\ \text{variables:} & \quad x_{i \in 1..n} \in \{0, 1\}, \quad C_{\max} \in \mathbb{R}_0^+ \end{aligned}$$

Application: the scheduling of nonpreemptive tasks $\{T_1, T_2, \dots, T_n\}$ with processing times $[p_1, p_2, \dots, p_n]$ on two parallel identical processors and minimization of the completion time of the last task (i.e. maximum completion time C_{\max}) - $P2 \parallel C_{\max}$. The fractional variant of 2-partition problem corresponds to the preemptive scheduling problem.

Z. Hanzálek (CTU) April 10, 2018 7 / 42

$$\begin{aligned} \text{Shortest Path in directed graph} \\ \bullet \text{Instance:} \text{ digraph } G \text{ with } n \text{ nodes, distance matrix } c : V \times V \rightarrow \mathbb{R}_0^+ \text{ and two nodes } s, t \in V. \\ \bullet \text{Goal:} \text{ find the shortest path from } s \text{ to } t \text{ or decide that } t \text{ is unreachable from } s. \end{aligned}$$

LP formulation using a physical analogy:

$$\begin{aligned} \bullet \text{node} &= \text{ball} \\ \bullet \text{edge} &= \text{string (we consider a symmetric distance matrix } c) \\ \bullet \text{node } s \text{ is fixed, other nodes are pulled by gravity} \\ \bullet \text{tightened string} &= \text{shortest path} \\ \text{Is it a polynomial problem?} \\ \max & \quad l_t \\ \text{subject to:} & \quad l_s = 0 \\ & \quad l_j \leq l_i + c_{i,j} \quad i \in 1..n, j \in 1..n \\ \text{parameters:} & \quad n \in \mathbb{Z}_0^+, \quad c_{i \in 1..n, j \in 1..n} \in \mathbb{R}_0^+ \\ \text{variables:} & \quad l_{i \in 1..n} \in \mathbb{R}_0^+ \end{aligned}$$

Z. Hanzálek (CTU) April 10, 2018 8 / 42

Example ILP3: Travelling Salesman Problem

Algorithms

Asymmetric Traveling Salesman Problem

- **Instance:** complete digraph K_n , distance matrix $c : V \times V \rightarrow \mathbb{Q}^+$.
- **Goal:** find the shortest Hamiltonian cycle. Cycle is a subgraph $(v_1, \dots, v_k, e_1, \dots, e_k)$ such that the sequence $v_1, e_1, v_2, \dots, v_k, e_k, v_1$ is a closed directed walk (tah) and $v_i \neq v_j$ for $1 \leq i < j \leq k$.

$x_{i,j} = 1$ iff node i is in the cycle just before node j

The enter and leave constraints do not capture the TSP completely, since any disjoint cycle (i.e. consisting of several sub-tours) will satisfy them. We use s_i , the "time" of entering node i , to **eliminate the sub-tours**.

$$\begin{array}{ll}\min & \sum_{i \in 1..n} \sum_{j \in 1..n} c_{i,j} * x_{i,j} \\ \text{subject to:} & x_{i,i} = 0 \quad i \in 1..n \quad \text{avoid self-loop} \\ & \sum_{j \in 1..n} x_{i,j} = 1 \quad i \in 1..n \quad \text{enter once} \\ & \sum_{j \in 1..n} x_{i,j} = 1 \quad i \in 1..n \quad \text{leave once} \\ & s_j + c_{i,j} - (1 - x_{i,j}) * M \leq s_i \quad i \in 1..n, j \in 2..n \quad \text{cycle indivisibility} \\ \text{parameters:} & M \in \mathbb{Z}_0^+, n \in \mathbb{Z}_0^+, c_{i,j} \in 1..n \in \mathbb{Q}^+ \\ \text{variables:} & x_{i,j} \in \{0, 1\}, s_i \in 1..n \in \mathbb{R}_0^+\end{array}$$

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 9 / 42

Enumerative Methods

The most successful methods to solve the ILP problem are:

- Enumerative Methods
- Branch and Bounds
- Cutting Planes Methods



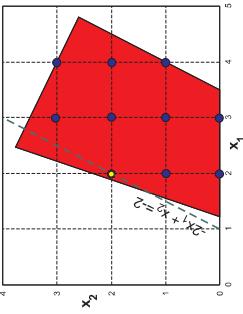
Ralph Gomory and Vášek Chvátal are prominent personalities in the field of ILP. Some of the solution methods are called: Gomory's Cuts or Chvátal-Gomory's cuts.

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 10 / 42

Enumerative Methods

$$\begin{array}{lll}\max & -2x_1 & + x_2 \\ \text{s.t.} & 9x_1 & - 3x_2 \geq 11 \\ & x_1 & + 2x_2 \leq 10 \\ & 2x_1 & - x_2 \leq 7 \\ & x_1, x_2 \geq 0, & x_1, x_2 \in \mathbb{Z}_0^+\end{array}$$

The figure below shows 10 feasible solutions. The optimal solution is $x_1 = 2, x_2 = 2$ with an objective function value of -2 .



Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 11 / 42

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 12 / 42

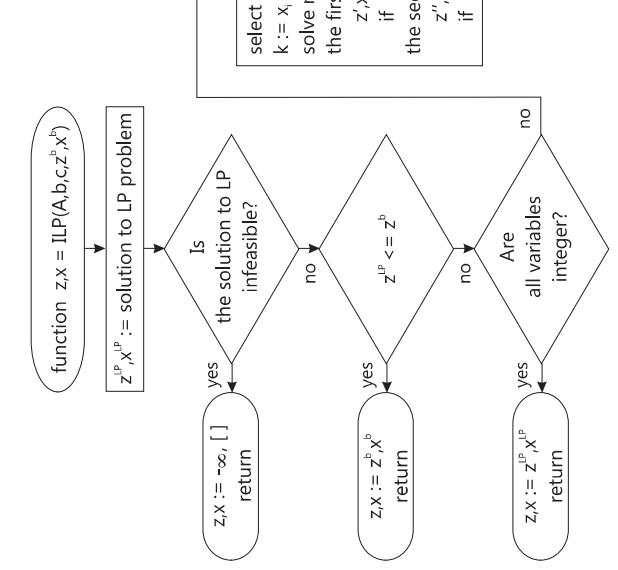
Branch and Bound Method

Branch and Bound Method

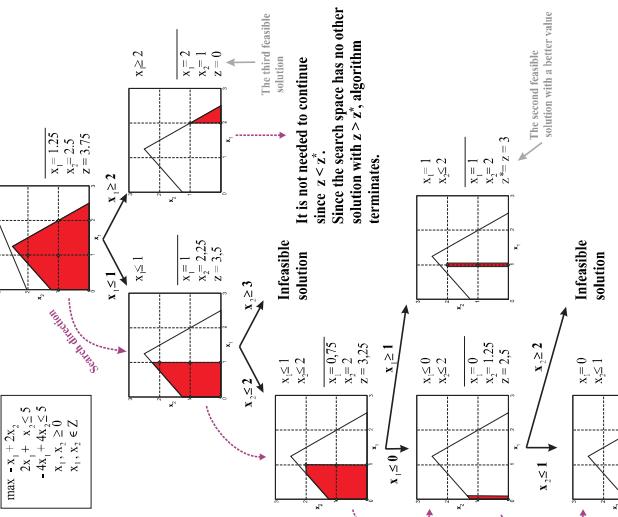
- The method is based on **splitting** the solution space into disjoint sets.
- It starts by relaxing on the integrality of the variables and **solves the LP problem**.
- If all variables $x_i \notin \mathbb{Z}$ are chosen and its value is assigned to k .
- Then the solution space is **divided into two sets** - in the first one we consider $x_i \leq [k]$ and in the second one $x_i \geq [k] + 1$.
- The algorithm **recursively repeats** computation for the both new sets till feasible integer solution is found.

Branch and Bound Algorithm - ILP maximization problem

Z. Hanzák (CTU)	Integer Linear Programming (ILP)	April 10, 2018	13 / 42
-----------------	----------------------------------	----------------	---------



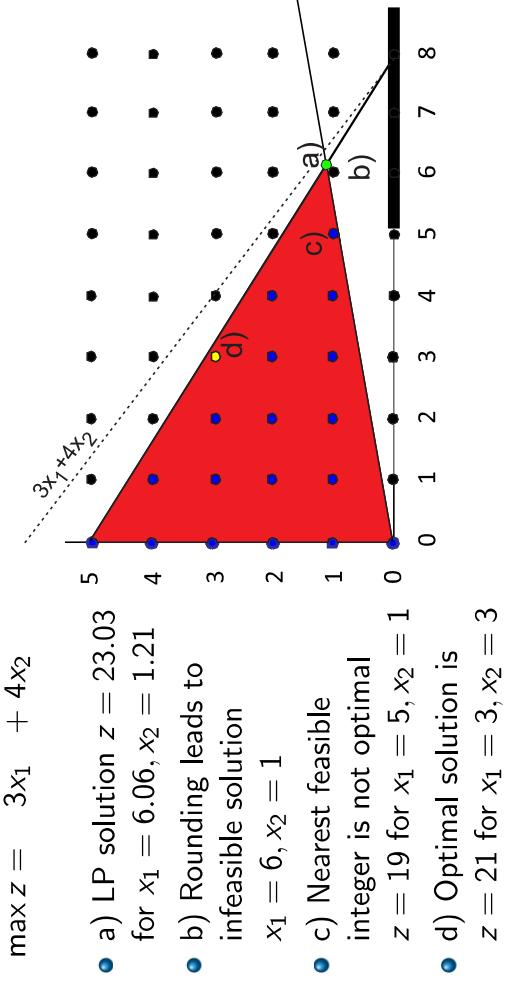
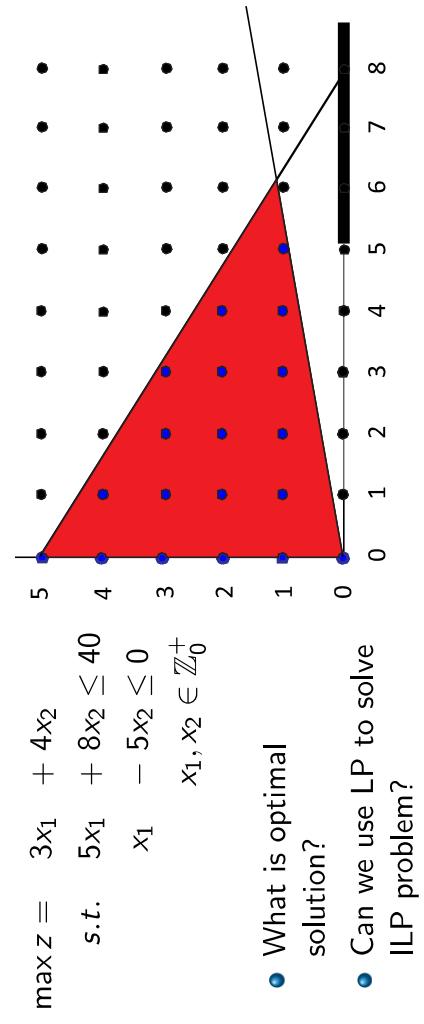
Z. Hanzák (CTU)	Integer Linear Programming (ILP)	April 10, 2018	14 / 42
-----------------	----------------------------------	----------------	---------



Z. Hanzák (CTU)	Integer Linear Programming (ILP)	April 10, 2018	15 / 42
-----------------	----------------------------------	----------------	---------

ILP Solution Space

Rounding is not always good choice



- Advantages of using integer variables
- more realistic (it does not make sense to produce 4.3 cars)
- flexible - e.g. binary variable can be used to model the decision (logical expression)
- we can formulate NP-hard problems

Drawbacks

- harder to create a model
- usually suited to solve the problems with less than 1000 integer variables



Shortest path in a graph

- Instance:** digraph G given by incidence matrix $W : V \times E \rightarrow \{-1, 0, +1\}$ (such that $w_{ij} = +1$ when edge e_j leaves vertex i and $w_{kj} = -1$ when edge e_j enters vertex k), distance vector $c \in \mathbb{R}_0^+$ and two nodes $s, t \in V$.
- Goal:** find the shortest path from s to t or decide that t is unreachable from s .

LP formulation:

- $x_j = 1$ iff edge j is chosen
- For every node except s and t we enter the node as many times as we leave it

$$\begin{aligned} \min \quad & \sum_{j \in 1..m} c_j * x_j \\ \text{subject to:} \quad & \sum_{j \in 1..m} w_{s,j} * x_j = 1 \quad \text{sources } s \\ & \sum_{j \in 1..m} w_{i,j} * x_j = 0 \quad i \in V \setminus \{s, t\} \\ & \sum_{j \in 1..m} w_{t,j} * x_j = -1 \quad \text{sink } t \\ \text{pars:} \quad & w_{i,j}, n_{i,j} \in \{-1, 0, 1\}, c_j \in 1..m \in \mathbb{R}_0^+ \\ \text{vars:} \quad & x_j \in 1..m \in \mathbb{R}_0^+ \end{aligned}$$

The returned values of x_j are integers (binary) even though it is LP. Why?



Z. Hanzálek (CTU) April 10, 2018 20 / 42 Integer Linear Programming (ILP)

Totally Unimodular Matrix leads to Integral Polyhedron

Integer Solution by Polynomial Algorithm

Polynomial time algorithm for general ILP is not known, however there are special cases which can be solved in polynomial time.

Definition - Totally unimodular matrix

Matrix $\mathbf{A} = [a_{ij}]$ of size m/n is totally unimodular if the determinant of every square submatrix of matrix \mathbf{A} is equal 0, +1 or -1.

Necessary condition: if \mathbf{A} is totally unimodular then $a_{ij} \in \{0, 1, -1\} \forall i, j$.

Proof: From the Lemma on previous slide - the simplex algorithm inspects vertices that are integer vectors.
Unfortunately, the simplex algorithm does not have polynomial complexity.

Fortunately, there are polynomial algorithms able to solve the LP problems and to find the vertex in the facet with optimal solutions. But this subject is studied in Linear Programming and Polyhedral Computation.

Lemma - Integral Polyhedron

Let A be a totally unimodular m/n matrix and let $b \in \mathbb{Z}^m$. Then each vertex of the polyhedron $P := \{x; \mathbf{Ax} \leq b\}$ is an integer vector.

Proof: [Schrijver] Theorem 8.1.

Sufficient Condition for Totally Unimodular Matrix

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 21 / 42

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 22 / 42

Problem Formulation Using ILP - Real Estate Investment

We consider 6 buildings for investment.
The price and rental income for each of them are listed in the table.

building	1	2	3	4	5	6
price[mil. Kč]	5	7	4	3	4	6
income[thousands Kč]	16	22	12	8	11	19

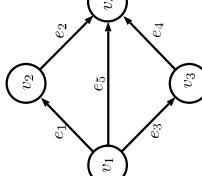
Goal:

- maximize income

Constraints:

- investment budget is 14 mil Kč
- each building can be bought only once

$$W = \begin{array}{ccccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & & & \\ v_1 & 1 & 0 & 1 & 0 & 1 & x_1 & & 1 \\ v_2 & -1 & 1 & 0 & 0 & 0 & x_2 & = & 0 \\ v_3 & 0 & 0 & -1 & 1 & 0 & x_3 & = & 0 \\ v_4 & 0 & -1 & 0 & -1 & -1 & x_4 & = & -1 \\ & & & & & & x_5 & & \end{array}$$



Proof: [Ahuja] Theorem 11.12. [KorteVygen] Theorem 5.26.

Example: ILP constraints of the Shortest Paths problem are: $W * x = b$

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 23 / 42

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 24 / 42

Problem Formulation Using ILP - Real Estate Investment

We consider 6 buildings for investment.

The price and rental income for each of them are listed in the table.

building	1	2	3	4	5	6
price[mil. Kč]	5	7	4	3	4	6
income[thousands Kč]	16	22	12	8	11	19

- maximize income

Constraints:

- investment budget is 14 mil Kč
- each building can be bought only once

Formulation

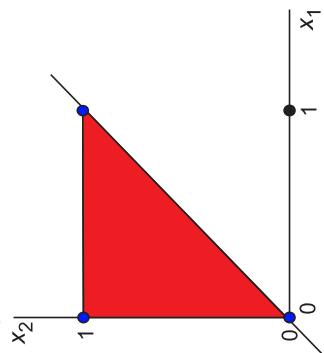
- $x_i = 1$ if we buy building i

$$\begin{aligned} \max \quad & z = 16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6 \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14 \\ & x_{i \in 1 \dots 6} \in \{0, 1\} \end{aligned}$$

Adding Logical Formula $x_1 \Rightarrow x_2$

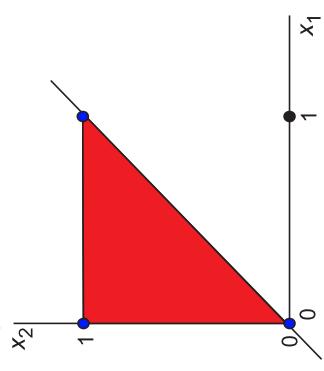
Another constraint:

- if building 1 is selected, then building 2 is selected too



$$\begin{array}{c|cc|cc|c} x_1 & x_2 & x_1 \Rightarrow x_2 \\ \hline 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array}$$

- Another constraint:
 - if building 1 is selected, then building 2 is selected too



$$\begin{array}{c|cc|cc|c} x_1 & x_2 & x_1 \Rightarrow x_2 \\ \hline 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array}$$

- Another constraint:
 - if building 1 is selected, then building 2 is selected too

$$\begin{aligned} \max \quad & z = 16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6 \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14 \\ & x_2 \geq x_1 \\ & x_{i \in 1 \dots 6} \in \{0, 1\} \end{aligned}$$

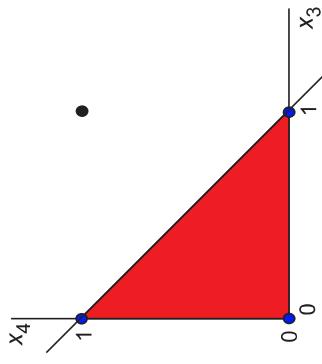
Adding Logical Formula $x_3 \Rightarrow \bar{x}_4$

Adding Logical Formula $x_3 \Rightarrow \bar{x}_4$

Another constraint:

- If building 3 is selected, then building 4 is not selected.

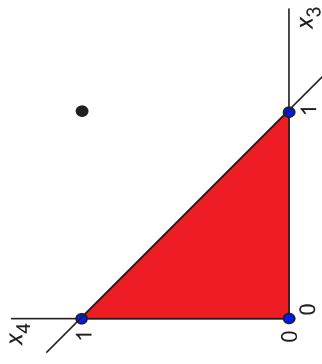
x_3	x_4	$x_3 \Rightarrow \bar{x}_4$
0	0	1
0	1	1
1	0	1
1	1	0



Another constraint:

- If building 3 is selected, then building 4 is not selected.

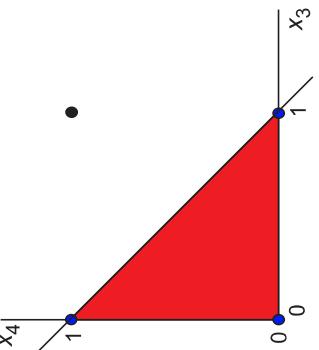
x_3	x_4	$x_3 \Rightarrow \bar{x}_4$
0	0	1
0	1	1
1	0	1
1	1	0



Adding Logical Formula $x_3 \Rightarrow \bar{x}_4$

Another constraint:

- If building 3 is selected, then building 4 is not selected.



x_5	x_6	$x_5 \text{ XOR } x_6$
0	0	0
0	1	1
1	0	1
1	1	0

$$\begin{aligned} \max \quad & z = 16x_1 + 22x_2 + 12x_3 + 8x_4 + 11x_5 + 19x_6 \\ \text{s.t.} \quad & 5x_1 + 7x_2 + 4x_3 + 3x_4 + 4x_5 + 6x_6 \leq 14 \\ & x_3 + x_4 \leq 1 \\ & x_i \in \{0, 1\} \end{aligned}$$

Adding Logical Formula $x_5 \text{ XOR } x_6$

Another constraint:

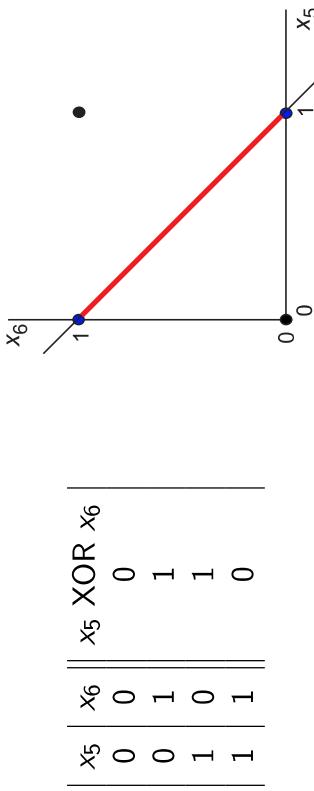
- either building 5 is chosen or building 6 is chosen, but not both

x_5	x_6	$x_5 \text{ XOR } x_6$
0	0	0
0	1	1
1	0	1
1	1	0

Adding Logical Formula $x_5 \text{ XOR } x_6$

Another constraint:

- either building 5 is chosen or building 6 is chosen, but not both



x_5	x_6	$x_5 \text{ XOR } x_6$
0	0	0
0	1	1
1	0	1
1	1	0

Adding Logical Formula - Homework

Labor demand, material demand and profit are listed in the table

product	T-shirt	shirt	trousers	capacity
labor	3	2	6	150
material	4	3	4	160
profit	6	4	7	

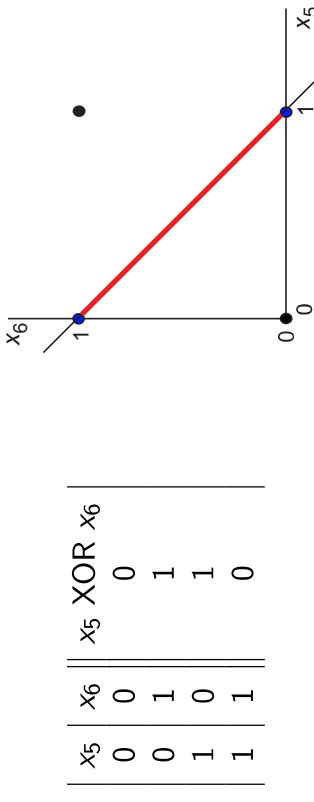
Goal:

- maximize the profit
- Constraints:
 - labor capacity is 150 person-hours
 - material capacity is 160 meters
 - building 1 must be chosen but building 2 can not
 - at least 3 estates must be chosen
 - exactly 3 estates must be chosen
 - if estates 1 and 2 have been chosen, then estate 3 must be chosen
too $(x_1 \text{ AND } x_2) \Rightarrow x_3$
 - exactly 2 estates can not be chosen

Adding Logical Formula $x_5 \text{ XOR } x_6$

Another constraint:

- either building 5 is chosen or building 6 is chosen, but not both



x_5	x_6	$x_5 \text{ XOR } x_6$
0	0	0
0	1	1
1	0	1
1	1	0

Problem Formulation Using ILP - Cloth Production

Problem Formulation Using ILP - Cloth Production

Adding Relationship between Binary and Integer Variable

Labor demand, material demand and profit are listed in the table

product	T-shirt	shirt	trousers	capacity
labor	3	2	6	150
material	4	3	4	160
profit	6	4	7	

Goal:

- maximize the profit

Constraints:

- labor capacity is 150 person-hours
- material capacity is 160 meters

Formulation

- x_i is the amount of product i

$$\begin{aligned} \max \quad & z = 6x_1 + 4x_2 + 7x_3 \\ \text{s.t.} \quad & 3x_1 + 2x_2 + 6x_3 \leq 150 \\ & 4x_1 + 3x_2 + 4x_3 \leq 160 \\ & x_{i \in 1..3} \in \mathbb{Z}_0^+ \end{aligned}$$

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 29 / 42

Adding Relationship between Binary and Integer Variable

Another constraint:

- the fixed cost has to be covered to rent the machine

Machine i is rented when product i is produced. We join binary y_i with integer x_i such that:

- $y_i = 0$ iff $x_i = 0$
- $y_i = 1$ iff $x_i \geq 1$

Formulation

- add binary variable y_i such that $y_i = 1$ when the machine producing product i is the rent
- the objective function will be changed to
 $\max z = 6x_1 + 4x_2 + 7x_3 - 200y_1 - 150y_2 - 100y_3$
- join binary y_i with integer x_i

product	T-shirt	shirt	trousers	capacity
machine cost	200	150	100	

Another constraint:

- the fixed cost has to be covered to rent the machine

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 30 / 42

Adding Relationship between Binary and Integer Variable

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 30 / 42

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 31 / 42

Adding Relationship between Binary and Integer Variable

Considering Several Values

Machine i is rented when product i is produced. We join binary y_i with integer x_i such that:

- $y_i = 0$ iff $x_i = 0$
- $y_i = 1$ iff $x_i \geq 1$

For range $x_i \in \langle 0, 100 \rangle$ these relations can be written as inequalities

- $x_i \leq 100 * y_i$
- $x_i \geq y_i$...we can omit this inequality due to the objective function ($x_i = 0, y_i = 1$ will not be chosen because we want x_i to be maximal and y_i minimal)

At least One of Two Constraints Must be Valid

Z. Hanzálek (CTU) April 10, 2018 31 / 42 Integer Linear Programming (ILP)

At least One of Two Constraints Must be Valid

Z. Hanzálek (CTU) April 10, 2018 32 / 42 Integer Linear Programming (ILP)

While modeling problems using ILP, we often need to express that the first, the second or both constraints hold. For example, $x_{i \in 1..4} \in \langle 0, 5 \rangle$, $x_{i \in 1..4} \in \mathcal{R}$

holds $2x_1 + 2x_2 \leq 8$
or $2x_3 - 2x_4 \leq 2$
or both

This can be modeled by a big M , i.e. big positive number (here 15), and variable $y \in \{0, 1\}$ so it can “switch off” one of the inequalities.

$2x_1 + 2x_2 \leq 8 + M \cdot y$
 $2x_3 - 2x_4 \leq 2 + M \cdot (1 - y)$

Another constraint:

- Total amount of work can be 40, 80 or 120 hours in order to better fit the work contract

- We only want some values of person-hours to be available:

$$3x_1 + 2x_2 + 6x_3 = \text{either } 40 \text{ or } 80 \text{ or } 120$$

Can be formulated using a set of additional variables $v_{i \in 1..3} \in \{0, 1\}$ as follows:

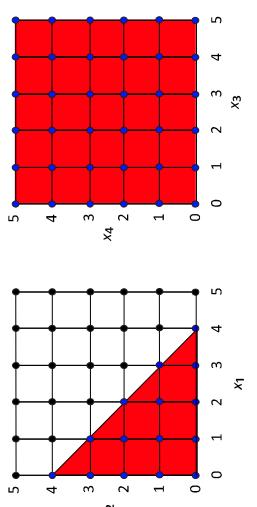
$$3x_1 + 2x_2 + 6x_3 = 40v_1 + 80v_2 + 120v_3$$
$$\sum_{i=1}^3 v_i = 1$$

for $y = 0$ inequalities:

$$2x_1 + 2x_2 \leq 8 + M \cdot y$$
$$2x_3 - 2x_4 \leq 2 + M \cdot (1 - y)$$

reduce to:

$$2x_1 + 2x_2 \leq 8$$



Z. Hanzálek (CTU) April 10, 2018 33 / 42 Integer Linear Programming (ILP)

Z. Hanzálek (CTU) April 10, 2018 34 / 42 Integer Linear Programming (ILP)

At least One of Two Constraints Must be Valid

Homework

for $y = 0$ inequalities:

$$\begin{aligned} 2x_1 + 2x_2 &\leq 8 + M \cdot y \\ 2x_3 - 2x_4 &\leq 2 + M \cdot (1-y) \end{aligned}$$

reduce to:

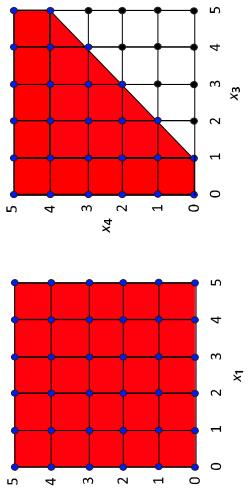
$$2x_1 + 2x_2 \leq 8$$

for $y = 1$ inequalities:

$$\begin{aligned} 2x_1 + 2x_2 &\leq 8 + M \cdot y \\ 2x_3 - 2x_4 &\leq 2 + M \cdot (1-y) \end{aligned}$$

reduce to:

$$2x_3 - 2x_4 \leq 2$$



$$2x_3 - 2x_4 \leq 2$$

At least One of Two Constraints Must be Valid

Example: Non-preemptive Scheduling

$1 \mid r_i, \tilde{d}_j \mid C_{\max} \dots$ NP-hard problem

- Instance:** A set of non-preemptive tasks $\mathcal{T} = \{T_1, \dots, T_i, \dots, T_n\}$ with release date r and deadline \tilde{d} should be executed on one processor. The processing times are given by vector p .
- Goal:** Find a feasible schedule represented by start times s that minimizes completion time $C_{\max} = \max_{i \in \{1, n\}} s_i + p_i$ or decide that it does not exist.

- T_i - chair to be produced by a joiner
- r_i - time, when the material is available
- \tilde{d}_i - time when the chair must be completed
- s_i - time when the chair production starts
- $s_i + p_i$ - time when the chair production ends

Example:

- In 2D draw the solution space of the system of inequalities:

$$\begin{aligned} 2x_1 + x_2 &\leq 5 + M \cdot y \\ 2x_1 - x_2 &\leq 2 + M \cdot (1-y) \\ y &\in \{0, 1\} \end{aligned}$$

- In 2D draw the solution space of the system of inequalities. Note that the equations correspond to parallel lines. Is it possible to find x_1, x_2 such that both equations are valid simultaneously?

$$\begin{aligned} 2x_1 + x_2 &\leq 5 + M \cdot y \\ M \cdot (1-y) + 2x_1 + x_2 &\geq 10 \\ y &\in \{0, 1\} \end{aligned}$$

At least One of Two Constraints Must be Valid

Example: Non-preemptive Scheduling

Since at the given moment, at most, one task is running on a given resource, therefore, for all task pairs T_i, T_j it must hold:

- T_i precedes T_j ($s_j \geq s_i + p_i$)
- or T_j precedes T_i ($s_i \geq s_j + p_j$)

Note that (for $p_i > 0$) both inequalities can't hold simultaneously. We need to formulate that at least one inequality holds. We will use variable $x_{ij} \in \{0, 1\}$ such that $x_{ij} = 1$ if T_i precedes T_j . For every pair T_i, T_j we introduce inequalities:

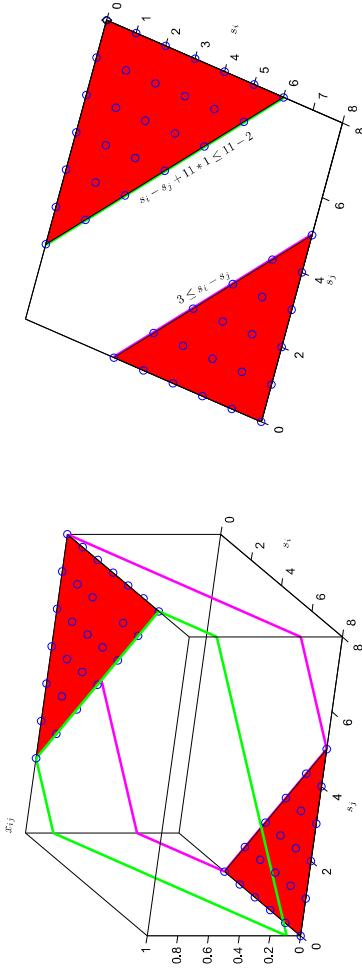
$$\begin{aligned} s_j + M \cdot (1 - x_{ij}) &\geq s_i + p_i && \text{"switched off" when } x_{ij} = 0 \\ s_i + M \cdot x_{ij} &\geq s_j + p_j && \text{"switched off" when } x_{ij} = 1 \end{aligned}$$

Scheduling - Illustration of Non-convex Space

At least K of N Constraints Must Hold

$$\begin{aligned}
 s_i &\geq r_i & i \in 1..n & \text{release date} \\
 \tilde{d}_i &\geq s_i + p_i & i \in 1..n & \text{deadline} \\
 s_j + M \cdot (1 - x_{ij}) &\geq s_i + p_i & i \in 1..n, j < i & T_i \text{ precedes } T_j \text{ GREEN} \\
 s_i + M \cdot x_{ij} &\geq s_j + p_j & i \in 1..n, j < i & T_j \text{ precedes } T_i \text{ VIOLET}
 \end{aligned}$$

For example: $p_i = 2, p_j = 3, r_i = r_j = 0, \tilde{d}_i = 10, \tilde{d}_j = 11, M = 11$



Non-convex 2D space is a projection of two cuts of a 3D polytope (determined by the set of inequalities) in planes $x_{ii} = 0$ and $x_{ii} = 1$.

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 38 / 42

ILP Solvers

- CPLEX - proprietary IBM <http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud>
- MOSEK - proprietary <http://www.mosek.com/>
- GLPK - free <http://www.gnu.org/software/glpk/>
- LP_SOLVE - free http://groups.yahoo.com/group/lp_solve/
- GUROBI - proprietary <http://www.gurobi.com/>
- YALMIP - Matlab toolbox for modelling ILP problems
- CVX - modeling framework

We have N constraints and we need at least K of them to hold.
Constraints are of type:

$$\begin{aligned}
 f(x_1, x_2, \dots, x_n) &\leq b_1 \\
 f(x_1, x_2, \dots, x_n) &\leq b_2 \\
 &\vdots \\
 f(x_1, x_2, \dots, x_n) &\leq b_N
 \end{aligned}$$

Can be solved by introducing N variables $y_{i \in 1..N} \in \{0, 1\}$ such that

$$\begin{aligned}
 f(x_1, x_2, \dots, x_n) &\leq b_1 + M \cdot y_1 \\
 f(x_1, x_2, \dots, x_n) &\leq b_2 + M \cdot y_2 \\
 &\vdots \\
 f(x_1, x_2, \dots, x_n) &\leq b_N + M \cdot y_N \\
 \sum_{i=1}^N y_i &= N - K
 \end{aligned}$$

If $K = 1$ and $N = 2$ we can use just one variable y_i and represent its negation as a $(1 - y_i)$, see above slides for details.

Z. Hanzálek (CTU) Integer Linear Programming (ILP) April 10, 2018 39 / 42

ILP - Conclusion

- NP-hard problem.
- Used to formulate majority of combinatorial problems.
- Often solved by branch and bound method.

References

-  Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin.
Network Flows: Theory, Algorithms, and Applications.
Prentice Hall, 1993.
-  B. H. Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms.
Springer, fourth edition, 2008.
-  James B. Orlin.
15.053 Optimization Methods in Management Science.
MIT OpenCourseWare, 2007.
-  Alexander Schrijver.
A Course in Combinatorial Optimization.
2006.

Table of contents

Z. Hanzálek (CTU)	Integer Linear Programming (ILP)	April 10, 2018	42 / 42
-------------------	----------------------------------	----------------	---------

Problem Statement

1 Introduction

- Problem Statement
- Negative Weights YES and Negative Cycles NO

a) Shortest Path

- **Instance:** Digraph G , weights $c : E(G) \rightarrow \mathbb{R}$, nodes $s, t \in V(G)$.
- **Goal:** Find the shortest $s - t$ path P , i.e. one of minimum weight $c(E(P))$, or decide that t is unreachable from s .

b) Shortest Path Tree - SPT

Other problems involving the shortest path:

- b) from source node s to every node (**Shortest Path Tree - SPT**)
- c) from every node to sink node t
- d) between all pairs of nodes (**All Pairs Shortest Path**)

Problem a) is often solved by algorithms for b), c) or d). There is no known algorithm with a better asymptotic time complexity. The algorithm can be terminated when t is reached.
Problem c) can be easily transformed to b) by reversing the edges.

Z. Hanzálek (CTU)	Shortest Paths	March 22, 2018	1 / 47
-------------------	----------------	----------------	--------

Problem Statement

- 1 **Algorithms and Examples of Problems Formulated as SP**
 - Dijkstra Algorithm
 - Shortest path from s to t
 - Bellman-Ford Algorithm
 - Dynamic Programming Perspective
 - Shortest Paths in DAGs
 - Floyd Algorithm
 - All Pairs Shortest Path
- 2 **Conclusion**

Z. Hanzálek (CTU)	Shortest Paths	March 22, 2018	2 / 47
-------------------	----------------	----------------	--------

Z. Hanzálek (CTU)	Shortest Paths	March 22, 2018	3 / 47
-------------------	----------------	----------------	--------

Similar Problems

Different Problems

- 1) The **longest path** can be transformed to the shortest path while reversing the sign of weights. Thus, we search for the minimum instead of the maximum.

- 2) When the **nodes are weighted**, or both nodes and edges are weighted, the weight of the path is the sum of the edges and the nodes weights along this path. This can be transformed to the weighted edges case as follows:

- Replace every node v from the original graph by the **pair of nodes** v_1 and v_2 and connect them by an edge with weight equal to the weight of node v .

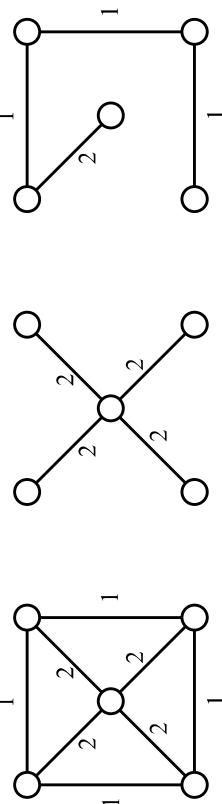
- the edge entering node v now enters v_1
- the edge leaving node v now leaves v_2

- every undirected edge (v, w) is transformed to two edges (v_1, w) and (w, v_2)

Negative Weights YES and Negative cycles NO

1) Minimum Spanning Tree - MST

In an undirected graph with the weights associated to arcs, find a spanning tree of minimum weight or decide that the graph is not connected.



The spanning tree in the middle (SPT from central node) has weight 8 and radius 4 (the longest path between two nodes) while the spanning tree on the right side (MST) has weight 5 and radius 5.

2) Steiner Tree

Given a connected undirected graph G , weights $c : E(G) \rightarrow \mathbb{R}^+$, and a set $T \subseteq V(G)$ of terminals, find a Steiner tree for T , i.e. a tree S with $T \subseteq V(S)$ and $E(S) \subseteq E(G)$, such that $c(E(S))$ is minimum.

Edge Progression (sled) and Path (cesta)

We consider oriented graphs:

- negative weights are allowed
- negative cycles are not allowed, since the shortest path problem becomes **NP-hard** when the graph contains a negative cycle,

When we transform **undirected graph** to directed graph, then we consider only instances with **nonnegative weights**:

- every undirected edge connecting v and w is transformed to two edges (v,w) and (w,v)
- this transformation of the negative undirected edge results in a negative cycle

Theorem - existence of the shortest path

If a path from s to t exists in the graph, then the shortest path from s to t exists too.

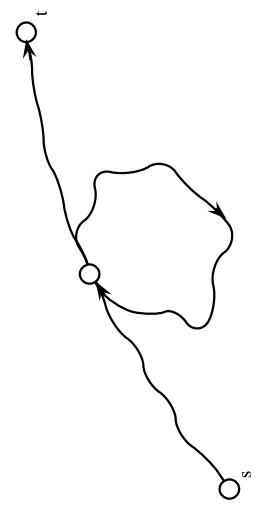
Note: the **theorem does not hold for edge progression** - in a graph with a negative cycle we can always find an even shorter edge progression which repeatedly goes through this negative cycle.

Definitions

- the **Length of the path P** is the **sum of the weights** of its edges (will be denoted simply as $c(E(P))$).
- $l(s, t)$, a distance from s to t , is defined as a length of the **shortest** path from s to t .

Theorem - shortest edge progression and path

- If there is no **negative weight or zero weight cycle** in the graph, then every shortest edge progression from s to t is the shortest path from s to t .
- If there is no **negative weight cycle** in the graph, then every shortest edge progression from s to t contains the shortest path from s to t and the length of this path is the same.



Basic Facts - Negative cycle

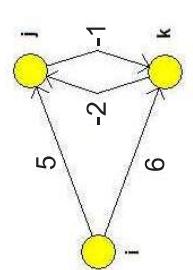
The algorithms listed below make use of the following

- (1) Their computational speed is based on the fact that they **do not care about the repetition of nodes along the path** (i.e. they do not distinguish the path from edge progression).
- (2) If the graph contains a negative cycle, **(1) can not be used** because the shortest edge progression does not need to exist (then it is NP-hard to find the shortest path).

Example: a graph with a negative cycle - consequently, the triangular inequality does not hold - the negative cycle of the edge progression is created while joining the two shortest paths.

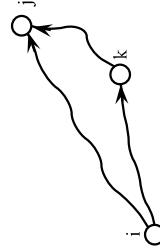
$$\begin{aligned} l(i,j) &\leq l(i,k) + l(k,j) \\ 6-2 &\leq (5-1) + (-2) \\ 4 &\leq 2 \quad \dots \text{contradiction} \end{aligned}$$

What the graph can not contain, when interested in the longest path?



Theorem - triangle inequality

If the graph does not contain a cycle of negative weight then distances between **all triplets of nodes** i, j, k satisfy: $l(i,j) \leq l(i,k) + l(k,j)$.



Corollary: Let $c(i,j)$ be the **weight of edge** from i to j .

Then if the graph does not contain a negative cycle:

$$l(i,j) \leq c(i,j), \quad l(i,j) \leq l(i,k) + c(k,j) \quad \text{and} \quad l(i,j) \leq c(i,k) + l(k,j)$$

Bellman's Principle of Optimality

Theorem - Bellman's Principle of Optimality

Suppose we have digraph G , weights $c : E(G) \rightarrow \mathbb{R}$, no negative cycles.

Let $k \in \mathbb{N}$, and let s and w be two vertices. Let P^k be a shortest one among all s - w -paths with at most k edges, and let $e = (v, w)$ be its final edge. Then $P^{k-1}[s, v]$ (i.e. P^k without the edge e) is a shortest one among all s - v -paths with at most $k-1$ edges.

Proof by contradiction:

We will study two cases:

- 1) Suppose Q_1 is an s - v -path shorter than $P^{k-1}[s, v]$, $|E(Q_1)| \leq k-1$ and Q_1 does not contain w , then:
 $c(E(Q_1)) < c(E(P^{k-1}[s, v]))$
 $c(E(Q_1)) + c(e) < c(E(P^{k-1}[s, v])) + c(e) = c(E(P^k))$
 This is a contradiction, since s - w -path consisting of Q_1 and e can't be shorter than P^k .

Bellman's Principle of Optimality - proof cont.

Bellman's equation and Generalization on Path Segments

Bellman's equation: $I(s, w) = \min_{v \neq w} \{I(s, v) + c(v, w)\}$ holds if the graph does not contain a negative cycle

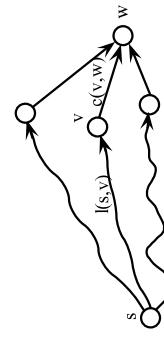
2) Suppose Q_2 is an s - v -path shorter than $P^{k-1}[s, v]$, $|E(Q_2)| \leq k - 1$ and Q_2 **does** contain w , then:

$$\begin{aligned} c(E(Q_2)) &< c(E(P^{k-1}[s, v])) \\ c(E(Q_2[s, w])) + c(E(Q_2[w, v])) &< c(E(P^{k-1}[s, v])) \dots \text{add edge } e \\ c(E(Q_2[s, w])) + c(E(Q_2[w, v])) + c(e) &< \underbrace{c(E(P^{k-1}[s, v])) + c(e)}_{\geq 0} + c(E(P^k)) \end{aligned}$$

... because $Q_2[w, v]$ and e is a non-negative cycle.

$$c(E(Q_2[s, w])) < c(E(P^k))$$

This is a contradiction, since $Q_2[s, w]$ can't be shorter than P^k .



A straight line indicates a single edge; a wavy line indicates a shortest path between the two vertices it connects (other nodes on these paths are not shown).

Generalization - the shortest path consists of the shortest paths

Suppose we have a graph without negative cycles. If the shortest path from s to w contains node v , the segment of the path from s to v is the shortest s - v -path and similarly the segment of the path from v to w is the shortest v - w -path and $I(s, w) = I(s, v) + I(v, w)$.

Z. Hanzálek (CTU) March 22, 2018 12 / 47

Shortest Paths March 22, 2018 13 / 47

Dijkstra Algorithm [1959] - Nonnegative Weights

Input: digraph G , weights $c : E(G) \rightarrow \mathbb{R}_0^+$ and node $s \in V(G)$.

Output: Vectors I and p . For $v \in V(G)$, $I(v)$ is the length of the shortest path from s and $p(v)$ is the previous node in the path. If v is unreachable from s , $I(v) = \infty$ and $p(v)$ is undefined.

$$I(s) := 0; I(v) := \infty \text{ for } v \neq s; R := \emptyset;$$

while $R \neq V(G)$ **do**

 Find $v \in V(G) \setminus R$ such that $I(v) = \min_{i \in V(G) \setminus R} I(i)$;

$$R := R \cup \{v\} // \text{ in the first run we add vertex } s$$

 // further calculate non-permanent value of $I(w)$ for every node on border of R

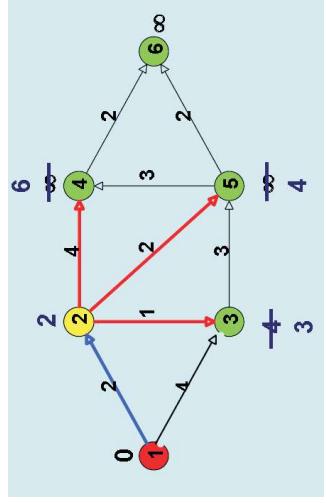
for $w \in V(G) \setminus R$ such that $(v, w) \in E(G)$ **do**

if $I(w) > I(v) + c(v, w)$ **then**

$$I(w) := I(v) + c(v, w); p(w) := v;$$

end

end



Iteration of Dijkstra Algorithm

Homework: What are the differences between Dijkstra algorithm for SPT and Prim algorithm for **minimum spanning tree - MST**? Find the smallest example with a different SPT and MST.

Z. Hanzálek (CTU) March 22, 2018 14 / 47

Shortest Paths March 22, 2018

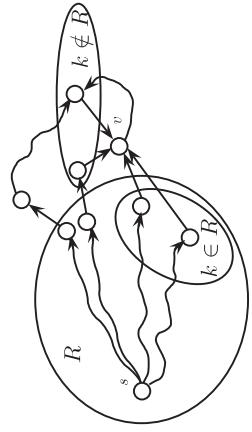
Z. Hanzálek (CTU) March 22, 2018 15 / 47

Correctness of Dijkstra Algorithm

Correctness of Dijkstra Algorithm - continuation

Proof of the inductive step: we show that Bellman's equation $I(s, v) = \min_{k \neq v} \{I(s, k) + c(k, v)\}$ holds:

- for **predecessors** $k \in R$, since $I(s, k)$ is optimal and Bellman's equation for these nodes was calculated in the internal loop (all $e \in E^+(R)$ were tested)
 - for **predecessors** $k \notin R$, since the value of $I(v)$ can not be improved by the path going through $k \notin R$ because $I(v) \leq I(k \notin R)$ and the $c(k, v)$ are nonnegative.



The value of $I(k \notin R)$ may get smaller in the next iterations (due to the path through $i \in V(G) \setminus R$), but while adding non-negative lengths it cannot get smaller than $I(v) = \min_{i \in V(G) \setminus R} I(i)$.

If we are interested in finding the shortest path from s to **just one node** t , Dijkstra algorithm can be terminated when we include t to R .
To accelerate the computation time we can add **additional information** to **Dijkstra algorithm** in order to perform **informed search**. This is idea of A* algorithm which is generalization of Dijkstra that cuts down on the size of the subgraph that must be explored.

- At the algorithm start, we set $h(v, t)$ for every $v \in V(G)$ such that it represents the **lower bound on distance from v to t** .
- For arbitrary nodes $v_1, v_2 \in V(G)$ inequality $c(v_1, v_2) \geq h(v_1, t) - h(v_2, t)$ **must be valid**.
 - we modify Dijkstra algorithm while choosing $v \in V(G) \setminus R$ such that $I(v) = \min_{i \in V(G) \setminus R} \{I(i) + h(i, t)\}$

This algorithm is faster, since vertices in wrong direction (with a high value of h) are never chosen. Unfortunately, in the extreme case, this algorithm has to go through all vertices as well.

Proof by induction:

For $|R| = 1$, the value of $I(s) = 0$ is optimal.

Inductive step: assuming that all nodes in R have optimal value of the shortest path we will prove the $I(v)$ value is **optimal/permanent** when we add the **best candidate** v to set R (the first line in the while loop).
The proof is completed on the next page.

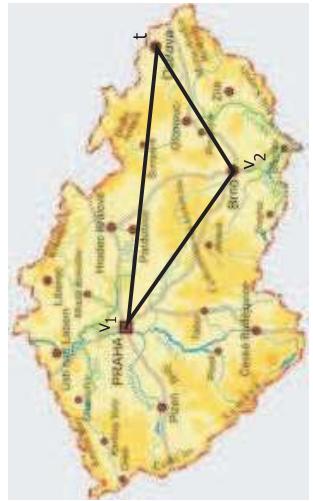
Properties of Dijkstra Algorithm

If we are interested in finding the shortest path from s to **just one node** t , Dijkstra algorithm can be terminated when we include t to R .
To accelerate the computation time we can add **additional information** to **Dijkstra algorithm** in order to perform **informed search**. This is idea of A* algorithm which is generalization of Dijkstra that cuts down on the size of the subgraph that must be explored.

- At the algorithm start, we set $h(v, t)$ for every $v \in V(G)$ such that it represents the **lower bound on distance from v to t** .
 - For arbitrary nodes $v_1, v_2 \in V(G)$ inequality $c(v_1, v_2) \geq h(v_1, t) - h(v_2, t)$ **must be valid**.
 - we modify Dijkstra algorithm while choosing $v \in V(G) \setminus R$ such that $I(v) = \min_{i \in V(G) \setminus R} \{I(i) + h(i, t)\}$
- There are algorithms with linear time complexity for **specific graphs**:
- Planar Graphs - Henzinger [1997]
 - Undirected graphs with integer non-negative weights - Thorup [1999]
 - DAGs - later in this lecture

Exploitation of Additional Information - Example

- Suppose the nodes are places in the two dimensional plane with coordinates $[x_v, y_v]$ and arc lengths equal to Euclidean distances between the points. The Euclidean distance from v to t is equal to $[(x_t - x_v)^2 + (y_t - y_v)^2]^{1/2}$, and it can be used as lower bound $h(v, t)$.
- Euclidean distances satisfy triangular inequality, i.e.
$$h(v_1, v_2) + h(v_2, t) \geq h(v_1, t).$$
 Therefore
$$c(v_1, v_2) \geq h(v_1, v_2) \geq h(v_1, t) - h(v_2, t)$$



Bidirectional Dijkstra Algorithm - Pseudocode

Z. Hanzálek (CTU)	Shortest Paths	March 22, 2018	20 / 47
Z. Hanzálek (CTU)	Shortest Paths	March 22, 2018	21 / 47

- Bidirectional Dijkstra algorithm:
- initialize $f^f(s) = 0$, $f^b(t) = 0$, $D = \infty$
 - maintain two sets of permanent vertices R^f and R^b
 - maintain joint priority queue of candidates PQ, where each item in the PQ knows whether it belongs to forward search or backward search
 - in each step, perform one iteration of Dijkstra
 - if top item in the PQ belongs to forward search
 - then perform the forward search from s
 - else perform the backward search from t
 - when v becomes permanent in forward search (i.e., $v \in R^f$) and there is an edge (v, w) such that $w \in R^b$ then update
$$D = \min \{ f^f(v) + c(v, w) + f^b(w), D \}$$
for every such edge (v, w)
 - similar update when a vertex becomes permanent in backward search
 - stop when $f^f + f^b \geq D$ where f^f and f^b stand for top items in PQ belonging to forward and backward search respectively

Bidirectional Dijkstra Algorithm - Idea

- Suppose the nodes are places in the two dimensional plane with coordinates $[x_v, y_v]$ and arc lengths equal to Euclidean distances between the points. The Euclidean distance from v to t is equal to $[(x_t - x_v)^2 + (y_t - y_v)^2]^{1/2}$, and it can be used as lower bound $h(v, t)$.

This algorithm finds the shortest path from s to t .

Basic idea:

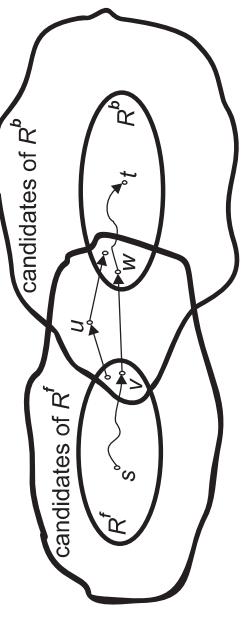
- We search "simultaneously"
 - forward from source with vector of distances f^f while using the original graph and
 - backward from target with vector of distances f^b while using the reverse graph (i.e., reversed orientation of edges)
- Stop when the search spaces "meet".
- This reduces the search space approximately by a factor of 2 only.
- On the other hand, Bidirectional Dijkstra is an **important inspiration** to many sophisticated algorithms (e.g., Contraction Hierarchies)

Z. Hanzálek (CTU)	Shortest Paths	March 22, 2018	21 / 47
Z. Hanzálek (CTU)	Shortest Paths	March 22, 2018	22 / 47

Bidirectional Dijkstra Algorithm - Correctness

Proof by contradiction:

- Suppose there exists an $s - t$ path Q_1 with length less than D going through vertex $u \notin R^f \cup R^b$.
 - Contradiction: Q_1 cannot exist since $f^f(u) \geq \text{top}^f$ and $f^b(u) \geq \text{top}^b$ and due to the stopping criterion $f^f(u) + f^b(u) \geq \text{top}^f + \text{top}^b \geq D$
- Suppose there exists an $s - t$ path Q_2 with length less than D going through edge (v, w) such that $v \in R^f$ and $w \in R^b$.
 - Contradiction: Q_2 cannot exist since the edge (v, w) was used in the update when later of vertices v and w became permanent and therefore $c(E(Q_2)) \geq D$



Example SPT.automaton.water: Measurement of Water Level [2]

You are on the bank of the lake and you have one 3-liter bottle and one 5-liter bottle. Both bottles are empty and your task is to have exactly 4 liters in the bigger bottle. You have no other equipment to measure the water level.

- a) Represent the problem by the graph.
- b) Set-up suitable weights and formulate the shortest path problem to find
 - solution with the minimum number of manipulations,
 - solution with the minimum amount of manipulated water,
 - solution with the minimum amount of water poured back in the lake.
- Homework c) During some manipulations you have to be very careful - for example when one bottle is filled completely but the other one does not become empty. Find the solution which minimizes the number of such manipulations.

Homework d) Is it possible to have 5 liters while using 4-liter and 6-liter bottles?

Example SPT.automation.bridge: Bridge and Torch Problem [2]

You have 4 people who want to cross a narrow bridge at night. The bridge can only hold two people at a time. They have one torch and, because it's night, the torch has to be used when crossing the bridge. One person can cross the bridge in 1 minute, one person in 2 minutes, one person in 5 minutes, and another person in 9 minutes. When two people cross the bridge together, they must move at the slower person's pace. The question is, can they all get across the bridge in 16 minutes or less?

Homework - Formulate the shortest path problem:

Four people come to a river in the night. There is a narrow bridge, but it can only hold two people at a time. They have one torch and, because it's night, the torch has to be used when crossing the bridge. One person can cross the bridge in 1 minute, one person in 2 minutes, one person in 5 minutes, and another person in 9 minutes. When two people cross the bridge together, they must move at the slower person's pace. The question is, can they all get across the bridge in 16 minutes or less?

Example SPT.diod.reliability: Maximum Reliability Path Problem [1]

Z. Hanzálek (CTU)

March 22, 2018

24 / 47

Shortest Paths

In the communication network we associate a reliability $p(i, j)$ with every arc from i to j . We assume the failures of links to be unrelated. The reliability of a directed path Q from s to t is the product of the reliability of the arcs in the path. Find the most reliable connection from s to t .

- a) Show that we can reduce the maximum reliability path problem to a shortest path problem.
- b) Suppose you are not allowed to make reduction. Specify $O(n^2)$ algorithm for solving the maximum reliability path problem.
- c) Will your algorithms in parts a) and b) work if some of the $p(i, j)$ coefficients are strictly greater than 1?

Bellman-Ford Algorithm [1958] (Moore [1959])

Z. Hanzálek (CTU)

March 22, 2018

25 / 47

Shortest Paths

Input: directed graph G without a negative cycle weights $c : E(G) \rightarrow \mathbb{R}$ and node $s \in V(G)$.

Output: vectors l and p . For all $v \in V(G)$, $l(v)$ is the length of the shortest path from s and $p(v)$ is the last but one node. If v is not reachable from s , then $l(v) = \infty$ and $p(v)$ is undefined.

```
l(s) := 0; l(v) := ∞ for v ≠ s;  
for i := 1 to n - 1 do  
    for every edge of graph (v, w) ∈ E(G) do  
        if l(w) > l(v) + c(v, w) then  
            l(w) := l(v) + c(v, w); p(w) := v;  
        end  
    end  
end
```

If an iteration of the main loop terminates without making any changes, the algorithm can be terminated, as subsequent iterations will not make changes any more. The worst case of the algorithm remains unchanged.

Z. Hanzálek (CTU)

March 22, 2018

26 / 47

Shortest Paths

Z. Hanzálek (CTU)

March 22, 2018

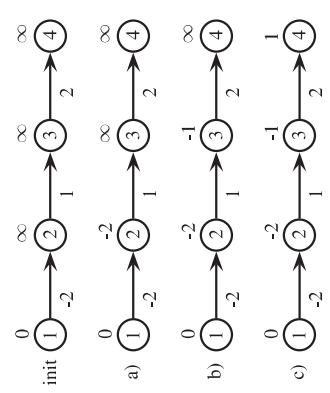
27 / 47

Shortest Paths

This example illustrates iterations of the Bellman-Ford algorithm finding shortest paths tree from node 1.

Assuming the edges to be processed in the internal loop

- in the worst order, from right to left
 - when only step a) is executed in the first iteration of the external loop, it requires 3 iterations of the external loop to obtain SPT
 - in the best order, from left to right
 - when steps a)b)c) are executed in the first iteration of the external loop, it requires only 1 iteration of the external loop to obtain SPT

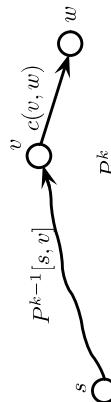


Theorem Proof

Prove of the theorem by induction:

- For $k = 1$ it holds, because $\ell^1(w) \leq c(s, w)$
- By the **induction hypothesis** H we have $\ell^{k-1}(v) \leq c(E(P^{k-1}[s, v]))$ after $k - 1$ iterations of external loop.
- In the k -th iteration of the external loop of the **algorithm ALG**, the **edge** (v, w) is examined by the **internal loop**, so the internal loop inequality holds: $\ell^k(w) \leq \ell^{k-1}(v) + c(v, w)$.
- By the **theorem assumption** (i.e., P^k is the shortest path and contains edge $e(v, w)$) and **Bellman Principle of Optimality BPO** it holds: $c(E(P^{k-1}[s, v])) + c(v, w) = c(E(P^k))$

Since in the k -th iteration edge $e(v, w)$ is also examined we conclude:
 $\ell^k(w) \stackrel{\text{ALG}}{\leq} \ell^{k-1}(v) + c(v, w) \stackrel{H}{\leq} c(E(P^{k-1}[s, v])) + c(v, w) \stackrel{\text{BPO}}{=} c(E(P^k))$.



We will base our reasoning on the following theorem:

Theorem: k-th iteration of Bellman-Ford algorithm

Let

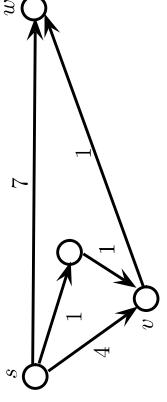
- $\ell^k(w)$ is the $\ell(w)$ label after k iterations of the external loop
- P^k is the shortest s - w -path with at most k edges and let $e = (v, w)$ be the last edge of this path
- $c(E(P^k))$ is the length of path P^k
- Then $\ell^k(w) \leq c(E(P^k))$.

Note: the $\ell^k(w) \leq c(E(P^k))$ is inequality, since the $\ell^k(w)$ is the length of the path which might go over more than k edges (see example with four nodes in the chain), but the P^k path is defined to have at most k edges.

Correctness of Bellman-Ford algorithm - Proof

Re-worded proof of the theorem:

While assuming H that we had $\ell^k(v)$ better than or equal to the shortest s - v -path with at most $k - 1$ edges in iteration $k - 1$, in iteration k of the algorithm ALG we will create $\ell^k(w)$ better than or equal to the shortest s - w -path with at most k edges since the shortest path consists of the shortest paths BPO .



Theorem Proof

- Correctness of the Bellman-Ford algorithm:**
In iteration $k = n - 1$ we will obtain $\ell^k(w) = c(E(P^k))$
- because the above theorem implies $\ell^k(w) \leq c(E(P^k))$
- and $\ell^k(w) \geq c(E(P^k))$ because no path has more than $n - 1$ edges and no path is shorter than the shortest one.

Time Complexity of Bellman-Ford Algorithm and Detection of Negative cycles

Best known algorithm for **SPT** without negative cycles.

- Time complexity is $O(nm)$.

- Note that every path consists at most of $n - 1$ edges and there exists an edge incident to every reachable node t .

The Bellman-Ford algorithm can **detect negative cycle** that is reachable from vertex s while checking triangular inequality for resulting I . There may be negative cycles not reachable from s , in such case we add node s' and connect it to all nodes v with $c(s', v) = 0$.

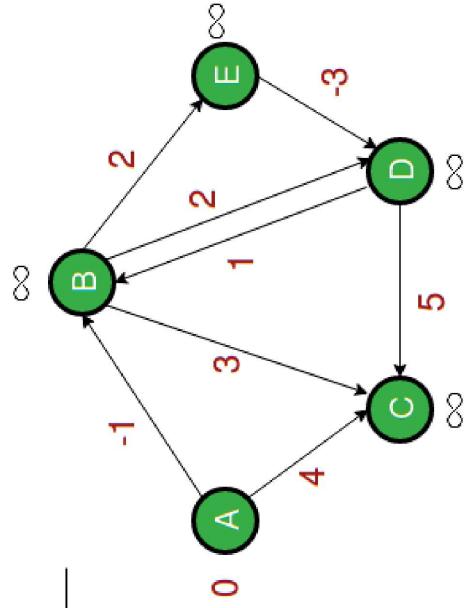
```
for for every edge of graph  $(v, t) \in E(G)$  do
    if  $I(t) > I(v) + c(v, t)$  then
        error "Graph contains a negative-weight cycle"
    end
```

However there are better methods for negative cycle detection [Cherkassky&Goldberg 1999].

Dynamic Programming Perspective

Example: Iterations of Bellman-Ford algorithm

A	B	C	D	E
0	∞	∞	∞	∞



(B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)

Dynamic Programming Perspective

Optimal substructure – the solution to a given optimization problem can be obtained by the combination of optimal solutions to its subproblems. Such optimal substructures are usually described by means of **recurrent formula** (e.g., Bellman equation).

Overlapping subproblems – computed solutions to subproblems are stored so that these don't have to recomputed. So Dynamic Programming is not useful when there are no common (overlapping) subproblems because there is no point to store the solutions if they are not needed again. One can observe so called **diamonds** in the in the solution space, in contrast to the solution space of Branch and Bound algorithm, which can be represented by the tree.

Observation on SPT algorithms:

- solve a collection of subproblems, e.g.

$$I(w) = \min_{v \neq w} \{I(v) + c(v, w)\}$$

- start with simplest one, i.e. $I(s) = 0$

- proceed with larger subproblems along the topological order

This is a general technique called Dynamic Programming, which requires two key attributes: **optimal substructure** and **overlapping subproblems**.

Example SPT.truck.negative: Truck Journey [2]

Let us assume a truck and n European cities with trailers.

- For each couple of cities (i, j) , we know $c(i, j)$, the cost of the truck transport from city i to city j .
- For some couple of cities (i, j) there are (infinitely many) trailers to be transported from city i to city j and the revenue for one trailer is $d(i, j)$.

Our task is to find the track journey from city s to city t and to maximize the profit regardless of the time.

- Represent a structure of the example by a directed graph.
- Formulate an optimization problem when each city can be entered at most once. What is the complexity of the problem?
- How the problem changes when each city can be entered several times?

Algorithm for DAGs

May be seen as simplified version of Bellman-Ford algorithm.

Input: directed acyclic graph G with topologically ordered vertices v_1, \dots, v_n , weights $c: E(G) \rightarrow \mathbb{R}$.

Output: vectors l and p . For all $i = 1 \dots n$, $l(v_i)$ is the length of the shortest path from v_1 and $p(v_i)$ is the last but one node. If v_i is not reachable from v_1 , then $l(v_i) = \infty$ and $p(v_i)$ is undefined.
 $l(v_1) := 0$; $l(v_i) := \infty$ for $i = 2 \dots n$;

```
for  $i := 2$  to  $n$  do
    for every edge of graph  $(v_j, v_i) \in E(G)$  do
        if  $l(v_i) > l(v_j) + c(v_j, v_i)$  then
             $l(v_i) := l(v_j) + c(v_j, v_i)$ ;  $p(v_i) := v_j$ ;
        end
    end
```

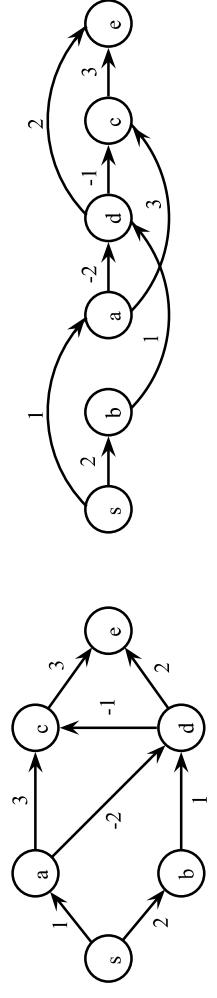
Correctness: induction on i and observation on previous slide.
Time complexity $O(|V| + |E|)$

Shortest Paths in Directed Acyclic Graphs (DAGs)

Definition: A **topological order** of G is an order of the vertices $V(G) = v_1, \dots, v_n$ such that for each edge $(v_i, v_j) \in E(G)$ we have $i < j$.

Proposition: A directed graph has a topological order if and only if it is acyclic (see the proof in [3]).

Consequence: DAG vertices can be arranged on a line so that all edges go from left to right.



Observation: shortest path from s to v_i cannot use any node from v_{i+1}, \dots, v_n , therefore we can find the shortest paths in topological order

Algorithm SPT.diod.negitive: Investment Opportunities [1]

Mr. Dow Jones, 50 years old, wishes to place his Individual Retirement Account funds in various investment opportunities so that at the age of 65

years, when he withdraws the funds, he has accrued maximum possible amount of money. Assume that Mr. Jones knows the investment alternatives for the next 15 years - each opportunity has starting year k , maturity period ρ (in years) and appreciation a it offers for the maturity period. How would you formulate this investment problem as a shortest path problem, assuming that at any point in time, Mr. Jones invests all his funds in a single investment alternative.

Opportunity	a	b	c	d	e	f	g	h	i	j
starting year	0	1	3	2	5	6	7	8	11	13
maturity period	4	5	6	5	4	5	6	5	4	2
appreciation	3.9%	4.7%	6.2%	4.2%	3.8%	4.1%	5.2%	5.8%	4.1%	3.2%

Z. Hanzálek (CTU) 38 / 47

Z. Hanzálek (CTU) Shortest Paths

March 22, 2018 37 / 47

Shortest Paths [1]

Floyd Algorithm [1962] (Warshall [1962])

Input: a digraph G free of negative cycles and weights $c : E(G) \rightarrow \mathbb{R}$.
Output: matrices I and p , where I_{ij} is the length of the shortest path from i to j , p_{ij} is the last but one node on such a path (if it exists).

```

 $I_{ij} := c((i,j))$  for all  $(i,j) \in E(G)$ ;  

 $I_{ij} := \infty$  for all  $(i,j) \notin E(G)$  where  $i \neq j$ ;  

 $I_{ii} := 0$  for all  $i$ ;  

 $p_{ij} := i$  for all  $(i,j)$ ;  

for  $k := 1$  to  $n$  do           // for all  $k$  check if  $I_{ij}$  improves
    for  $j := 1$  to  $n$  do
        for  $i := 1$  to  $n$  do
            if  $I_{ij} > I_{ik} + I_{kj}$  then
                 $I_{ij} := I_{ik} + I_{kj}$ ;  $p_{ij} := p_{kj}$ 
            end
        end
    end
end

```

Floyd Algorithm - Complexity and Properties

The best known algorithm for **All Pairs Shortest Path** problem without negative cycles.

- Time complexity $O(n^3)$.
- A graph contains a **negative cycle** iff (i.e. if and only if) there exists i such that $I_{ii} < 0$.
- By a small modification of the algorithm, where I_{ii}^0 is set to ∞ , one can find the nonnegative **minimal weight cycle** - see the diagonal in the next example. Then the value of I_{ii}^n should not be interpreted as a distance.

Johnson's Algorithm is better suited for the sparse graphs

- uses Dijkstra and Bellman-Ford
- complexity: $O(|V| \cdot |E| \cdot \log|V|)$ (in the simplest case)

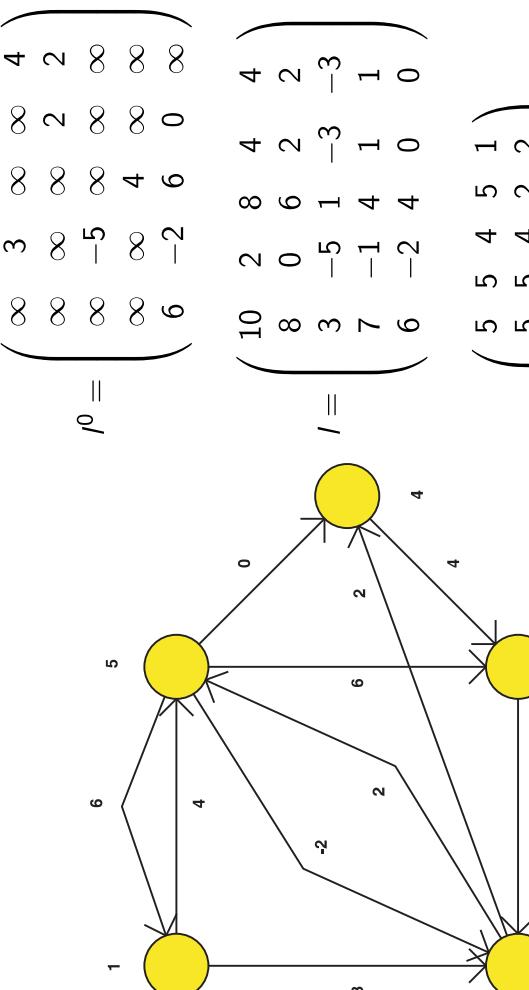
Floyd Algorithm

Input: initialize matrix I^0 by the weights of the edges
Output: computes the sequence of matrices $I^0, I^1, I^2, \dots, I^n$ where:
 I_{ij}^k is the length of the shortest path from i to j such that with the exception of i, j it goes only through nodes from the set $\{1, 2, \dots, k\}$

- one can easily compute matrix I^k from matrix I^{k-1} :

$$I_{ij}^k = \min\{I_{ij}^{k-1}, I_{ik}^{k-1} + I_{kj}^{k-1}\}$$

Floyd Algorithm - Example



$$I^0 = \begin{pmatrix} \infty & 3 & \infty & \infty \\ \infty & \infty & \infty & 2 \\ \infty & \infty & 4 & \infty \\ 6 & -2 & 6 & 0 \end{pmatrix}$$

$$I = \begin{pmatrix} 10 & 2 & 8 & 4 & 4 \\ 8 & 0 & 6 & 2 & 2 \\ 3 & -5 & 1 & -3 & -3 \\ 7 & -1 & 4 & 1 & 1 \\ 6 & -2 & 4 & 0 & 0 \end{pmatrix}$$

$$p = \begin{pmatrix} 5 & 5 & 4 & 5 & 1 \\ 5 & 5 & 4 & 2 & 2 \\ 5 & 3 & 4 & 2 & 2 \\ 5 & 3 & 4 & 2 & 2 \\ 5 & 4 & 5 & 2 & \end{pmatrix}$$

Example SPT.matrix.fire: Location of Fire Station [2]

Example SPT.matrix.warehouse: Warehouse Location

Formulate a shortest path problem:

Consider a road system in the city.

- a) We are looking for the best location of the fire station while minimizing its distance from the most distant place.

Homework b) How the problem changes (gets more difficult) when we are looking for the best location of two fire stations?

Homework c) How the problem changes (gets more difficult) when the maximum allowed distance is given and we are looking for the minimum number of stations.

Formulate a shortest path problem:

Consider a road system in the region. We are looking for the best location of the warehouse which supplies n customers consuming q_1, \dots, q_n units of goods per week. There is a suitable place for warehouse nearby each of the customers. Each customer is served by a separate car and the transport cost is a product (i.e. multiplication) of the distance and transported volume. Objective is to minimize the weekly transport costs.

Shortest Path - Conclusion

References

- An easy optimization problem with a lot of practical applications
 - OSPF (Open Shortest Path First) is a widely used protocol for Internet routing that uses Dijkstra algorithm.
 - RIP (Routing Information Protocol) is another routing protocol based on the Bellman-Ford algorithm.
 - looking for shortest/cheapest/fastest route in the map

- Basic routine for many optimization problems
 - scheduling with dependencies
 - ...

 Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin.
Network Flows: Theory, Algorithms, and Applications.
Prentice Hall, 1993.

 Jiří Demel.
Grafy a jejich aplikace.
Academia, 2002.

 B. H. Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms.
Springer, fourth edition, 2008.

Table of contents

Network Flows

1 Flows

- Maximum Flow Problem
 - Ford-Fulkerson Algorithm
 - Minimum Cut Problem
 - Integrality
- Feasible Flow with Balances - Decision Problem
 - Initial feasible flow for Ford-Fulkerson algorithm
- Minimum Cost Flow
- Minimum Cost Multicommodity Flow

April 3, 2019

Z. Hanzálek (CTU FEE)

April 3, 2019

2 / 49

2 Matching

- Maximum Cardinality Matching in Bipartite Graphs
- Assignment Problem - minimum weight perfect matching in complete bipartite graph
- Hungarian Algorithm

What is Network?

By network we mean a 5-tuple (G, I, u, s, t) , where G denotes the oriented graph, $u : E(G) \rightarrow \mathbb{R}_0^+$ and $I : E(G) \rightarrow \mathbb{R}_0^+$ denote the maximum and minimum capacity of the arcs and finally s represents the source node (there is an oriented path from the source node to every other node) while t represents the sink node (there is an oriented path from every other node to the sink node).

Network Flow

$f : E(G) \rightarrow \mathbb{R}_0^+$ is the flow if Kirchhoff law $\sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$ is valid for every node except s and t .

Feasible Flow

Feasible flow must satisfy $f(e) \in \langle I(e), u(e) \rangle$.
There might be no feasible flow when $I(e) > 0$.

Maximum Flow Problem

Maximum flow

Given a network (G, I, u, s, t) . The goal is to find the **feasible flow** f from the source to the sink that maximizes $\sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e)$ (i.e. to send the maximum volume of the flow from s to t).

$\delta^+(s)$ is a set of arcs leaving node s

$\delta^-(s)$ is a set of arcs entering node s (often we do not consider these arcs).

Example - Transportation problem: We wish to transport the maximum amount of goods from s to t . The problem is described by the network where the arc represents the route (pipeline, railway, motorway, etc.). The flows on the arcs are assumed to be steady and lossless.
Example constraints:

- $u_i = 10$ - arc i is capable of transporting 10 units maximum
- $I_j = 3$ - arc j must transport at least 3 units
- $I_k = u_k = 20$ - arc k transports exactly 20 units

Z. Hanzálek (CTU FEE)

April 3, 2019

2 / 49

Network Flows

April 3, 2019

2 / 49

Maximum Flow

Given a network (G, I, u, s, t) . The goal is to find the **maximum flow** f from the source to the sink that maximizes $\sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e)$ (i.e. to send the maximum volume of the flow from s to t).

$\delta^+(s)$ is a set of arcs leaving node s

$\delta^-(s)$ is a set of arcs entering node s (often we do not consider these arcs).

Example - Transportation problem: We wish to transport the maximum amount of goods from s to t . The problem is described by the network where the arc represents the route (pipeline, railway, motorway, etc.). The flows on the arcs are assumed to be steady and lossless.
Example constraints:

- $u_i = 10$ - arc i is capable of transporting 10 units maximum
- $I_j = 3$ - arc j must transport at least 3 units
- $I_k = u_k = 20$ - arc k transports exactly 20 units

Z. Hanzálek (CTU FEE)

April 3, 2019

2 / 49

Network Flows

April 3, 2019

2 / 49

Example Flow.scheduling: Multiprocessor Scheduling Problem with Preemption, Release Date and Deadline

Consider a $P \left| \text{pmtn}, r_j, \tilde{d}_j \right| -$ problem - we have **n tasks** which we want to assign to **R identical resources** (processors). Each task has its own **processing time** p_j , **release date** r_j and **deadline** \tilde{d}_j . **Preemption** is allowed (including migration from one resource to another).

Example for 3 parallel identical resources:

task	T_1	T_2	T_3	T_4
p_j	1.5	1.25	2.1	3.6
r_j	3	1	3	5
\tilde{d}_j	5	4	7	9

Goal

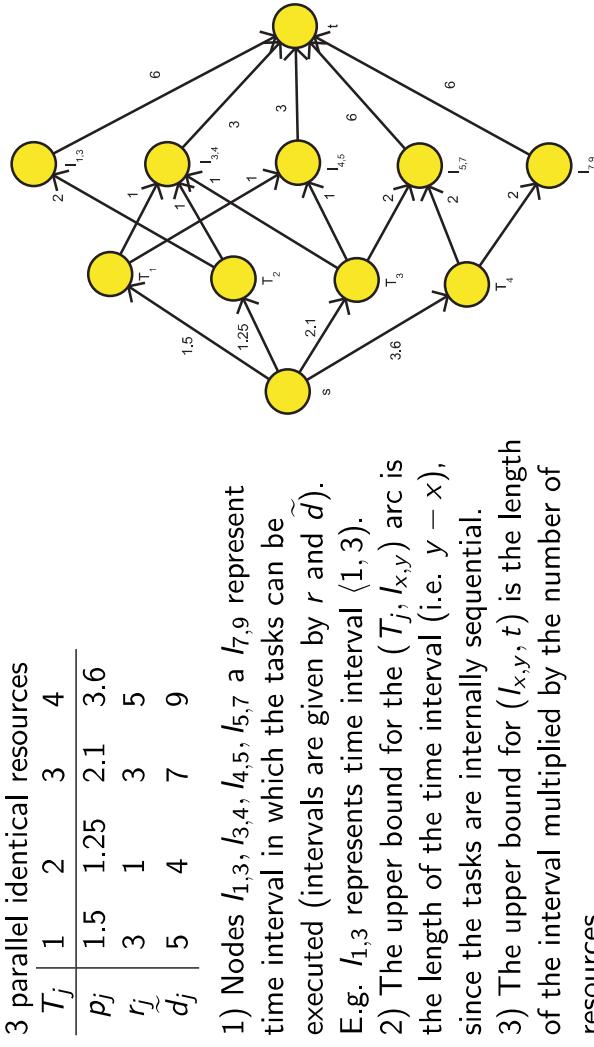
Assign all tasks to processors so that every processor will execute no more than one task at a moment and no task will be executed simultaneously on more than one processor.

Example Flow.dynamic: Dynamic Flow [Dem02]

Dynamic flow is changing its volume in time. We will show, how to formulate this problem while introducing discrete time and using (static) flow.

For example: let us consider cities a_1, a_2, \dots, a_n with q_1, q_2, \dots, q_n cars that should be transported to city a_n in K hours. When the cities a_i, a_j are connected directly, the duration of driving is denoted by d_{ij} and the capacity of the road in number of cars per hour is denoted by u_{ij} . Finally, p_i represents capacity of parking area in city a_i . The objective is to transport the maximum number of cars to city a_n in K hours.

Example Flow.scheduling: Multiprocessor Scheduling Problem with Preemption, Release Date and Deadline



- Nodes $l_{1,3}, l_{3,4}, l_{4,5}, l_{5,7}$ a $l_{7,9}$ represent time interval in which the tasks can be executed (intervals are given by r and \tilde{d}). E.g. $l_{1,3}$ represents time interval $\langle 1, 3 \rangle$.
- The upper bound for the $(T_j, l_{x,y})$ arc is the length of the time interval (i.e. $y - x$), since the tasks are internally sequential.
- The upper bound for $(l_{x,y}, t)$ is the length of the interval multiplied by the number of resources.

Maximum Flow Problem Formulated as LP

Variable $f(e) \in \mathbb{R}_0^+$ represents flow through arc $e \in E(G)$.

$$\begin{aligned} & \max \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e) \\ \text{s.t. } & \sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e) \\ & l(e) \leq f(e) \leq u(e) \\ & v \in V(G) \setminus \{s, t\} \\ & e \in E(G) \end{aligned}$$

Note that the following equation is valid for any set A containing source s but not containing sink t :

$$\sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e) = \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e)$$
.

Homework: Prove it while using Kirchhoff's law.

Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

Pioneers in the field of networks and network flows are L. R. Ford, Jr. and D. R. Fulkerson (in the picture). In 1956, they published a widely known **algorithm for the maximum flow problem**, the Ford-Fulkerson Algorithm.



Input: Network (G, l, u, s, t) .

Output: Maximum feasible flow f from s to t .

- ➊ Find the feasible flow $f(e)$ for all $e \in E(G)$.
- ➋ Find an augmenting path P . If none exists then stop.
- ➌ Compute γ , the capacity of an augmenting path P . Augment the flow from s to t and go to 2.

Increase flow by γ on forward arcs and decrease flow by γ on backward arcs. This preserves feasibility of the flow and Kirchhoff's law moreover the flow is augmented by γ .
This augmenting path can't be used again since the flow along this path is the highest possible.

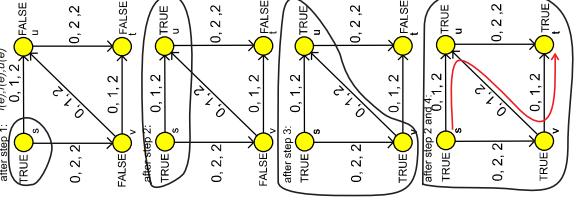
The flow from s to t is the **maximum** if and only if there is no augmenting path.

Input: Network (G, l, u, s, t) , feasible flow f .

Output: Augmenting path P .

- ➊ Label $m_s = \text{TRUE}$, $m_v = \text{FALSE} \forall v \in V(G) \setminus s$.
- ➋ If there exists $e \in E(G)$ (where e is the edge from v_i to v_j) that satisfies $m_i = \text{TRUE}$, $m_j = \text{FALSE}$ and $f(e) < u(e)$ **Then** $m_j = \text{TRUE}$.
- ➌ If there exists $e \in E(G)$ (where e is the edge from v_i to v_j) that satisfies $m_i = \text{FALSE}$, $m_j = \text{TRUE}$ and $f(e) > l(e)$ **Then** $m_i = \text{TRUE}$.

- ➍ If t is reached, then the search stops as we have found the augmenting path P . If it is not possible to mark another node, then P does not exists. In other cases go to step 2.



Finding the Augmenting Path (Labeling Procedure) for Ford-Fulkerson Algorithm

April 3, 2019 10 / 49

Network Flows

Z. Hanzálek (CTU FEE)

April 3, 2019 9 / 49

Network Flows

Z. Hanzálek (CTU FEE)

Ford-Fulkerson Algorithm

April 3, 2019 10 / 49

Network Flows

Z. Hanzálek (CTU FEE)

April 3, 2019 11 / 49

Network Flows

Z. Hanzálek (CTU FEE)

April 3, 2019 12 / 49

Network Flows

Z. Hanzálek (CTU FEE)

April 3, 2019 11 / 49

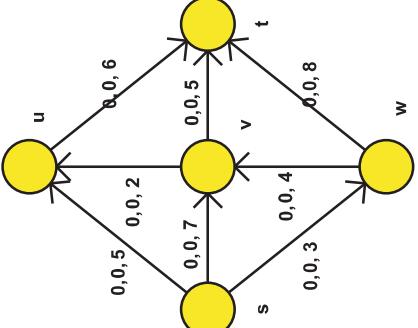
Network Flows

Z. Hanzálek (CTU FEE)

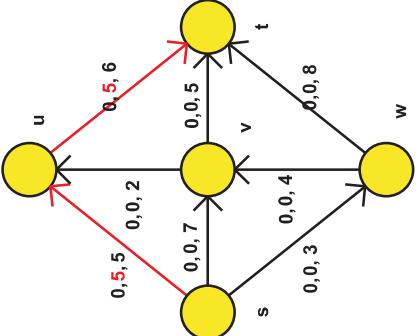
Ford-Fulkerson Algorithm Simple Example

Ford-Fulkerson Algorithm Simple Example

arcs are labeled by $l(e), f(e), u(e)$



arcs are labeled by $l(e), f(e), u(e)$

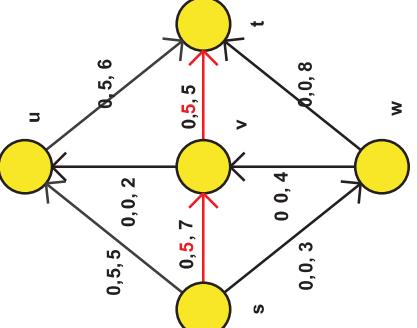


arcs are labeled by $l(e), f(e), u(e)$

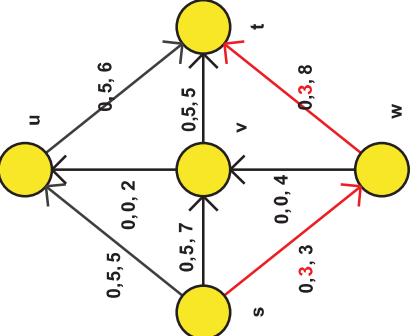
Ford-Fulkerson Algorithm Simple Example

Ford-Fulkerson Algorithm Simple Example

arcs are labeled by $l(e), f(e), u(e)$



arcs are labeled by $l(e), f(e), u(e)$

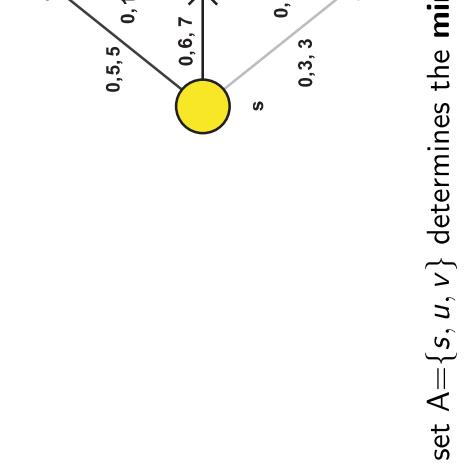
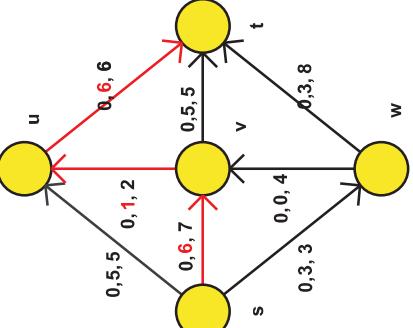


Ford-Fulkerson Algorithm Simple Example

Ford-Fulkerson Algorithm Simple Example

arcs are labeled by $l(e), f(e), u(e)$

arcs are labeled by $l(e), f(e), u(e)$



set $A = \{s, u, v\}$ determines the **minimum capacity cut**

Ford-Fulkerson Algorithm Example with Backward Edge

This example illustrates that we **can't omit the backward edges** when creating an augmenting path. Otherwise we are not able to obtain the maximum flow (right) from the initial flow (left).
The arcs are labeled by: $l(e), f(e), u(e)$.

Capacity of augmenting path is equal to 2.

Final flow is the maximal one.
Find a minimum capacity cut.

Ford and Fulkerson [1956]

The value of the maximum flow from s to t is equal to the capacity of the **minimum cut**. This property follows from LP duality.

When the labeling procedure stops, since there is no augmenting path, the minimum cut is given by the labeled vertices (the minimum cut is equal to the set of edges that do not allow further labeling). Therefore, $f(e) = u(e)$ holds for all $e \in \delta^+(A)$ and $f(e) = l(e)$ holds for all $e \in \delta^-(A)$.

The value of the max. flow is equal to the capacity of the minimum cut: $\sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) = \sum_{e \in \delta^+(A)} u(e) - \sum_{e \in \delta^-(A)} l(e)$.

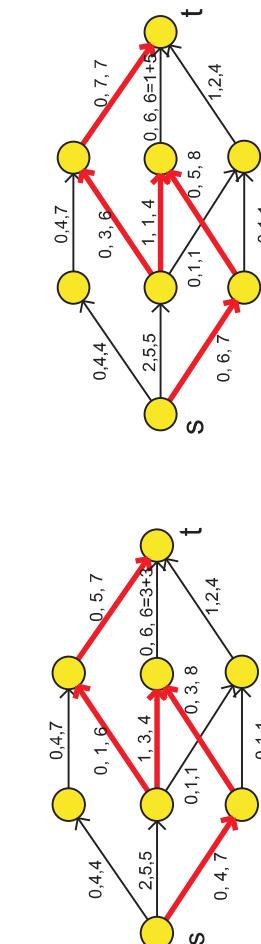
Minimum Cut Problem

Cut

The cut in G is an edge set $\delta(A)$ with $s \in A$ and $t \in V(G) \setminus A$ (i.e. the cut separates nodes s and t). The **minimum cut** is the cut of minimum capacity $C(A) = \sum_{e \in \delta^+(A)} u(e) - \sum_{e \in \delta^-(A)} l(e)$.

Capacity of augmenting path is equal to 2.

The value of the maximum flow from s to t is equal to the capacity of the minimum cut: $\sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e) = \sum_{e \in \delta^+(A)} u(e) - \sum_{e \in \delta^-(A)} l(e)$.



Example Flow.cut.blocking: Blocking of Gasoline Supply

Integrality

A tank is located at one node t in gasoline pipeline directed graph $G(E, V)$. Gasoline supply nodes are denoted by the set $S \subset V$ such that $t \notin S$. Let pipe e has minimum throughput $l(e)$ and maximum throughput $u(e)$. The adversary might decrease the maximum throughput by $\alpha(e) \in \langle l(e), u(e) \rangle$ if he/she makes effort of volume $k\alpha(e)$, where k is a positive constant.

- Consider $l_{ij} = 0$ and determine the minimal effort required to isolate the tank from a gasoline.
- How the problem changes when we consider non-zero minimum throughput?

Integral Flow Theorem (Dantzig and Fulkerson [1956])

If the capacities of the network are integers, and there exists a feasible flow, then there exists an **integer-valued maximum flow**.

This follows from total unimodularity of the incidence matrix of a digraph G , which is matrix \mathbf{A} in LP formulation $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$.

To find an augmenting path takes $O(|E|)$ time.
For integer capacities, each iteration increases the flow by at least 1.

The maximum number of iterations is equal

to the volume of minimum cut F and it is equal to the capacity of the minimum cut bounded by $|E| \cdot U$ where U is the highest edge capacity, giving alg. complexity of $O(|E| \cdot F)$ and $O(|E|^2 \cdot U)$ respectively.

For non-integer capacities and non-integer flow the algorithm might not terminate at all [Korte Vygen] Flows, exercise 2.
Edmonds and Karp [1972])

When choosing the augmenting path, if we always choose the **shortest one** (e.g., by Breadth First Search), time complexity is $O(|E|^2 \cdot |V|)$.

Proof: [Korte Vygen] Theorem 8.14.

Ford-Fulkerson Algorithm - Time Complexity

To find an augmenting path takes $O(|E|)$ time.

For integer capacities, each iteration increases the flow by at least 1.

The maximum number of iterations is equal to the capacity of the minimum cut bounded by $|E| \cdot U$ where U is the highest edge capacity, giving alg. complexity of $O(|E| \cdot F)$ and $O(|E|^2 \cdot U)$ respectively.

For non-integer capacities and non-integer flow the algorithm might not terminate at all [Korte Vygen] Flows, exercise 2.

Edmonds and Karp [1972])

When choosing the augmenting path, if we always choose the **shortest one** (e.g., by Breadth First Search), time complexity is $O(|E|^2 \cdot |V|)$.

Example Flow.lower.representatives: Distinct Representatives - Existence of Lower Bound [AMO93]

Assignment problem with an additional constraint.

Let us have n residents of a town, each of them is a member of at least one club k_1, \dots, k_l and belongs to exactly one age group ρ_1, \dots, ρ_r . Each club must nominate one of its members to the town's governing council so that the number of council members belonging to the age group is constrained by its minimum and maximum. One resident represents at most one club.

Find a feasible assignment of the representatives or prove that it does not exist. Formulate as the Maximum Flow Problem.

Example Flow.lower.rounding: [AMO93]

Matrix Rounding Problem - Existence of Lower Bound

This application is concerned with **consistent rounding of the elements, row sums, and column sums** of a matrix. We are given a 3×3 matrix of real numbers x_{ij} with row sums vector r and column sums vector c . We can round any real number $a \in \{x_{11} \dots x_{33}, r_1 \dots r_3, c_1 \dots c_3\}$ to the next smaller integer $\lfloor a \rfloor$ or to the next larger integer $\lceil a \rceil$, and the decision is completely up to us. The problem requires that we round all matrix elements and the following constraints hold:

- the sum of the rounded elements in each row is equal to the rounded row sum r_i
 - the sum of the rounded elements in each column is equal to the rounded column sum c_j
- We refer to such rounding as a *consistent rounding*.
- The objective is to maximize the sum of matrix entries (due to the constraint it is equal to the sum of the row sums and at the same time to the sum of the column sums).

Example - Transport Problem

Z. Hanzálek (CTU FEE) April 3, 2019 20 / 49

Feasible Flow with Balances - Reduction to Maximum Flow

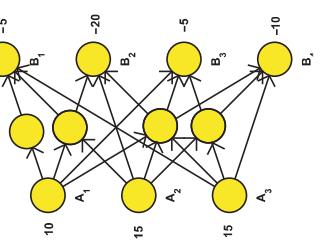
Z. Hanzálek (CTU FEE) April 3, 2019 21 / 49

We assume several sources and several sinks.
Later we show its polynomial reduction to the Maximum Flow problem (with one source and one sink).

Feasible flow with balances

- Instance:** Let (G, I, u, b) be a network where G is a digraph with upper bounds $u : E(G) \rightarrow \mathbb{R}_0^+$, $I : E(G) \rightarrow \mathbb{R}_0^+$ and with:
 - balance $b : V(G) \rightarrow \mathbb{R}$ that represents the supply/consumption of the nodes and satisfies $\sum_{v \in V(G)} b(v) = 0$.
- Goal:** Decide if there exists a feasible flow f which satisfies $\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v)$ for all $v \in G(V)$.

There are **suppliers** (represented by nodes with $b(v) > 0$) and **consumers** (represented by nodes with $b(v) < 0$) of a **product**. Our goal is to decide whether it is possible to transport all the product from the suppliers to the consumers through the network with upper bounds u . The problem is described by a network (or graph) where the edges represent pipelines, highways or railways of some transportation capacity.



Goal

The goal is to decide whether it is possible to transport all the product from suppliers A to consumers B through the network with capacities u .

We transform this problem to the **maximum flow problem**:

- We add a new node s called the source and add edges (s, v) with the lower bound $l_v = 0$ and the upper bound $u_v = b(v)$ for every node that satisfies $b(v) > 0$
- We add a new node t called the sink and add edges (v, t) with the lower bound $l_v = 0$ and the upper bound $u_v = -b(v)$ for every node that satisfies $b(v) < 0$
- We solve the maximum flow problem

- If the maximum flow saturates all edges leaving s and/or entering t , then the answer to the feasible flow decision problem is YES.

Z. Hanzálek (CTU FEE) April 3, 2019 22 / 49

Z. Hanzálek (CTU FEE) April 3, 2019 23 / 49

Network Flows

Z. Hanzálek (CTU FEE)

Network Flows

How to Find an Initial Feasible Flow for Ford-Fulkerson Algorithm?

How to find an initial feasible flow for Ford-Fulkerson algorithm?

If $\forall e \in E(G); l(e) = 0$ - easy solution - we use zero flow which satisfies Kirchhoff's law.

If $\exists e \in E(G); l(e) > 0$, we transform the feasible flow problem to the **Feasible Flow with Balances and zero lower bounds** as follows:

1) We transform the maximum flow problem (with non-zero lower bounds) to a circulation problem by **adding an arc from s to t to s of infinite capacity**. Consequently, the Kirchhoff's law applies to nodes s and t . Therefore, a feasible circulation must satisfy:

$$\begin{aligned} l(e) \leq f(e) \leq u(e) \\ \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = 0 \end{aligned}$$

2) Substituting $f(e) = f(e)' + l(e)$, we obtain the transformed problem:

$$\begin{aligned} 0 \leq f(e)' \leq u(e) - l(e) \\ \sum_{e \in \delta^+(v)} f(e)' - \sum_{e \in \delta^-(v)} f(e)' = \sum_{e \in \delta^-(v)} l(e) - \sum_{e \in \delta^+(v)} l(e) \end{aligned}$$

3) This is a **Feasible Flow with Balances and zero lower bounds**

because $\sum_{v \in V(G)} b(v) = 0$ (notice that $l(e)$ appears twice in summation, once with a positive and once with a negative sign).

4) While solving this decision problem (i.e. adding s' , t' and solving the maximum flow problem with zero lower bounds) we obtain the initial feasible circulation/flow or decide that it does not exist.

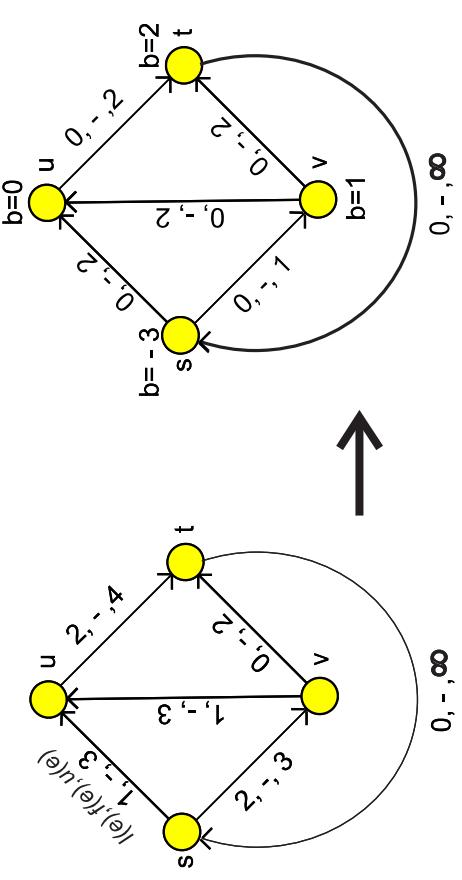
Conclusion: finding of the **initial flow with nonzero lower bounds** can be transformed to the **Feasible Flow with Balances and zero lower bounds** which can be transformed to the **Maximum Flow with zero lower bounds**.

Z. Hanzálek (CTU FEE)

April 3, 2019

24 / 49

Example: Transformation of MaxFlow with nonzero lower bounds to FeasFlow with Balances and zero lower bounds

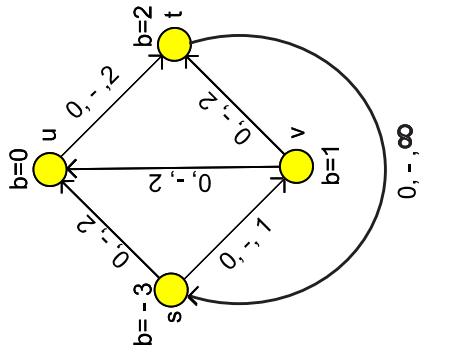


Z. Hanzálek (CTU FEE)

April 3, 2019

25 / 49

Example: Transformation of Feasible Flow with Balances to Maximum Flow, both with zero lower bounds

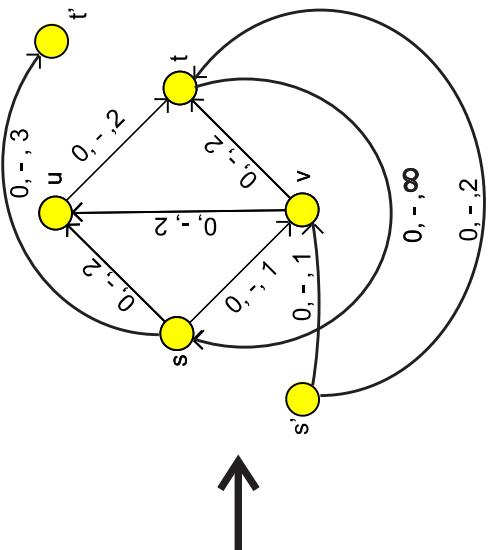


Z. Hanzálek (CTU FEE)

April 3, 2019

26 / 49

Example: Transformation of Feasible Flow with Balances and zero lower bounds to Maximum Flow with zero lower bounds

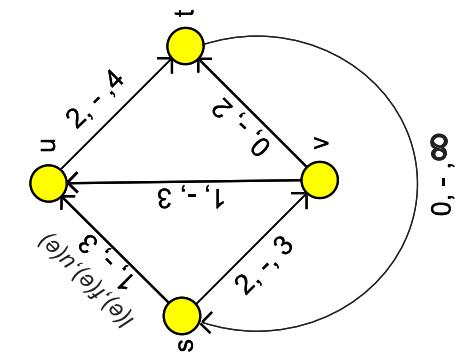


Z. Hanzálek (CTU FEE)

April 3, 2019

27 / 49

Maximum Flow with nonzero lower bounds

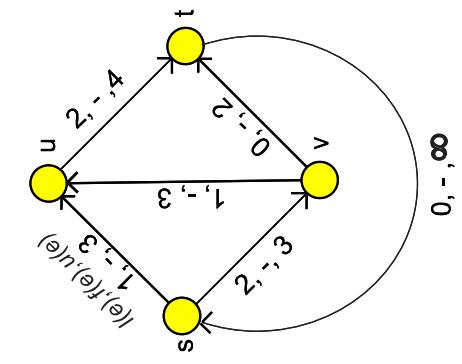


Z. Hanzálek (CTU FEE)

April 3, 2019

28 / 49

Maximum Flow with Balances and zero lower bounds

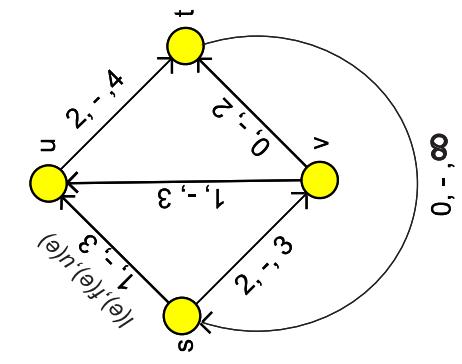


Z. Hanzálek (CTU FEE)

April 3, 2019

29 / 49

Maximum Flow with Balances and zero lower bounds



Z. Hanzálek (CTU FEE)

April 3, 2019

30 / 49

Minimum Cost Flow

Minimum Cost Flow - LP Formulation

Extension of the Maximum flow problem - we consider the edge costs and the supply/consumption of the nodes.

Minimum Cost Flow

- Instance:** 5-tuple (G, l, u, c, b) where G is a digraph, $u : E(G) \rightarrow \mathbb{R}_0^+$ represents the upper and $l : E(G) \rightarrow \mathbb{R}_0^+$ the lower bounds and:

- cost of arcs** $c : E(G) \rightarrow \mathbb{R}$
- balance $b : V(G) \rightarrow \mathbb{R}$ that represents the supply/consumption of the nodes and satisfies $\sum_{v \in V(G)} b(v) = 0$.

- Goal:** Find the feasible flow f that minimizes $\sum_{e \in E(G)} f(e) \cdot c(e)$ (we want to transport the flow through the network at the lowest possible cost) and satisfies $\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v)$ for all $v \in G(V)$, or decide that it does not exist.

Variable $f(e) \in \mathbb{R}_0^+$ represents the flow on edge $e \in E(G)$.

$$\begin{array}{ll} \min & \sum_{e \in E(G)} c(e) \cdot f(e) \\ \text{s.t.} & \sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v) \\ & l(e) \leq f(e) \leq u(e) \end{array} \quad \begin{array}{l} v \in V(G) \\ e \in E(G) \end{array}$$

- The **Maximum Flow problem** can be polynomially reduced to the Minimum Cost Flow problem:
- Add an edge from t to s with the upper bound equal to ∞ and the cost -1 .
 - Set the cost of every other edge to 0.
 - Set $b(v) = 0$ for all nodes including s and t .
 - Minimum cost circulation (it will be negative) maximizes the flow on the added edge.

Minimum Cost Flow is quite general problem

The **Shortest Path from s to t** can be reduced to the Min-Cost Flow:

- Use LP formulation of Min-Cost Flow.
- Set $b(s) = 1$ and $b(t) = -1$. Set $b(v) = 0 \quad \forall v \in V \setminus \{s, t\}$.
- Set $l(e) = 0$ and $u(e) = \infty \quad \forall e \in E$.
- You obtain (primal) LP formulation of the Shortest Path Problem (follow example of totally unimodular matrix A in ILP lecture).

The **Chinese Postman Problem** (A postman has to deliver the mail within his district given by strongly connected directed graph. To do this, he must start at the post office, walk along each street at least once, and finally return to the post office. The problem is to find a postman's tour of minimum length.) can be reduced to the Min-Cost Flow:

- Set $b(v) = 0 \quad \forall v \in V$.
- Set $l(e) = 1$ and $u(e) = \infty \quad \forall e \in E$.

There is a postman's tour using each edge exactly once (i.e. Eulerian walk) iff every vertex has equal indegree and outdegree (i.e. Eulerian digraph).

$$\begin{array}{ll} \text{Input:} & \text{Network } (G, l, u, b, c). \\ \text{Output:} & \text{Minimum cost flow } f. \\ \text{①} & \text{Find feasible flow } f \text{ while solving Feasible Flow with Balances in } (G, l, u, b). \\ \text{②} & \text{Build residual graph } G_f \text{ with respect to } f. \text{ Find a negative-cost cycle } C \text{ in residual graph } G_f. \text{ If none exists then stop.} \\ \text{③} & \text{Compute } \gamma = \min_{e \in E(C)} u_f(e). \text{ Augment } f \text{ along } C \text{ by } \gamma. \text{ Go to 2.} \end{array}$$

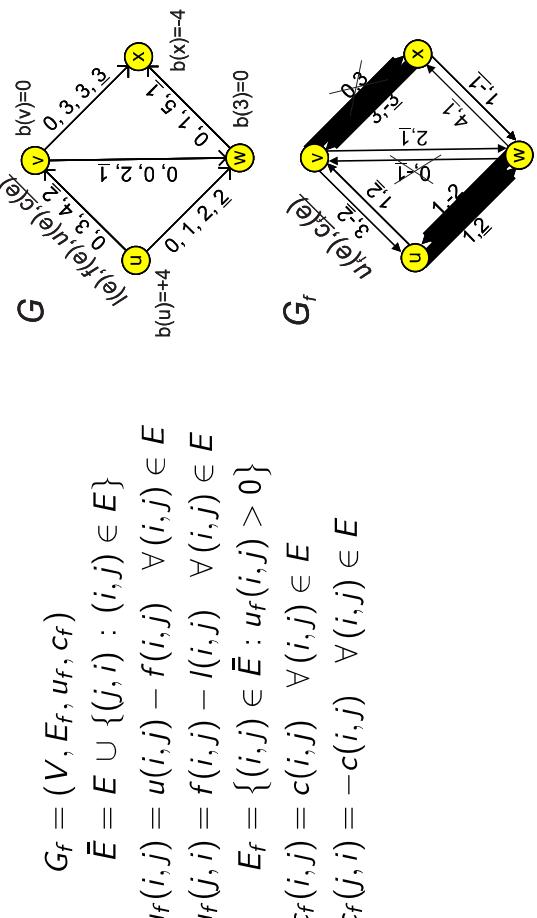
The negative-cost cycle C in residual graph can be found as follows:

- Remove all edges with $u_f(e) = 0$ from residual graph (G_f, u_f, c_f) .
- Add node s and connect it to every $v \in V(G_f)$ by edge $e(s, v)$ with cost $c_f(e) = 0$.
- Use, for example, Bellman-Ford algorithm to detect a negative-cost cycle (with respect to cost c_f).
- Recover negative-cost cycle C or state that it does not exist.

Residual Graph

Example: Cycle Cancelling Algorithm with Residual Graph

The residual graph shows where an extra capacity might be found.
The residual graph G_f is constructed from network (G, I, u, b, c) and specific feasible flow f as follows:



Cycle Cancelling Algorithm - Complexity and its improvements

Analysis of the Cycle Cancelling Algorithm:

Let $U = \max_{(i,j) \in E} u(i,j)$ and $C = \max_{(i,j) \in E} |c(i,j)|$.

For any feasible flow f its cost $c(f)$ is bounded as:
 $-|E| \cdot C \cdot U \leq c(f) \leq |E| \cdot C \cdot U$

Each iteration decreases the objective by at least 1.

Conclusion: there are at most $2|E| \cdot C \cdot U$ iterations.

The Bellman-Ford complexity is $O(|V| \cdot |E|^2 \cdot C \cdot U)$.

Idea for improvement:

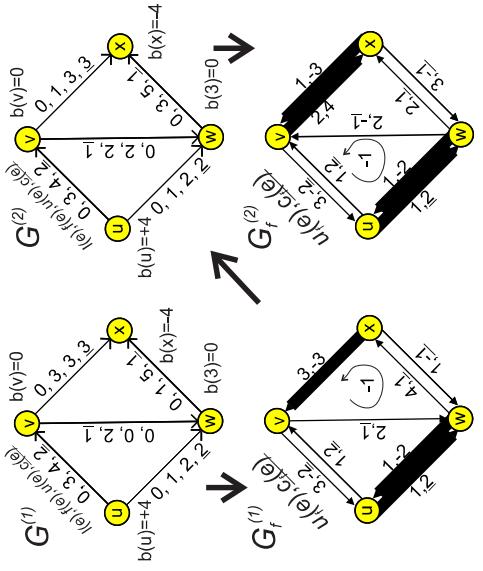
Find the most negative cycle. But this problem is NP-hard.

Solution by a Minimum Mean Cycle Cancelling Algorithm:

Find a circuit C whose mean weight $\frac{c(E(C))}{|E(C)|}$ is minimum in $O(|V| \cdot |E|)$.

Minimum Mean Cycle Cancelling Algorithm runs in $O(|V|^2 \cdot |E|^3 \cdot \log |V|)$.

- $G^{(1)}$ with a feasible flow in the first iteration.
- $G_f^{(1)}$ with a negative-cost cycle $C = (v, w, x)$, $\gamma = \min_{e \in E(C)} u_f(e) = 2$, min cost criterion will change by $-1 \cdot 2$.
- $G^{(2)}$ with the feasible flow in the second iteration.
- $G_f^{(2)}$ with a negative-cost cycle $C = (v, u, w)$, $\gamma = \min_{e \in E(C)} u_f(e) = 1$, min cost criterion will change by $-1 \cdot 1$, new flow will be feasible.



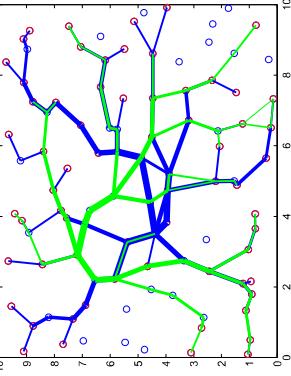
Minimum Cost Multicommodity Flow

So far, we have assumed to be transporting just one commodity.
Let M be the **commodity set** transported through the network.

Every commodity has several sources and several sinks.

Variable $f^m(e) \in \mathbb{R}_0^+$ is the flow of commodity $m \in M$ along edge $e \in E(G)$.

- Example:** sensor network with two commodities and one sink for every commodity:
- source nodes are measuring **temperature(green)** and/or **humidity(blue)** and sending the data to one concentrator (sink) for temperature and one for humidity
 - flow (amount of data per time unit) Communication links:
 - capacity (amount of data per time unit)



Network Flows

Network Flows

Network Flows

Network Flows

Minimum Cost Multicommodity Flow

LP Formulation

Minimum cost multicommodity flow

- Instance: 5-tuple $(G, I, u, c, b^1 \dots b^{|M|})$ where G is a digraph with upper bounds $u : E(G) \rightarrow \mathbb{R}_0^+$, lower bounds $I : E(G) \rightarrow \mathbb{R}_0^+$ and costs $c : E(G) \rightarrow \mathbb{R}$ and with:
 - vectors $b^m : V(G) \rightarrow \mathbb{R}$ that express **(supply/consumption)** of nodes by commodity m . $\sum_{v \in V(G)} b^m(v) = 0$ for all commodities $m \in M$.

- Goal: Find the feasible flow f whose cost

$\sum_{e \in E(G)} \sum_{m \in M} f^m(e) \cdot c(e)$ is minimal (we want to transport the flow as cheap as possible) or decide that such a flow does not exist.

Feasible flow satisfies $\sum_{e \in \delta^+(v)} f^m(e) - \sum_{e \in \delta^-(v)} f^m(e) = b^m(v)$ for all $v \in G(V)$ and all $m \in M$.

Variable $f^m(e) \in \mathbb{R}_0^+$ represents the flow of commodity $m \in M$ along edge $e \in E(G)$.

$$\begin{aligned} & \min \sum_{e \in E(G)} \sum_{m \in M} f^m(e) \cdot c(e) \\ \text{s.t. } & \sum_{e \in \delta^+(v)} f^m(e) - \sum_{e \in \delta^-(v)} f^m(e) = b^m(v) \\ & I(e) \leq \sum_{m \in M} f^m(e) \leq u(e) \end{aligned} \quad \begin{matrix} v \in V(G), m \in M \\ e \in E(G) \end{matrix}$$

- 1st Kirchhoff's law is satisfied in every node for **every commodity**.
- Multicommodity flow can be solved by LP - **in polynomial time**.
- Integer-valued flow is not assured since matrix A in LP is **not totally unimodular** (see $I(e) \leq \sum_{m \in M} f^m(e) \leq u(e)$)
- Practical experience: ILP formulation which guarantees integer-valued solution can be solved even for big instances in acceptable time.

Matching - Introduction

Matching is the set of arcs $P \subseteq E(G)$ in graph G , such that the endpoints are all different (no arcs from P are incident with the same node). When all nodes of G are incident with some arc in P , we call P a **perfect matching**. Problems:

- Maximum Cardinality Matching Problem** - we are looking for matching with the maximum number of edges.
- Maximum Cardinality Matching in Bipartite Graphs** - special case of problem a.

- Minimum Weight Matching** in a weighted graph - the cheapest matching from the set of all **Maximum Cardinality Matchings**.
- Minimum Weight Perfect Matching** in a complete bipartite graph whose parts have the same number of nodes. This problem is also called an **Assignment Problem** and it is a special case of problem c) and also a special case of the **minimum cost flow problem**. These problems can be solved in polynomial time. We will present some algorithms for a), b) and d) only.

a) Maximum Cardinality Matching Problem - Solution Using M -alternating Path

Definition: Let G be a graph (bipartite or not), and let M be a matching in G . A path P is an **M -alternating path** if $E(P) \setminus M$ is a matching. An M -alternating path is **M -augmenting** if its endpoints are not covered by M .

Theorem: Let G be a graph (bipartite or not) with some matching M . Then M is maximum if and only if there is no M -augmenting path.

Algorithm:

- Find the arbitrary matching.
- Find the M -alternating path with uncovered endpoints.
- Exchange (i.e. augment) the matching along the alternating path. Repeat as long as such a path does exist.

b) Maximum Cardinality Matching in Bipartite Graphs - Solution by Maximum Flow

d) Assignment Problem - Solution by Minimum Cost Flow

We have n employees and n tasks and we know the cost of execution for each possible employee-task pair.

Goal

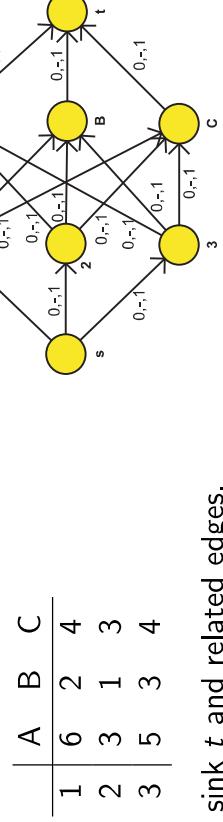
Can be transformed to the **maximum flow** problem:

- Add source s and edge (s, i) for all $i \in X$
- Add sink t and edge (j, t) for all $j \in Y$
- Edge orientation should be from s to X , from X to Y and from Y to t
- The upper bound of all edges is equal to 1 and the lower bound is equal to 0
- By solving the maximum flow from s to t we obtain maximum cardinality matching

Assign one task per employee while minimizing the total cost.

Can be solved by transformation to **minimum cost flow problem**:

- Example: cost of execution of task 1,2,3 by employee A,B,C



	A	B	C
1	6	2	4
2	3	1	3
3	5	3	4

Add source s , sink t and related edges.

Alternatively we can solve the assignment problem by Hungarian Alg.

Hungarian Algorithm - Solution of Assignment Problem (i.e. Minimum Weight Perfect Matching in Complete Bipartite Graph whose Sets Have the Same Cardinality)

Description: G - Complete undirected bipartite graph containing sets X, Y which satisfy $|X| = |Y| = n$.

- Edge costs are arranged in the matrix, where element $c_{ij} \in \mathbb{R}_0^+$ represents the cost of edge $(i, j) \in X \times Y$.

The basic ideas of the Hungarian algorithm:

- **Assign** an arbitrary real number $p(v)$, to every vertex $v \in V(G)$. These numbers are used to transform the costs $c_{ij}^P = c_{ij} - p_i^X - p_j^Y$.
- For every perfect matching, this transformation **changes the total cost by the same value** (every node participates exactly once).
- Thanks to this, the cheapest perfect matching is still given by the **same set of edges**.

Theorem

P is the optimal solution to the assignment problem if the equality graph G^P contains perfect matching.

Idea of the proof: The cost of matching P in G^P is equal to zero. There is no cheaper matching, since it is the feasible rating, where $c_{ij}^P \geq 0$.

Solution of Assignment Problem - Hungarian Algorithm

Definition: We call numbers assigned to nodes p a **feasible rating** if all transformed costs are nonnegative, i.e. $c_{ij}^P \geq 0$.

Definition: When p is the feasible rating, we call G^P an **equality graph** if it is a factor of graph G which contains only edges with **zero cost**.

April 3, 2019 40 / 49

Z. Hanzálek (CTU FEE) Network Flows

April 3, 2019 41 / 49

Z. Hanzálek (CTU FEE) Network Flows

April 3, 2019 41 / 49

Z. Hanzálek (CTU FEE) Network Flows

April 3, 2019 42 / 49

Z. Hanzálek (CTU FEE) Network Flows

April 3, 2019 43 / 49

Z. Hanzálek (CTU FEE) Network Flows

Hungarian Algorithm - Example

- ② Several zeros appeared in the matrix and several edges appeared in G^P . On the contrary, edge (5,6) disappeared.
- ③ The cardinality of the matching cannot be increased now.

④ Find sets A (blue) and B (green). Minimum $d = 1$.

	0	0	4	1	2	5	p_i^x
1	0	4	3	0	1	0	
10	1	0	0	0	0	0	
3	11	1	0	8	11	1	
5	3	3	3	0	4	1	
1	7	0	2	6	7	1	
p_j^y	2	0	0	0	0	-4	

- ⑤ Now, perfect matching does exist in graph G^P . The cost is equal to 18 (sum of the ratings).

	0	0	5	2	3	6	p_i^x
0	0	3	3	0	1	0	
9	0	0	0	0	0	0	
2	10	1	0	8	11	2	
4	2	3	3	0	4	2	
0	6	0	2	6	7	2	
p_j^y	2	0	-1	-1	-1	-5	

Z. Hanzálek (CTU FEE)

Network Flows April 3, 2019 48 / 49

Knapsack Problem

- 1 Problem formulation
- Knapsack Problem
 - Fractional Knapsack Problem

Zdeněk Hanzálek
hanzalek@fel.cvut.cz

CTU in Prague

April 10, 2017

- 2 Solutions and Algorithms
- Simple Solution to Fractional Knapsack Problem
 - 2-Approximation Algorithm
 - Dynamic programming
 - Approximation Scheme for Knapsack

- 3 Summary

References

- ① Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin.
Network Flows: Theory, Algorithms, and Applications. Prentice Hall, 1993.
- ② Jiří Demel.
Grafy a jejich aplikace. Academia, 2002.
- ③ B. H. Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms. Springer, fourth edition, 2008.

Table of contents

Knapsack problem

- **Instance:** Nonnegative integers $n, c_1, \dots, c_n, w_1, \dots, w_n, W$, where n represents the number of items, c_1, \dots, c_n represents the cost of each item, w_1, \dots, w_n represents the weight of each item and W is the maximum weight to be carried in the knapsack.
- **Goal:** Find a subset $S \subseteq \{1, \dots, n\}$ such that $\sum_{j \in S} w_j \leq W$ and $\sum_{j \in S} c_j$ is the maximum.
- It is one of the "easiest" NP-hard problems.
- Sometimes it is called **0/1 Knapsack problem**.

Solution of Fractional Knapsack Problem - Dantzing [1957]

- if $\sum_{j=1}^n w_j > W$ (otherwise it has a trivial solution)
- order and re-index the items by their relative cost:

$$\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \dots \geq \frac{c_n}{w_n}$$
- in the ordered sequence find the first item which does not fit in the knapsack ($h := \min \left\{ j \in \{1, \dots, n\} : \sum_{i=1}^j w_i > W \right\}$)
- in order to find the optimal solution, we cut off a part of h -th item, so that this part fits in the knapsack:
 - $x_j := 1$ for $j = 1, \dots, h-1$
 - $x_h := \frac{w_j}{W - \sum_{i=1}^{h-1} w_i}$
 - $x_j := 0$ for $j = h+1, \dots, n$
- sorting of the items takes $O(n \log n)$, computing h can be done in $O(n)$ by simple linear scanning, so this algorithm solves the **Fractional Knapsack problem** in $O(n \log n)$
- there is an even faster algorithm ([1] page 440) which can solve this problem in $O(n)$ by reduction to the **Weighted Median problem**

While relaxing on the indivisibility of each item, we formulate a new problem:

Fractional Knapsack problem

- **Instance:** Nonnegative integers $n, c_1, \dots, c_n, w_1, \dots, w_n, W$, where n represents the number of items, c_1, \dots, c_n represents the cost of each item, w_1, \dots, w_n represents the weight of each item and W is the maximum weight to be carried in the knapsack.
- **Goal:** Find the rational numbers $x_1, \dots, x_j, \dots, x_n$ such that $0 \leq x_j \leq 1$ and $\sum_{j=1}^n x_j \cdot w_j \leq W$ and $\sum_{j=1}^n x_j \cdot c_j$ is the maximum.

- Since the items can be divided (continuous variable x_j), we can solve this problem in polynomial time.

2-Approximation Algorithm for Knapsack

r-approximation algorithm for maximization

- Algorithm A for objective function J maximization is called r-approximation if there exists a number $r \geq 1$ such that $J^A(I) \geq \frac{1}{r} J^*(I)$ for all instances I of this problem.

Theorem

Let $n, c_1, \dots, c_n, w_1, \dots, w_n, W, h$ are nonnegative integers that satisfy:

- $w_j \leq W$ pro $j = 1, \dots, n$
- $\sum_{i=1}^n w_i > W$
- $\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \dots \geq \frac{c_n}{w_n}$
- $h = \min \left\{ j \in \{1, \dots, n\} : \sum_{i=1}^j w_i > W \right\}$

Then choosing the better of the two solutions $\{1, \dots, h-1\}$ or $\{h\}$ is **2-approximation algorithm for the Knapsack problem** with the time complexity $O(n)$.

2-Approximation Algorithm for Knapsack

Dynamic Programming (Integer Costs) for Knapsack

Proof:

- Given any instance of the **Knapsack problem**, we can omit all items whose weight is bigger than the knapsack maximum load.
- If $\sum_{i=1}^n w_i \leq W$ then the whole set of items is an optimal solution.
- Since $\sum_{i=1}^h c_i$ is an upper bound on the optimum value, the better of two solutions $\{1, \dots, h-1\}$ and $\{h\}$ achieves at least half of the optimum value.

Notes about approximation algorithms:

- An approximation algorithm guarantees, that even in the worst case, the value of the objective function will be proportional to the optimum value. The frequency of the worst case is not considered by the approximation algorithm.
- ϵ , the relative deviation from the optimum, is sometimes used instead of the asymptotic performance ratio r , so that $r = 1 + \epsilon$.
- In this case, it is meaningless to state the absolute error. Why?

Z. Hanzák (CTU) April 10, 2017 7 / 15 Knapsack Problem

Dynamic Programming (Integer Costs) for Knapsack

Input: Costs $c_1, \dots, c_n \in \mathbb{Z}_0^+$, weights $w_1, \dots, w_n, W \in \mathbb{R}_0^+$.

Output: $S \subseteq \{1, \dots, n\}; \sum_{j \in S} w_j \leq W$ and $\sum_{j \in S} c_j$ is maximum.

Let C be the arbitrary upper bound of the solution, e.g. $C = \sum_{j=1}^n c_j; x_0^0 := 0; x_k^0 := \infty$ for $k = 1, \dots, C$;

```
for j := 1 to n do
    for k := 0 to C do
        x_k^j := x_k^{j-1}; s_k^j := 0;
        for k' := c_j to C do
            if x_k^{j-1} + w_j <= min{W, x_k^{j-1}} then
                x_k^j := x_k^{j-1} + w_j; s_k^j := 1;
            end
        end
    end
end
```

```
i := max{k in {0, ..., C} : x_k^n < infinity}; S := ∅;
for j := n downto 1 do
    if s_j^i = 1 then S := S ∪ {j}; i := i - c_j;
end
```

- Pseudopolynomial algorithm with time complexity $O(nC)$.
- Variable x_k^j represents the **minimum weight with cost k which can be achieved as a selection of items from set $\{1, \dots, j\}$**
- (*) Item j is added to the selection of items from $1, \dots, j$ if for the given price k this set reaches the **lower or equal weight as set $1, \dots, j-1$** .

The algorithm computes these values using the recursion formula:

$$x_k^j = \begin{cases} x_{k-c_j}^{j-1} + w_j & \text{if item } j \text{ was added;} \\ x_k^{j-1} & \text{if item } j \text{ wasn't added.} \end{cases}$$

- In variable s_k^j we memorize which of the two possible cases has happened. It is later used to reconstruct the selection.

Blackboard example (integer costs):

$n = 4, w = (21, 35, 52, 17), c = (10, 20, 30, 10), W = 100.$

Z. Hanzák (CTU) April 10, 2017 8 / 15 Knapsack Problem

Dynamic programming for Knapsack

Dynamic programming overview

- Pseudopolynomial Algorithm
- State space (may be represented by a graph) is constructed due to **integer weights or costs** of items.
- Due to the **optimal substructure** property of the problem, the optimal solution may be found by the **recurrent formula** using **Bellman's Principle of Optimality**.
- The problem has **overlapping subproblems**. State space is not the tree, it contains **diamonds**, places in the state space where we keep only the better of two possible solutions - refer to item (*) - prevents exponential growth of the state space size.

- If the **weights are integers**, we can solve the problem by dynamic programming while selecting the solution having the **higher** cost for given weight (*). While initializing $x_k^0 := -\infty$ for $k = 1, \dots, C$.
- Blackboard example (integer weights):
- $n = 4, w = (2, 3, 4, 5), c = (3.1, 4.2, 5.1, 4.3), W = 8.$

Z. Hanzák (CTU) April 10, 2017 9 / 15 Knapsack Problem

Z. Hanzák (CTU) April 10, 2017 10 / 15 Knapsack Problem

Complexity Reduction by Rounding Data

Approximation Scheme for Knapsack

Input: Nonnegative integer numbers $n, c_1, \dots, c_n, w_1, \dots, w_n, W$. Number $\epsilon > 0$.

Output: Subset $S \subseteq \{1, \dots, n\}$ such that $\sum_{j \in S} w_j \leq W$ and $\frac{\sum_{j \in S} c_j}{\sum_{j \in U} w_j} \geq \frac{1}{1+\epsilon}$ for all $U \subseteq \{1, \dots, n\}$ satisfying $\sum_{j \in U} w_j \leq W$.

- ① run **2-approximation algorithm for Knapsack**:
label the solution as S_1 with cost $c(S_1) = \sum_{j \in S_1} c_j$;

Divide all costs c_1, \dots, c_n by 2 and round them down.

The algorithm becomes faster, but we can obtain a suboptimal solution.
This allows us to find a tradeoff between the speed and desired optimality.
By

$$\bar{c}_j := \left\lfloor \frac{c_j}{t} \right\rfloor \text{ for } j = 1, \dots, n$$

the time complexity of the Dynamic programming algorithm for Knapsack is reduced t -times.

By

The time complexity of the dynamic programming algorithm for knapsack depends on C .

Idea

The time complexity of the dynamic programming algorithm for knapsack depends on C .

- ② $t := \max\{1, \frac{\epsilon c(S_1)}{n}\};$
 $c'_j := \left\lfloor \frac{c_j}{t} \right\rfloor \text{ for } j = 1, \dots, n;$
- ③ Run the **Dynamic programming algorithm for Knapsack** with instance $(n, c'_1, \dots, c'_n, w_1, \dots, w_n, W)$ using the upper bound $C := \frac{2c(S_1)}{t}$;

label the solution as S_2 with cost $c(S_2) = \sum_{j \in S_2} c_j$;

- ④ **if** $c(S_1) > c(S_2)$ **then** $S := S_1$;
else $S := S_2$;

Z. Hanzálek (CTU)	Knapsack Problem	April 10, 2017	11 / 15
Z. Hanzálek (CTU)	Knapsack - Summary		

- **Knapsack** is one of a few problems whose approximation algorithm can have an arbitrary small ϵ , i.e. relative deviation from the optimum.

- The choice of $t := \frac{\epsilon c(S_1)}{n}$ leads to $(1 + \epsilon)$ -approximation algorithm, but we do not show the proof of this statement.
- If ϵ is too small, i.e. $\epsilon \leq \frac{n}{c(S_1)}$, then $t = 1$, i.e. we find optimal solution while using the **Dynamic programming algorithm for Knapsack** with the upper bound from the 2-approximation algorithm for Knapsack.
- Time complexity is $O(nC) = O(n \frac{c(S_1)}{t}) = O(n \frac{c(S_1)n}{\epsilon c(S_1)}) = O(n^2 \cdot \frac{1}{\epsilon})$.

Z. Hanzálek (CTU)	Knapsack Problem	April 10, 2017	12 / 15
Z. Hanzálek (CTU)	Knapsack - Summary		

References

 B. H. Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms.
Springer, fourth edition, 2008.

Traveling Salesman Problem

Zdeněk Hanzálek
hanzalek@fel.cvut.cz

CTU in Prague

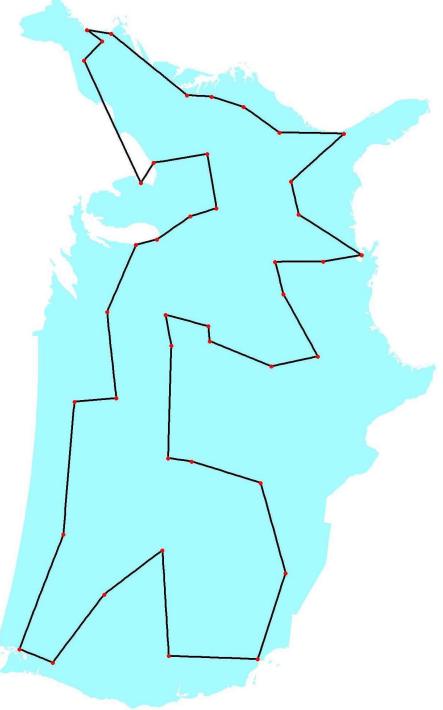
April 18, 2018

Z. Hanzálek (CTU) Knapack Problem April 10, 2017 15 / 15

Z. Hanzálek (CTU) Traveling Salesman Problem April 18, 2018 1 / 33

Solved TSP instances

- 1954 George Dantzig, Ray Fulkerson, and Selmer Johnson 49 cities, one city from each of the 48 states in the U.S.A. (Alaska and Hawaii became states only in 1959) plus Washington, D.C.



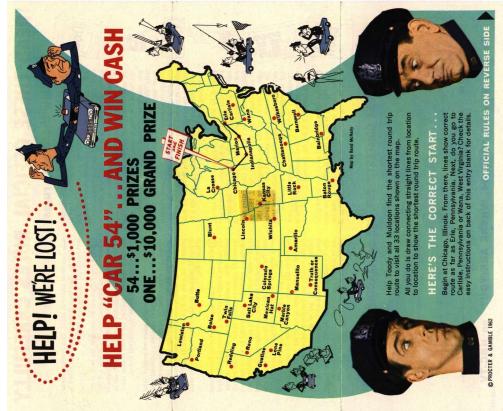
- 1 Content
- 2 Solved TSP instances in pictures
- 3 General TSP and Its Time Complexity
 - Likely Nonexistence of Polynomial r-approximation Algorithm for General TSP
- 4 Metric TSP and Approximation Algorithms
 - Double-tree Algorithm
 - Christofides' Algorithm
- 5 Tour Improvement Heuristics - Local Search k-OPT
- 6 Conclusion

Z. Hanzálek (CTU) Traveling Salesman Problem April 18, 2018 2 / 33

Z. Hanzálek (CTU) Traveling Salesman Problem April 18, 2018 3 / 33

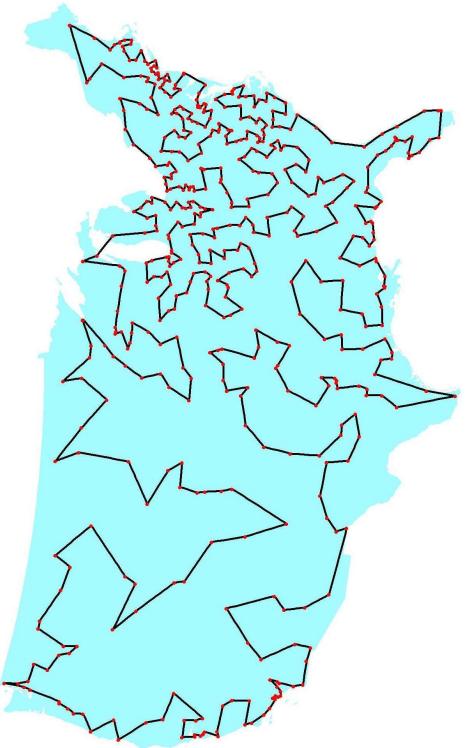
Solved TSP instances

1962
Procter and Gamble's Contest
33 cities



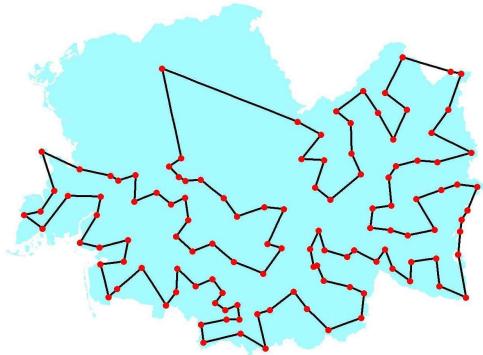
Solved TSP instances

1987
Padberg and Rinaldi
532 AT&T switch locations in the USA



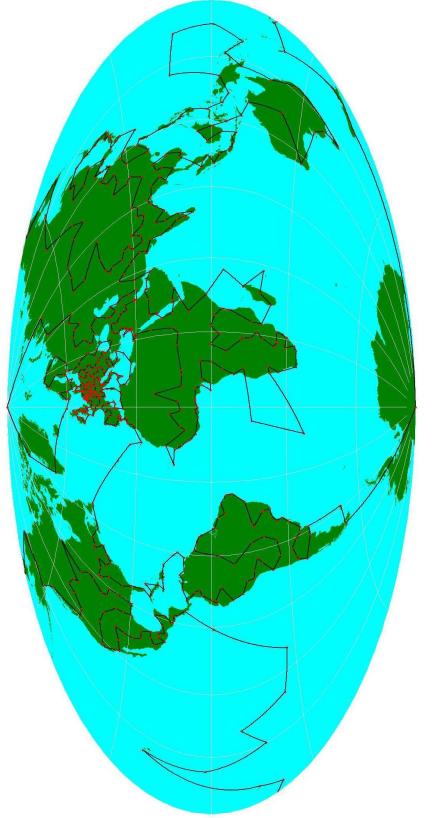
Solved TSP instances

1977
Groetschel
120 cities of West Germany



Solved TSP instances

1987
Groetschel and Holland
666 interesting places in the world



Solved TSP instances

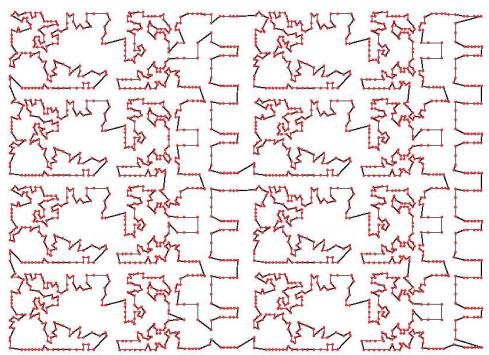
April 18, 2018 5 / 33
Z. Hanzálek (CTU) Travelling Salesman Problem

April 18, 2018 7 / 33
Z. Hanzálek (CTU) Travelling Salesman Problem

Solved TSP instances

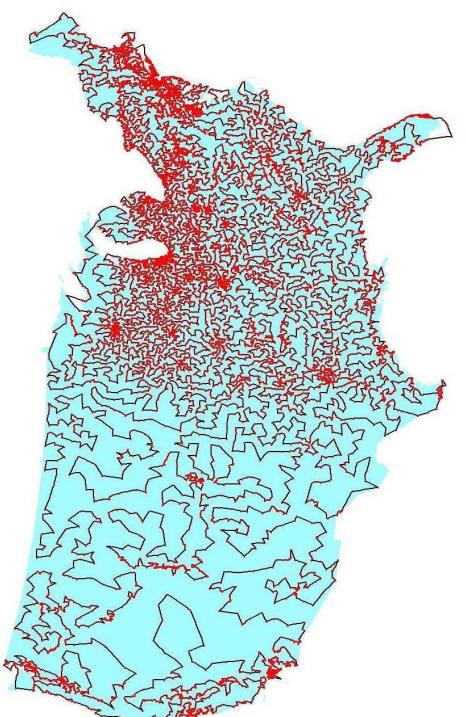
Solved TSP instances

1987
Padberg and Rinaldi
a layout of 2 392 points obtained from Tektronics Incorporated

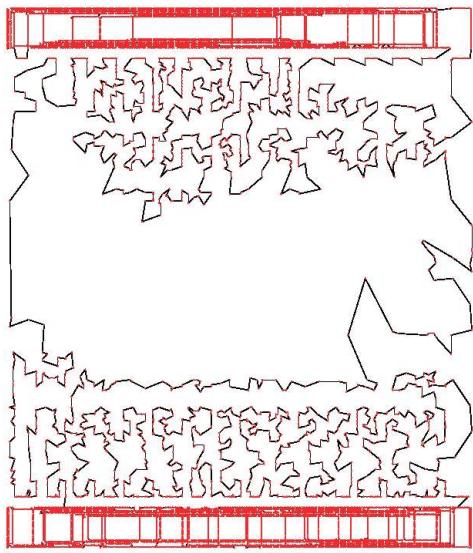


Solved TSP instances

1998
D.Applegate, R.Bixby, V.Chvatal, W.Cook
13 509 cities in the USA with populations greater than 500



1994
D.Applegate, R.Bixby, V.Chvatal, W.Cook
7 397 points in a programmable logic array application at AT&T Bell Laboratories



Solved TSP instances

2001
D.Applegate, R.Bixby, V.Chvatal, W.Cook
15 112 cities in Germany



April 18, 2018 11 / 33
Z. Hanzálek (CTU) Travelling Salesman Problem

April 18, 2018 10 / 33
Z. Hanzálek (CTU) Travelling Salesman Problem

April 18, 2018 11 / 33
Z. Hanzálek (CTU) Travelling Salesman Problem

Solved TSP instances

Related Decision Problem

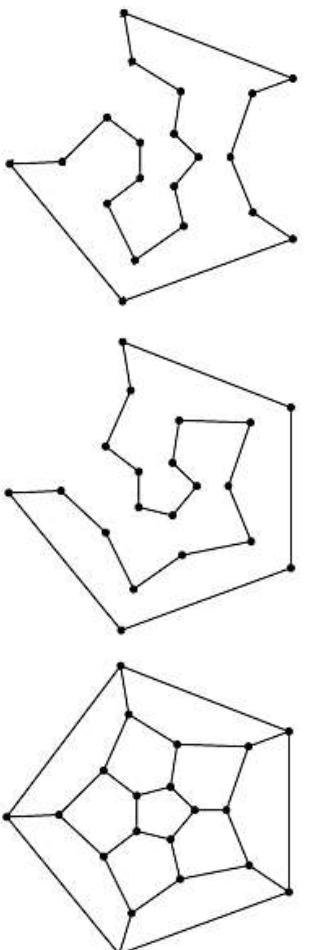
2004

D.Applegate, R.Bixby, V.Chvatal, W.Cook
24 978 cities in Sweden



Hamilton's Puzzle

Z. Hanzálek (CTU) April 18, 2018 12 / 33



Existence of Hamiltonian Circuit (HC, Hamiltonovská kružnice)

- **Instance:** Undirected graph G .
- **Goal:** Decide if Hamiltonian circuit (circuit visiting every node exactly once) exists in graph G .

NP-complete problem

- The directed version of this problem is: (**Hamiltonian cycle**) for a directed graph
- HC belongs to NP problems. For each yes-instance G we take any Hamiltonian circuit of G as a certificate. To check whether a given edge set is in fact a Hamiltonian circuit of a given graph is obviously possible in polynomial time.

Problem Formulation

Z. Hanzálek (CTU) April 18, 2018 13 / 33

Traveling Salesman Problem - TSP

- **Instance:** A complete undirected graph K_n ($n \geq 3$) and weights $c : E(K_n) \rightarrow \mathbb{R}^+$.
- **Goal:** Find a Hamiltonian circuit T whose weight $\sum_{e \in E(T)} c(e)$ is minimum.

- Nodes correspond to cities and weights to distances or travel costs
- This problem is called **symmetric TSP**, since it is given by a complete undirected graph
 - If the distance from city A to city B differs from the one from B to A, we have to use a directed graph and we deal with an **asymmetric TSP**

The Hamiltonian circuit is named after William Rowan Hamilton who invented the Icosian game, now also known as Hamilton's puzzle, which involves finding a Hamiltonian circuit in the edge graph of the dodecahedron. (picture can be viewed as a look inside the dodecahedron through one of its **twelve faces**). Like all platonic solids, the dodecahedron is Hamiltonian.

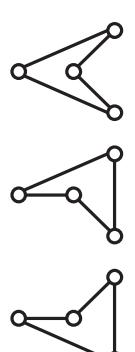
Z. Hanzálek (CTU) April 18, 2018 14 / 33

Z. Hanzálek (CTU) Traveling Salesman Problem

April 18, 2018 15 / 33

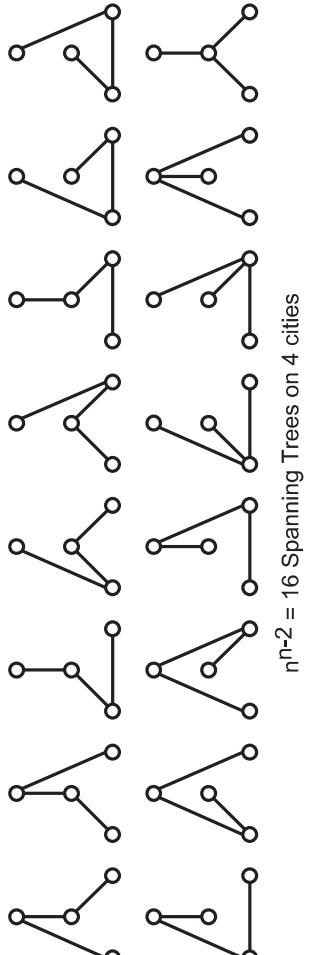
Z. Hanzálek (CTU) Traveling Salesman Problem

Number of Circuits Is Not Exclusive Cause of TSP Complexity



What makes the TSP so hard?
This question stimulates the study of NP-completeness.

$$1/2(n-1)! = 3 \text{ Hamiltonian Circuits through 4 cities}$$



$n^{n-2} = 16$ Spanning Trees on 4 cities

TSP Complexity and Likely Nonexistence of Pseudopolynomial Algorithm

Proposition

TSP is strongly NP-hard.

Proof: We show that the **TSP** is NP-hard even when restricted to instances where all distances are 1 or 2 using polynomial transformation from the **HC** problem

- Let G be an undirected graph in which we want to find the Hamiltonian circuit.
- Create a **TSP** instance such that every node from G is associated to one node in the complete undirected graph K_n . Weight of $\{i,j\}$ in K_n equals:

$$c(\{i,j\}) = \begin{cases} 1 & \text{if } \{i,j\} \in E(G); \\ 2 & \text{if } \{i,j\} \notin E(G). \end{cases}$$

- G has a Hamiltonian circuit iff optimal **TSP** solution is equal to n .

Strongly NP-hard Problems

Let L be an optimization problem.

For a polynomial p let L_p be the **restriction** of L to such instances / that consist of nonnegative integers with $\text{largest}(I) \leq p(\text{size}(I))$, i.e. numerical **parameters** of L_p are **bounded** by a polynomial in the size of the input.

L is called **strongly NP-hard** if there is a polynomial p such that L_p is NP-hard.

If L is strongly NP-hard, then L **cannot be solved by a pseudopolynomial time algorithm unless $P = NP$.**

In the following we will study the case, where:

- $L \dots \text{TSP}, L_p \dots \text{TSP with restriction } c(e) \in \{1, 2\}$,
i.e. $\text{largest}(I) = 2 \leq n = p(\text{size}(I))$

On the other hand: $L \dots \text{Knapsack}, L_p \dots \text{Knapsack with bounded integer costs}$ for which the Dynamic prog. table has polynomial number $(n * p(\text{size}(I)))$ of columns, i.e. **Knapsack** is not strongly NP-hard.

April 18, 2018

Z. Hanzálek (CTU) Traveling Salesman Problem

Likely Nonexistence of Polynomial r-approximation Algorithm for General TSP

Theorem

If we believe $P \neq NP$, than there is no polynomial r -approximation algorithm for **TSP** for $r \geq 1$.

Proof by contradiction:

Assume there exists a polynomial r -approximation algorithm \mathcal{A} for **TSP**. We further show that we can solve the **HC** problem while using such an “inaccurate” algorithm \mathcal{A} . Since **HC** is NP-complete, $P = NP$.

In other words: if there exists a polynomial r -approximation algorithm \mathcal{A} solving **TSP**, then the NP-complete **HC** problem can be solved in polynomial time by \mathcal{A} .

Likely Nonexistence of Polynomial r-approximation Algorithm for General TSP

Every **HC** instance can be polynomially reduced to a TSP instance “inaccurately” solved by r-approximation algorithm \mathcal{A} :

- Let G be an undirected graph in which we want to find the Hamiltonian circuit.

Create a **TSP** instance such that every node from G is associated to one node (city) in the complete undirected graph K_n . Weight (distance) of $\{i,j\}$ in K_n equals:

$$c(\{i,j\}) = \begin{cases} 1 & \text{if } \{i,j\} \in E(G); \\ 2 + (r-1)*n & \text{if } \{i,j\} \notin E(G). \end{cases}$$

- We use \mathcal{A} to solve the instance.

- if the result is in interval $\langle n, r*n \rangle$, then the Hamiltonian circuit exists,
- otherwise the result is greater or equal to $(n-1) + 2 + (r-1)*n = r*n+1$ and G has no Hamiltonian circuit.

Z. Hanzálek (CTU) April 18, 2018 20 / 33

Metric TSP and Triangle Inequality

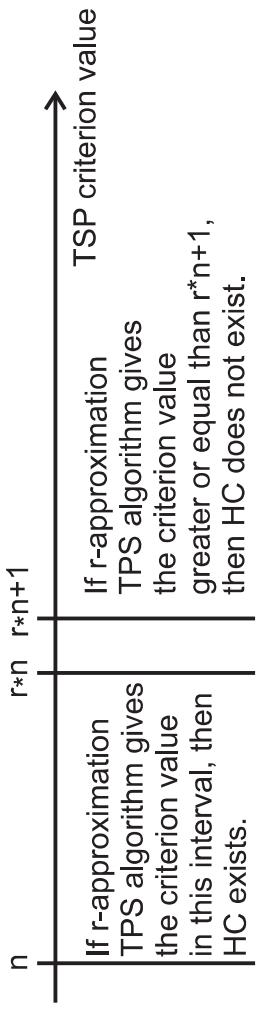
In most common applications the distances of the **TSP** satisfies the triangle inequality.

Metric TSP

- Instance:** Complete undirected graph K_n ($n \geq 3$) with weights $c : E(K_n) \rightarrow \mathbb{R}^+$ such that $c(\{i,j\}) + c(\{j,k\}) \geq c(\{k,i\})$ for all $i, j, k \in V(K_n)$.
- Goal:** Find the Hamiltonian circuit \mathcal{T} such that $\sum_{e \in E(\mathcal{T})} c(e)$ is minimal.

- The **metric TSP** is strongly NP-hard. Can be proved in the same way as the complexity of **TSP** because weights 1 and 2 preserve the triangle inequality. Therefore the pseudopolynomial algs do not exist.
- But approximation algorithms do exist.
- It is not difficult to make the TSP instance metric simply by adding the same constant h to the cost of every edge (the criterion function is higher by $n * h$).

Likely Nonexistence of Polynomial r-approximation Algorithm for General TSP - Illustration



Z. Hanzálek (CTU) April 18, 2018 21 / 33

Nearest Neighbor - Heuristic Algorithm

Input: An instance (K_n, c) of **metric TSP**.

Output: Hamiltonian circuit H .

```
Choose arbitrary node  $v_1] \in V(K_n)$  ;
for  $i := 2$  to do
| choose  $v_i] \in V(K_n) \setminus \{v_1], \dots, v_{i-1}\}$  such that  $c(\{v_{i-1}, v_i]\})$  is
| minimal;
end
Hamiltonian circuit  $H$  is defined by the sequence  $\{v_1], \dots, v_n], v_1]\}$  ;
```

- The nearest unvisited city is chosen in each step
- This is not an approximation algorithm
- Time complexity is $O(n^2)$

Z. Hanzálek (CTU) April 18, 2018 22 / 33

Z. Hanzálek (CTU) April 18, 2018 23 / 33

Input: An instance (K_n, c) of the **metric TSP**.

Output: Hamiltonian circuit H .

- ➊ Find a **minimum weight spanning tree** T in K_n ;
- ➋ By **doubling every edge** in T we get multigraph in which we find the **Eulerian walk** L ;
- ➌ **Transform the Eulerian walk L to the Hamiltonian circuit H in the complete graph K_n :**
 - create a sequence of nodes on the Eulerian walk L ;
 - we **skip nodes that are already in the sequence**;
 - the rest creates the Hamiltonian circuit H ;

Christofides' Algorithm [1976]

Input: An instance (K_n, c) of **metric TSP**.

Output: Hamiltonian circuit H .

- ➊ Find a minimum weight spanning tree T in K_n ;
- ➋ Let W be the set of vertices having an **odd degree** in T ;
- ➌ Find a **minimum weight matching** M of nodes from W in K_n ;
- ➍ Merge of T and M forms a multigraph $(V(K_n), E(T) \cup M)$ in which we find the Eulerian walk L ;
- ➎ Transform the Eulerian walk L into the Hamiltonian circuit H in the complete graph K_n ;

Observation: Each edge connects 2 nodes \Rightarrow the sum of the degree of all nodes is $2|E| \Rightarrow$ there are an even number of nodes with an odd degree in every graph (and an arbitrary number of nodes with an even degree). With respect to the previous observation and completeness of K_n , it follows that it is possible to find the perfect matching.

Time complexity is $O(n^2)$

It is a 2-approximation algorithm for the **metric TSP**:

- ➊ due to the triangle inequality, the skipped nodes don't prolong the route, i.e. $c(E(L)) \geq c(E(H))$
- ➋ while deleting one edge in the circuit, we create the tree. Therefore, inequality $OPT(K_n, c) \geq c(E(T))$ holds
- ➌ $2c(E(T)) = c(E(L))$ holds due to the creation of L by doubling edges in T
- ➍ above points imply $2OPT(K_n, c) \geq c(E(H))$ since:

$$2OPT(K_n, c) \stackrel{2.}{\geq} 2c(E(T)) \stackrel{3.}{\geq} c(E(L)) \stackrel{1.}{\geq} c(E(H))$$

Z. Hanzálek (CTU) April 18, 2018 24 / 33

Travelling Salesman Problem

April 18, 2018 25 / 33

Christofides' Algorithm is a $\frac{3}{2}$ Approximation

Time complexity is $O(n^3)$

It is a $\frac{3}{2}$ approximation algorithm for the **metric TSP**:

- ➊ due to the triangle inequality the skipped nodes do not prolong the route, i.e. $c(E(L)) \geq c(E(H))$
- ➋ while deleting one edge in the circuit, we create the tree. Therefore, inequality $OPT(K_n, c) \geq c(E(T))$ holds
- ➌ since the perfect matching M considers every second edge in the alternating path and being the minimum weight matching it chooses the smaller half, $\frac{OPT(K_n, c)}{2} \geq c(E(M))$ holds
- ➍ due to the construction of L it holds

$$c(E(M)) + c(E(T)) = c(E(L))$$
- ➎ finally we obtain:

$$\frac{3}{2}OPT(K_n, c) \stackrel{2,3.}{\geq} c(E(T)) + c(E(M)) \stackrel{4.}{=} c(E(L)) \stackrel{1.}{\geq} c(E(H))$$

Z. Hanzálek (CTU) April 18, 2018 26 / 33

Travelling Salesman Problem

April 18, 2018 27 / 33

Z. Hanzálek (CTU) Travelling Salesman Problem

Tour Improvement Heuristics - Local Search k -OPT

k -OPT algorithm for TSP

One of the most successful techniques for **TSP** instances in practice.
A simple idea which can be used to solve other optimization problems as well:

- Find any Hamiltonian circuit by some heuristic
- Improve it by “local modifications” (for example: delete 2 edges and reconstruct the circuit by some other edges).

- Local search is an algorithmic principle based on two decisions:
- Which modifications are allowed
 - When to modify the solution (one possibility is, to allow improvements only)

Example of local search is k -OPT algorithm for TSP

Input: An instance (K_n, c) of **TSP**, number $k \geq 2$.

Output: Hamiltonian circuit H .

1. Let H be any Hamiltonian circuit;
2. Let \mathcal{S} be the family of k -element subsets S of $E(H)$;

```

for all  $S \in \mathcal{S}$  do                                // outer loop deals with removed edges
    for all Ham.circuits  $H' \neq H$  such that  $E(H') \supseteq E(H) \setminus S$  do
        if  $c(E(H')) < c(E(H))$  then  $H := H'$  to 2.
    end
end

```

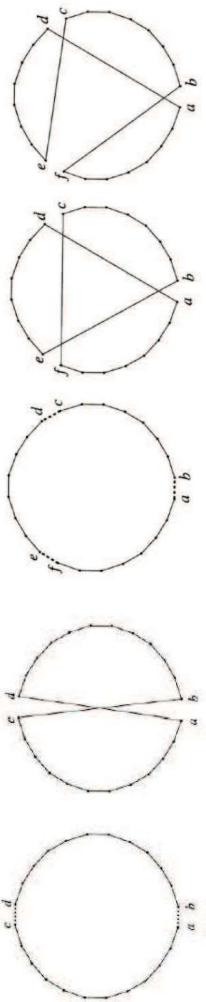
Note:

- H' is constructed, so that it is Hamiltonian circuit as well.
- **When $k=2$** , the inner loop, which creates the Hamiltonian circuits H' from the remaining edges of H , executes only once, since there is just **one way to construct the new Hamiltonian circuit H'** .

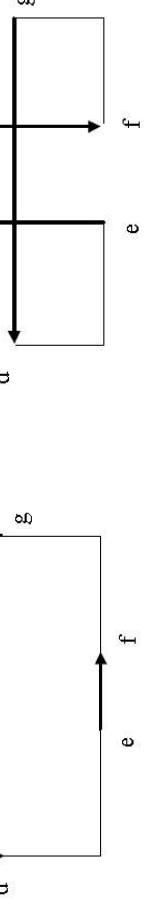
Examples of 2-opt and 3-opt for TSP

Z. Hanzálek (CTU) April 18, 2018 28 / 33

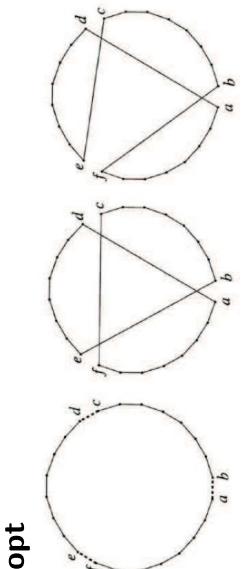
2-opt



just one way to construct the new Hamiltonian circuit:



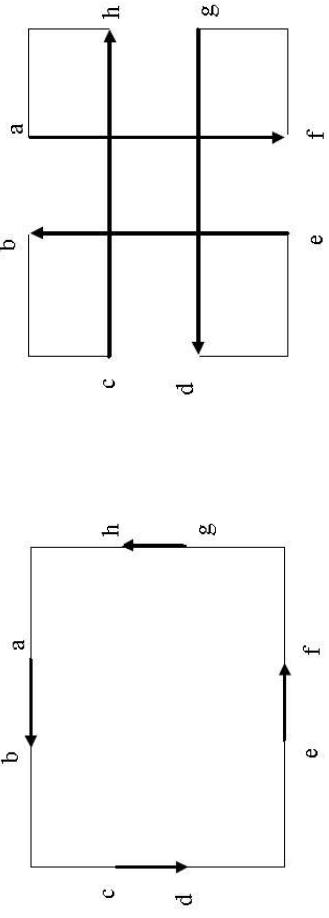
- the gain if the improvement is:



at least two ways to construct the new Hamiltonian circuit:

- $c(E(H')) - c(E(H)) = (a,d)+(e,b)+(c,f)-(a,b)-(c,d)-(e,f)$
no path has changed orientation
- $c(E(H'')) - c(E(H)) = (a,d)+(e,c)+(b,f)-(a,b)-(c,d)-(e,f)$
path (c,\dots,b) has changed orientation

One possible solution called the “double bridge” – no path has changed orientation:



- One of the “most popular” NP-hard problems
- 49 – 120 – 550 – 2,392 – 7,397 – 19,509 – 24,978 cities from year 1954 to year 2004
- Lot of constraints must be added when solving real life problems
 - CVRP – Capacitated Vehicle routing Problem – limited number of cars and limited load capacity of cars, every customer buys a different amount of the product
 - VRPTW – Time Windows – customers define time windows in which they accept cargo
 - VRPPD – Pick-up and Delivery – customers return some amount of the product (or wrapping) that takes place in the car

Table of Contents

- Basics notions
- Scheduling on One Resource
 - Minimizing C_{max}
 - Bratley’s Algorithm for 1 | r_j, \tilde{d}_j | C_{max}
 - Minimizing $\sum w_j C_j$
 - Branch&Bound with LP for 1 | prec | $\sum w_j C_j$
 - Minimizing L_{max}
 - Horn’s Algorithm for 1 | pmtn, $r_j, d_j = \tilde{d}_j$ | L_{max}
 - Scheduling on Parallel Identical Resources
 - Minimizing C_{max}
 - Temporal constraints
 - Minimizing C_{max}
 - ILP Model for PS1 | temp | C_{max} – One Resource
 - ILP Model for PSm, 1 | temp | C_{max} – m Dedicated Resources of Unit Capacity
 - Modeling with Temporal Constraints
 - ILP Model for PSm, R | temp | C_{max} – m Dedicated Resources of Capacities R_k
 - Project Scheduling

Scheduling

Zdenek Hanzálek
zdenek.hanzalek@cvut.cz

CTU in Prague

May 15, 2018

Motivation Example: Lacquer Production Scheduling



Made-to-order lacquer production, where jobs are determined by type of lacquer, quantity and delivery date.

- Goal:
- minimize tardiness (delivery date overrun)
 - minimize storage costs

Constraints:

- batch production of various kinds of lacquer
- varying production process/time for different kinds
- time constraints between start times and/or completion times of operations
- working hours (processing times of some operations exceed working hours)
- preparation (set-up time)

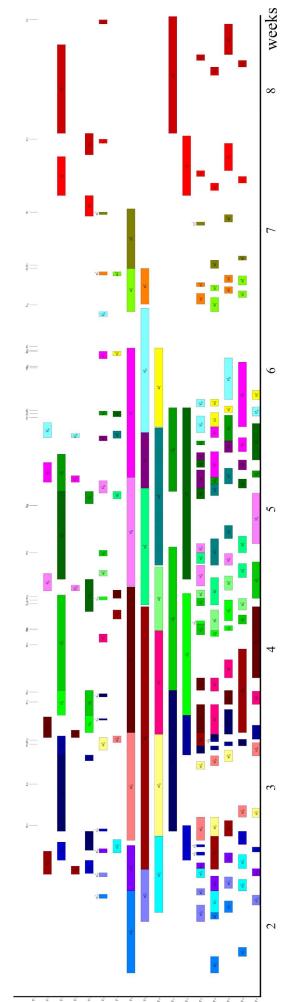
Motivation Example: Lacquer Production Scheduling

May 15, 2018 3 / 83

Scheduling - Basic Terminology

Example

- production of 29 jobs
- 3 types of lacquer
- 9 weeks time horizon



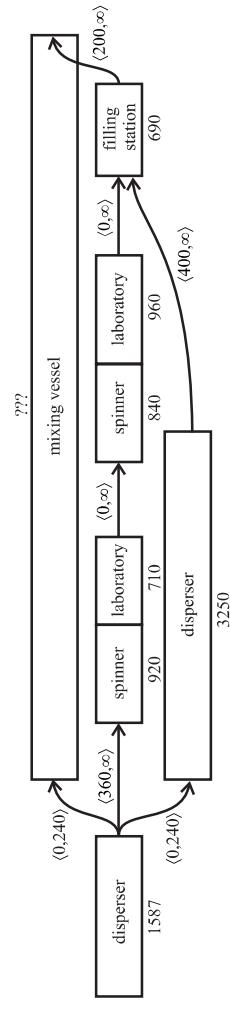
Motivation Example: Lacquer Production Scheduling - Formalization

Formalization

Can be formalized as $PSm, 1 \mid temp, o_{ij}, tg \mid \sum w_j \cdot T_j$

There are temporal constraints on operations. We must consider:

- (1) minimal delay between the end of one operation and the start of the next one (e.g. minimal delay needed to dissolve an ingredient)
- (2) maximal delay between the end of one operation and the start of the next one (e.g. the lacquer can solidify).



- Moreover, the processing time on some resource (e.g. reservoir) is given by the start and completion of different operations.

May 15, 2018 4 / 83

Scheduling - Basic Terminology

- set of n tasks $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$

set of m types of resources (processors, machines, employees,...) with capacities R_k , $\mathcal{P} = \{P_1^1, P_1^2, \dots, P_{R_1}^1, P_{R_1}^2, \dots, P_1^m, \dots, P_{R_m}^m\}$

- Scheduling is an assignment of a task to a resources in time**

- Each task must be **completed**
 - this differs from planning which task will be scheduled and processed

- Set of tasks is known when executing the scheduling algorithm (this is called **off-line scheduling**)
 - this differs from on-line scheduling - OS scheduler, for example, schedules new tasks using some policy (e.g. priority levels)

- A result is a schedule which determines which task is run on which resource and when. Often depicted as a **Gantt chart**.

May 15, 2018 5 / 83

May 15, 2018 6 / 83

Scheduling

Z. Hanzalek (CTU)

General and Specific Constraints

Task Parameters and Variables

General constraints:

- Each task is to be processed by **at most one resource** at a time (task is sequential)
- Each resource is capable of processing **at most one task** at a time
- Task T_i has to be processed during time interval $\langle r_i, d_i \rangle$
- When the precedence constraint is defined between T_i and T_j , i.e. $T_i \prec T_j$, then the processing of task T_j can't start before task T_i was completed
- If scheduling is non-preemptive, a task cannot be stopped and completed later
- If scheduling is preemptive, the number of preemptions must be finite

Graham's Notation $\alpha | \beta | \gamma$

Classify scheduling problems by
resources | **tasks** | **criterion**

Example: $P2|pmtn|C_{max}$ represents scheduling on two parallel identical resources, and preemption is allowed. The optimization criterion is the completion time of the last task.

α - resources

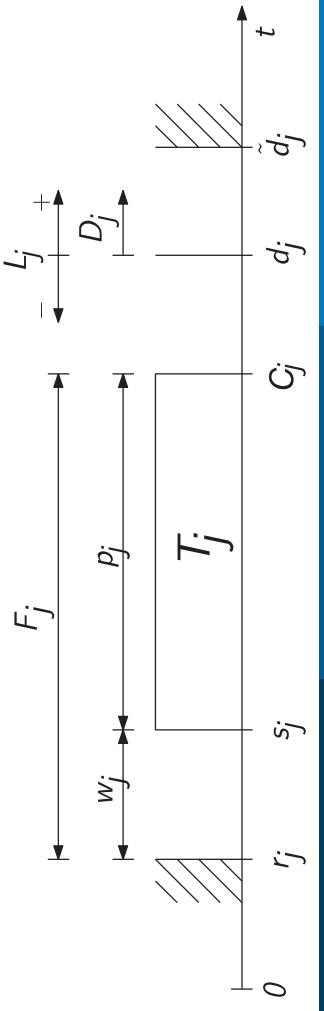
- **Parallel resources** - a task can run on any resource (only one type of resource exists with capacity R , i.e. $\mathcal{P} = \{P^1, \dots, P^R\}$).
- **Dedicated resources** - a task can run only on one resource (m resource types with unit capacity, i.e. $\mathcal{P} = \{P^1, P^2, \dots, P^m\}$).
- **Project Scheduling** - m resource types, each with capacity R_k , i.e. $\mathcal{P} = \{P_1^1, \dots, P_{R_1}^1, P_1^2, \dots, P_{R_2}^2, \dots, P_1^m, \dots, P_{R_m}^m\}$.

Parameters

- **release time** r_j
- **processing time** p_j
- **due date** d_j , time in which task T_j should be completed
- **deadline** \tilde{d}_j , time in which task T_j has to be completed

Variables

- **start time** s_j
- **completion time** C_j
- **waiting time** $w_j = s_j - r_j$
- **flow (lead) time** $F_j = C_j - r_j$
- **lateness** $L_j = C_j - d_j$
- **tardiness** $D_j = \max\{C_j - d_j, 0\}$



Resources Characteristics α_1, α_2

$$\alpha_1 = \frac{1}{P} \text{ single resource}$$

- **parallel identical resources**
- **parallel uniform resources**, computation time is **inversely related to resource speed**

- **parallel unrelated resources**, computation times are given as a matrix (resources \times tasks)
- **dedicated resources** - **open-shop** - tasks are independent in the sequences (jobs) in the same order, each job visits each machine once

- **dedicated resources** - **job-shop** - order of tasks in jobs is arbitrary, resource can be used several times in a job
- **flow-shop** - tasks are grouped in the sequences (jobs) in the same order, each job visits each machine once

- **Project Scheduling** - most general (several resource types with capacities, general precedence constraints)

- **arbitrary number of resources**
- **2 resources (or other specified number)**

- **m, R** m resource types with capacities R (Project Scheduling)

Task Characteristics $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8$

Optimality Criterion γ

$\beta_1 = \text{pmtn}$	\emptyset	preemption is allowed
$\beta_2 = \text{prec}$	in-tree,out-tree chain tmpn \emptyset	precedence constraints tree constraints chain constraints temporal constraints (for Project Sched.) independent tasks
$\beta_3 = r_j$	release time	
$\beta_4 = p_j = k$	$p_L \leq p_j \leq p_U$	uniform processing time restricted processing time arbitrary processing time
$\beta_5 = \tilde{d}_j, d_j$	deadline, due-date	
$\beta_6 = n_j \leq k$	maximal number of tasks in a job	
$\beta_7 = \text{no-wait}$	buffers of zero capacity	
$\beta_8 = \text{set-up}$	time for resource reconfiguration	

Scheduling on One Resource
Minimizing Makespan (i.e. schedule length C_{\max})

- $1 \mid \text{prec} \mid C_{\max}$ - easy
- the tasks are processed in an arbitrary order that satisfies the precedence relation (i.e. topological order), $C_{\max} = \sum_{j=1}^n p_j$
- $1 \parallel C_{\max}$ - easy
- the tasks are processed in a non-descending order of r_j (T_j with the lowest r_j first)
- $1 \mid \tilde{d}_j \mid C_{\max}$ - easy
- the tasks are processed in a non-descending order of \tilde{d}_j
- can be solved by EDF - Earliest Deadline First
- the feasible schedule doesn't have to exist
- $1 \mid r_j, \tilde{d}_j \mid C_{\max}$ - NP-hard
- NP-hardness proved by the pol. reduction from 3-Partition problem
- for $p_j = 1$ there exists a polynomial algorithm

$\gamma = C_{\max}$	minimize schedule length $C_{\max} = \max \{ C_j \}$
$\sum C_j$	(makespan, i.e. completion time of the last task)
$\sum w_j C_j$	minimize the sum of completion times
L_{\max}	minimize weighted completion time
\emptyset	minimize max. lateness $L_{\max} = \max \{ C_j - d_j \}$
...	decision problem

An Example: $P \parallel C_{\max}$ means:

Scheduling on an arbitrary number of parallel identical resources, no preemption, independent tasks (no precedence), tasks arrive to the system at time 0, processing times are arbitrary, objective is to minimize the schedule length.

$1 \mid r_j, \tilde{d}_j \mid C_{\max}$ Problem is NP-hard
Theorem

The $1 \mid r_j, \tilde{d}_j \mid C_{\max}$ problem is NP-hard in the strong sense.

By reduction from the 3-Partition problem, which is strongly NP-complete.

- a multiset A of $3m$ integers a_1, a_2, \dots, a_{3m} (sizes of **items**), and a positive integer B (size of **bins**) such that $\forall i \in \{1, 2, \dots, 3m\}: \frac{B}{4} < a_i < \frac{B}{2}$ and $\sum_{i=1}^{3m} a_i = mB$.

The problem is to determine whether A can be **partitioned** into m disjoint subsets A_1, A_2, \dots, A_m such that, $\forall j \in \{1, 2, \dots, m\}: \sum_{a_i \in A_j} a_i = B$.

Note: if we show that there is a subset A_j which contains integers summing to B , then it must contain **three integers**. This follows from the assumption $\frac{B}{4} < a_i < \frac{B}{2}$ (try to sum-up 4 integers or 2 integers).

Reduction from 3-Partition to $1 \mid r_j, \tilde{d}_j \mid C_{\max}$

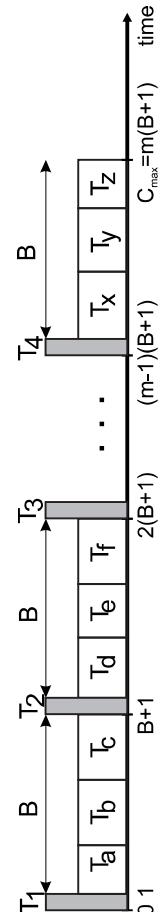
Position based ILP formulation for $1 \mid r_j, \tilde{d}_j \mid C_{\max}$

From the given instance of the 3-Partition problem $I_{3P} = (A, B)$, we build $1 \mid r_j, \tilde{d}_j \mid C_{\max}$ scheduling problem instance I_{SCH} comprised of $4m$ tasks

$T_j = (p_j, r_j, \tilde{d}_j)$ as follows:

- $\forall j \in \{1, \dots, m\} : T_j = (1, (B+1) \cdot (j-1), (B+1) \cdot (j-1)+1)$. These are "additional/artificial" tasks used to separate the subsets.
- $\forall j \in \{m+1, \dots, 4m\} : T_j = (a_i, 0, \infty), i = j - m$.

Each of these tasks T_j corresponds to the element a_i of I_{3P} .

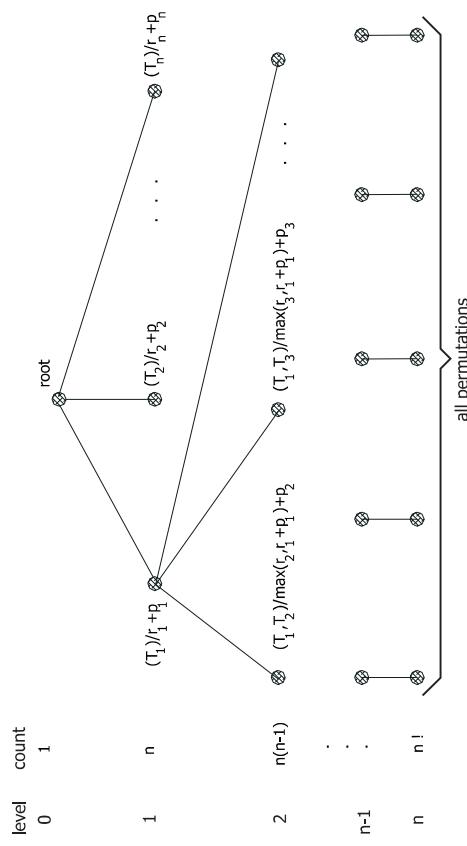


It is easy to prove that the $I_{3P} = (A, B)$ has a solution if and only if the optimal solution of the related I_{SCH} has value of $C_{\max} = m \cdot (B+1)$.

Bratley's Algorithm for $1 \mid r_j, \tilde{d}_j \mid C_{\max}$

A **branch and bound** (B&B) algorithm.

Branching - without bounding it is an **enumerative method** that creates a solution tree (some of the nodes are infeasible). Every node is labeled by: (the order of tasks)/(completion time of the last task).



$x_{iq} = 1$ iff task i is at the q -th position in the sequence of tasks

$$\min C_{\max}$$

subject to:

$$\begin{aligned} \sum_{q=1}^n x_{iq} &= 1 & i = 1..n \\ \sum_{i=1}^n x_{iq} &= 1 & q = 1..n \\ t_q &\geq \sum_{i=1}^n r_i \cdot x_{iq} & q = 1..n \\ t_q &\geq t_{q-1} + \sum_{i=1}^n p_i \cdot x_{i,q-1} & q = 2..n \\ t_q &\leq \sum_{i=1}^n d_i \cdot x_{i,q} - \sum_{i=1}^n p_i \cdot x_{i,q} & q = 1..n \\ C_{\max} &\geq t_n + \sum_{i=1}^n p_i \cdot x_{in} \end{aligned}$$

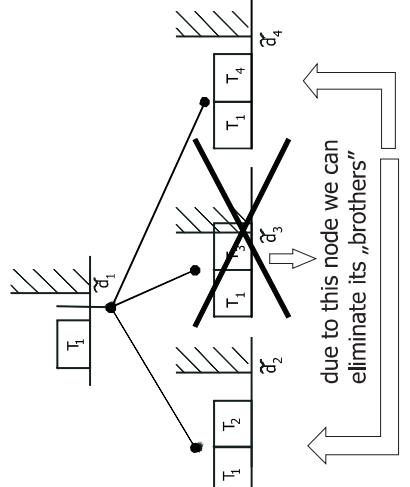
variables: $x_{i1..n}, q \in 1..n \in \{0, 1\}$, $C_{\max} \in \langle 0, UB \rangle$, $t_{q1..n} \in \langle 0, UB \rangle$

Reduction of the Tree - Bounding

May 15, 2018

Scheduling

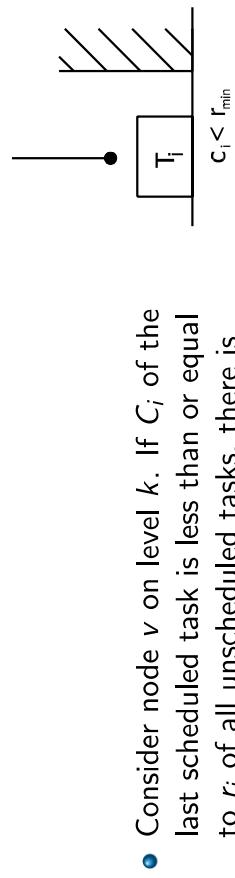
- (i) **eliminate the node** exceeding the deadline (and all its "brothers")



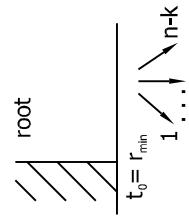
- If there is a node which exceeds any deadline, all its descendants should be eliminated
- Critical task (here T_3) will have to be scheduled anyway - therefore, all of its "brothers" should be eliminated as well

Tree Size Reduction - Decomposition

- (ii) **problem decomposition** due to idle waiting - e.g. when the employee waits for the material, his work was optimal



- Consider node v on level k . If C_i of the last scheduled task is less than or equal to r_i of all unscheduled tasks, there is no need for backtrack above v
- v becomes a new root and there are $n - k$ levels ($n - k$ unscheduled tasks) to be scheduled

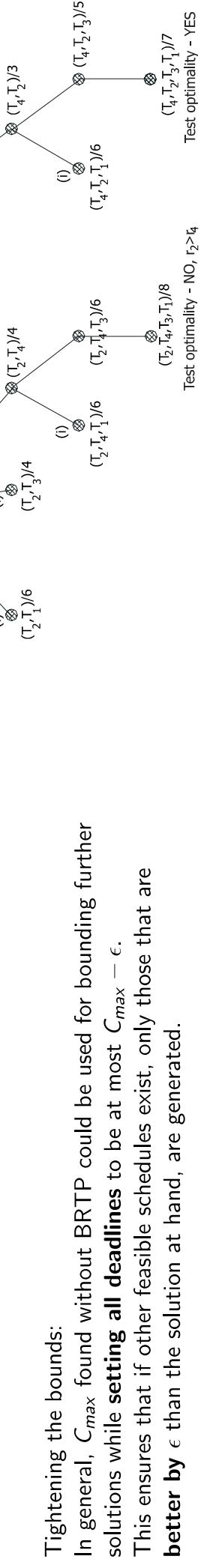


BRTTP is not Necessary Condition of Optimality

Example:



In this particular case, the schedule is optimal, but it does not have BRTTP.



- (ii) **problem decomposition** due to idle waiting - e.g. when the employee waits for the material, his work was optimal

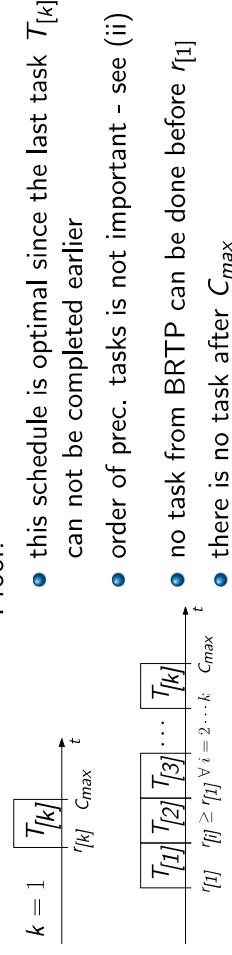
- first task $T_{[1]}$ starts at its release time
- all k tasks till the end of the schedule run without "idle waiting"
- $r_{[1]} \leq r_{[i]}$ for all $i = 2 \dots k$

Note: "till the end of the schedule" implies there is at most one BRTTP

Lemma: sufficient condition of optimality

If BRTTP exists, the schedule is optimal (the search is finished).

Proof:



Z. Hanzalek (CTU) Scheduling May 15, 2018 19 / 83

Bratley's Algorithm - Example

$$r = (4, 1, 1, 0), p = (2, 1, 2, 2), \tilde{d} = (8, 5, 6, 4)$$

Example:



Tightening the bounds:
In general, C_{max} found without BRTTP could be used for bounding further solutions while **setting all deadlines** to be at most $C_{max} - \epsilon$.
This ensures that if other feasible schedules exist, only those that are **better by ϵ** than the solution at hand, are generated.

Z. Hanzalek (CTU) Scheduling May 15, 2018 21 / 83

May 15, 2018 22 / 83

Z. Hanzalek (CTU) Scheduling May 15, 2018 20 / 83

Scheduling on One Resource Minimizing $\sum w_j C_j$

- $1 \mid \sum C_j$ - easy
 - SPT rule (Shortest Processing Time first) - schedule the tasks in a non-decreasing order of p_j
- $1 \mid \sum w_j C_j$ - easy
 - Weighted SPT - schedule the tasks in a non-decreasing order of $\frac{p_j}{w_j}$
- $1 \mid r_j \mid \sum C_j$ - NP-hard
 - $1 \mid \text{pmtn}, r_j \mid \sum C_j$ - can be solved by modified SPT
 - $1 \mid \text{pmtn}, r_j \mid \sum w_j C_j$ - NP-hard
- $1 \mid \tilde{d}_j \mid \sum C_j$ - can be solved by modified SPT
- $1 \mid \tilde{d}_j \mid \sum w_j C_j$ - NP-hard
- $1 \mid \text{prec} \mid \sum C_j$ - NP-hard

ILP formulation for $1 \mid \text{prec} \mid \sum w_j C_j$

Z. Hanzalek (CTU) Scheduling May 15, 2018 23 / 83

Branch and Bound with LP Bounding

Z. Hanzalek (CTU) Scheduling May 15, 2018 24 / 83

First, we formulate the problem as an ILP:

- we use **variable** $x_{ij} \in \{0, 1\}$ such that $x_{ij} = 1$ iff T_i precedes T_j or $i = j$
- we encode **precedence relations** into $e_{ij} \in \{0, 1\}$ such that $e_{ij} = 1$ iff there is a directed edge from T_i to T_j in the precedence graph G or $i = j$
- criterion** - completion time of task T_j consists of p_j and the processing time of its predecessors:

$$\begin{aligned} C_j &= \sum_{i=1}^n p_i \cdot x_{ij} \\ w_j \cdot C_j &= \sum_{i=1}^n p_i \cdot x_{ij} \cdot w_j \\ J &= \sum_{j=1}^n w_j \cdot C_j = \sum_{j=1}^n \sum_{i=1}^n p_i \cdot x_{ij} \cdot w_j \end{aligned}$$

from all feasible schedules x we look for the one that minimizes $J(x)$, i.e. $\min_x J(x)$

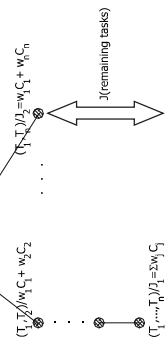
We relax on the integrality of variable x :

- $0 \leq x_{ij} \leq 1$ and $x_{i \in 1..n, j \in 1..n} \in \mathbb{R}$
- This does not give us the right solution, however we can use the J^{LP} (remaining tasks) value of this LP formulation as a lower bound on the "amount of remaining work"

The Branch and Bound algorithm creates a similar tree as Bradley's algorithm.

- Let J_1 be the value of the best solution known up to now
- We discard the partial solution of value J_2 not only when $J_2 \geq J_1$, but also when $J_2 + J^{LP}$ (remaining tasks) $\geq J_1$.
- Since the solution space of ILP is a subspace of LP we know:
 $J(\text{remaining tasks}) \geq J^{LP}$ (remaining tasks).

parameters: $w_{i \in 1..n}, p_{i \in 1..n} \in \mathbb{R}_0^+$ $e_{i \in 1..n, j \in 1..n} \in \{0, 1\}$
variables: $x_{i \in 1..n, j \in 1..n} \in \{0, 1\}$



min $\sum_{j=1}^n \sum_{i=1}^n p_i \cdot x_{ij} \cdot w_j$
subject to:
 $x_{ij} \geq e_{ij} \quad i, j \in 1..n$

if T_i precedes T_j in G ,
then it precedes T_j
in the schedule
either T_i precedes T_j ,
or vice versa
no cycle exists in the
digraph of x

$1 \leq x_{ij} + x_{jk} + x_{ki} \leq 2 \quad i, j, k \in 1..n,$
 $i \neq j \neq k$
 $x_{i,j} = 1 \quad i \in 1..n$

parameters: $w_{i \in 1..n}, p_{i \in 1..n} \in \mathbb{R}_0^+$ $e_{i \in 1..n, j \in 1..n} \in \{0, 1\}$
variables: $x_{i \in 1..n, j \in 1..n} \in \{0, 1\}$

Z. Hanzalek (CTU) Scheduling May 15, 2018 25 / 83

Z. Hanzalek (CTU) Scheduling May 15, 2018 26 / 83

Scheduling on One Resource Minimizing L_{max}

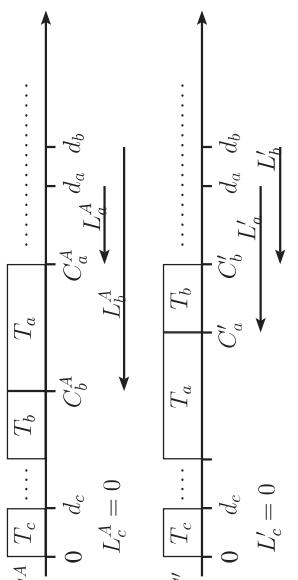
- $1 \parallel L_{max}$ - solved by EDD (Earliest Due Date first) rule in polynomial time
- $1 \mid r_j \mid L_{max}$ - NP-hard
- $1 \mid r_j, p_j = 1 \mid L_{max}$ - polynomial - iterating EDD
- $1 \mid pmtn, r_j \mid L_{max}$ - polynomial - iterating EDD by Horn
- $1 \mid pmtn, r_j, d_j = \tilde{d}_j \mid L_{max}$ - polynomial - the same Horn's algorithm often called EDF
- $1 \mid pmtn, prec, r_j, d_j = \tilde{d}_j \mid L_{max}$ - polynomial - transformation to independent task set and then EDF

Important notes:

- minimization of L_{max} implies existence of due-date d_j (even if it is not in β notation)
- L_{max} may have negative value. In such case, minimization of lateness
- L_{max} may be interpreted as maximization of earliness.

Z. Hanzalek (CTU) Scheduling May 15, 2018 27 / 83

Problem 1 || L_{max} - Optimality of EDD - Illustration



Two cases must be considered when $p_a > 0$ and $p_b > 0$:

- If $L'_a \geq L'_b$ then $L'_max(\{T_a, T_b\}) = C'_a - d_a < C_a^A - d_a$
- If $L'_a \leq L'_b$ then $L'_max(\{T_a, T_b\}) = C'_b - d_b = C_b^A - d_b < C_a^A - d_a$

Since, in both cases, $L'_max(\{T_a, T_b\}) < L_max(\{T_a, T_b\})$ we can conclude that the swap of T_a and T_b decreases $L_{max}(\{T_a, T_b\})$ and thus it cannot increase $L_{max}(T)$, the maximum lateness of all tasks.

- Can be solved by EDD (Earliest Due Date first), i.e. tasks are **scheduled in order of nondecreasing due dates**.

- Time complexity is $O(n \cdot \log n)$.

Optimality can be proven by simple swaps:

- Let S^A be an **optimal schedule** produced by an algorithm A.
- Let S^{EDD} be a schedule produced by EDD.
- If $S^A \neq S^{EDD}$, then there exist two tasks T_a and T_b with $d_a \leq d_b$, such that T_b immediately precedes T_a in S^A .
- Swap of T_a and T_b cannot increase the maximum lateness of the task set L_{max} .
- By finite number of swaps, S^A is changed to S^{EDD} ,

- $S^A \xrightarrow{\text{swap}} S' \xrightarrow{\text{swap}} \dots \xrightarrow{\text{swap}} S^{EDD}$

Z. Hanzalek (CTU) Scheduling May 15, 2018 28 / 83

Problem 1 | $r_j, p_j = 1 \mid L_{max}$

Optimal schedule can be found by **iterative calls of EDD**:

- at every moment we schedule the task which is ready and has the lowest d_j among all ready tasks.

Input: \mathcal{T} , set of n non-preemptive tasks with unit processing time.
Output: Start times (s_1, s_2, \dots, s_n) .

```

t := 0;
while  $\mathcal{T} \neq \emptyset$  do
    t := max{ $t, \min_{T_j \in \mathcal{T}} r_j$ }; // shift time if no task is ready
     $\mathcal{T}' = \{T_j | T_j \in \mathcal{T}, r_j \leq t\}$ ; // set of ready tasks
    choose  $T_j \in \mathcal{T}'$  with minimal  $d_j$ ;
     $\mathcal{T} := \mathcal{T} \setminus \{T_j\}$ ;
     $s_j = t$ ;
    t := t + 1;
end

```

Example: $r = (2, 0, 2, 3, 3)$, $d = (3, 5, 4, 4, 6)$ results in $L_{max} = 1$.

Z. Hanzalek (CTU) Scheduling May 15, 2018 29 / 83

May 15, 2018 30 / 83

Problem 1 |pmtn, r_j | L_{max} - Horn's Algorithm

Problem 1 |pmtn, r_j | L_{max}

Input: \mathcal{T} , set of n preemptive tasks. Processing times (p_1, p_2, \dots, p_n) , release dates (r_1, r_2, \dots, r_n) and due-dates (d_1, d_2, \dots, d_n) .

Output: Start times of preempted parts of tasks.

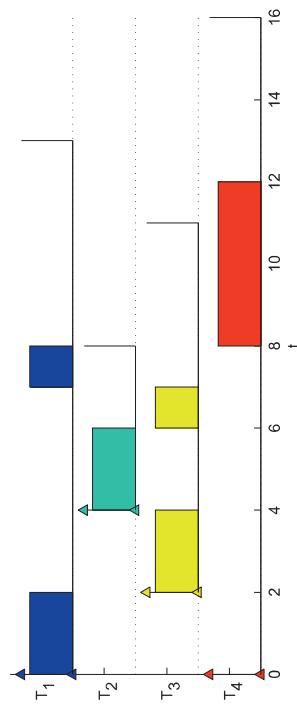
```

while  $\mathcal{T} \neq \emptyset$  do
     $t_1 := \min_{T_j \in \mathcal{T}} \{r_j\}$ ;
    if all tasks are ready at time  $t_1$  then  $t_2 = \infty$ ;
    else  $t_2 = \min_{T_j \in \mathcal{T}} \{r_j | r_j > t_1\}$ ;
     $\mathcal{T}' = \{T_j | T_j \in \mathcal{T}, r_j = t_1\}$ ; // set of ready tasks
    choose  $T_k \in \mathcal{T}'$  with minimal  $d_j$ ;
     $\delta := \min \{p_k, t_2 - t_1\}$ ;
    schedule  $T_k$  or its part in interval  $\langle t_1, t_1 + \delta \rangle$ ;
    if  $\delta = p_k$  then  $\mathcal{T} := \mathcal{T} \setminus \{T_k\}$ ;
    else  $p_k := p_k - \delta$ ;
        for  $T_j \in \mathcal{T}'$  do  $r_j := t_1 + \delta$ ;
    end

```

Example:

$$\begin{aligned}\mathcal{T} &= \{T_1, T_2, T_3, T_4\} \\ p &= (3, 2, 3, 4) \\ r &= (0, 4, 2, 0) \\ d &= (13, 8, 11, 16)\end{aligned}$$



Optimality of Horn's Algorithm

Given a set of n independent preemptive tasks with arbitrary release times, any algorithm that at any instant executes the task with earliest due date among all the ready tasks is optimal with respect to L_{max} minimization.

Theorem - Optimality of Horn's Algorithm

When we assume $1 | pmtn, r_j, d_j = \tilde{d}_j | L_{max}$ then the algorithm is often called **EDF (Earliest Deadline First)**. Such algorithm:

- **minimizes** L_{max}
- **decides schedulability** – if there exists a feasible schedule ($L_{max} \leq 0$) for the given instance, then the EDF is able to find it

Assuming the input parameters to be nonnegative **integers**, the proof of the theorem is based on the following reasoning:

- Let S^A be an **optimal schedule** produced by algorithm A
- Let S^{EDF} be a schedule produced by EDF
- Let schedule S^A starts at time $t = 0$ and D is the latest due date.
- Without loss of generality S^A can be **divided into unit-time slices**.
- Let i_t is the id of the task executing slice t .
- Let j_t is the id of the ready task with earliest due date at time t .
- If $S^A \neq S^{EDF}$ then there is slice t such that $i_t \neq j_t$.
- Swap of slices of T_{i_t} and T_{j_t} cannot increase the maximum lateness of the task set L_{max} .
- S^{EDF} is obtained from S^A by at most D swaps.

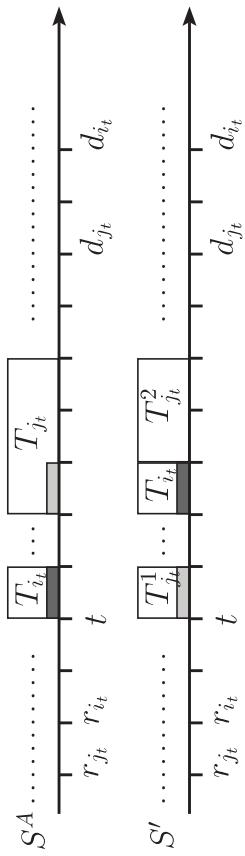
Optimality of Horn's Algorithm

Z. Hanzalek (CTU) May 15, 2018 31 / 83

Optimality of Horn's Algorithm for $1 | pmtn, r_j | L_{max}$ and its application to $1 | pmtn, r_j, d_j = \tilde{d}_j | L_{max}$

Z. Hanzalek (CTU) May 15, 2018 31 / 83

Optimality of Horn's Algorithm - Illustration



Using the same argument adopted in the proof of EDD optimality for $1 \parallel L_{\max}$, it is easy to show that each swap cannot increase the maximum lateness of the task set L_{\max} .

- For any task without predecessors set $\tilde{r}_j = r_j$.
- Select task T_j such that its release date has not been modified but the release dates of all immediate predecessors T_h have been modified. If no such task exists, exit.
- Set $\tilde{r}'_j = \max\{r_j, \max\{r_h + p_h \mid T_h \text{ is immediate predecessor of } T_j\}\}$ and jump to step 2.

modification of deadlines

- For any task without successors set $\tilde{d}_j = \tilde{d}_j$.
- Select a task T_j such that its deadline has not been modified but the deadlines of all immediate successors T_k have been modified. If no such task exists, exit.
- Set $\tilde{d}'_j = \min\{\tilde{d}_j, \min\{\tilde{d}_k \mid T_k \text{ is immediate successor of } T_j\}\}$ and jump to step 2.

- EDF is executed on independent tasks in \mathcal{T}'

Algorithm Optimality

Scheduling on Parallel Identical Resources Minimizing C_{\max}

Idea of the proof in one direction:

- $r'_j \geq r_j$ and $\tilde{d}'_j \leq \tilde{d}_j$, therefore, the schedulability of \mathcal{T}' with respect to the timing constraints (release dates and deadlines) implies also schedulability of \mathcal{T} with respect to the timing constraints due to the logics of the Horn's alg. and modification of the timing constraints the scheduled tasks of \mathcal{T}' are ordered in the same way as given by the precedence constraints, therefore, precedence constraints of \mathcal{T} are not violated

- $P \parallel C_{\max}$ - NP-hard
 - schedule n non-preemptive tasks on two parallel identical resources minimizing makespan, i.e. the completion time of the last task
 - the problem is NP-hard because the **2 partition problem** (see ILP lecture) can be reduced to $P2 \parallel C_{\max}$ while comparing the optimal C_{\max} with the threshold of $0.5 * \sum_{i=1..n} p_i$.
- $P \mid \text{pmtn}, r_j, \tilde{d}_j \mid$ - easy
 - can be solved by the **McNaughton** algorithm in $O(n)$
- $P \mid \text{pmtn}, r_j, \tilde{d}_j \mid$ - easy
 - decision version of **maximum flow problem** (see the lecture on Flows)
- $P \mid \text{prec} \mid C_{\max}$ - NP-hard
 - **LS** - approximation algorithm with factor $r_{LS} = 2 - \frac{1}{R}$, where R is the number of parallel identical resources
- $P \parallel C_{\max}$ - NP-hard
 - **LPT** - approximation algorithm with factor $r_{LPT} = \frac{4}{3} - \frac{1}{3R}$
 - **dynamic programming** - Rothkopf's pseudopolynomial algorithm
- $P \mid \text{pmtn}, \text{prec} \mid C_{\max}$ - NP-hard
 - Muntz&Coffman's **level algorithm** with factor $r_{MC} = 2 - \frac{2}{R}$

McNaughton's Algorithm for $P \mid \text{pmtn} \mid C_{\max}$

McNaughton's Algorithm for $P \mid \text{pmtn} \mid C_{\max}$

Input: R , number of parallel identical resources, n , number of preemptive tasks and computation times (p_1, p_2, \dots, p_n) .

Output: n -element vectors s^1, s^2, z^1, z^2 where s_i^1 (resp. s_i^2) is start time of the first (resp. second) part of task T_i and z_i^1 (resp. z_i^2) is the resource ID on which the first (resp. second) part of task T_i will be executed.

```

 $s_i^1 = s_i^2 = z_i^1 = z_i^2 := 0 \text{ for all } i \in 1 \dots n;$ 
 $t := 0; v := 1; i := 1;$ 
 $C_{\max}^* = \max \{ \max_{i=1 \dots n} \{ p_i \}, \frac{1}{R} \sum_1^n p_i \};$ 
while  $i \leq n$  do
    if  $t + p_i \leq C_{\max}^*$  then
         $s_i^1 := t; z_i^1 := v; t := t + p_i; i := i + 1;$ 
    else
         $s_i^2 := t; z_i^2 := v; p_i := p_i - (C_{\max}^* - t); t := 0; v := v + 1;$ 
    end
end

```

List Scheduling - Approximation Alg. for $P \mid \text{prec} \mid C_{\max}$

Input: R , number of parallel identical resources, n , number of non-preemptive tasks and computation times (p_1, p_2, \dots, p_n) .
DAG of precedence constraints.

Output: n -element vectors s and z where s_i is the start time of T_i and z_i is the resource ID.

```

 $t_v := 0 \text{ for all } v \in 1 \dots R;$  // availability of resource
 $s_i = z_i := 0 \text{ for all } i \in 1 \dots n;$ 
Sort tasks in list L;
for  $i := 1$  to  $n$  do // for all tasks
     $k = \arg \min_{v=1 \dots R} \{ t_v \};$  // choose res. with the lowest  $t_v$ 
    In the set of free tasks, choose  $T_i$  which is first in the list L
    Remove  $T_i$  from the list L;
     $s_i = \max \{ t_k, \max_{j \in \text{Pred}(T_i)} \{ s_j + p_j \} \}; z_i = k;$  // assign  $T_i$  to  $P_k$ 
     $t_k = s_i + p_i;$  // update availability time of  $P_k$ 
end

```

Task T_i is **free** if its predecessors have been completed. $\text{Pred}(T_i)$ is a set of the task IDs that are **predecessors** of T_i . Complexity is $O(n)$.

May 15, 2018 41 / 83

The term $C_{\max}^* = \max \{ \max_{i=1 \dots n} \{ p_i \}, \frac{1}{R} \sum_1^n p_i \}$ should be interpreted as follows:

- component $\max_{i=1 \dots n} \{ p_i \}$ represents the sequential nature of each task - it's parts can be assigned to different resources, but these parts can not be run simultaneously. Note that each task can be divided into two parts at most.
- component $\frac{1}{R} \sum_1^n p_i$ represents a situation when all resources work without idle waiting

Example 1:

```

 $\rho = (2, 3, 2, 3, 2), R = 3$ 
compute  $C_{\max}^* = \max \{ 3, \frac{12}{3} \} = 4$ 

```

Example 2:

```

 $\rho = (10, 8, 4, 14, 1), R = 3$ 
compute  $C_{\max}^* = \max \{ 14, \frac{37}{3} \} = 14$ 

```

Z. Hanzalek (CTU) May 15, 2018 39 / 83

Scheduling

List Scheduling - Approximation algorithm for $P \mid \text{prec} \mid C_{\max}$

List Scheduling (LS) is a general heuristic useful in many problems.

- We have a list (n -tuple) of tasks and when some resource is free, we assign the first free task from the list to this resource.
- The accuracy of LS depends on the criterion and sorting procedure.

Approximation factor of LS algorithm [Graham 1966]

For $P \mid \text{prec} \mid C_{\max}$ (and also for $P \parallel C_{\max}$) and arbitrary (unsorted) list L, List Scheduling is an approximation algorithm with factor $r_{LS} = 2 - \frac{1}{R}$

An example illustrating the case when the factor is attained:

$n = (R - 1) \cdot R + 1,$
 $\rho = (1, 1, \dots, 1, R),$
 $\curvearrowleft \text{empty.}$

Illustration for $R = 4$

$P_1 T_1$	T_5	T_9	T_{13}
$P_2 T_2$	T_4	$T_7 T_{10}$	
$P_3 T_3$	T_5	$T_8 T_{11}$	
$P_4 T_4$	T_6	$T_9 T_{12}$	
	0	1	2
	1	2	3
	2	3	4
	3	4	5
	4	5	6
	5	6	7

$r_{LS} = 2 - \frac{1}{4} = \frac{7}{4}$

$L = (T_n, T_1, \dots, T_{n-1})$ May 15, 2018 40 / 83

$L' = (T_1, T_2, \dots, T_n)$ May 15, 2018 42 / 83

Z. Hanzalek (CTU) May 15, 2018 43 / 83

Anomalies of List Scheduling Algorithm

The LS algorithm depends not only on the **order of tasks in L**, but it exhibits **anomalies** (C_{max} surprisingly increases when relaxing some constraints/parameters) caused by:

- (1) **the decrease of processing time p_i**
- (2) **the removal of some precedence constraints**
- (3) **the increase of the number of resources R**

Example illustrating different anomalies for $R = 2$, $n = 8$, $\rho = (3, 4, 2, 4, 4, 2, 13, 2)$



Using list $L = (T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8)$,
LS finds solution with $C_{max}^* = 17$.

Scheduling Z. Hanzalek (CTU) May 15, 2018 43 / 83

LPT (Longest Processing Time First)
- Approximation Algorithm for $P \parallel C_{max}$

- The approximation factor of the LS algorithm can be decreased using the **Longest Processing Time first (LPT)** strategy

- During initialization of LS, we sort list L in a non-increasing order of p_i

Approximation factor of LPT algorithm [Graham 1966]

$$r_{LPT} = \frac{4}{3} - \frac{1}{9} = \frac{11}{9}$$

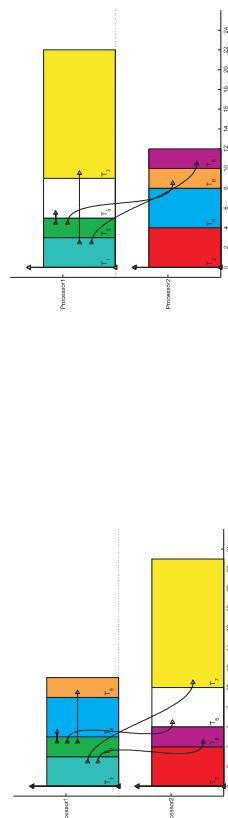
Time complexity of LPT algorithm is $O(n \cdot \log(n))$ due to the sorting.

Z. Hanzalek (CTU) May 15, 2018 45 / 83

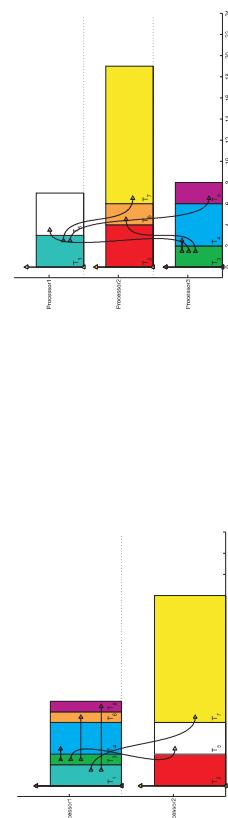
List Scheduling Anomalies - Prolongation of C_{max}

(2) Remove prec. constr. $T_3 \prec T_4$.

Exchange position of T_7 and T_8
 $L = (T_1, T_2, T_3, T_4, T_5, T_6, T_8, T_7)$.



(3) Add resource ($R = 3$).



May 15, 2018 44 / 83

LPT (Longest Processing Time First)
- Approximation Algorithm for $P \parallel C_{max}$

An example illustrating the case when the factor is attained:
 $\rho = (2R - 1, 2R - 1, 2R - 2, 2R - 2, \dots, R + 1, R + 1, R, R, R)$
 $n = 2 \cdot R + 1 \prec$ empty,

optimum:

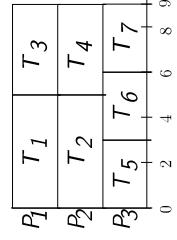


Illustration for $R = 3$



Factor of LPT algorithm

If the number of tasks is big, the factor can get better depending on k - the number of tasks assigned to the resource which finishes last:
 $r_{LPT} = 1 + \frac{1}{k} - \frac{1}{kR}$

May 15, 2018 46 / 83

Z. Hanzalek (CTU) May 15, 2018 47 / 83

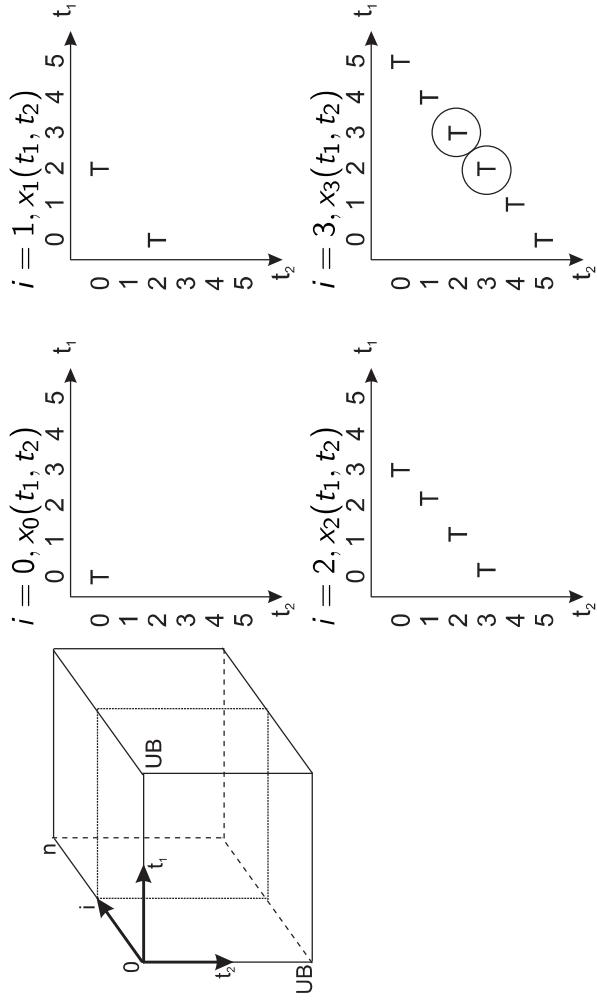
Dynamic Programming for $P \parallel C_{max}$ [Rothkopf]

Dynamic Programming for $P \parallel C_{max}$ [Rothkopf]

For fixed R , the number of processors, there is a pseudopolynomial algorithm - input instance is restricted to bounded nonnegative integers: number of tasks and their processing times.

- we add a binary variable $x_i(t_1, t_2, \dots, t_R)$ where
 - $i = 1, 2, \dots, n$ is the task index
 - $v = 1, 2, \dots, R$ is the index of the resource
 - $t_v = 0, 1, 2, \dots, UB$ is the time variable associated to the resource v
 - UB is upper bound on C_{max}
 - $x_i(t_1, t_2, \dots, t_R) = 1$ iff tasks T_1, T_2, \dots, T_i can be assigned to the resource such that P_v is occupied during the time interval $\langle 0, t_v \rangle; v = 1, 2, \dots, R$

Example n=3, R=2, p=(2,1,2), UB=5.



Example for $P \parallel C_{max}$ [Rothkopf]

Scheduling

May 15, 2018

47 / 83

Complexity of $P \parallel C_{max}$ [Rothkopf]

Scheduling

May 15, 2018

48 / 83

Input: R , the number of parallel identical resources, n , the number of nonpreemptive tasks and their processing time (p_1, p_2, \dots, p_n) .
Output: n -elements vectors s and z where s_i is the start time and z_i is the resource ID.

```

for  $(t_1, t_2, \dots, t_R) \in \{1, 2, \dots, UB\}^R$  do  $x_0(t_1, t_2, \dots, t_R) := 0$ ;
for  $x_0(0, 0, \dots, 0) := 1$ ; // for all tasks
for  $i := 1$  to  $n$  do
    for  $(t_1, t_2, \dots, t_R) \in \{0, 1, 2, \dots, UB\}^R$  do  $x_i(t_1, t_2, \dots, t_R) := 0$  // in the whole space
     $x_i(t_1, t_2, \dots, t_R) := OR_{v=1}^{R-1} x_{i-1}(t_1, t_2, \dots, t_v - p_i, \dots, t_R)$ ;
    //  $x_i()$  = 1 iff there existed
    //  $x_{i-1}()$  = 1 ‘‘smaller’’, by  $p_i$  in any direction
    end
end
 $C_{max}^* = \min_{x_n(t_1, t_2, \dots, t_R)=1} \{\max_{v=1,2,\dots,R} \{t_v\}\}$ ;
Assign tasks  $T_n, T_{n-1}, \dots, T_1$  in the reverse direction;
```

Time complexity is $O(n \cdot UB^R)$.

Z. Hanzalek (CTU)

May 15, 2018

48 / 83

For polynomially bounded R , (i.e., $R = poly(n)$) the problem is strongly NP-hard.

- Can be shown by reduction from 3-partition problem with $C_{max} = B$, $R = m$, $n = 3m$, $p_i = a_i$ (take care, in 3-partition the number 3 means number of items in the bin).
- Here, the Rothkopf algorithm is not pseudopolynomial, since its complexity is $O(n \cdot UB^R) = O(n \cdot UB^{poly(n)})$, which is equal to $O(n \cdot UB^{n/3})$ for the instances reduced from 3-partition.

- For any constant R , (e.g., $R = 2$) the problem is weakly NP-hard
- Here, the Rothkopf algorithm is pseudopolynomial, since its complexity is $O(n \cdot UB^2)$.

- It can not be used to solve the instance reduced from 3-partition, but it can be used to solve the instances reduced from 2-partition problem, which is weakly NP-hard (take care, in 2-partition the number 2 means number of bins).

May 15, 2018

49 / 83

Z. Hanzalek (CTU)

May 15, 2018

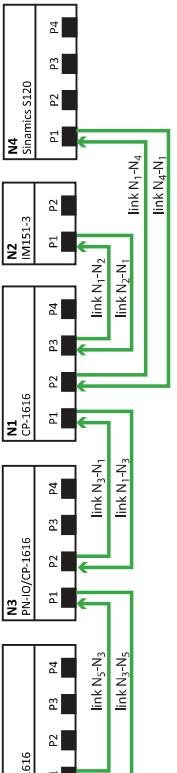
50 / 83

Motivation Example: Message Scheduler for Profinet IO IRT - Specification

Motivation Example: Message Scheduler for Profinet IO IRT - Specification

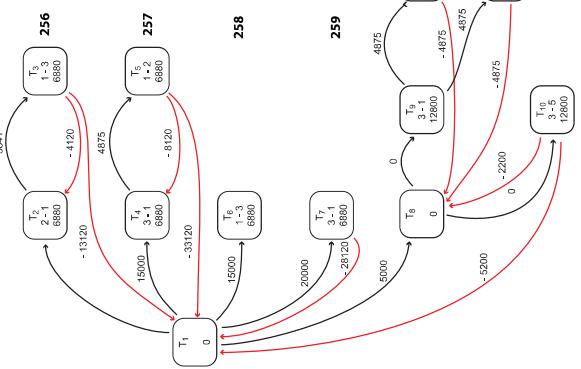
Profinet IO IRT is an Ethernet-based hard-real time communication protocol, which uses static schedules for time-critical data. Each node contains a special hardware switch that intentionally breaks the standard forwarding rules for a specified part of the period to ensure that no queuing delays occur for time-critical data.

Goal: Minimize the makespan (the schedule length) for time critical messages.



line | $N_1 \rightarrow N_3$ | $N_1 \rightarrow N_4$ | $N_1 \rightarrow N_2$ | $N_2 \rightarrow N_1$ | $N_3 \rightarrow N_1$ | $N_4 \rightarrow N_1$ | $N_3 \rightarrow N_5$ | $N_5 \rightarrow N_3$
line delay [ns] | 4875 | 5130 | 5862 | 3841 | 4875 | 4895 | 4875 | 4875

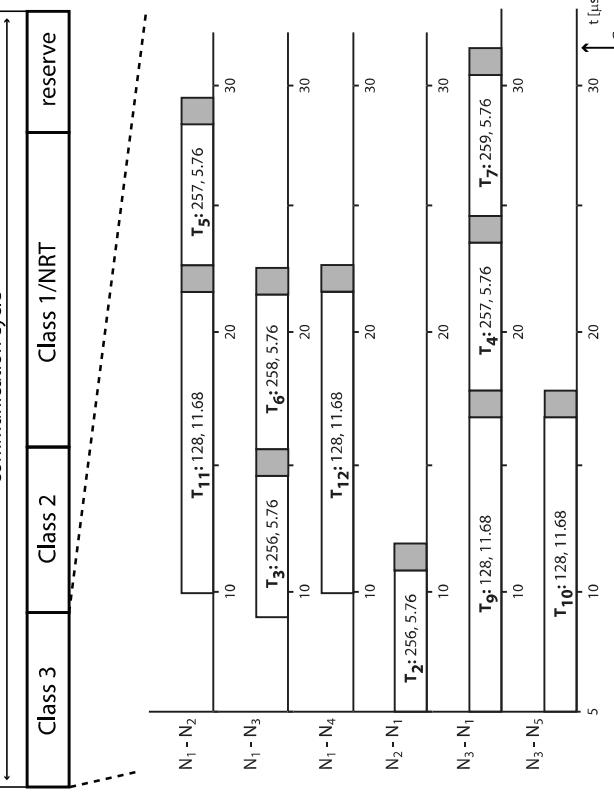
Motivation Example: Message Scheduler for Profinet IO IRT - Formalization



Can be formulated as $PSm, 1 \mid temp \mid C_{max}$ problem.

- task = message on a given line
- positive cost edge = r , precedence relations
- negative cost edge = d , end-to-end delay
- unicast message = chain of tasks (assuming positive edges)
- multicast message = out-tree of tasks (assuming positive edges)

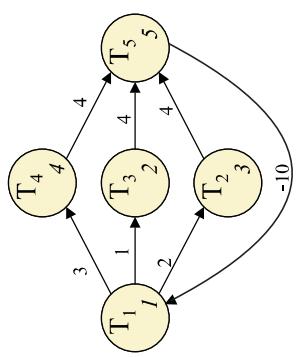
Motivation Ex.: Message Sch. for Profinet IO IRT - Result



Temporal Constraints

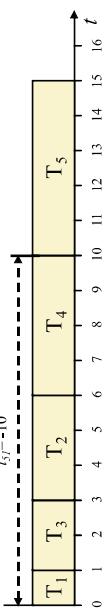
Temporal Constraints

- Set of non-preemptive tasks $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ is represented by the nodes of the directed graph G (may include negative cycles).
- Processing time p_i is assigned to each task.



- The edges represent temporal constraints. Each edge from T_i to T_j has the length l_{ij} .
- Each temporal constraint is characterized by one inequality

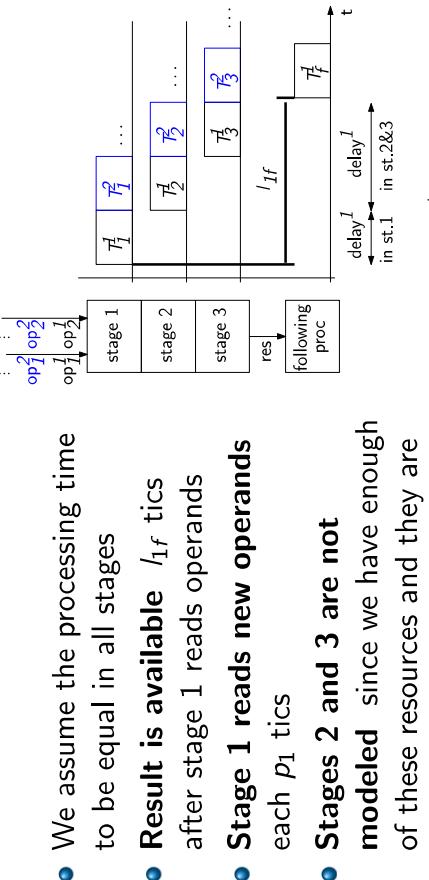
$$s_i + l_{ij} \leq s_j.$$



Temporal Constraints $s_i + l_{ij} \leq s_j$ with Positive l_{ij}

Temporal Constraints $s_i + l_{ij} \leq s_j$ with Positive l_{ij}

- b.2) another example with $l_{ij} > p_i$ - pipe-lined ALU

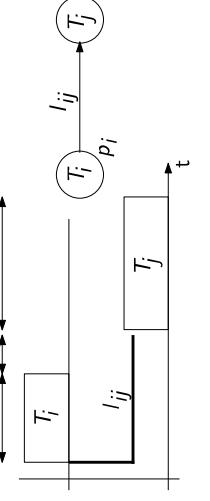


- We assume the processing time to be equal in all stages
- **Result is available** l_{if} tics after stage 1 reads operands
- **Stage 1 reads new operands** each p_1 tics
- **Stages 2 and 3 are not modeled** since we have enough of these resources and they are synchronized with stage 1

Temporal Constraints (also called a generalized precedence constraint or a positive-negative time lag)

- the start time of one task depends on the start time of another task

- a) $l_{ij} = p_i$
 - “normal” precedence relation
 - the second task can start when the previous task is finished



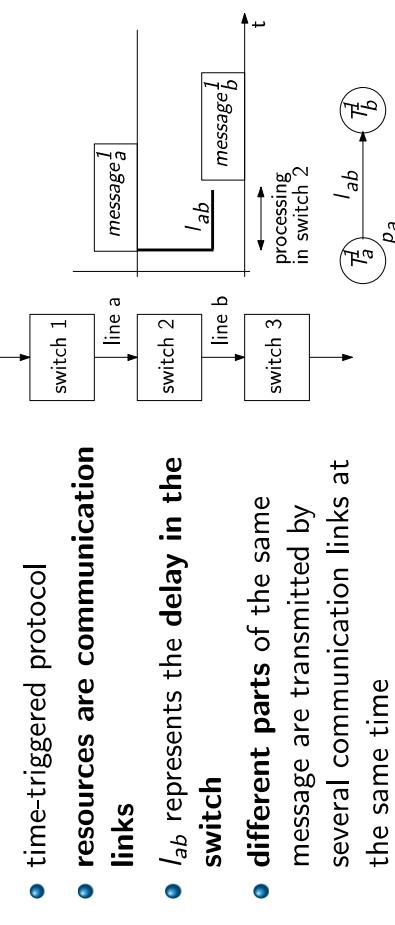
- b) $l_{ij} > p_i$
 - the second task can start some time after the completion of previous task
 - b.1) example of a dry operation performed in sufficiently large space

Temporal Constraints $s_i + l_{ij} \leq s_j$ with Positive l_{ij}

- c) $0 < l_{ij} < p_i$

Partial results of the previous task may be used to start the execution of the following task.

E.g. the **cut-through** mechanism, where the switch starts transmission on the output port earlier than it receives the complete message on the input port.



- time-triggered protocol
- **resources are communication links**
- l_{ab} represents the delay in the switch
- **different parts** of the same message are transmitted by several communication links at the same time

Z. Hanzalek (CTU) May 15, 2018 61 / 83

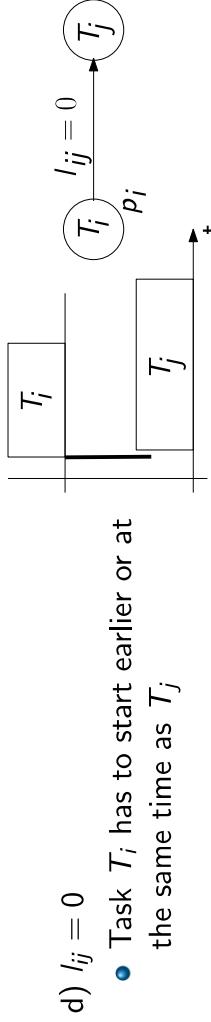
Z. Hanzalek (CTU) May 15, 2018 60 / 83

Z. Hanzalek (CTU) May 15, 2018 59 / 83

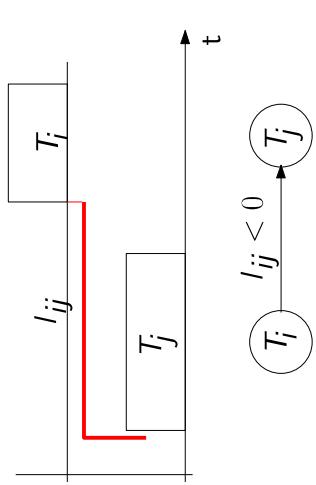
Z. Hanzalek (CTU) May 15, 2018 62 / 83

Temporal Constraints $s_i + l_j \leq s_j$ with Zero or Negative l_j

Cycles and Relative Time Windows



- e) $l_{ij} < 0$
- Task T_i has to start earlier or at most $|l_{ij}|$ later than T_j
 - It loses the sense of "normal" precedence relation, since T_i does not have to precede T_j
 - It represents the relative deadline of T_i related to the start-time of T_j



ILP formulation of PS1 |temp| C_{max}

Z. Hanzalek (CTU) May 15, 2018 63 / 83

Time-indexed Model for PS1 |temp| C_{max}

Z. Hanzalek (CTU) May 15, 2018 64 / 83

Task can be represented in two ways:

- Time-indexed** - ILP model is based on variable x_{it} , which is equal to 1 iff $s_i = t$. Otherwise, it is equal to zero. Processing times are assumed to be positive integers.
- Relative-order** - ILP model is based on the relative order of tasks given by variable x_{ij} , which is equal to 1 iff task T_i precedes task T_j . Otherwise, it is equal to zero. The processing times are nonnegative real numbers (tasks with zero processing time may be used to represent events).

Both models contain two types of constraints:

- temporal constraints
- resource constraints - prevent overlapping of tasks

$\min C_{max}$

$$\begin{aligned} \sum_{t=0}^{UB-1} (t \cdot x_{it}) + l_{ij} &\leq \sum_{t=0}^{UB-1} (t \cdot x_{jt}) \quad \forall l_{ij} \neq -\infty \text{ a } i \neq j \text{ (temp. const.)} \\ \sum_{i=1}^n \left(\sum_{k=\max(0, t-p_i+1)}^t x_{ik} \right) &\leq 1 \quad \forall t \in \{0, \dots, UB-1\} \text{ (resource)} \\ \sum_{t=0}^{UB-1} x_{it} &= 1 \quad \forall i \in \{1, \dots, n\} \text{ (} T_i \text{ is scheduled)} \\ \sum_{t=0}^{UB-1} (t \cdot x_{it}) + p_i &\leq C_{max} \quad \forall i \in \{1, \dots, n\} \end{aligned}$$

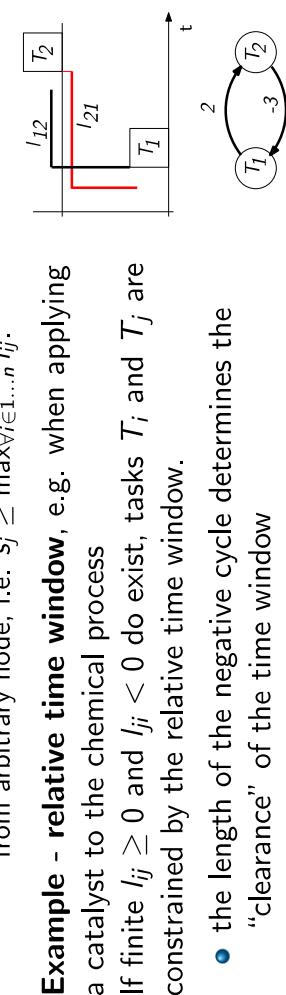
variables: $x_{it} \in \{0, 1\}$, $C_{max} \in \{0, \dots, UB\}$

UB - upper bound of C_{max} (e.g. $UB = \sum_{i=1}^n \max\{p_i, \max_{j \in \{1, \dots, n\}} l_{ij}\}$).

Start time of T_i is $s_i = \sum_{t=0}^{UB-1} (t \cdot x_{it})$.

Model contains $n \cdot UB + 1$ variables and $|E| + UB + 2n$ constraints. Constant $|E|$ represents the number of temporal constraints (edges in G).

- Absence of a positive cycle in graph G**
- is a **necessary condition** for schedulability of PS1 |temp| C_{max}
 - is a **necessary and sufficient condition** for schedulability of the instance with **unlimited capacity of resources**. The schedule, which is restricted only by the temp. constraints, can be found in pol. time by LP or
 - by the longest paths. For G we can create G' , a **complete digraph of longest paths**, where weight l_{ij} is the length of the longest directed path from T_i to T_j in G (if no directed path in G exists, the weight is $l_{ij} = -\infty$). A start time of T_j is lower bounded by the longest path from arbitrary node, i.e. $s_j \geq \max_{i \in 1 \dots n} l_{ij}$.



Scheduling May 15, 2018 64 / 83

Time-indexed Model for PS1|temp| C_{max}

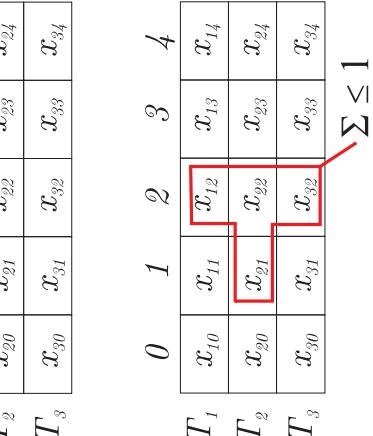
Relative-order Model for PS1|temp| C_{max}

$$\mathcal{T} = \{T_1, T_2, T_3\}, \rho = (1, 2, 1), UB = 5$$

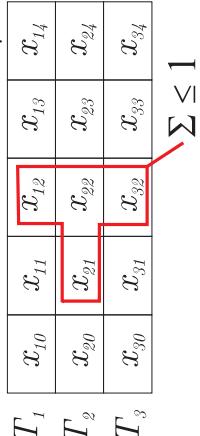
$$0 \quad 1 \quad 2 \quad 3 \quad 4 \quad \Sigma = 1$$

T_1	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
T_2	x_{20}	x_{21}	x_{22}	x_{23}	x_{24}
T_3	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}

T_1 is scheduled:



Resource constr. at time 2:



Z. Hanzalek (CTU) May 15, 2018 67 / 83

Relative-order Model for PS1|temp| C_{max}

$$\begin{aligned} \min C_{max} \\ s_i + l_{ij} \leq s_j \\ \rho_j \leq s_i - s_j + UB \cdot x_{ij} \leq UB - \rho_i \quad \forall i, j \in \{1, \dots, n\} \text{ and } i < j \end{aligned}$$

(resource constraint)

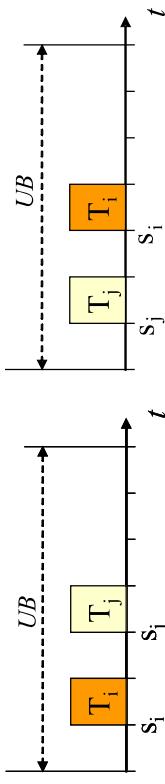
$$\begin{aligned} \rho_i + \rho_j \leq C_{max} \\ \forall i \in \{1, \dots, n\} \end{aligned}$$

$$\text{variables: } x_{ij} \in \{0, 1\}, C_{max} \in \langle 0, UB \rangle, s_i \in \langle 0, UB - \rho_i \rangle$$

The model contains $n + (n^2 - n)/2 + 1$ variables and $|E| + (n^2 - n) + n$ constraints. $|E|$ is a number of temporal constraints (edges in G).

Resource constraint for couple of tasks:
 $\rho_j \leq s_i - s_j + UB \cdot x_{ij} \leq UB - \rho_i$

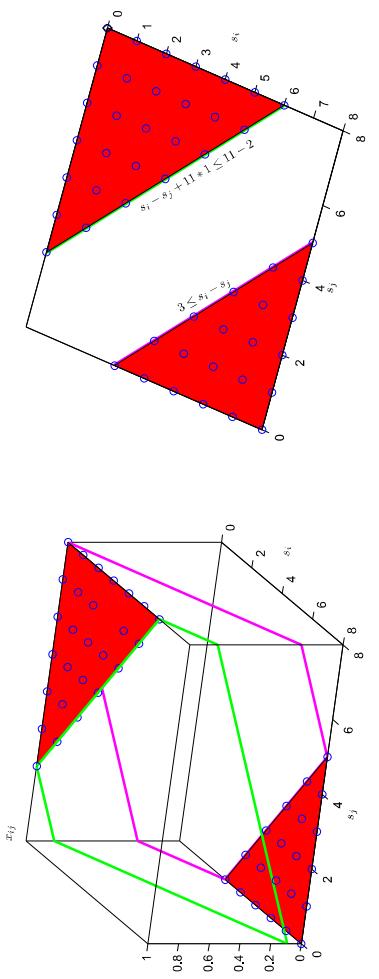
The constraint uses "big M" (here UB - upper bound on C_{max}).
If $x_{ij} = 1$, T_i precedes task T_j and the constraint is formulated as
 $s_i + \rho_i \leq s_j$.



Z. Hanzalek (CTU) May 15, 2018 68 / 83

Relative-order Model for PS1|temp| C_{max}

Example: no temporal constraints, two tasks T_i, T_j with $\rho_i = 2$ and $\rho_j = 3$. We set $UB = 11$ and we study $s_i \in \langle 0, 8 \rangle$.
3D polytope (left) is determined by the resource constr. given by violet and green hyperplanes (see colors on the previous slide). Its projection to 2D space (right) shows both sequences of tasks. When we change UB , the hyperplanes in 3D decline -each of the moves the vertex with acute angle.



Z. Hanzalek (CTU) May 15, 2018 69 / 83

Scheduling

May 15, 2018 70 / 83

Z. Hanzalek (CTU)

May 15, 2018 70 / 83

Scheduling

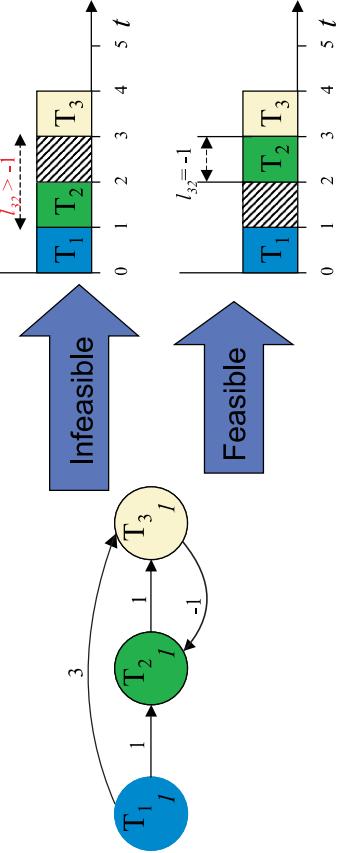
Comparison of the Two Models

Feasibility Test for Heuristic Algorithms

If the partial schedule (found for example by a greedy algorithm which inserts tasks in a topological order of edges with positive weight, or the partial result during the Branch and Bound algorithm) violates some time constraints, the order of tasks does not need to be feasible.

Time-indexed model:

- (+) Can be easily extended for parallel identical processors.
 - (+) ILP formulation does not need many constraints.
 - (-) The size of the model grows with the size of UB .
- Relative-order model:
- (+) The size of ILP model does not depend on UB .
 - (-) Requires a big number of constraints.



When the optimal order of the tasks in the schedule is known (variables x_{ij} are constants), it is easy to find the start time of the tasks (for example by LP formulation involving time constraints only).

Relative-order Model for Project Scheduling with Dedicated Resources of Unit Capacity $PSm, 1 \mid \text{temp} \mid C_{\max}$

Part of the input parameters are the number of resources m and **assignment of the tasks to the resources** ($a_1, \dots, a_i, \dots, a_n$), where a_i is index of the resource type on which task T_i will be running.

$$\min C_{\max}$$

$$s_i + l_{ij} \leq s_j$$

$$\forall l_{ij} \neq -\infty \text{ and } i \neq j$$

(temporal constraints)

$$p_j \leq s_i - s_j + UB \cdot x_{ij} \leq UB - p_i$$

$$\forall i, j \in \{1, \dots, n\}, i < j \text{ and } a_i = a_j$$

(on the same resource type)

$$\forall i \in \{1, \dots, n\}$$

variables: $x_{ij} \in \{0, 1\}$, $C_{\max} \in \langle 0, UB \rangle$, $s_i \in \langle 0, UB \rangle$

Model consists of less than $n + (n^2 - n)/2 + 1$ variables (exact number depends on the number of tasks scheduled on each resource type).

Z. Hanzalek (CTU)	Scheduling	May 15, 2018	71 / 83
Z. Hanzalek (CTU)	Scheduling	May 15, 2018	72 / 83

Modeling with Temporal Constraints

Using $PSm, 1 \mid \text{temp} \mid C_{\max}$ we will model:

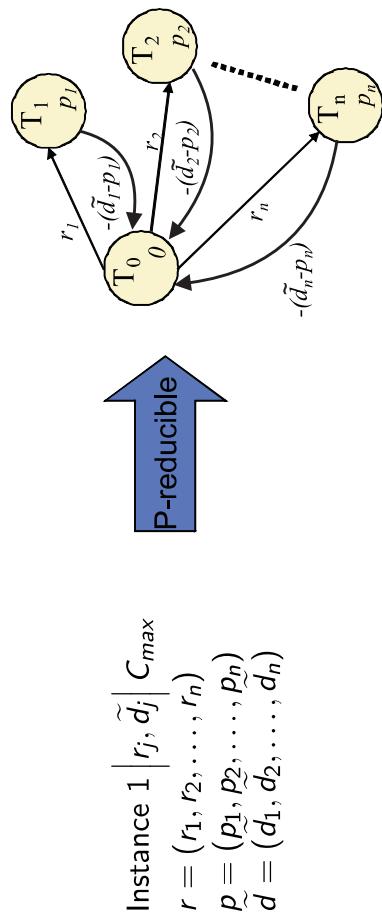
- $1 \mid r_j, d_j \mid C_{\max}$
 - scheduling on dedicated resources $PSm, 1 \mid \text{temp} \mid C_{\max}$
- Using $PSm, 1 \mid \text{temp} \mid C_{\max}$ we will model:
- scheduling of **multiprocessor task** - task needs more than one resource type at a given moment,

Z. Hanzalek (CTU)	Scheduling	May 15, 2018	73 / 83
Z. Hanzalek (CTU)	Scheduling	May 15, 2018	74 / 83

Reduction from $PSm, 1 | r_j, \tilde{d}_j | C_{max}$ to $PS1 | temp | C_{max}$

Reduction from $PSm, 1 | temp | C_{max}$ to $PS1 | temp | C_{max}$

This polynomial reduction proves that $PS1 | temp | C_{max}$ is NP-hard, since Brattey's problem is NP-hard.



Reduction from $PSm, 1 | temp | C_{max}$ to $PS1 | temp | C_{max}$

Reduction from $PSm, 1 | temp | C_{max}$ to $PS1 | temp | C_{max}$

Reduction from $PSm, 1 | temp | C_{max}$ to $PS1 | temp | C_{max}$ is based on the projection of **each resource to the independent time window**. In other words, the schedule of tasks on P^j is projected into interval $\langle (j-1) \cdot UB, j \cdot UB \rangle$

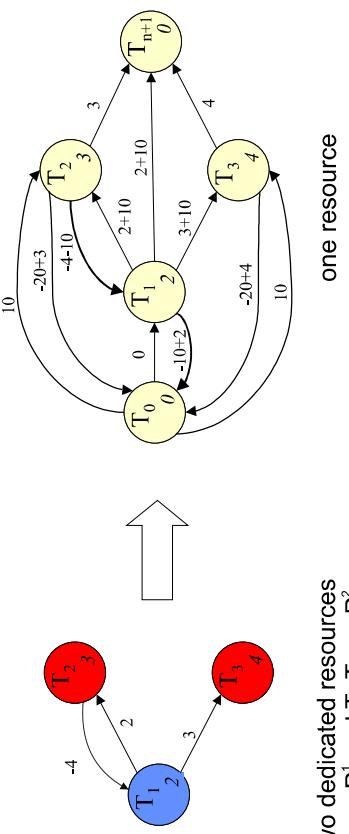
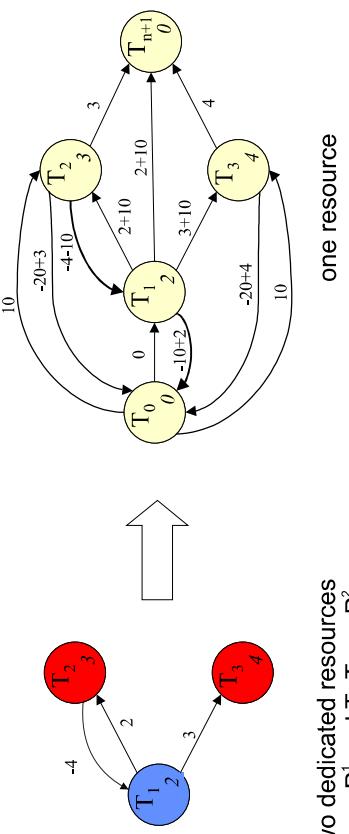
Transformation consists of two steps:

- **Add dummy tasks** T_0 and T_{n+1} with $p_0 = p_{n+1} = 0$.
 - Task T_0 , processed on P^1 , precedes all tasks $T_i \in \mathcal{T}$, ie. $s_0 \leq s_i$.
 - Task T_{n+1} , processed on P^m , follows all task $T_i \in \mathcal{T}$, tj. $s_i + p_i \leq s_{n+1}$.
- **Transform the original temporal constraints** to

$$l'_j = l_j + (a_j - a_i) \cdot UB.$$

Reduction from $PSm, 1 | temp | C_{max}$ to $PS1 | temp | C_{max}$

Reduction from $PSm, 1 | temp | C_{max}$ to $PS1 | temp | C_{max}$



While minimizing the completion time of T_{n+1} , we push tasks T_1, T_2 and T_3 "to the left" due to the edges entering T_{n+1}

The new start time s'_i of each task on processor a_i is:
 $s'_i = s_i + (a_j - 1) \cdot UB$.

Temporal constraints $s_i + l_{ij} \leq s'_j$ are transformed to:

$$s'_i - (a_i - 1) \cdot UB + l_{ij} \leq s'_j - (a_j - 1) \cdot UB$$

$$s'_i + l_{ij} + (a_j - a_i) \cdot UB \leq s'_j$$

$$l'_{ij} = l_{ij} + (a_j - a_i) \cdot UB$$

The transformed temporal constraint will look like $s'_i + l'_{ij} \leq s'_j$, where:

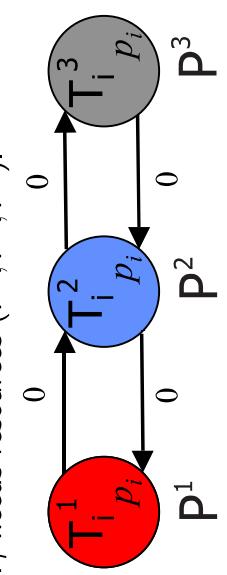
Multiprocessors Task

Project Scheduling with Dedicated Resources of Different Capacity $PSm, R | \text{temp} | C_{max}$

Transformation of multiprocessor task problem to $PSm, 1 | \text{temp} | C_{max}$

- create as many virtual tasks as there are processors needed to execute the physical tasks
- ensure that the virtual tasks of the given physical task start at the same time - this is done by two edges with weight $l_{ij} = l_{ji} = 0$. Consequently $s_i \leq s_j$ and $s_j \leq s_i$.

Example: Task T_i needs resources (P^1, P^2, P^3) .



Relative-order ILP Model for $PSm, R | \text{temp} | C_{max}$

The assignment $Z_{ivk} \in \{0, 1\}$ is equal to 1 if task i is assigned to unit v of resource k , and 0 otherwise.

We define $\{i, j\} \in \mathcal{M}$ iff **task i and task j are assigned to resource k of finite capacity** (i.e., $\exists k \in \mathcal{R} : r_{ik} \cdot r_{jk} \geq 1$ and $R_k < \infty$) and therefore we **have to avoid a collision** of task i and task j . We define $\mathcal{V} = \{1, \dots, n\}$.

$$\min C_{max}$$

subject to:

$$\begin{aligned} (1) \quad & s_j - s_i \geq l_{ij} && \forall (i, j) \in \mathcal{V}^2 : i \neq j \\ & s_j - s_j + UB \cdot x_{ij} + UB \cdot y_{ij} \geq \rho_j && \forall (i, j) \in \mathcal{V}^2 : i \neq j, \{i, j\} \in \mathcal{M} \\ & s_j - s_j + UB \cdot x_{ij} - UB \cdot y_{ij} \leq UB - \rho_i && \forall (i, j) \in \mathcal{V}^2 : i \neq j, \{i, j\} \in \mathcal{M} \\ & -x_{ij} + y_{ij} \leq 0 && \forall (i, j) \in \mathcal{V}^2, \forall k \in \{1, \dots, M\}, \forall v \in \{1, \dots, R_k\} : \\ & z_{ivk} + z_{jk} - 1 \leq 1 - y_{ij} && i \neq j, \{i, j\} \in \mathcal{M} \\ & \sum_{v=1}^{R_k} z_{ivk} = r_{ik} && \forall i \in \mathcal{V}, \forall k \in \mathcal{R} : r_{ik} \geq 1, R_k < \infty \\ & s_i + \rho_i \leq C_{max} && \forall i \in \mathcal{V} \end{aligned} \quad (7) \quad (8)$$

Project Scheduling with Dedicated Resources of Different Capacity $PSm, R | \text{temp} | C_{max}$

Resource $k \in \{1, 2, \dots, m\}$ has a capacity of $R_k \in \mathbb{Z}^+ \cup \{\infty\}$ units.

Task i requires $r_{ik} \in \mathbb{Z}_0^+$ units of resource k , with $0 \leq r_{ik} \leq R_k$.

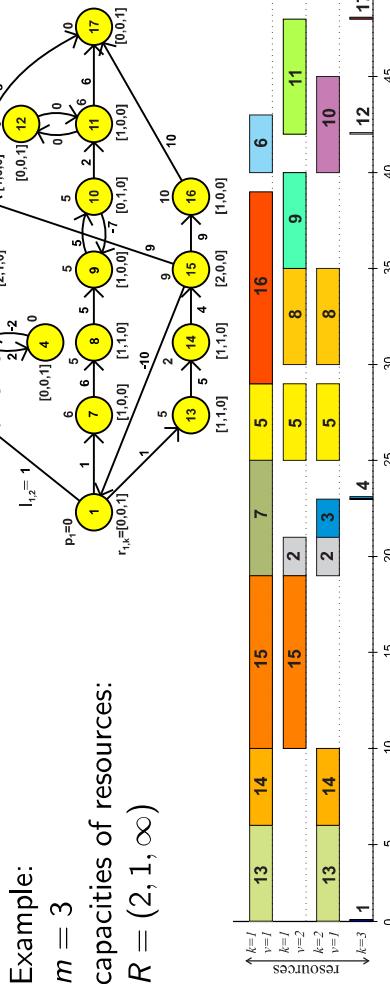
Multiprocessor tasks - Multiple resources may be required by one task.

Example:

$$m = 3$$

capacities of resources:

$$R = (2, 1, \infty)$$



Relative-order ILP Model for $PSm, R | \text{temp} | C_{max}$

- Constraints (3), (4), (5) and binary variables x_{ij} and y_{ij}
- When $x_{ij} = 0$ and $y_{ij} = 0$, constraints (3) and (4) reduce to **$s_j + \rho_j \leq s_i$, i.e., j is followed by i on the same unit.**
- When $x_{ij} = 1$ and $y_{ij} = 0$, constraints (3) and (4) reduce to **$s_i + \rho_i \leq s_j$, i.e., i is followed by j on the same unit.**
- When $x_{ij} = 1$ and $y_{ij} = 1$, constraints (3) and (4) are eliminated in effect and the activities i and j do not share the same unit.

Combination $x_{ij} = 0$ and $y_{ij} = 1$ is not feasible due to constraint (5).

- Constraint (6) states that **when $y_{ij} = 1$ then the activities do not share the same unit v of resource k since $Z_{ivk} + Z_{jk} \leq 1$.**
- Constraint (7) states that each **task i is assigned to the appropriate number of units r_{ik} for each resource k .**

References

- 1 J. Blažewicz, K. Ecker, G. Schmidt, and J. Węglarz.
Scheduling Computer and Manufacturing Processes.
Springer, second edition, 2001.
- 2 Klaus Neumann, Christoph Schwindt, and Jürgen Zimmermann.
Project Scheduling with Time Windows and Scarce Resources.
Springer, 2003.
- 3 Sigrid Knust Peter Brucker.
Complexity results for scheduling problems.
<http://www.ict.kth.se/courses/ID2204/index.html>.

Constraint Programming

Zdeněk Hanzálek, Jan Kelbel
hanzalek@fel.cvut.cz

CTU in Prague

May 24, 2017

What is Constraint Programming?

May 24, 2017 1 / 33

Z. Hanzálek (CTU)

May 15, 2018 83 / 83

Z. Hanzálek (CTU)

May 24, 2017 1 / 33

1 Inspiration - Sudoku

2 Constraint Satisfaction Problem (CSP)

- Search and Propagation
- Arc Consistency
- AC-3 Algorithm
- Global Constraints

What is Constraint Programming?

- Sudoku is Constraint Programming

May 24, 2017 3 / 33

Z. Hanzálek (CTU)

May 24, 2017 2 / 33

Z. Hanzálek (CTU)

May 24, 2017 3 / 33

Z. Hanzálek (CTU)

Motivation - Sudoku

Sudoku

		2	5					
9				7	3			
	2		9		6			
2			4	9				
		7						
6	9			1				
8	4		1					
6	3			8				
		6	8					

Assign digits to blank fields such that:
digits distinct per row, column, block

May 24, 2017 4 / 33

Z. Hanzálek (CTU) Constraint Programming

Sudoku

		9	2	5				
	2			7	3			
2			4	9				
		7						
6	9			1				
8	4		1					
6	3			8				
		6	8					

Assign digits to blank fields such that:
digits distinct per row, column, block

May 24, 2017 5 / 33

Z. Hanzálek (CTU) Constraint Programming

Sudoku

			2	5				
		9						
		2						
2			4	9				
		7						
6	9			1				
8	4		1					
6	3			8				
		6	8					

Assign digits to blank fields such that:
digits distinct per row, column, block

May 24, 2017 6 / 33

Z. Hanzálek (CTU) Constraint Programming

Sudoku

		9	2	5				
	2			7	3			
2			4	9				
		7						
6	9			1				
8	4		1					
6	3			8				
		6	8					

Assign digits to blank fields such that:
digits distinct per row, column, block

May 24, 2017 7 / 33

Z. Hanzálek (CTU) Constraint Programming

Sudoku - Propagation in the Lower Left Block

	8	
	6	3

No blank field in the block can have a value of 3,6,8

Sudoku - Propagation in One Field

1,2,4,5,7,9	8	1,2,4,5,7,9
1,2,4,5,7,9	6	3
1,2,4,5,7,9	1,2,4,5,7,9	1,2,4,5,7,9

No blank field in the block can have a value of 3,6,8

- propagate to all blank fields
- Use the same propagation for rows and columns

Sudoku - Propagation in One Field

Z. Hanzálek (CTU)

Constraint Programming

May 24, 2017

8 / 33

	2	5				
	9					
	2					
2				4	9	

	2	5				
	9					
	2					
2				4	9	

1,3,5,6,7,8

Prune digits from the fields such that:
digits distinct per row, column, block

Prune digits from the fields such that:
digits distinct per row, column, block

1,2,3,4,5,6,7,8,9

	2	5				
	9					
	2					
2				4	9	

Z. Hanzálek (CTU)

Constraint Programming

May 24, 2017

10 / 33

Z. Hanzálek (CTU)

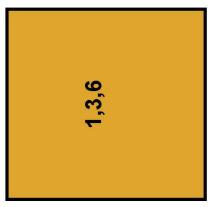
Constraint Programming

May 24, 2017

11 / 33

Sudoku - Propagation in One Field

	2	5						
9			7	3				
2		9	6					
2		4	9					
		7						
6	9							
8	4		1					
6	3			8				
		6	8					



Prune digits from the fields such that:
digits distinct per row, column, block

Sudoku - Iterated Propagation

May 24, 2017 12 / 33
Z. Hanzálek (CTU) Constraint Programming

	2	5						
9			7	3				
2		9	6					
2		4	9					
		7						
6	9							
8	4		1					
6	3			8				
		6	8					

- Iterate propagation for rows, columns and blocks
 - When to stop?
 - What if more assignments exist?
 - What if no assignment exists?

- Constraint programming is **declarative programming**:
 - Model:** variables, domains, constraints
 - Solver:** propagation, searching



	2	5						
9			7	3				
2		9	6					
2		4	9					
		7						
6	9							
8	4		1					
6	3			8				
		6	8					

Prune digits from the fields such that:
digits distinct per row, column, block

Sudoku is Constraint Programming

May 24, 2017 13 / 33
Z. Hanzálek (CTU) Constraint Programming

	2	5						
9			7	3				
2		2			9	6		
2					4	9		
					7			
6	9							
8	4		1					
6	3			8				
		6	8					

Constraint Satisfaction Problem - Formal Description

Terminology - CSP, CSOP, Constraint Solving and CP

Constraint Satisfaction Problem (CSP) is defined by the triplet (X, D, C) , where:

- $X = \{x_1, \dots, x_n\}$ is a finite set of variables
- $D = \{D_1, \dots, D_n\}$ is a finite set of domains of variables
- $C = \{C_1, \dots, C_t\}$ is a finite set of constraints.

Domain $D_i = \{v_1, \dots, v_k\}$ is a **finite** set of all possible values of x_i .

Constraint C_i is a couple (S_i, R_i) where $S_i \subseteq X$ and R_i is a **relation** over the set of variables S_i . For $S_i = \{x_{i_1}, \dots, x_{i_r}\}$ is $R_i \subseteq D_{i_1} \times \dots \times D_{i_r}$.

CSP is an NP-complete problem.

Example: $x \in \{3, 4, 5\}$, $y \in \{3, 4, 5\}$, $x \geq y$, $y > 3$

- propagate $y > 3$: $x \in \{3, 4, 5\}$, $y \in \{4, 5\}$

How it Works - Search and Propagation

May 24, 2017

16 / 33

Constraint Programming

Z. Hanzálek (CTU)

May 24, 2017

17 / 33

How it Works - Search and Propagation

Example: $x \in \{3, 4, 5\}$, $y \in \{3, 4, 5\}$, $x \geq y$, $y > 3$

- propagate $y > 3$: $x \in \{3, 4, 5\}$, $y \in \{4, 5\}$

How it Works - Search and Propagation

Constraint Programming

May 24, 2017

18 / 33

Z. Hanzálek (CTU)

May 24, 2017

18 / 33

Constraint Programming

Z. Hanzálek (CTU)

May 24, 2017

18 / 33

Constraint Programming

Z. Hanzálek (CTU)

May 24, 2017

18 / 33

How it Works - Search and Propagation

Example: $x \in \{3, 4, 5\}, y \in \{3, 4, 5\}, x \geq y, y > 3$

- ➊ propagate $y > 3$: $x \in \{3, 4, 5\}, y \in \{4, 5\}$
- ➋ propagate $x \geq y$: $x \in \{4, 5\}, y \in \{4, 5\}$

- ➌ propagation alone is not enough
 - the product of the domains (including infeasible $x = 4, y = 5$) is a superset of the solution
 - the search helps - we create subproblems

How it Works - Search and Propagation

Example: $x \in \{3, 4, 5\}, y \in \{3, 4, 5\}, x \geq y, y > 3$

- ➊ propagate $y > 3$: $x \in \{3, 4, 5\}, y \in \{4, 5\}$
- ➋ propagate $x \geq y$: $x \in \{4, 5\}, y \in \{4, 5\}$

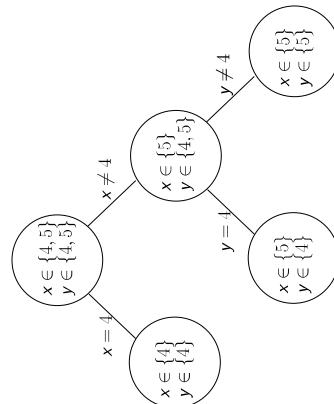
- ➌ propagation alone is not enough
 - the product of the domains (including infeasible $x = 4, y = 5$) is a superset of the solution
 - the search helps - we create subproblems

How it Works - Search and Propagation

Example: $x \in \{3, 4, 5\}, y \in \{3, 4, 5\}, x \geq y, y > 3$

- ➊ propagate $y > 3$: $x \in \{3, 4, 5\}, y \in \{4, 5\}$
- ➋ propagate $x \geq y$: $x \in \{4, 5\}, y \in \{4, 5\}$

- ➌ propagation alone is not enough
 - the product of the domains (including infeasible $x = 4, y = 5$) is a superset of the solution
 - the search helps - we create subproblems

- ➍ in the subproblems we use the propagation again
 -  The diagram illustrates the propagation of constraints. It starts with two nodes: Node 1 has domain {x: {4, 5}, y: {4, 5}} and constraint x ≠ 4. Node 2 has domain {x: {5}, y: {4, 5}} and constraint y = 4. Node 3 has domain {x: {5}, y: {5}} and constraint y ≠ 4. Node 4 has domain {x: {5}, y: {5}} and constraint y = 4. Arrows show the flow from Node 1 to Node 2, and from Node 2 to Node 3, and finally from Node 3 to Node 4.

Comparison with ILP

Z. Hanzálek (CTU) Constraint Programming May 24, 2017 18 / 33

Comparison with ILP

- In both cases we deal with declarative programming
- Performance differs from problem to problem
- CSP allows one to formulate **complex constraints**
- (ILP uses inequalities only, CSP uses an arbitrary relation - e.g. a binary relation may be given by a set of compatible tuples)

- CSP is more flexible, formulation is easier to understand
- it is difficult to represent continuous problems by CSP
 - domains of real variables can be bypassed by using hybrid approaches
 - e.g. combination with LP
- CP is new technique, it is more open

- By the **propagation** of the constraints we **filter the domains** of the variables.

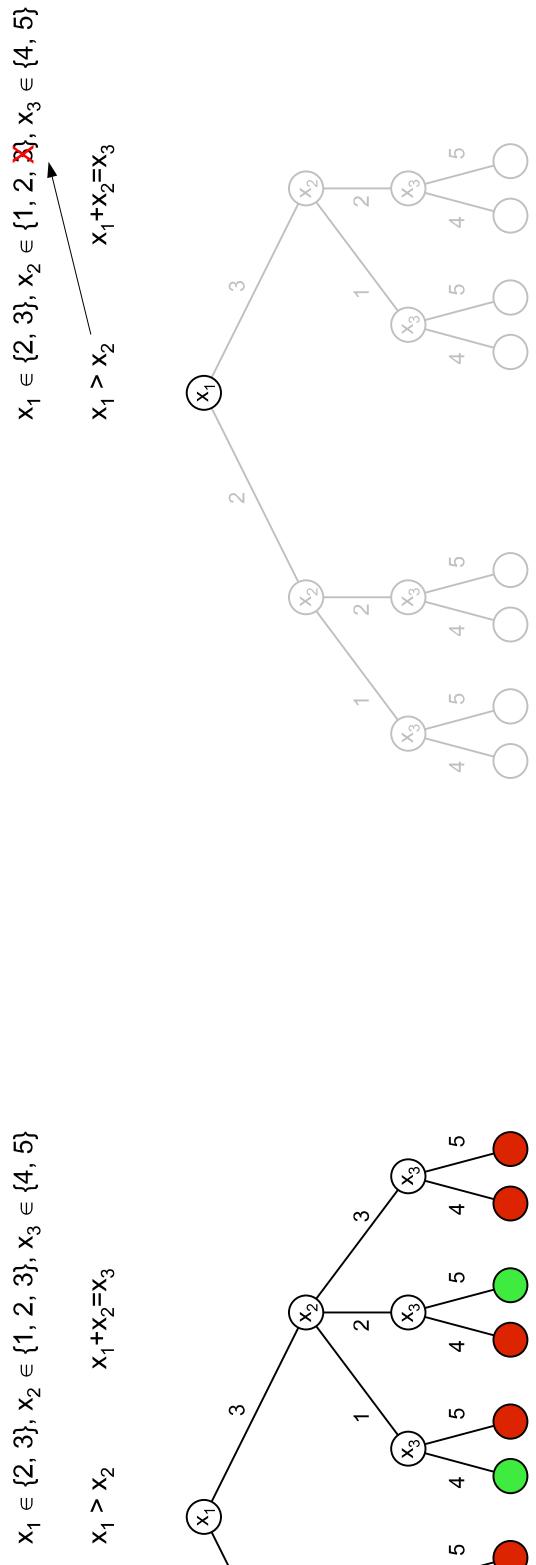
Z. Hanzálek (CTU) Constraint Programming May 24, 2017 18 / 33

Z. Hanzálek (CTU) Constraint Programming May 24, 2017 19 / 33

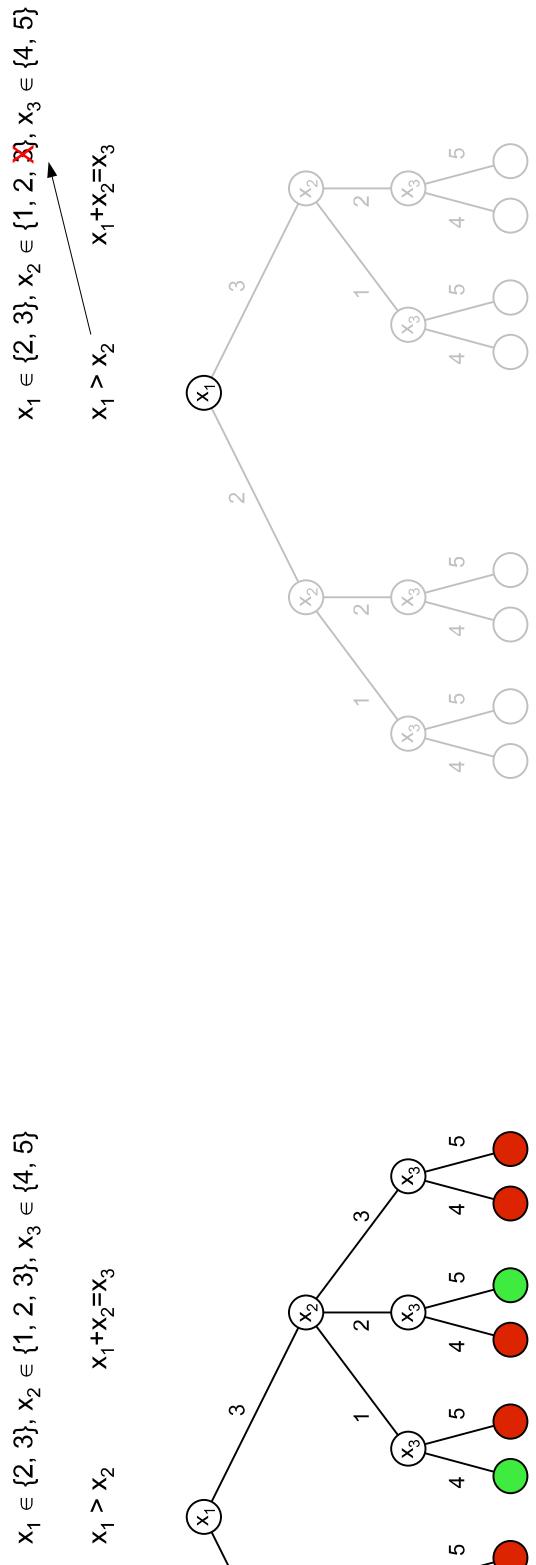
Example: Search and Propagation

Example: Search and Propagation

Complete search (for example Depth First Search):



Initial propagation of constraints:



Choose $x_1 = 2$ and propagate constraints:



Example: Search and Propagation

Example: Search and Propagation

Choose $x_1 = 3$ and propagate constraints:



Example: Search and Propagation

Example: Search and Propagation

Choose $x_1 = 3$ and propagate constraints:

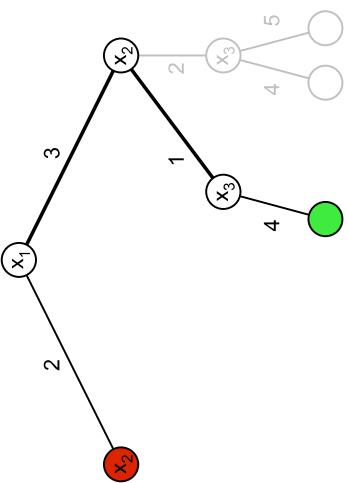


Example: Search and Propagation

Example: Search and Propagation

Choose $x_2 = 1$ and propagate constraints:

$$\begin{array}{l} x_1 \in \{2, 3\}, x_2 \in \{1, 2, 3\}, x_3 \in \{4, 5\} \\ x_1 > x_2 \\ x_1 + x_2 = x_3 \end{array}$$



Arc consistency

We will continue to consider only **binary CSP**, where every constraint is a binary relation

- general (n-ary) CSP can be converted to binary CSP
- binary CSP can be represented by **digraph G**

- nodes are variables
- if there is a constraint involving x_i, x_j , then the nodes x_i, x_j are interconnected by oriented arcs (x_i, x_j) and (x_j, x_i)

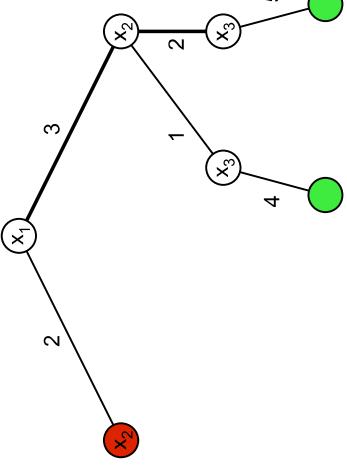
Arc consistency is an essential method for propagation.

- Arc (x_i, x_j) is **Arc Consistent (AC)** iff for each value $a \in D_i$ there exists a value $b \in D_j$ such that the assignment $x_i = a, x_j = b$ meets all binary constraints for the variables x_i, x_j .
- A **CSP** is **arc consistent** if all arc are arc consistent.
- Note that AC is **oriented** - the consistency of arc (x_i, x_j) does not guarantee the consistency of arc (x_j, x_i) .

There are other local consistencies (path consistency, k-consistency, singleton arc consistency,...). Some of them are stronger, some are weaker.

Choose $x_2 = 2$ and propagate constraints:

$$\begin{array}{l} x_1 \in \{2, 3\}, x_2 \in \{1, 2\}, x_3 \in \{4, 5\} \\ x_1 > x_2 \\ x_1 + x_2 = x_3 \end{array}$$



REVISE Procedure

From domain D_i delete any value a , which is not consistent with domain D_j .

procedure REVISE

Input: Domain D_i to be revised. Domain D_j . Set of constraints C .

Output: Binary variable *deleted* indicating deletion of some value from D_i . Revised domain D_i .

```
deleted := 0;
for a ∈ Di do
    if there is no b ∈ Dj ; xi = a, xj = b satisfies all constraints on xi, xj
    then
        Di := Di \ a;
        deleted := 1;
    end
end
```

Constraint Programming

May 24, 2017

Z. Hanzálek (CTU)

May 24, 2017

Constraint Programming

25 / 33

Proprietary:

- SICStus Prolog
- IBM CP, CP Optimizer (C++)
- IBM OPL Studio (OPL)
- Koalog (Java)

Open source:

- ECLiPSe (Prolog)
- Gecode (C++)
- Choco Solver (Java)
- Python constraints

- Roman Barták.
Programování s omezujícími podmínkami.
<http://kti.mff.cuni.cz/~bartak/podminky/index.html>,
2010.
- Rina Dechter.
Constraint Processing.
Morgan Kaufmann, 2003.
- Christian Schulte.
Constraint programming.
<http://www.informatik.uni-osnabrueck.de/knust/class/>,
2010.