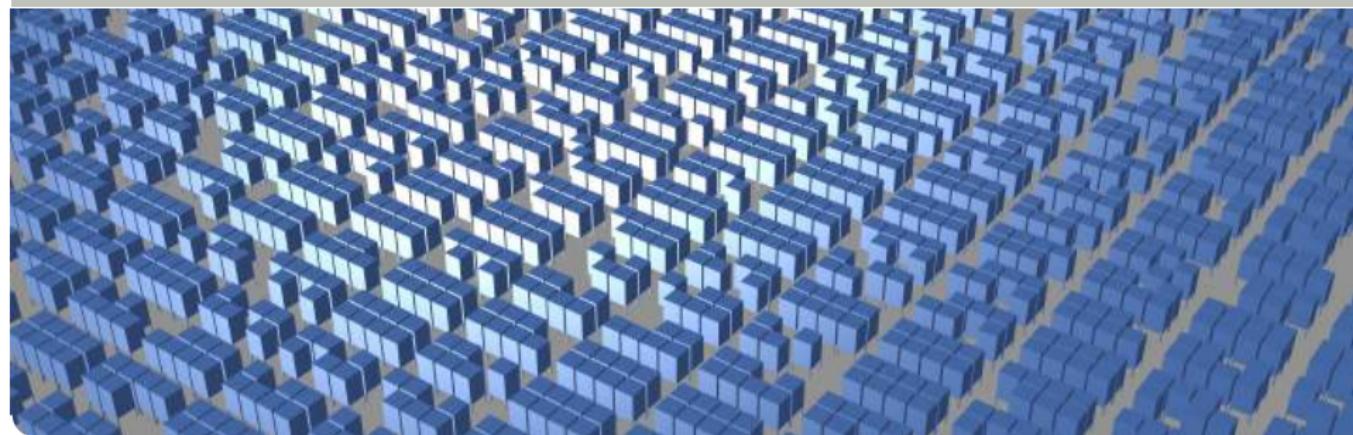


# Analysing Networks with OR-Methods

Part 0 – Organization

Lin Xie | 06.04.2021

PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH



# Course goals

## You will learn...

- how to solve difficult, real-world optimization problems,
- how to use and advance state-of-the-art operations research techniques to solve a wide range of network problems.

# Course structure

## **Course components:**

- Network Modelling and Algorithms
- Metaheuristics

## **Meeting times:**

- Tuesdays 10:15 – 11:45

# Grading

- Homework (40%)
  - Group assignment (2–3 persons per group)
  - Two assignments; Mathematical Modelling.
- Project (60%)
  - Group assignment (2–3 persons per group)
  - Implementation of metaheuristics for a given optimization problem

# Grading

- Homework (40%)
  - Group assignment (2–3 persons per group)
  - Two assignments; Mathematical Modelling.
- Project (60%)
  - Group assignment (2–3 persons per group)
  - Implementation of metaheuristics for a given optimization problem

## Bonus points

- 1 bonus point for presenting one task of your homework (at most 2 bonus points)
- 1 bonus point for attendance at the guest lecture
- 2 bonus points for the best project performance on the leaderboard assignment

# Guest lecture

- 18.05.21      **Dr. Tim Schöneberg**  
Volkswagen AG

# Dates and deadlines

## Homework dates:

Task	Assigned	Due
Homework 1	27.04.21	18.05.21
Homework 2	18.05.21	08.06.21

## Project dates:

- Project assignment: 22.06.21
- Leaderboard deadline: 01.08.21
- Project due: 15.08.21 (with project documentation)

# Course schedule: Network modelling and algorithms

Date	Topic
06.04.21	Organization; Introduction to OR
13.04.21	(Re-)introduction to mathematical modelling
20.04.21	Transshipment, TSP
27.04.21	Vehicle routing
04.05.21	Max flow / min cut
11.05.21	Multi commodity flow
18.05.21	Guest lecture (Dr. Tim Schöneberg)
25.05.21	Resource constrained shortest path problems
01.06.21	Solution of 1st homework

# Course schedule: Metaheuristics

Date	Topic
08.06.21	Solving problems with heuristics, Local search
15.06.21	LNS / VNS, Genetic Algorithms
22.06.21	Construction Heuristics / ACO
29.06.21	Solution of 2nd homework
06.07.21	Question time for the project

## Course “rules”

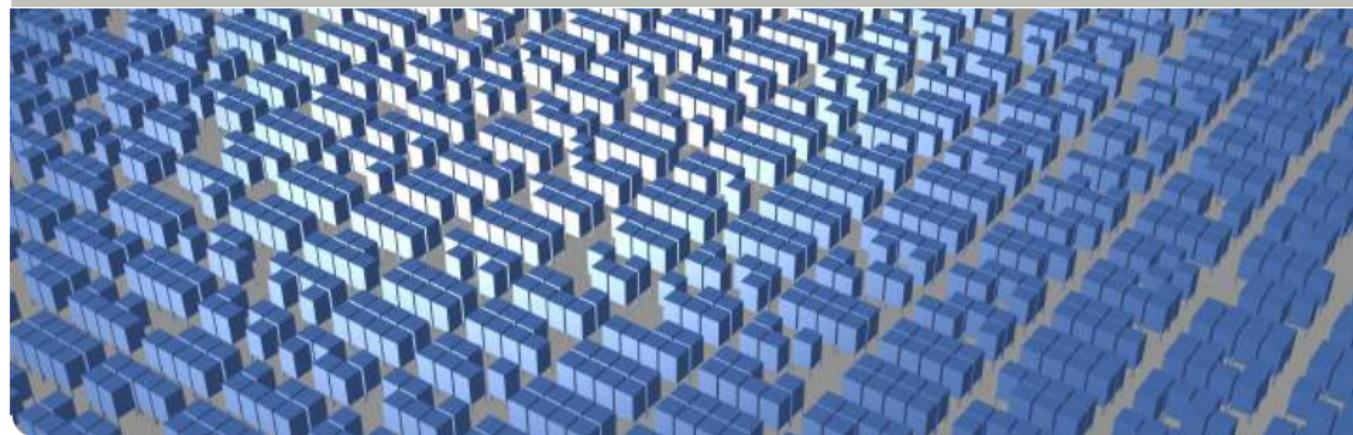
- Please turn on your camera, if you can.
- Do ask questions anytime you have any. You can turn on your audio. Please keep your audio off after that. Or you can use chat in Zoom.
- Feedback of this course: <https://pingo.coactum.de/>  
Access number: 923946

# Analysing Networks with OR-Methods

Part 1 – Introduction to OR

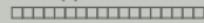
Lin Xie | 06.04.2021

PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH

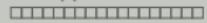


# Outline

- What is OR?
- What types of problems does OR solve?
- How do we solve problems with OR techniques?
- What problems will we focus on?



# What is OR?



# What is operations research?

Operations Research (OR) . . . is a discipline that deals with the application of advanced analytical methods to help make better decisions. – INFORMS



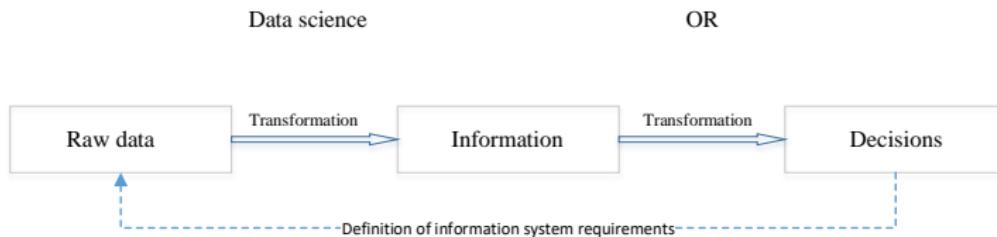
# What is operations research?

Operations Research (OR) . . . is a discipline that deals with the application of advanced analytical methods to help make better decisions. – INFORMS

## **Neighboring disciplines sometimes used as synonyms for OR:**

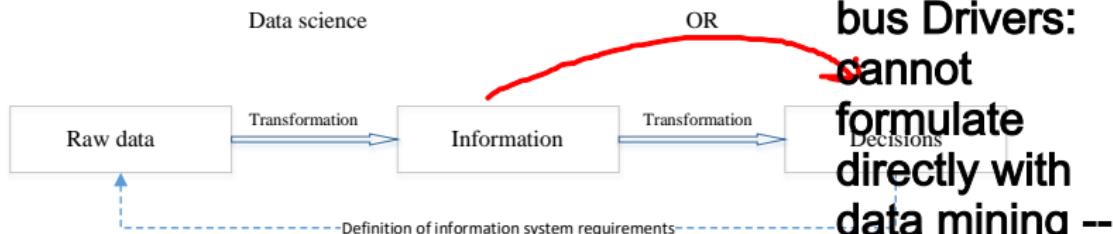
- Management science
- Operations management
- Business analytics
- Decision science

# Relation between OR and Data Science



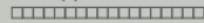
- **Data Science** tries to extract knowledge from the available data and uses mostly *data-centric techniques*, e.g. time-series analysis, regression models, machine learning methods and deep neural networks.

# Relation between OR and Data Science



**optimal schedule for bus Drivers:  
cannot formulate directly with data mining --> Need to optimize**

- **Data Science** tries to extract knowledge from the available data and uses mostly *data-centric techniques*, e.g. time-series analysis, regression models, machine learning methods and deep neural networks.
- **OR** uses the information in order to make decisions. *Problem-centric methods* are mostly used, e.g. optimization, simulation, queuing theory and Markov decision process.



# OR: A short history

- OR precursors stretching back before the second world war.
- British OR analysts found better ways to wage war.
- (Also researched in USSR by Kantorovich in parallel).



# OR: A short history

- OR precursors stretching back before the second world war.
- British OR analysts found better ways to wage war.
- (Also researched in USSR by Kantorovich in parallel).
- After the war, OR is applied widely in economic areas.

See: "An Annotated Timeline of Operations Research: An Informal History" by Saul I. Gass and Arjang A. Assad



# The OR process



Find a problem



Formalize the problem

Refine model

$$\begin{aligned} \min & x + 3y + 2z - a \\ \text{subject to } & 10x + 5y \leq 10 \\ & 42y - 1.2z + a \geq x \\ & x, y, z \geq 0 \end{aligned}$$

$$\begin{aligned} \min & x + 3y + 2z \\ \text{subject to } & 2x + 5y \leq 10 \\ & 4y - 1.2z \geq x \\ & x, y, z \geq 0 \end{aligned}$$

Construct a model

Carry out decisions



Implement a solution

Solve the model



What types of problems does OR solve?

# Problem class for this course

## ■ Optimization

“What is the best possible solution to this problem?”



# Problem class for this course

- Optimization

“What is the best possible solution to this problem?”

## Examples:

- What is the lowest cost route to deliver a set of packages with a fleet of vehicles?



# Problem class for this course

- Optimization

“What is the best possible solution to this problem?”

## Examples:

- What is the lowest cost route to deliver a set of packages with a fleet of vehicles?
- Where should I build new factories such that my profit is maximized?

# What can we solve with OR?

**In this course:** Mainly NP-Hard (or harder) problems.

NP-Hard (and NP-complete) problems exhibit larger than polynomial growth in runtime as the input size increases.

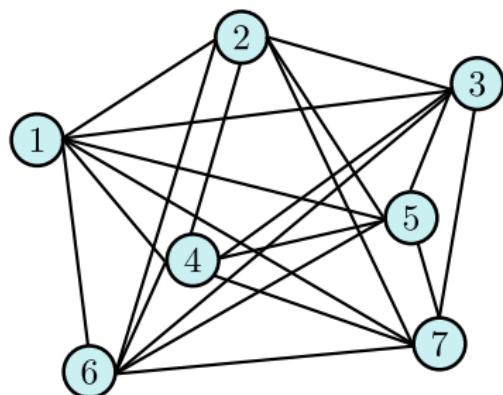
**Example problem:** The traveling salesman problem (TSP)

# TSP example

## Traveling Salesman Problem

**Given:** A graph  $G = (V, E)$  with symmetric edge costs  $c_e$ .

**Objective:** Find the minimum cost path traversing every node exactly once.



Graph

	1	2	3	4	5	6	7
1	0	2.2	5.0	2.8	4.1	5.0	8.5
2	2.2	0	3.0	3.0	2.8	6.0	8.0
3	5.0	3.0	0	4.2	2.2	7.2	7.0
4	2.8	3.0	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5.0	5.4
6	5.0	6.0	7.2	3.1	5.0	0	5.1
7	8.5	8.0	7.0	5.7	5.4	5.1	0

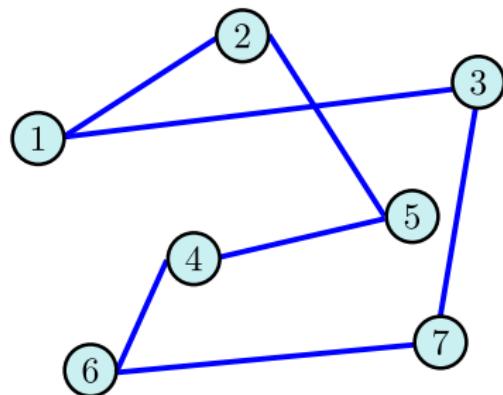
Edge costs

# TSP example

## Traveling Salesman Problem

**Given:** A graph  $G = (V, E)$  with symmetric edge costs  $c_e$ .

**Objective:** Find the minimum cost path traversing every node exactly once.



A solution (Cost: 27.4)

	1	2	3	4	5	6	7
1	0	2.2	5.0	2.8	4.1	5.0	8.5
2	2.2	0	3.0	3.0	2.8	6.0	8.0
3	5.0	3.0	0	4.2	2.2	7.2	7.0
4	2.8	3.0	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5.0	5.4
6	5.0	6.0	7.2	3.1	5.0	0	5.1
7	8.5	8.0	7.0	5.7	5.4	5.1	0

Edge costs

# The growth of P and NP problems

number nodes	Shortest path $O(n^2)$	Traveling salesman $O(n!)$
	$n$	$ S $
5	25	120
10	100	3,628,800
15	225	$1.3 \times 10^{12}$
20	400	$2.4 \times 10^{18}$
25	625	$1.5 \times 10^{25}$
30	900	$2.6 \times 10^{32}$

**total solutions**



# The growth of P and NP problems

	Shortest path $O(n^2)$	Traveling salesman $O(n!)$
$n$	$ S $	$ S $
5	25	120
10	100	3,628,800
15	225	$1.3 \times 10^{12}$
20	400	$2.4 \times 10^{18}$
25	625	$1.5 \times 10^{25}$
30	900	$2.6 \times 10^{32}$

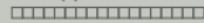
Given a computer that can process 200,000 solutions per second,  
you would need over 42,000,000,000,000,000 years to solve an  
 $n = 30$  traveling salesman problem!



# The growth of P and NP problems

	Shortest path $O(n^2)$	Traveling salesman $O(n!)$
$n$	$ S $	$ S $
5	25	120
10	100	3,628,800
15	225	$1.3 \times 10^{12}$
20	400	$2.4 \times 10^{18}$
25	625	$1.5 \times 10^{25}$
30	900	$2.6 \times 10^{32}$

**Despite the size of these problems, we have techniques to solve them!**



## How do we solve problems with OR techniques?

# Solving problems

The two classes of methods for solving (most) OR problems:

- Exact algorithms

- Linear programming
- Mixed-integer linear programming
- Constraint programming
- Non-linear methods
- Network programming

- (Meta)Heuristic algorithms

- Genetic algorithms
- Ant colony optimization
- Simulated annealing
- Construction algorithms
- Domain-specific heuristics

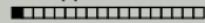


# Solving problems

The two classes of methods for solving (most) OR problems:

- Exact algorithms
  - Linear programming
  - Mixed-integer linear programming
  - Constraint programming
  - Non-linear methods
  - **Network programming**
- (Meta)Heuristic algorithms
  - **Genetic algorithms**
  - **Ant colony optimization**
  - **Simulated annealing**
  - **Construction algorithms**
  - **Domain-specific heuristics**

This course uses the methods in **bold** (among others).



## OR applications

## Application areas

- Decision support with quantitative methods in:
    - Production planning
    - Supply Chain Management
    - Location planning
    - Network design
    - Crew scheduling/rostering
    - Warehouses
    - Logistics, transportation, traffic
    - Project planning, portfolio selection
    - Financial planning, marketing
    - Also operations, not restricted to planning

## Operations research examples

- Many production planning problems can be defined as optimization problems (e.g. product mix).
  - **Different resources**, such as production machines, raw materials and staffs, have to be optimally distributed.
  - Complex sequences and limited time windows often have to be considered.



## Operations research examples

- A power grid is usually supplied by several energy sources, which have different capacities and cost functions.
  - The usage varies between different times of the day (because of industry, television viewing, etc.), and the production has to be adjusted.
  - Production should be performed with minimum costs.
  - Another example of a problem is the regulation of tariffs, which are in balance with the production.



## Operations research examples

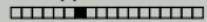
- Transportation and vehicle routing problems are formulated as optimization problems.
  - E.g. search for the shortest or minimum-cost transport routes (GPS navigation).



Current project 1 of X-OR Lab: vehicle and crew scheduling

- Optimum numbers of vehicles and drivers to cover the customers' needs.
  - Generate robust plans for vehicles and drivers by using historical data.
  - While planning reserves of vehicles and drivers.





# Current project 1 of X-OR Lab: vehicle and crew scheduling

- Combining two neural networks with a genetic algorithm and a Kalman filter to predict real-world bus travel time (cooperation: Hochbahn AG).
- The genetic algorithm is implemented to tune the neural network parameters.
- The parameter tuning can improve the accuracy by more than 30%.

# Current project 2 of X-OR Lab: AI in sports

- Developing decision support systems by using data analysis for the following sport types (goal: find the best strategies to win games)
  - Team sport (football, handball, basketball)



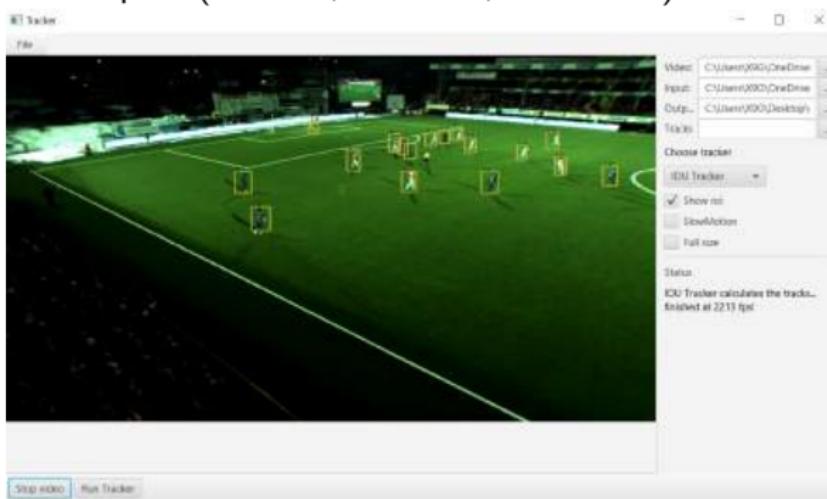
- Winter Olympic sport (curling, bobsleigh, luge, skeleton, ski jumping)





# Current project 2 of X-OR Lab: AI in sports

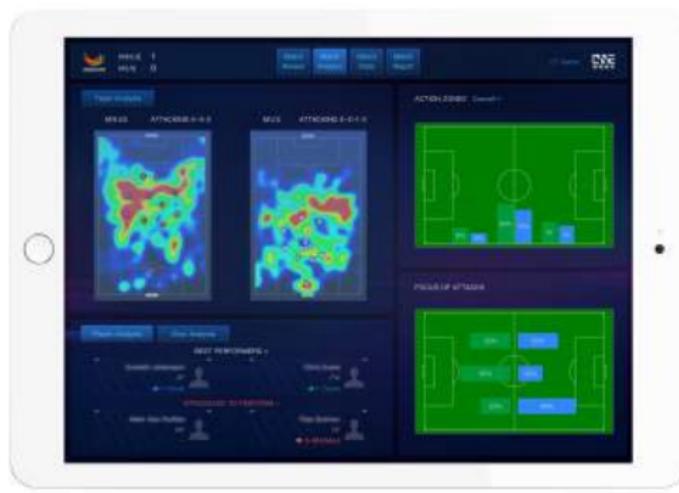
- Developing decision support systems by using data analysis for the following sport types (goal: find the best strategies to win games)
  - Team sport (football, handball, basketball)





# Current project 2 of X-OR Lab: AI in sports

- Developing decision support systems by using data analysis for the following sport types (goal: find the best strategies to win games)
  - Team sport (football, handball, basketball)



# Current project 3 of X-OR Lab: RMFS

- Robotic mobile fulfillment system (RMFS)
  - Uses robots in warehouses.
  - Instead of sending pickers to the storage area to fetch the goods, the shelves from the storage area are moved to the pickers in picking stations.
  - A complex system consists of strategic, operative and real-time planning, e.g. path planning for robots.
  - A mathematical optimization technique analyzes all planning processes. It reduces errors of commission and increases resource efficiency.

# Current Project 3 of X-OR Lab: RMFS

## ■ Robotic mobile fulfillment system (RMFS)



AmazonRobotics



Swisslog



GreyOrange



Scallog



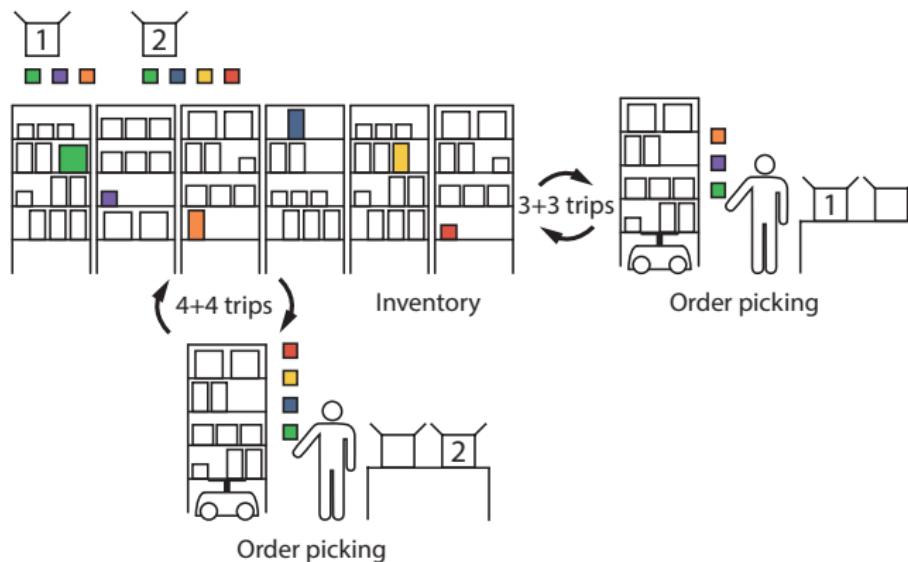
Hitachi



Alibaba Group

# Current Project 3 of X-OR Lab: RMFS

An example without considering optimization (worst case)



# Current Project 3 of X-OR Lab: RMFS

Same example by considering optimization and data analysis



# Looking for student assistants

- supporting the current projects in the area of data-driven analysis
- working hours per week between 9.5 and 19.5 hours
- persons we are looking for:
  - good programming skills
  - basic data analysis / machine learning skills
- interest in this job?
  - CV, grade list, bachelor thesis



What problems will we focus on?

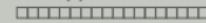
# Problems in this course

## ■ Network problems

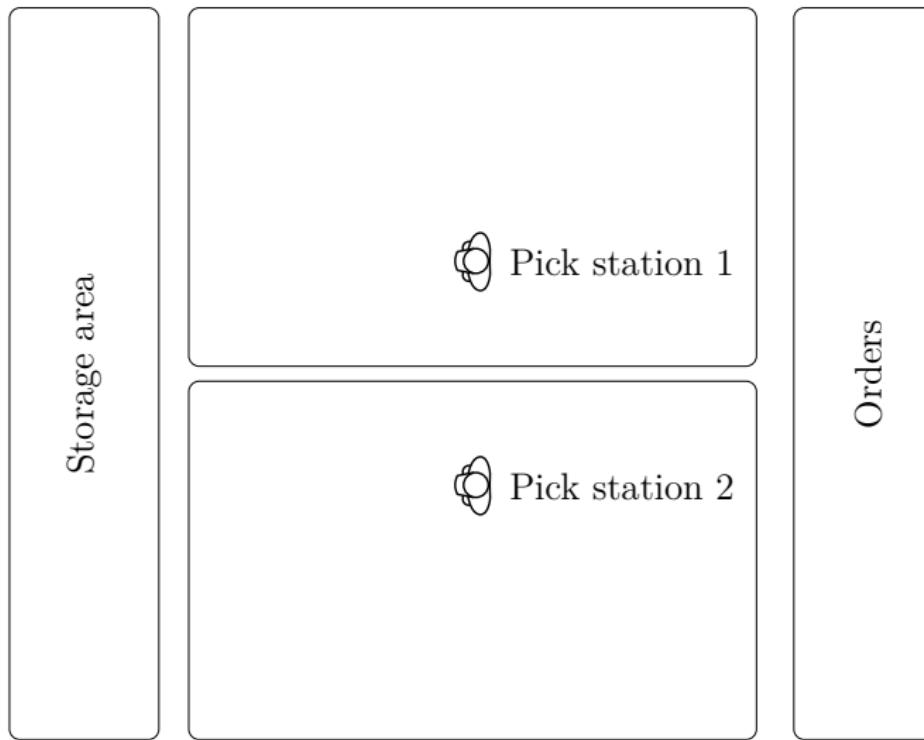
- Transport problem
- Traveling salesman problem and variants
- Routing “flow” in networks
- Logistics transportation

## ■ Miscellaneous problems

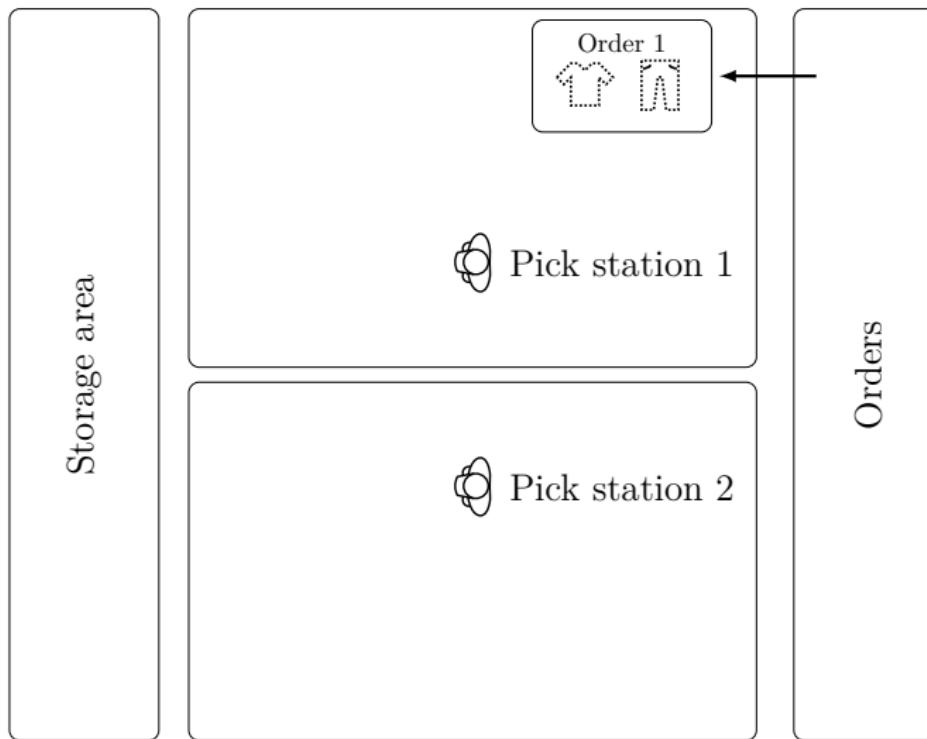
- Scheduling problems
- Assignment problems
- etc.



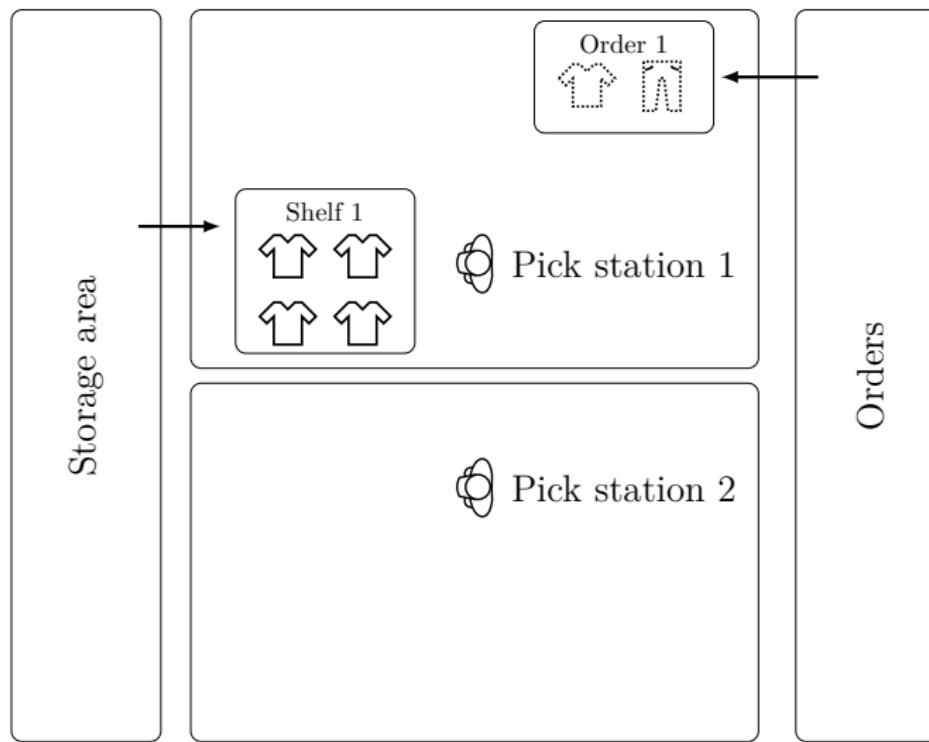
# Shelves and orders



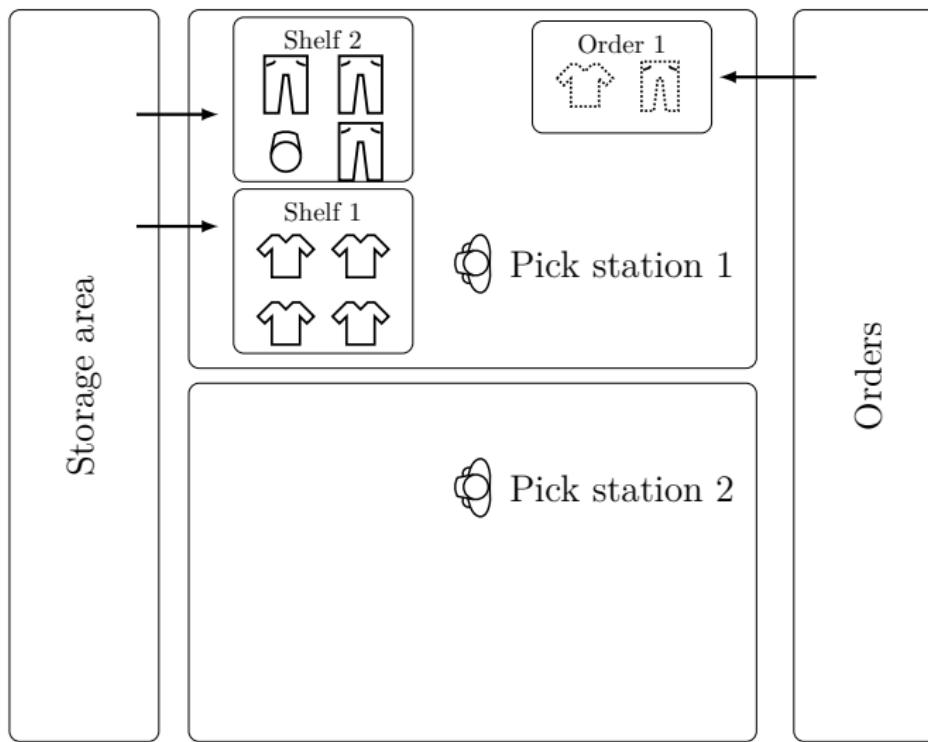
# Shelves and orders



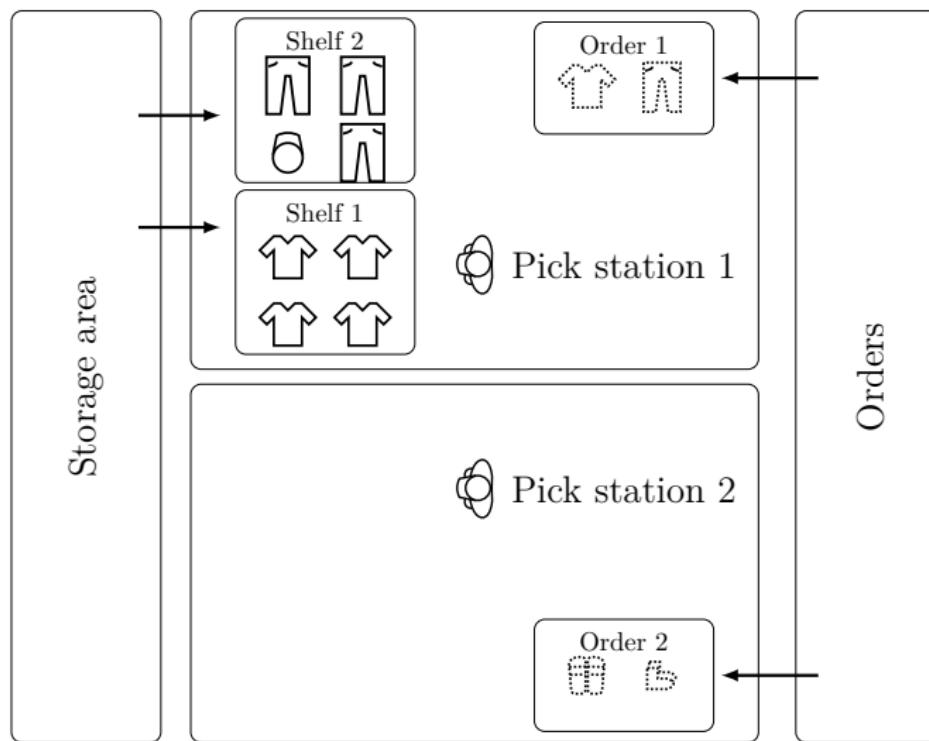
# Shelves and orders



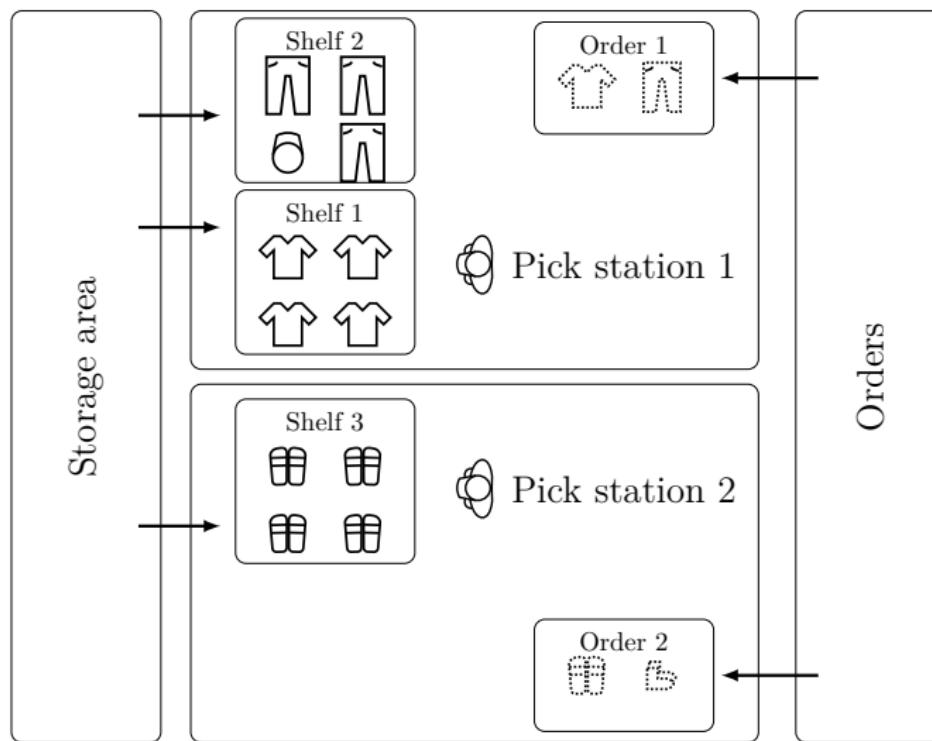
# Shelves and orders



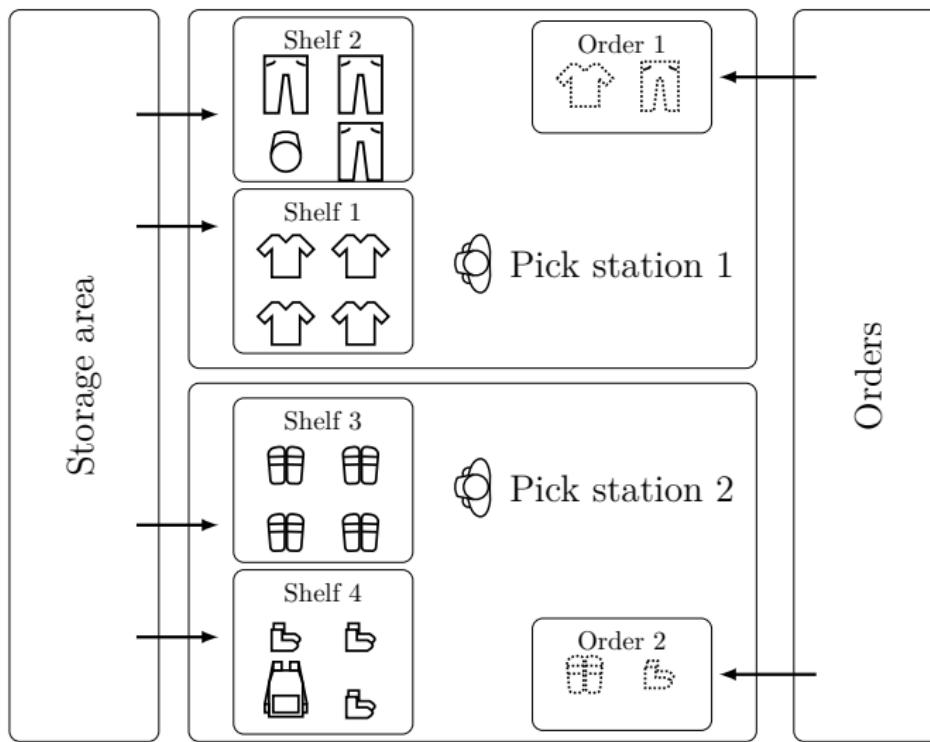
# Shelves and orders



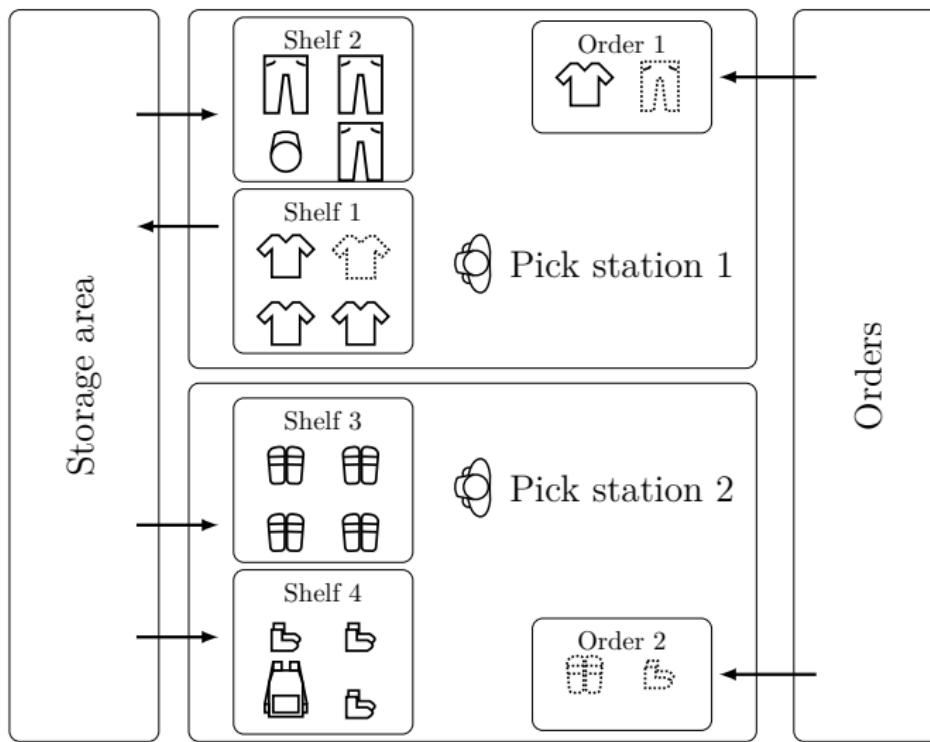
# Shelves and orders



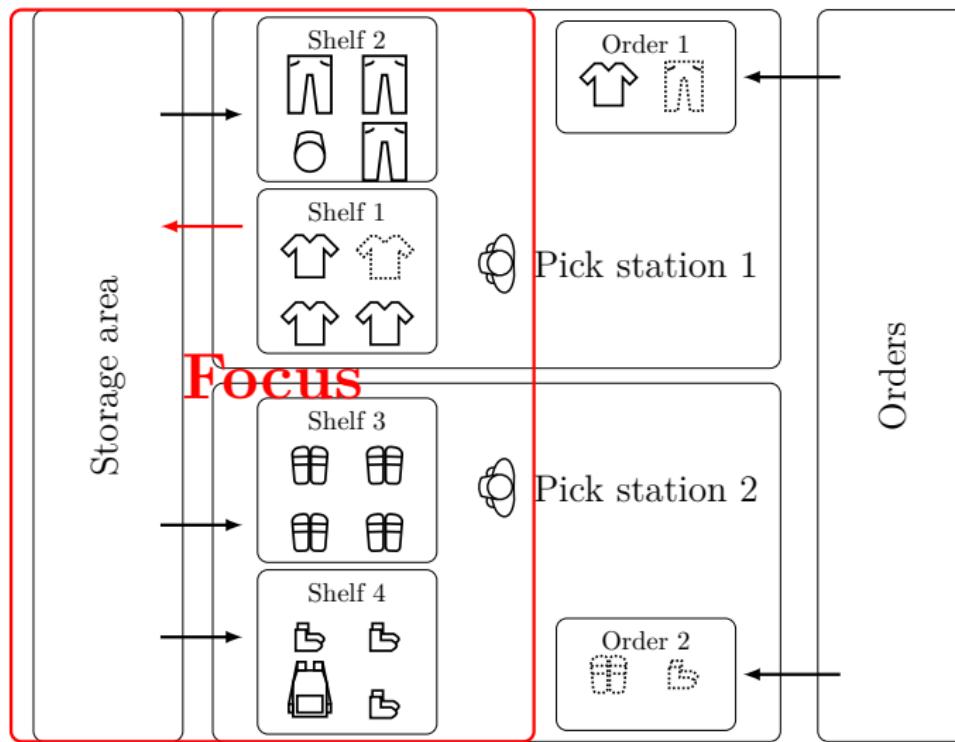
# Shelves and orders



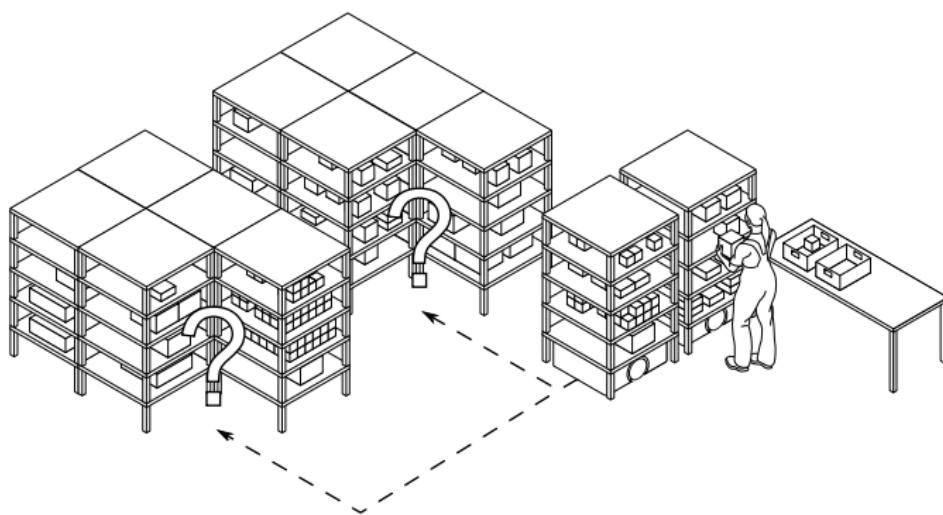
# Shelves and orders



# Shelves and orders



# Where to put the shelf?



# The pod repositioning problem

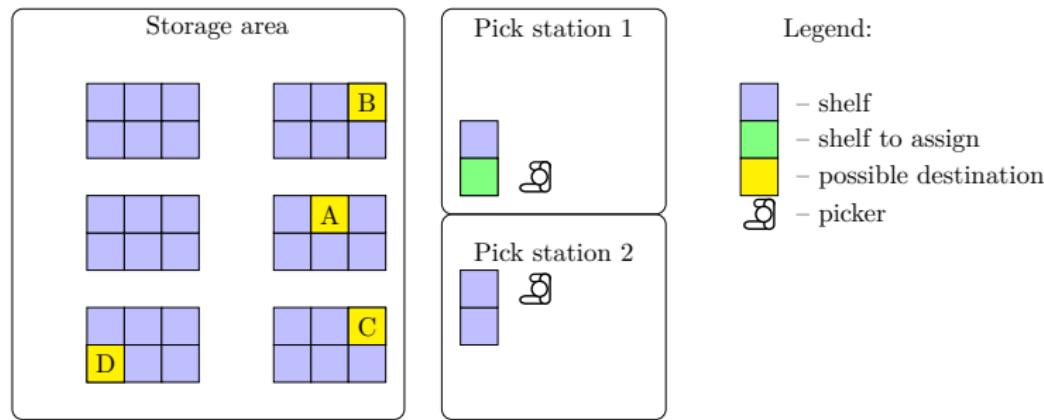
**Given:** A warehouse consisting of  $P$  storage places of shelves and we consider  $T$  time periods. Each time  $t$  a shelf leaves a picking station and which place  $p$  should be picked for storage (decision variable  $x_{pt} \in \{0, 1\}$ ).

**Goal:** Find the minimal moving distances of returning shelves.

## Constraints:

- 1 Assign all shelves.
- 2 Start from initial order in the storage.
- 3 Do not assign two shelves to one place at the same time.

# Try solving a problem



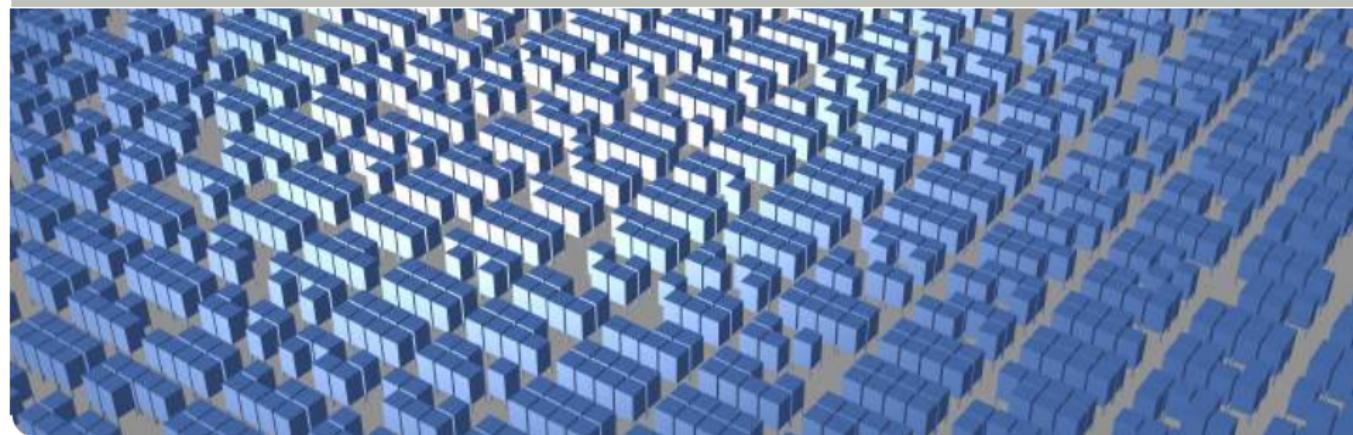
Sequence of place assignments

# Analysing network using OR-methods

Part 2 – LP modeling and general notation

Lin Xie | 13.04.2021

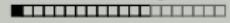
PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH



## 1 Linear modeling

## 2 General notation

## 3 An example



# Linear modeling

# Modeling

Example: Bike manufacturer

- We examine a bike manufacturer, which produces different types of bikes (Standard and Deluxe).
- The maximum number of bikes of each type is limited by the amount of available materials.
- The bikes are produced using the same machines. These machines have a limited total operating time.
- We look for the combination of bikes to produce that maximizes profit.

# Modeling

What is the problem?

- There is a decision situation.
- There are at least two decisions being made.
- Each partial decision influences the outcome.
- The effects of these decisions are not immediately visible.

# Modeling

What is the problem?

- There is a decision situation. **Which?**
- There are at least two decisions being made.
- Each partial decision influences the outcome.
- The effects of these decisions are not immediately visible.

# Modeling

What is the problem?

- There is a decision situation. **Which?**
- There are at least two decisions being made. **Which?**
- Each partial decision influences the outcome.
- The effects of these decisions are not immediately visible.

# Modeling

What is the problem?

- There is a decision situation. **Which?**
- There are at least two decisions being made. **Which?**
- Each partial decision influences the outcome. **Why?**
- The effects of these decisions are not immediately visible.

# Modeling

Example: Bike manufacturer

- Decide: How many products of the Standard and Deluxe types should be produced to maximize profit?
- Two decisions: Number of Standard and Deluxe types
- Different influence on the outcome: The Standard and Deluxe types have different production costs and profits
- Effect of partial decisions: A higher number of Deluxe bikes could result in a lower number of Standard bikes being produced.

# Modeling

Definition of an optimization model

An optimization model to support decision making consists of three elements:

## **1 Decision variables**

Relevant values which we can change

## **2 Objective function**

One expression which should be minimized or maximized (usually profit or costs)

## **3 Constraints**

Restrictions. Usually we cannot arbitrarily change the decision variables – we need to consider available resources, minimum order quantity etc.

# Modeling

Definition of an optimization model

An optimization model to support decision making consists of three elements:

## 1 Decision variables

Relevant values which we can change

## 2 Objective function

One expression which should be minimized or maximized  
(usually profit or costs)

## 3 Constraints

Restrictions. Usually we cannot arbitrarily change the decision variables – we need to consider available resources, minimum order quantity etc.

**Write down the model! → Values?**

# Modeling

Example: Bike manufacturer

- We examine a bike manufacturer, which produces two types of bikes: Standard and Deluxe. The profits for each type are €90 and €120 respectively.
- The maximum numbers produced for each type are 700 and 400 units respectively.
- The maximum number of bikes to be produced is limited to 800.
- Production of the Deluxe type takes 12 minutes, which is double the time of the Standard type. Every day, 100 operating hours of the machines are available.

# Modeling

Example: Bike manufacturer

## 1 Decision variables

$x_1$  = Number of Deluxe type to be manufactured

$x_2$  = Number of Standard type to be manufactured

## 2 Objective function

$$\max z = 120x_1 + 90x_2$$

$z$  describes profit

## 3 Constraints

$$x_1 + x_2 \leq 800 \quad \text{Production maximum for both types: 800}$$

Constraints for time constraints ?

$$x_1 \geq 0, x_2 \geq 0 \quad \text{No negative production}$$

# Modeling

Example: Bike manufacturer – model

Mathematical model:

$\max z = 120x_1 + 90x_2$       Objective function (Profit maximization)

subject to (s.t.)

$x_1 + x_2 \leq 800$       Constraint (maximum number produced)

$x_1 \leq 400$       Constraint (maximum number of Deluxe)

$x_2 \leq 700$       Constraint (maximum number of Standard)

$12x_1 + 6x_2 \leq 6000$       Constraint (time)

$x_1, x_2 \geq 0$       Constraint (non-negativity)

# Modeling

Example: Bike manufacturer – model

Mathematical model:

$\max z = 120x_1 + 90x_2$       Objective function (Profit maximization)

subject to (s.t.)

$x_1 + x_2 \leq 800$       Constraint (maximum number produced)

$x_1 \leq 400$       Constraint (maximum number of Deluxe)

$x_2 \leq 700$       Constraint (maximum number of Standard)

$12x_1 + 6x_2 \leq 6000$       Constraint (time)

$x_1, x_2 \geq 0$       Constraint (non-negativity)

## Linear models

Linear objective function:

$$z = \sum_{i \in I} (c_i \cdot x_i)$$

Linear constraints:

$$\sum_{i \in I} (a_{ki} \cdot x_i) \stackrel{\geq}{=} b_k \quad \forall k \in K$$

$$x_i \in \mathbb{R}^+ \quad \forall i \in I$$

# Linear models

The mathematical model we have just made is *linear*.

# Linear models

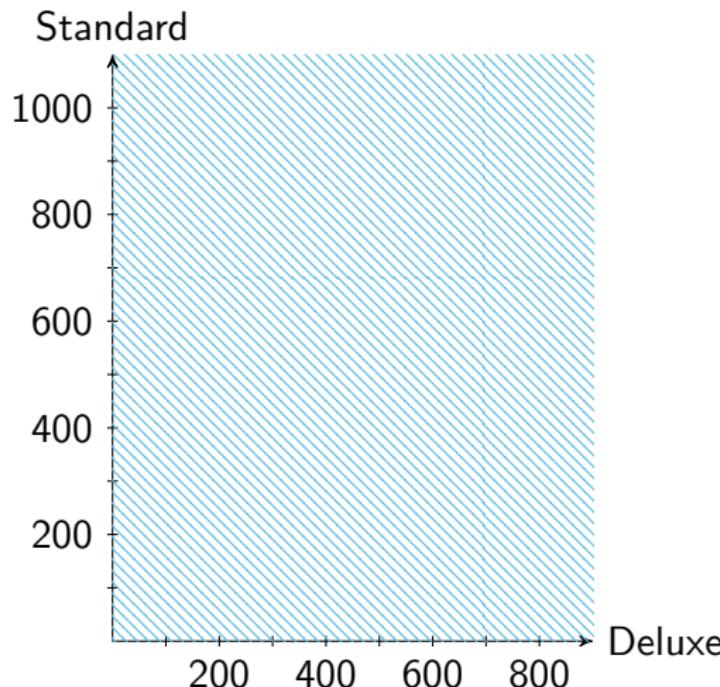
The mathematical model we have just made is *linear*.

## What this means

- The domains of the variables are continuous.
- The objective and constraints are *linear combinations* of variables.
- It is solvable in polynomial time.

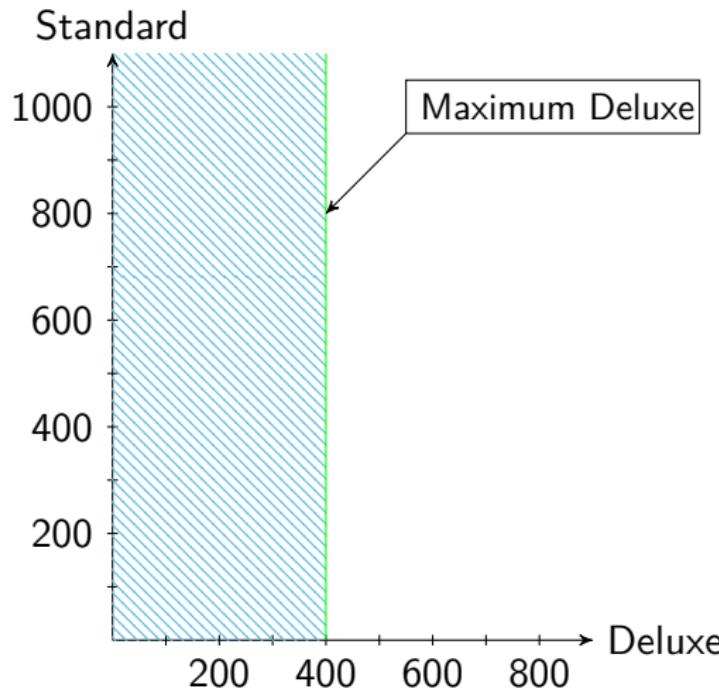
# The graphical representation of LPs

2D problem



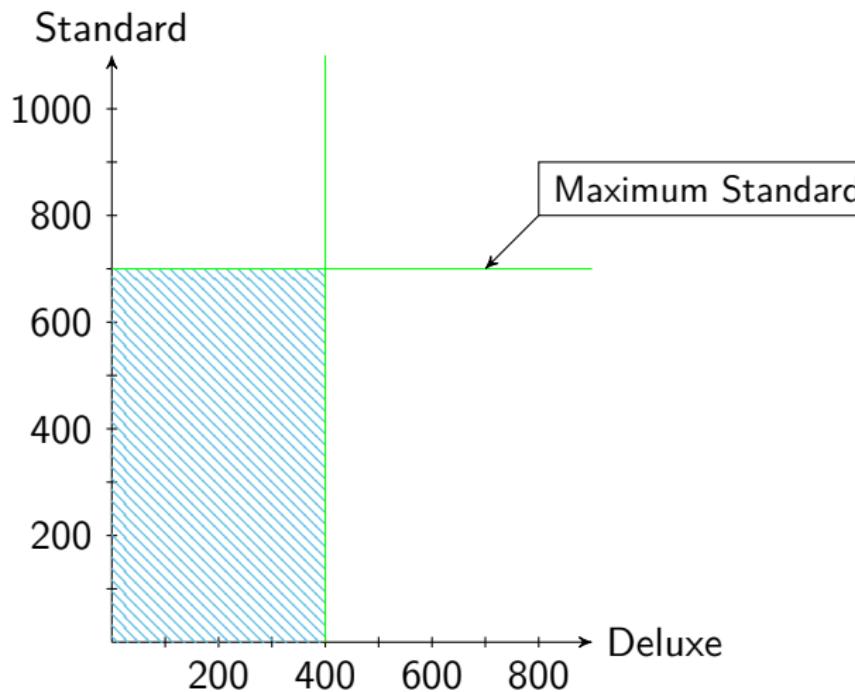
# The graphical representation of LPs

2D problem: The solution space is limited by a constraint



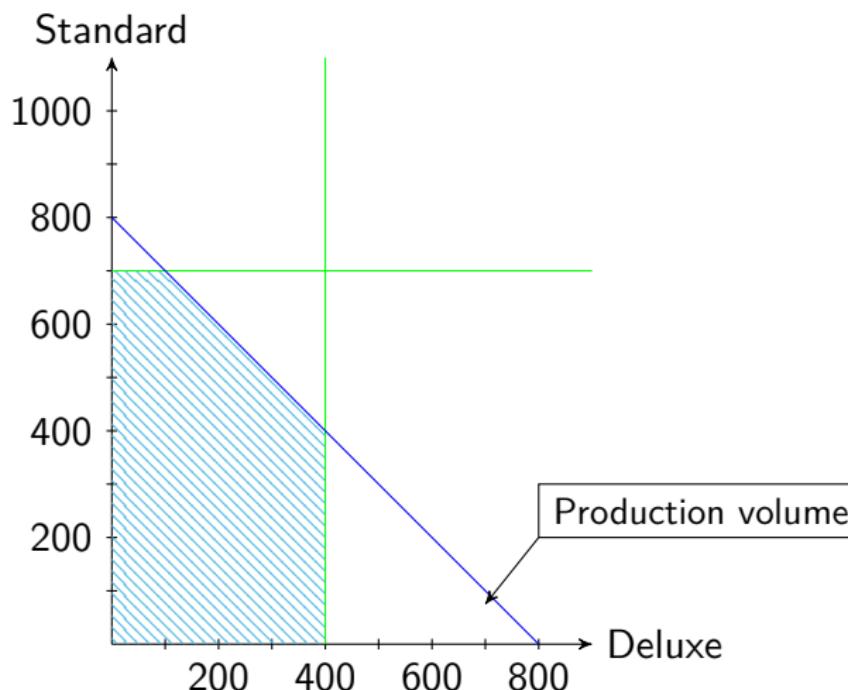
# The graphical representation of LPs

2D problem: The solution space with constraints



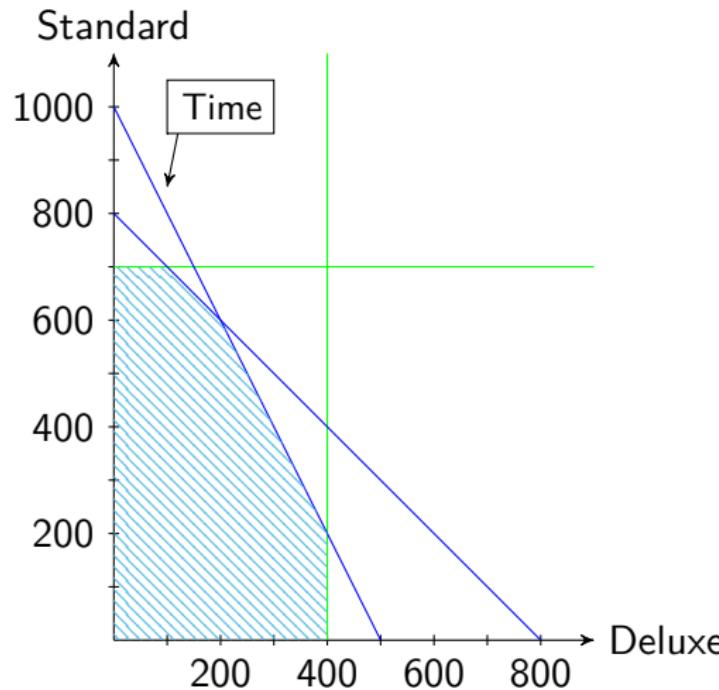
# The graphical representation of LPs

2D problem: The solution space with constraints



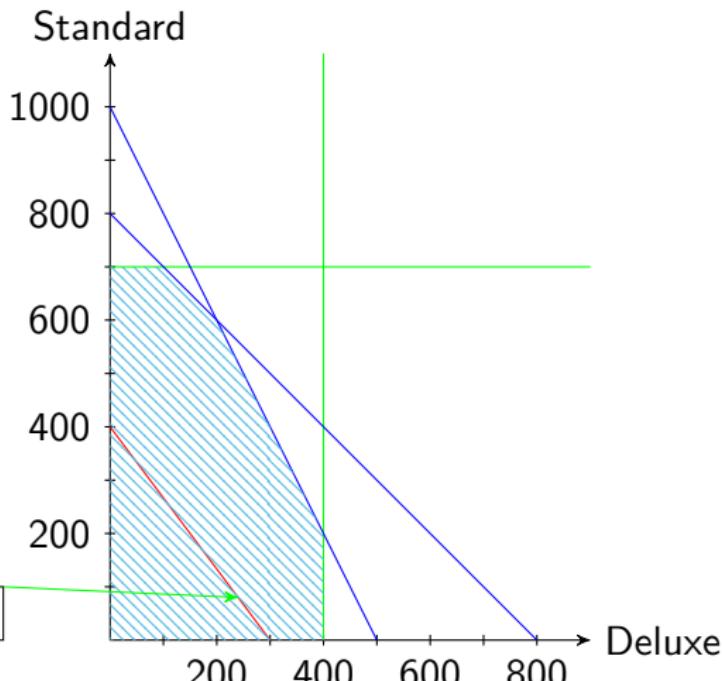
# The graphical representation of LPs

2D problem: The solution space with constraints



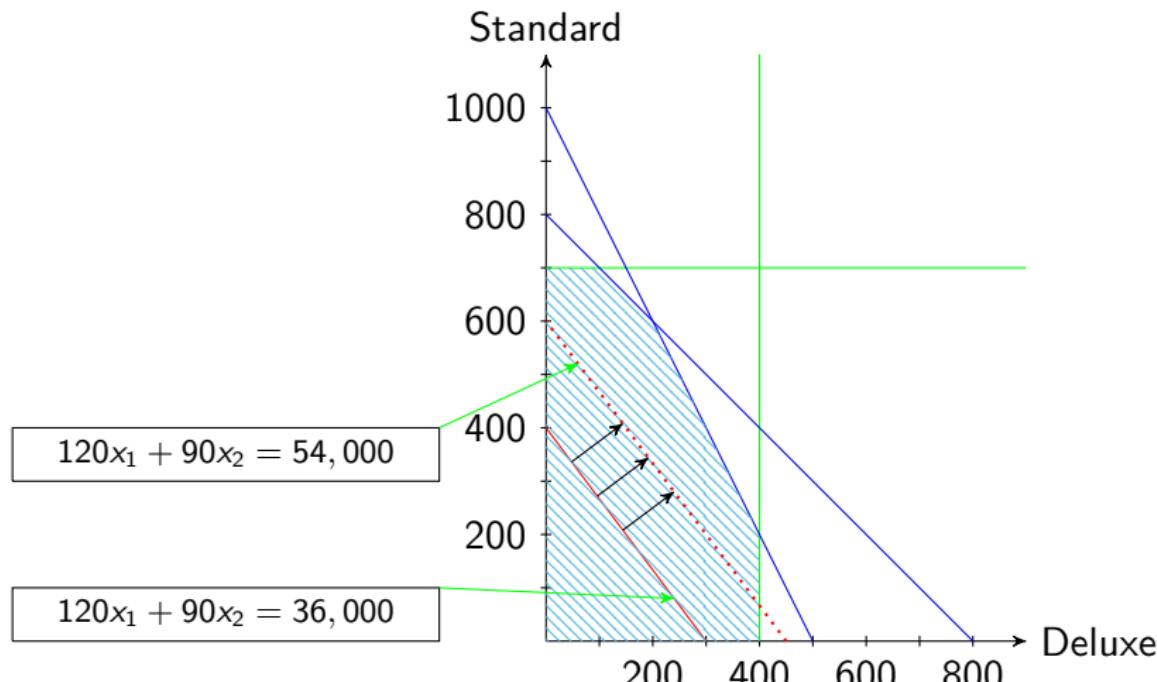
# The graphical representation of LPs

2D problem: The solution space with objective function



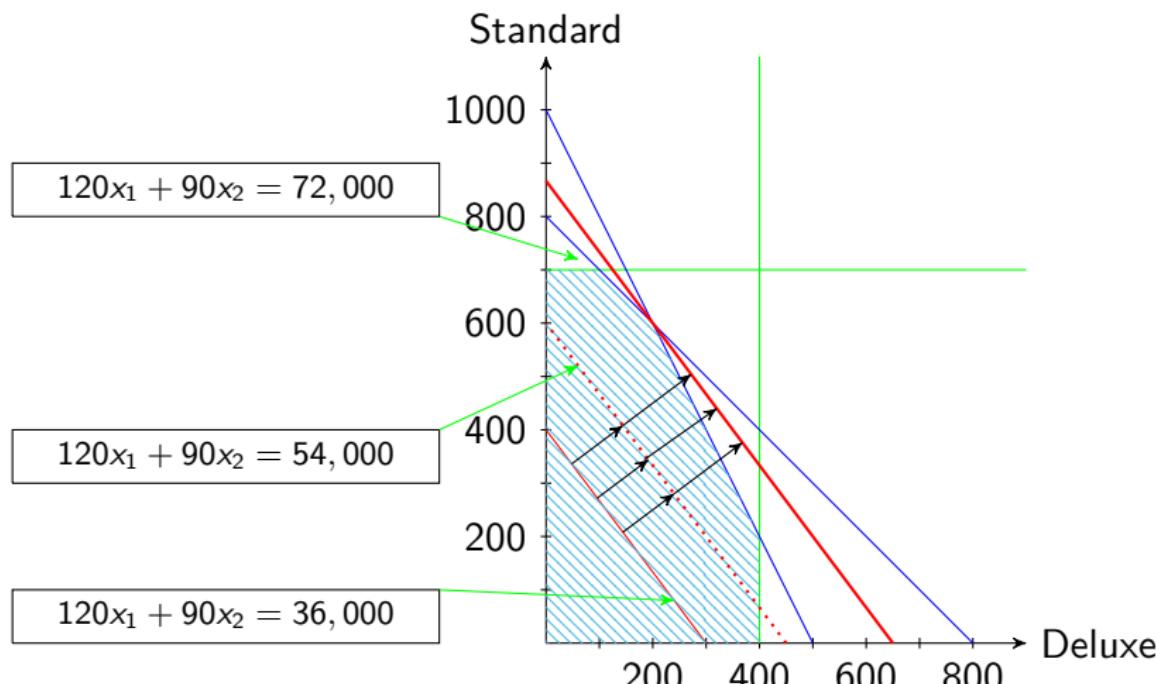
# The graphical representation of LPs

2D problem: The solution space with objective function



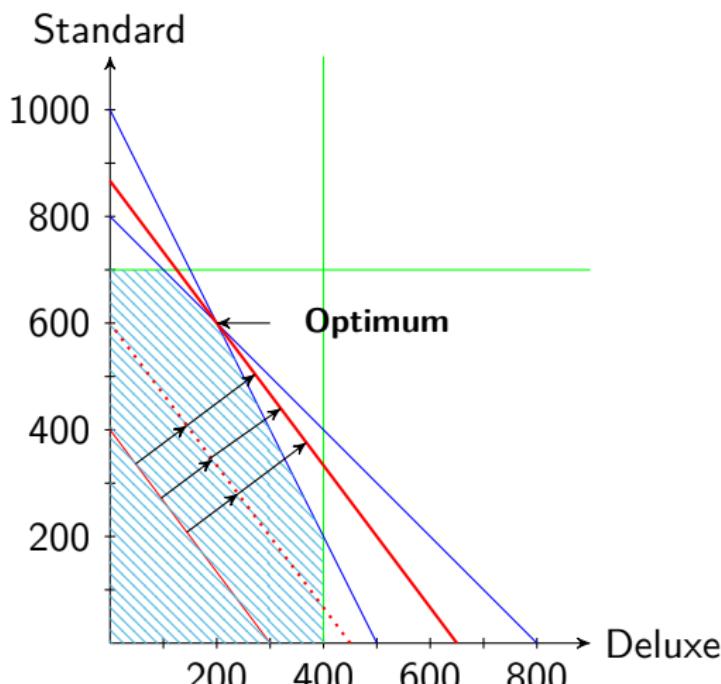
# The graphical representation of LPs

2D problem: The solution space with objective function



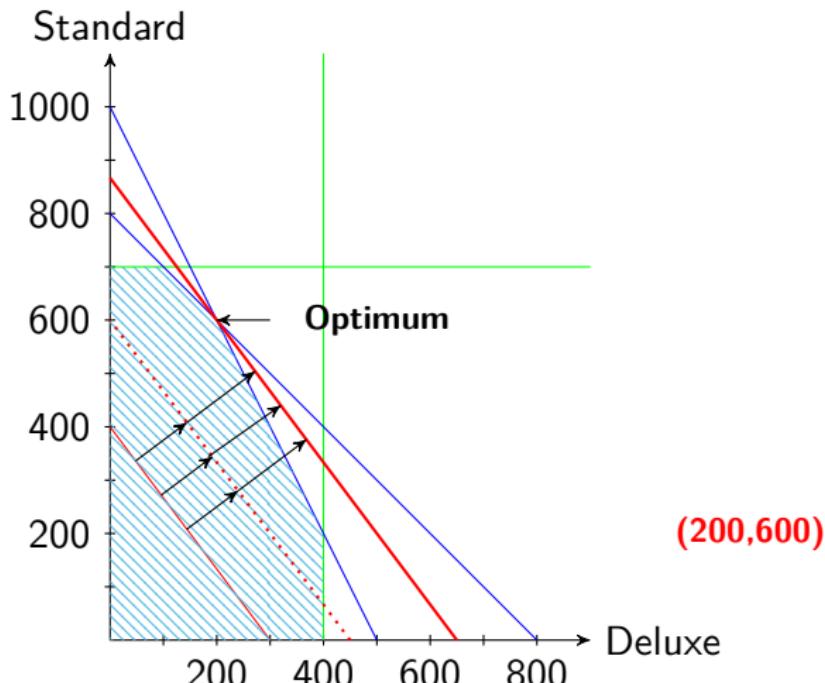
# The graphical representation of LPs

2D problem: The solution space with objective function

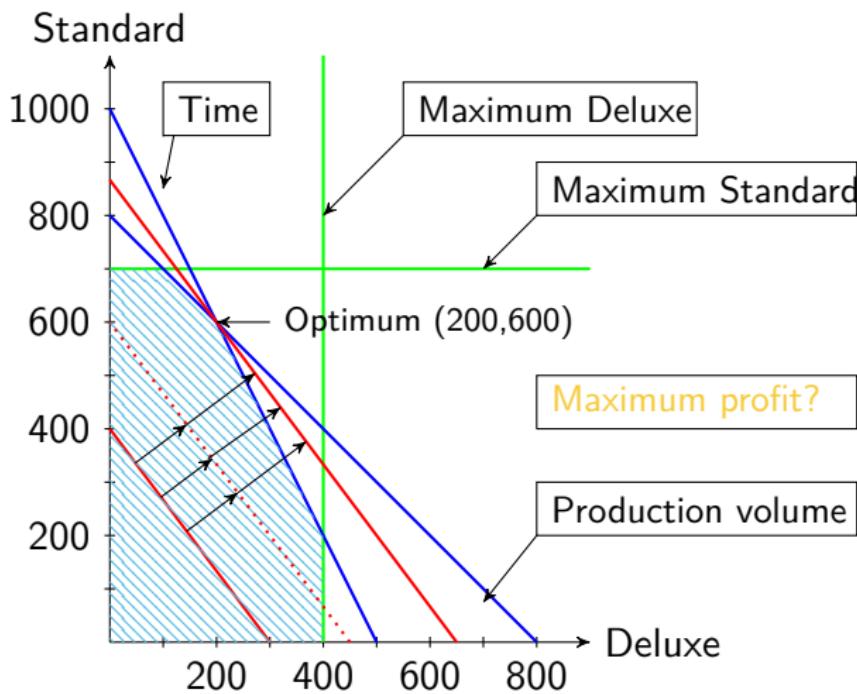


# The graphical representation of LPs

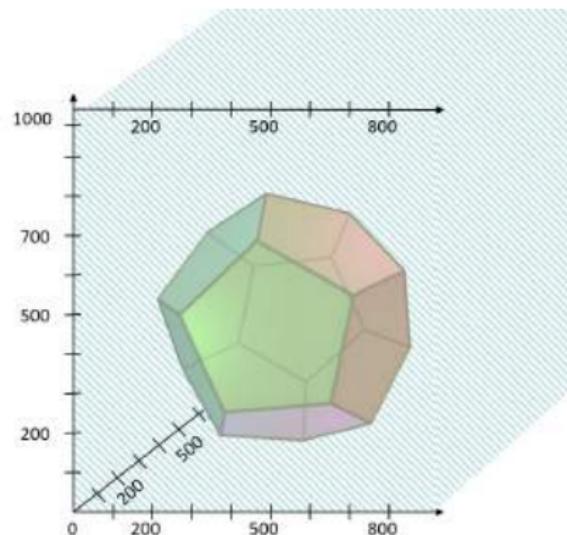
2D problem: The solution space with objective function



# Example: Bike manufacturer – the graphical solution



## 3D problem: the solution space



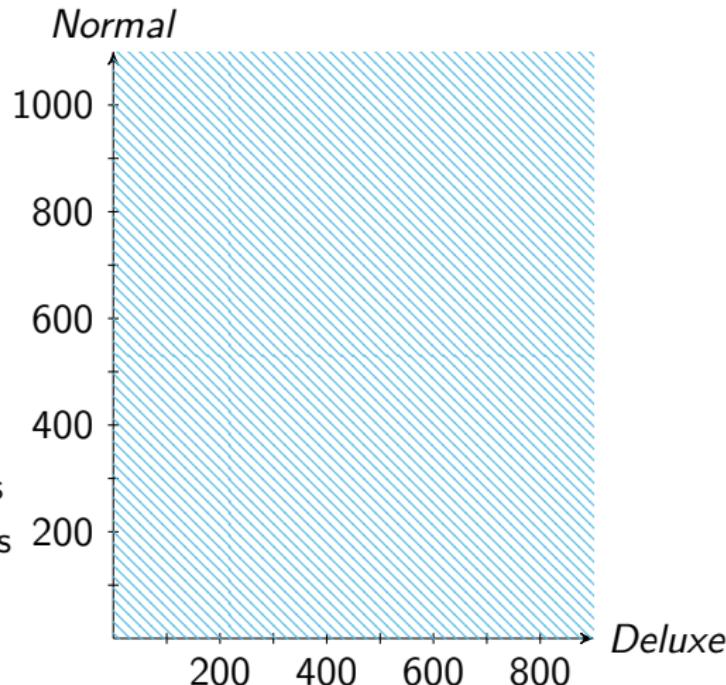
# Infeasible solution

We get an infeasible solution, if the solution space is empty.

Solution space

**Minimum** restrictions

**Maximum** restrictions



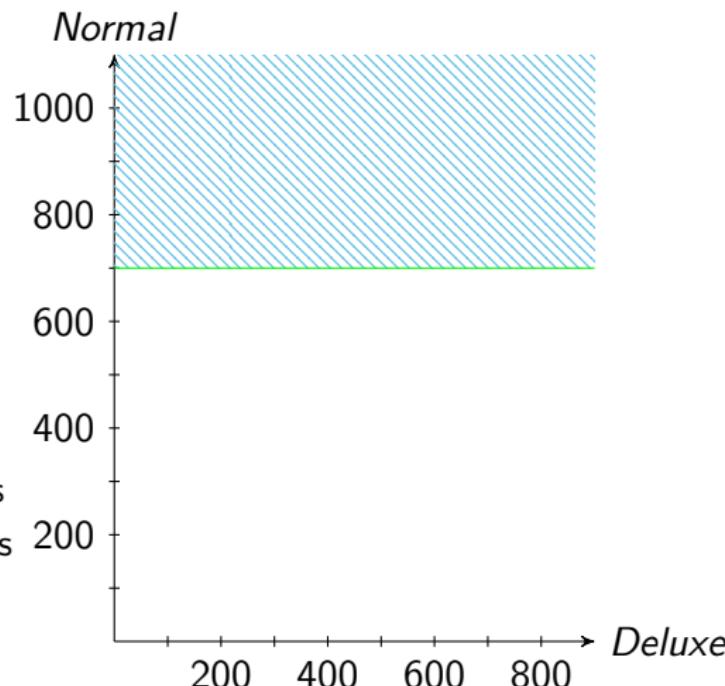
# Infeasible solution

We get an infeasible solution, if the solution space is empty.

Solution space

**Minimum** restrictions

**Maximum** restrictions



# Infeasible solution

We get an infeasible solution, if the solution space is empty.



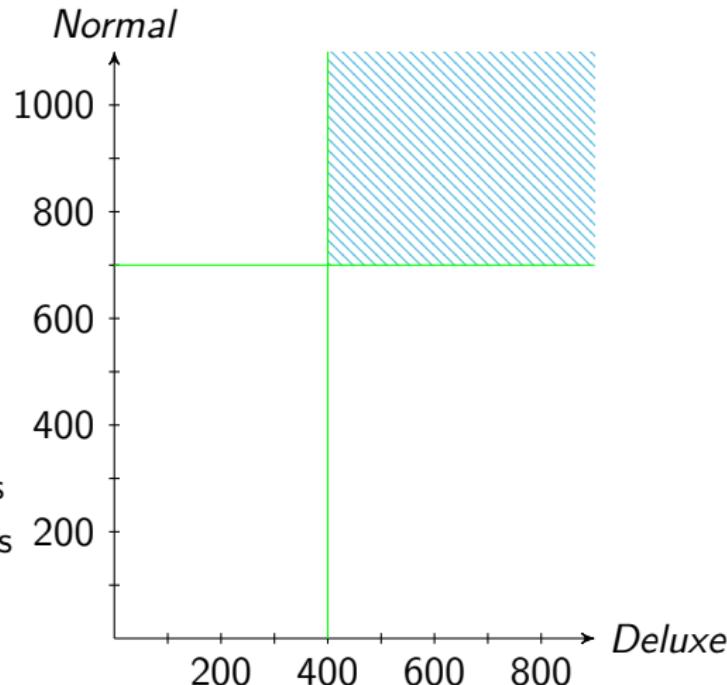
Solution space



Minimum restrictions



Maximum restrictions



# Infeasible solution

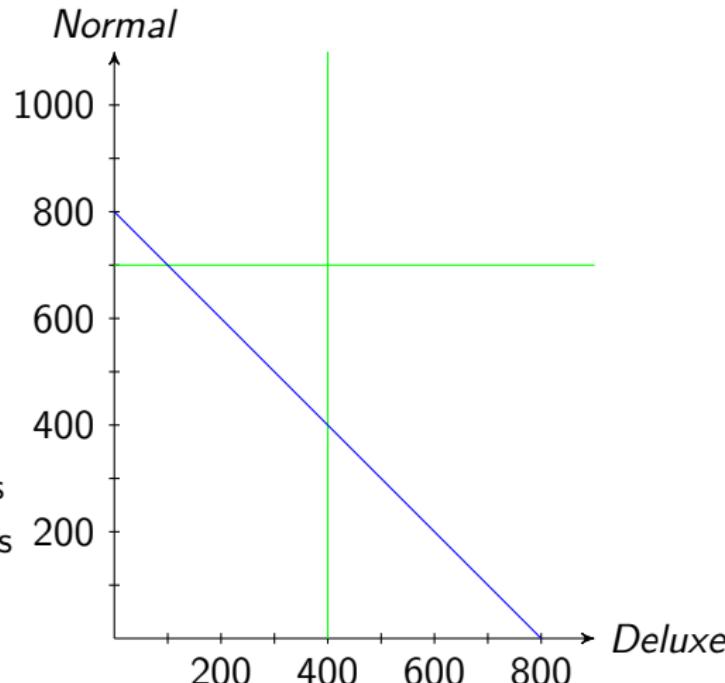
We get an infeasible solution, if the solution space is empty.



Solution space

**Minimum** restrictions

**Maximum** restrictions



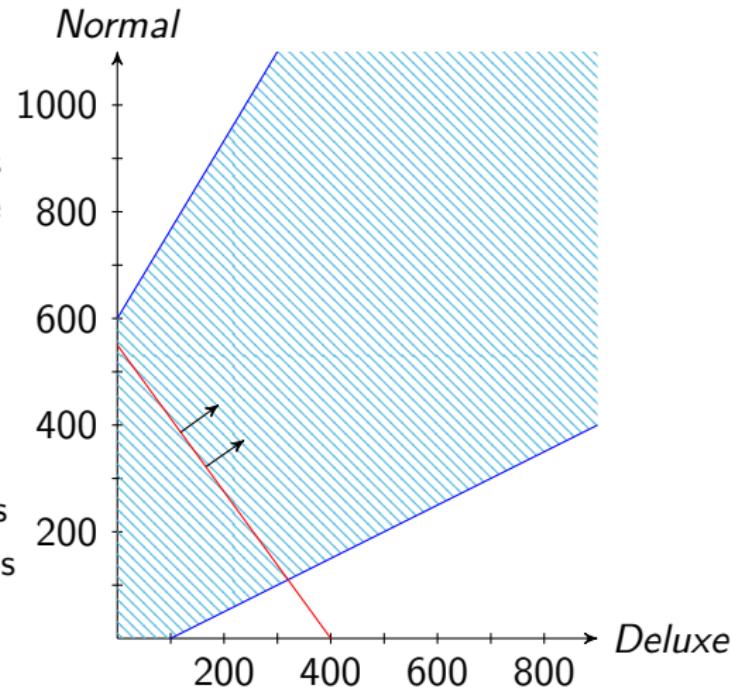
# Unbounded solution

We don't have an optimal solution, if the solution space is unbounded according to the objective function.



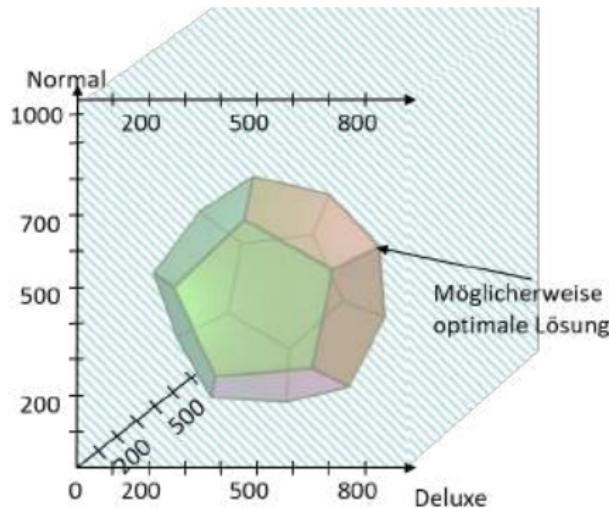
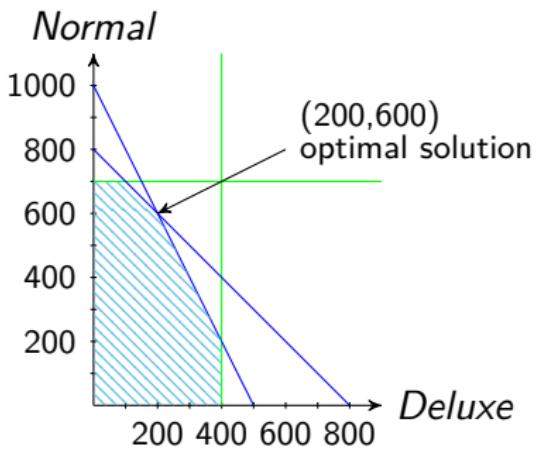
Solution space

- **Minimum** restrictions
- **Maximum** restrictions



# Property of an optimal solution

An optimal solution is always located at the border of the solution space.



# Solving LPs

## Solution methods for LPs

- The graphical method is only suitable for problems with two or three variables.
- Simplex method.
- Interior point method.

# Bike manufacturer

Model extension with one additional type

- We add a new type: Sport.
- Profit amounts to €80 per unit, production is capped at 500 units and production time is 5 minutes.
- Decision variables:

$d, n, s$  = Number of bikes to be produced of Deluxe,  
Standard and Sport types per day

$$\max z = 120d + 90n + 80s$$

Objective function (profit max.)

subject to

$d + n + s \leq 800$	Constraint (production volume)
$d \leq 400$	Constraint (maximum Deluxe)
$n \leq 700$	Constraint (maximum Standard)
$s \leq 500$	Constraint (maximum Sport)
$12d + 6n + 5s \leq 6000$	Constraint (time)
$d, n, s \geq 0$	Constraint (non-negativity)

# Bike manufacturer

Model extension with one additional type

- We add a new type: Sport.
- Profit amounts to €80 per unit, production is capped at 500 units and production time is 5 minutes.
- Decision variables:

$d, n, s$  = Number of bikes to be produced of Deluxe, Standard and Sport types per day

$$\max z = 120d + 90n + 80s$$

Objective function (profit max.)

subject to

$$d + n + s \leq 800$$

$$d \leq 400$$

$$n \leq 700$$

$$s \leq 500$$

$$12d + 6n + 5s \leq 6000$$

$$d, n, s \geq 0$$

Constraint (production volume)

Constraint (maximum Deluxe)

Constraint (maximum Standard)

Constraint (maximum Sport)

Constraint (time)

Constraint (non-negativity)

## Bike manufacturer: Extended model

- The model has to be extended if the bike manufacturer wants to add new types of bikes
  - New decision variables
  - For each new decision variable, a new maximum restriction has to be added
  - Existing restrictions have to be modified
  - The objective function has to be extended
- Matrix view: Existing model is stiff and its modification is time-consuming!
- Therefore: General notation!

## General notation

# Bike manufacturer

## General notation

Define index sets for bike types:

$F$  Set of bike types

Define the necessary parameters for the model:

$d_f$	Profit for bike type $f \in F$	[€/unit]
$m_f$	Maximum production for bike type $f \in F$	[units/day]
$e_f$	Manufacturing time for bike type $f \in F$	[min/unit]
$g$	Maximum number of bikes	[units/day]
$a$	Available operating hours	[min/day]

$$\max z = 120x_1 + 90x_2$$

subject to (s.t.)

$$x_1 + x_2 \leq 800$$

$$x_1 \leq 400$$

$$x_2 \leq 700$$

$$12x_1 + 6x_2 \leq 6000$$

$$x_1, x_2 \geq 0$$

# Bike manufacturer

General notation

Define decision variables:

$x_f$  Number produced of bike type  $f \in F$  [units/day]

Define objective function:

$$\max \sum_{f \in F} (x_f * d_f)$$

Constraints: s.t.	$x_f \leq m_f \quad \forall f \in F$	Maximum production
	$\sum_{f \in F} x_f \leq g$	Total number of bikes
	$\sum_{f \in F} x_f * e_f \leq a$	Manufacturing time
	$x_f \geq 0 \quad \forall f \in F$	Non-negativity

# General models

- General models are used to formulate optimization problems without specifying the concrete characteristics.
- Components:
  - Objects of the model (E.g. Types of bikes, products, resources,...) as elements of sets
  - Data and parameters (E.g. Operating time of machines, cost, ...)
  - Decision variables (E.g. Production volumes, ...)
  - Constraints (E.g. Maximum production, ...)
  - Optimization goal (E.g. Profit maximization, cost minimization,...)

# General models

## Formulation

Approach:

1. Define index sets
  2. Define parameters
  3. Define decision variables
  4. Define objective function
  5. Define constraints
- 
- } Data modeling

## An example

# Planning of production: Problem description

- Petra likes to bake cookies and plans on selling them at the christmas market. She's producing three types of cookies: cinnamon rolls, nut cookies and chocolate cookies.
- Ingredients per 50 cookies are given in the following table:

	Cinnamon rolls	Nut cookies	Chocolate cookies
flour [g]	180	250	400
sugar [g]	140	75	200
butter [g]	60	-	250
cinnamon [g]	10	-	-
nut [g]	-	150	-
chocolate [g]	-	-	200

- Petra has 3kg flour, 2kg sugar, 1kg butter, 250g cinnamon, 500g nut and 200g chocolate at home.
- She sells each cinnamon roll for €0.25, each nut cookie for €0.30 and each chocolate cookie for €0.30.
- *Create a general linear model to maximize the sales.*

# Planning of production: General model

Build the sets:

- $C$  set of types of cookies
- $I$  set of ingredients

Define parameters:

$p_c$	price per cookie $c \in C$	[€/piece]
$z_{ci}$	required amount of ingredient $i \in I$ of $c \in C$	[g/piece]
$a_i$	available amount of ingredient $i \in I$	[g]

# Planning of production – general model

Define decision variables:

$x_c$  number produced of cookie  $c \in C$  [piece]

Set up objective function:

$$\max \sum_{c \in C} x_c * p_c$$

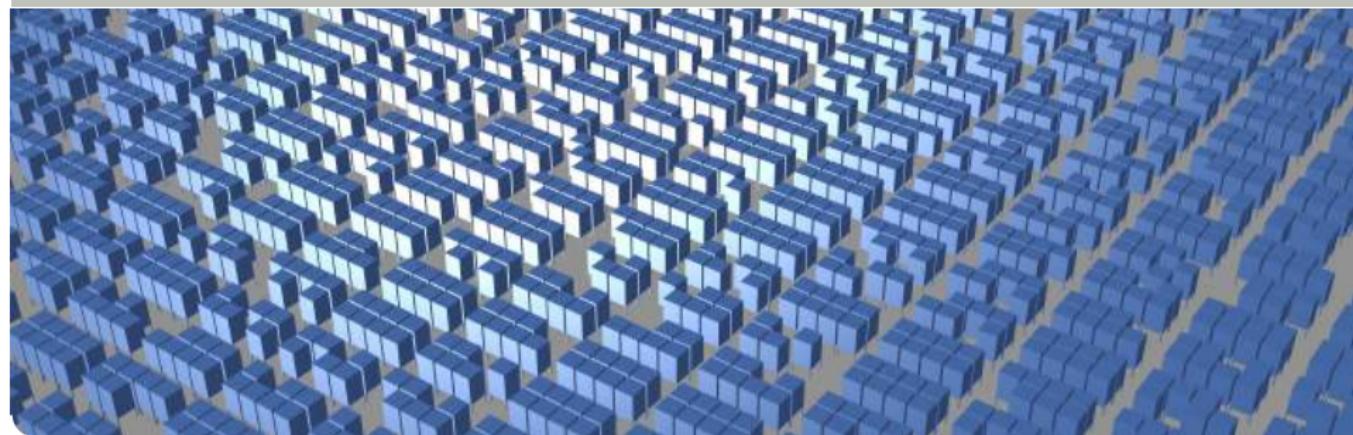
$$\begin{aligned} \text{s.t. } & \sum_{c \in C} z_{ci} * x_c \leq a_i \quad \forall i \in I && \text{limited amount of ingredient} \\ & x_c \geq 0 \quad \forall c \in C && \text{NNB} \end{aligned}$$

# Analysing networks using OR-methods

Part 3 – Transshipment

Lin Xie | 27.04.2021

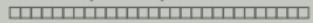
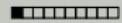
PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH



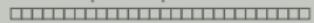
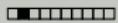
## 1 Network models

## 2 Network flow models

## 3 Transshipment problem



## Network models



# Applications

Many human-made systems are based on networks.

For example:

- telecommunications networks (internet, telephone)
- transportation networks (streets, railways)
- supply networks (water, gas, electricity).

Network structures can be found in:

- subway/bus services
- gas pipelines.

The structures of these networks are usually complex.

- A lot of research into network models and problems.

# Application problems

For example:

- finding the shortest path (Google Maps, navigation systems)
- minimizing transportation cost (in delivery chains)
- planning vehicle deployment
- planning personnel deployment
- planning supply networks and their usage with efficient costs.



## Example: Bus network



# Network models

- A network model is an abstraction of reality.  
⇒ Concentration on the essential question.
- Network models represent structure or dependencies.
- Network models improve comprehension of reality, in particular because parts of the model have equivalence with reality.

Until now: Visual network models.

Now: Formulation.

First step: Display structure or dependencies as graph.

# Network models as graphs

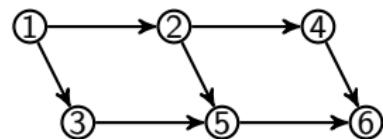
- Graph theory is one of the branches of mathematics.
- It enables, e.g. mathematical models of networks.
- Definition: **Graph**
  - Consists of **nodes** and **arcs**
  - Graph  $G = (N, A)$   
with a set of nodes  $N$   
and a set of arcs  $A$
- Arcs might have a **direction**.

Here: Directed graph  $G = (N, A)$  with a set of directed arcs  $A \subseteq N \times N$

$$N = \{1, 2, 3, 4, 5, 6\}$$

$$A =$$

$$\{(1, 2), (1, 3), (2, 4), (2, 5), (3, 5), (4, 6), (5, 6)\}$$



# Network models as graphs

## Definitions of directed graphs

- $G' = (N', A')$  is a **subgraph** of  $G = (N, A)$ , if  $N' \subseteq N$  and  $A' \subseteq A$ .

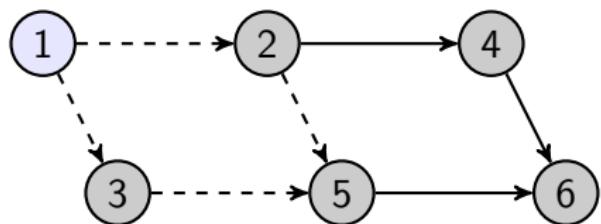
Under condition: arcs in  $A'$  connect nodes  $N'$ .

Reason:  $G'$  has to be a graph, which means arcs have to connect nodes.

*Example:*

$$N' = \{1, 2, 3, 5\}$$

$$A' = \{(1, 2), (1, 3), (2, 5), (3, 5)\}$$



# Network models as graphs

## Definitions of directed graphs

- $G' = (N', A')$  is a **subgraph** of  $G = (N, A)$ , if  $N' \subseteq N$  and  $A' \subseteq A$ .

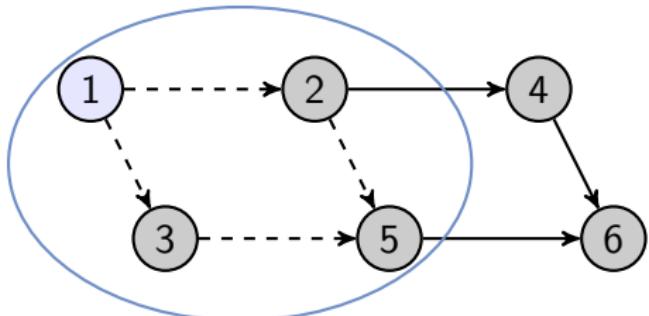
Under condition: arcs in  $A'$  connect nodes  $N'$ .

Reason:  $G'$  has to be a graph, which means arcs have to connect nodes.

*Example:*

$$N' = \{1, 2, 3, 5\}$$

$$A' = \{(1, 2), (1, 3), (2, 5), (3, 5)\}$$



# Network models as graphs

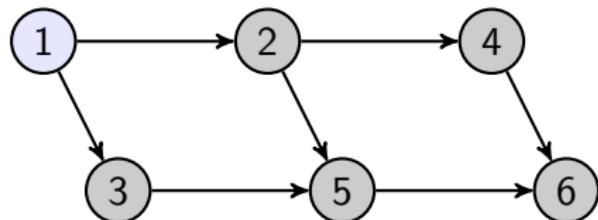
## Definitions of directed graphs

- A **route** is a sequence  $(e_1, e_2, \dots, e_t)$  of arcs in  $A$  with  $t \geq 1$ , where a given sequence of nodes are connected (i.e.,  $e_k = (n_{k-1}, n_k)$  for all  $k = 1, \dots, t$ ) and each arc is only traversed in one direction. So each route has a start and an end node.
- A route is **elemental** if, except for the start and end node, no node is visited twice.

*Example:*

Elemental route  $((1, 2), (2, 4))$

In this example, all routes are elemental.



# Network models as graphs

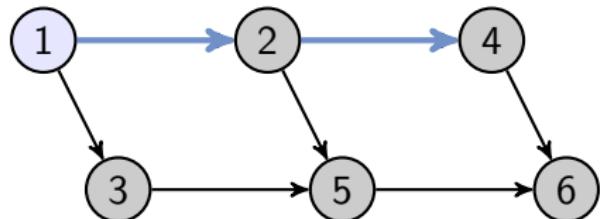
## Definitions of directed graphs

- A **route** is a sequence  $(e_1, e_2, \dots, e_t)$  of arcs in  $A$  with  $t \geq 1$ , where a given sequence of nodes are connected (i.e.,  $e_k = (n_{k-1}, n_k)$  for all  $k = 1, \dots, t$ ) and each arc is only traversed in one direction. So each route has a start and an end node.
- A route is **elemental** if, except for the start and end node, no node is visited twice.

*Example:*

Elemental route  $((1, 2), (2, 4))$

In this example, all routes are elemental.



## Network models as graphs

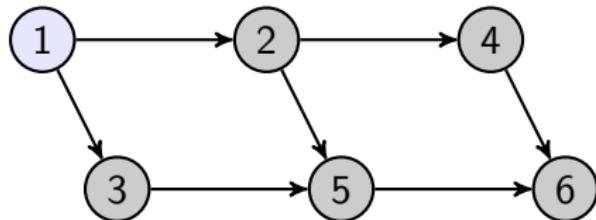
## Definitions of directed graphs

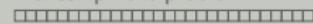
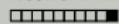
- A route is a cyclic route if its start and end nodes are the same.
  - A graph that has no cycles is called a **directed acyclic graph** (DAG). This is an important characteristic, because many algorithms only run on DAGs.
  - A directed graph is **strongly connected** if there are routes from  $i$  to  $j$  and from  $j$  to  $i$  for each node pair  $i$  and  $j$  ( $i \neq j$ ).

In this example:

DAG

Not strongly connected





# Network models as graphs

## Properties of nodes and arcs

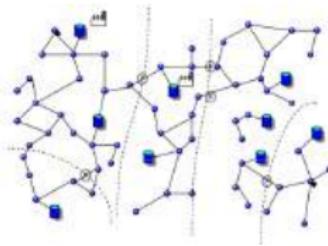
- So far: Simple network models as graphs:

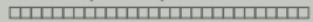
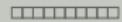
- nodes
- arcs, which connect the nodes
- length of arcs (non-negative).

- Nodes and arcs can have more properties in reality.

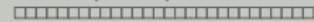
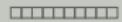
**For example, in a water network:**

- source nodes are different from other nodes
- properties of hydraulics of the pipes, such as capacities
- variable water flow at different times.





## Network flow models



## Network flow models

Network flow models are network models, in which „something“ flows over the arcs from one node to another.

For example:

- water flow in cubic meters per second
- number of passengers, vehicles, etc.
- number of packages.

This means that for each arc a decision variable is declared:

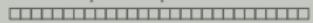
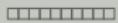
Flow  $x_{ij}$  on the arc  $(i,j)$

For example:  $x_{ij} = 5 \Rightarrow 5$  units flow from i to j

# Network flow models

## Parameters of nodes

- Nodes can have a supply or a demand.  
parameter is defined for each node  $i$ :
  - Node  $i$  is a supply node with supply  $s(i)$ .
  - Node  $i$  is a demand node with demand  $d(i)$ .
  - Node  $i$  is a transshipment node without supply or demand.
  
- For example, in a water supply network:
  - Sources/wellheads are illustrated as supply nodes.
  - Industrial sites and private homes are illustrated as demand nodes.



# Network flow models

## Parameters of arcs

- Arcs can have minimum and maximum capacities:
  - parameter lower bound  $l_{ij}$  for all arcs  $(i,j)$
  - parameter upper bound  $\kappa_{ij}$  for all arcs  $(i,j)$
- For example, in a water supply network :
  - minimum amount of water, in cubic meters per second, that has to flow through pipes (for hygiene)
  - maximum amount of water, in cubic meters per second, that can flow through pipes (because of pressure).
- The arcs are directed!

# Network flow models

Additional parameters of arcs

Usage of arcs causes costs:

**Costs**  $c_{ij}$  for all arcs  $(i, j)$

In network flow models, the cost arises for each unit that flows through the arc  $(i, j)$ :

$$c_{ij}x_{ij}$$

For example, in a water supply network: costs for extracting water from a source through the arcs.

## Transshipment problem

## Single-stage transportation problem

## Supply      Demand



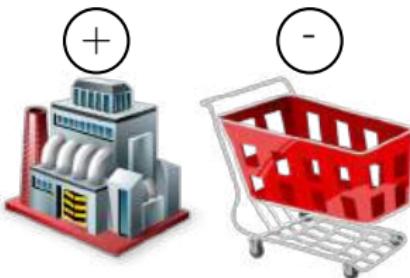
# Single-stage transportation problem

- Also called the “classic transportation problem.”
- The objective is to determine the shipping schedule that minimizes the total shipping cost while satisfying supply and demand limits.
- The supply nodes  $i = 1, \dots, m$  offer  $s_i$  units of a product.
- The demand nodes  $j = 1, \dots, n$  have a demand  $d_j$  for the product.
- Total supply equals total demand.
- Each demander can be shipped from each supplier, even partially!

# Transshipment problem

Supply

Demand



Transshipment



# Transshipment problem

Supply

Demand



Transshipment



# Transshipment problem

- The single-stage transportation problem is called “single-stage” because there is no transshipment.
- If there is one transshipment on the route from each supplier to each demander, it is called the “two-stage transportation problem”, etc.
- The term “transshipment problem” is used if:
  - different numbers of transshipments are located on the routes, or
  - not all routes are allowed, or
  - additionally, the arcs have minimum capacities.
  - In particular, the transshipping is allowed at supply and demand nodes.

# Transshipment problem

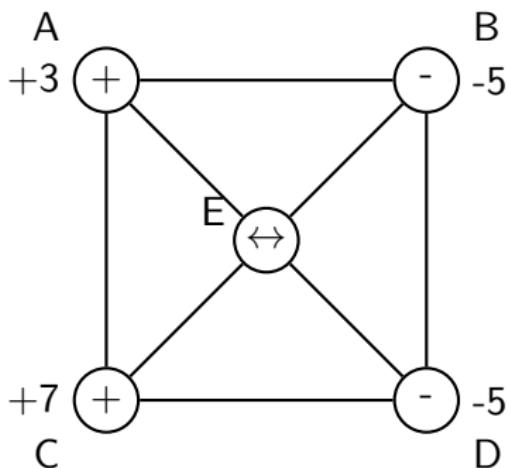
## Application example



Source: <https://www.tendersontime.com/blogdetails/water-supply-works-25461/>

## Transshipment problem

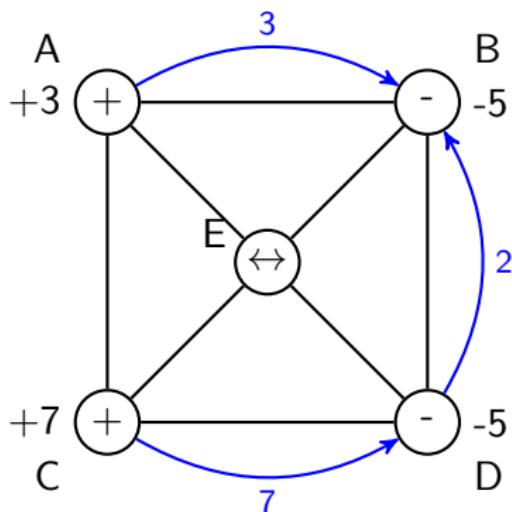
## Example



uncapacitated;  
without cost;  
supply=demand

# Transshipment problem

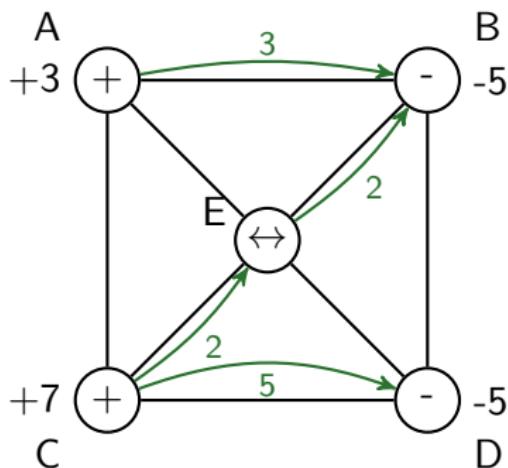
## Example



uncapacitated;  
without cost;  
supply=demand

# Transshipment problem

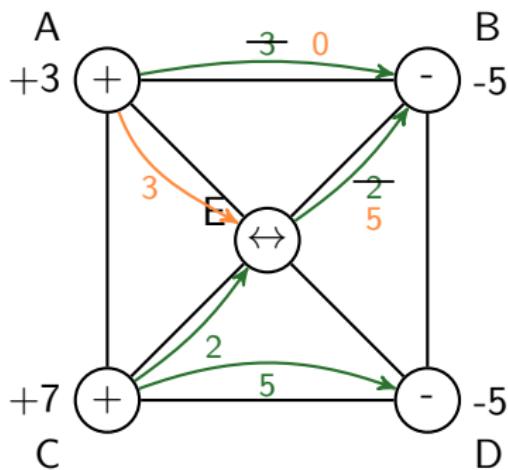
## Example



uncapacitated;  
without cost;  
supply=demand

# Transshipment problem

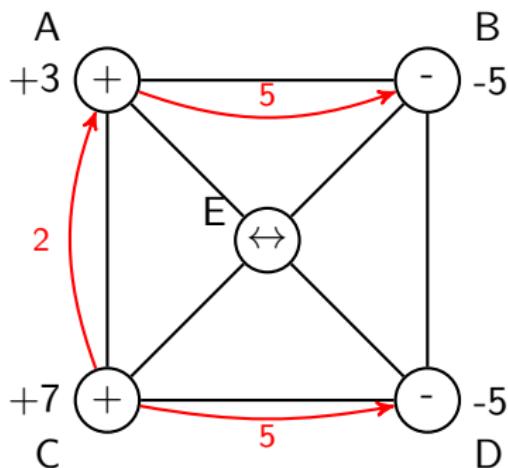
## Example



uncapacitated;  
without cost;  
supply=demand

# Transshipment problem

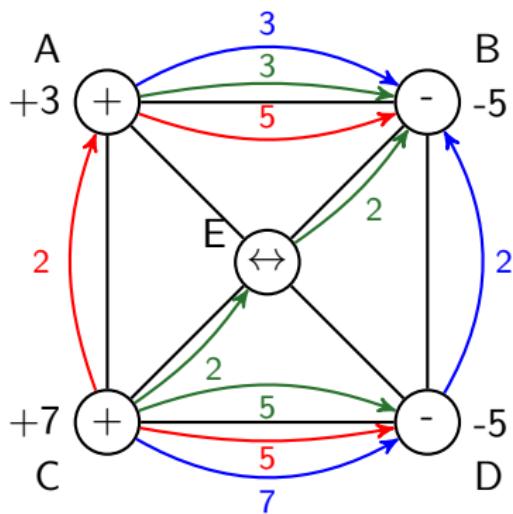
## Example



uncapacitated;  
without cost;  
supply=demand

# Transshipment problem

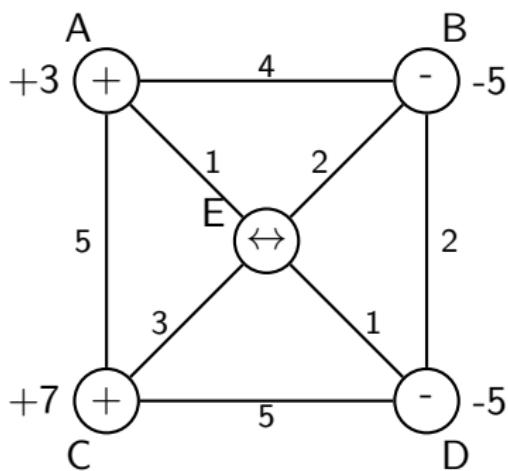
## Example



uncapacitated;  
without cost;  
supply=demand

# Transshipment problem

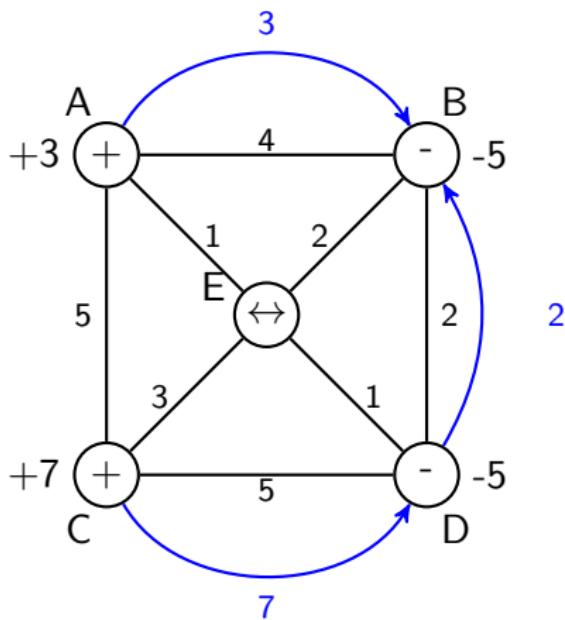
## Example



**capacitated;**  
without cost;  
supply=demand

# Transshipment problem

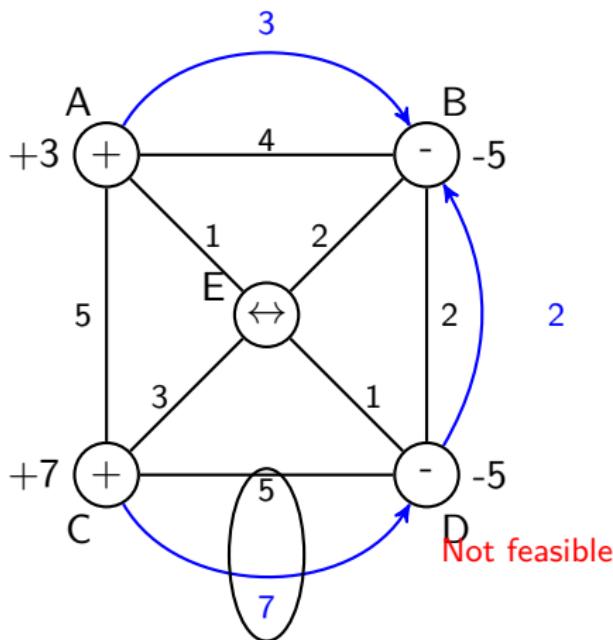
## Example



**capacitated;**  
without cost;  
supply=demand

# Transshipment problem

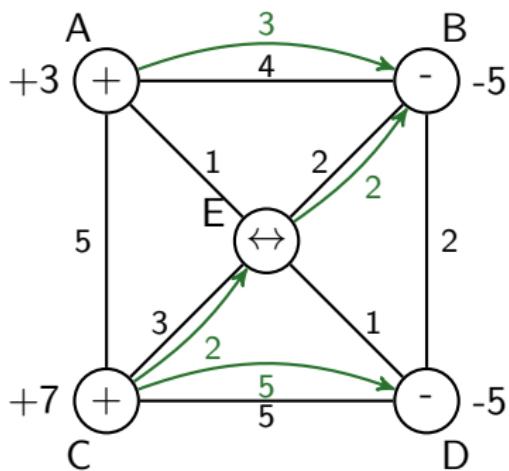
## Example



**capacitated;**  
without cost;  
supply=demand

# Transshipment problem

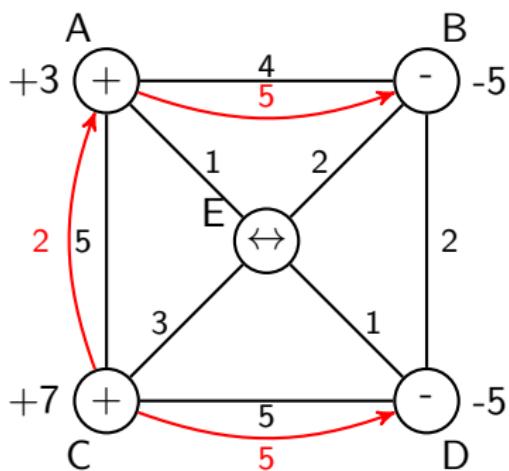
## Example



**capacitated;**  
without cost;  
supply=demand

# Transshipment problem

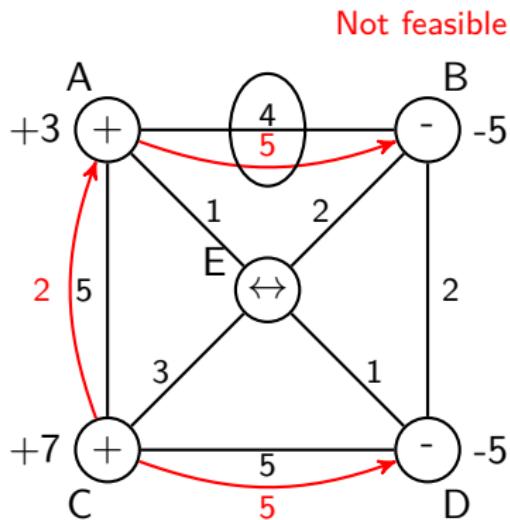
## Example



**capacitated;**  
without cost;  
supply=demand

# Transshipment problem

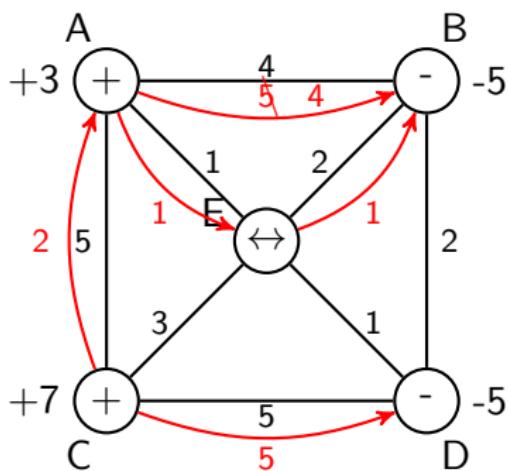
## Example



**capacitated;**  
without cost;  
supply=demand

# Transshipment problem

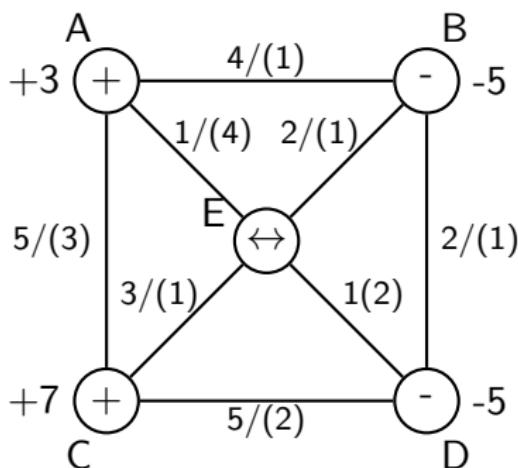
## Example



**capacitated;**  
without cost;  
supply=demand

# Transshipment problem

## Example



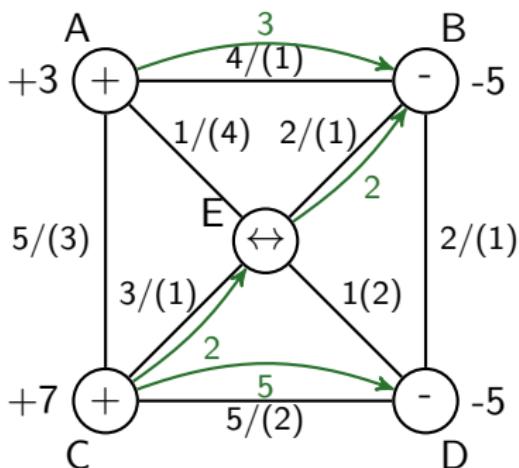
capacitated;  
**with cost;**  
 supply=demand

capacity/(cost)

We want a transportation plan  
 that minimizes transportation  
 cost while satisfying demand.

## Transshipment problem

## Example



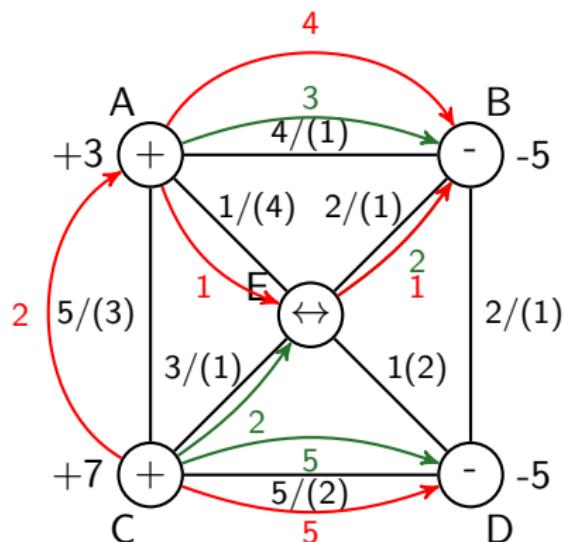
capacitated;  
**with cost;**  
supply=demand

capacity/(cost)

total cost of the green path:  $3(1) + 2(1) + 2(1) + 5(2) = 17$

## Transshipment problem

## Example



capacitated;  
**with cost;**  
supply=demand

capacity/(cost)

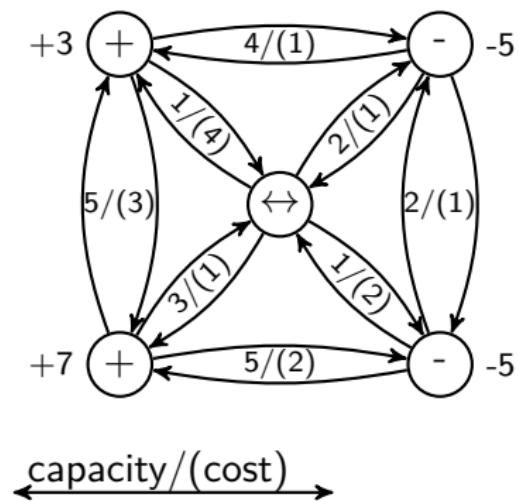
total cost of the green path:  $3(1) + 2(1) + 2(1) + 5(2) = 17$

total cost of the red one:  $4(1) + 1(4) + 1(1) + 2(3) + 5(2) = 21$

# Transshipment problem – Mathematical model

Graph  $G = (N, A)$

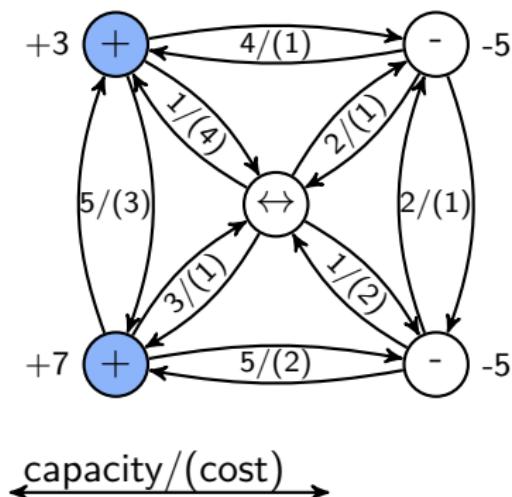
$$N = N^+ \cup N^{\leftrightarrow} \cup N^-$$



# Transshipment problem – Mathematical model

Graph  $G = (N, A)$

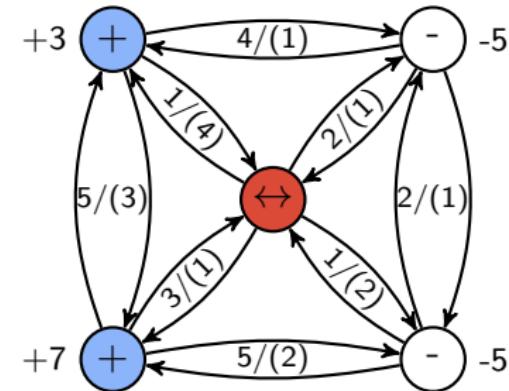
$$N = N^+ \cup N^\leftrightarrow \cup N^-$$



# Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

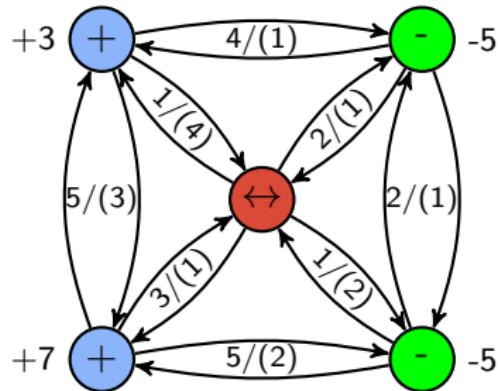


$\xleftarrow{\text{capacity}/(\text{cost})}$

# Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$



capacity/(cost)

## Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

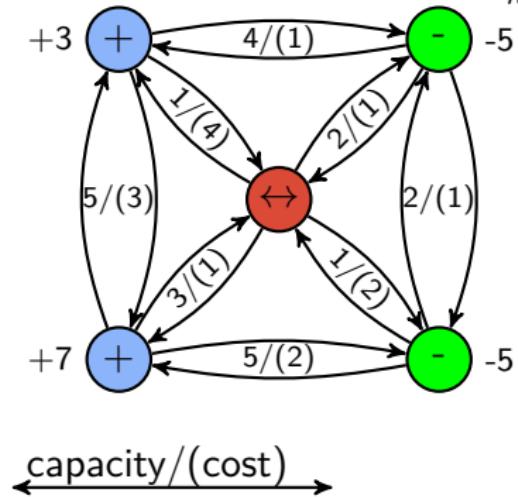
$x_{ij}$  Number of units flowing on arc  $(i,j) \in A$

$s_i$ : Supply of units at node  $i \in N^+$

$d_i$ : Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i, j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i,j) \in A$



## Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

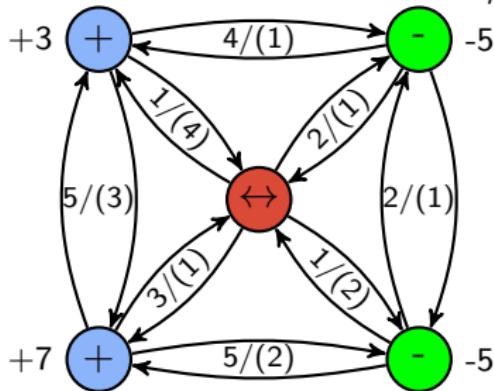
$x_{ij}$  Number of units flowing on arc  $(i,j) \in A$

$s_i$ : Supply of units at node  $i \in N^+$

$d_i$ : Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i, j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i,j) \in A$



$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

capacity/(cost)

## Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

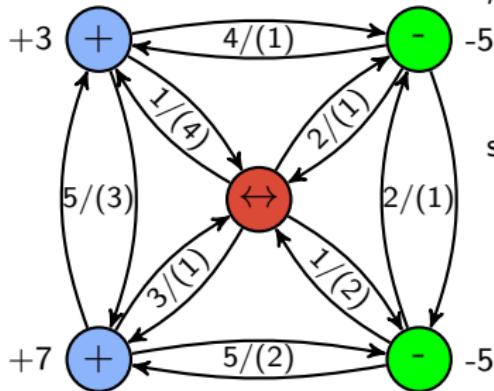
$x_{ij}$  Number of units flowing on arc  $(i,j) \in A$

$s_i$ : Supply of units at node  $i \in N^+$

$d_i$ : Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i,j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i,j) \in A$



$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+$$

capacity/(cost)

## Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

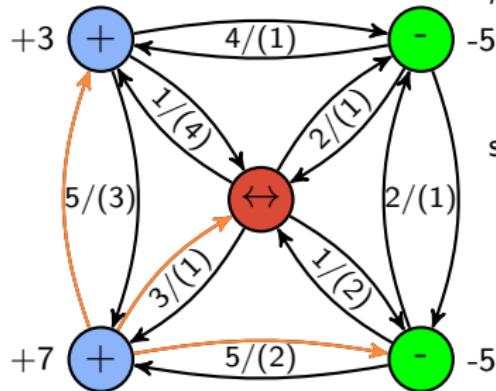
$x_{ij}$  Number of units flowing on arc  $(i,j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$ : Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i,j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i,j) \in A$



$$-5 \quad \min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+$$

capacity/(cost)

## Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

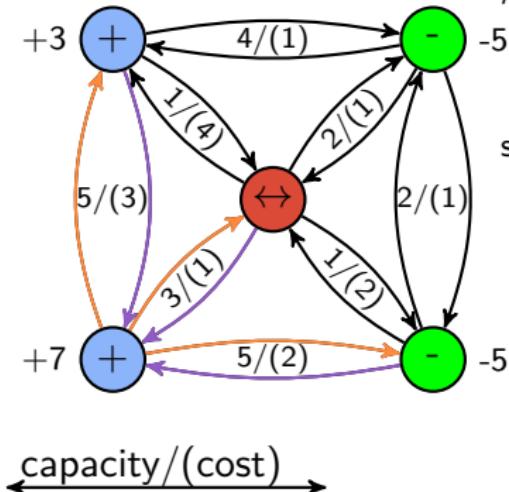
$x_{ij}$  Number of units flowing on arc  $(i,j) \in A$

$s_i$ : Supply of units at node  $i \in N^+$

$d_i$ : Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i,j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i,j) \in A$



$$-5 \quad \min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+$$

# Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

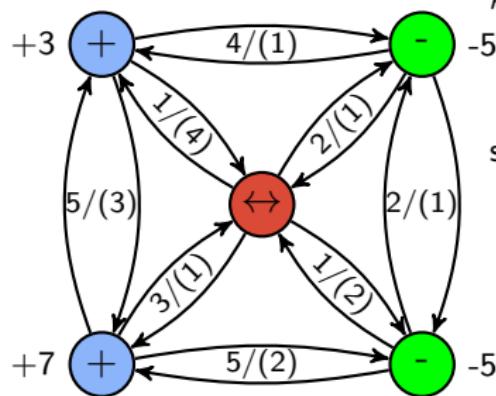
$x_{ij}$  Number of units flowing on arc  $(i, j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$  Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i, j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i, j) \in A$



$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = -d_i \quad \forall i \in N^-$$

capacity/(cost)

# Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

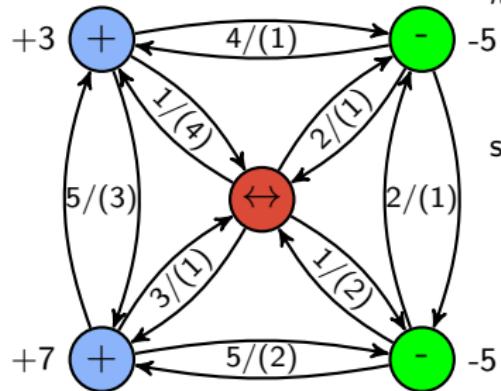
$x_{ij}$  Number of units flowing on arc  $(i, j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$  Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i, j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i, j) \in A$



$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = -d_i \quad \forall i \in N^-$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in N^\leftrightarrow$$

capacity/(cost)

# Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

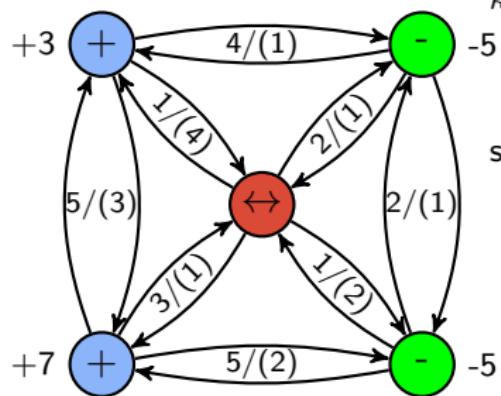
$x_{ij}$  Number of units flowing on arc  $(i, j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$  Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i, j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i, j) \in A$



$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = -d_i \quad \forall i \in N^-$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in N^\leftrightarrow$$

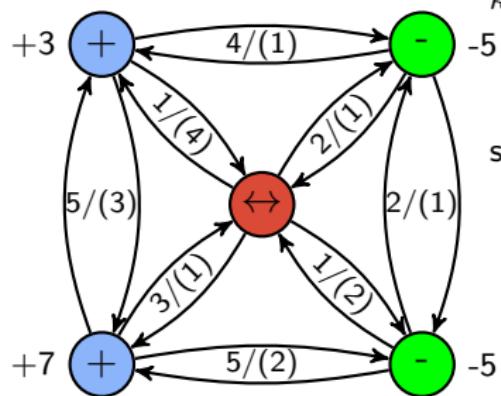
capacity/(cost)

$$0 \leq x_{ij} \leq \kappa_{ij} \quad \forall (i,j) \in A$$

# Transshipment problem – Mathematical model

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$



$x_{ij}$  Number of units flowing on arc  $(i, j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$  Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i, j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i, j) \in A$

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

**Flow conservation constraints**

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = -d_i \quad \forall i \in N^-$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in N^\leftrightarrow$$

capacity/(cost)

$$0 \leq x_{ij} \leq \kappa_{ij} \quad \forall (i,j) \in A$$

# Extending the transshipment problem

## Task

There is a production cost at each supply node. You have to decide whether the production will occur at each supply node. Adjust the previous mathematical model to support this new constraint.

# Transshipment problem

- The transshipment problem determines who supplies whom and where to tranship goods to.

# Transshipment problem

- The transshipment problem determines who supplies whom and where to tranship goods to.
- What if we want to serve several customers on one route?  
What is the shortest route?
- How many vehicles do we need, if they have limited capacity?

# Transshipment problem

- The transshipment problem determines who supplies whom and where to tranship goods to.
- What if we want to serve several customers on one route?  
What is the shortest route?
- How many vehicles do we need, if they have limited capacity?
- Next week:
  - traveling salesman problem
  - vehicle routing problem
  - vehicle routing problem with time windows
  - further variants and solution methods.

# Literature

- L. Suhl and T. Mellouli (2013). *Optimierungssysteme – Modelle, Verfahren, Software, Anwendungen*. 3rd ed., Springer Gabler, pp. 146–154.

# Thank you for your attention!

Leuphana University of Lüneburg  
Business information systems, in particular Operations Research  
Prof. Dr. Lin Xie  
Universitätsallee 1  
Building 4, Room 314  
21335 Lüneburg  
Phone +49 4131 677 2305  
Fax +49 4131 677 1749  
[xie@leuphana.de](mailto:xie@leuphana.de)

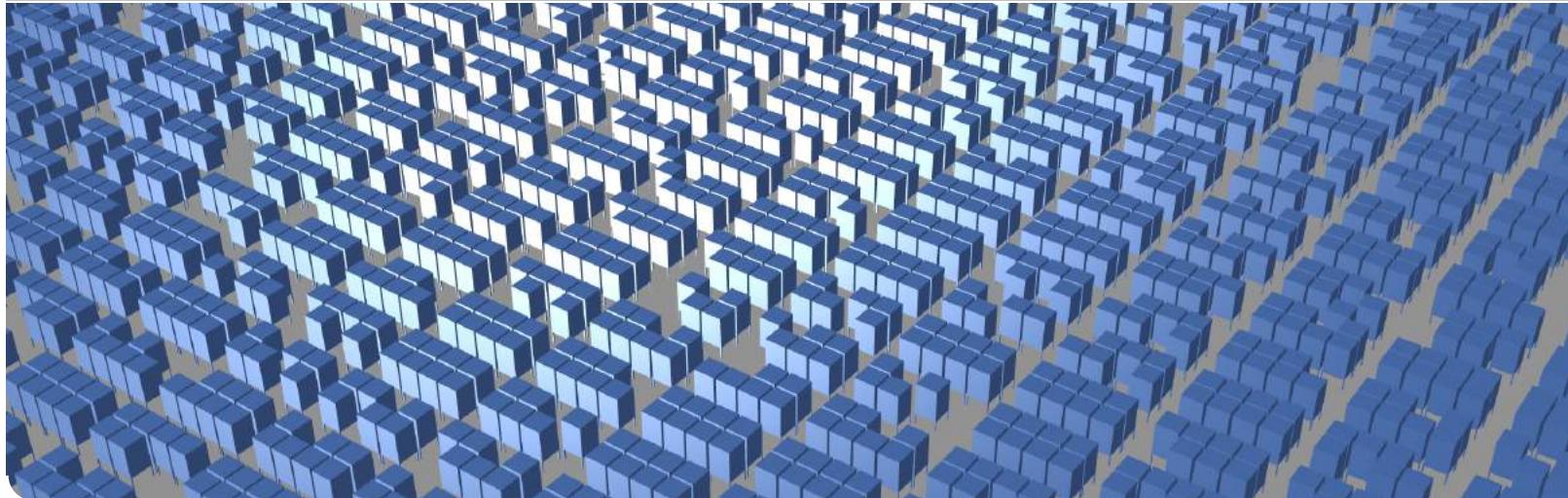


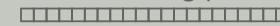
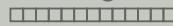
# Analysing networks using OR-methods

Part 4 – Traveling salesman, vehicle routing

Lin Xie | 04.05.2021

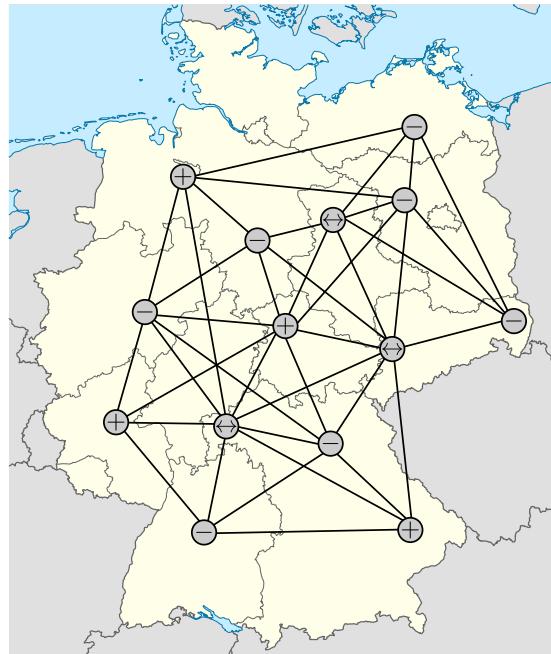
PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH





*Recap*

# Transshipment problem

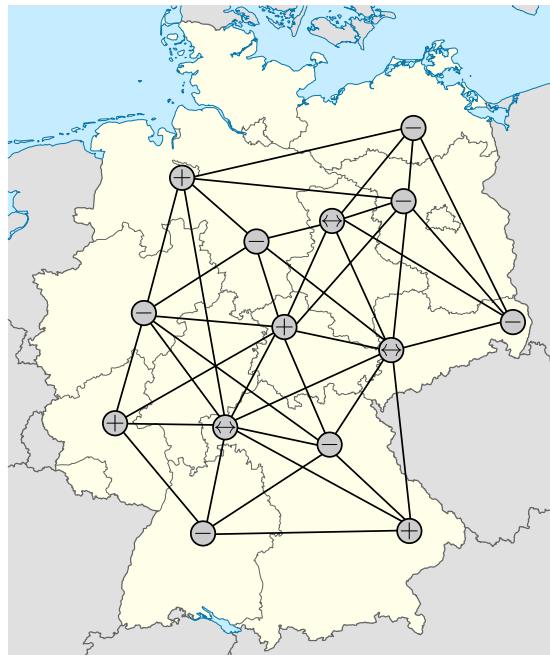


## Location types:

- Production
- Consumption
- Transshipment

# Transshipment problem

*Recap*



## Location types:

-  Production
-  Consumption
-  Transshipment

## Constraints:

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i)} x_{ji} = \begin{cases} s_i & \text{if } i \in V^+ \\ -d_i & \text{if } i \in V^- \\ 0 & \text{otherwise} \end{cases}$$

# Agenda for today

- The transshipment problem determines who supplies whom and where to tranship to

# Agenda for today

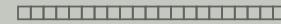
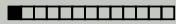
- The transshipment problem determines who supplies whom and where to transship to
- What if we want to serve several customers on one route?  
What is the shortest route?
- How many vehicles do we need if they have limited capacity?

# Agenda for today

- The transshipment problem determines who supplies whom and where to tranship to
- What if we want to serve several customers on one route?  
What is the shortest route?
- How many vehicles do we need if they have limited capacity?

## Today:

- Traveling salesman problem
- Vehicle routing problem
- Vehicle routing problem with time windows
- Outlook on further variants and solution methods



# Traveling salesman problem

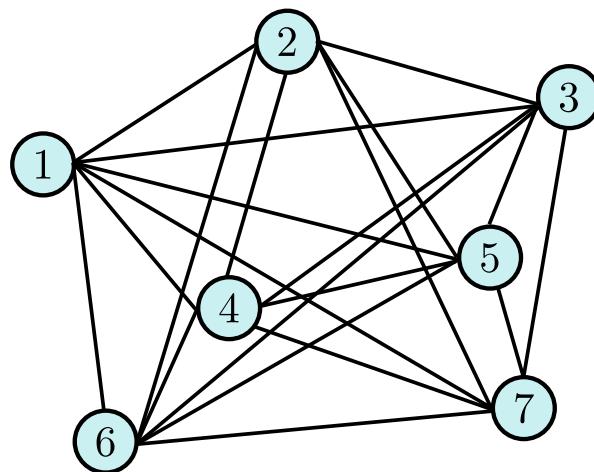
# TSP example

*Recap*

## Traveling Salesman Problem

**Given:** A graph  $G = (V, E)$  with symmetric edge costs  $c_e$ .

**Objective:** Find the minimum cost path traversing every node exactly once.



Graph

	1	2	3	4	5	6	7
1	0	2.2	5.0	2.8	4.1	5.0	8.5
2	2.2	0	3.0	3.0	2.8	6.0	8.0
3	5.0	3.0	0	4.2	2.2	7.2	7.0
4	2.8	3.0	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5.0	5.4
6	5.0	6.0	7.2	3.1	5.0	0	5.1
7	8.5	8.0	7.0	5.7	5.4	5.1	0

Edge costs

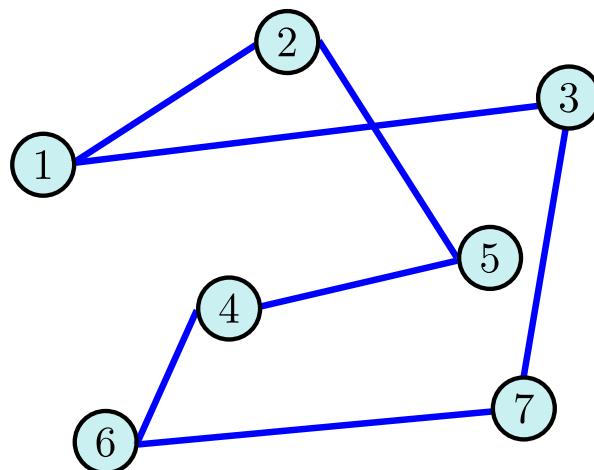
# TSP example

*Recap*

## Traveling Salesman Problem

**Given:** A graph  $G = (V, E)$  with symmetric edge costs  $c_e$ .

**Objective:** Find the minimum cost path traversing every node exactly once.



A solution (Cost: 27.4)

	1	2	3	4	5	6	7
1	0	2.2	5.0	2.8	4.1	5.0	8.5
2	2.2	0	3.0	3.0	2.8	6.0	8.0
3	5.0	3.0	0	4.2	2.2	7.2	7.0
4	2.8	3.0	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5.0	5.4
6	5.0	6.0	7.2	3.1	5.0	0	5.1
7	8.5	8.0	7.0	5.7	5.4	5.1	0

Edge costs

# The growth of P and NP problems

*Recap*

	Shortest path $O(n^2)$	Traveling salesman $O(n!)$
$n$	$ S $	$ S $
5	25	120
10	100	3,628,800
15	225	$1.3 \times 10^{12}$
20	400	$2.4 \times 10^{18}$
25	625	$1.5 \times 10^{25}$
30	900	$2.6 \times 10^{32}$

# The growth of P and NP problems

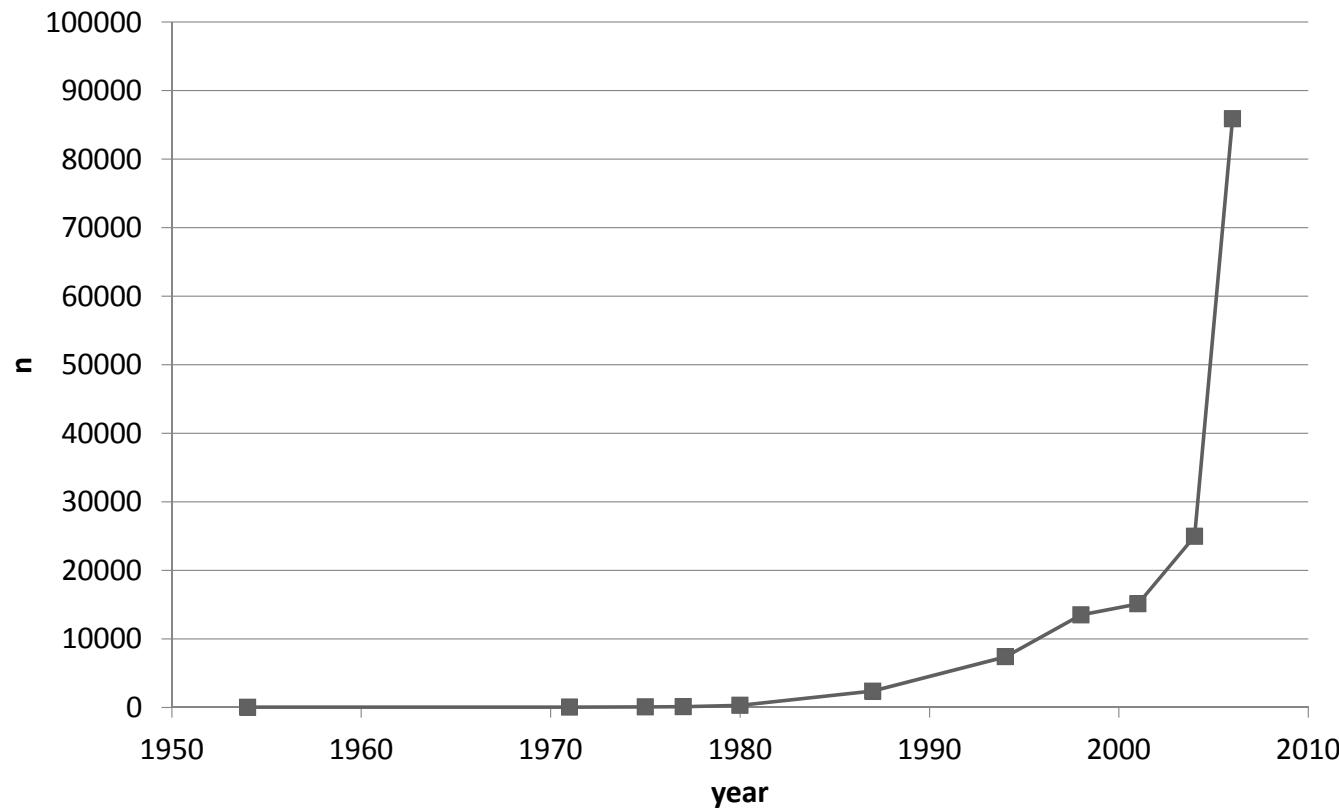
*Recap*

	Shortest path $O(n^2)$	Traveling salesman $O(n!)$
$n$	$ S $	$ S $
5	25	120
10	100	3,628,800
15	225	$1.3 \times 10^{12}$
20	400	$2.4 \times 10^{18}$
25	625	$1.5 \times 10^{25}$
30	900	$2.6 \times 10^{32}$

Given a computer that can process 200,000 solutions per second, you would need over 42,000,000,000,000,000 years to solve an  $n = 30$  traveling salesman problem!

# Sizes of solved instances

Solved TSP instances, i.e. proven optimality ( $n$  = number of nodes)



# Sweden

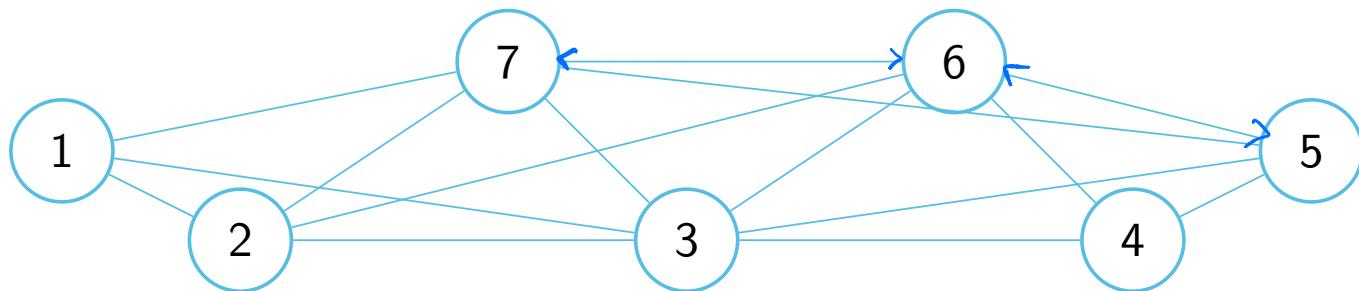
- 24,978 cities in Sweden
- solved at Georgia Tech, 2004
- highly specialized method
- 96 dual-core processors: Intel Xeon 2.8GHz
- ~84.4 CPU years on one processor
- [www.tsp.gatech.edu](http://www.tsp.gatech.edu)

# Mathematical formulation

Given: directed graph  $G = (N, A)$ , cost/distance  $c_{ij}$  for each arc  $(i, j) \in A$

Decision variable  $x_{ij}$

- $x_{ij} = 1$ , if arc  $(i, j)$  is part of the solution
- $x_{ij} = 0$ , else



*ensure each city is visited 1 time*

# Mathematical formulation

Given: directed graph  $G = (N, A)$ , cost/distance  $c_{ij}$  for each arc  $(i, j) \in A$

Decision variable  $x_{ij}$

- $x_{ij} = 1$ , if arc  $(i, j)$  is part of the solution
- $x_{ij} = 0$ , else

A possible optimization model for the TSP?

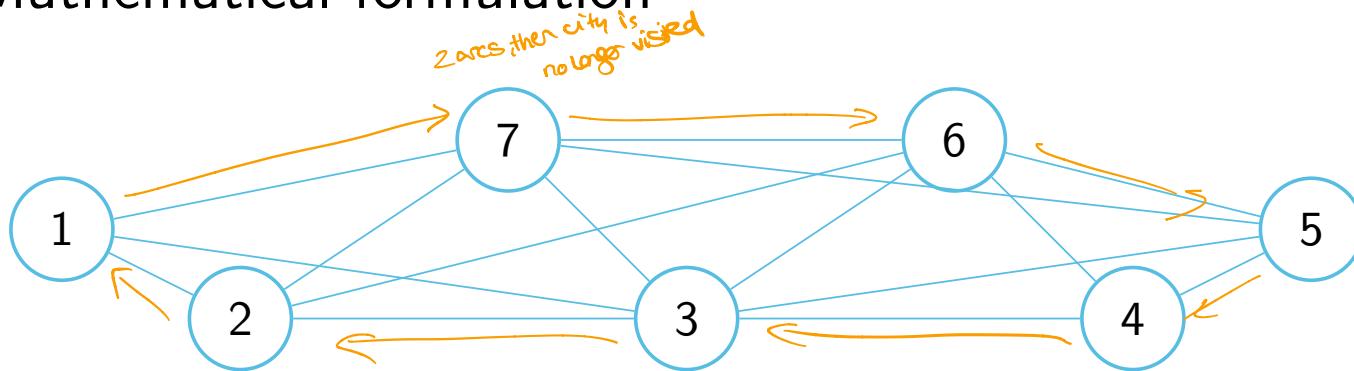
$$\text{Min } \sum_{(i,j) \in A} \xrightarrow{\text{costs}} c_{ij} x_{ij} \quad (\text{Minimize cost/distance})$$

$$s.t. \sum_{\{i:(i,j) \in A\}} x_{ij} = 1 \quad \forall j \in N \quad (\text{Exactly one incoming arc})$$

$$\sum_{\{k:(j,k) \in A\}} x_{jk} = 1 \quad \forall j \in N \quad (\text{Exactly one outgoing arc})$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A$$

# Mathematical formulation



$$\text{Min } \sum_{(i,j) \in A} c_{ij} x_{ij}$$

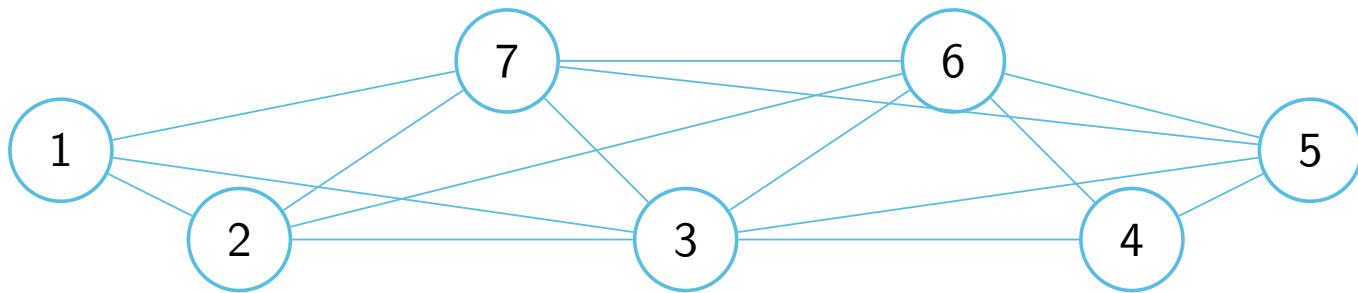
$$\text{s.t. } \sum_{\{i:(i,j) \in A\}} x_{ij} = 1 \quad \forall j \in N \text{ one incoming}$$

$$\sum_{\{k:(j,k) \in A\}} x_{jk} = 1 \quad \forall j \in N \text{ one outgoing}$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A$$

Problem:

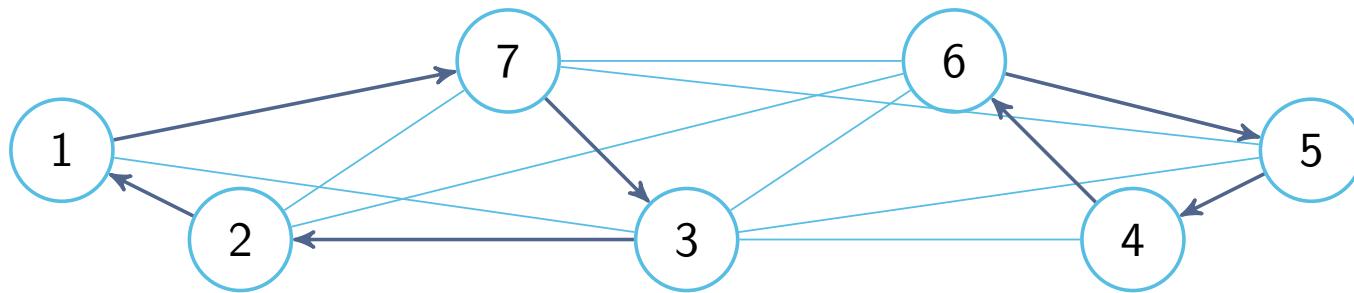
# Mathematical formulation



$$\begin{aligned}
 & \text{Min} \sum_{(i,j) \in A} c_{ij} x_{ij} \\
 \text{s.t. } & \sum_{\{i:(i,j) \in A\}} x_{ij} = 1 \quad \forall j \in N \\
 & \sum_{\{k:(j,k) \in A\}} x_{jk} = 1 \quad \forall j \in N \\
 & x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A
 \end{aligned}$$

The model is not correct! What is the problem with this formulation?

# Mathematical formulation



satisfies constraints  
but we don't want several tours!

$$\text{Min } \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{\{i:(i,j) \in A\}} x_{ij} = 1 \quad \forall j \in N$$

$$\sum_{\{k:(j,k) \in A\}} x_{jk} = 1 \quad \forall j \in N$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A$$

The model is not correct! What is the problem with this formulation? → *multiple independent subtours possible*  
*Need to be connected*



# Subtour elimination constraints

- Subtours need to be excluded by **subtour elimination constraints (SEC)**.
- SECs are **cuts**, because they cut a part of the solution space off.
- Three types of SECs:

$$(SEC1) \quad \sum_{\substack{i \in S, j \notin S \\ (i,j) \in A}} x_{ij} \geq 1 \quad \forall \emptyset \subset S \subseteq N$$

$$(SEC2) \quad \sum_{\substack{i, j \in R \\ (i,j) \in A}} x_{ij} \leq |R| - 1 \quad \forall \emptyset \subset R \subseteq \{2, 3, \dots, n\}$$

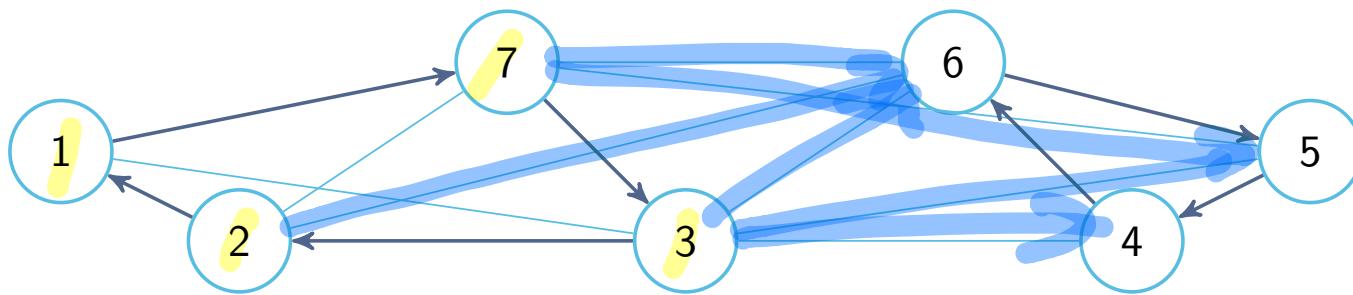
$$(SEC3) \quad z_i - z_j + nx_{ij} \leq n - 1 \quad \forall i, j := 2, \dots, n \quad (i \neq j)$$

# Subtour Elimination Constraints

$$(SEC1) \quad \sum_{\substack{i \in S, j \notin S \\ (i,j) \in A}} x_{ij} \geq 1 \quad \forall \emptyset \subset S \subseteq N$$

subset of   
 set of nodes  $N$

For each  $S \subset N$  the nodes in  $S$  need to be connected to the nodes not in  $S$ . Thus, subtours of nodes in  $S$  are excluded.



$$S = \{1, 2, 3, 7\}$$

$$\begin{aligned} x_{76} + x_{75} + x_{26} + x_{36} + x_{35} + x_{34} &\geq 1 \\ = 0 &= 0 = 0 = 0 \dots = 0 \end{aligned}$$

then this solution will be eliminated

# Subtour Elimination Constraints

$$(SEC1) \quad \sum_{\substack{i \in S, j \notin S \\ (i,j) \in A}} x_{ij} \geq 1 \quad \forall \emptyset \subset S \subseteq N$$

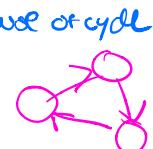
$R$ : any subset of nodes from 2 to  $n$

For each  $S \subset N$  the nodes in  $S$  need to be connected to the nodes not in  $S$ . Thus, subtours of nodes in  $S$  are excluded.

$$(SEC2) \quad \sum_{\substack{i,j \in R \\ (i,j) \in A}} x_{ij} \leq |R| - 1 \quad \forall \emptyset \subset R \subseteq \{2, 3, \dots, n\}$$

any subspace of set of nodes

↳ 1 is not included



if # of nodes = # arcs we have a cycle



arcs cut nodes: no cycle

Cycles not containing node 1 are excluded. Every subset  $R$  of  $\{2, 3, \dots, N\}$  containing  $r$  nodes can have at most  $r-1$  arcs (no cycle possible).

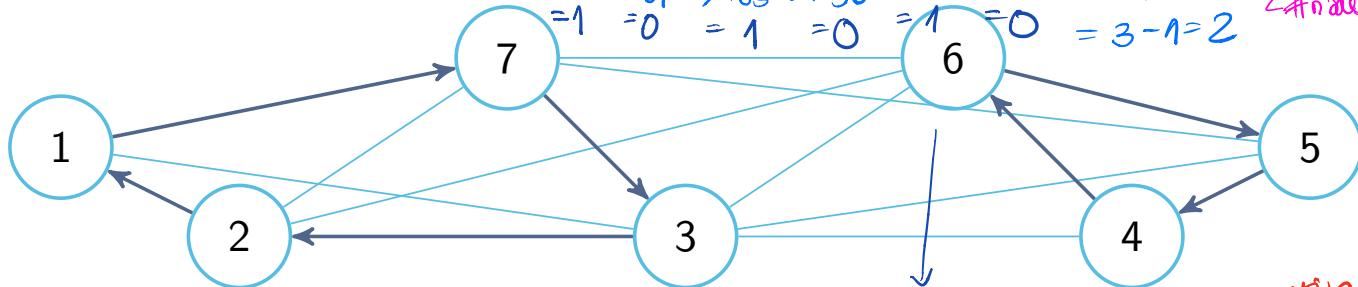
$R = \{4, 5, 6\}$  collect all  $j$  in set of  $R$

Nodes

Arches

$$x_{46} + x_{64} + x_{65} + x_{56} + x_{54} + x_{45} \leq |R| - 1$$

$$= 1 \quad = 0 \quad = 1 \quad = 0 \quad = 1 \quad = 0 \quad = 3 - 1 = 2$$



3 ≤ 2

Cut off of solution space

(3)

## Subtour Elimination Constraints

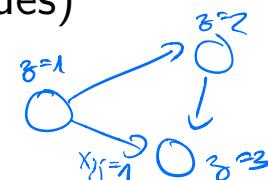
portion of route longer than number of nodes

$$(SEC3) \quad z_i - z_j + nx_{ij} \leq n - 1 \quad \forall i, j := 2, \dots, n \quad (i \neq j)$$

$z_i$  = position of node  $i$  in a route (i.e. ordering of the nodes)

This means if  $x_{ij} = 1$ , then  $z_i + 1 \leq z_j \quad (\leq n)$

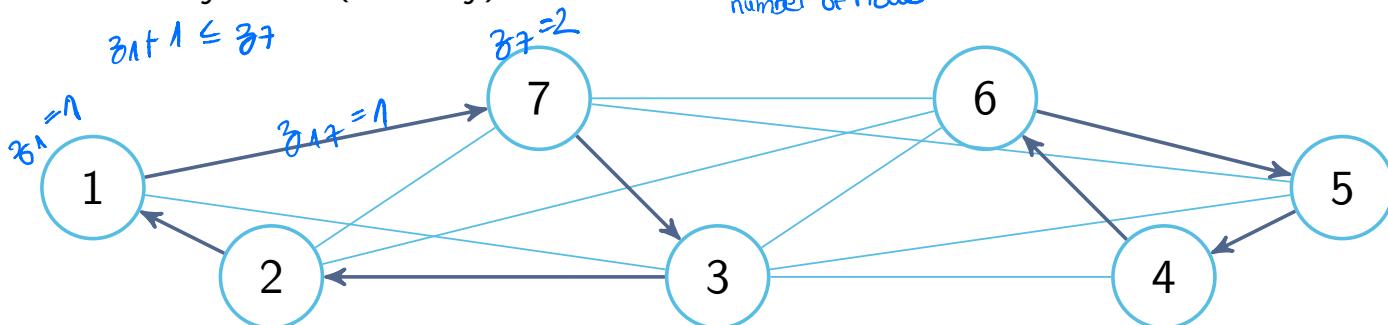
$$\Leftrightarrow z_i + 1 - z_j \leq 0 \text{ must hold.}$$



Using a big-M formulation to express the logical statement:

$$z_i + 1 - z_j \leq M(1 - x_{ij}) \text{ with } M = n.$$

*If  $x_{ij}=1$  then side=0*



If we have acycle:  $z_i + 1 \leq z_j$  ELIMINATE

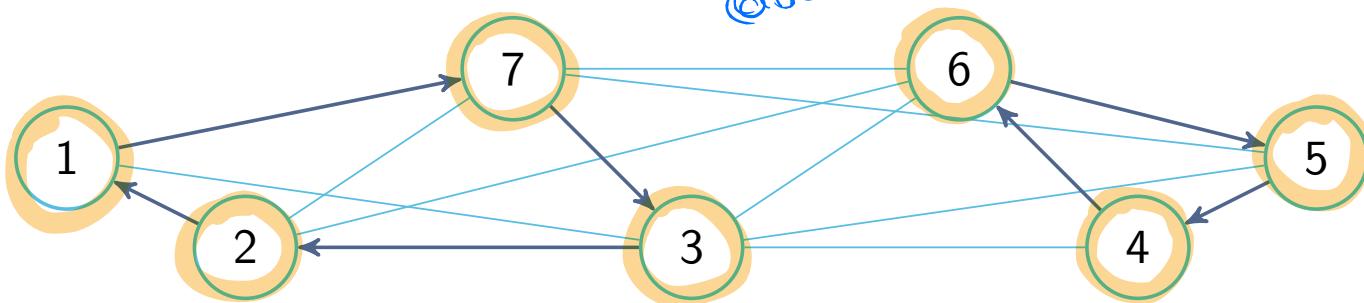
$$\sum_{j \in cycle} z_j + 1 - z_i \leq 7 \quad (1 - x_{ij}) = 1$$

# SEC comparison

	SEC1 and SEC2	SEC3
Variable	$O(n^2)$	$O(n^2 + n)$
Constraints	$O(2^n)$	$O(n^2)$

last type is better  
because smaller  
size of  
constraints

large set of  
constraints



# Solve with Branch & Cut method

Idea:

- 1 Solve MIP (without SECs) to optimality.
- 2 Analyze optimal solution:
  - 1 No subtours? → Optimal solution without subtours found.
  - 2 Otherwise, generate cuts for the subtours found (using one type of SECs).  
Solve MIP again.  
→ Back to step 2



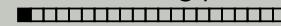
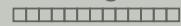
# Solve with Branch & Cut method

## Idea:

- 1 Solve MIP (without SECs) to optimality.
- 2 Analyze optimal solution:
  - 1 No subtours? → Optimal solution without subtours found.
  - 2 Otherwise, generate cuts for the subtours found (using one type of SECs).  
Solve MIP again.  
→ Back to step 2

## Task

Try yourself the TSP problem with SECs as shown in  
[https://www.gurobi.com/documentation/9.0/examples/tsp\\_py.html](https://www.gurobi.com/documentation/9.0/examples/tsp_py.html).



# Vehicle routing problem

# Introduction

Generalization of TSP

Also known as Capacitated Vehicle Routing Problem (CVRP)



# Introduction

Generalization of TSP

Also known as Capacitated Vehicle Routing Problem (CVRP)

## Goal:

A set of customers is supplied by a set of vehicles at minimal transportation cost

- All vehicles start at the depot.
- Each vehicle has a given capacity.
- Customer demands are known a priori.



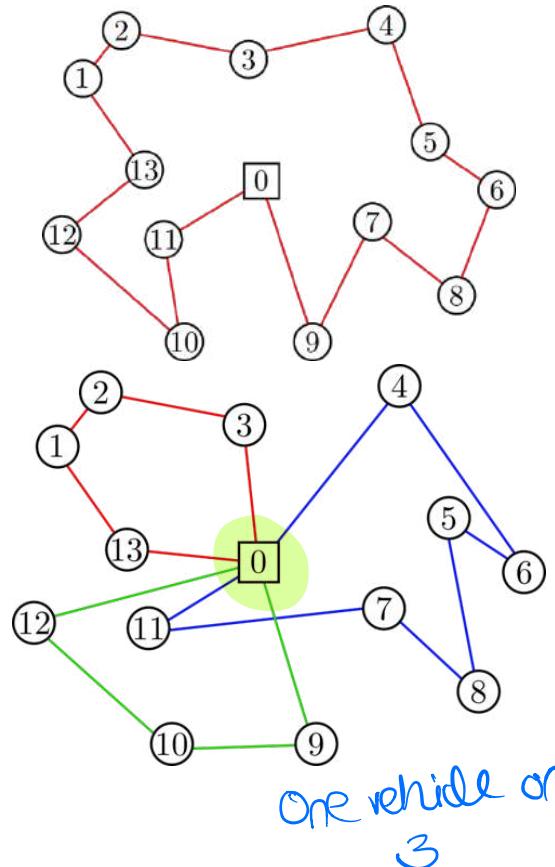
# TSP vs. VRP

TSP: Exactly one tour

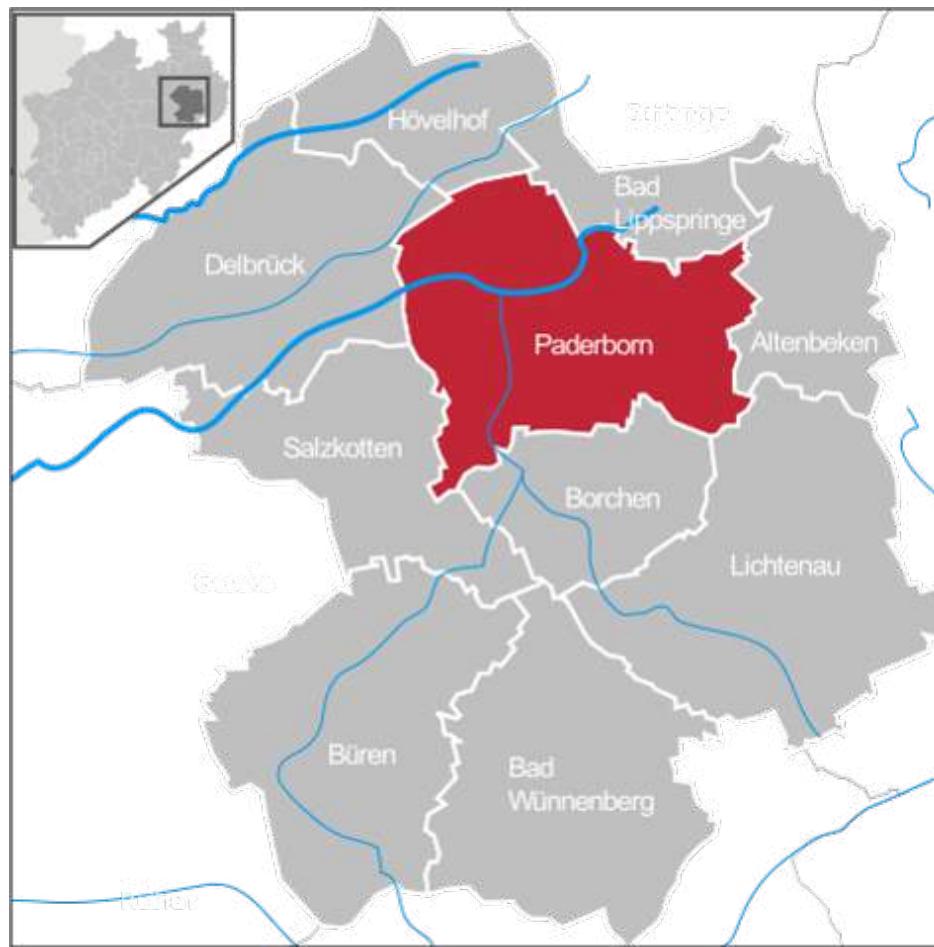
VRP: Several tours

cannot get all customers in  
one tour

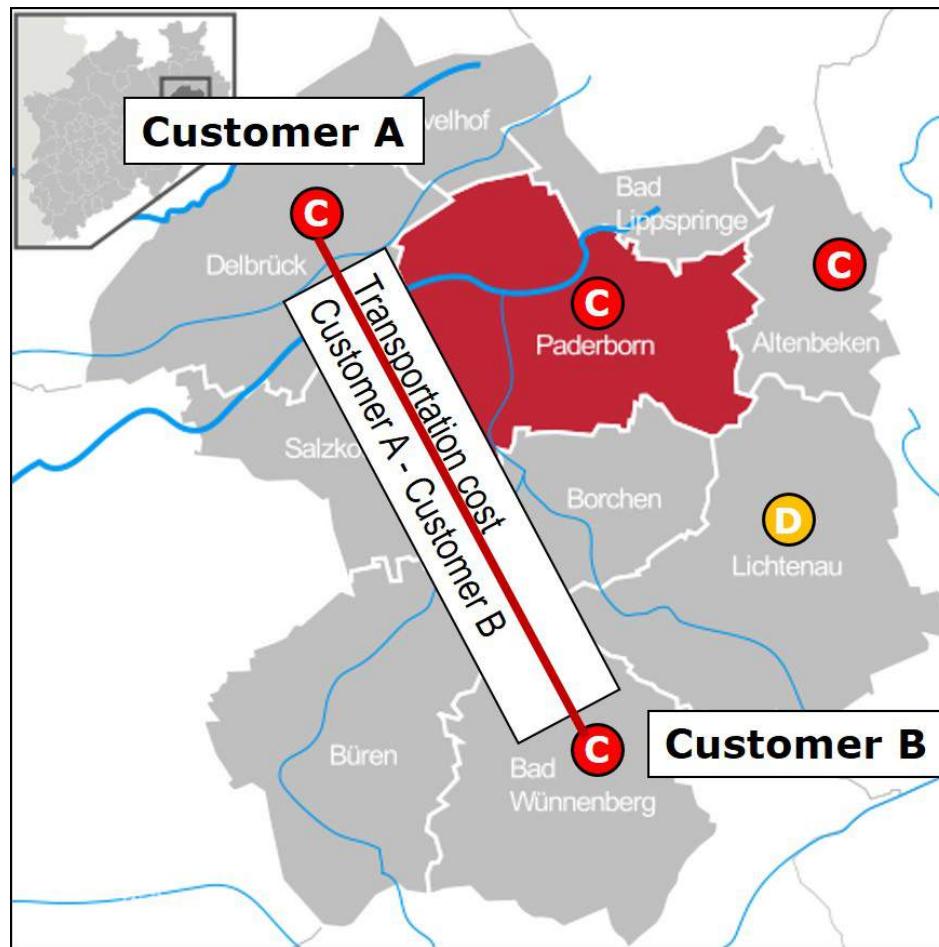
→ need several



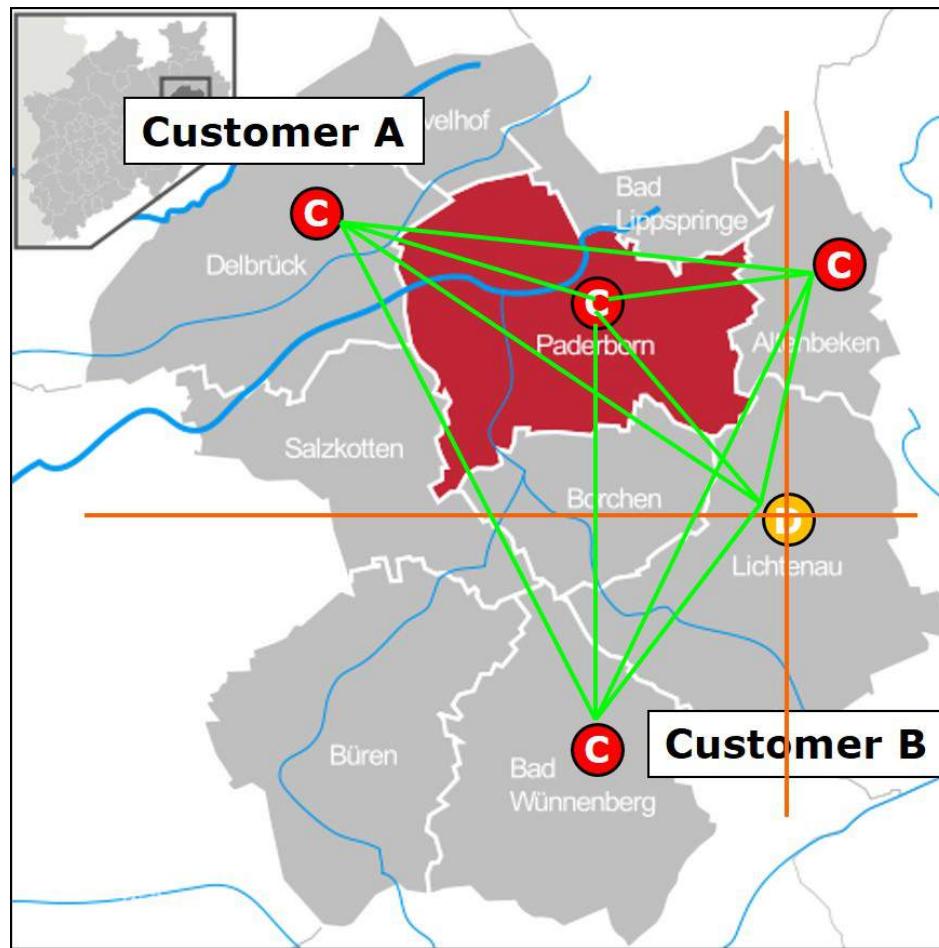
# Example



# Example



# Example



# Terminology

- **Depot:** location at which the tour starts and ends
- **Tour:** set of customers who are to receive deliveries within one trip (depot-customers-depot)
- **Route:** order in which the customers are to receive deliveries on a tour *Traveling Salesman Problem*
- **Route plan:** a feasible solution to the vehicle routing problem; a set of tours and the corresponding routes, which contains each customer exactly once and does not violate the restrictions (vehicle capacities or route length)
- **Optimal route plan:** a plan with minimum length of tours

# Three-indexed vehicle flow formulation

## ■ Sets:

Set of node  $V = \{0, \dots, n\}$ , Customers= $\{1, \dots, n\}$ ,  
Depot= $\{0\}$ , Set of arcs  $A \subseteq V \times V$

## ■ Parameters:

$d_i$ : Demand of customer  $i \in V$

$c_{ij}$ : Cost of traveling arc  $(i, j) \in A$  (e.g. driving time)

$K$ : Number of vehicles

$C$ : Capacity of a vehicle

# Three-indexed vehicle flow formulation

$$y_{ik} = \begin{cases} 1, & \text{if node } i \text{ is assigned to vehicle } k \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ijk} = \begin{cases} 1, & \text{if vehicle } k \text{ travels directly from node } i \text{ to node } j \\ 0, & \text{otherwise} \end{cases}$$

# Three-indexed vehicle flow formulation

$$y_{ik} = \begin{cases} 1, & \text{if node } i \text{ is assigned to vehicle } k \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ijk} = \begin{cases} 1, & \text{if vehicle } k \text{ travels directly from node } i \text{ to node } j \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ik} = \sum_{j \in V} x_{ijk}$$

relationship  
between 2  
variables: Use it to  
replace stuff

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} \sum_{k=1}^K x_{ijk} \quad (\text{minimize transportation cost})$$

$$s.t. \sum_{k=1}^K y_{ik} = 1 \quad \forall i \in V \setminus \{0\} \quad (\text{each customer is assigned to one vehicle})$$

$$\sum_{k=1}^K y_{0k} = K \quad (\text{all vehicles are assigned to the depot})$$

# Three-indexed vehicle flow formulation

for each node: if one Vehicle has visited  
node: it has exactly 1 income and 1 outgoing

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} \quad \forall i \in V, k = 1, \dots, K \quad (\text{vehicle flow})$$

$$\sum_{i \in V} d_i y_{ik} \leq C \quad \forall k = 1, \dots, K \quad (\text{vehicle capacity})$$

$i \in V$  sum of all nodes visited by this  
Vehicle: this delivery smaller equal  
capacity

# Three-indexed vehicle flow formulation

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} \quad \forall i \in V, k = 1, \dots, K \quad (\text{vehicle flow})$$

$$\sum_{i \in V} d_i y_{ik} \leq C \quad \forall k = 1, \dots, K \quad (\text{vehicle capacity})$$

eliminate subtours: 2nd type of constraint

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \quad \forall S \subseteq V \setminus \{0\}, |S| \geq 1, k = 1, \dots, K \quad (\text{SEC2})$$

for each subset S: We Need to avoid the cycle

# Three-indexed vehicle flow formulation

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} \quad \forall i \in V, k = 1, \dots, K \quad (\text{vehicle flow})$$

$$\sum_{i \in V} d_i y_{ik} \leq C \quad \forall k = 1, \dots, K \quad (\text{vehicle capacity})$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \quad \forall S \subseteq V \setminus \{0\}, |S| \geq 1, k = 1, \dots, K \quad (\text{SEC2})$$

$$y_{ik} \in \{0, 1\} \quad \forall i \in V, k = 1, \dots, K$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in V, j \in V, k = 1, \dots, K$$

# Three-indexed vehicle flow formulation

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} \sum_{k=1}^K x_{ijk}$$

Can we  
reduce this  
formulation?

$$s.t. \sum_{k=1}^K y_{ik} = 1 \quad \forall i \in V \setminus \{0\}$$

$$\sum_{k=1}^K y_{0k} = K$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} \quad \forall i \in V, k = 1, \dots, K$$

$$\sum_{i \in V} d_i y_{ik} \leq C \quad \forall k = 1, \dots, K$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \quad \forall S \subseteq V \setminus \{0\}, |S| \geq 1, k = 1, \dots, K$$

$$y_{ik} \in \{0, 1\} \quad \forall i \in V, k = 1, \dots, K$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in V, j \in V, k = 1, \dots, K$$

# Three-indexed vehicle flow formulation

yes, because  
related 2 types of  
binary variables

Setting  $y_{ik} = \sum_{j \in V} x_{ijk}$  results in the following formulation:

$$\min \sum_{i \in V} \sum_{j \in V} \sum_{k=1}^K c_{ij} x_{ijk}$$

$$s.t. \sum_{k=1}^K \sum_{j \in V} x_{ijk} = 1 \quad \forall i \in V \setminus \{0\} \quad (\text{each customer once})$$

$$\sum_{j \in V} x_{0jk} = 1 \quad \forall k = 1, \dots, K \quad (\text{all vehicles leave depot})$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} \quad \forall i \in V, k = 1, \dots, K \quad (\text{vehicle flow})$$

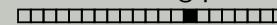
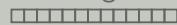
$$\sum_{i \in V} d_i \sum_{j \in V} x_{ijk} \leq C \quad \forall k = 1, \dots, K \quad (\text{vehicle capacity})$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \quad \forall S \subseteq V \setminus \{0\}, |S| \geq 1, k = 1, \dots, K \quad (\text{SEC2})$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in V, j \in V, k = 1, \dots, K$$

# Extension of vehicle routing

- An extension of VRP is the **vehicle routing problem with time windows** (VRPTW).
- A set of customers is supplied by a set of vehicles at minimal transportation cost.
  - All vehicles start at the depot.
  - Each vehicle has a given capacity.
  - Customer demands are known *a priori*.
  - **Each customer has a time window for delivery.**
  - The delivery has to **start in the time window**.
  - If the vehicle arrives too early, it has to **wait** for the time window to open.



# Extension of Vehicle routing

**TASK: Extend the VRP formulation on slide 27 to handle time windows**

# Extension of Vehicle routing

**TASK:** Extend the VRP formulation on slide 27 to handle time windows

■ Constraints:

- All vehicles start and end at the depot in a given time window.
- Each vehicle has a given capacity.
- Each customer has a demand and a time window for delivery.
- The delivery has to start in the time window. If the vehicle arrives too early, it has to wait for the time window to open.

■ Parameters:

$\underset{\text{between}}{n}$   
 $[a_i, b_i] : \text{Time window of customer } i \in V$

$t_{ij} :$  Driving time between customers  $i$  and  $j$

$s_i :$  Service time at customer  $i$

$[E, L] :$  Time window at the depot  $\{0, n + 1\}$

Vehicle should leave from Depot and go back to Depot within this time window

# Vehicle routing problem with time windows

Introduce a new variable set for the start times at the customers:

$w_{ik}$  = Start time of delivery by vehicle  $k$  at customer  $i$

# Vehicle routing problem with time windows

Introduce a new variable set for the start times at the customers:

$w_{ik}$  = Start time of delivery by vehicle  $k$  at customer  $i$

New constraints:

$$\begin{array}{c} \text{if } x_{ijk}=1 \\ \text{first visit } i \text{ and then } j, \\ \text{so if } ijk=1 \\ w_{ik} + s_i + t_{ij} \leq w_{jk} \end{array} \quad \left. \right\} \text{if } x_{ijk}=1$$

$$w_{ik} + s_i + t_{ij} - (1 - x_{ijk}) M_{ij} \leq w_{jk} \quad \forall (i, j) \in E, k = 1, \dots, K \quad (\text{start times})$$

otherwise: if  $x_{ijk} = 0$ :  
then:

$$\sum_{j \in V} x_{j,n+1,k} = 1 \quad w_{ik} + s_i + t_{ij} - M_{ij} \leq w_{jk} \quad \forall k = 1, \dots, K \quad (\text{back to depot})$$

$$a_i \left( \sum_{j \in V} x_{ijk} \right) \leq w_{ik} \leq b_i \sum_{j \in V} x_{ijk} \quad \forall i \in V \setminus \{0\}, k = 1, \dots, K \quad (\text{time windows})$$

$$E \leq w_{ik} \leq L \quad \forall i \in \{0, n+1\}, k = 1, \dots, K \quad (\text{depot time window})$$

$$w_{ik} \geq 0 \quad \begin{array}{l} \text{go back to} \\ \text{Depot latest at } L \end{array} \quad \forall i \in V, k = 1, \dots, K$$

# Vehicle routing problem with time windows

$$\min \sum_{i \in V} \sum_{j \in V} \sum_{k=1}^K c_{ij} x_{ijk}$$

$$s.t. \sum_{k=1}^K \sum_{j \in V} x_{ijk} = 1 \quad \forall i \in V \setminus \{0\}$$

$$\sum_{j \in V} x_{0jk} = \sum_{j \in V} x_{j,n+1,k} = 1 \quad \forall k = 1, \dots, K$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} \quad \forall i \in V, k = 1, \dots, K$$

$$\sum_{i \in V} d_i \sum_{j \in V} x_{ijk} \leq C \quad \forall k = 1, \dots, K$$

$$a_i \sum_{j \in V} x_{ijk} \leq w_{ik} \leq b_i \sum_{j \in V} x_{ijk} \quad \forall i \in V \setminus \{0\}, k = 1, \dots, K$$

$$E \leq w_{ik} \leq L \quad \forall i \in \{0, n+1\}, k = 1, \dots, K$$

$$w_{ik} + s_i + t_{ij} - (1 - x_{ijk}) M_{ij} \leq w_{jk} \quad \forall (i, j) \in E, k = 1, \dots, K$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in V, j \in V, k = 1, \dots, K$$



# Vehicle routing problem with time windows

$$\min \sum_{i \in V} \sum_{j \in V} \sum_{k=1}^K c_{ij} x_{ijk}$$

$$s.t. \quad \sum_{k=1}^K \sum_{j \in V} x_{ijk} = 1 \quad \forall i \in V \setminus \{0\}$$

Subtour elimination  
constraints?

$$\sum_{j \in V} x_{0jk} = \sum_{j \in V} x_{j,n+1,k} = 1 \quad \forall k = 1, \dots, K$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} \quad \forall i \in V, k = 1, \dots, K$$

$$\sum_{i \in V} d_i \sum_{j \in V} x_{ijk} \leq C \quad \forall k = 1, \dots, K$$

$$a_i \sum_{j \in V} x_{ijk} \leq w_{ik} \leq b_i \sum_{j \in V} x_{ijk} \quad \forall i \in V \setminus \{0\}, k = 1, \dots, K$$

$$E \leq w_{ik} \leq L \quad \forall i \in \{0, n+1\}, k = 1, \dots, K$$

$$w_{ik} + s_i + t_{ij} - (1 - x_{ijk}) M_{ij} \leq w_{jk} \quad \forall (i, j) \in E, k = 1, \dots, K$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in V, j \in V, k = 1, \dots, K$$



# Vehicle routing problem with time windows

$$\min \sum_{i \in V} \sum_{j \in V} \sum_{k=1}^K c_{ij} x_{ijk}$$

$$s.t. \quad \sum_{k=1}^K \sum_{j \in V} x_{ijk} = 1 \quad \forall i \in V \setminus \{0\}$$

Subtour elimination  
constraints?

$$\sum_{j \in V} x_{0jk} = \sum_{j \in V} x_{j,n+1,k} = 1 \quad \forall k = 1, \dots, K$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} \quad \forall i \in V, k = 1, \dots, K$$

$$\sum_{i \in V} d_i \sum_{j \in V} x_{ijk} \leq C \quad \forall k = 1, \dots, K$$

$$a_i \sum_{j \in V} x_{ijk} \leq w_{ik} \leq b_i \sum_{j \in V} x_{ijk} \quad \forall i \in V \setminus \{0\}, k = 1, \dots, K$$

$$E \leq w_{ik} \leq L \quad \forall i \in \{0, n+1\}, k = 1, \dots, K$$

$$w_{ik} + s_i + t_{ij} - (1 - x_{ijk}) M_{ij} \leq w_{jk} \quad \forall (i, j) \in E, k = 1, \dots, K \quad \leftarrow \text{SEC3}$$

$$x_{ijk} \in \{0, 1\} \quad \forall i \in V, j \in V, k = 1, \dots, K$$



# Vehicle routing problem with time windows

Start time constraint from VRPTW:

$$\begin{aligned} w_{ik} + s_i + t_{ij} - (1 - x_{ijk})M_{ij} &\leq w_{jk} & \forall (i, j) \in E, k = 1, \dots, K \\ \Leftrightarrow w_{ik} + s_i + t_{ij} - w_{jk} &\leq (1 - x_{ijk})M_{ij} & \forall (i, j) \in E, k = 1, \dots, K \end{aligned}$$

# Vehicle routing problem with time windows

Start time constraint from VRPTW:

$$\begin{aligned} w_{ik} + s_i + t_{ij} - (1 - x_{ijk})M_{ij} &\leq w_{jk} \quad \forall (i, j) \in E, k = 1, \dots, K \\ \Leftrightarrow w_{ik} + s_i + t_{ij} - w_{jk} &\leq (1 - x_{ijk})M_{ij} \quad \forall (i, j) \in E, k = 1, \dots, K \end{aligned}$$

Subtour elimination constraint – type 3:

$$\begin{aligned} z_i - z_j + nx_{ij} &\leq n - 1 \quad \forall i, j := 2, \dots, n \ (i \neq j) \quad (\text{SEC3}) \\ \Leftrightarrow z_i - z_j &\leq n - nx_{ij} - 1 \quad \forall i, j := 2, \dots, n \ (i \neq j) \\ \Leftrightarrow z_i - z_j &\leq (1 - x_{ij})n - 1 \quad \forall i, j := 2, \dots, n \ (i \neq j) \end{aligned}$$

# Vehicle routing problem with time windows

Start time constraint from VRPTW:

$$\begin{aligned} w_{ik} + s_i + t_{ij} - (1 - x_{ijk})M_{ij} &\leq w_{jk} \quad \forall (i, j) \in E, k = 1, \dots, K \\ \Leftrightarrow w_{ik} + s_i + t_{ij} - w_{jk} &\leq (1 - x_{ijk})M_{ij} \quad \forall (i, j) \in E, k = 1, \dots, K \end{aligned}$$

Subtour elimination constraint – type 3:

$$\begin{aligned} z_i - z_j + nx_{ij} &\leq n - 1 \quad \forall i, j := 2, \dots, n \ (i \neq j) \quad (\text{SEC3}) \\ \Leftrightarrow z_i - z_j &\leq n - nx_{ij} - 1 \quad \forall i, j := 2, \dots, n \ (i \neq j) \\ \Leftrightarrow z_i - z_j &\leq (1 - x_{ij})n - 1 \quad \forall i, j := 2, \dots, n \ (i \neq j) \end{aligned}$$



visiting number of node i

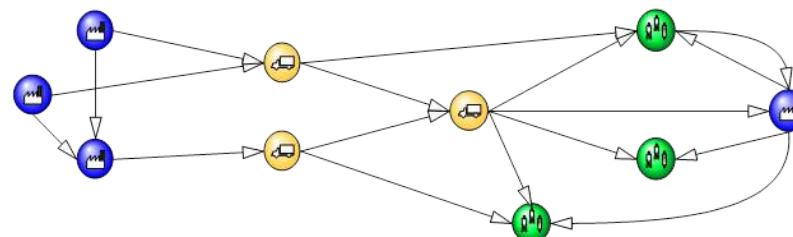
$$\begin{aligned} z_i - w_{ik} + \underbrace{s_i + t_{ij}}_{\text{constant}} - w_{jk} &\leq (1 - x_{ij})n - 1 \quad \forall i, j := 2, \dots, n \ (i \neq j) \\ &\leq (1 - x_{ijk})M_{ij} \quad \forall (i, j) \in E, k = 1, \dots, K \end{aligned}$$

if Vehicle arrives too early it has to wait for the time window to open: So left side could be smaller!

number of nodes

# Further variants of the VRP

<b>Multi depot VRP</b>	(MDVRP)	multiple depots
<b>Heterogeneous VRP</b>	(HVRP)	different vehicle types
<b>Period VRP</b>	(PVRP)	multiple time slots for planning
<b>Inventory routing</b>	(IRP)	consideration of customer inventory levels
<b>Dynamic VRP</b>	(DVRP)	changes of routes during execution
<b>Arc routing</b>	(ARP)	visit all arcs/edges instead of vertices
<b>Consistent VRP</b>	(ConVRP)	multiperiod, no changes in drivers, and avoid fluctuations in start times
<b>Pickup &amp; delivery</b>	(PDP)	pick up goods at locations and deliver them afterwards to customers



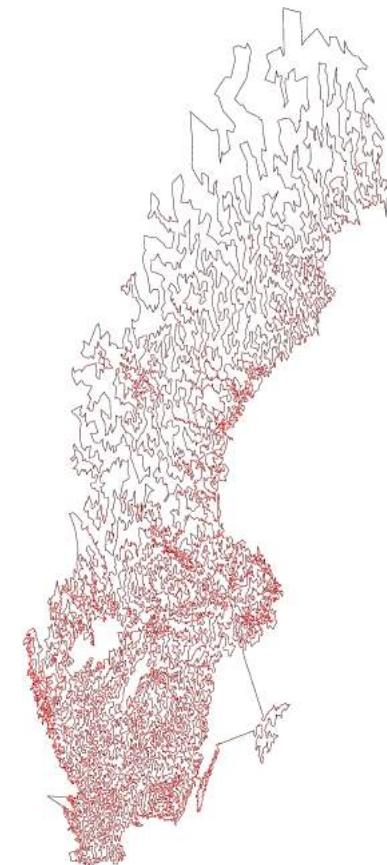
# Solution methods

VRP and its variants are **NP-hard**

⇒ solution methods are often based on  
**(meta)heuristics**, e.g.:

- Large neighborhood search (LNS)
- Variable neighborhood search (VNS)
- Ant colony optimization (ACO).

⇒ lectures later on



# Application areas



VRP and its variants are a part of many real-world problems in:

- logistics
- package delivery
- delivery services
- routing of home care nurses
- and many more ...

## Further reading

### TSP and subtour elimination constraints

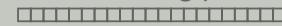
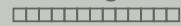
Suhl L, Mellouli T (2013) *Optimierungssysteme – Modelle, Verfahren, Software, Anwendungen*. 3rd ed., Springer Gabler, pp. 238–246.  
pp. 238-246

### VRP and VRPTW

Toth P, Vigo D (2002) *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications  
pp. 11–23, pp. 158–159

### Overview of VRP variants

Golden B, Raghavan S, Wasil E (2008) *The Vehicle Routing Problem: Latest Advances and New Challenges*, Springer



## Conclusion and outlook

# Conclusion

## What we did:

- We learned about and modeled the traveling sales problem.
- We learned about and modeled the vehicle routing problem and its variations.

## Next week we will learn:

- Max flow / min cut.

# Thank you for your attention!

Leuphana University of Lüneburg  
Business information systems, in particular Operations Research  
Prof. Dr. Lin Xie  
Universitätsallee 1  
Building 4, Room 314  
21335 Lüneburg  
Phone +49 4131 677 2305  
Fax +49 4131 677 1749  
[xie@leuphana.de](mailto:xie@leuphana.de)



Soft constraint

$$\underbrace{a + bx \leq c}_{\text{soft constraint}}$$

$$a + bx - d \leq c$$

objective:  $z = \text{profit} - wst + \alpha$

Punishment costs

} For homework  
NR.1

- ① Multi commodity Time
- ② Next Week: get it next week

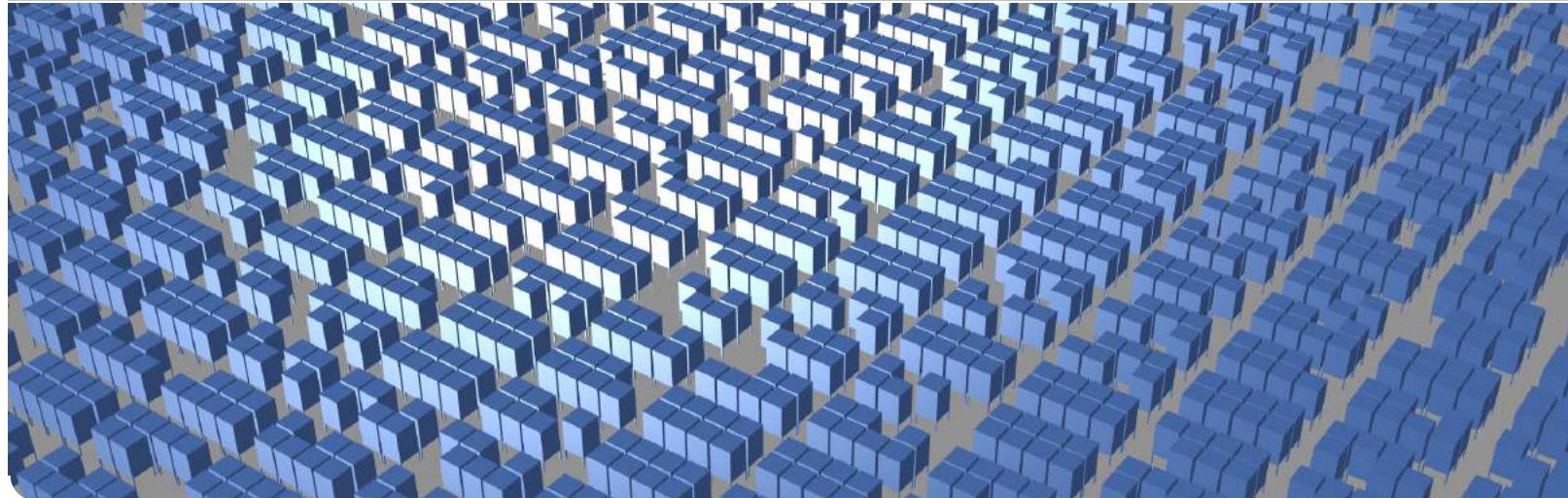


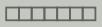
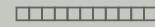
# Analysing Networks with OR-Methods

Part 5 – Max flow / mit cut

Lin Xie | 11.05.2021

PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH

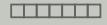
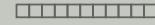




## 1 Polynomial time network algorithms

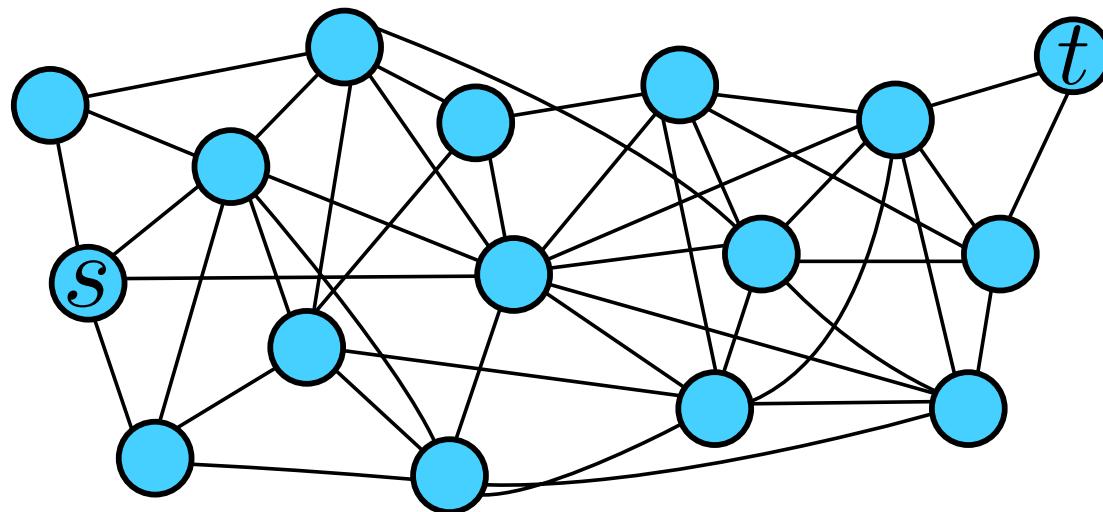
## 2 Max flow

## 3 Min cut



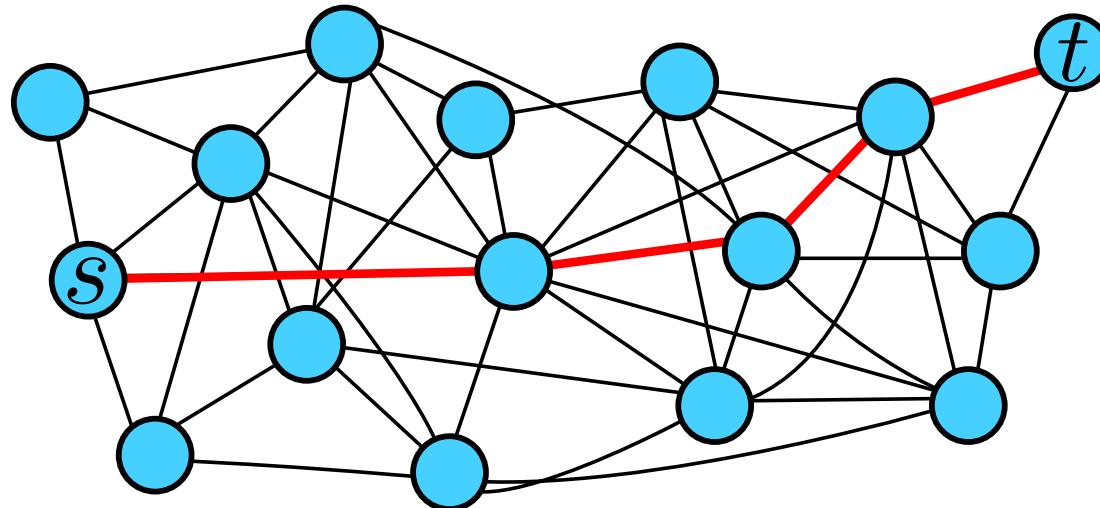
# Polynomial time network algorithms

# (Some) Polynomial time network algorithms



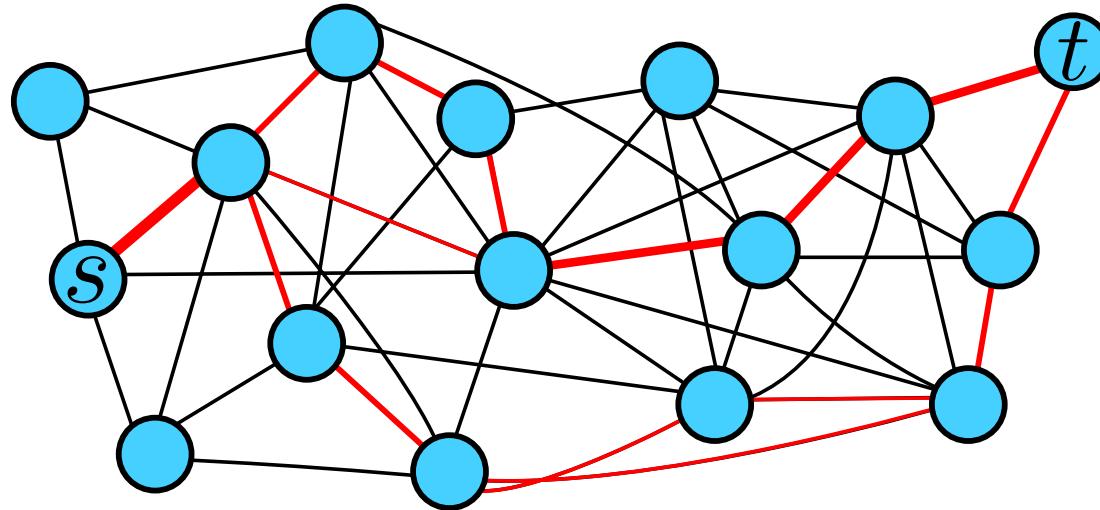
# (Some) Polynomial time network algorithms

- Shortest path problem



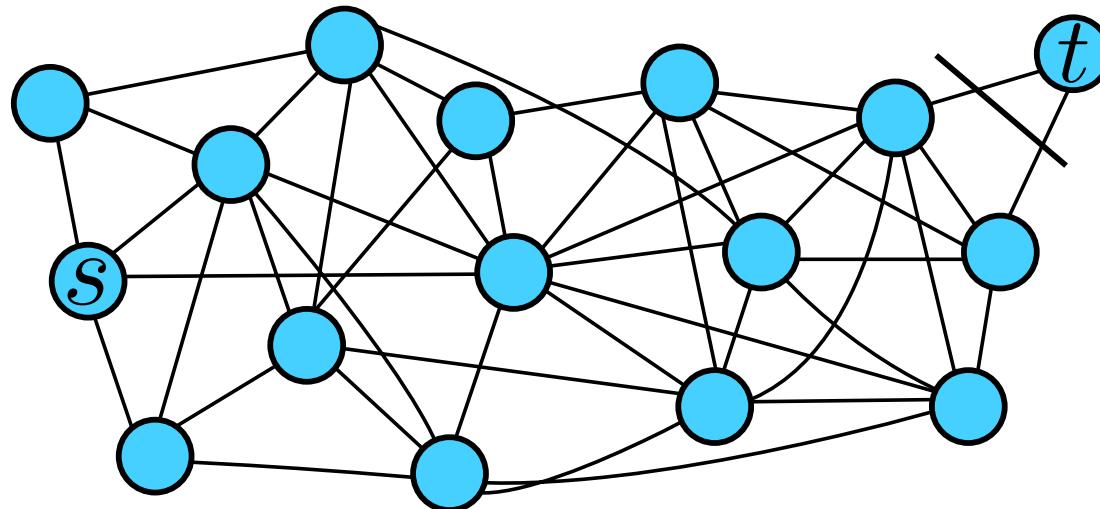
# (Some) Polynomial time network algorithms

- ## ■ Max flow problem



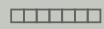
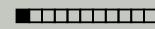
# (Some) Polynomial time network algorithms

- Min cut problem



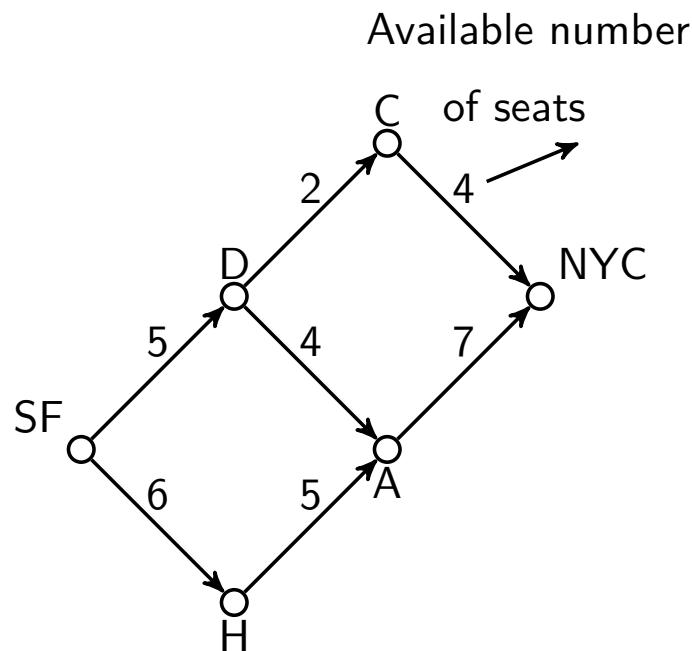
# Why not model everything as an LP?

- Some problems do not have linear objectives
- Many problems can be solved faster with specialized algorithms
- Knowing these algorithms can inspire heuristics for harder problems



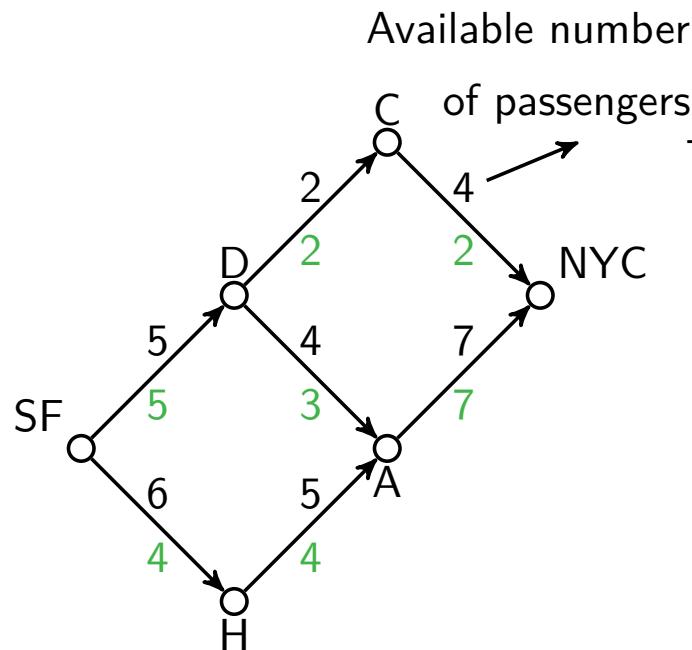
# Max flow

# The travelers' example



- Find the maximum number of passengers from San Francisco to reach New York City.
- Direct flights are booked up.
- What is the optimal solution?

# The travelers' example



Total 9 passengers from SF to NYC

- 3 passengers fly SF → D → A → NYC
- 4 passengers fly SF → H → A → NYC
- 2 passengers fly SF → D → C → NYC

# Max flow problem

- The max flow problem tries to maximize the flow from a source  $s$  to a sink  $t$ .
- Each arc  $(i,j)$  will carry a positive upper bound  $u_{ij}$ .
- Costs of arcs are not taken into account.
- There is no demand associated with nodes.

Examples: maximum flow of

- oil in a pipeline network
- cars in a traffic network
- data packages in a communication network

# Max flow problem

## Parameters:

- Graph  $G = (V, A)$
- Arc capacities  $u_{ij}$
- Flow source  $s$  and sink  $t$

## Decision variables:

- Amount of flow  $x_{ij}$  on arc  $(i, j)$

## Objective and constraints:

$$\max \sum_{(s,j) \in A} x_{sj}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in V \setminus \{s, t\}$$

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i,j) \in A$$

Note the lack of arc costs!

# Dinitz's Algorithm (Dinic's algorithm)

skipped!

## ■ Residual capacity

If  $(i, j) \in A$ :  $r_{ij} = u_{ij} - x_{ij}$ ;  $r_{ji} = x_{ij}$

Else:  $r_{ij} = 0$

# Dinitz's Algorithm (Dinic's algorithm)

## ■ Residual capacity

If  $(i, j) \in A$ :  $r_{ij} = u_{ij} - x_{ij}$ ;  $r_{ji} = x_{ij}$

Else:  $r_{ij} = 0$

## ■ The **residual graph** $G^R = (V, A^R)$ with $A^R = \{(i, j) \in A \mid r_{ij} > 0\}$

# Dinitz's Algorithm (Dinic's algorithm)

- **Residual capacity**

If  $(i, j) \in A$ :  $r_{ij} = u_{ij} - x_{ij}$ ;  $r_{ji} = x_{ij}$

Else:  $r_{ij} = 0$

- The **residual graph**  $G^R = (V, A^R)$

with  $A^R = \{(i, j) \in A \mid r_{ij} > 0\}$

- An **augmenting path** is a path from  $s$  to  $t$  in  $G^R$

# Dinitz's Algorithm (Dinic's algorithm)

- **Residual capacity**

If  $(i, j) \in A$ :  $r_{ij} = u_{ij} - x_{ij}$ ;  $r_{ji} = x_{ij}$

Else:  $r_{ij} = 0$

- The **residual graph**  $G^R = (V, A^R)$

with  $A^R = \{(i, j) \in A \mid r_{ij} > 0\}$

- An **augmenting path** is a path from  $s$  to  $t$  in  $G^R$

- The **layered graph**  $G^L = (V, A^L)$ .

$d_i$  is the length of the shortest path between  $s$  and  $i$  in  $G^R$

$A^L = \{(i, j) \in A^R \mid d_i = d_j + 1\}$

# Dinitz's Algorithm (Dinic's algorithm)

- **Residual capacity**

If  $(i, j) \in A$ :  $r_{ij} = u_{ij} - x_{ij}$ ;  $r_{ji} = x_{ij}$   
 Else:  $r_{ij} = 0$

- The **residual graph**  $G^R = (V, A^R)$

with  $A^R = \{(i, j) \in A \mid r_{ij} > 0\}$

- An **augmenting path** is a path from  $s$  to  $t$  in  $G^R$

- The **layered graph**  $G^L = (V, A^L)$ .

$d_i$  is the length of the shortest path between  $s$  and  $i$  in  $G^R$

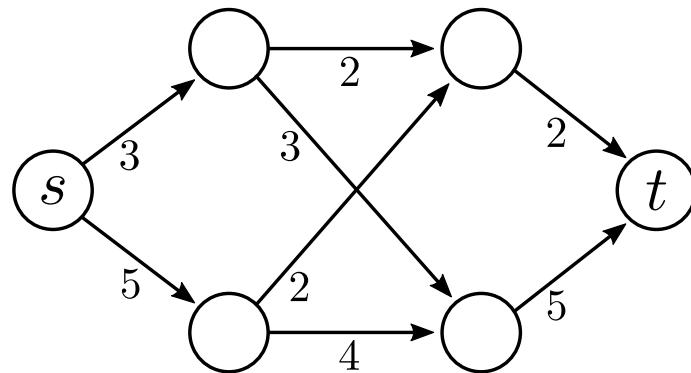
$A^L = \{(i, j) \in A^R \mid d_i = d_j + 1\}$

- A **blocking flow** is a flow  $x_{ij}$  such that the graph  $G'$  contains no path between  $s$  and  $t$ , where  $G' = (V, A')$  with

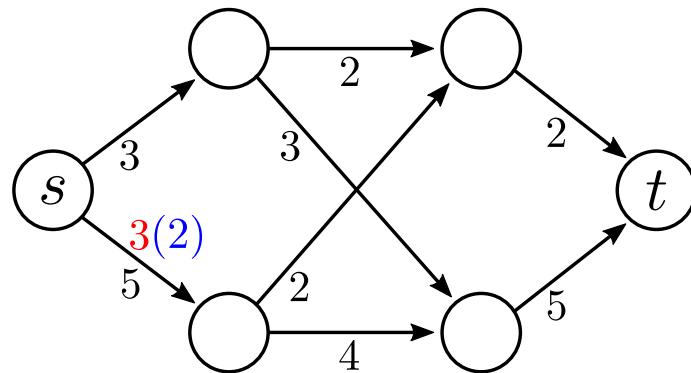
$A' = \{(i, j) \in A^L \mid x_{ij} < u_{ij}\}$

Plain text: Every  $s$ - $t$  path in  $G$  has at least one saturated arc.

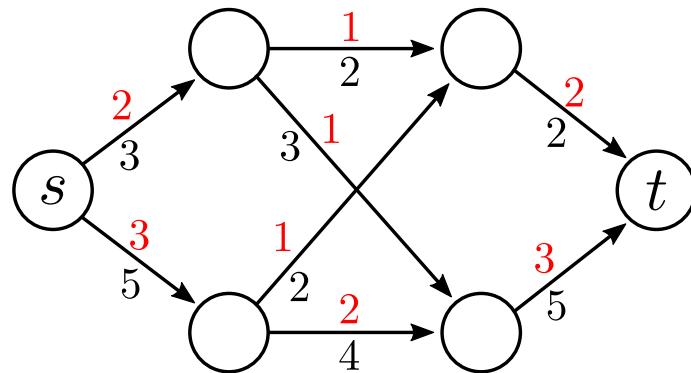
# Residual capacity



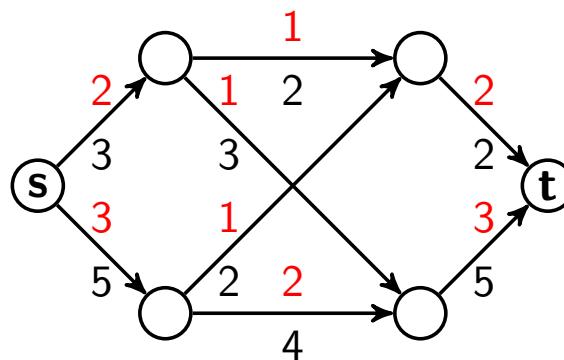
# Residual capacity



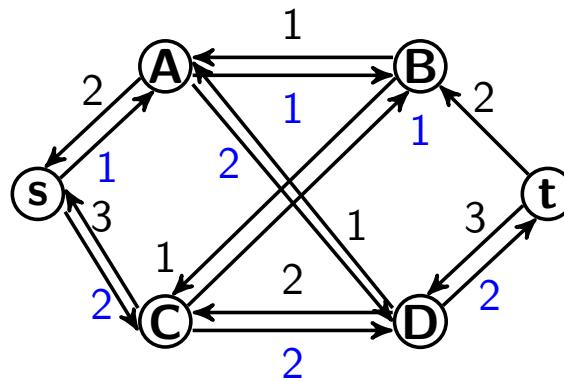
# Residual capacity



# Residual capacity



Residual graph:



# Dinitz's Algorithm

- 1 Set  $x_{ij} = 0 \quad \forall (i,j) \in A$
- 2 Construct  $G^L$ . If  $d_t = \infty$  return  $\mathbf{x}$
- 3 Find a blocking flow  $f$  in  $G^L$
- 4 Set the relevant  $x_{ij}$ 's according to  $f$  and go to step 2 ("Augmentation")

# Dinitz's Algorithm

- 1 Set  $x_{ij} = 0 \ \forall (i,j) \in A$
- 2 Construct  $G^L$ . If  $d_t = \infty$  return  $\mathbf{x}$
- 3 Find a blocking flow  $f$  in  $G^L$
- 4 Set the relevant  $x_{ij}$ 's according to  $f$  and go to step 2 ("Augmentation")

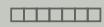
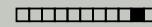
Worst case run time:  $O(V^2A)$ . Note that there is now a max flow algorithm that runs in (worst-case) time  $O(VA)$ !

# Dinitz's Algorithm

- 1 Set  $x_{ij} = 0 \ \forall (i,j) \in A$
- 2 Construct  $G^L$ . If  $d_t = \infty$  return  $\mathbf{x}$
- 3 Find a blocking flow  $f$  in  $G^L$
- 4 Set the relevant  $x_{ij}$ 's according to  $f$  and go to step 2 ("Augmentation")

Worst case run time:  $O(V^2A)$ . Note that there is now a max flow algorithm that runs in (worst-case) time  $O(VA)$ !

The key question is how to compute a blocking flow.



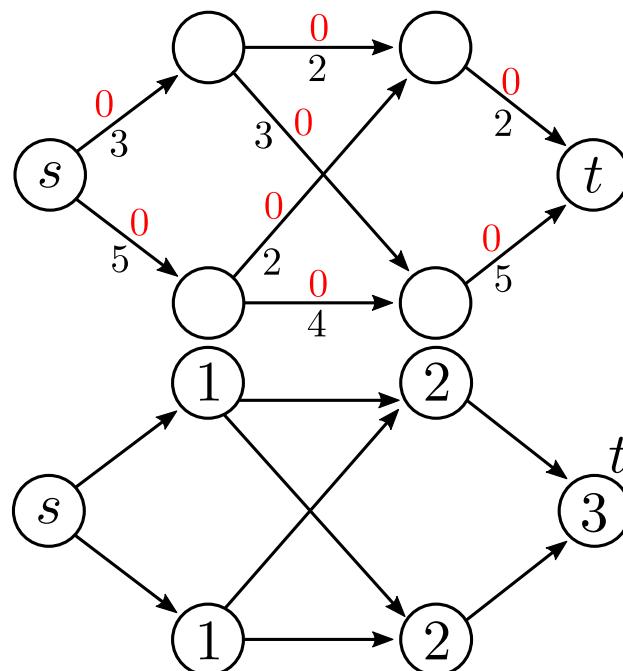
# Dinitz's Algorithm

We will use the **advance** and **retreat** algorithm (DFS) to find a blocking flow. Note that more efficient algorithms exist.

# Dinitz's Algorithm

We will use the **advance** and **retreat** algorithm (DFS) to find a blocking flow. Note that more efficient algorithms exist.

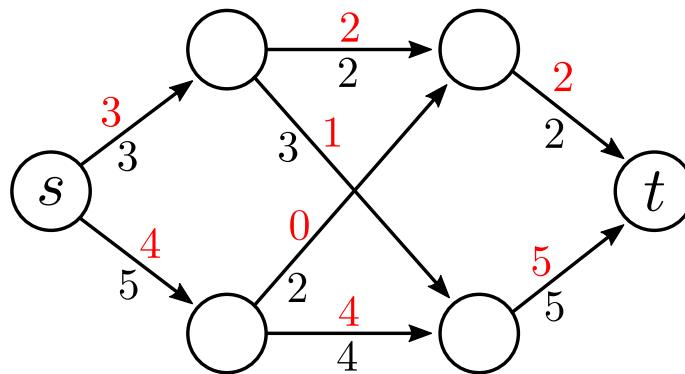
Starting point of the algorithm:

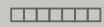
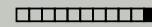


# Dinitz's Algorithm

We will use the **advance** and **retreat** algorithm (DFS) to find a blocking flow. Note that more efficient algorithms exist.

Solution:





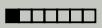
# Dinitz's Algorithm

Any ideas as to why this algorithm works? (Proof sketch)

# Dinitz's Algorithm

Any ideas as to why this algorithm works? (Proof sketch)

In each iteration at least one edge is saturated and thus removed from the residual graph. After at most  $|A|$  iterations, no edges would remain, and a maximum flow must be present.

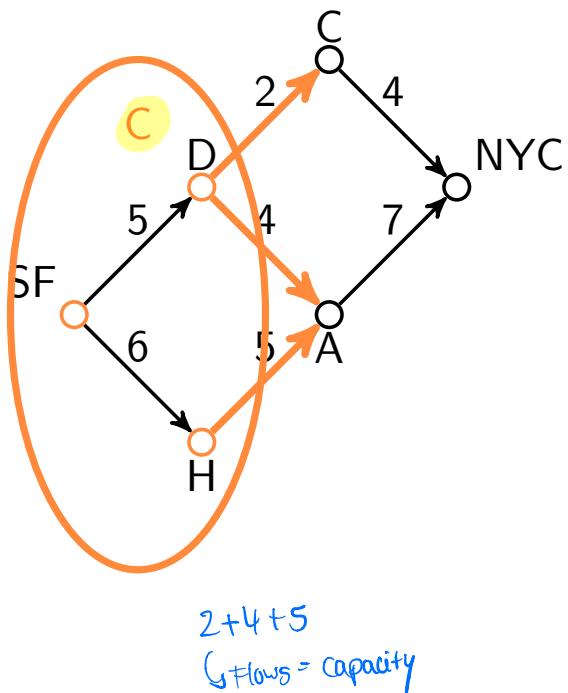


## Min cut





# Flows and cuts



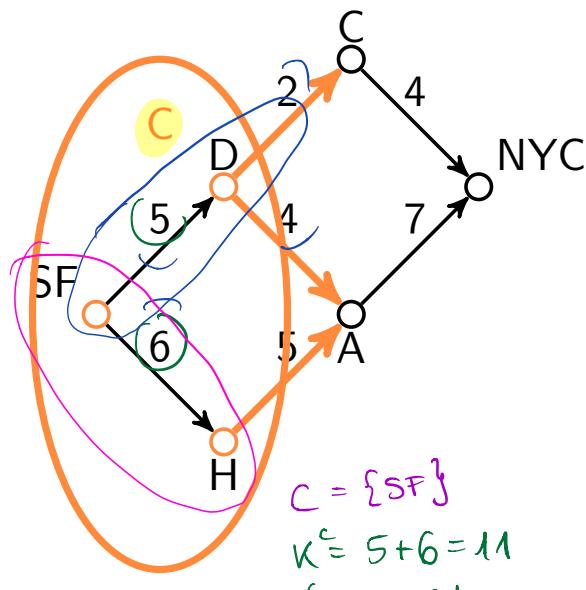
- A **cut** is a partition of node set  $N$  into two subsets  $C$  and  $\bar{C}$ , where  $C$  contains the source  $s$  but without the sink  $t$ .
- For example:  $C = \{SF, D, H\}$
- We refer to a cut as **s-t cut** if  $s \in C$  and  $t \in \bar{C}$
- **Flow** across an s-t cut  $v$ : denotes the total net flow across the cut

$$v = \sum_{j \in C, k \notin C} x_{jk} - \sum_{i \notin C, j \in C} x_{ij}$$

incoming to  $C$



# Flows and cuts



$$C = \{SF, D\}$$

$$K^C = 5 + 6 + 4 + 2 = 17$$

$$C = \{SF, H\} \quad K^C = 5 + 5 = 10$$

- **Capacity** of an s-t cut  $K^C$ : an upper bound on the maximum amount of flow we can send from the nodes in  $C$  to the nodes in  $\bar{C}$  while honoring the capacities of arcs.

$$K^C = \sum_{i \in C, j \notin C} u_{ij}$$

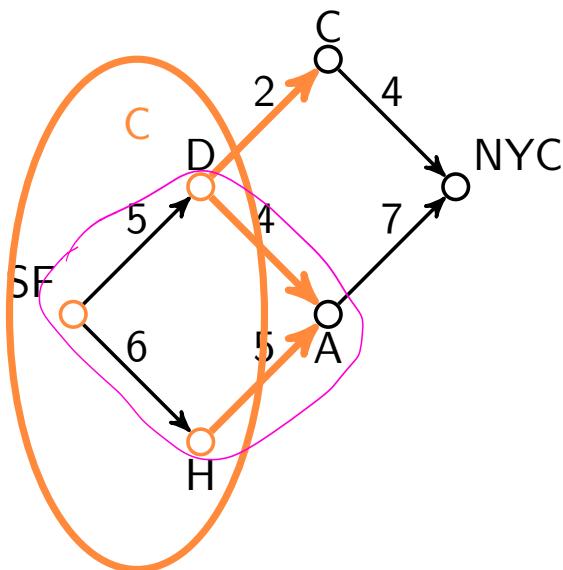
Sum of all Capacities from Arcs leaving out

- We refer to an s-t cut whose capacity is minimum among all s-t cuts as a **minimum cut**.

↳ Then we know Max cut as well



# Flows and cuts



- **Capacity** of an s-t cut  $K^C$ : an upper bound on the maximum amount of flow we can send from the nodes in  $C$  to the nodes in  $\bar{C}$  while honoring the capacities of arcs.

$$K^C = \sum_{i \in C, j \notin C} u_{ij}$$

- We refer to an s-t cut whose capacity is minimum among all s-t cuts as a **minimum cut**.

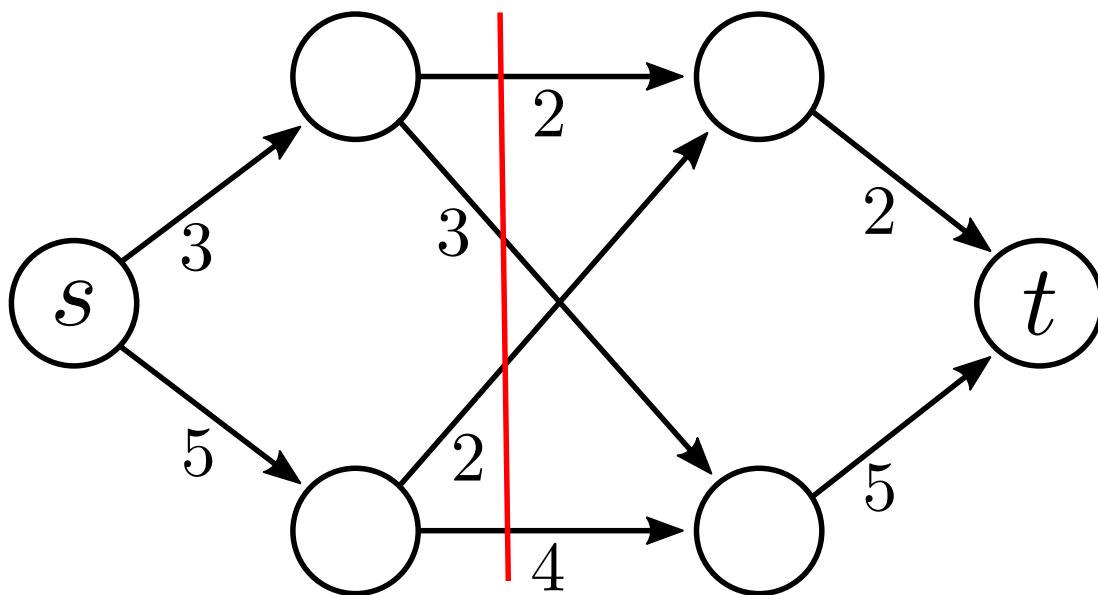
What is the minimum cut of this example on the left?

Min st-cut:  $\frac{u_C}{C} = 5 + 4 = 9$   
 $C = \{SF, D, A, H\}$

} Min cut corresponds to max Flow

# Minimum cut

$$C = \{1, 2, 3, 4\} \quad UC = 7$$



An  $s - t$  cut, but is it of minimum weight?

## Min-cut max-flow connection

Consider the capacity of the minimum cut  $K^C = \sum_{i \in C, j \notin C} u_{ij}$ .

The maximum flow is the same as the capacity of the minimum cut!

$$\max \sum_{(s,j) \in A} x_{sj} = \min K^C$$

A proof can be found in Chvatal V (1983) Linear programming, pp. 370–371  
(Max-Flow Min-Cut Theorem)

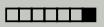
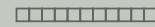
# Further reading

Ahuja et al. (2014) Network flows - Theory, Algorithms, and Application, Pearson

pp. 166–196

Chvatal V (1983) Linear programming

pp. 367–386



# Thank you for your attention!

Leuphana University of Lüneburg  
Business information systems, in particular Operations Research  
Prof. Dr. Lin Xie  
Universitätsallee 1  
Building 4, Room 314  
21335 Lüneburg  
Phone +49 4131 677 2305  
Fax +49 4131 677 1749  
[xie@leuphana.de](mailto:xie@leuphana.de)



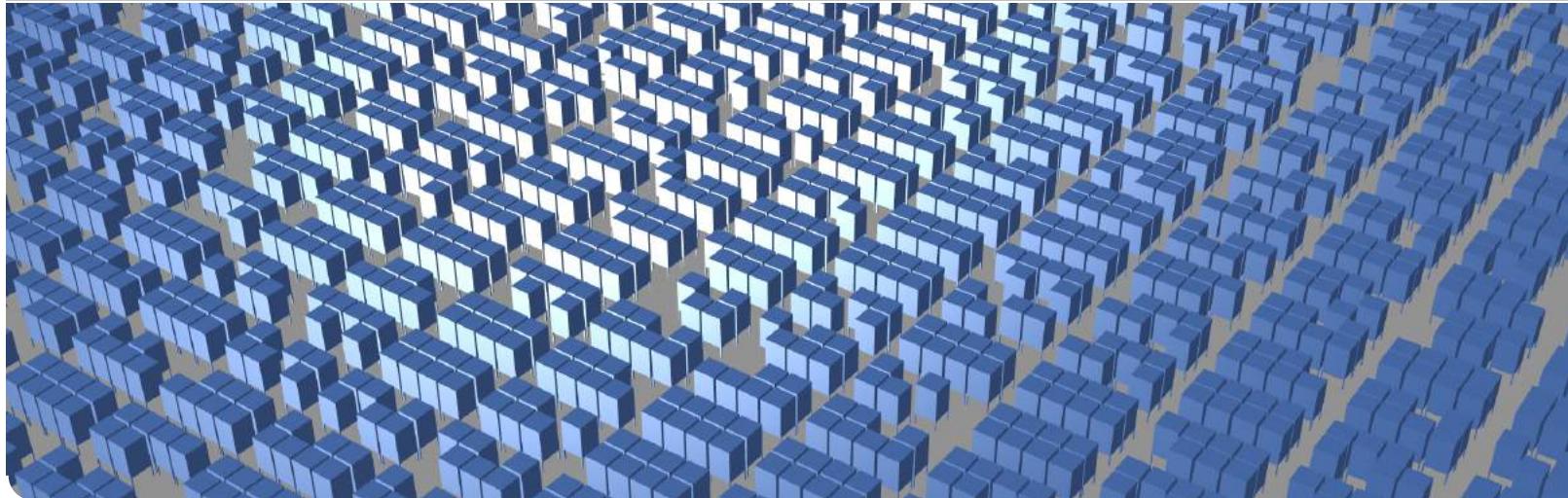


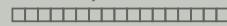
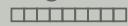
# Analysing Networks with OR-Methods

Part 6 – Modelling flows and time

Lin Xie | 25.05.2021

PROF. DR. LIN XIE - BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH

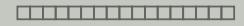
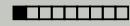




## 1 Single-commodity flows

## 2 Multicommodity flows

## 3 Time-space models



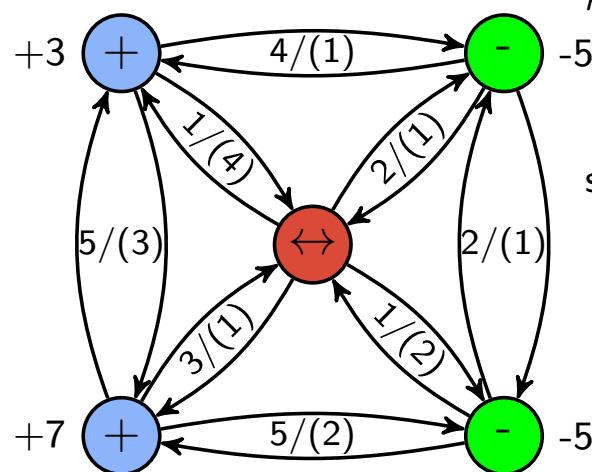
## Single-commodity flows

## Transshipment problem – Mathematical model

## Recap

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$



capacity/(cost)

$x_{ij}$  Number of units flowing on arc  $(i,j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$ : Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i,j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i,j) \in A$

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = -d_i \quad \forall i \in N^-$$

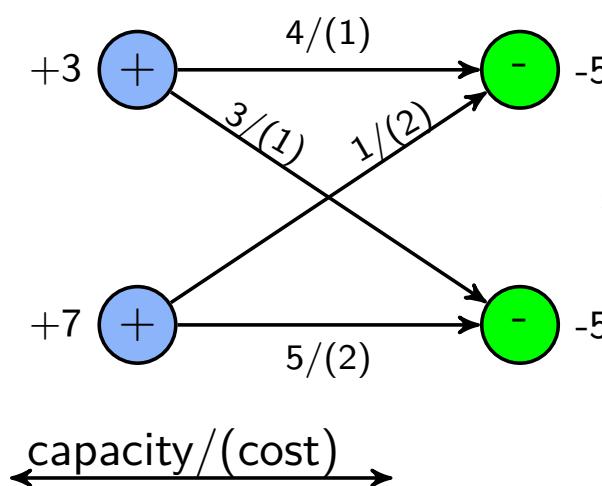
$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in N^{\leftrightarrow}$$

$$0 \leq x_{ij} \leq \kappa_{ij} \quad \forall (i,j) \in A$$

## Special case: Single-stage problem

Graph  $G = (N, A)$

$$N = N^+ \cup \cancel{N^\leftrightarrow} \cup N^-$$



$x_{ij}$  Number of units flowing on arc  $(i,j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$  Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i,j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i, j) \in A$

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = -d_i \quad \forall i \in N^-$$

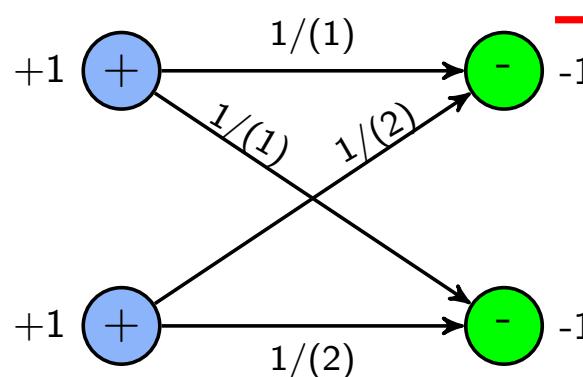
$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in N^{\leftrightarrow}$$

$$0 \leq x_{ij} \leq \kappa_{ij} \quad \forall (i,j) \in A$$

## Special case: Assignment problem

Graph  $G = (N, A)$

$$N = N^+ \cup \cancel{N^\leftrightarrow} \cup N^-$$



$$|N^+| = |N^-|$$

capacity/(cost)

$x_{ij}$  Number of units flowing on arc  $(i,j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$  Demand for units at node  $i \in N^+$

$c_{ij}$  Transportation cost per unit on arc  $(i,j) \in A$

$\kappa_{ij}$ : Maximum capacity on arc  $(i,j) \in A$

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

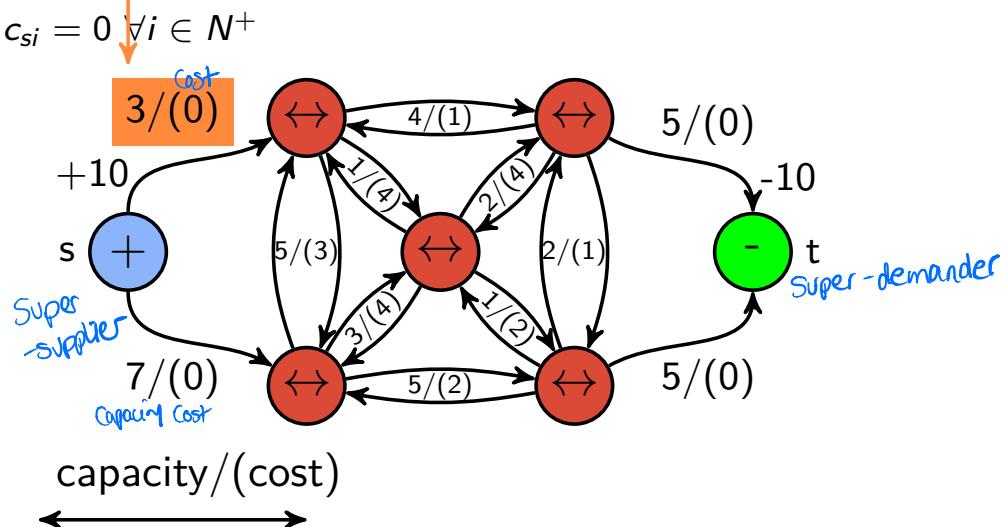
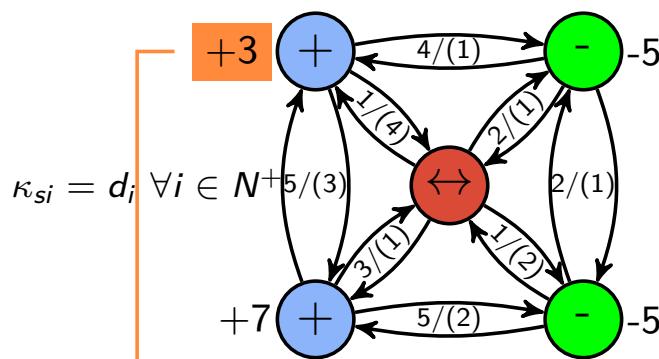
$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i^+ \quad \forall i \in N^+$$

$$\sum_{(i,i) \in A} x_{ij} - \sum_{(i,i) \in A} x_{ji} = -d_i \quad \forall i \in N^-$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in N^{\leftrightarrow}$$

$$0 \leq x_{ij} \leq \kappa_{ij} \quad \forall (i,j) \in A$$

# Special case: Min-cost-flow problem



exactly 1 supplier  
1 demander ) all other  
nodes = tranship-  
ment

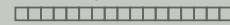
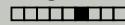
for this:

① Add 1 artificial supplier  
2 demander

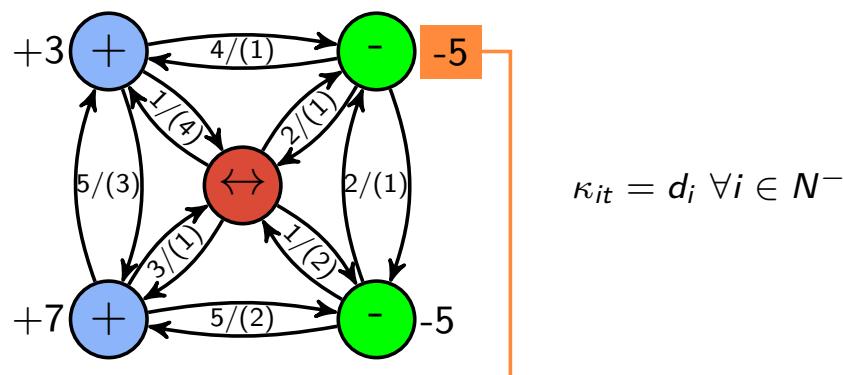
② connect artificial supplier  
to all other  
and demanders

...  
super-damander &  
super-supplier

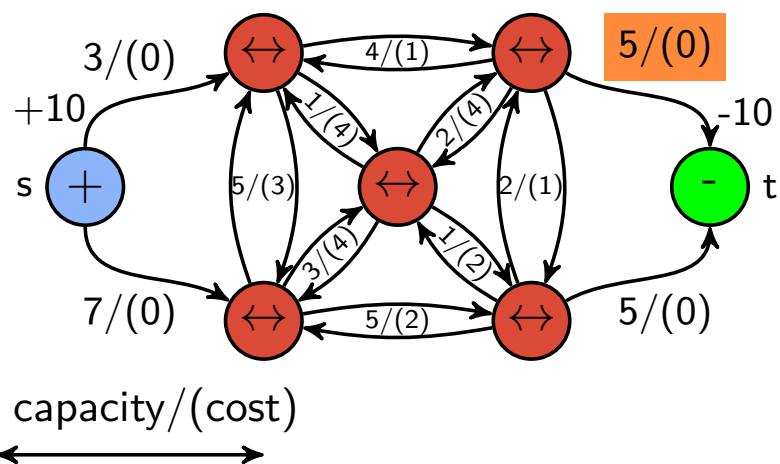
find capacity,  
supply & demand unit



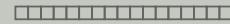
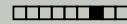
# Special case: Min-cost-flow problem



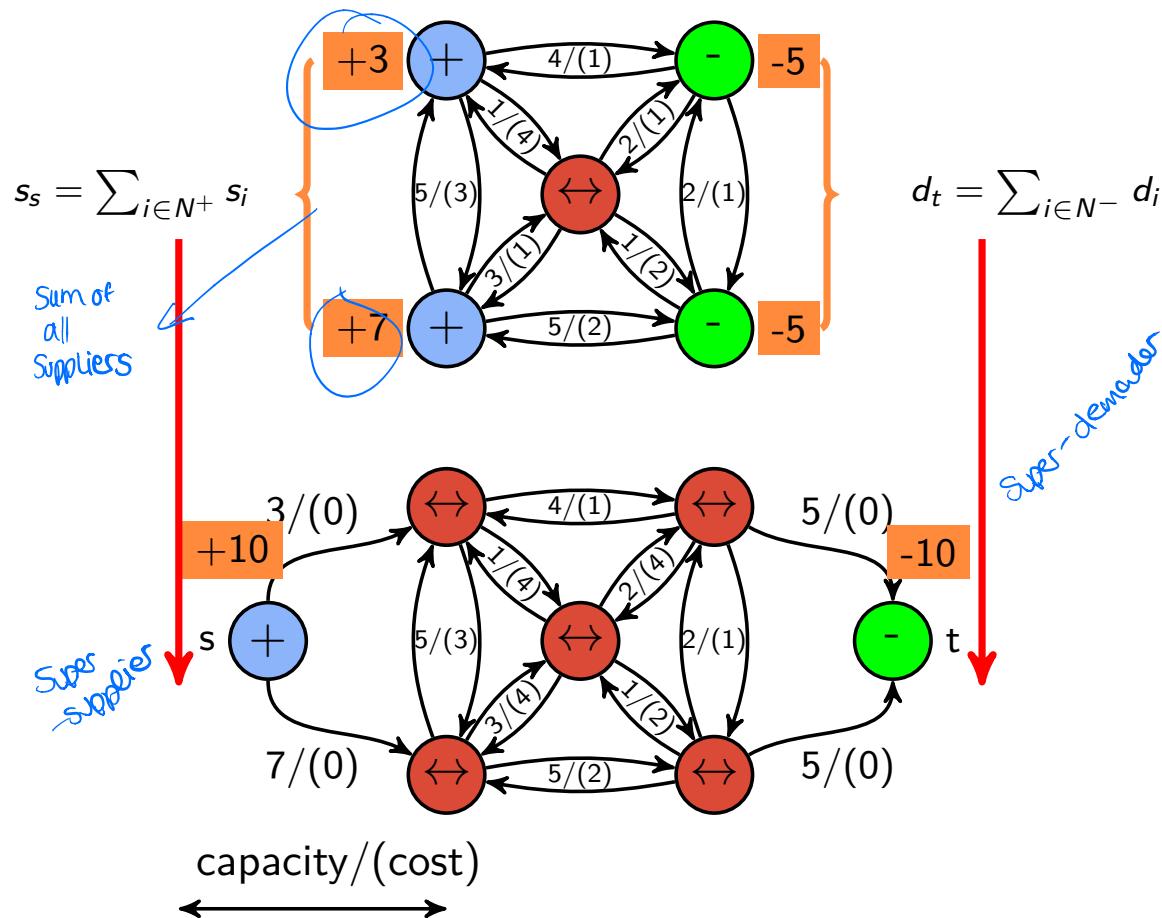
$$\kappa_{it} = d_i \quad \forall i \in N^-$$



$$c_{it} = 0 \quad \forall i \in N^-$$



# Special case: Min-cost-flow problem



## Special case: Min-cost-flow problem

Graph  $G = (N, A)$

$$N = N^+ \cup N^{\leftrightarrow} \cup N^-$$

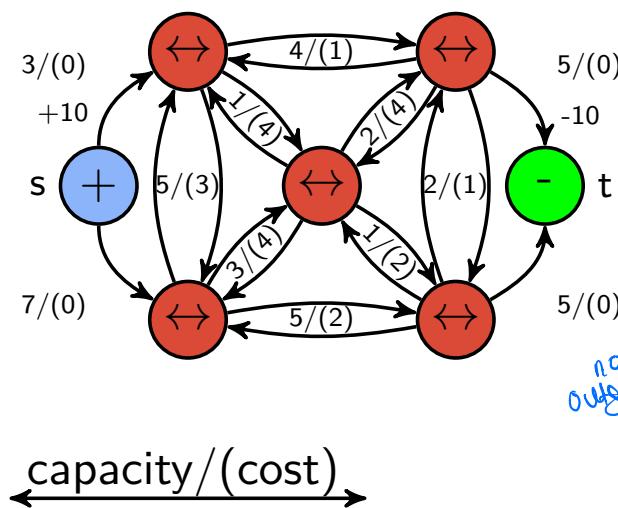
$x_{ij}$  Number of units flowing on arc  $(i, j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$ : Demand for units at node  $i \in N^-$

$c_{ij}$  Transportation cost per unit on arc  $(i,j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i,j) \in A$



$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

s.t

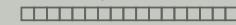
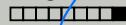
$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+ \quad \text{no } \checkmark$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = -d_i \quad \forall i \in N^-$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in N^{\leftrightarrow}$$

$$0 \leq x_{ij} \leq \kappa_{ij} \quad \forall (i,j) \in A$$

( Single Commodity Flow: Have 1 kind of good; keep flow conservation and capacity of arcs )



# Max flow problem

## Parameters:

- Graph  $G = (V, A)$
- Arc capacities  $u_{ij}$
- Flow source  $s$  and sink  $t$

for multiple:

$K$ : a set of commodity

$$x_{ij}^k$$

Recap

→ exercise slide 14

## Decision variables:

- Amount of flow  $x_{ij}$  on arc  $(i, j)$

Sum over all commodities, maximize outgoing

## Objective and constraints:

$$\max \sum_{(s,j) \in A} x_{sj}$$

$$\max \sum_{k \in K} \sum_{(s^k, j) \in A} x_{s^k j}^k$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij}^k - \sum_{(j,i) \in A} x_{ji}^k = 0$$

$$\forall i \in V \setminus \{s, t\}, \sum_{k \in K} x_{ij}^k = 0$$

$$0 \leq x_{ij}^k \leq u_{ij}$$

No negativity

$$\forall (i, j) \in A$$

We get an integer solution

Note the lack of arc costs!

$$x_{ij}^k \geq 0 \quad \forall k \in K, (i, j) \in A$$

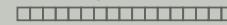
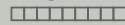
non-negative for all capacity and arc

$$\sum_{k \in K} x_{ij}^k \leq u_{ij}$$

Arc capacity



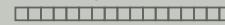
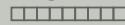
## Multicommodity flows



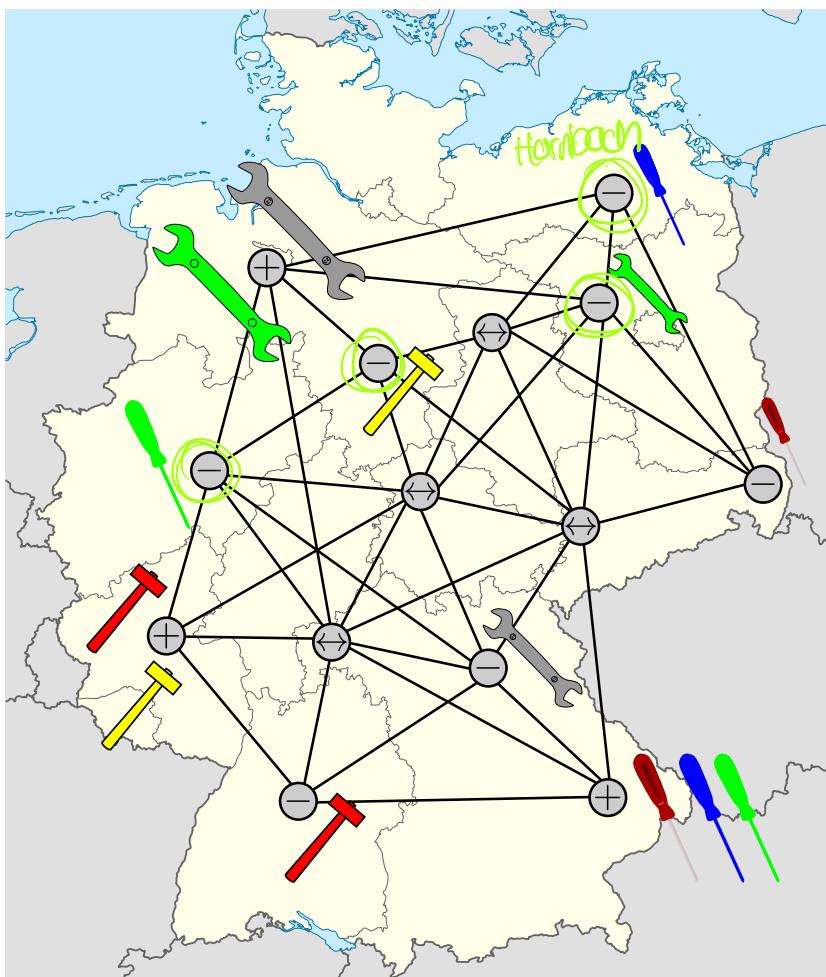
## Multicommodity flows

→ Several product types -.

- Each commodity has its own origin and destination,
- and it has a separate flow conservation constraint at each node,
- whereas all commodities share the same flow capacity constraint at each arc.



# Multicommodity flows



- 1 company produces tools
- 3 different types = 1 commodity
- Horlbeck needs to receive stock of different tools

capacity = Load of vehicle (...)

↳ we transport different tools altogether

↳ share same arc capacity

## Mathematical model: Min cost MCFP

## Parameters:

- Graph  $G = (N, A)$
  - Arc capacity  $\kappa_{ij}$  (Load vehicle)
  - Amount  $a_k$  of commodity  $k$
  - Set of commodities  $K$  (Tools)
  - Cost  $c_{ij}^k$  to send flow  $k$  on arc  $(i, j)$
  - Commodity source, sink  $s_k, t_k \in V$   
↳ depends on commodity

**Variables:**  $x_{ij}^k$  – flow of commodity  $k$  on  $(i, j)$

**Objective and constraints:**  $\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k$

$$\sum_{(i,j) \in A} x_{ij}^k - \sum_{(j,i) \in A} x_{ji}^k = \begin{cases} a_k & \text{if } s_k = i \\ -a_k & \text{if } t_k = i \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, i \in V$$

trans  
shipment  
nodes      If have flown arc: sum  
capacity      check for each arc

$$\sum_{k \in K} x_{ij}^k \leq \kappa_{ij} \quad \forall (i, j) \in A$$

$$0 \leq x_{ij}^k \leq a_k \quad \forall k \in K, (i, j) \in A$$

keep upper band of each arc

# Mathematical model: Max flow MCFP

## Maximum flow MCFP

Adjust the previous model to **create an MCFP that maximizes the total flow.** Assume that  $a_k = \infty$  for all  $k$  and there are **no longer fixed supply and demand constants**; every **supply location can produce an infinite amount of goods** and every **demand location can consume an infinite amount of goods.**

# Solution difficulty of MCFP

## Differences to single commodity:

- Solutions to the single-commodity network flow problems are always integral.
- However, solutions to the multicommodity flow problems are not necessarily integral.

# Solution difficulty of MCFP

## Differences to single commodity:

- Solutions to the single-commodity network flow problems are always integral.
- However, solutions to the multicommodity flow problems are not necessarily integral.
- **Continuous flows:** Polynomial time solvable with an LP
- **Integer flows:** NP-Complete, even with only 2 flows!

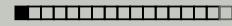
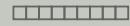
# Solution difficulty of MCFP

## Differences to single commodity:

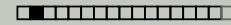
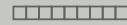
- Solutions to the single-commodity network flow problems are always integral.
- However, solutions to the multicommodity flow problems are not necessarily integral.
- **Continuous flows:** Polynomial time solvable with an LP
- **Integer flows:** NP-Complete, even with only 2 flows!

## Solving the MCFP:

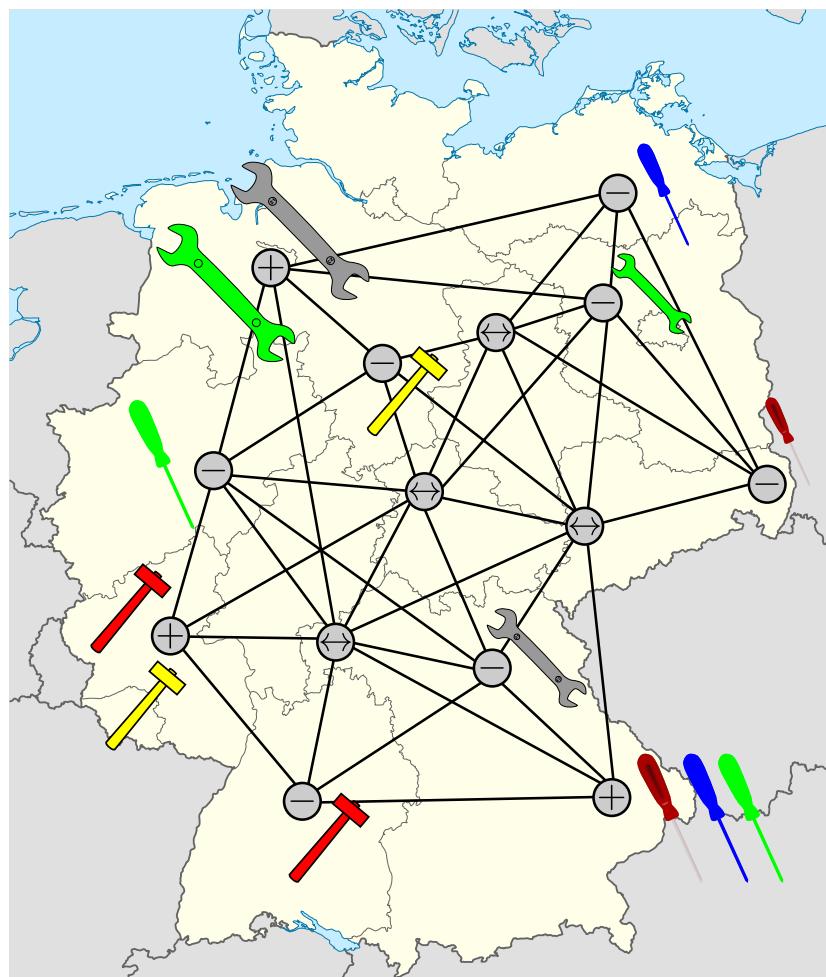
- Mixed-integer programming solver (**Gurobi**, CPLEX, etc.)
- **Specialized algorithms**
- **Approximation techniques** (Lagrangian relaxation, Dantzig-Wolfe decompositions)



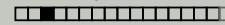
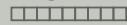
## Time-space models



# Multicommodity flows



Now we know exactly the store, they have certain windows



# Time

Consider now the multicommodity transport problem with deadlines. We will add transport times to the model and deadlines for deliveries.

# Time

Consider now the multicommodity transport problem with deadlines. We will add transport times to the model and deadlines for deliveries.

## Parameters:

- Graph  $G = (V, A)$
- Arc capacity  $\kappa_{ij}$
- Amount  $a_k$  of commodity  $k$
- **Transit time  $t_{ij}$**
- Set of commodities  $K$
- Cost  $c_{ij}^k$  to send flow  $k$  on arc  $(i, j)$
- Commodity source, sink  $s_k, t_k \in V$
- **Earliest availability  $l^k$**
- **Latest delivery time  $u^k$**

Similar to  
vehicle time  
window

How can we model the time component of this problem?

# Modelling options

Variables:  $x_{ij}^k$  – flow of commodity  $k$  on  $(i,j)$

## Arrival time variables:

- For each commodity at each node, we introduce a variable  $w_i^k$  that indicates the time the commodity flows through node  $i$ .

# Modelling options

**Variables:**  $x_{ij}^k$  – flow of commodity  $k$  on  $(i,j)$

**Arrival time variables:**

- For each commodity at each node, we introduce a variable  $w_i^k$  that indicates the time the commodity flows through node  $i$ .

**Objective and constraints:**  $\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k$  *Flow costs for all commodities*

## Modelling the flow of time

First, let's assume the flow is **unsplittable** (1 path per commodity)

These **constraints** need to be connected to the availability and delivery, meaning we need a variable  $y_{ij}^k \in \{0, 1\}$  indicating whether or not flow of  $k$  traverses arc  $(i, j)$  ( $1 = \text{yes}$ ,  $0 = \text{no}$ )

$$x_{ij}^k \leq a^k y_{ij}^k \quad \forall k \in K, (i, j) \in A$$

$\stackrel{=1}{\rightarrow}$  Capacity  
 $\stackrel{=0}{\rightarrow}$  no flow in arc

We also need to make the flows **unsplittable**:

$$y_{ij}^k \leq 1 \quad \forall k \in K, (i, j) \in A$$

# Modelling the flow of time

What we have so far:

$$\sum_{(i,j) \in A} x_{ij}^k - \sum_{(j,i) \in A} x_{ji}^k = \begin{cases} a_i & \text{if } s_k = i \\ -a_i & \text{if } t_k = i \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, i \in V$$

$$x_{ij}^k \leq a^k y_{ij}^k \quad \forall k \in K, (i,j) \in A$$

$$y_{ij}^k \leq 1 \quad \forall k \in K, (i,j) \in A$$

(Note that I am ignoring the capacity right now on purpose)

# Modelling the flow of time

## What we have so far:

$$\begin{aligned} \sum_{(i,j) \in A} x_{ij}^k - \sum_{(j,i) \in A} x_{ji}^k &= \begin{cases} a_i & \text{if } s_k = i \\ -a_i & \text{if } t_k = i \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, i \in V \\ x_{ij}^k &\leq a^k y_{ij}^k \quad \forall k \in K, (i,j) \in A \\ y_{ij}^k &\leq 1 \quad \forall k \in K, (i,j) \in A \end{aligned}$$

(Note that I am ignoring the capacity right now on purpose) Now we need to connect the flow to the time:

$$w_i^k + t_{ij} \leq w_j^k + M(1 - y_{ij}^k) \quad \forall k \in K, (i, j) \in A$$

Starting time next location  
 ↓  
 If = 1  
 then activate  
 time constraint active



# Modelling the flow of time

What we have so far:

$$\sum_{(i,j) \in A} x_{ij}^k - \sum_{(j,i) \in A} x_{ji}^k = \begin{cases} a_i & \text{if } s_k = i \\ -a_i & \text{if } t_k = i \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, i \in V$$

$$x_{ij}^k \leq a^k y_{ij}^k \quad \forall k \in K, (i,j) \in A$$

$$y_{ij}^k \leq 1 \quad \forall k \in K, (i,j) \in A$$

(Note that I am ignoring the capacity right now on purpose) Now we need to connect the flow to the time:

$$w_i^k + t_{ij} \leq w_j^k + M(1 - y_{ij}^k) \quad \forall k \in K, (i,j) \in A$$

When  $y_{ij}^k$  is set to 1, the time constraint is enforced. If it is 0, the  $w$ 's can take any value.

## Modelling the flow of time

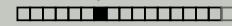
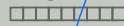
Now we need to connect the availability and deadline of the commodity to the  $w$  variables.

## Any ideas?

$w_i^k$   $l^k$   $u^k \xrightarrow{\text{Latest delivery time}} \text{Arrival}$

$$w_s^K \geq l^K$$





# Modelling the flow of time

Now we need to connect the availability and deadline of the commodity to the  $w$  variables.

**Any ideas?**

For this commodity: The beginning is  $\geq$  Arrival time

$$w_{s_k}^k \geq l^k \quad \forall k \in K$$

$$w_{t_k}^k \leq u^k \quad \forall k \in K$$

Arrival time  $\leq$  Latest arrival time of commodity  
(lost)

↳ that's how we set time window for each commodity

# Modelling the flow of time

Now we need to connect the availability and deadline of the commodity to the  $w$  variables.

**Any ideas?**

$$w_{s_k}^k \geq l^k \quad \forall k \in K$$

$$w_{t_k}^k \leq u^k \quad \forall k \in K$$

Now all that remains are the capacity restrictions...

# Question

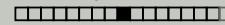
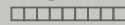
## Question

In the binding constraints for  $y_{ij}^k$  we have (logically):

$$y_{ij}^k = 0 \rightarrow x_{ij}^k = 0$$

Why do we not need a constraint for the following: } For home!

$$x_{ij}^k = 0 \rightarrow y_{ij}^k = 0$$



# Modelling the flow of time

How should we model the capacity restrictions? Options:

- 1 The total amount of flow on an arc at any point in time **may not exceed the capacity**
- 2 The capacity of the arc may not be exceeded over the time span that the arc is activated (transit time)
- 3 The capacity of the arc may not be exceeded over a discretized time span
- 4 The sum of an arc's usage over the entire planning horizon may not exceed the arc's capacity

# Modelling the flow of time

How should we model the capacity restrictions? Options:

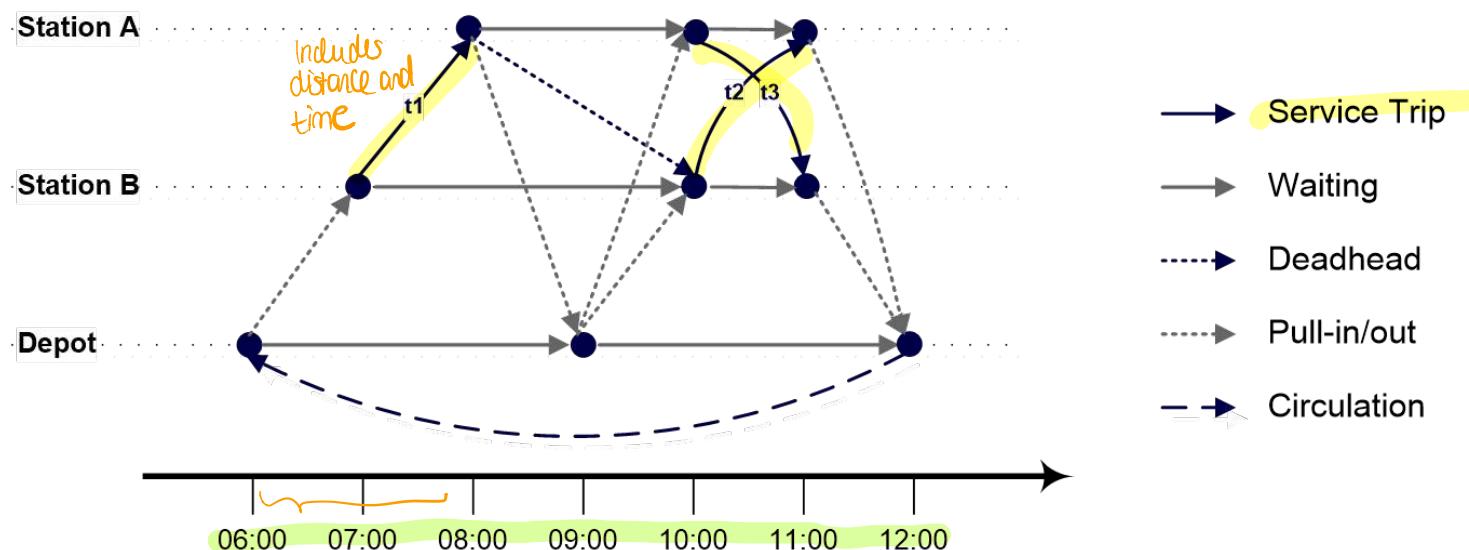
- 1 The total amount of flow on an arc at any point in time may not exceed the capacity
- 2 The capacity of the arc may not be exceeded over the time span that the arc is activated (transit time)
- 3 The capacity of the arc may not be exceeded over a discretized time span
- 4 The sum of an arc's usage over the entire planning horizon may not exceed the arc's capacity

There is no “correct” answer. It depends on the application and the requirements of the problem being modelled.

# Time Space discretization

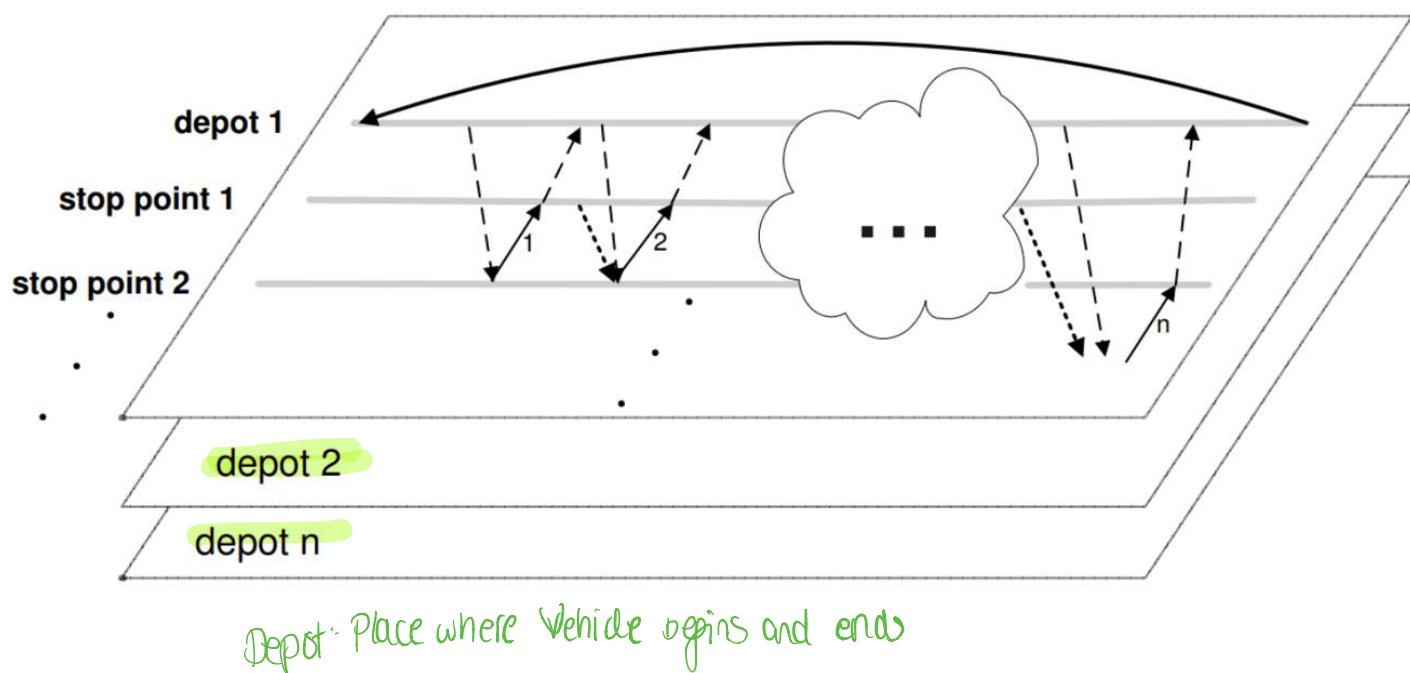
Define  $T$  time periods. Time will no longer be controlled by  $w$  variables, but rather be modelled directly in the graph.

# Time-space network (Vehicle scheduling in public transit)

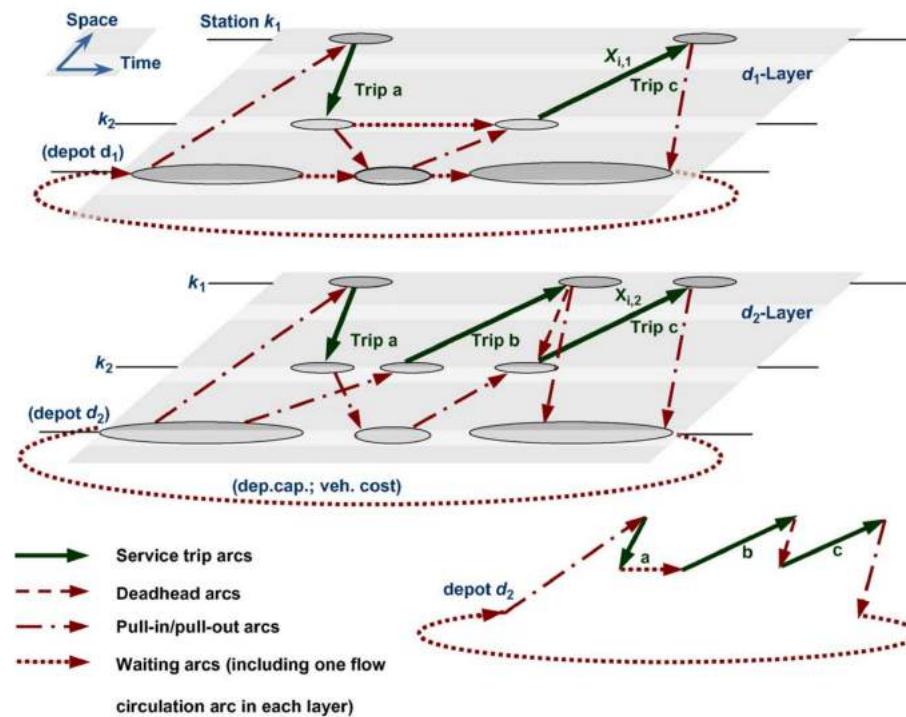


# Time-space network with multicommodity flows

## (Vehicle scheduling in public transit)



# Time-space network with multicommodity flows (Vehicle scheduling in public transit)



# Vehicle scheduling in public transit

Discussion:

- How does the model look like?

# Vehicle scheduling in public transit

## Parameters:

- Set of all networklayers  $S$
- Graph  $G^s = (V^s, A^s)$
- Set of service trips  $F$
- Set of all circulation arcs  $CA^s$
- Set of all arcs  $SA_f$  for service trip  $f$
- Cost  $c_{ij}$  of  $(i, j) \in A^s, s \in S$
- Lower and upperbound  $l_{ij}, u_{ij}$  for  $(i, j) \in A^s, s \in S$
- Layer capacity  $u_s$

# Vehicle scheduling in public transit

**Variables:**  $x_{ij}^s$  – flow on  $(i, j)$

**Objective and constraints:**

$$\min \sum_{s \in S} \sum_{(i,j) \in A^s} c_{ij}^s x_{ij}^s$$

$$\sum_{(i,j) \in A^s} x_{ij}^s - \sum_{(j,i) \in A^s} x_{ji}^s = 0 \quad \forall s \in S, j \in V^s$$

$$\sum_{(i,j) \in SA_f} x_{ij}^s = 1 \quad \forall f \in F$$

*Max and min vehicles*

$$l_{ij}^s \leq x_{ij}^s \leq u_{ij}^s \quad \forall (i,j) \in A^s, s \in S$$

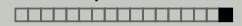
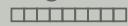
$$\sum_{(i,j) \in CA^s} x_{ij}^s = u_s \quad \forall s \in S$$

$$x_{ij}^s \in \mathbb{N} \quad \forall (i,j) \in A^s, s \in S$$

# Conclusion

- Modelling multiple commodities is hard.
- Modelling time is hard.
- Modelling both is harder.
- But we can do it! We know how.





# Thank you for your attention!

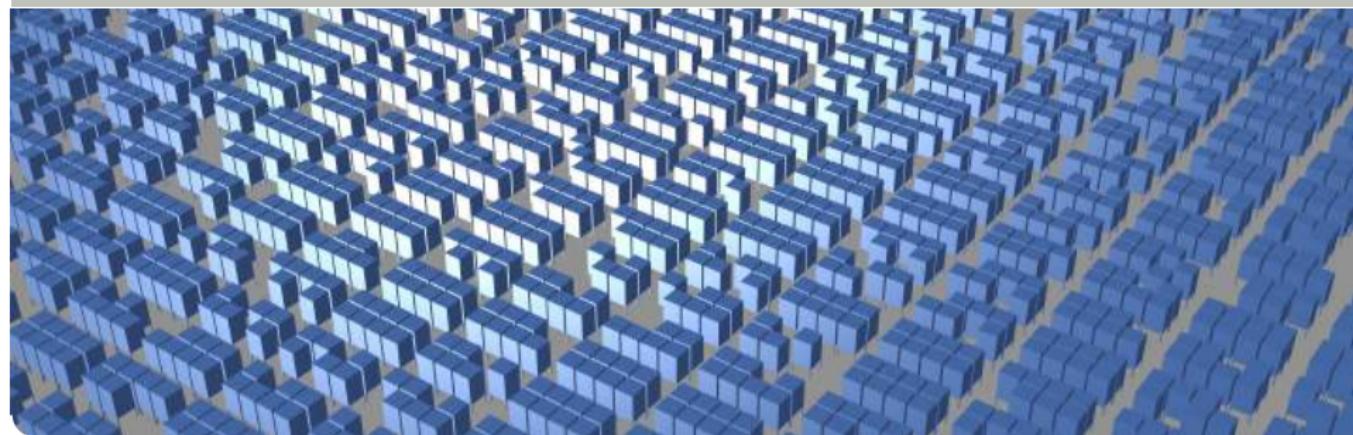
Leuphana University of Lüneburg  
Business information systems, in particular Operations Research  
Prof. Dr. Lin Xie  
Universitätsallee 1  
Building 4, Room 314  
21335 Lüneburg  
Phone +49 4131 677 2305  
Fax +49 4131 677 1749  
[xie@leuphana.de](mailto:xie@leuphana.de)

# Analysing Networks with OR-Methods

Part 7 – Shortest path problem with resource constraints

Lin Xie | 01.06.2021

PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH

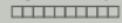




## 1 Dijkstra's Algorithm

## 2 Shortest path problems with resource constraints (SPPRC)

## 3 Dynamic programming



## Dijkstra's Algorithm

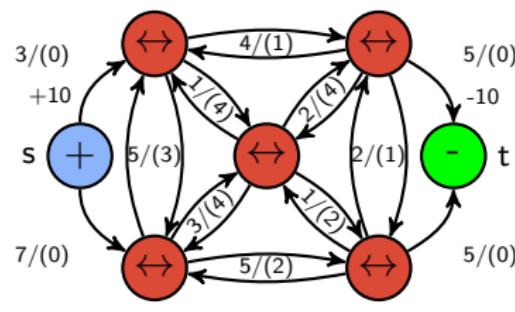
## Special case of transshipment: Min-cost-flow problem

## Recap

Graph  $G = (N, A)$

$$N = N^+ \cup N^\leftrightarrow \cup N^-$$

One supplier,  
1 demander



capacity/(cost)

$x_{ij}$  Number of units flowing on arc  $(i, j) \in A$

$s_i$  Supply of units at node  $i \in N^+$

$d_i$ : Demand for units at node  $i \in N^+$

$c_{ij}$  Transportation cost per unit on arc  $(i, j) \in A$

$\kappa_{ij}$  Maximum capacity on arc  $(i, j) \in A$

demander: only outgoing  
supplier: only incoming

$$\min \sum_{(i,j) \in A} c_{ij} x_j$$

st

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = s_i \quad \forall i \in N^+ \quad \text{↑ demand up}$$

$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = -d_i \quad \forall i \in N^- \quad \text{↑ demand down}$$

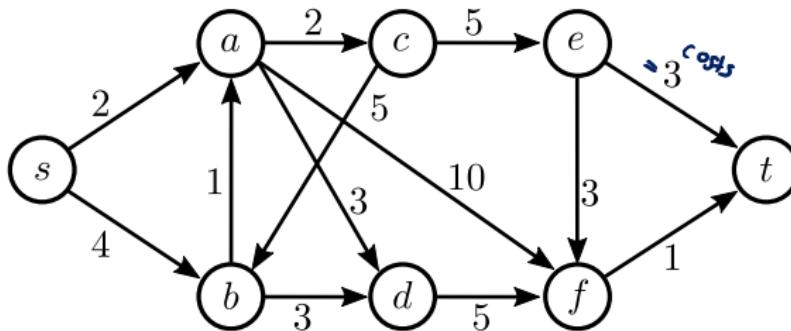
$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in N^{\leftrightarrow}$$

$$0 \leq x_{ij} \leq \kappa_{ij} \quad \forall (i,j) \in A$$

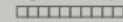
2

## Dijkstra's Algorithm

**Goal:** Given a graph  $G = (V, A)$ , find the minimum cost shortest path from node  $s$  to node  $t$  given positive arc costs  $c_{ij}$ .



We don't need to visit all nodes exactly once  
↓ Just with cost important



# Dijkstra's Algorithm

1: **function** SPP-DIJKSTRA( $G = (V, A)$ ,  $s$ ,  $t$ ,  $c$ )

2:    $open \leftarrow V$

3:    $dist \leftarrow$  array of length  $|V|$  initialized to  $\infty$

4:    $prev \leftarrow$  empty array of length  $|V|$ , initialized to  $\perp$

5:    $dist[s] \leftarrow 0$  (beginning node)

6:   **while**  $open$  is not empty **do**

7:      $u \leftarrow \operatorname{argmin}_{v \in open} \{dist[v]\}$

Check all distance of nodes  
↳ find node with smallest dist.

8:     **if**  $u = t$  **then return**  $dist[u]$

↳ If  $u$  is already end node

9:      $open \leftarrow open \setminus u$  Remove  $u$  from list after checking all outgoing arcs of  $u$

10:    **for**  $(u, v) \in A$  **do**

11:       $d' \leftarrow dist[u] + c_{uv}$

12:      **if**  $d' < dist[v]$  **then**

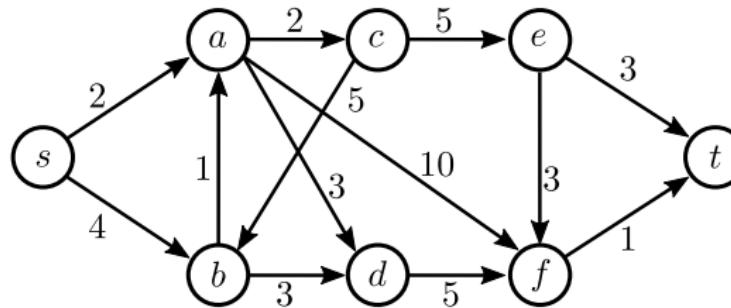
13:         $dist[v] \leftarrow d'$

14:         $prev[v] \leftarrow u$

15:   **return** No path between  $s$  and  $t$  exists.

*begin*   *end*   *matrix of cost from each arc*

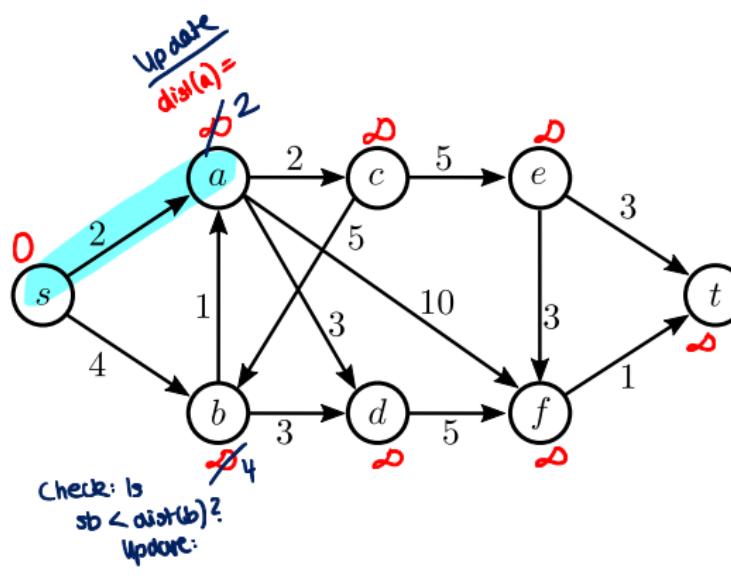
# Graphical view of Dijkstra's algorithm



Node	<i>dist</i>	<i>prev</i>
<i>s</i>	$\infty$	$\perp$
<i>a</i>	$\infty$	$\perp$
<i>b</i>	$\infty$	$\perp$
<i>c</i>	$\infty$	$\perp$
<i>d</i>	$\infty$	$\perp$
<i>e</i>	$\infty$	$\perp$
<i>f</i>	$\infty$	$\perp$
<i>t</i>	$\infty$	$\perp$

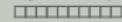
*open* =  
 $\{s, a, b, c, d, e, f, t\}$   
*All nodes*

# Graphical view of Dijkstra's algorithm

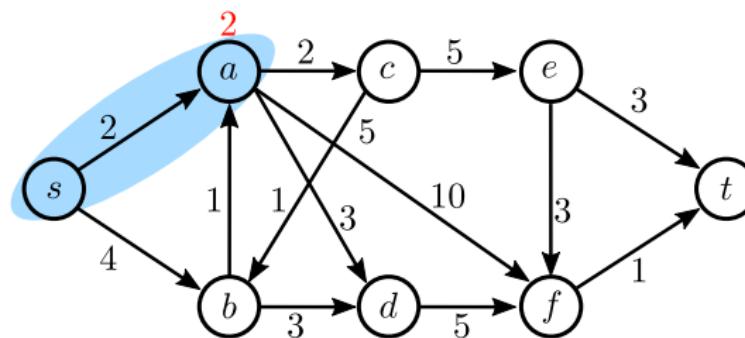


Node	dist	prev
$s = u$	0	$\perp$
$a$	$\infty$	$\perp$
$b$	$\infty$	$\perp$
$c$	$\infty$	$\perp$
$d$	$\infty$	$\perp$
$e$	$\infty$	$\perp$
$f$	$\infty$	$\perp$
$t$	$\infty$	$\perp$

Remove  $u$  from open list  
 $\text{open} = \{s, a, b, c, d, e, f, t\}$

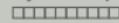


# Graphical view of Dijkstra's algorithm

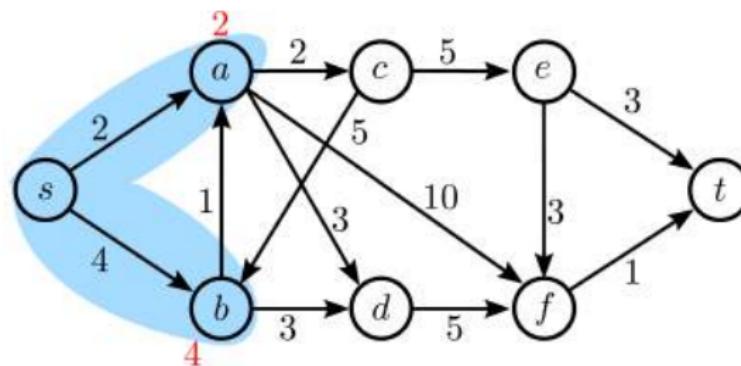


Node	<i>dist</i>	<i>prev</i>
<i>s</i>	0	$\perp$
<i>a</i>	2	<i>s</i>
<i>b</i>	$\infty$	$\perp$
<i>c</i>	$\infty$	$\perp$
<i>d</i>	$\infty$	$\perp$
<i>e</i>	$\infty$	$\perp$
<i>f</i>	$\infty$	$\perp$
<i>t</i>	$\infty$	$\perp$

*open* =  
 $\{a, b, c, d, e, f, t\}$

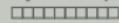


# Graphical view of Dijkstra's algorithm

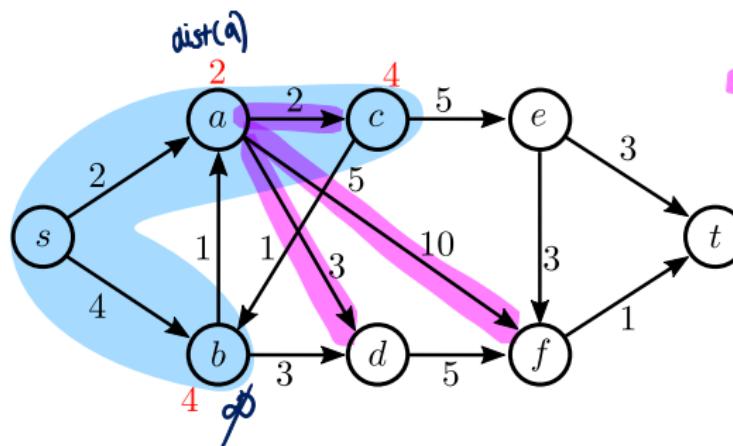


Node	<i>dist</i>	<i>prev</i>
<i>s</i>	0	$\perp$
<i>a</i>	2	<i>s</i>
<i>b</i>	4	<i>s</i>
<i>c</i>	$\infty$	$\perp$
<i>d</i>	$\infty$	$\perp$
<i>e</i>	$\infty$	$\perp$
<i>f</i>	$\infty$	$\perp$
<i>t</i>	$\infty$	$\perp$

*open* =  
 $\{a, b, c, d, e, f, t\}$

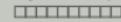


# Graphical view of Dijkstra's algorithm

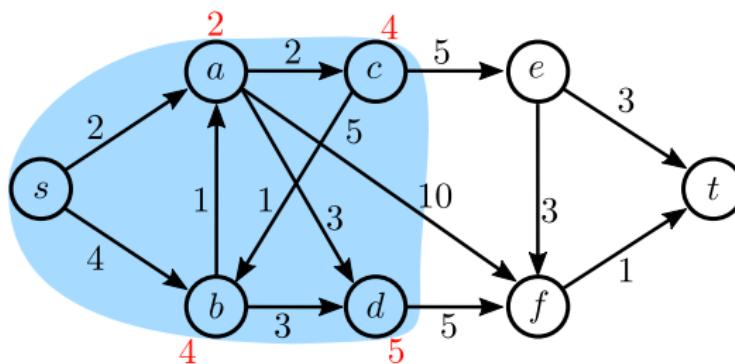


Node	<i>dist</i>	<i>prev</i>
<i>s</i>	0	$\perp$
<i>a</i>	2	<i>s</i>
<i>b</i>	4	<i>s</i>
<i>c</i>	4	<i>a</i>
<i>d</i>	$\infty$	$\perp$
<i>e</i>	$\infty$	$\perp$
<i>f</i>	$\infty$	$\perp$
<i>t</i>	$\infty$	$\perp$

*open* =  
 $\{b, c, d, e, f, t\}$

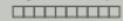


# Graphical view of Dijkstra's algorithm choose b or c

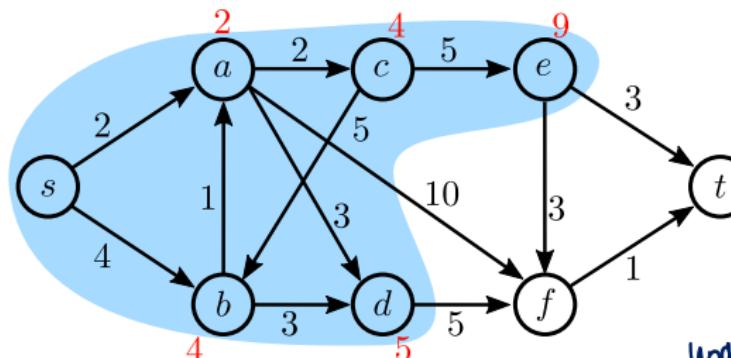


Node	<i>dist</i>	<i>prev</i>
<i>s</i>	0	$\perp$
<i>a</i>	2	<i>s</i>
<i>b</i>	4	<i>s</i>
<i>c</i>	4	<i>a</i>
<i>d</i>	5	<i>a</i>
<i>e</i>	$\infty$	$\perp$
<i>f</i>	12	<i>a</i>
<i>t</i>	$\infty$	$\perp$

*open* =  
 $\{b, c, d, e, f, t\}$



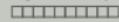
# Graphical view of Dijkstra's algorithm



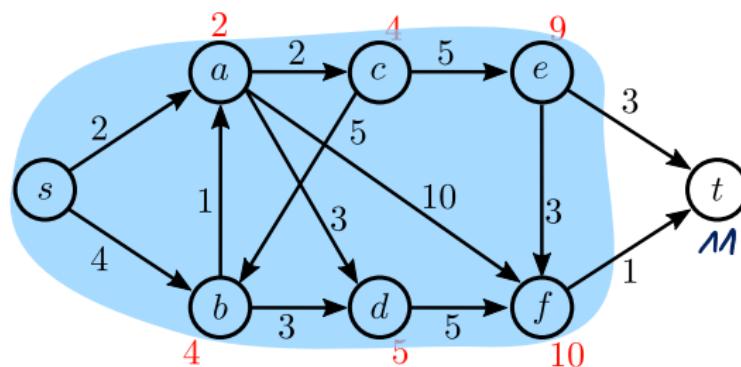
update  
dist(f)

$$open = \{d, e, f, t\}$$

Node	dist	prev
s	0	⊥
a	2	s
b	4	s
c	4	a
d	5	a
e	9	c
f	12	a
t	∞	⊥



# Graphical view of Dijkstra's algorithm

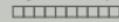


Node	<i>dist</i>	<i>prev</i>
<i>s</i>	0	$\perp$
<i>a</i>	2	<i>s</i>
<i>b</i>	4	<i>s</i>
<i>c</i>	4	<i>a</i>
<i>d</i>	5	<i>a</i>
<i>e</i>	9	<i>c</i>
<i>f</i>	10	<i>d</i>
<i>t</i>	$\infty$	$\perp$

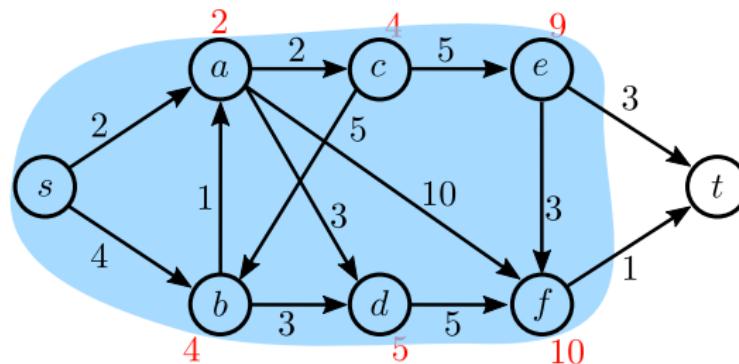
$$10 + 11 < 12 \\ \text{go update}$$

$open = \{e, f, t\}$

smallest  $dist \cdot e$



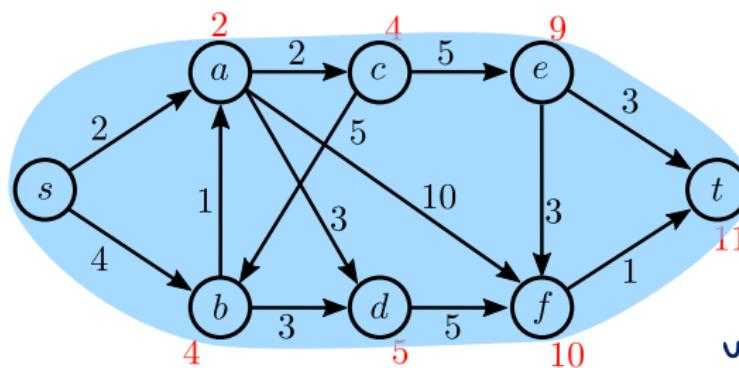
# Graphical view of Dijkstra's algorithm



Node	<i>dist</i>	<i>prev</i>
<i>s</i>	0	$\perp$
<i>a</i>	2	<i>s</i>
<i>b</i>	4	<i>s</i>
<i>c</i>	4	<i>a</i>
<i>d</i>	5	<i>a</i>
<i>e</i>	9	<i>c</i>
<i>f</i>	10	<i>d</i>
<i>t</i>	12	<i>e</i>

$$\text{open} = \{f, t\}$$

# Graphical view of Dijkstra's algorithm



Node	dist	prev
s	0	$\perp$
a	2	s
b	4	s
c	4	a
d	5	a
e	9	c
f	10	d
t	11	f

$dist[t] = 11$

$open = \{t\}$

$u = t$  then return

$dist(u)$

$s \rightarrow a \rightarrow d \rightarrow f \rightarrow t$

Shortest path with  $dist = 11$   
found from s to t

# Shortest path problems with resource constraints (SPPRC)

# Types of shortest path problems

- **Elementary:** No cycles allowed in the path
- **Negative arc costs:** Arc costs do not have to be in  $\mathbb{R}^+$
- **Resource constrained:** Edges have attributes other than cost subject to capacity constraints  
*TODAY*

cannot apply  
Algorithm

# Types of shortest path problems

- **Elementary**: No cycles allowed in the path
- **Negative arc costs**: Arc costs do not have to be in  $\mathbb{R}^+$
- **Resource constrained**: Edges have attributes other than cost subject to capacity constraints

The first two problems are still polynomial time solvable (assuming no negative cost cycles), but the last one is NP-hard.

# MIP model of the SPPRC

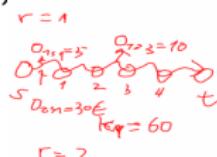
## Parameters:

- Graph  $G = (V, A)$
- Arc costs  $c_{ij}$
- Set of resources  $R$

- Max. amt. of resource  $r$ ,  $k_r$
- Resource  $r$  extension  $o_{rij}$  on arc

## Variables:

- $x_{ij} \in \{0, 1\}$ : 1 iff arc  $(i, j)$  on the shortest path



$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \quad \text{total cost}$$

$$\left\{ \begin{array}{ll} \text{s.t.} & \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 & \forall i \in V \setminus \{s, t\} \\ & \sum_{(i,j) \in A} x_{ij} \leq 1 & \forall i \in V \setminus \{t\} \\ & \sum_{(i,t) \in A} x_{it} = 1 & \text{exactly 1 incoming arc to } t \\ & \sum_{(i,j) \in A} o_{rij} x_{ij} \leq k_r & \forall r \in R \\ & x_{ij} \in \{0, 1\} & \forall (i,j) \in A \end{array} \right.$$

For all transhipment node:  
 flow conservation constraint  
 At least 1 outgoing arc

All selected arcs and resource extension < max resource

edges that  
not necessary to  
visit all nodes  
except s, t

capacity  
constraint

## Key insight from previous model

Note how in the previous model we do not have to keep track of how much of each resource is being carried on each arc!

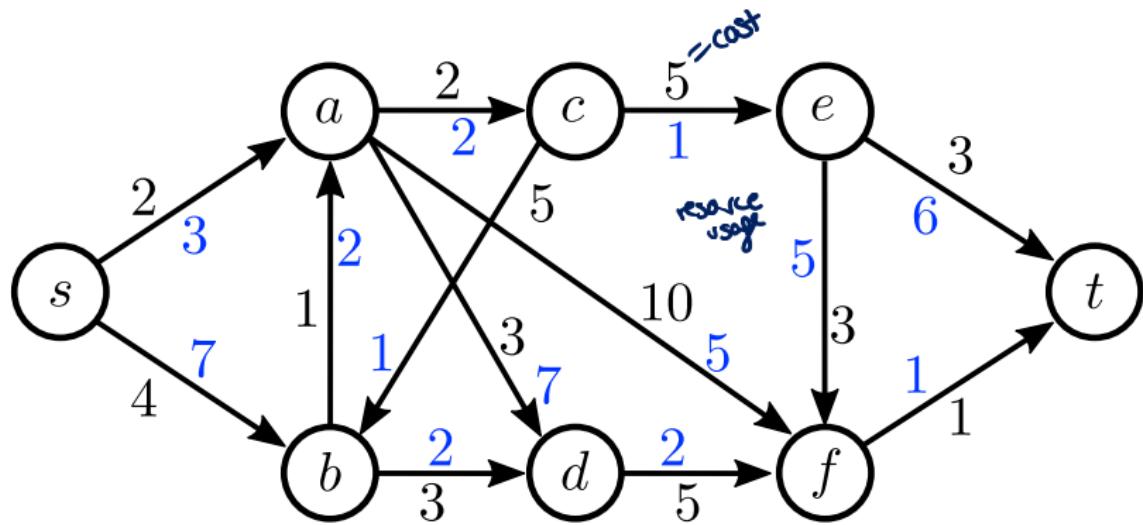
**Why?**

(Hint: consider the assumptions of the previous model)

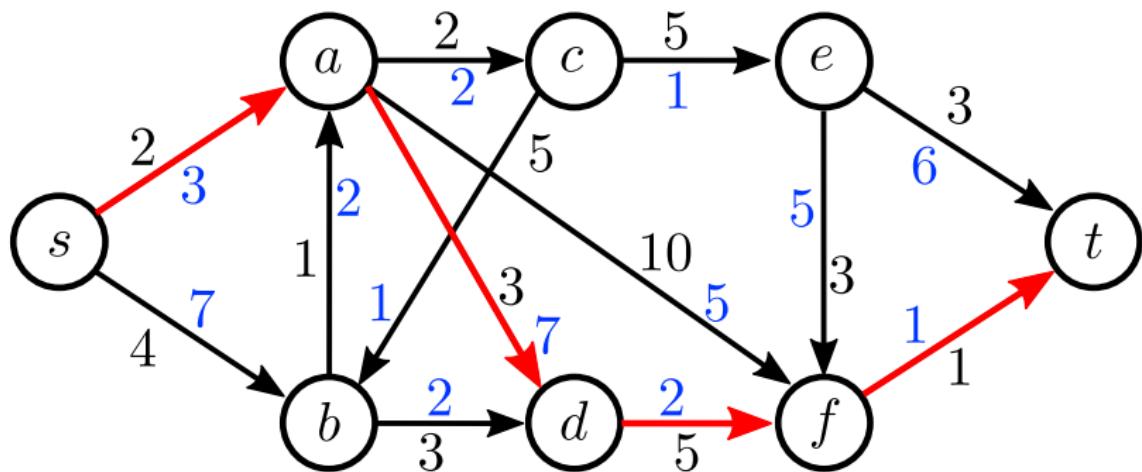
Because on each arc there is resource already

If one arc is active, the usage/resource is also active ?

# Example SPPRC with resource constraints



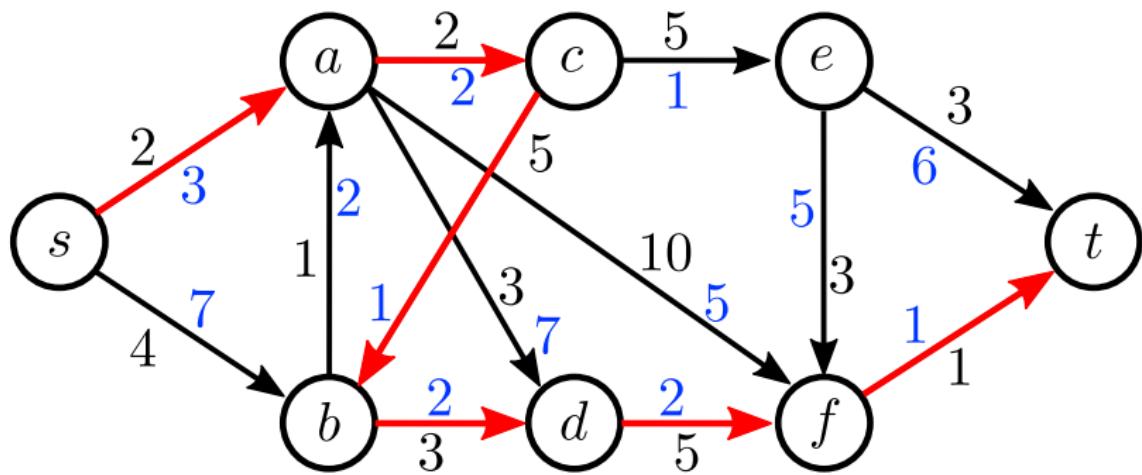
# Example SPPRC with resource constraints



**Path cost:** 11

**Resource usage:** 13  
(also cost)

# Example SPPRC with resource constraints



**Path cost:** 18

**Resource usage:** 11

# SPPRC with intermediate resource constraints

**Extra parameters:** Lower/upper bound  $l_{ri}$ ,  $u_{ri}$  for each resource  
**Extra variables:**  $y_{ri}$  – the amount of resource  $r$  at node  $i$

Similar to vehicle rolling time window

time window  
[10, 12]  
①

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij}$$

$$\text{s.t. } \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in V \setminus \{s, t\}$$

$$\sum_{(i,j) \in A} x_{ij} \leq 1 \quad \forall i \in V \setminus \{t\}$$

$$y_{ri} + o_{rij} \leq M(1 - x_{ij}) + y_{rj} \quad \begin{matrix} \text{Arrive time} \\ \text{at node } j \\ \text{if node active (customer visit)} \end{matrix} \quad \forall (i,j) \in A, r \in R$$

$$x_{ij} \in \{0, 1\} \quad \forall (i,j) \in A$$

$$\forall r \in R, i \in V$$

$$l_{ri} \left( \sum_{(j,i) \in A} x_{ji} \right) \leq y_{ri} \leq u_{ri} \quad \left( \sum_{(j,i) \in A} x_{ji} \right) = 1$$

Lower bound (like no delay)  
If = 1: keep time window  
Incoming Arc to customer

# SPPRC with intermediate resource constraints

**Extra parameters:** Lower/upper bound  $l_{ri}$ ,  $u_{ri}$  for each resource

**Extra variables:**  $y_{ri}$  – the amount of resource  $r$  at node  $i$

$$\begin{aligned}
 & \min \sum_{(i,j) \in A} c_{ij} x_{ij} \\
 \text{s.t. } & \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in V \setminus \{s, t\} \\
 & \sum_{(i,j) \in A} x_{ij} \leq 1 \quad \forall i \in V \setminus \{t\} \\
 & y_{ri} + o_{rij} \leq M(1 - x_{ij}) + y_{rj} \quad \forall (i,j) \in A, r \in R \\
 & x_{ij} \in \{0, 1\} \quad \forall (i,j) \in A \\
 & l_{ri} \sum_{(i,i) \in A} x_{ji} \leq y_{ri} \leq u_{ri} \sum_{(i,i) \in A} x_{ji} \quad \forall r \in R, i \in V
 \end{aligned}$$

**Question 1:** What happened to the constraint on  $k_r$ ?

↳  $k_r = u_{rk}$  Upper bound for this resource in last node

# SPPRC with intermediate resource constraints

**Extra parameters:** Lower/upper bound  $l_{ri}$ ,  $u_{ri}$  for each resource

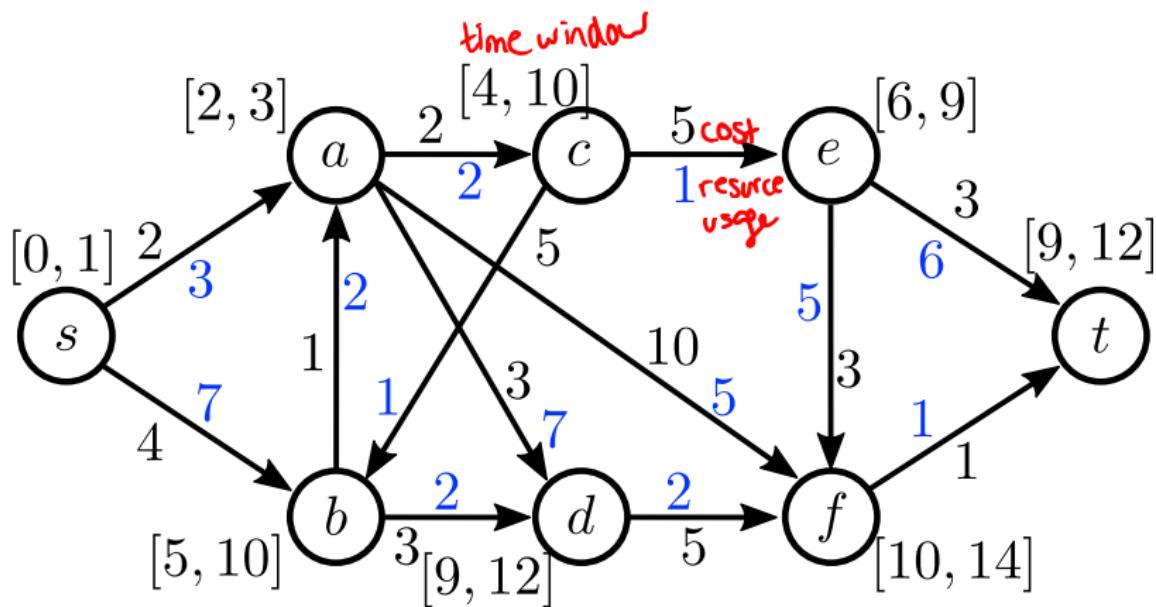
**Extra variables:**  $y_{ri}$  – the amount of resource  $r$  at node  $i$

$$\begin{aligned}
 & \min \sum_{(i,j) \in A} c_{ij} x_{ij} \\
 \text{s.t. } & \sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall i \in V \setminus \{s, t\} \\
 & \sum_{(i,j) \in A} x_{ij} \leq 1 \quad \forall i \in V \setminus \{t\} \\
 & y_{ri} + o_{rij} \leq M(1 - x_{ij}) + y_{rj} \quad \forall (i,j) \in A, r \in R \\
 & x_{ij} \in \{0, 1\} \quad \forall (i,j) \in A \\
 & l_{ri} \sum_{(i,i) \in A} x_{ji} \leq y_{ri} \leq u_{ri} \sum_{(i,i) \in A} x_{ji} \quad \forall r \in R, i \in V
 \end{aligned}$$

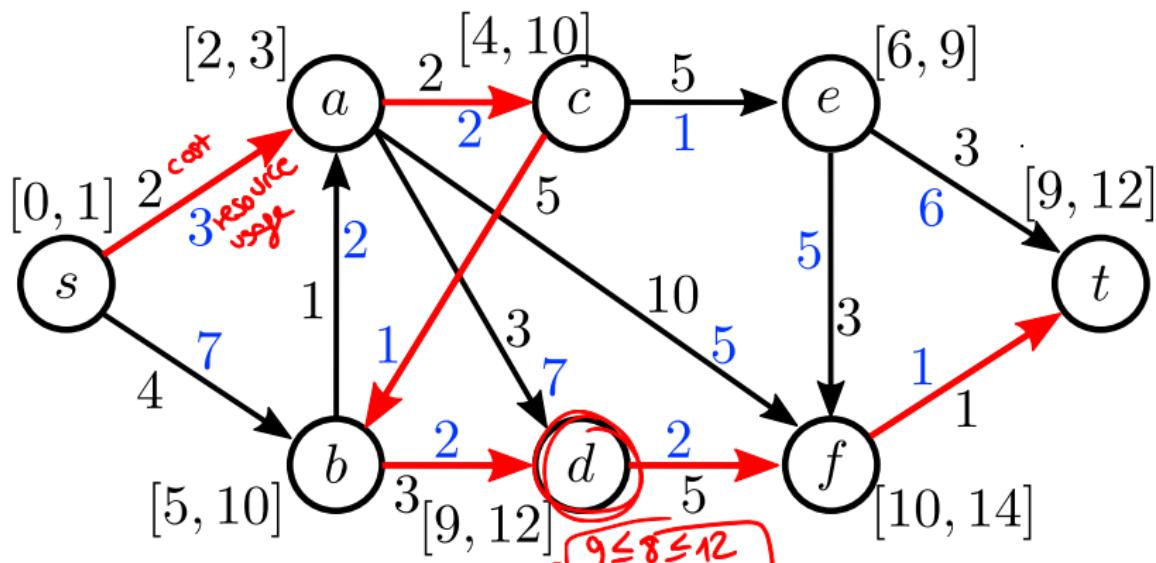
**Question 2:** What is an admissible bound for  $M$ ?

↙  $u_{ri}$  upper bound for each resource on node  $i$

# Example SPPRC with intermediate resource constraints



# Example SPPRC with intermediate resource constraints

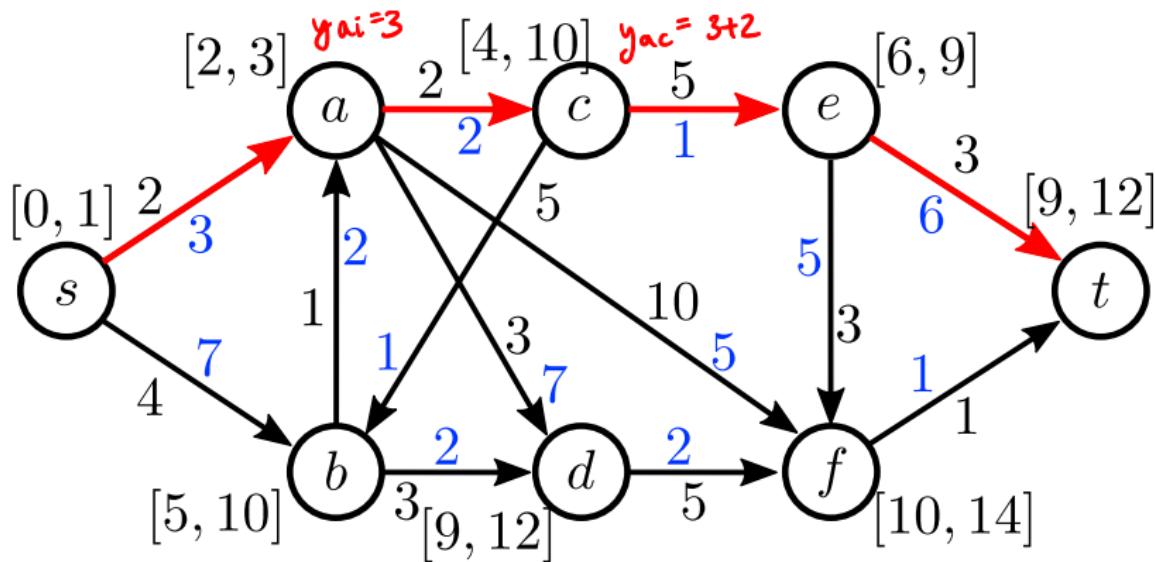


Now the path is infeasible at node  $d$ !

$$\begin{aligned} (3+2+1+2+2+1=11) \\ 2+2+5+3+5+1=18 \end{aligned}$$

# Example SPPRC with intermediate resource constraints

*yes feasible*



→ calculate  $y$  based on last visit node → get  $y$  value and  
→ check between lower and upper bound

# Dynamic programming

# Dynamic programming

## Definition

“... dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions - ideally, using a memory-based data structure” – Wikipedia

# Dynamic programming

## Definition

“... dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions - ideally, using a memory-based data structure” – Wikipedia

**Idea:** DP performs systematic enumeration while avoiding unnecessary work.

# Dynamic programming

## Definition

“... dynamic programming is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions - ideally, using a memory-based data structure” – Wikipedia

**Idea:** DP performs systematic enumeration while avoiding unnecessary work.

### Unnecessary work:

- Repeatedly solving subproblems
- Solving dominated subproblems

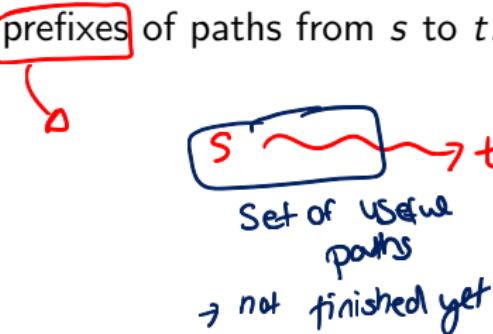
# Dynamic programming for the SPPRC

First, we need definitions.

Let:

- $U$  be the set of unprocessed paths
- $P$  be the set of useful paths

Note that paths in  $P$  can be prefixes of paths from  $s$  to  $t$ .



# Dynamic programming for the SPPRC

First, we need definitions.

Let:

- $U$  be the set of unprocessed paths
- $P$  be the set of useful paths

Note that paths in  $P$  can be prefixes of paths from  $s$  to  $t$ .

All paths in  $P$  are pareto optimal or are prefixes of pareto optimal paths.

# Dynamic programming for the SPPRC

First, we need definitions.

**Let:**

- $U$  be the set of unprocessed paths
- $P$  be the set of useful paths

Note that paths in  $P$  can be prefixes of paths from  $s$  to  $t$ .

All paths in  $P$  are pareto optimal or are prefixes of pareto optimal paths.

Given a path  $p = (v_1, v_2, \dots, v_{|I(p)|})$ , let  $\text{end}(p) = v_{|I(p)|}$  be the last node of the path.

if  $\text{end}(p) = p$   
     : complete

# Pareto optimality

## Assume:

- A path  $p$  has resource values  $r_1(p) \dots r_n(p)$ , for  $n$  resources, where the cost is resource  $r_1$ .
- Resource extensions are all non-negative

## Pareto optimality

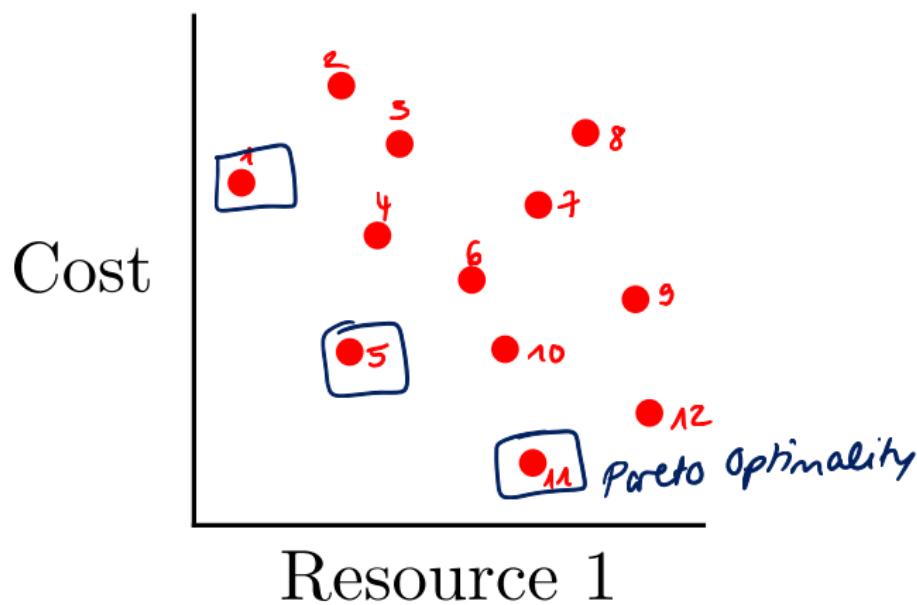
A path  $p \in P$  is pareto optimal in  $P$  if there exists no  $p' \in P$  with  $\text{end}(p) = \text{end}(p')$  such that  $\forall i \leq n : r_i(p') \leq r_i(p)$ .

same ending node

In words: There is no path that is “better” than  $p$  in all resources simultaneously.

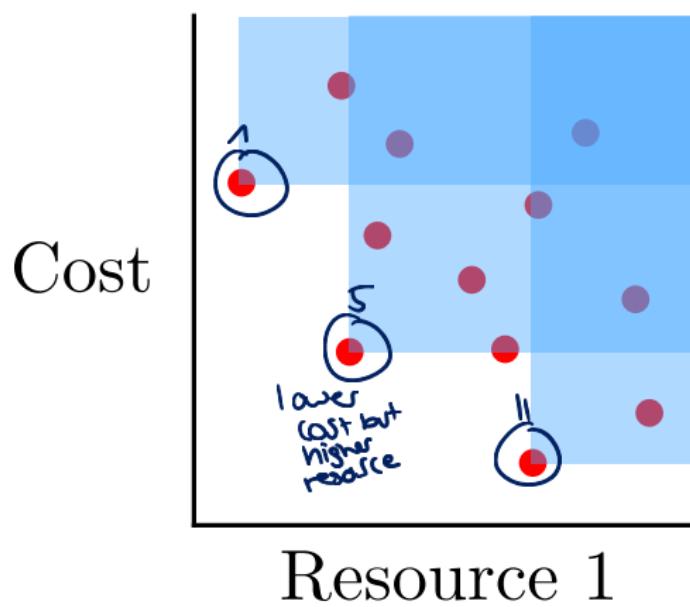
If we find another path where all resources are smaller then the path is not pareto optimal

# Pareto optimality in pictures



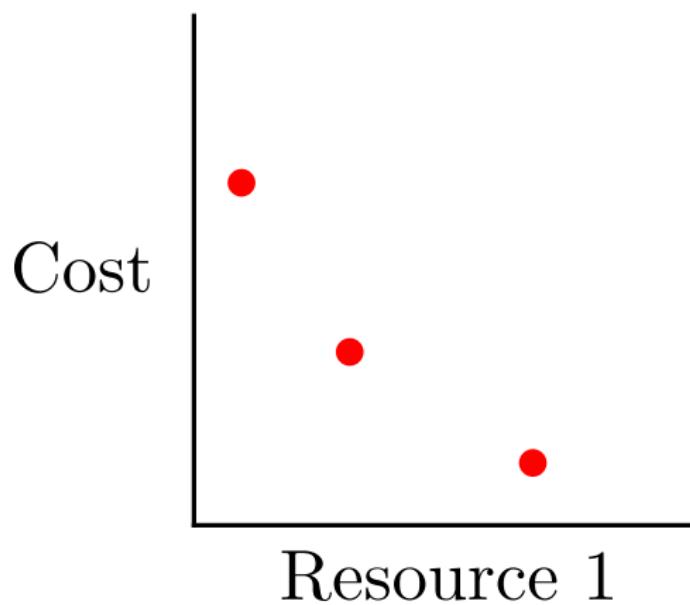
Solutions are shown in two dimensions

# Pareto optimality in pictures



Areas dominated by pareto optimal points are shown

# Pareto optimality in pictures



Dominated solutions are removed, leaving only the pareto front

# Dynamic programming for the SPPRC

```

1: function SPPRC( $G = (V, A)$ ,  $s, t, f, r_1 \dots r_n$ )
2:    $U \leftarrow \{s\}$ ,  $P \leftarrow \emptyset$  Set of unprocessed paths → used paths
3:   while  $U \neq \emptyset$  do
4:     Select a path  $q \in U$  and remove it from  $U$ 
5:     for  $(end(q), w) \in A$  do
6:       if  $w$  is a feasible extension to  $q$  then
7:          $U \leftarrow U \cup (q, w)$  Add part of path to U
8:          $P \leftarrow P \cup q$  Part removed goes to P
9:       Apply dominance criteria to  $U \cup P$  ending at node  $v$ 
10:      return Identify the solution  $s \in P$  with minimal cost  $f(s)$ 

```

*extension only for  
non-dominated  
path*

**Domination step:** ⑨  
**CORE PART**

(See: Irnich and Desaulniers (2005) in *Column Generation*)

If we skip ⑨: Then we need to check all possible extensions of all paths to find out min cost

# Dynamic programming for the SPPRC

```

1: function SPPRC( $G = (V, A)$ ,  $s, t, f, r_1 \dots r_n$ )
2:    $U \leftarrow \{s\}$ ,  $P \leftarrow \emptyset$ 
3:   while  $U \neq \emptyset$  do
4:     Select a path  $q \in U$  and remove it from  $U$ 
5:     for  $(end(q), w) \in A$  do
6:       if  $w$  is a feasible extension to  $q$  then
7:          $U \leftarrow U \cup (q, w)$ 
8:        $P \leftarrow P \cup q$ 
9:     Apply dominance criteria to  $U \cup P$  ending at node  $v$ 
10:    return Identify the solution  $s \in P$  with minimal cost  $f(s)$ 

```

(See: Irnich and Desaulniers (2005) in *Column Generation*)

**Question:** What happens if we skip the domination step?

# SPPRC domination

**Given:** An SPPRC where all resource extensions are positive and there are no intermediate resource constraints.

What is the domination criteria for this type of SPPRC?

Check all resources

If all are better: Non-dominated paths

If Intermediate resource constraint

↳ also check feasibility

(HW)

For Ex 2 Part 2:

If outside time window, can wait

Important: What is pareto optimum and what dominance criteria

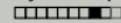
## SPPRC domination

**Given:** An SPPRC where all resource extensions are positive and there are no intermediate resource constraints.

What is the domination criteria for this type of SPPRC?

Assume now that you have an SPPRC with intermediate resource constraints. What would the domination criteria look like?

Need to check Intermediate Resource constraint,  
e.g. Time window



# Where is the SPPRC used?

Primarily as a subproblem:

- In column generation
- In metaheuristics

## Further reading

- 1 S. Irnich and G. Desaulniers (2005) Column Generation, Chapter 2,  
pp. 33–65

for HW: ① the trucks can  
carry different  
types of masks

Task 2:

Important: Dominance Criteria  
little bit different to problem in lecture

Transport between different customers all  
allowed as long as transport capacity net

# Conclusion

- SPPRC: NP-complete version of the shortest path problem
- Can be solved with dynamic programming (label setting algorithm)
- The key is having good domination criteria

# Thank you for your attention!

Leuphana University of Lüneburg  
Business information systems, in particular Operations Research  
Prof. Dr. Lin Xie  
Universitätsallee 1  
Building 4, Room 314  
21335 Lüneburg  
Phone +49 4131 677 2305  
Fax +49 4131 677 1749  
[xie@leuphana.de](mailto:xie@leuphana.de)



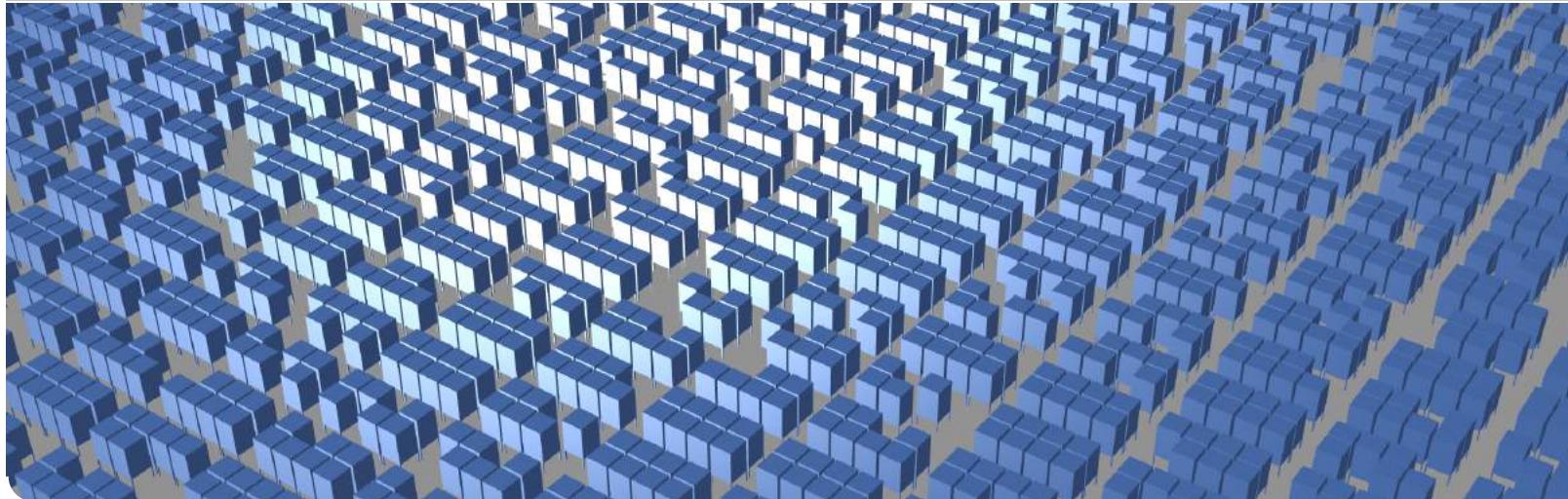


# Analysing Networks with OR-Methods

Part 8 – Heuristics and Local Search

Lin Xie | 08.06.2021

PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH



## Question for Homework 2:

$r$  = Iteration of dynamic programming

$$f'^{+1}(j, k) = \min[f'(j, k), \min_{i, p_i} \{f'(i, j) + p_i + t_{ij}\}] \quad (9)$$

## Dynamic programming for the SPPRC

```

1: function SPPRC( $G = (V, A)$ ,  $s, t, f, r_1 \dots r_n$ )
2:    $U \leftarrow \{s\}$ ,  $P \leftarrow \emptyset$ 
3:   while  $U \neq \emptyset$  do
4:     Select a path  $q \in U$  and remove it from  $U$ 
5:     for  $(end(q), w) \in A$  do
6:       if  $w$  is a feasible extension to  $q$  then
7:          $U \leftarrow U \cup (q, w)$ 
8:        $P \leftarrow P \cup q$ 
9:     Apply dominance criteria to  $U \cup P$  ending at node
10:    return Identify the solution  $s \in P$  with minimal cost  $f(s)$ 

```

(See: Irnich and Desaulniers (2005) in *Column Generation*)

Satisfy  
Constraints

Which one should be  
eliminated due to  
dominance criteria?

① Beginning:  $U = \{1\}$   $P \leftarrow \emptyset$   
 $U = \{(1,2), (1,3), (1,5)\}$   $P \leftarrow \{(1)\}$

$f(1,2) = 0$      $f(1,3) = 0$      $f(1,5) = 0$

$p_1 = 10$      $p_1 = 10$      $p_1 = 15$

ode v  
(s)

②  $U = \{(1,3), (1,5), (1,2,4)\}$  ~~(1,2,16)~~

$f(2,4) = f(1,2) + p_1 + t_{12}^{10}$   
   =  $0 + 10 + 10$   
   =  $20$

check from table:  $18 \leq f(2,4) \leq 30$

check if feasible  $28 \leq f(2,4) + p_2 \leq 35$   
 $\underbrace{+ p_2}_{20+8=28}$

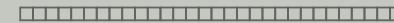
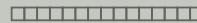
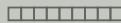
$p_2 = 8$

check  $p_2 \leq \min \{30 - 20, 35 - 20\}$

[Check old solution  
Compare with current  
Take smaller one]

- ① Check feasibility
- ② Check dominance criteria

If found path' check all outgoing paths  
Check path with same ending = Dominance criteria from lecture

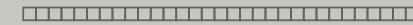
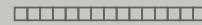


# Agenda

- 1 Heuristic methods
- 2 Heuristics vs. metaheuristics
- 3 Example heuristics
- 4 Fitness functions
- 5 Local search example
  - 1 Local improvement
  - 2 High-level strategies
- 6 Local search methods
  - 1 Iterative improvement
  - 2 Simulated annealing (SA)
  - 3 Tabu search (TS)
  - 4 Iterated local search (ILS)

check path with same ending = Dominance criterion  
node  
we want to eliminate some paths that not lead to optimum  
which one is worse?  
remove larger one

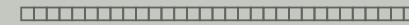
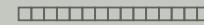
# Heuristic methods



# Heuristic methods

## Definition

Heuristic methods attempt to solve problems without providing guarantees of optimality or solution quality using tricks or a “rule of thumb”.



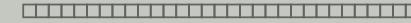
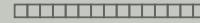
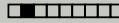
# Heuristic methods

## Definition

Heuristic methods attempt to solve problems without providing guarantees of optimality or solution quality using tricks or a “rule of thumb”.

## What heuristics provide:

- Fast solutions
- Multi-objective capabilities



# Heuristic methods

## Definition

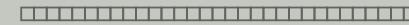
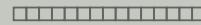
Heuristic methods attempt to solve problems without providing guarantees of optimality or solution quality using tricks or a “rule of thumb”.

## What heuristics provide:

- Fast solutions
- Multi-objective capabilities

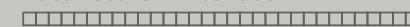
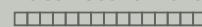
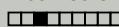
## What heuristics don't provide:

- Solution proofs (positive or negative)
- Guarantees



# Exact vs. heuristic methods

Until today, we used *exact* methods to find solutions, meaning if we found a solution it was guaranteed to be optimal, and if we did not find a solution, then it was guaranteed that none exists.

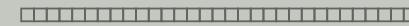
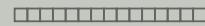


# Exact vs. heuristic methods

Until today, we used *exact* methods to find solutions, meaning if we found a solution it was guaranteed to be optimal, and if we did not find a solution, then it was guaranteed that none exists. Now,

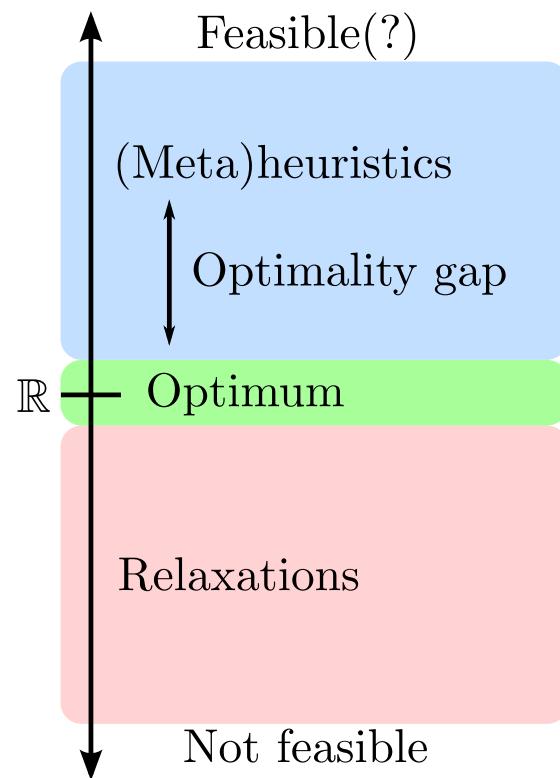
we will investigate *heuristic* methods that

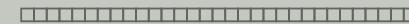
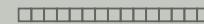
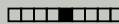
- provide no proof of optimality or infeasibility
- do not have standardized solution software\*
- have no guarantees of solution quality



# The optimality gap

Goal: Minimization



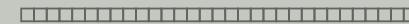
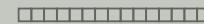
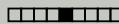


# Metaheuristics

## Definition 2

“[M]etaheuristics are 'solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space.' ”

– Michel Gendreau and Jean-Yves Potvin, *Handbook of Metaheuristics, Second Edition*

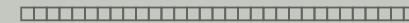
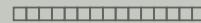
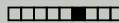


# Metaheuristics

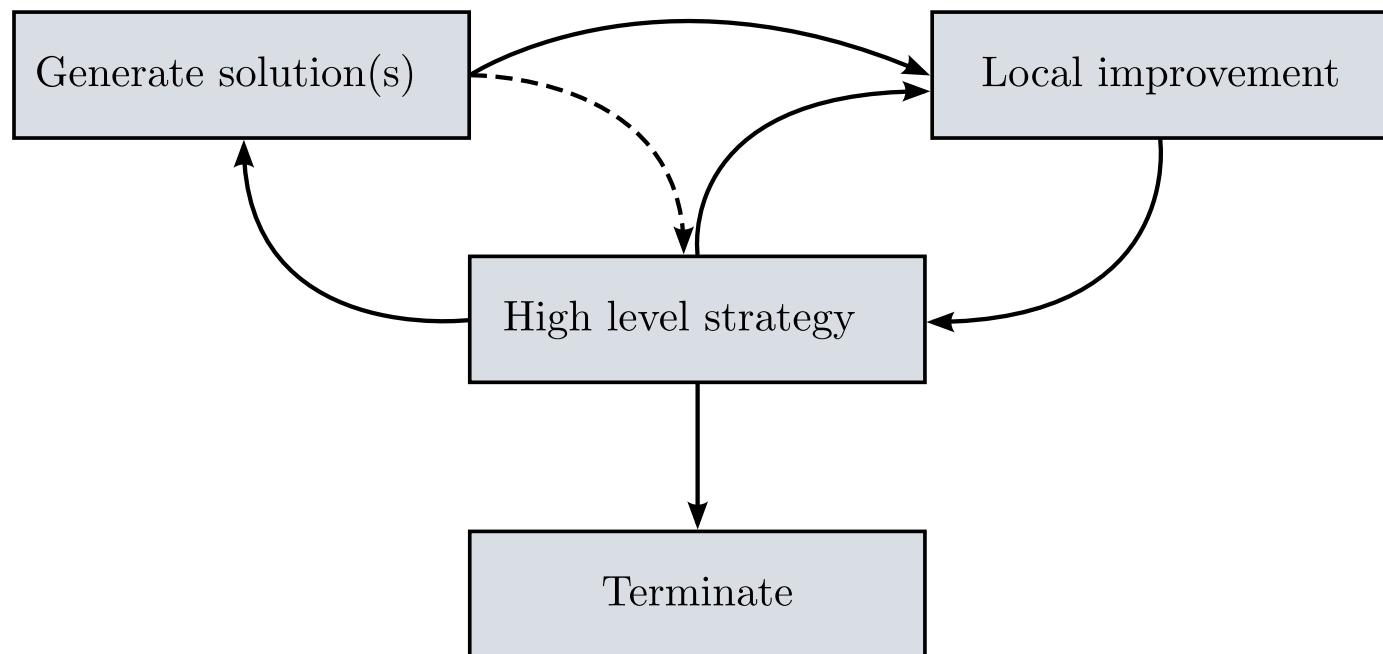
## Definition 2

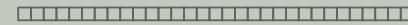
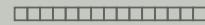
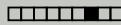
“[M]etaheuristics are 'solution methods that orchestrate an interaction between **local improvement procedures** and **higher level strategies** to create a process capable of escaping from **local optima** and performing a robust search of a **solution space**. ' ”

– Michel Gendreau and Jean-Yves Potvin, *Handbook of Metaheuristics, Second Edition*



# Overview of Metaheuristics

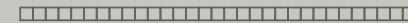
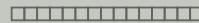
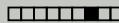




# Heuristics vs. Metaheuristics

## Heuristics

- Problem specific
- Find good solutions, but can get stuck in a local optimum
- Often focus on solving a particular part or structure of a problem



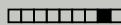
# Heuristics vs. Metaheuristics

## Heuristics

- Problem specific
- Find good solutions, but can get stuck in a local optimum
- Often focus on solving a particular part or structure of a problem

## Metaheuristics

- General, problem independent
- Built in strategies to avoid local optima (can and will get stuck anyway)
- “High-level” methods for selecting and applying heuristics



# Types of heuristics

## Construction heuristics:

- Iteratively builds a solution
- (Usually) decomposes a solution into solution components



# Types of heuristics

## Construction heuristics:

- Iteratively builds a solution
- (Usually) decomposes a solution into solution components

## Neighborhood heuristics:

- Provides one or more solutions based on a given solution
- More on these today and in next weeks...



# Classes of heuristics

**The three main classes of heuristics:**

- Greedy

(There are of course many other types of heuristics, but most fall into one of those three categories.)

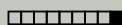


# Classes of heuristics

**The three main classes of heuristics:**

- Greedy
- Random

(There are of course many other types of heuristics, but most fall into one of those three categories.)

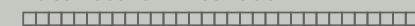
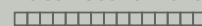
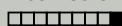


# Classes of heuristics

**The three main classes of heuristics:**

- Greedy
- Random
- Naive

(There are of course many other types of heuristics, but most fall into one of those three categories.)



# Classes of heuristics

**The three main classes of heuristics:**

- Greedy
- Random
- Naive

(There are of course many other types of heuristics, but most fall into one of those three categories.)

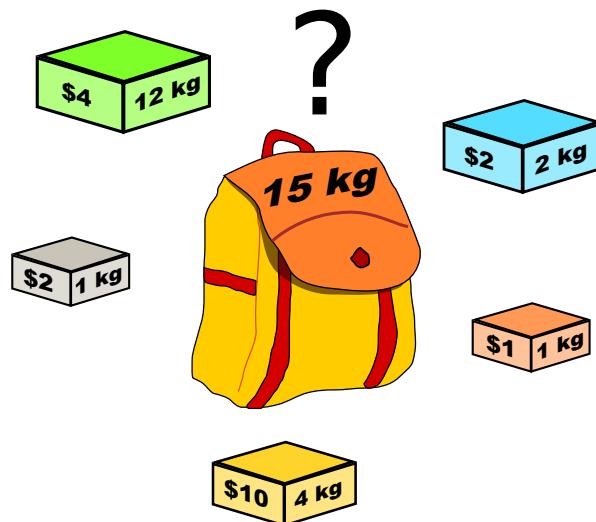


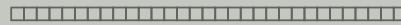
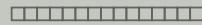
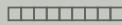
# Classes of heuristics

The three main classes of heuristics:

- Greedy
- Random
- Naive

(There are of course many other types of heuristics, but most fall into one of those three categories.)





# Example heuristics

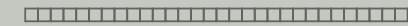
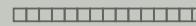
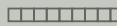


# Example: Set covering problem (SCP)

- Given: A set  $U$  and  $n$  subsets  $S_i \subset U$  with costs  $c_i$ .
- Goal: Find a cover with minimal costs such that the union of the cover is equal to  $U$ .

## Example

	$U = \{1, \dots, 6\}$								
$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11



# Example: Set covering problem (SCP)

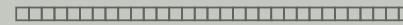
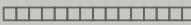
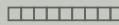
- Given: A set  $U$  and  $n$  subsets  $S_i \subset U$  with costs  $c_i$ .
- Goal: Find a cover with minimal costs such that the union of the cover is equal to  $U$ .

## Example

	$U = \{1, \dots, 6\}$								
$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

## Question

How would you create a solution for this problem?



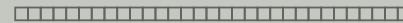
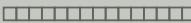
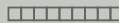
# Initial solution: a greedy algorithm

- Algorithm sketch: add the subset with the lowest costs to the cover such that the number of uncovered elements that become covered is maximized.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- Cover:**  $\emptyset$
- Missing Elements:** { 1 2 3 4 5 6 }
- Costs:** 0



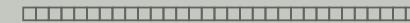
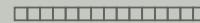
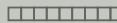
# Initial solution: a greedy algorithm

- Algorithm sketch: add the subset with the lowest costs to the cover such that the number of uncovered elements that become covered is maximized.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- Cover:** {1 3 }
- Missing Elements:** { 2 4 5 6 }
- Costs:** 3



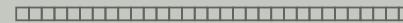
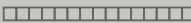
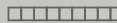
# Initial solution: a greedy algorithm

- Algorithm sketch: add the subset with the lowest costs to the cover such that the number of uncovered elements that become covered is maximized.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- Cover: {1 3 5}
- Missing Elements: { 2 4 6 }
- Costs: 7



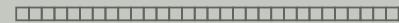
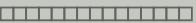
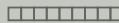
# Initial solution: a greedy algorithm

- Algorithm sketch: add the subset with the lowest costs to the cover such that the number of uncovered elements that become covered is maximized.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- Cover:** {1 2 3 5 6}
- Missing Elements:** { }
- Costs:** 14



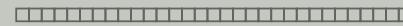
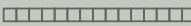
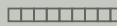
# Initial solution: a greedy algorithm

- Algorithm sketch: add the subset with the lowest costs to the cover such that the number of uncovered elements that become covered is maximized.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- Cover:** {1 2 3 4 5 6}
- Missing Elements:**  $\emptyset$
- Costs:** 23



# Initial solution: a greedy algorithm

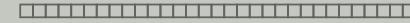
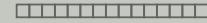
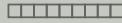
- Algorithm sketch: add the subset with the lowest costs to the cover such that the number of uncovered elements that become covered is maximized.

Example with  $U = \{1, \dots, 6\}$

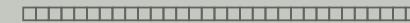
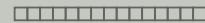
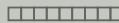
$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- Cover:** {1 2 3 4 5 6}
- Missing Elements:**  $\emptyset$
- Costs:** 23

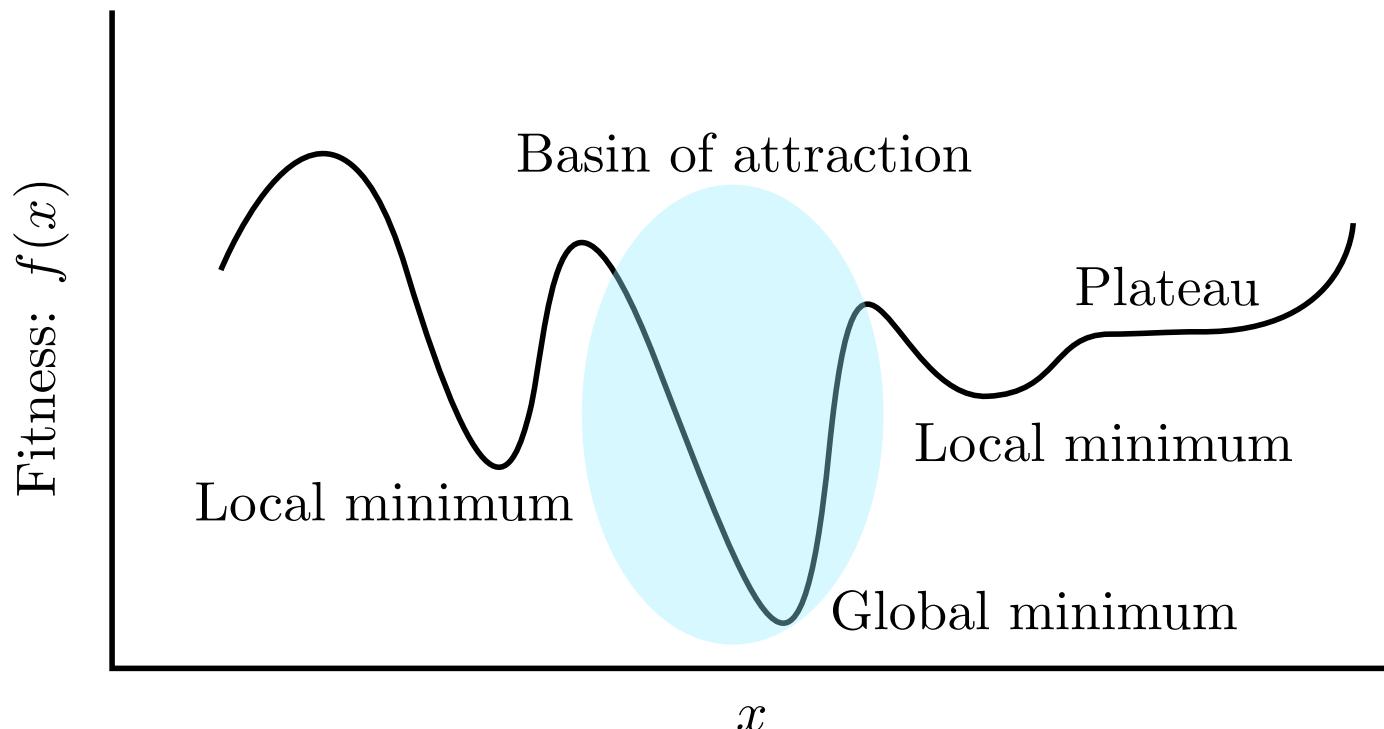
But how good is this solution?

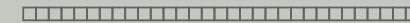
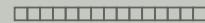
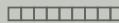


# Fitness functions

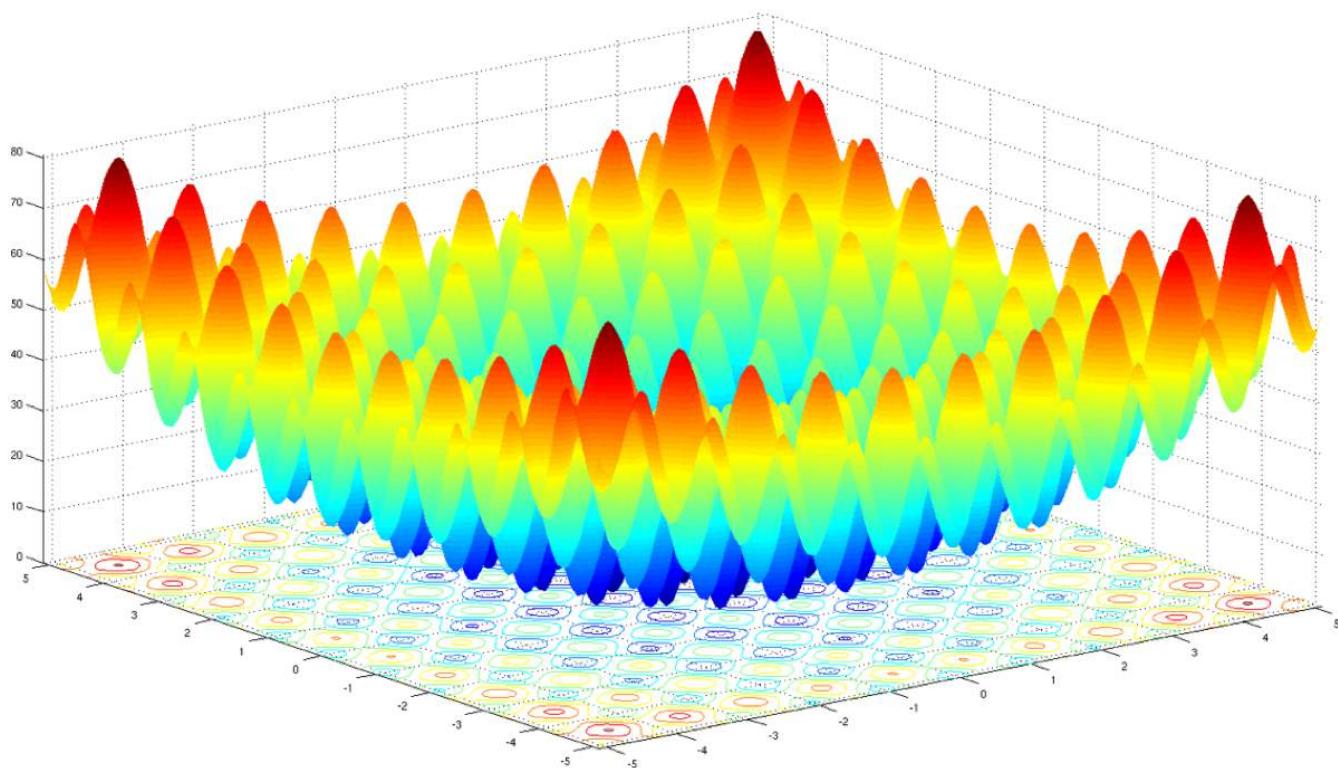


# Objective function (a.k.a. the fitness function)

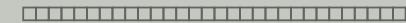
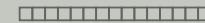
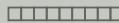




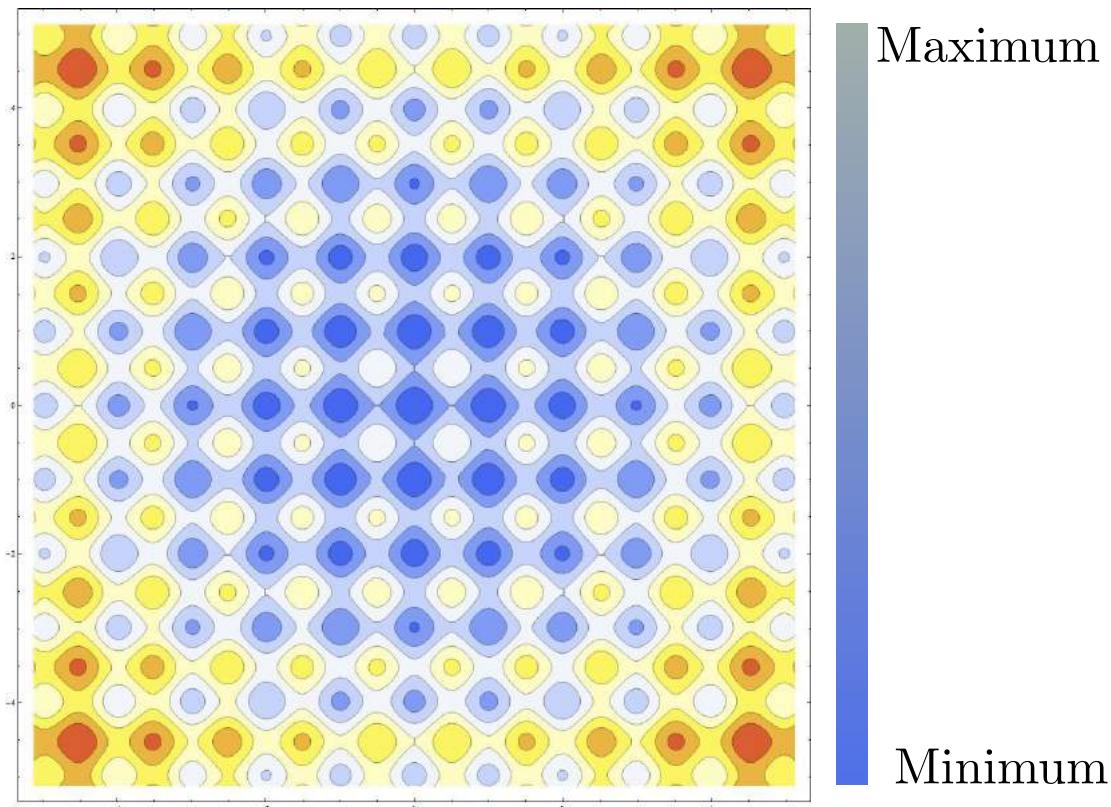
# Objective function (cont.)



Rastrigin function with  $n = 2$ .

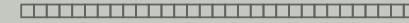
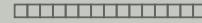
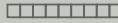


# Objective function (cont.)



Rastrigin function contour plot ( $n = 2$ ).

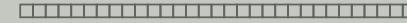
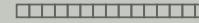
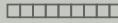
(Modified from original; [Original](#) from Wikipedia User Tos.)



# Systematic vs. local search

## Systematic vs. local search

“*Systematic search algorithms* traverse the search space of a problem instance in a systematic manner which guarantees that eventually either a (optimal) solution is found, or, if no solution exists, this fact is determined with certainty” – Holger H. Hoos und Thomas Stützle, *Stochastic Local Search: Foundations and Applications*.



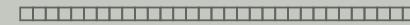
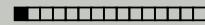
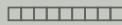
# Systematic vs. local search

## Systematic vs. local search

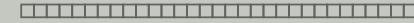
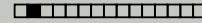
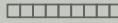
“*Systematic search algorithms* traverse the search space of a problem instance in a systematic manner which guarantees that eventually either a (optimal) solution is found, or, if no solution exists, this fact is determined with certainty” – Holger H. Hoos und Thomas Stützle, *Stochastic Local Search: Foundations and Applications*.

## Optimality

As with most Metaheuristics, local search gives no guarantee that the optimal solution will be found.



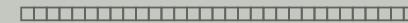
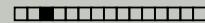
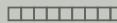
## Local search example



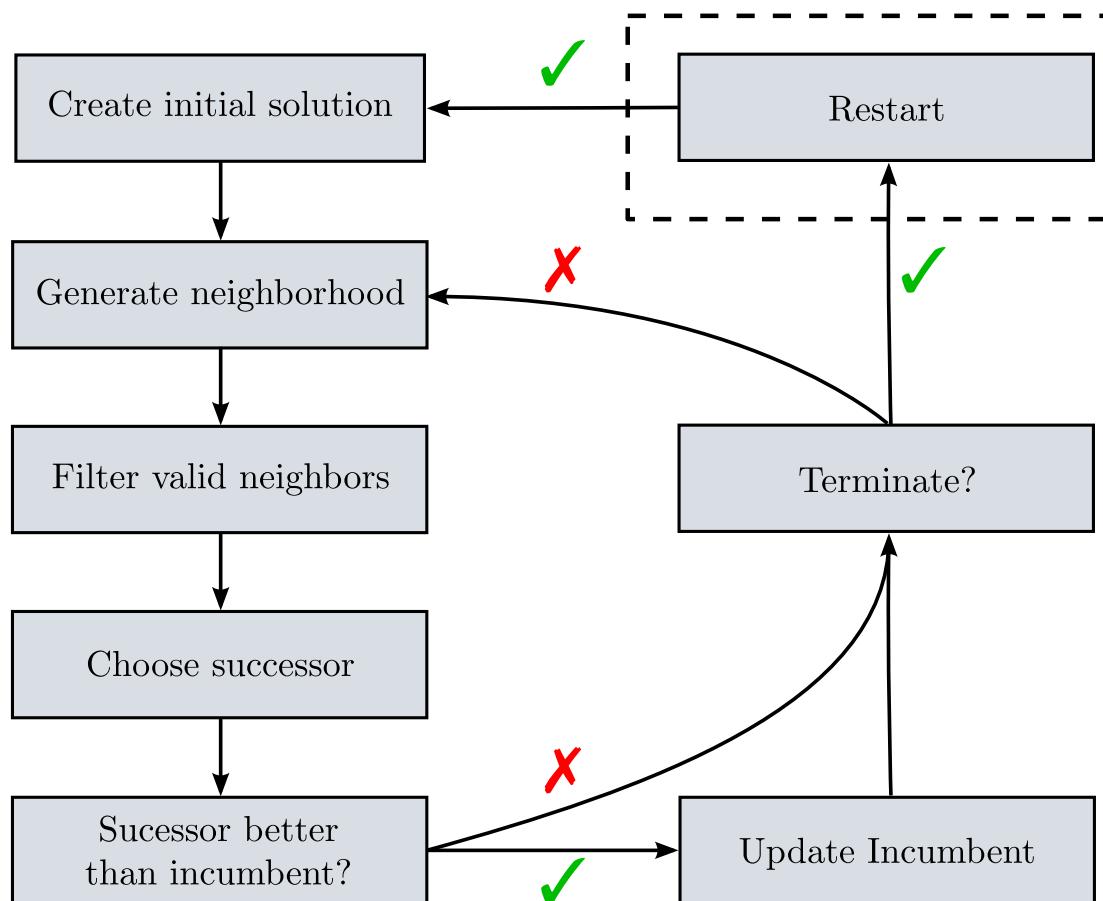
# Local search

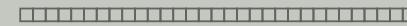
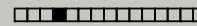
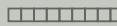
## Definition

Local search methods are a group of metaheuristics that iteratively improve a solution according to a search strategy.



# Local search





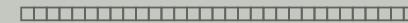
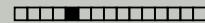
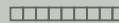
# Local search formalized

```

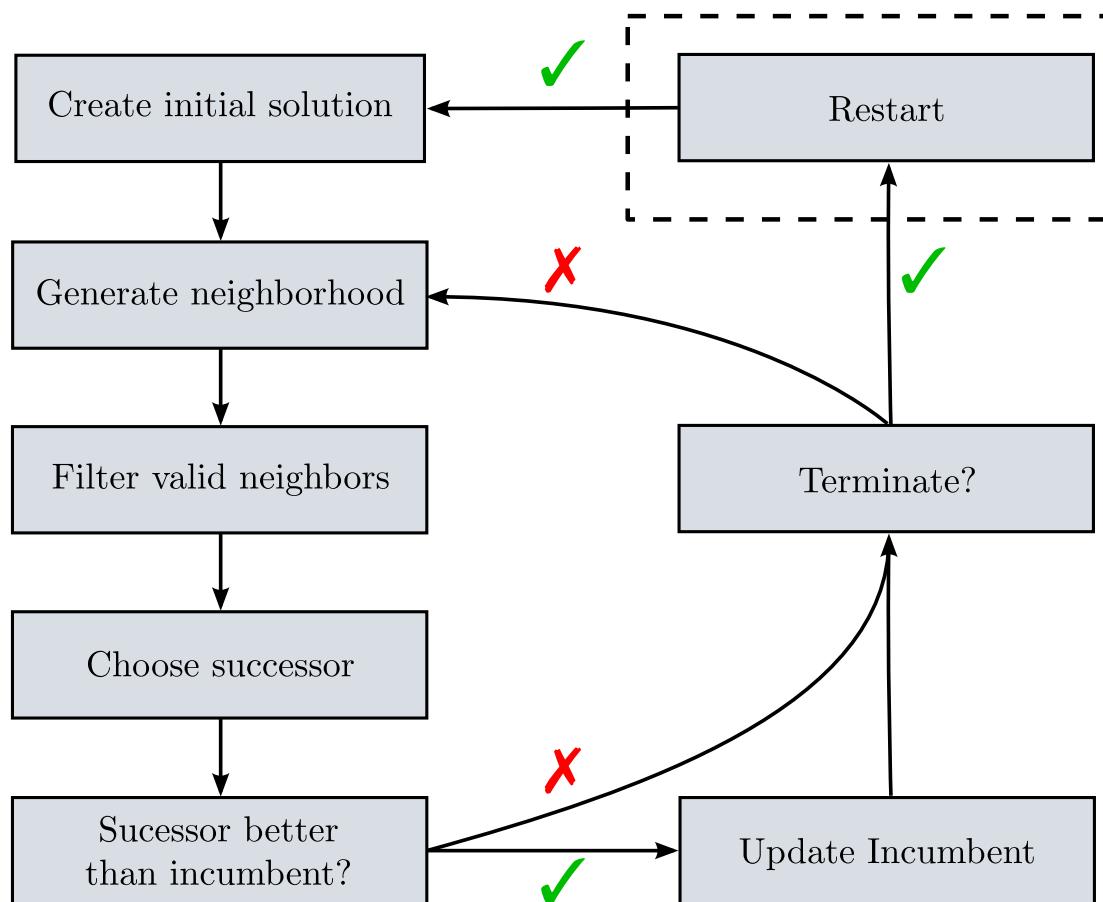
1: function LS-MIN( $f, C, N, L, StartSolution, Terminate$ )
2:    $s \leftarrow StartSolution()$ 
3:    $s^* \leftarrow s$ 
4:   repeat
5:      $s \leftarrow C(L(N(s)))$ 
6:     if  $f(s) < f(s^*)$  then
7:        $s^* \leftarrow s$ 
8:   until  $Terminate(s)$ 

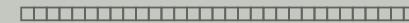
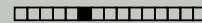
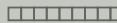
```

$S$	Solution space
$f$	Objective function, a.k.a fitness function ( $f : s \rightarrow \mathbb{R}, s \in S$ )
$N$	Neighborhood relation ( $N : S \rightarrow 2^{ S }$ )
$L$	Neighborhood filter ( $L : 2^{ S } \rightarrow 2^{ S }$ )
$C$	Neighbor selection function ( $C : 2^{ S } \rightarrow S$ )



# Local Improvement





# Neighborhoods: SCP example

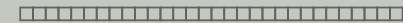
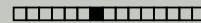
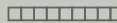
## Question

This is our SCP solution from the previous slide. What could we do to improve the solution?

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 4 5 6}
- **Missing Elements:**  $\emptyset$
- **Costs:** 23



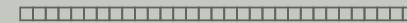
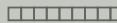
# Neighborhoods: Drop & Add

- 1 Remove a random subset from the cover with uniform probability.
- 2 Repair the cover with some algorithm (greedy, random, etc).
- 3 Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 4 5 6}
- **Missing Elements:**  $\emptyset$
- **Costs:** 23



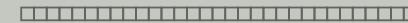
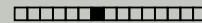
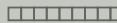
# Neighborhoods: Drop & Add

- 1  $\Rightarrow$  Remove a random subset from the cover with uniform probability.
- 2 Repair the cover with some algorithm (greedy, random, etc).
- 3 Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 4 5 6}
- **Missing Elements:**  $\emptyset$
- **Costs:** 23



# Neighborhoods: Drop & Add

- 1  $\Rightarrow$  Remove a random subset from the cover with uniform probability.
- 2 Repair the cover with some algorithm (greedy, random, etc).
- 3 Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 4 6}
- **Missing Elements:** {5}
- **Costs:** 19



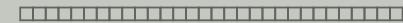
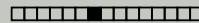
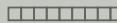
# Neighborhoods: Drop & Add

- 1 Remove a random subset from the cover with uniform probability.
- 2  $\Rightarrow$  Repair the cover with some algorithm (greedy, random, etc).
- 3 Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 4 5 6}
- **Missing Elements:**  $\emptyset$
- **Costs:** 26



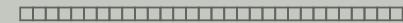
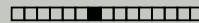
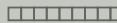
# Neighborhoods: Drop & Add

- 1  $\Rightarrow$  Remove a random subset from the cover with uniform probability.
- 2 Repair the cover with some algorithm (greedy, random, etc).
- 3 Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 4 5 6}
- **Missing Elements:**  $\emptyset$
- **Costs:** 26



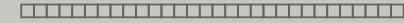
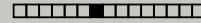
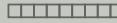
# Neighborhoods: Drop & Add

- 1  $\Rightarrow$  Remove a random subset from the cover with uniform probability.
- 2 Repair the cover with some algorithm (greedy, random, etc).
- 3 Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 5 6}
- **Missing Elements:** {4}
- **Costs:** 17



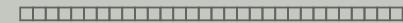
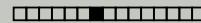
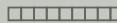
# Neighborhoods: Drop & Add

- 1 Remove a random subset from the cover with uniform probability.
- 2  $\Rightarrow$  Repair the cover with some algorithm (greedy, random, etc).
- 3 Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 4 5 6}
- **Missing Elements:**  $\emptyset$
- **Costs:** 27



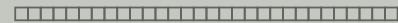
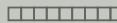
# Neighborhoods: Drop & Add

- 1 Remove a random subset from the cover with uniform probability.
- 2 Repair the cover with some algorithm (greedy, random, etc).
- 3  $\Rightarrow$  Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 4 5 6}
- **Missing Elements:**  $\emptyset$
- **Costs:** 27



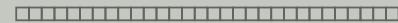
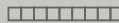
# Neighborhoods: Drop & Add

- 1 Remove a random subset from the cover with uniform probability.
- 2 Repair the cover with some algorithm (greedy, random, etc).
- 3 Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

- **Cover:** {1 2 3 4 5 6}
- **Missing Elements:**  $\emptyset$
- **Costs:** 17



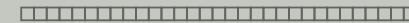
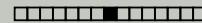
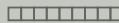
# Neighborhoods: Drop & Add

- 1 Remove a random subset from the cover with uniform probability.
- 2 Repair the cover with some algorithm (greedy, random, etc).
- 3 Throw away superfluous subsets.

Example with  $U = \{1, \dots, 6\}$

$i$	1	2	3	4	5	6	7	8	9
$S_i$	{1,3}	{3,5}	{1,2,5}	{2,6}	{1,2}	{4}	{3,4,6}	{5}	{4,5,6}
$c_i$	3	4	7	7	8	9	10	10	11

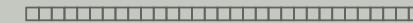
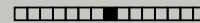
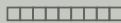
- **Cover:** {1 2 3 4 5 6}
- **Missing Elements:**  $\emptyset$
- **Costs:** 17 → **Optimal!**



# Neighborhoods formalized

## Neighborhood

Given a solution, a *neighborhood relation* (often just referred to as a neighborhood), provides “near-by” solutions in the solution space.



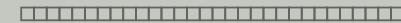
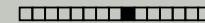
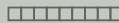
# Neighborhoods formalized

## Neighborhood

Given a solution, a *neighborhood relation* (often just referred to as a neighborhood), provides “near-by” solutions in the solution space.

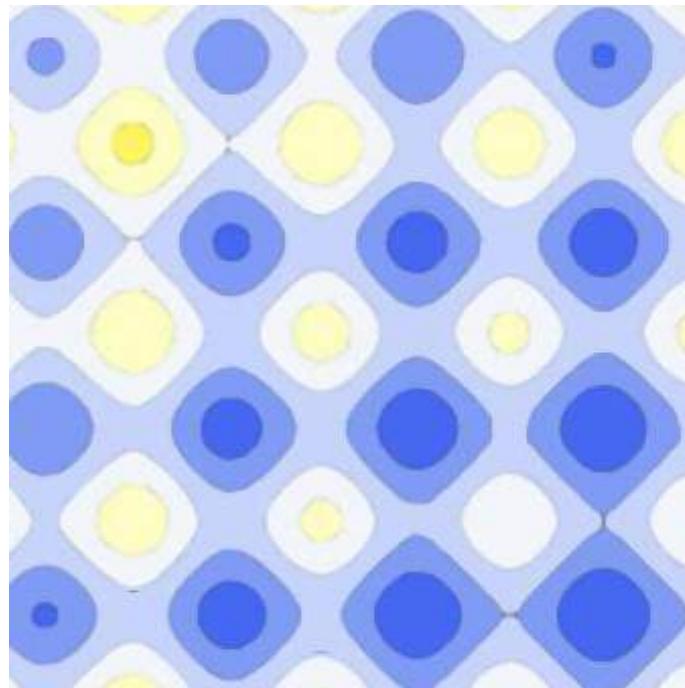
## Neighborhood relation

Given a solution  $s \in S$ , where  $S$  is the space of solutions, a neighborhood relation is a function  $n : S \rightarrow 2^{|S|}$ , accepting  $s$  and returning a set of solutions from  $S$  (i.e.,  $n(s) \subseteq S$ ).

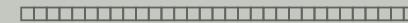
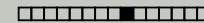
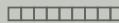


# Fitness landscapes

- Fitness landscapes are graphs defined by a *neighborhood relation*.

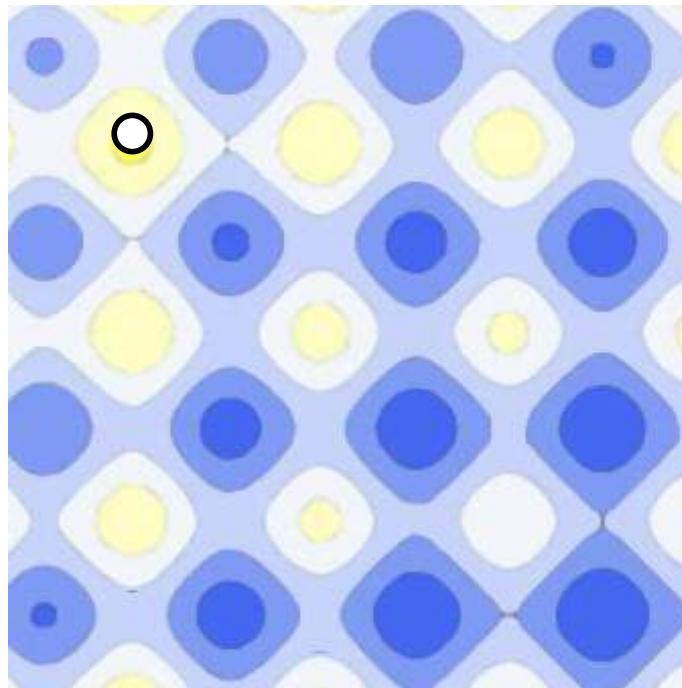


Rastrigin function contour plot (Enlarged) ( $n = 2$ ).  
(Modified from original; [Original](#) from Wikipedia User Tos.)

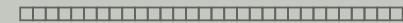
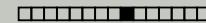
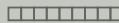


# Fitness landscapes

- Fitness landscapes are graphs defined by a *neighborhood relation*.

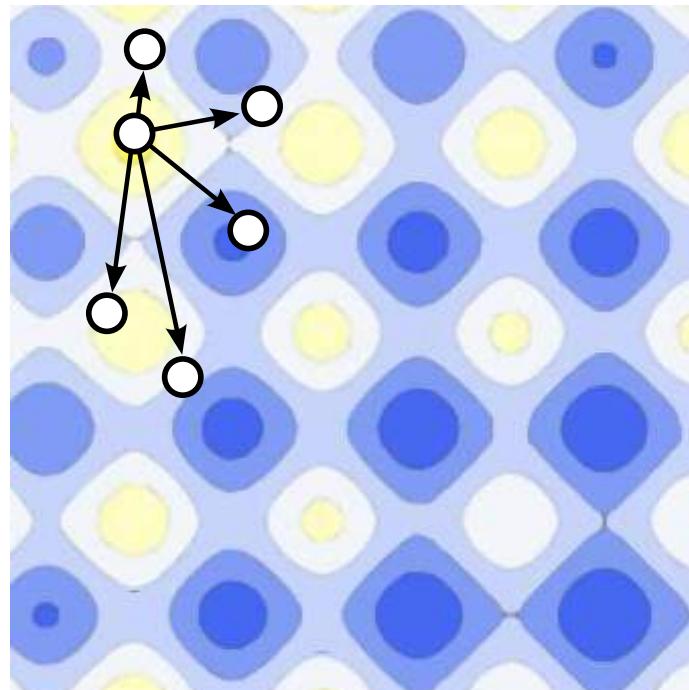


Rastrigin function contour plot (Enlarged) ( $n = 2$ ).  
(Modified from original; [Original](#) from Wikipedia User Tos.)

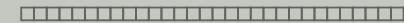
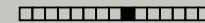
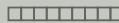


# Fitness landscapes

- Fitness landscapes are graphs defined by a *neighborhood relation*.

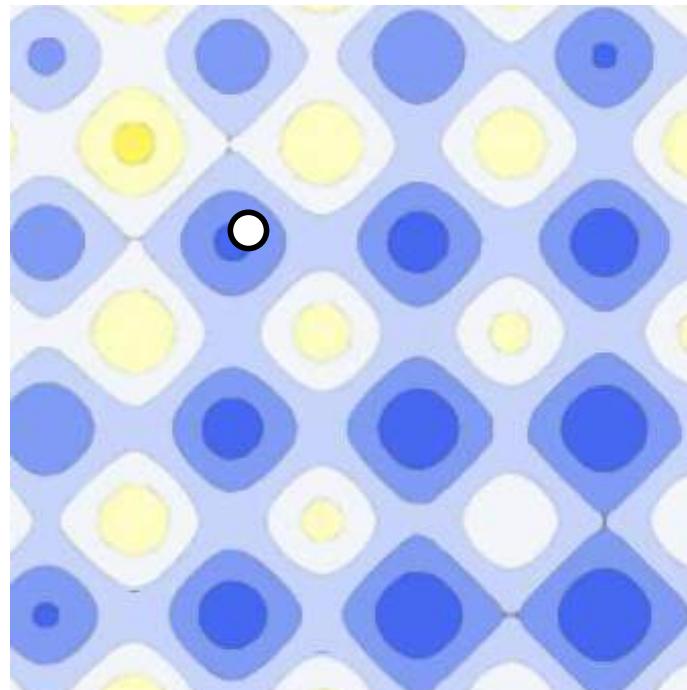


Rastrigin function contour plot (Enlarged) ( $n = 2$ ).  
(Modified from original; [Original](#) from Wikipedia User Tos.)

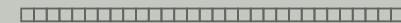
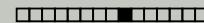
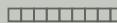


# Fitness landscapes

- Fitness landscapes are graphs defined by a *neighborhood relation*.

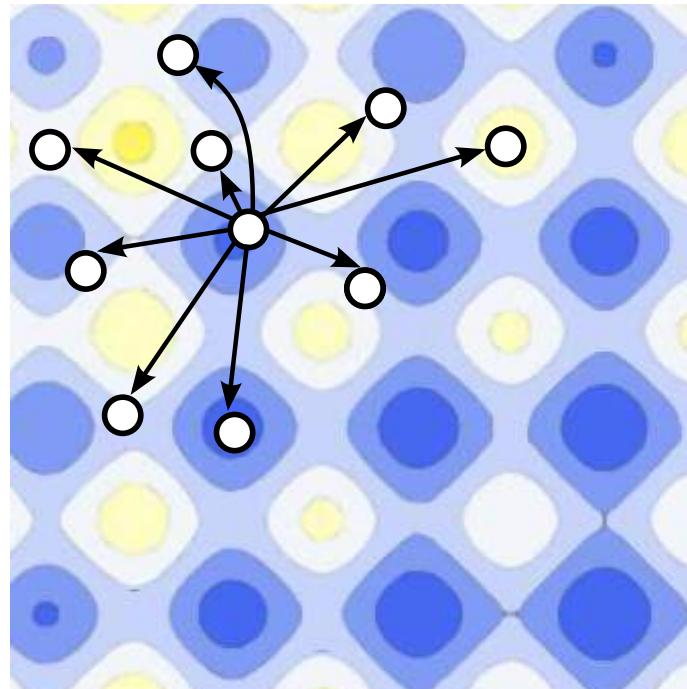


Rastrigin function contour plot (Enlarged) ( $n = 2$ ).  
(Modified from original; [Original](#) from Wikipedia User Tos.)

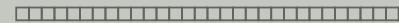
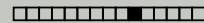
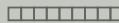


# Fitness landscapes

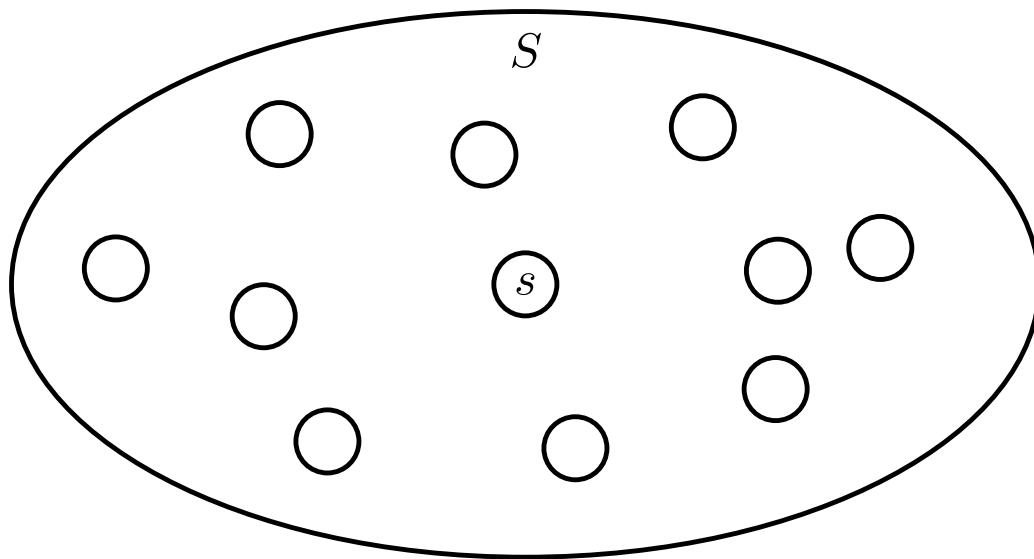
- Fitness landscapes are graphs defined by a *neighborhood relation*.

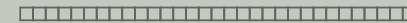
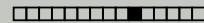
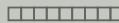


Rastrigin function contour plot (Enlarged) ( $n = 2$ ).  
(Modified from original; [Original](#) from Wikipedia User Tos.)

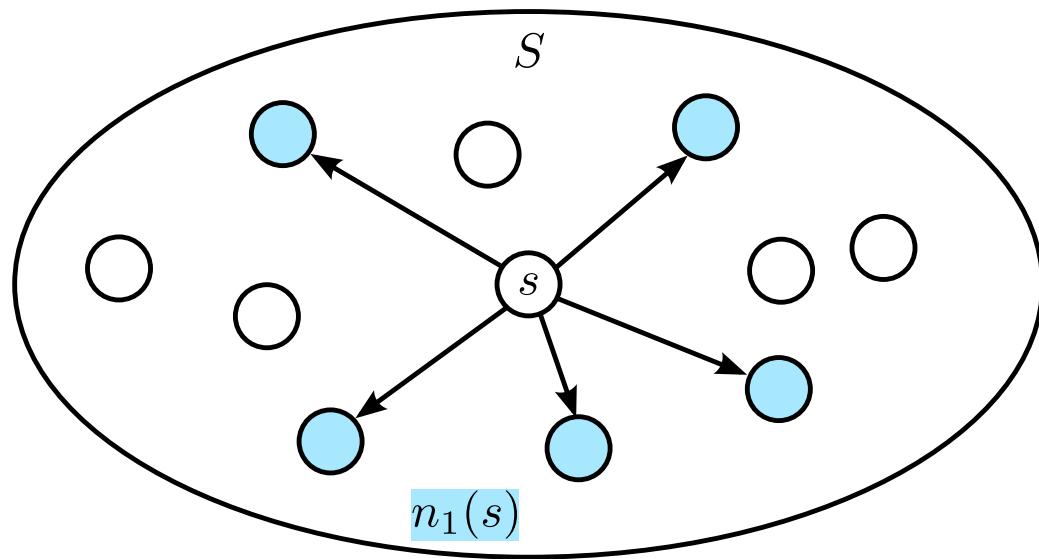


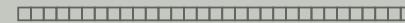
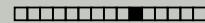
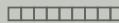
# Neighborhoods: in pictures



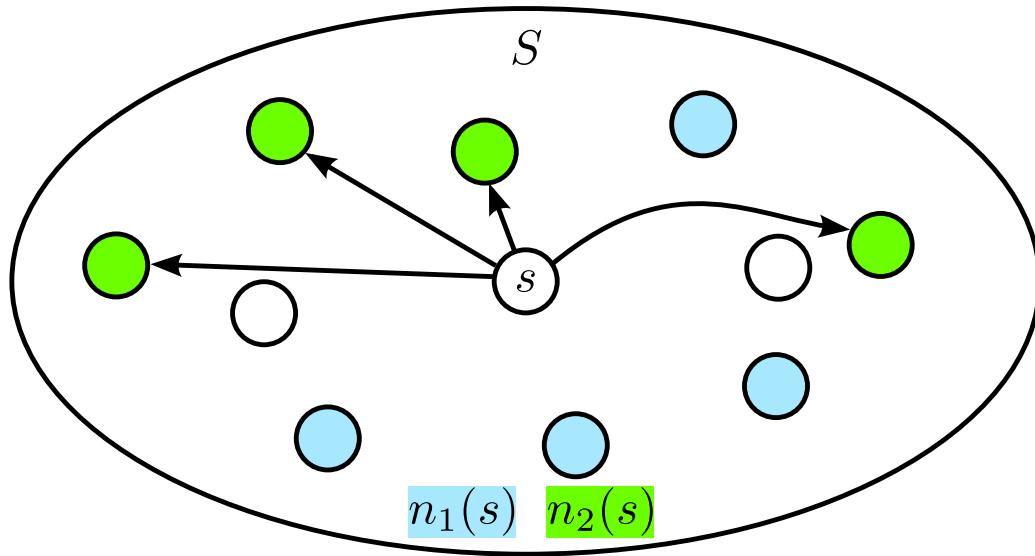


# Neighborhoods: in pictures

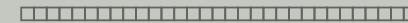
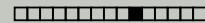
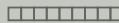




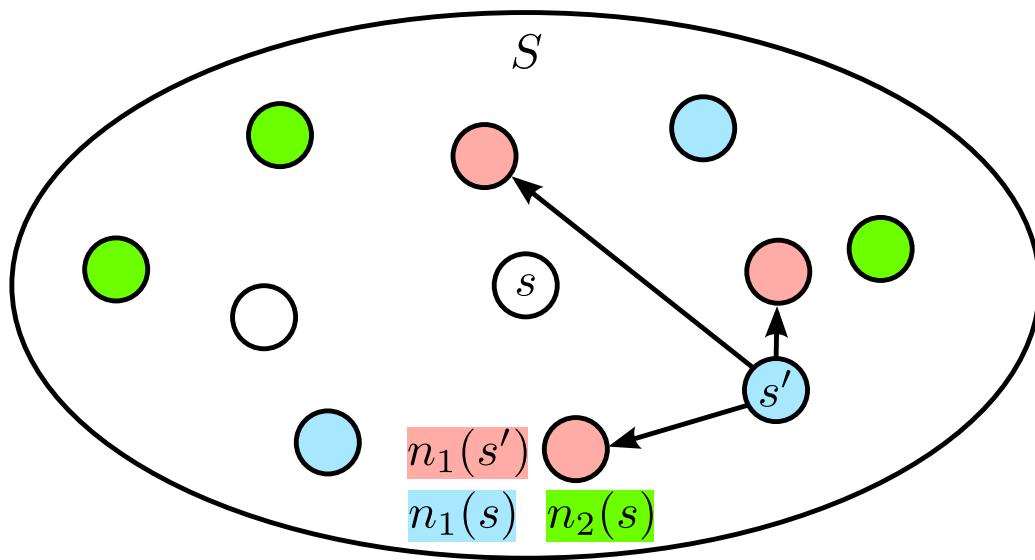
# Neighborhoods: in pictures



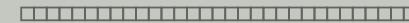
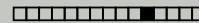
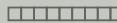
- Different neighborhoods may share common solutions.



# Neighborhoods: in pictures



- Different neighborhoods may share common solutions.
- Neighbor relations are not necessarily symmetric.

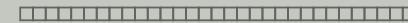
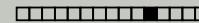
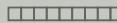


# High-level strategies

- We chose a neighbor from the neighborhood.
- What now?

## Question

Should we accept this neighbor? (i.e., replace our current solution with it)



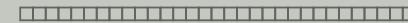
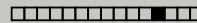
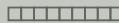
# High-level strategies

- We chose a neighbor from the neighborhood.
- What now?

## Question

Should we accept this neighbor? (i.e., replace our current solution with it)

- Many different metaheuristics try to answer this question.
- But: No strategy dominates all other strategies! (See: No Free Lunch Theorem)

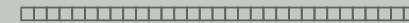
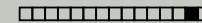
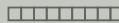


# Neighborhoods: choosing a successor

- Which neighbor of a solution should we choose to replace our current solution?
- Common options:
  - 1 Best successor
  - 2 First successor with a better objective function value in the neighborhood
  - 3 Random successor
  - 4 Combinations of 1 – 3
- Different strategies work best for different problems: Just try out multiple options!

# High-level strategies

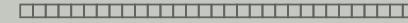
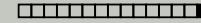
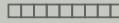
- Several well-known local search strategies:
  - **Hill climbing** [iterative improvement]: only accept better solutions
  - **Tabu search**: only accept solutions that are not in the list of already seen solutions
  - **Simulated annealing**: better solutions are accepted always, worse solutions are accepted according to a distribution which changes during the search
- More complicated strategies:
  - **Genetic algorithms**: The evolution of a population is simulated
  - **GRASP**: a greedy heuristic is iteratively applied
  - ... and a lot more



# Termination criteria

## Question

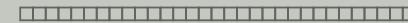
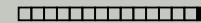
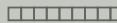
When should we stop the local search / improvement procedure?



# Termination criteria

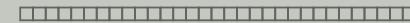
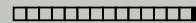
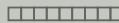
- Budget based criteria

- Maximum number of iterations/evaluations reached
- CPU time limit exceeded



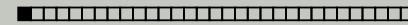
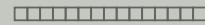
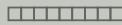
# Termination criteria

- Budget based criteria
  - Maximum number of iterations/evaluations reached
  - CPU time limit exceeded
- Solution quality
  - Within  $x\%$  of a lower bound
  - Business requirements satisfied

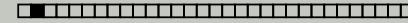
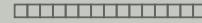
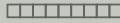


# Termination criteria

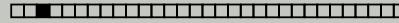
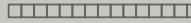
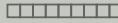
- Budget based criteria
  - Maximum number of iterations/evaluations reached
  - CPU time limit exceeded
- Solution quality
  - Within  $x\%$  of a lower bound
  - Business requirements satisfied
- Convergence criteria
  - No improvement in last  $y$  iterations
  - Average improvement below threshold,  $\epsilon$



## Local search methods



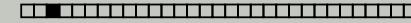
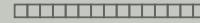
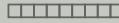
# Iterative improvement



# Iterative improvement

## Definition

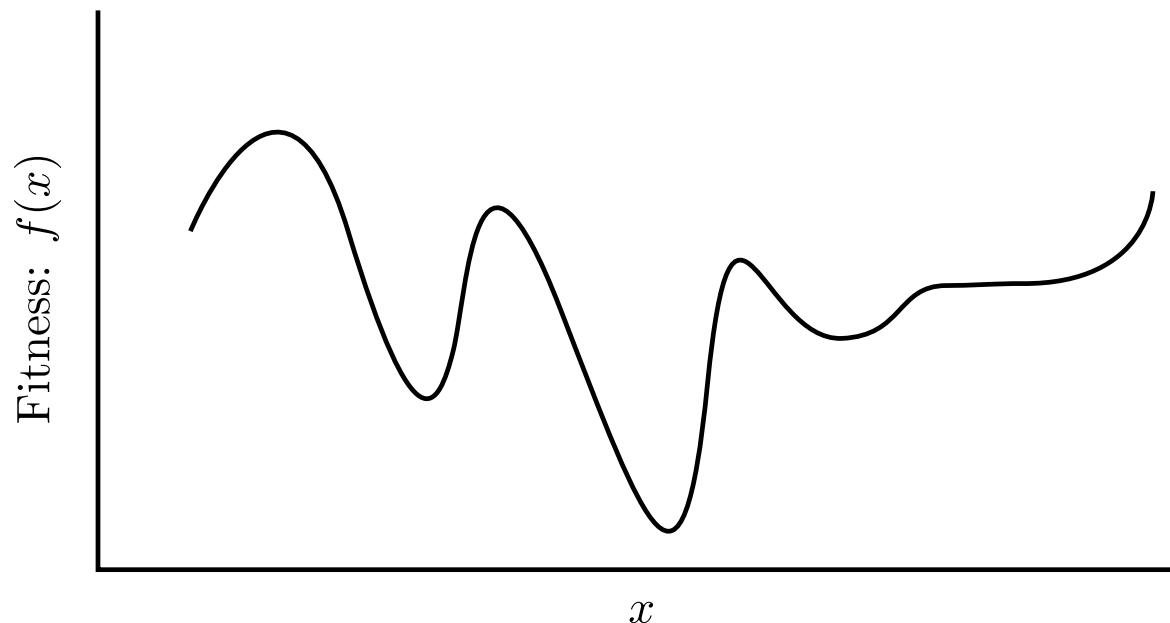
Iterative Improvement (also called Hill Climbing) is a greedy local search method that always selects the best neighbor in the neighborhood as its successor.

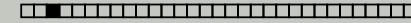
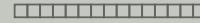
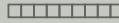


# Iterative improvement

## Definition

Iterative Improvement (also called Hill Climbing) is a greedy local search method that always selects the best neighbor in the neighborhood as its successor.

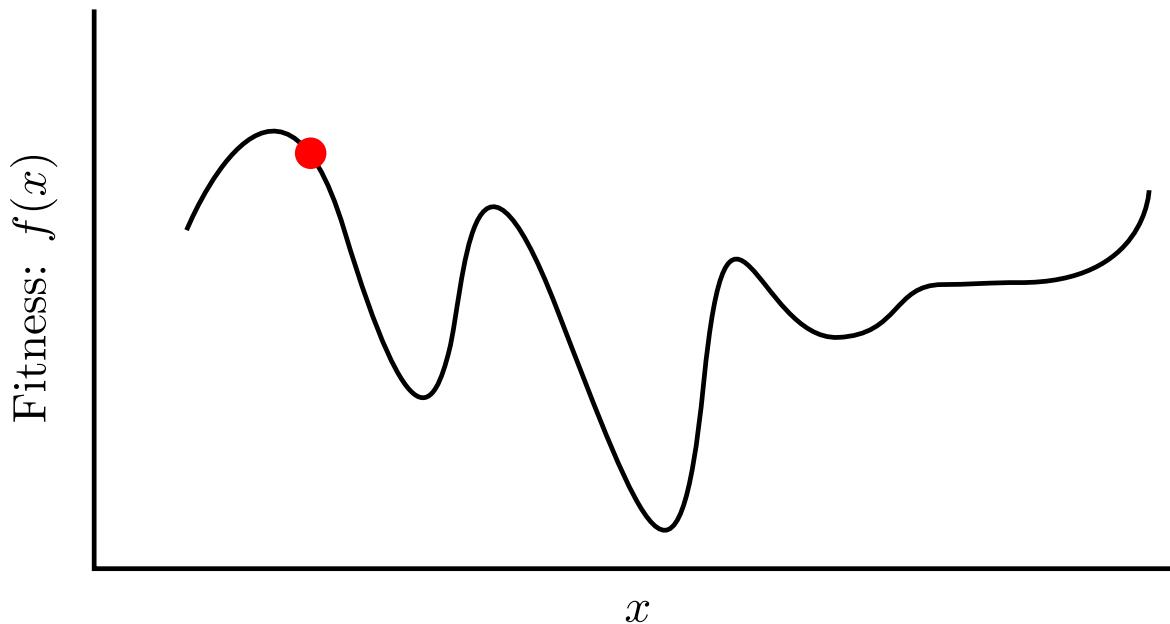


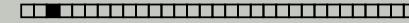
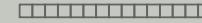
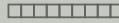


# Iterative improvement

## Definition

Iterative Improvement (also called Hill Climbing) is a greedy local search method that always selects the best neighbor in the neighborhood as its successor.

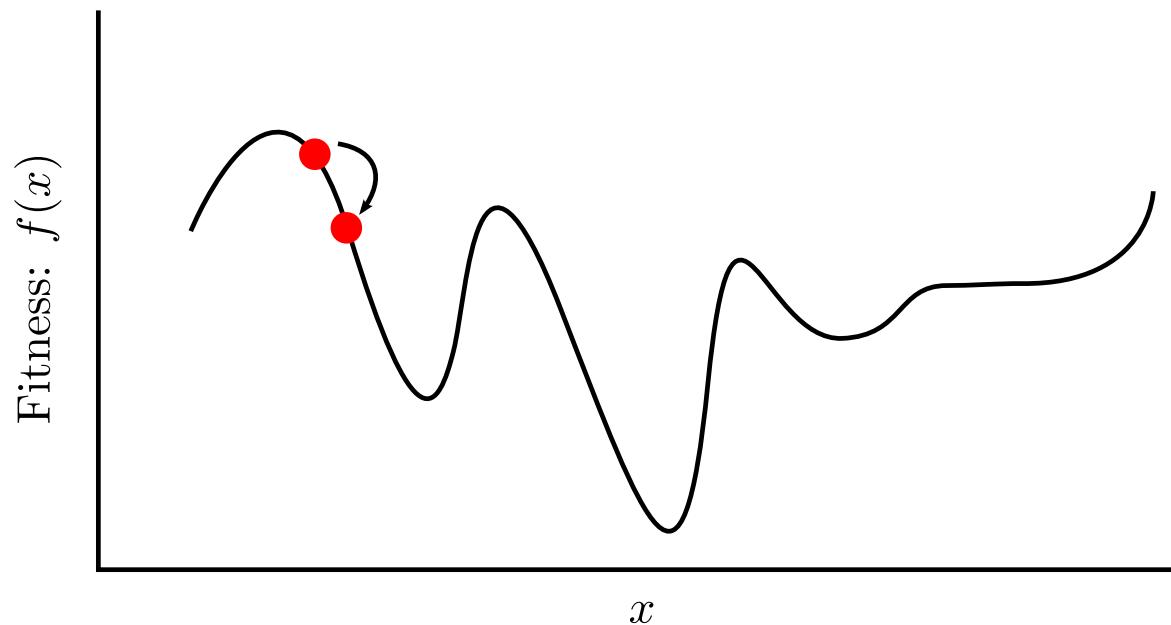


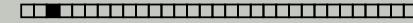
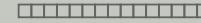
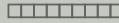


# Iterative improvement

## Definition

Iterative Improvement (also called Hill Climbing) is a greedy local search method that always selects the best neighbor in the neighborhood as its successor.

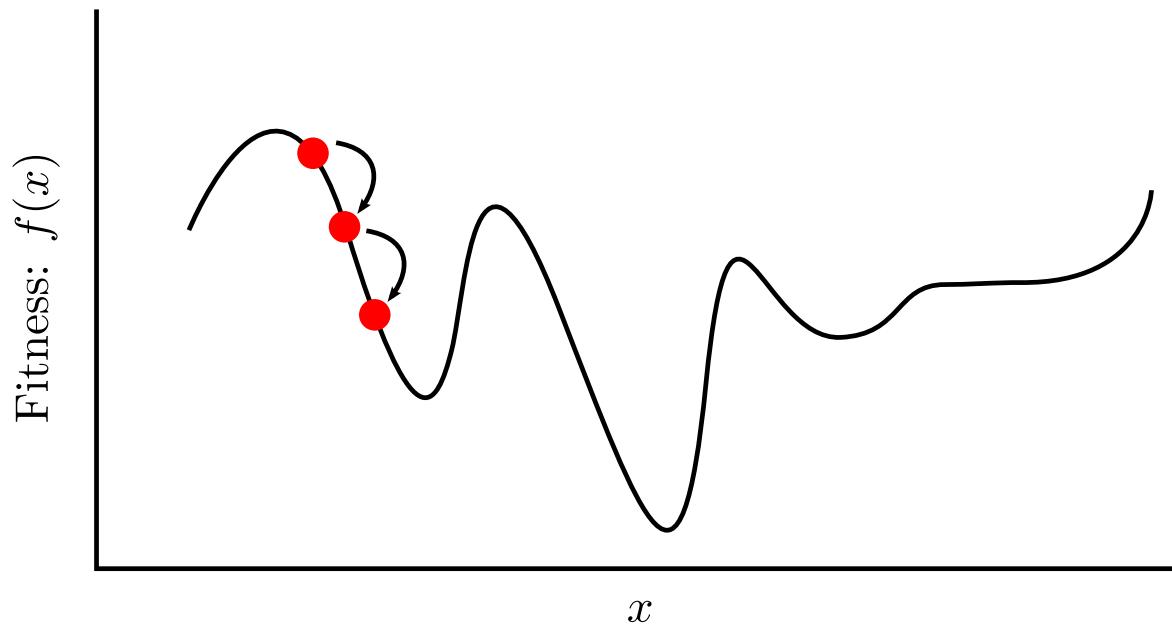


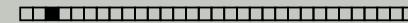
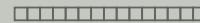
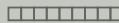


# Iterative improvement

## Definition

Iterative Improvement (also called Hill Climbing) is a greedy local search method that always selects the best neighbor in the neighborhood as its successor.

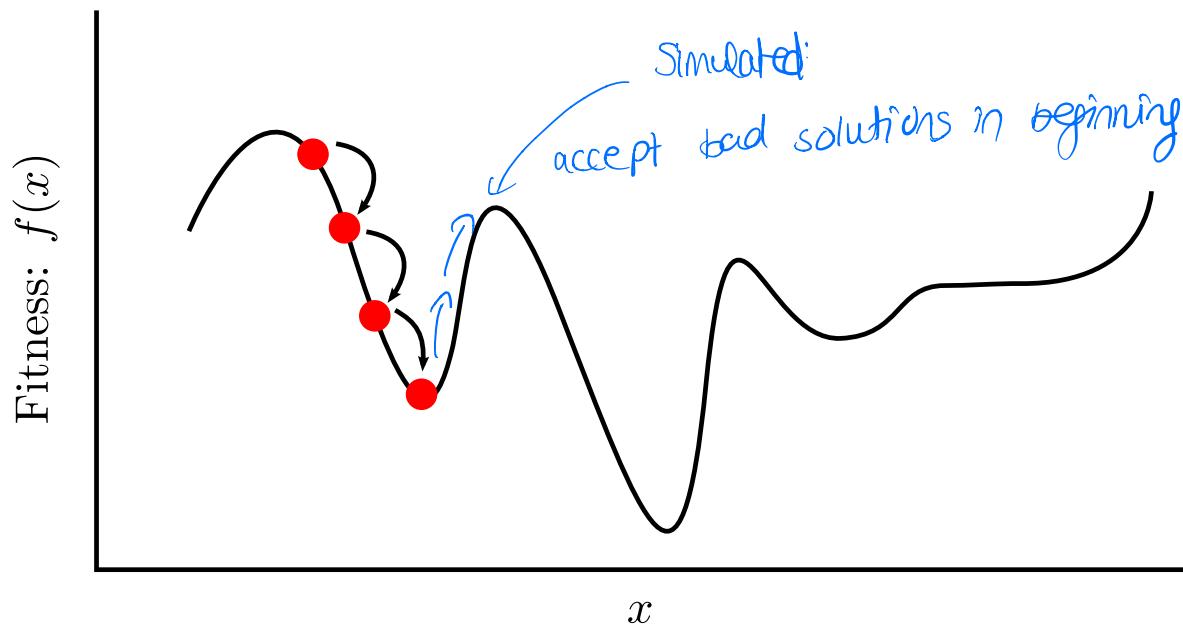


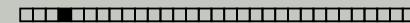
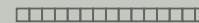
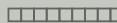


# Iterative improvement

## Definition

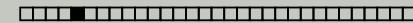
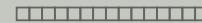
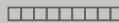
Iterative Improvement (also called Hill Climbing) is a greedy local search method that always selects the best neighbor in the neighborhood as its successor.





# Iterative improvement algorithm

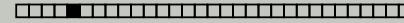
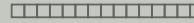
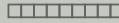
```
1: function II-MIN( $f$ ,  $C$ ,  $N$ ,  $L$ ,  $StartSolution$ ,  $Terminate$ )
2:    $s \leftarrow StartSolution()$ 
3:    $N' \leftarrow \emptyset$ 
4:   repeat
5:      $N' \leftarrow \{s' \in N(s) \mid f(s') < f(s)\}$ 
6:     if  $N' \neq \emptyset$  then
7:        $s \leftarrow \operatorname{argmin}_{s \in N'}\{f(s)\}$ 
8:   until  $N' = \emptyset$ 
```



# Pro and cons

## Pro

- Finds local optimum quickly (usually)



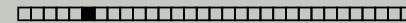
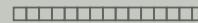
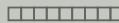
# Pro and cons

## Pro

- Finds local optimum quickly (usually)

## Con

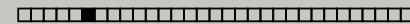
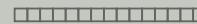
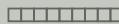
- Gets stuck in local optima easily
- Only works with a small neighborhood (otherwise too slow)



# Iterative improvement variations

## First improvement iterated descent (“Simple hill climbing”)

- Doesn't choose best neighbor from neighborhood
- Instead chooses first neighbor found with improving cost function
- “Lazy” heuristic



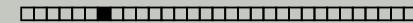
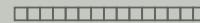
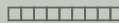
# Iterative improvement variations

## First improvement iterated descent (“Simple hill climbing”)

- Doesn't choose best neighbor from neighborhood
- Instead chooses first neighbor found with improving cost function
- “Lazy” heuristic

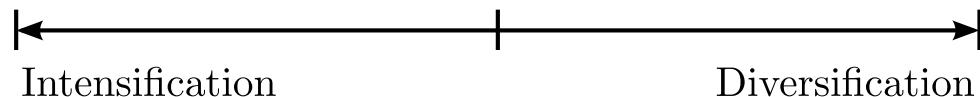
## Random restart iterative improvement

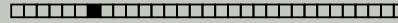
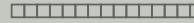
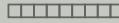
The iterative improvement algorithm is restarted from a new start solution after it converges.



# Intensification vs. diversification

Iterative-Improvement

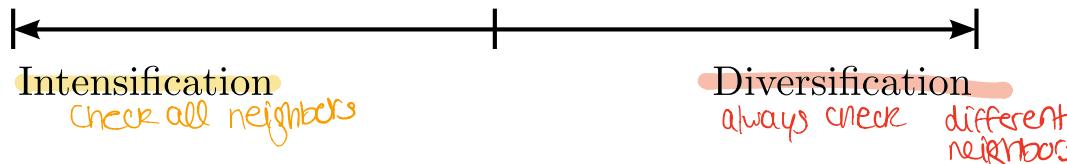




# Intensification vs. diversification

(  
- Greedy  
- Random  
- Iterative

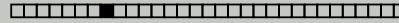
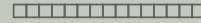
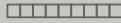
Iterative-Improvement



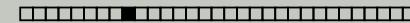
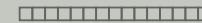
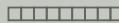
## Question

What would be a possible algorithm that is based only on “diversification”?

Random has higher possibility to always choose solutions that are different



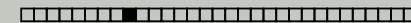
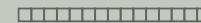
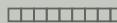
# Simulated Annealing



# SA: introduction

## History

- SA simulates the cooling process of metal.
- The metal is heated and then cooled slowly.
- This gives the atoms in the metal a stable form.



# SA: introduction

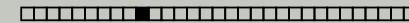
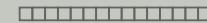
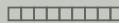
## History

- SA simulates the cooling process of metal.
- The metal is heated and then cooled slowly.
- This gives the atoms in the metal a stable form.

## Core idea

Accept “bad” successors according to a probability distribution.

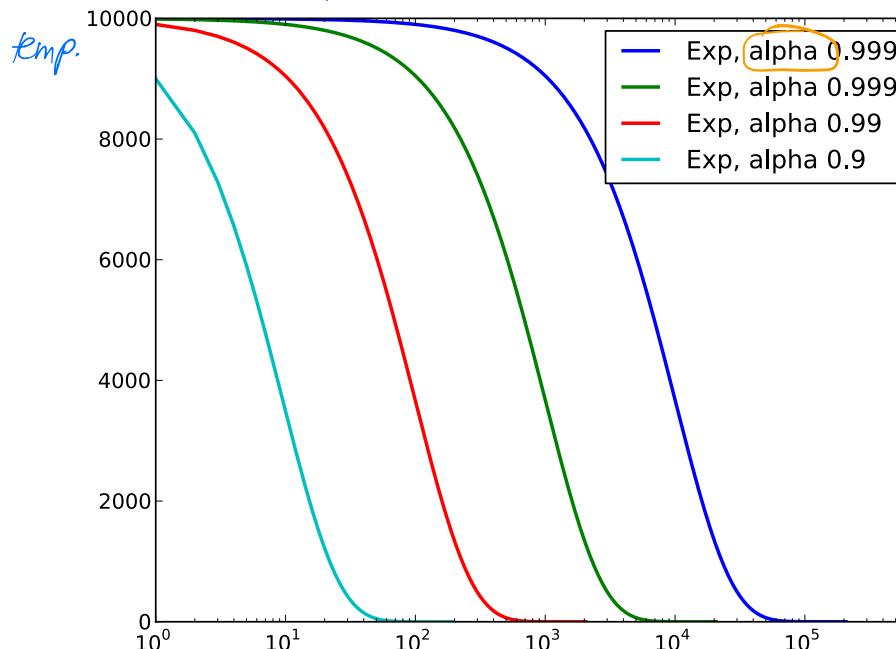
- At the beginning of the search: accept virtually everything
- At the end of the search: accept only good solutions.



# Temperature

- The “temperature” of the system is determined by a *cooling schedule*

*accept all*



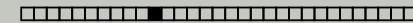
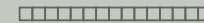
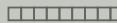
*Cool down step by step*

*only accept good*

Temperature vs. iteration with (standard) exponential cooling:

$$T(i) = \alpha T(i - 1), T(0) = 10000$$

*[0.9; 0.99] temp of last iteration*



# Acceptance criteria

..

## Metropolis criteria

Acceptance probability function:

$$P(s, s', T) = \begin{cases} 1 & \text{if } f(s') < f(s) \\ \exp(-(f(s') - f(s))/T) & \text{otherwise.} \end{cases}$$

$T$  is the previously mentioned *temperature*.

$$T \rightarrow \infty \quad \exp(0) = 1$$

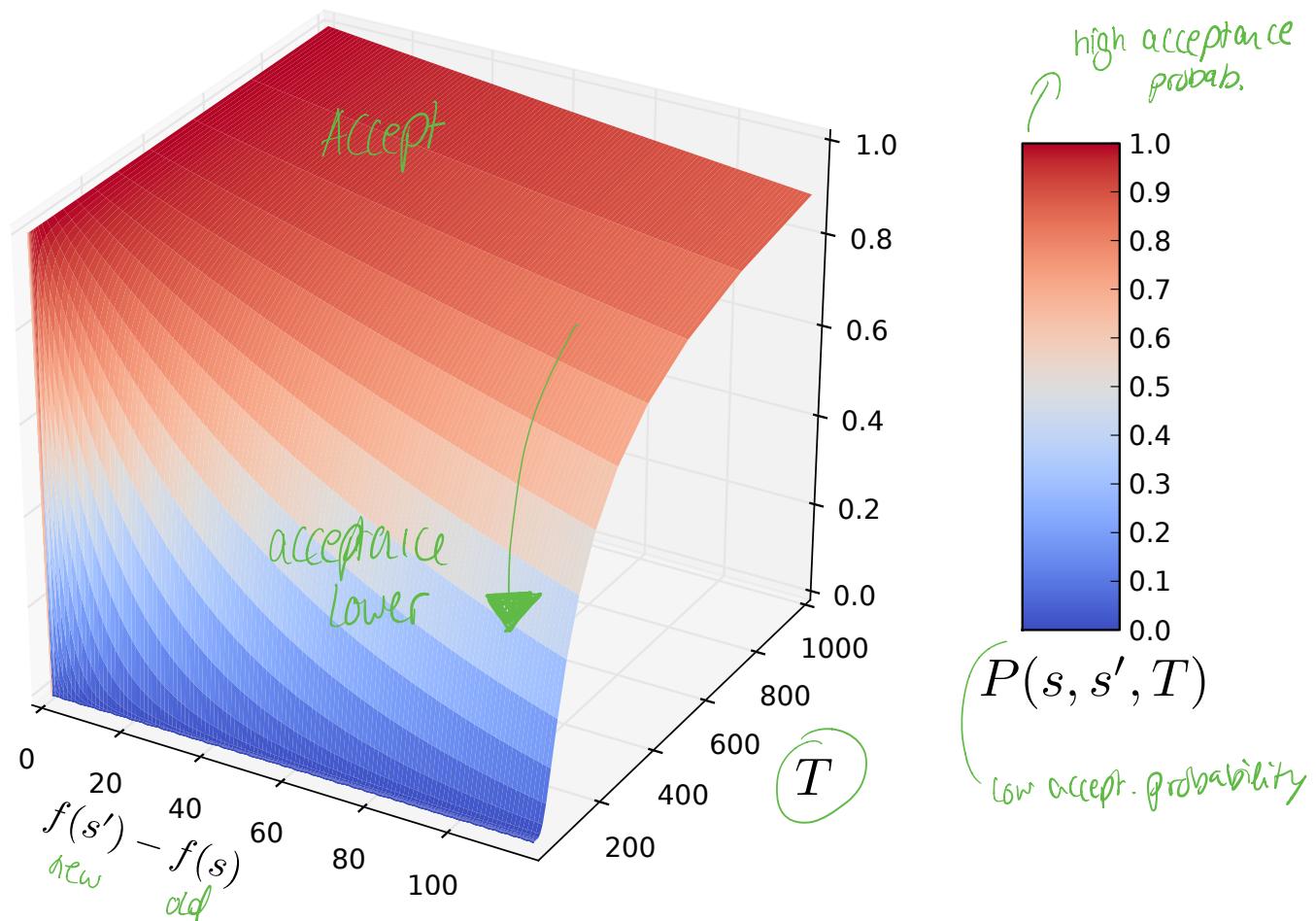
Large temp Probability to accept = 1

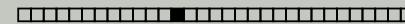
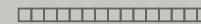
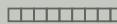
} Accept bad solutions in beginning

$$T \rightarrow 0 \quad \exp(-\infty) = 0$$

Reduce temp Prob to accept = 0

# Metropolis criteria





# SA algorithm

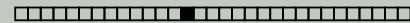
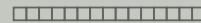
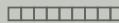
```

1: function SA-MIN( . . . ,  $\alpha$ ,  $T^0$ ,  $\epsilon$ )
2:    $s \leftarrow StartSolution()$ 
3:    $s^* \leftarrow s$ ,  $T \leftarrow T^0$ 
4:   repeat
5:      $s' \leftarrow$  random neighbor in  $N(s)$ 
6:     if  $f(s') < f(s)$  or  $\exp(-\frac{(f(s')-f(s))}{T}) > U(0, 1)$  then
7:        $s \leftarrow s'$ 
8:       if  $f(s) < f(s^*)$  then
9:          $s^* \leftarrow s$ 
10:     $T \leftarrow \alpha T$ 
11:   until  $T < \epsilon$  or  $Terminate(s)$ 

```

*like # iterations  
or quality of solution*

$T^0$	Start temperature	$T$	Actual temperature
$\alpha$	Cooling coefficient	$\epsilon$	Minimum temperature

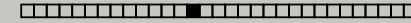
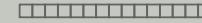
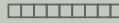


# When does SA work best?

## Question

See above.

If we have a flat local Min



# When does SA work best?

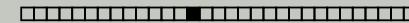
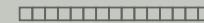
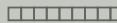
## Question

See above.

- 1** “Flat” local minima are important according to Eglese, R.W. (1990)<sup>1</sup>
- 2** Few plateaus

---

<sup>1</sup>R.W. Eglese (1990). Simulated annealing: A tool for operational research. European Journal of Operational Research 46, 271–281

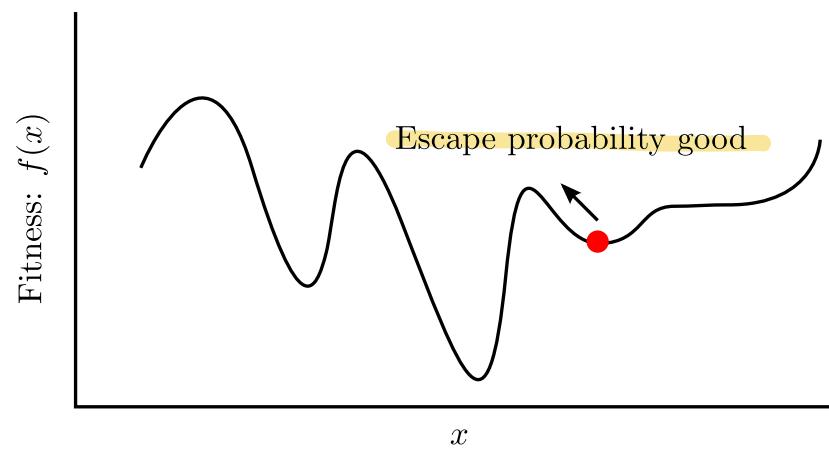


# When does SA work best?

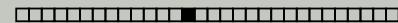
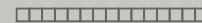
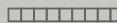
## Question

See above.

- 1 “Flat” local minima are important according to Eglese, R.W. (1990)<sup>1</sup>
- 2 Few plateaus



<sup>1</sup>R.W. Eglese (1990). Simulated annealing: A tool for operational research. European Journal of Operational Research 46, 271–281

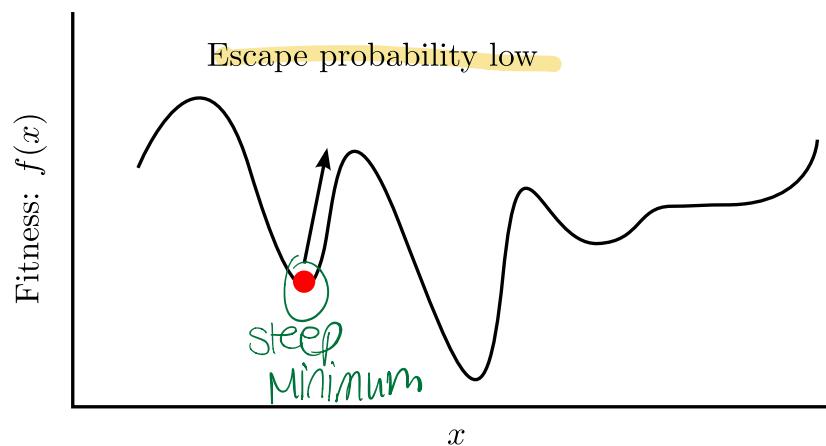


# When does SA work best?

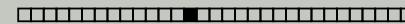
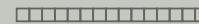
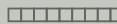
## Question

See above.

- 1 “Flat” local minima are important according to Eglese, R.W. (1990)<sup>1</sup>
- 2 Few plateaus



<sup>1</sup>R.W. Eglese (1990). Simulated annealing: A tool for operational research. European Journal of Operational Research 46, 271–281

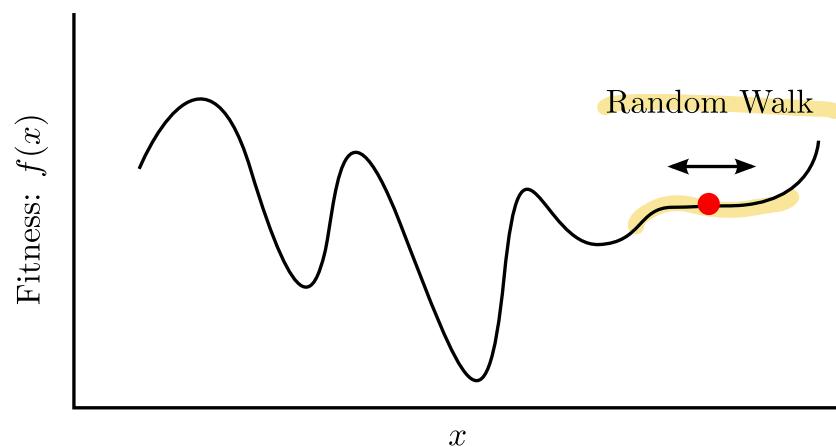


# When does SA work best?

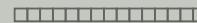
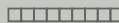
## Question

See above.

- 1 “Flat” local minima are important according to Eglese, R.W. (1990)<sup>1</sup>
- 2 Few plateaus



<sup>1</sup>R.W. Eglese (1990). Simulated annealing: A tool for operational research. European Journal of Operational Research 46, 271–281



# Parameter

## ■ Cooling function

- Generally exponential with  $\alpha \geq 0.8$
- Step and quadratic functions are also possible

## ■ Acceptance criteria

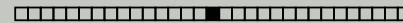
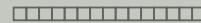
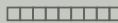
- Metropolis
- Other distributions are also possible, but rarely used.

## ■ Reheating<sup>2</sup> (Temperature is set to $\beta T^0$ at the end of an SA run, with $\beta$ often 0.8)

## ■ Otherwise, the usual parameters for local search

---

<sup>2</sup>J. Taheri, and A. Y. Zomaya (2007). "A simulated annealing approach for mobile location management." Computer Communications 30.4: 714-730.

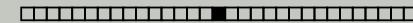
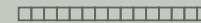
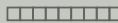


# Properties

## Question



Does SA finds a global optimum **when there is no time limit?**



# Properties

## Question

Does SA finds a global optimum when there is no time limit?

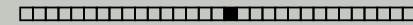
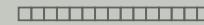
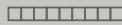
YES!

(With a suitable neighborhood relation)

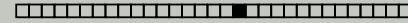
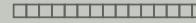
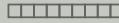
For the proof see: Aarts, et. al (2005), Corollary 7.2.<sup>3</sup>

---

<sup>3</sup>E. Aarts, J. Korst, and W. Michiels (2005). "Simulated annealing." Search methodologies. Springer US. 187-210. → Proof

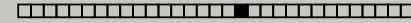
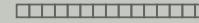
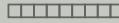


# Tabu search



# Memory vs. memoryless Search

- IL and SA are *memoryless* search methods
- This means that the decisions are based on the current solution and its neighborhood.

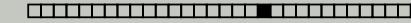
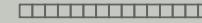
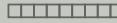


# Memory vs. memoryless Search

- IL and SA are *memoryless* search methods
- This means that the decisions are based on the current solution and its neighborhood.

## TS: core idea

Tabu Search remembers all solutions using a *tabu list* and avoids visiting these solutions again. The goal is to avoid cycles.

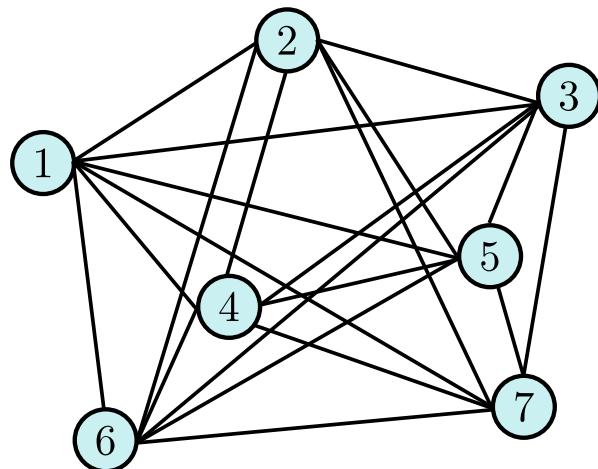


# TSP Example

## Traveling salesman problem

Given: a symmetric TSP problem on the following graph.

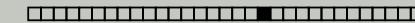
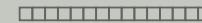
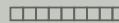
Objective: Find the minimum cost path traversing every node exactly once.



Graph

	1	2	3	4	5	6	7
1	0	2.2	5.0	2.8	4.1	5.0	8.5
2	2.2	0	3.0	3.0	2.8	6.0	8.0
3	5.0	3.0	0	4.2	2.2	7.2	7.0
4	2.8	3.0	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5.0	5.4
6	5.0	6.0	7.2	3.1	5.0	0	5.1
7	8.5	8.0	7.0	5.7	5.4	5.1	0

Edge costs

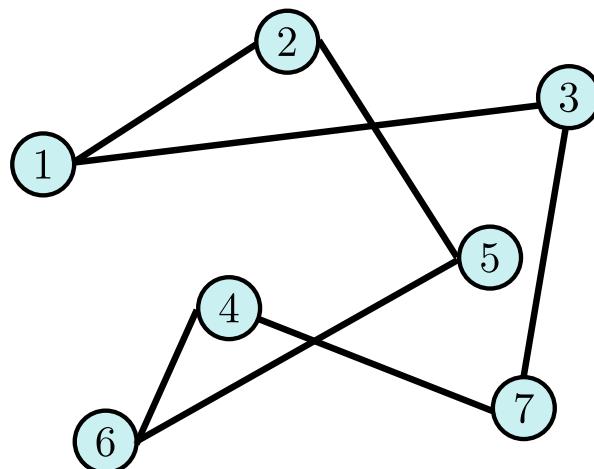


# TSP Example

## Traveling salesman problem

Given: a symmetric TSP problem on the following graph.

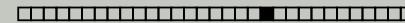
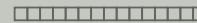
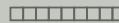
Objective: Find the minimum cost path traversing every node exactly once.



	1	2	3	4	5	6	7
1	0	2.2	5.0	2.8	4.1	5.0	8.5
2	2.2	0	3.0	3.0	2.8	6.0	8.0
3	5.0	3.0	0	4.2	2.2	7.2	7.0
4	2.8	3.0	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5.0	5.4
6	5.0	6.0	7.2	3.1	5.0	0	5.1
7	8.5	8.0	7.0	5.7	5.4	5.1	0

A solution (Costs: 30.8)

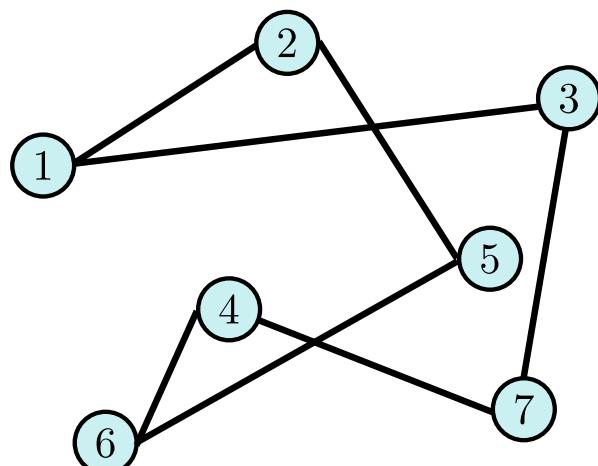
Edge costs



# TSP Example

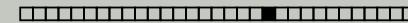
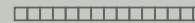
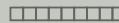
## 2-Opt Neighborhood

We examine all pairs of edges and calculate the new cost if we swap the two edges.



	1	2	3	4	5	6	7
1	0	2.2	5.0	2.8	4.1	5.0	8.5
2	2.2	0	3.0	3.0	2.8	6.0	8.0
3	5.0	3.0	0	4.2	2.2	7.2	7.0
4	2.8	3.0	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5.0	5.4
6	5.0	6.0	7.2	3.1	5.0	0	5.1
7	8.5	8.0	7.0	5.7	5.4	5.1	0

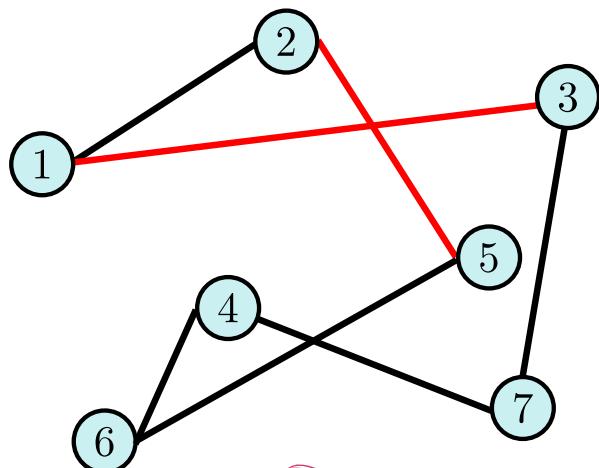
■ Costs: 30.8



# TSP Example

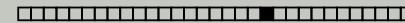
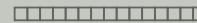
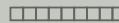
## 2-Opt Neighborhood

We examine all pairs of edges and calculate the new cost if we swap the two edges.



■ Costs: 30.8

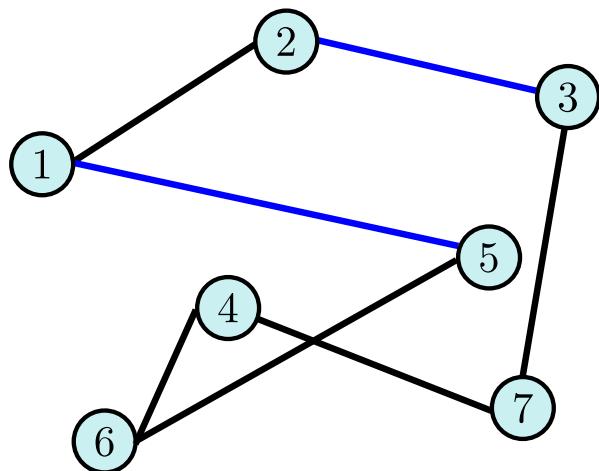
	1	2	3	4	5	6	7
1	0	2.2	5.0	2.8	4.1	5.0	8.5
2	2.2	0	3.0	3.0	2.8	6.0	8.0
3	5.0	3.0	0	4.2	2.2	7.2	7.0
4	2.8	3.0	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5.0	5.4
6	5.0	6.0	7.2	3.1	5.0	0	5.1
7	8.5	8.0	7.0	5.7	5.4	5.1	0



# TSP Example

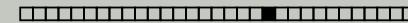
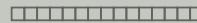
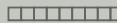
## 2-Opt Neighborhood

We examine all pairs of edges and calculate the new cost if we swap the two edges.



	1	2	3	4	5	6	7
1	0	2.2	5.0	2.8	4.1	5.0	8.5
2	2.2	0	3.0	3.0	2.8	6.0	8.0
3	5.0	3.0	0	4.2	2.2	7.2	7.0
4	2.8	3.0	4.2	0	2.2	3.1	5.7
5	4.1	2.8	2.2	2.2	0	5.0	5.4
6	5.0	6.0	7.2	3.1	5.0	0	5.1
7	8.5	8.0	7.0	5.7	5.4	5.1	0

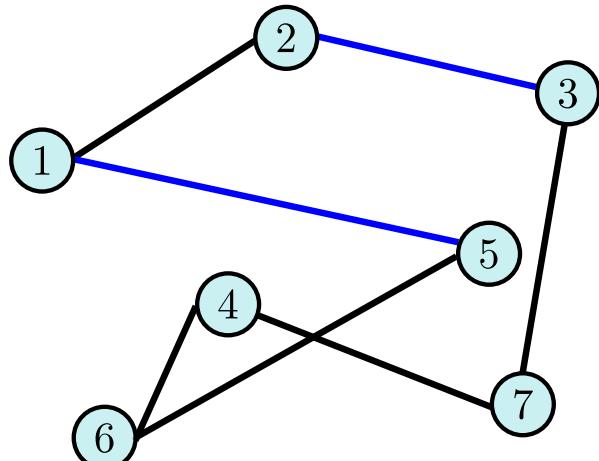
■ Costs: 30.1  
*better*



# TSP Example

## 2-Opt Neighborhood

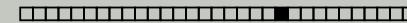
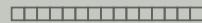
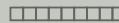
We examine all pairs of edges and calculate the new cost if we swap the two edges.



■ Costs: 30.1

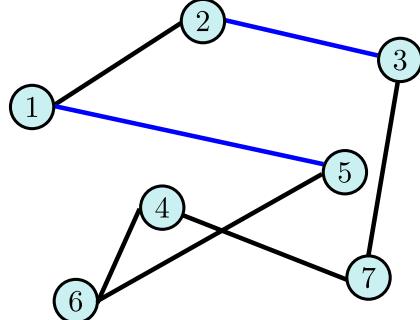
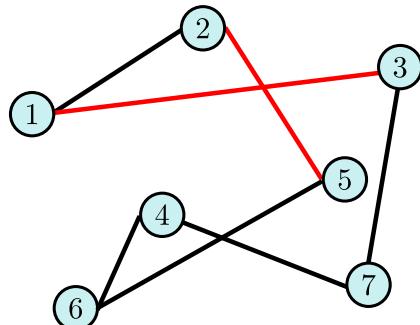
## Tabu list

We now want to ~~forbid the previous solution~~. But how?



# Tabu list options

- Forbid complete solution

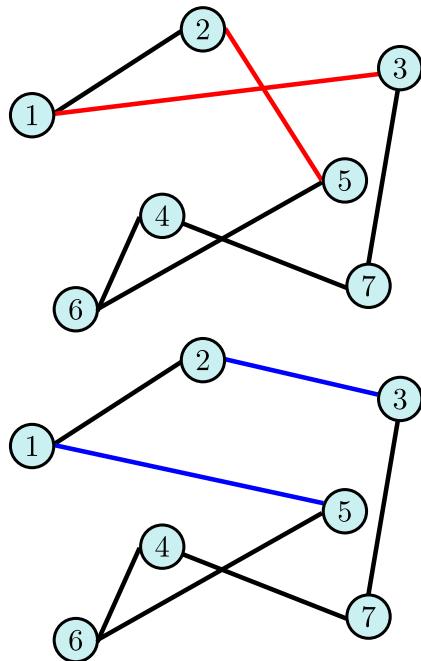


Tabu list:

- 1 2 5 6 4 7 3



# Tabu list options



Tabu list:

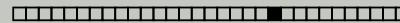
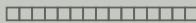
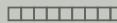
- 1 2 5 6 4 7 3

## Question

What are the strengths and weaknesses of keeping a ~~storing~~ complete solution? in the tabu list

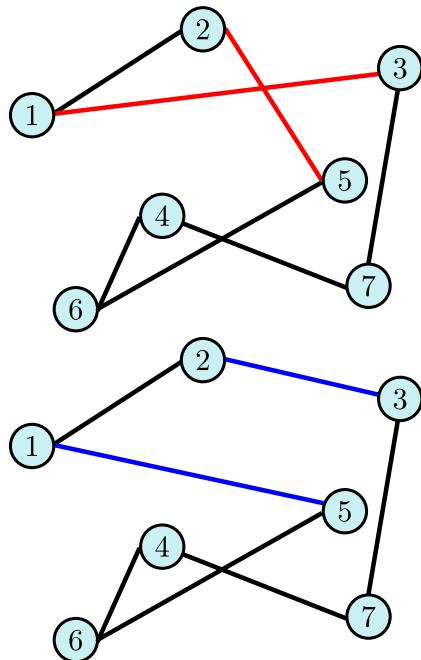
- (-) - Memory & Search  
each time check through whole  
solution  
↳ Is it already in Tabulist?

(+)



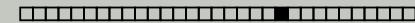
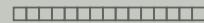
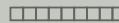
# Tabu list options

- Forbid partial solution

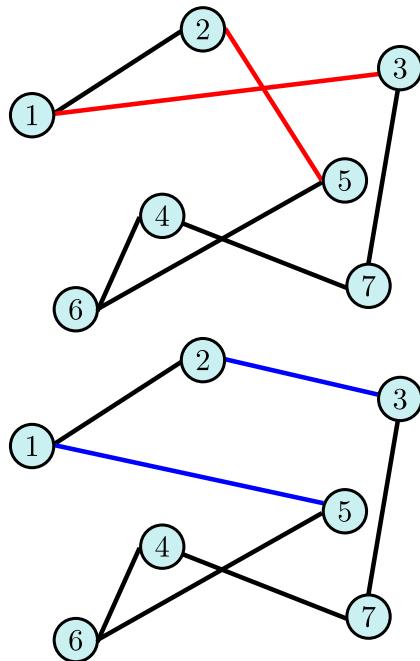


Tabu list:

- 3 1 \* 2 5 \* \* \*
- or: \* \* \* 2 5 \* \* \*
- or: 3 1 \* \* \* \* \*



# Tabu list options

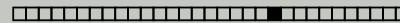
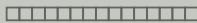
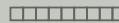


Tabu list:

- 3 1 \* 2 5 \* \* \*
- or: \* \* \* 2 5 \* \* \*
- or: 3 1 \* \* \* \* \*

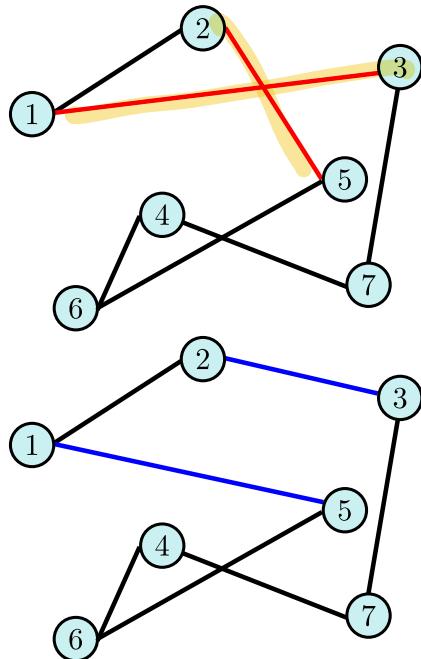
Question

What are the strengths and weaknesses of retaining a partial solution?



# Tabu list options

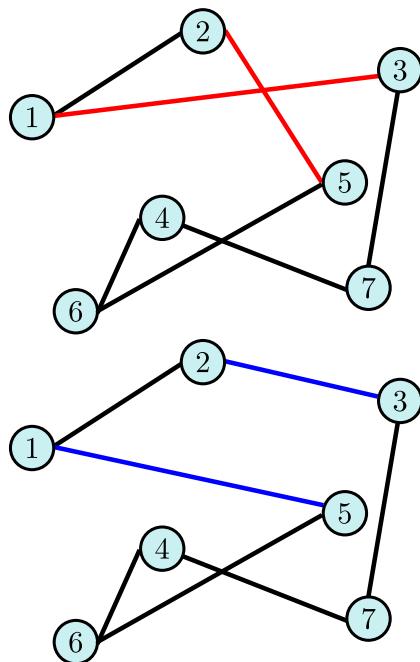
- Ban neighborhood transition



*Most commonly used*

Tabu list:

- $2 \rightarrow 5, 1 \rightarrow 3 \Rightarrow$   
 $2 \rightarrow 3, 1 \rightarrow 5$



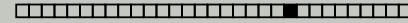
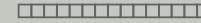
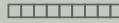
## Tabu list:

- $2 \rightarrow 5, 1 \rightarrow 3 \Rightarrow$   
 $2 \rightarrow 3, 1 \rightarrow 5$

### Question

What are the strengths and weaknesses of storing a neighborhood transition?

- (+) easy search, memory
- (-) depends on implementation  
how to define?



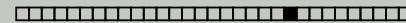
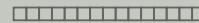
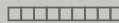
# Tabu list management

Limited memory

- Fixed length list

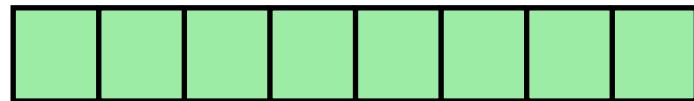


Tabu list with length  $\ell$



# Tabu list management

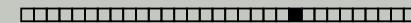
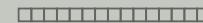
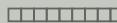
- Fixed length list



Tabu list with length  $\ell$

## Question

How should we manage the tabu list? What happens if the list is full?



# Tabu list management

- Fixed length list



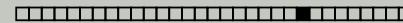
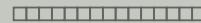
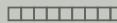
Tabu list with length  $\ell$

## Question

How should we manage the tabu list? What happens if the list is full?

- FIFO strategy: First In First Out *Remove oldest one*
- Or a list item is selected by a heuristic

*Random* ↗ *Greedy: this element is visited most: keep not visited more: delete*

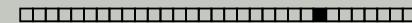
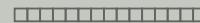
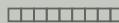


# Tabu List Management

- List with variable length: Reactive Tabu Search<sup>4</sup>
- In summary:
  - Increase list length when previously seen solutions are repeated.
  - Decrease list length when solutions are rarely seen a second time.



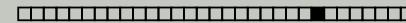
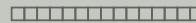
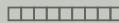
<sup>4</sup>R. Battiti, and G. Tecchiolli (1994). "The reactive tabu search." ORSA journal on computing 6.2: 126-140.



# Aspiration criteria

## Question

What should we do when a successor is forbidden by the tabu list, but has a better objective function value than our current solution?



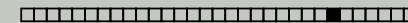
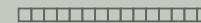
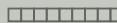
# Aspiration criteria

## Question

What should we do when a successor is forbidden by the tabu list, but has a better objective function value than our current solution?

- Aspiration criteria allows tabu search to accept successors that would otherwise be prohibited
- The criteria uses a heuristic:
  - $f(\text{Solution}) - f(\text{Successor}) \geq \gamma$
  - The successor has a desired structure or property

*(we want diversity)*



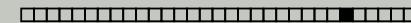
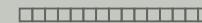
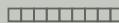
# TS Algorithm

```

1: function TS-MIN( $f, C, N, L, StartSolution, Terminate, \ell$ )
2:    $s \leftarrow StartSolution()$ 
3:    $s^* \leftarrow s, A \leftarrow \emptyset, s^P \leftarrow s$  previous  

in best neighbors forbidden solution
4:    $R \leftarrow \emptyset$  List of possible neighbors not in tabu ▷ Tabu List
5:   repeat
6:      $A \leftarrow \{s' \in N(s) \mid s' \notin R\}$  ▷ Permitted neighbors
7:      $s \leftarrow \operatorname{argmin}_{s' \in A}$ 
8:     if  $f(s) < f(s^*)$  then
9:        $s^* \leftarrow s$ 
10:      if  $|R| \geq \ell$  then Is length of tabulist full?
11:         $R \leftarrow FIFO(R)$  Remove oldest part in tabu list
12:       $R \leftarrow R \cup s^P$  Add solution to tabulist
13:       $s^P \leftarrow s$  ▷ Previous solution
14:    until  $Terminate(s)$  or  $A = \emptyset$  empty list of possible neighbors

```

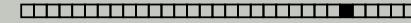
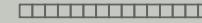
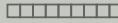


# The black hole

## Question

Can the tabu list cause the neighborhood to be empty?

all elements in  $A$  are in tabu?



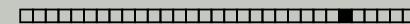
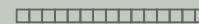
# The black hole

## Question

Can the tabu list cause the neighborhood to be empty?

Unfortunately, yes. This is called a “black hole”.





# The black hole

## Question

Can the tabu list cause the neighborhood to be empty?

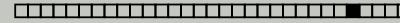
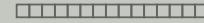
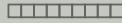
Unfortunately, yes. This is called a “black hole”.



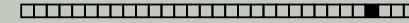
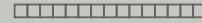
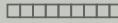
## Question

What should we do if we find ourselves in a black hole?

- Use aspiration criteria : successor with better objective function
- Clear tabu list → Diversification again
- Restart the search

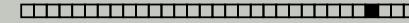
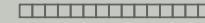
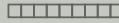


# Iterated local search



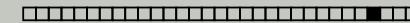
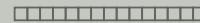
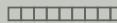
# Restarts

- After an LS (if there is time) ~~repeat the LS again~~. This is called a **“restart”**
- A ~~new starting solution~~ may (but need not) be used
- The ~~neighborhoods~~ can also be ~~changed~~ (e.g., more / less greedy, etc)



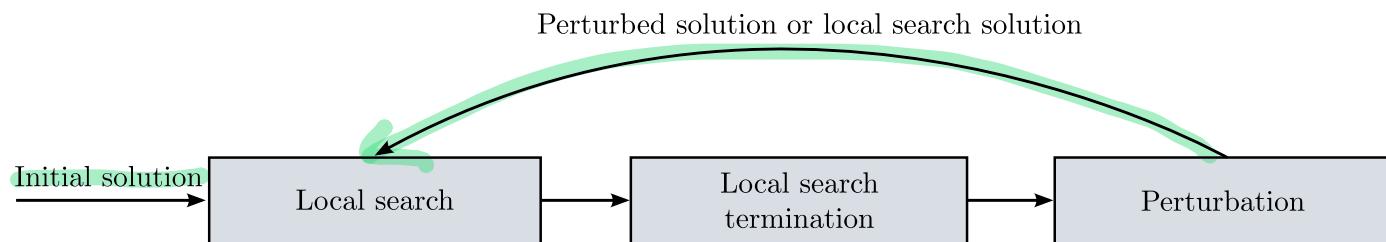
# Restarts

- After an LS (if there is time) repeat the LS again. This is called a “restart”
- A new starting solution may (but need not) be used
- The neighborhoods can also be changed (e.g., more / less greedy, etc)
- A well known restart strategy is *iterated local search*

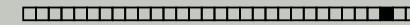
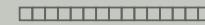
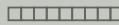


# Restarts

- After an LS (if there is time) repeat the LS again. This is called a “restart”
- A new starting solution may (but need not) be used
- The neighborhoods can also be changed (e.g., more / less greedy, etc)
- A well known restart strategy is *iterated local search*

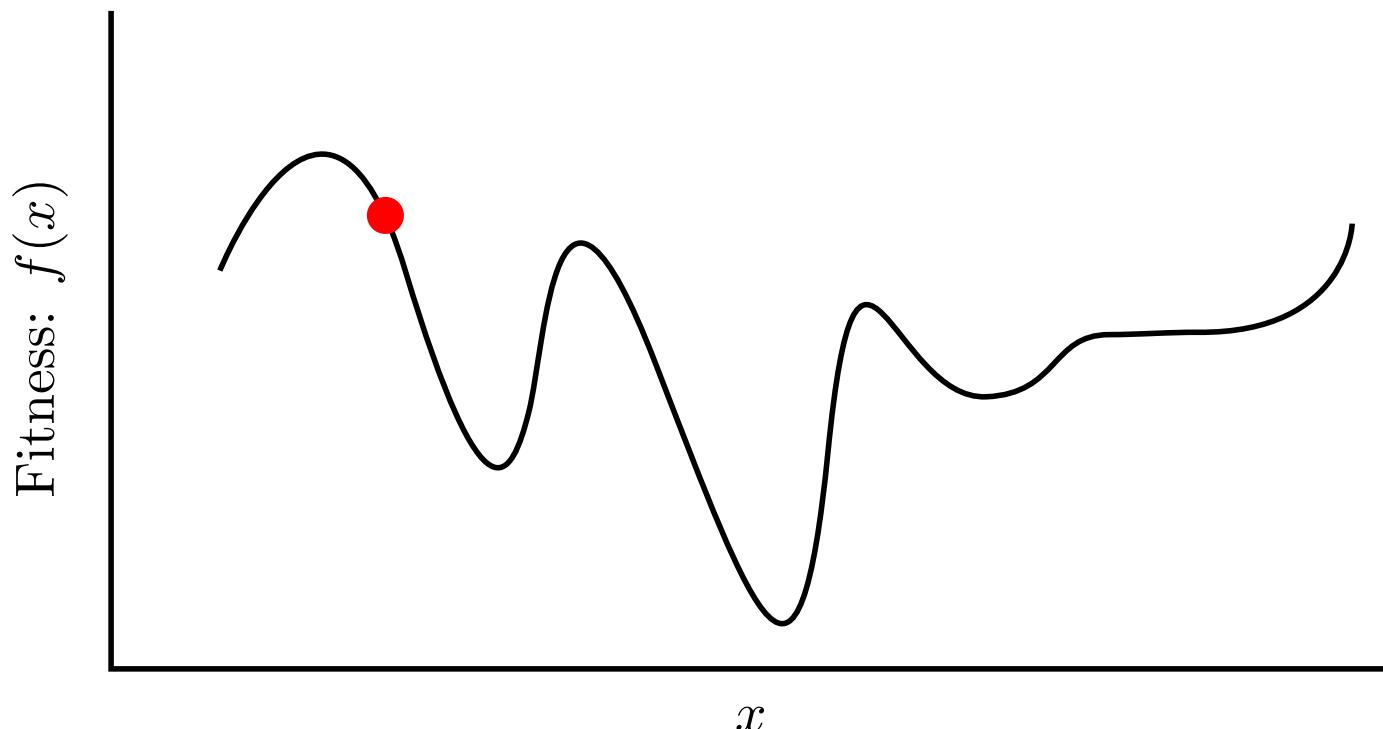


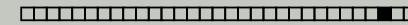
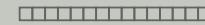
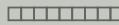
- An acceptance criterion decides whether the perturbed solution or the local search solution is used



# Perturbation

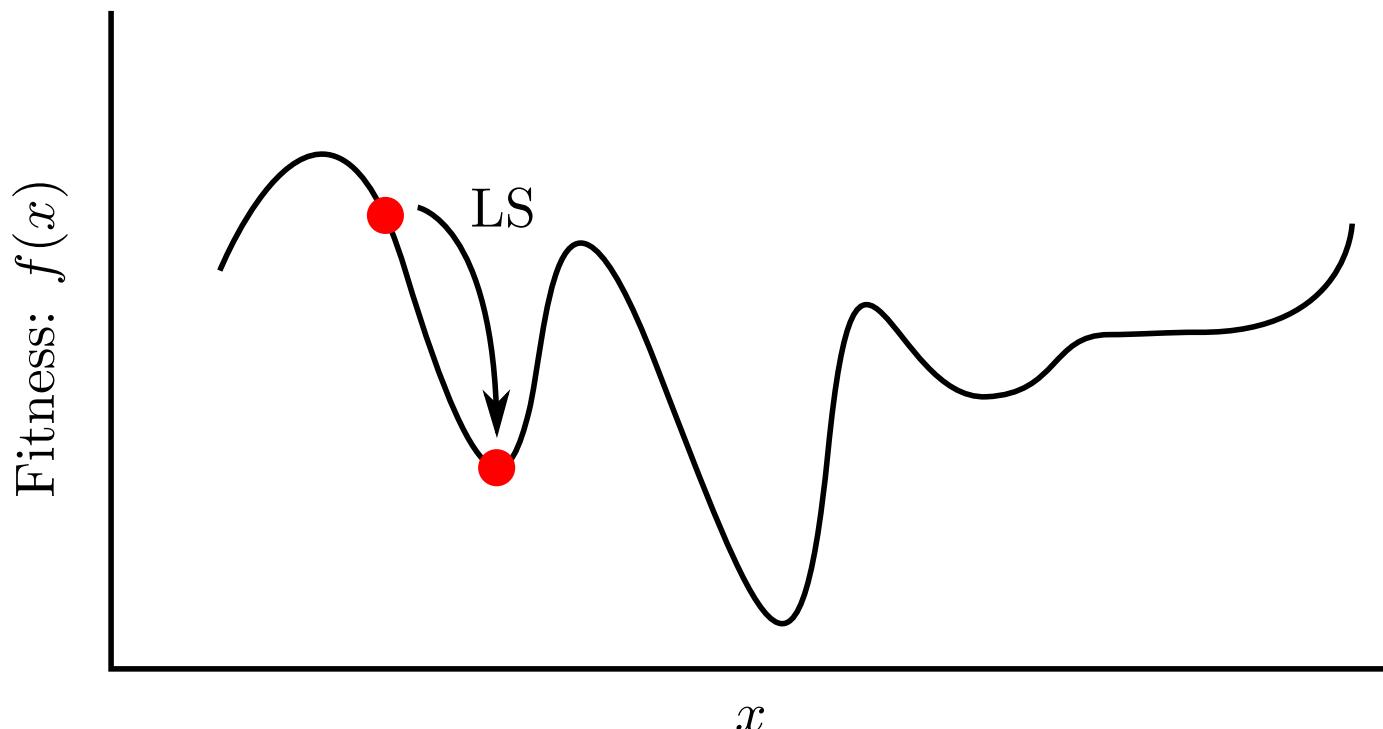
- The goal of the perturbation is push the current solution from one local optimum into the attraction basin of another optimum.

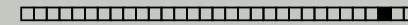
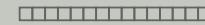
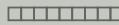




# Perturbation

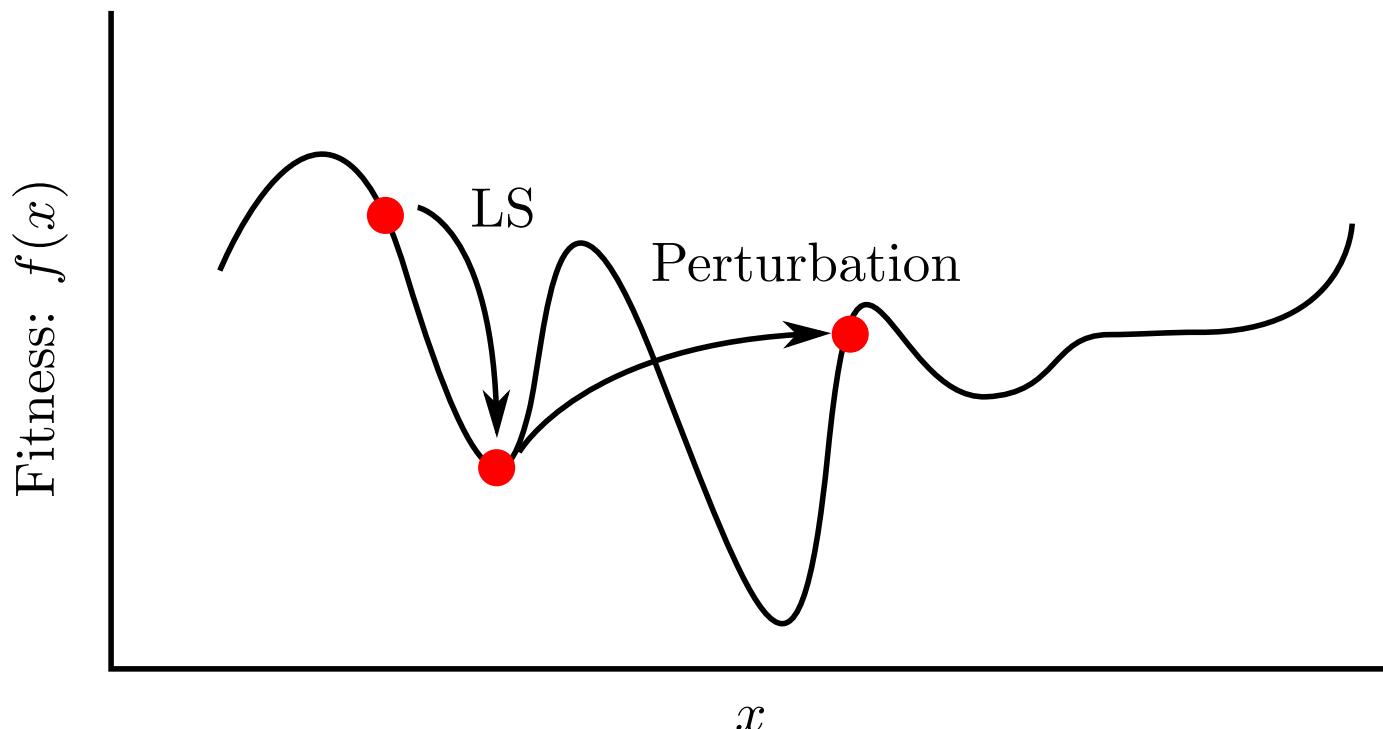
- The goal of the perturbation is push the current solution from one local optimum into the attraction basin of another optimum.

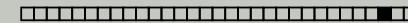
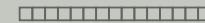
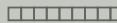




# Perturbation

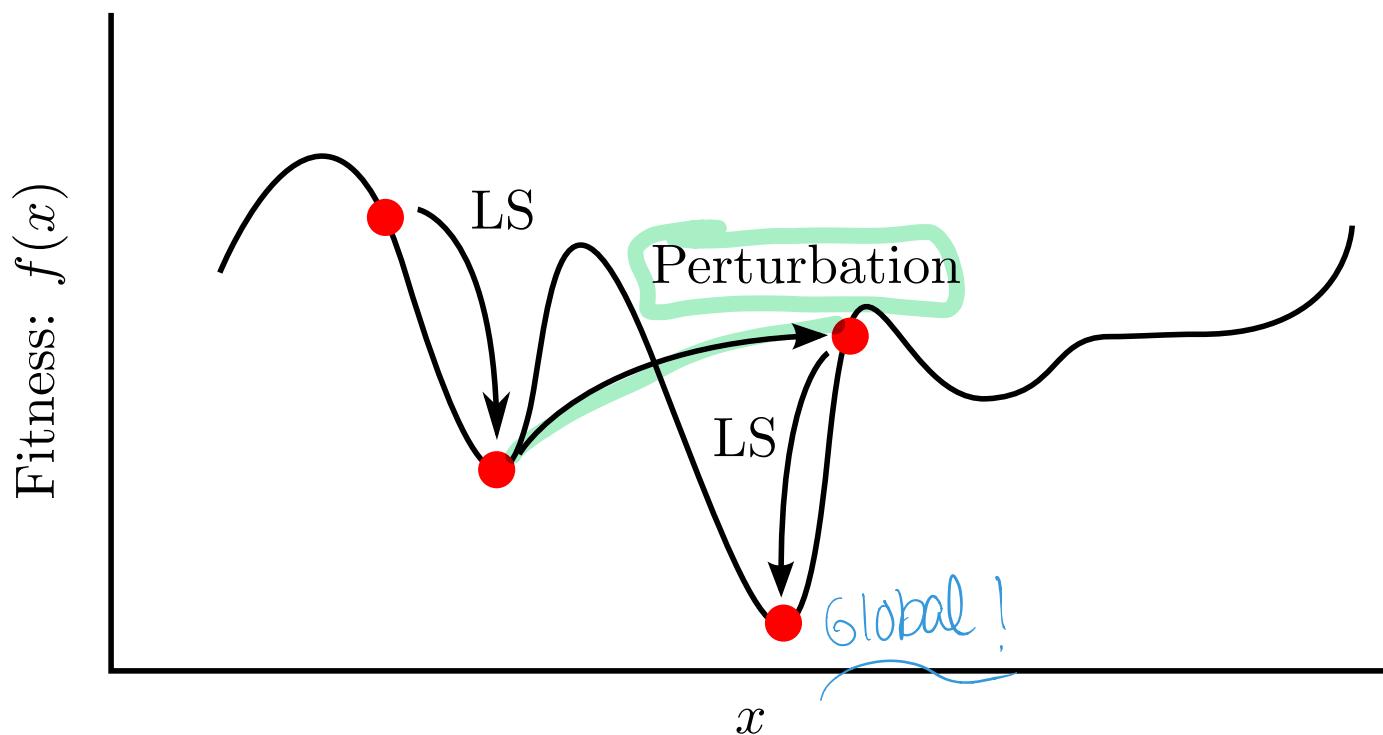
- The goal of the perturbation is push the current solution from one local optimum into the attraction basin of another optimum.

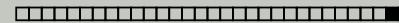
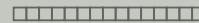
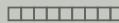




# Perturbation

- The goal of the perturbation is push the current solution from one local optimum into the attraction basin of another optimum.

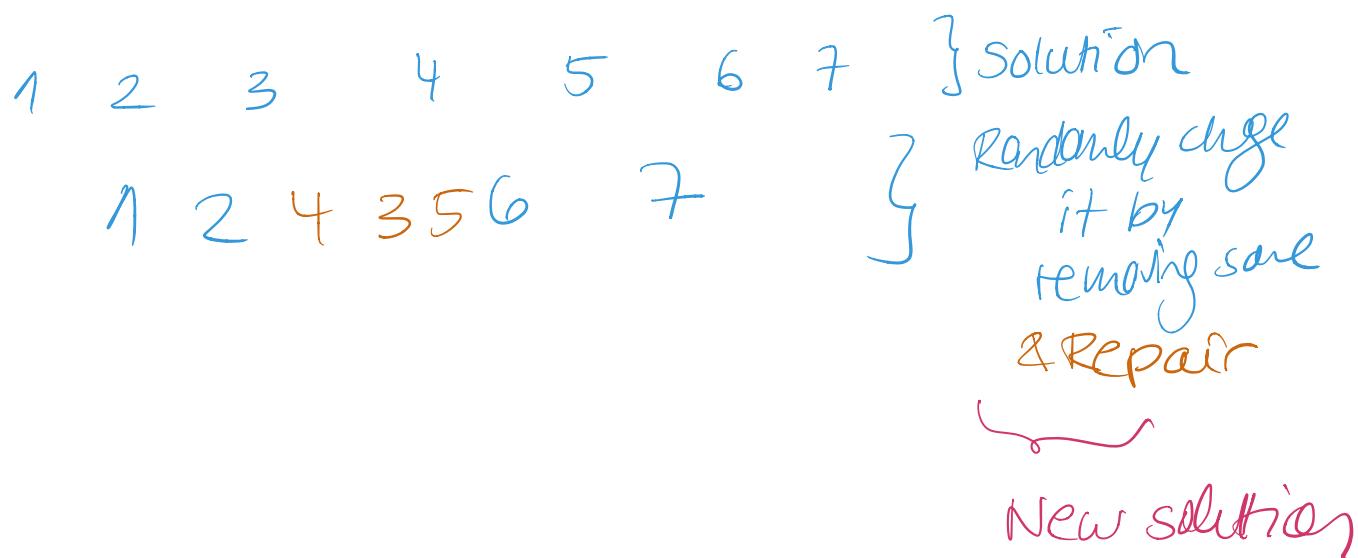


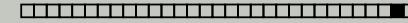
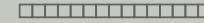
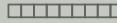


# Perturbation

## Question

One possible perturbation heuristic is to randomly change a part of the solution. Is this a good or bad perturbation heuristic?





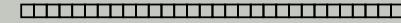
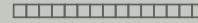
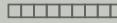
# Perturbation

## Question

One possible perturbation heuristic is to randomly change a part of the solution. Is this a good or bad perturbation heuristic?

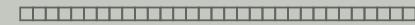
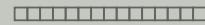
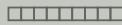
- Goals of a perturbation:

- Leave the current basin of attraction / enter a different basin
- Jump to a near-by basin of attraction
- It should be hard for local search to undo the perturbation!

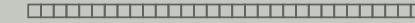
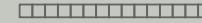
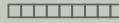


# Terminology

Problem	Defines a general type of task to be solved.
Instance	A specific setting of problem parameters to be solved.
Decision variables	Open questions about the solution requiring answers.
Constraints	Restrictions on the allowed settings of decision variables.
Objective function	Mathematical expression over the decision variables defining the value (cost/profit) of a particular set of decisions. (a.k.a. Fitness function)
Optimization problem	Problem in which the goal is to find a satisfying assignment to variables with a minimal (or maximal) objective function value.

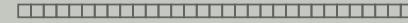
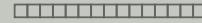
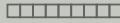


# Literature



# Literatur

- H. Hoos and T. Stützle (2005). Stochastic Local Search: Foundations and Applications. Elsevier.
- M. Gendreau, and J.-Y. Potvin, eds (2010). Handbook of Metaheuristics. Vol. 146. Springer.



# Thank you for your attention!

Leuphana University of Lüneburg  
Business information systems, in particular Operations Research  
Prof. Dr. Lin Xie  
Universitätsallee 1  
Building 4, Room 314  
21335 Lüneburg  
Phone +49 4131 677 2305  
Fax +49 4131 677 1749  
[xie@leuphana.de](mailto:xie@leuphana.de)

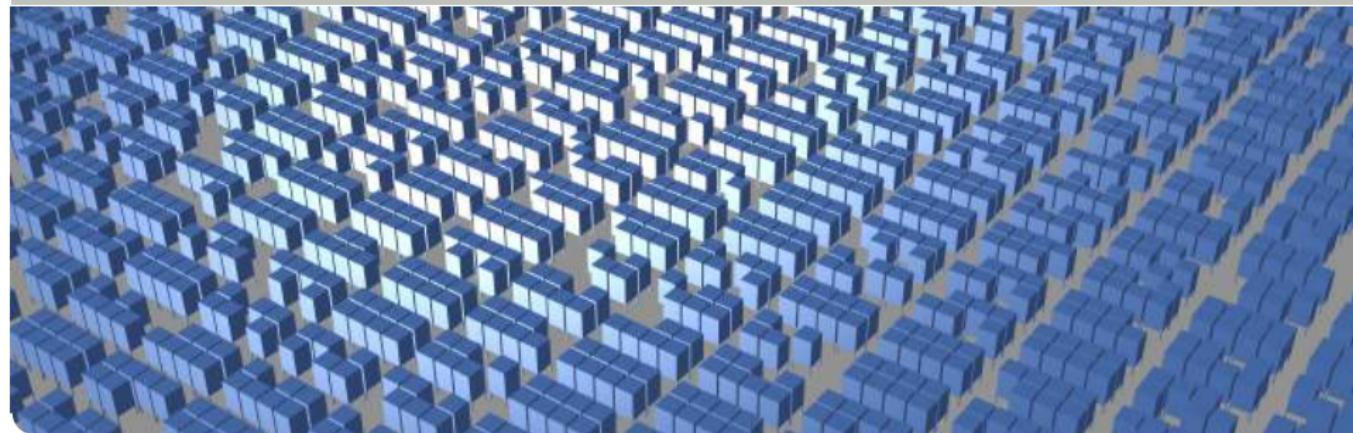


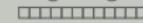
# Analysing Network using OR-Methods

Part 9 – VNS and LNS

Lin Xie | 22.06.2021

PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH





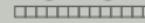
# Agenda

## 1 Variable Neighborhood Search

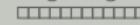
- 1 Nested Neighborhoods
- 2 Variable Neighborhood Descent
- 3 Reduced VNS

## 2 Large Neighborhood Search

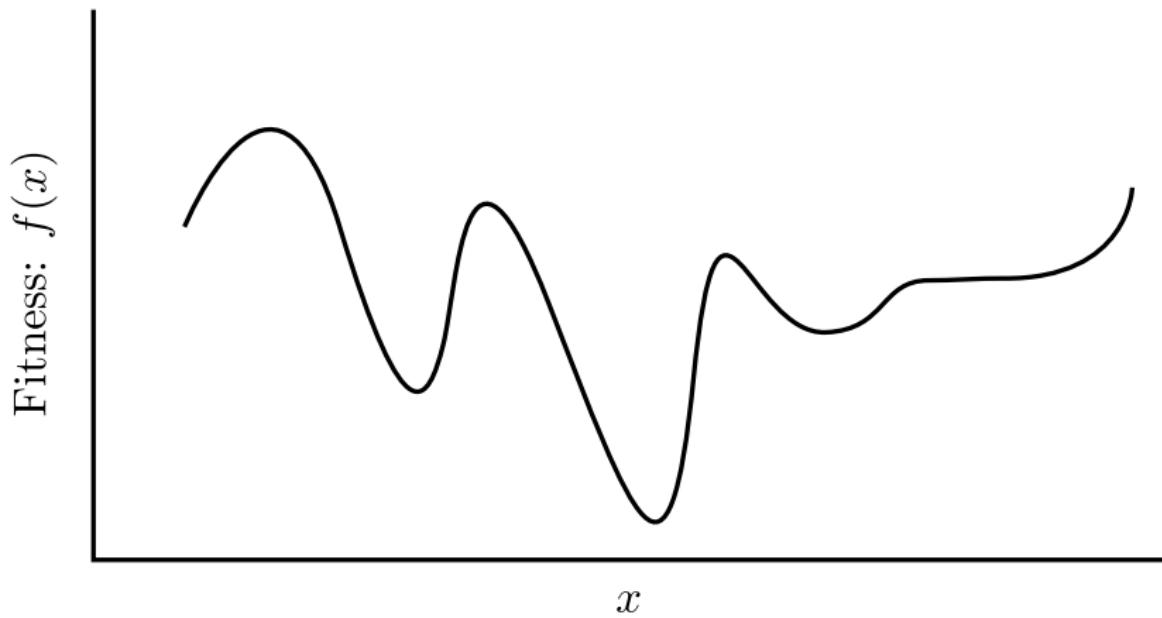
- 1 Adaptive LNS
- 2 Job-Shop Example

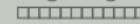


# Variable neighborhood search

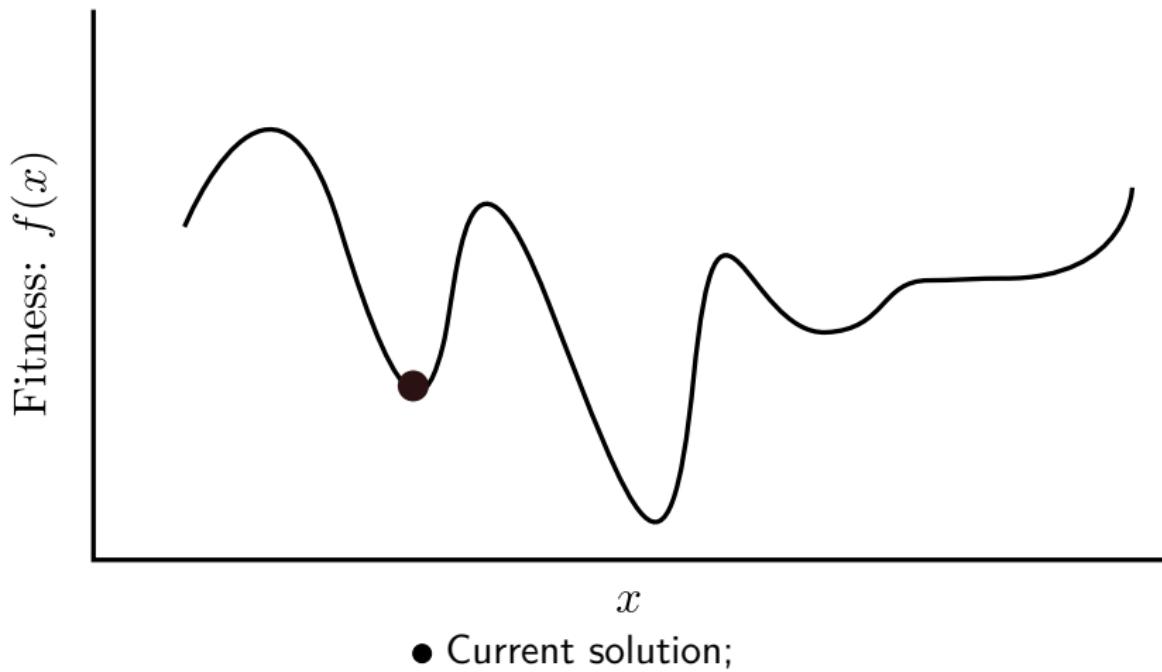


# Multiple neighborhoods

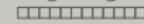




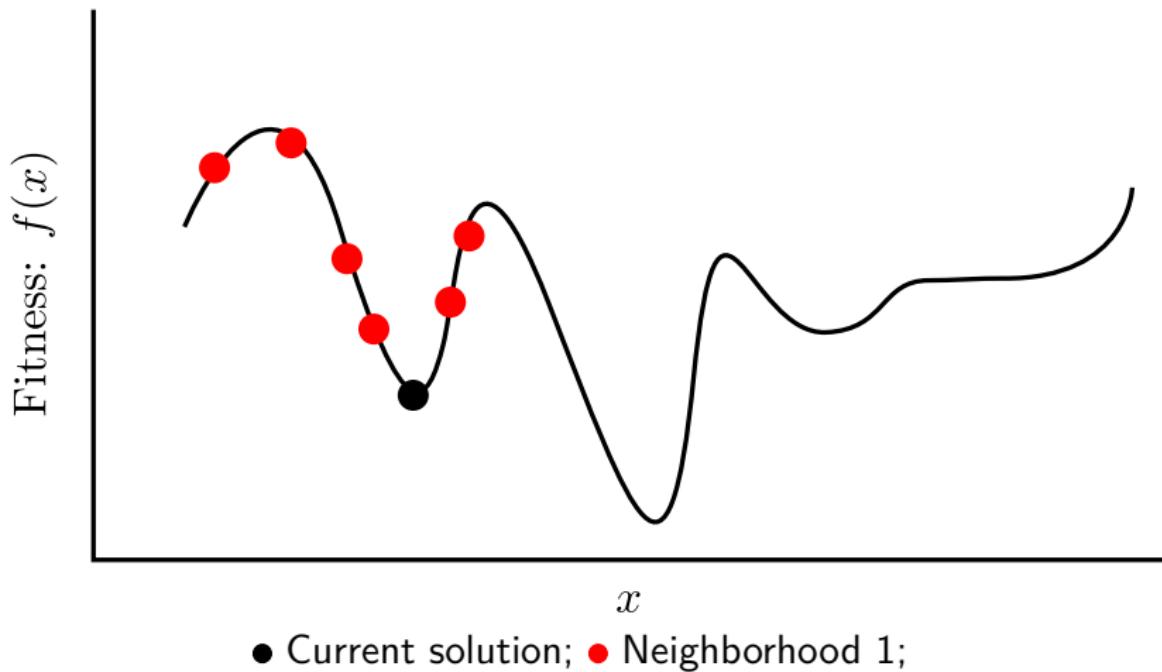
# Multiple neighborhoods

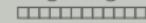
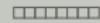


● Current solution;

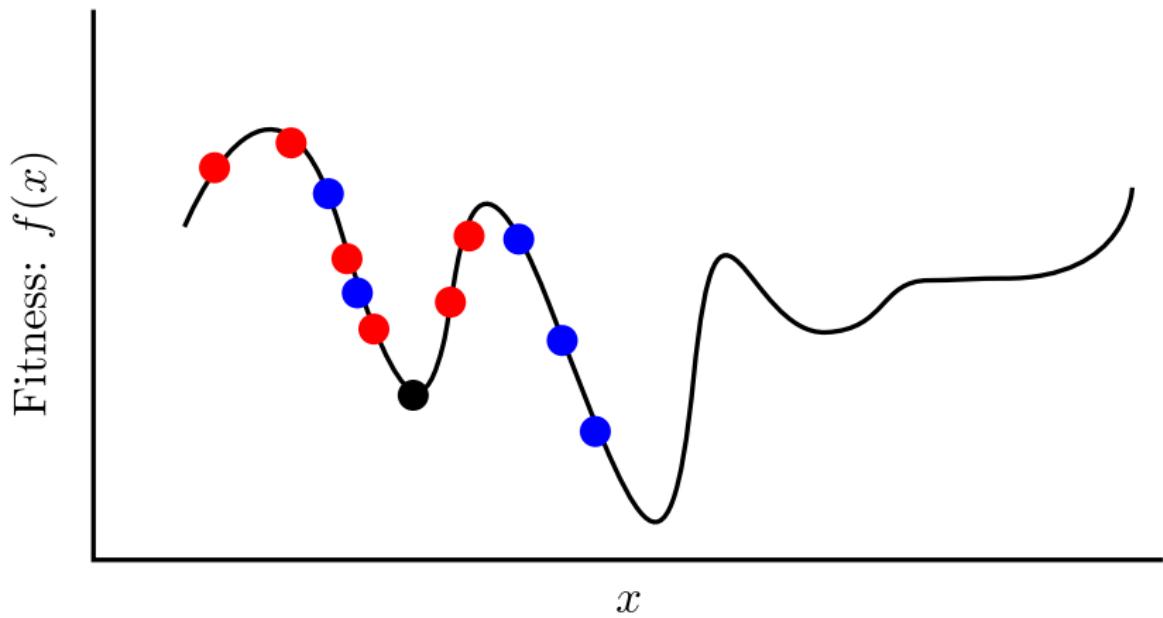


# Multiple neighborhoods



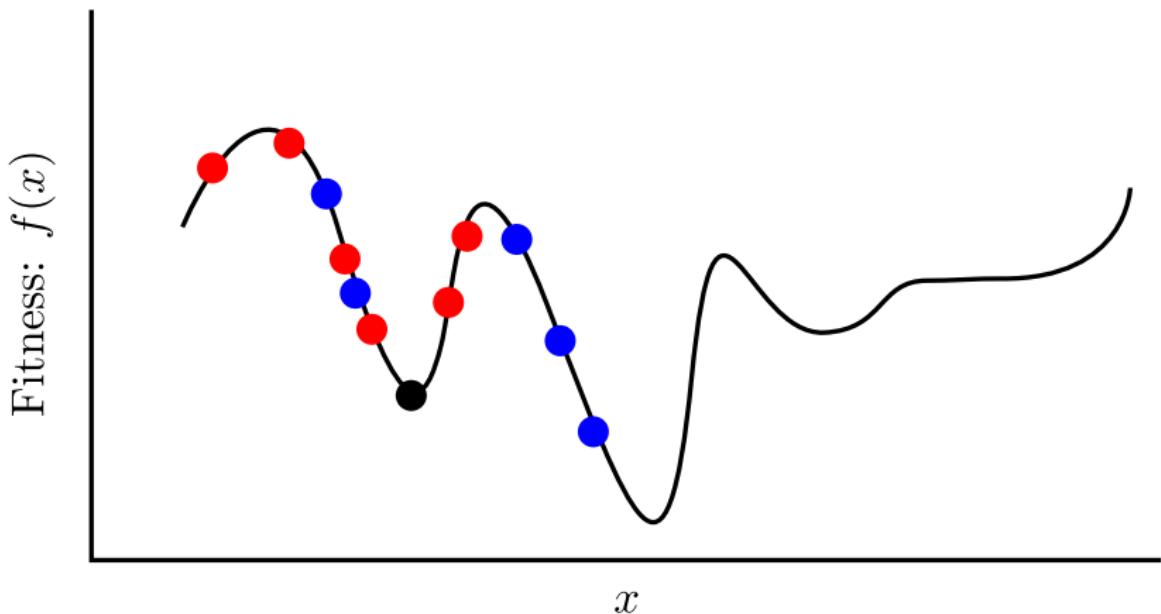


# Multiple neighborhoods



- Current solution; ● Neighborhood 1; ● Neighborhood 2;

# Multiple neighborhoods

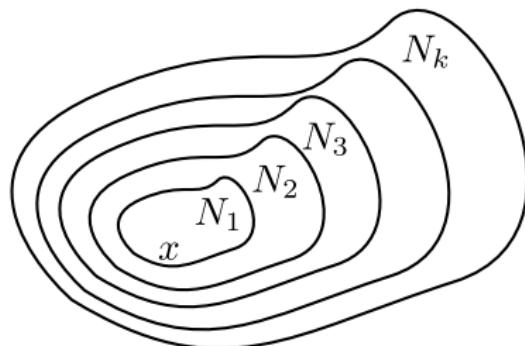


- Current solution; ● Neighborhood 1; ● Neighborhood 2;

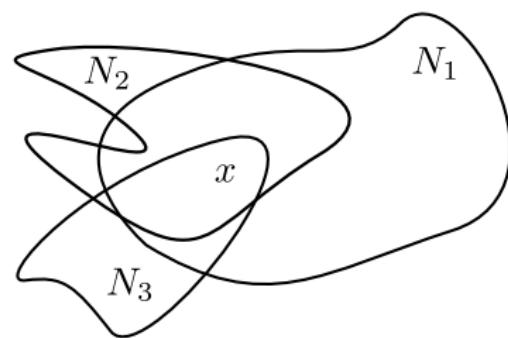
While one neighborhood may lead to a local optimum, other neighborhoods may lead elsewhere.

# Variable neighborhood search variants

- Two possibilities for the use of multiple neighborhoods: VDNS und VNS



Variable neighborhood  
descent (nested)



Variable neighborhood search

Figures adapted from Pisinger and Røpke (2010) Figure 13.2

## TSP Example

# TSP example: $k$ -opt neighborhood

## Traveling salesman problem

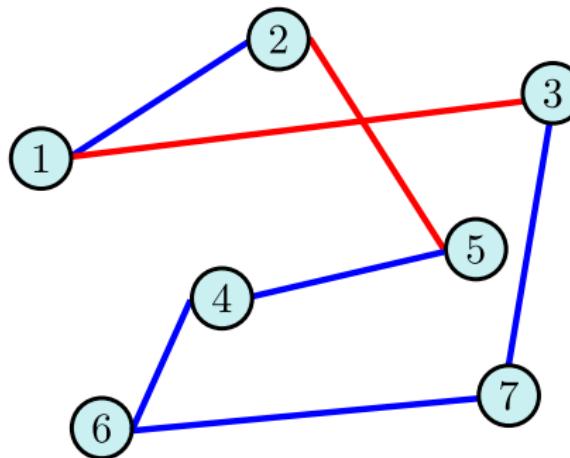
The goal is to visit each city (node in  $V$  of a graph  $G = (V, E)$ ) exactly once at a cost-minimal route and return to the original city.

# TSP example: $k$ -opt neighborhood

## Traveling salesman problem

The goal is to visit each city (node in  $V$  of a graph  $G = (V, E)$ ) exactly once at a cost-minimal route and return to the original city.

2-opt neighborhood: Tries swapping all pairs of edges in a solution.

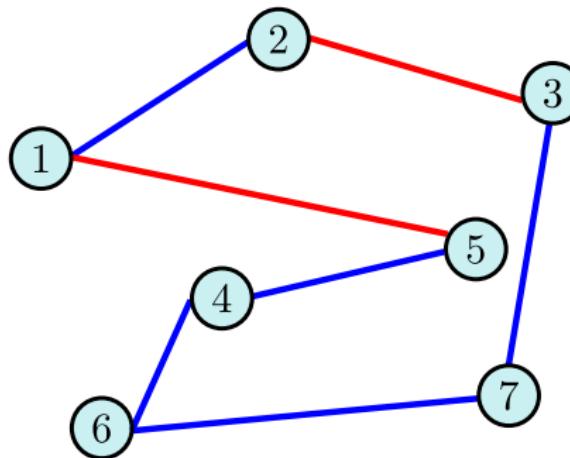


# TSP example: $k$ -opt neighborhood

## Traveling salesman problem

The goal is to visit each city (node in  $V$  of a graph  $G = (V, E)$ ) exactly once at a cost-minimal route and return to the original city.

2-opt neighborhood: Tries swapping all pairs of edges in a solution.

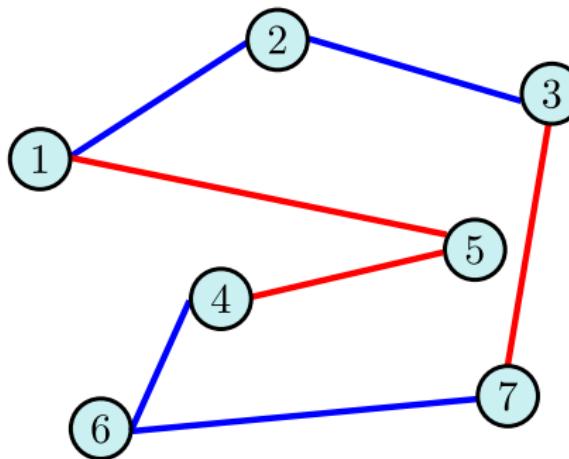


# TSP $k$ -opt neighborhood

The 2-opt neighborhood can be generalized into the  $k$ -opt neighborhood.

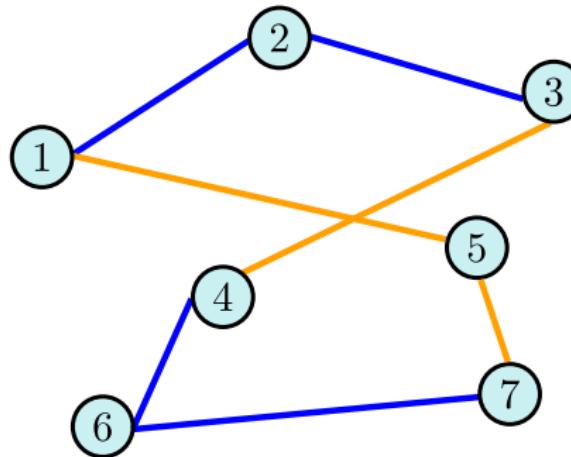
# TSP $k$ -opt neighborhood

The 2-opt neighborhood can be generalized into the  $k$ -opt neighborhood.



# TSP $k$ -opt neighborhood

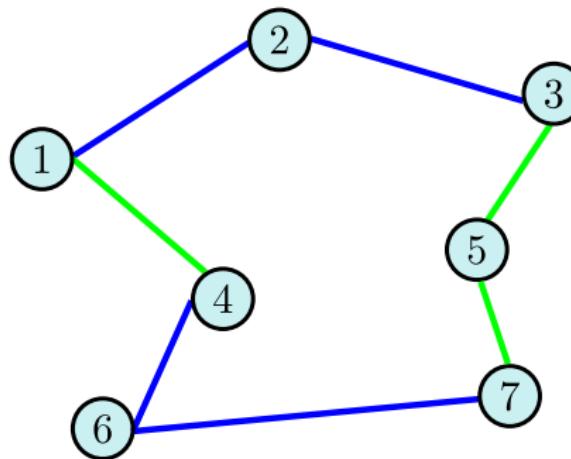
The 2-opt neighborhood can be generalized into the  $k$ -opt neighborhood.



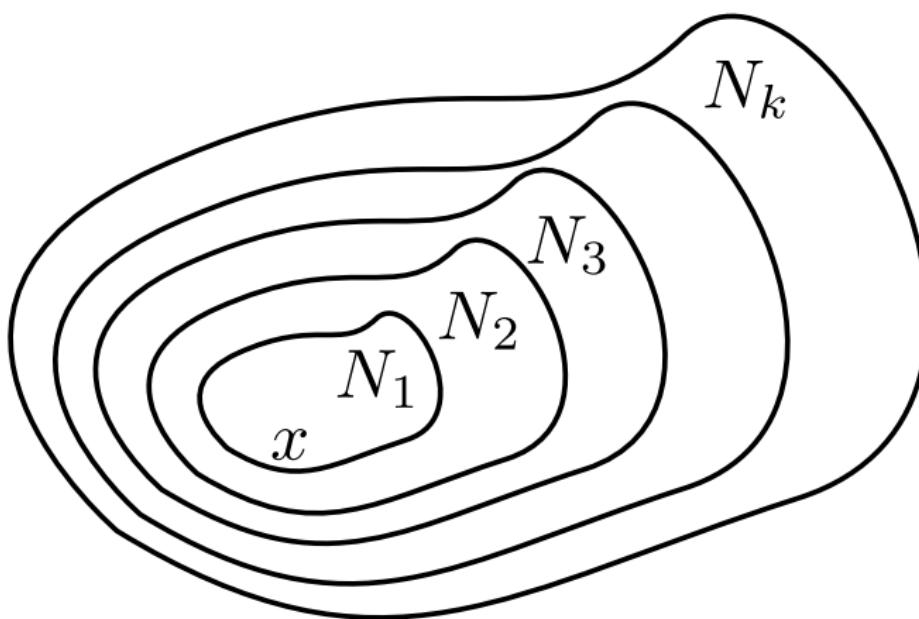


# TSP $k$ -opt neighborhood

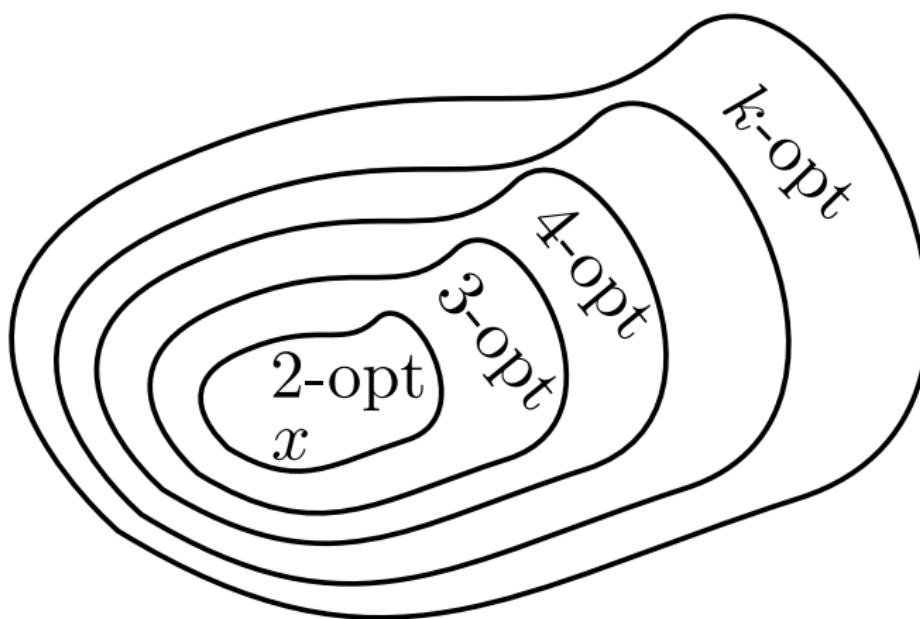
The 2-opt neighborhood can be generalized into the  $k$ -opt neighborhood.

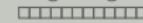


# TSP $k$ -opt neighborhood



# TSP $k$ -opt neighborhood





# Variable neighborhood descent (VND)

## Definition

Variable neighborhood descent uses nested, parameterized neighborhoods and varies the size of the neighborhoods it uses to avoid local optima.

$f$  object.function

$k$  neighborhood

$k_{max}$

# VND pseudocode

```

1: function VND-MIN( $f$ ,  $N_k$ ,  $k^{max}$ ,  $s^I = StartSolution$ )
2:    $s \leftarrow s^I$ ,  $s^* \leftarrow s$ 
3:    $k \leftarrow 1$  best solution so far
4:   repeat neighbors with min obj. funct.
5:      $s' \leftarrow \operatorname{argmin}_{s'' \in N_k(s)} \{f(s'')\}$ 
6:     if  $f(s') < f(s)$  then
7:        $s \leftarrow s'$  Better than solution? then add to s
8:        $k \leftarrow 1$  Best Replace
9:       if  $f(s) < f(s^*)$  then  $s^* \leftarrow s$  end if
10:    else
11:       $k \leftarrow k + 1$  enlarge neighborhood if haven't found best yet?
12:    until  $k = k^{max}$ 
13: return  $s^*$  until checked all neighbors
best solution

```



# VND vs. reduced VNS

VND

- VND searches the entire neighborhood.
- Slow for large neighborhoods

# VND vs. reduced VNS

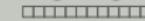
## VND

- VND searches the entire neighborhood.
- Slow for large neighborhoods

## RVNS

- Reduced VNS selects a random neighbor, rather than the seeking out the best.

Instead of choosing best  
→ chose random



# Reduced VNS

```

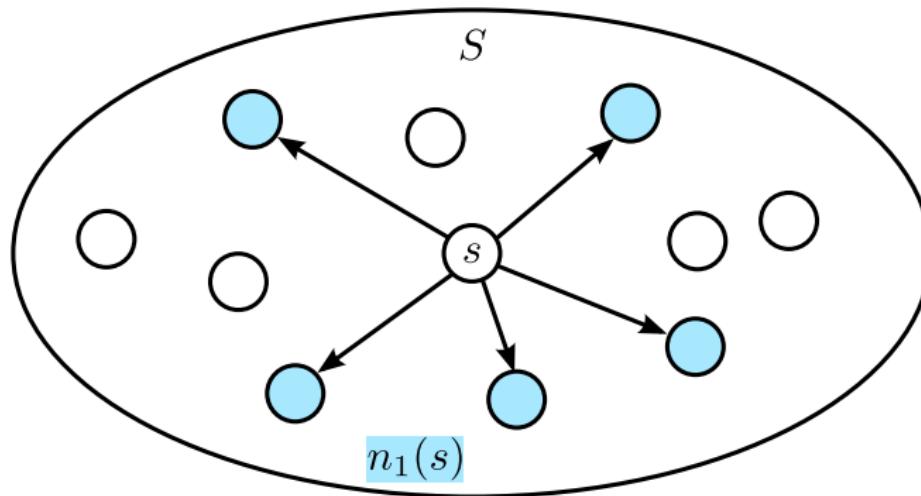
1: function RVNS-MIN( $f$ ,  $N_k$ ,  $k^{max}$ )
2:    $s \leftarrow StartSolution$ ,  $s^* \leftarrow s$ 
3:   repeat
4:      $k \leftarrow 1$ 
5:     repeat
6:        $s' \leftarrow$  random solution in  $N_k$            ▷ “Shake” Step
7:       if  $f(s') < f(s)$  then
8:          $s \leftarrow s'$ 
9:          $k \leftarrow 1$ 
10:        if  $f(s) < f(s^*)$  then  $s^* \leftarrow s$  end if
11:        else
12:           $k \leftarrow k + 1$ 
13:        until  $k = k^{max}$ 
14:      until Terminate
15:  return  $s^*$ 

```

## Large neighborhood search

# Large neighborhoods

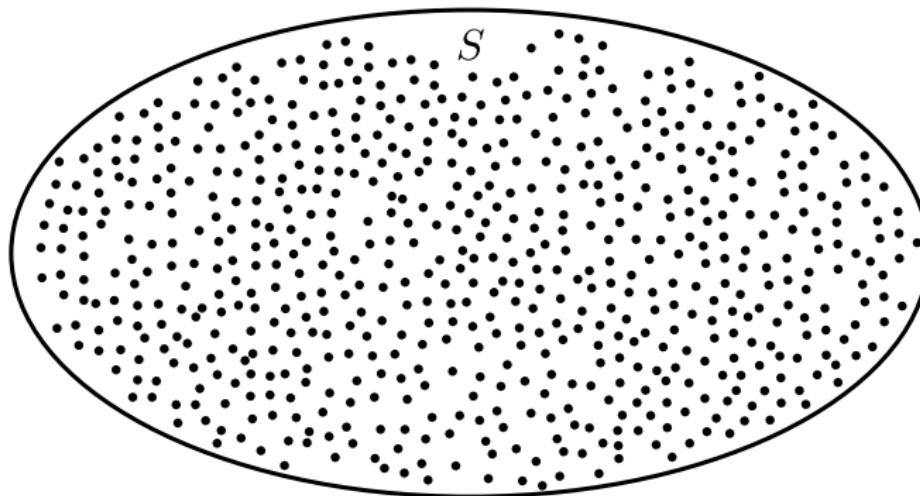
- We have already seen local neighborhoods ...



- These neighborhoods are generally of polynomial size.

# Large neighborhoods

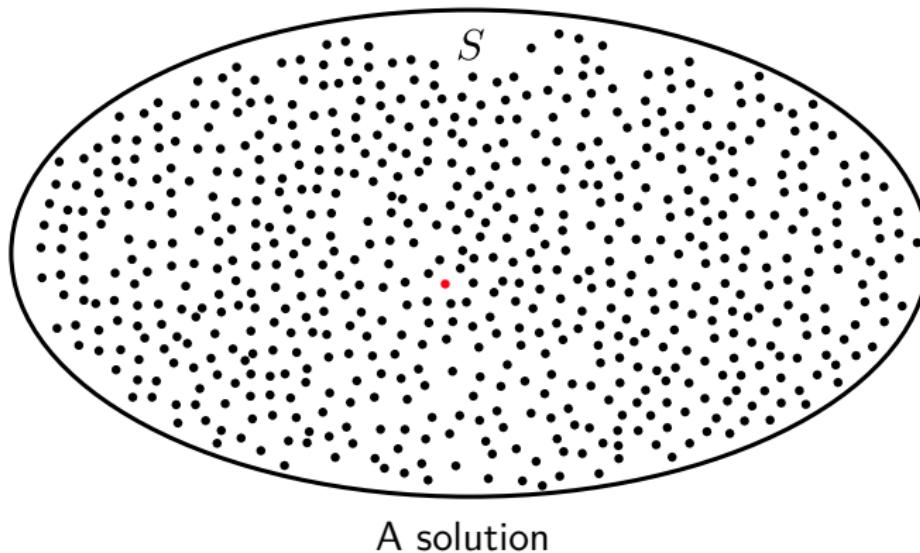
- Now we will look at neighborhoods that normally have an exponential or high polynomial size.



A large solution space

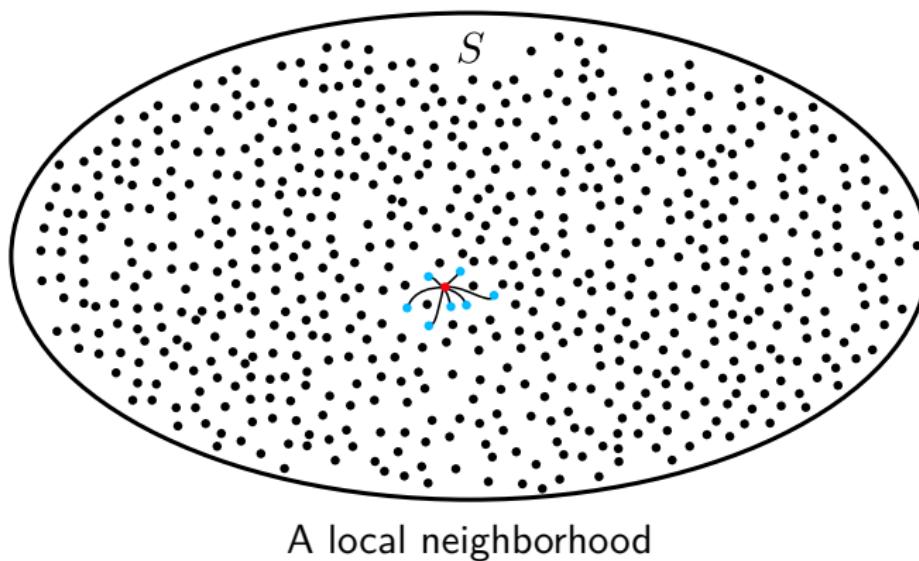
# Large neighborhoods

- Now we will look at neighborhoods that normally have an exponential or high polynomial size.



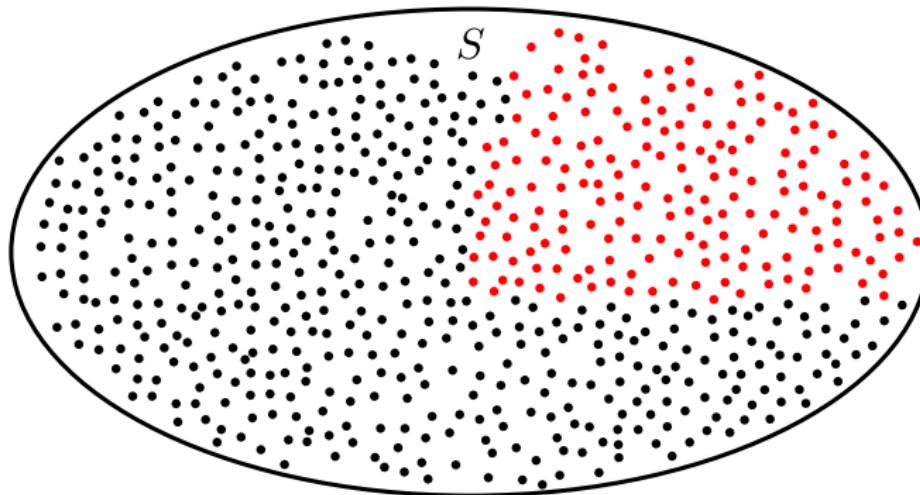
# Large neighborhoods

- Now we will look at neighborhoods that normally have an exponential or high polynomial size.



# Large neighborhoods

- Now we will look at neighborhoods that normally have an exponential or high polynomial size.



# Large neighborhood search: destroy and repair

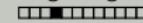
- *Destroy and repair* operators implicitly define neighborhoods.

# Large neighborhood search: destroy and repair

- *Destroy* and *repair* operators implicitly define neighborhoods.

## Destroy

Destroys (deletes) a part of the solution using a heuristic.



# Large neighborhood search: destroy and repair

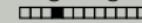
- *Destroy* and *repair* operators implicitly define neighborhoods.

## Destroy

Destroys (deletes) a part of the solution using a heuristic.

## Repair

Constructs a new solution from a destroyed solution using a heuristic.



# Large neighborhood search: destroy and repair

- *Destroy* and *repair* operators implicitly define neighborhoods.

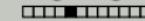
## Destroy

Destroys (deletes) a part of the solution using a heuristic.

## Repair

Constructs a new solution from a destroyed solution using a heuristic.

- Destroy & repair are repeated (with an acceptance criterion) until a stopping criterion is met.



# LNS pseudo-code

```

1: function LNS-MIN
2:    $s \leftarrow StartSolution()$ 
3:    $s^* \leftarrow s$ 
4:   repeat
5:      $s' \leftarrow Repair(Destroy(s))$ 
6:     if  $Accept(s, s')$  then
7:        $s \leftarrow s'$ 
8:     if  $f(s) < f(s^*)$  then
9:        $s^* \leftarrow s$ 
10:    until  $Terminate$ 
11:   return  $s^*$ 
```

See Algorithm 1: Pisinger, D., and Røpke, S. (2010). "Large neighborhood search." Handbook of Metaheuristics. Springer US. 399-419.



# LNS example: VRP

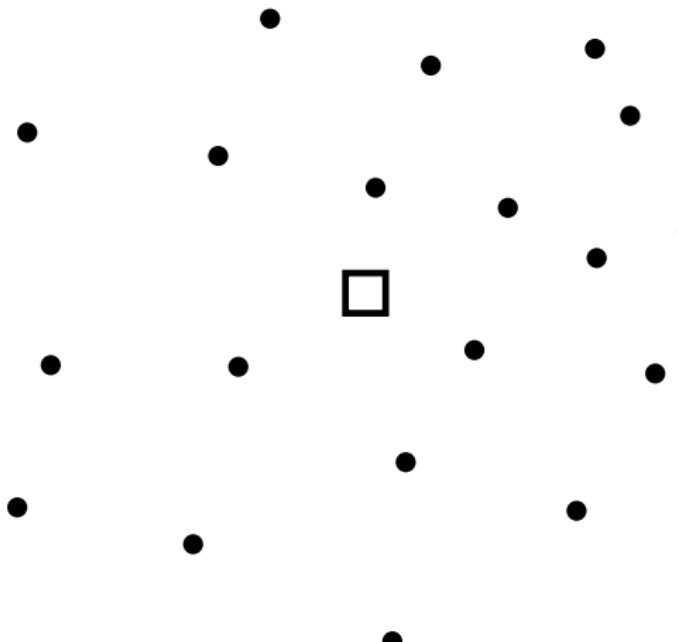
## Vehicle routing problem

Given:

- Graph  $G = (V, E)$ ,  $V = \{Customers\} \cup \{Depot\}$
- $m$  vehicles
- Edge distances  $d_e$

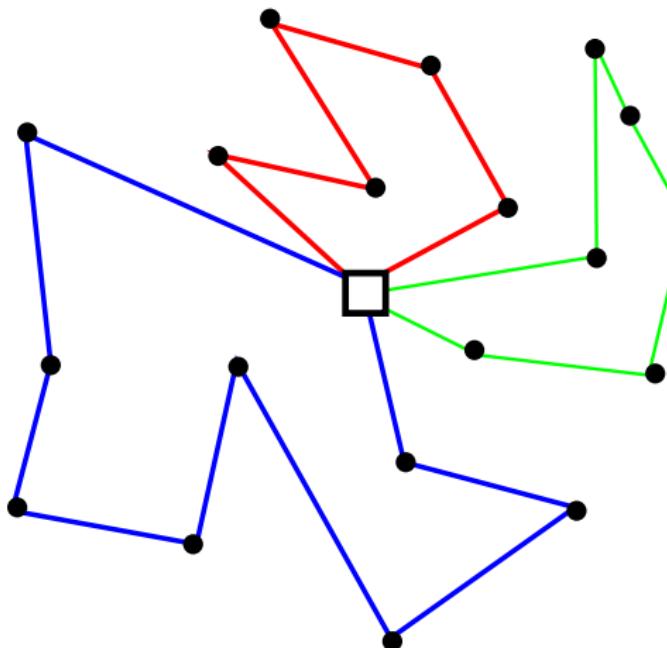
The goal is to find the routes with the minimal total distances travelled, such that all clients are visited exactly once.

## LNS example: VRP

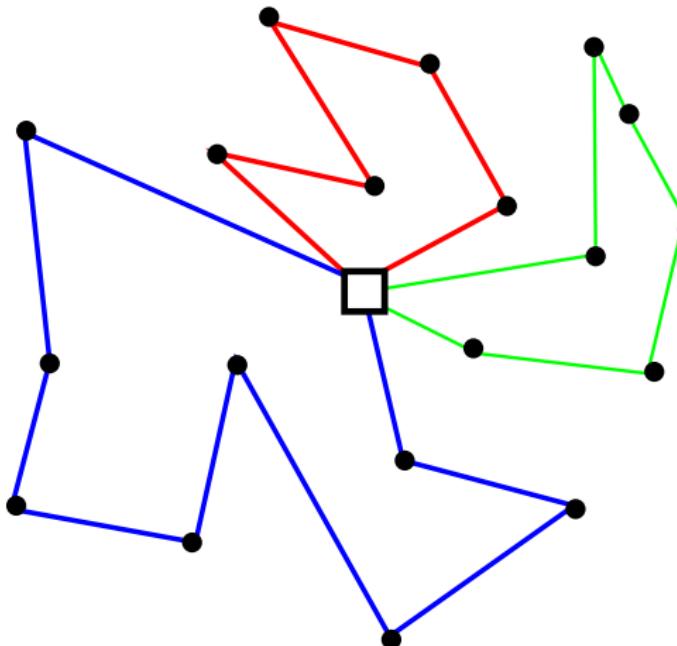


● = Customer, □ = Depot,  $m = 3$

## LNS example: VRP



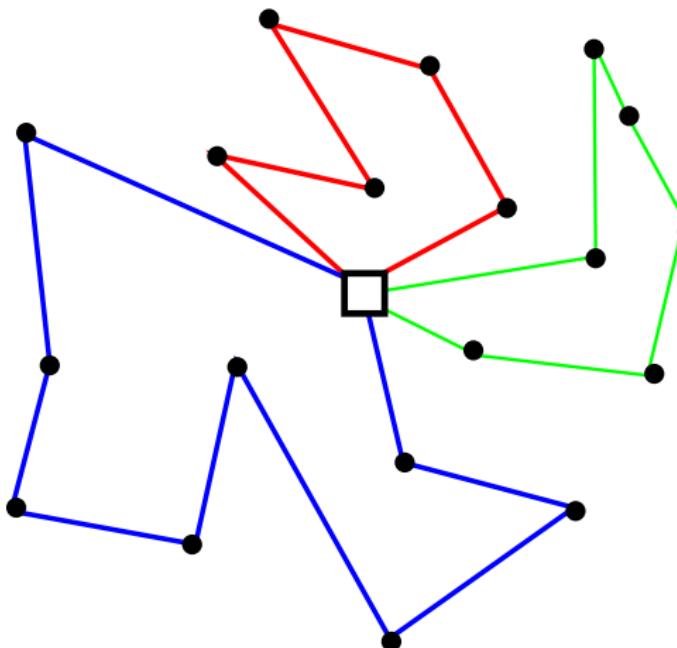
# LNS example: VRP



## Question

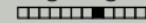
What are some possible destroy operators for the VRP?

# LNS example: VRP



## Question

What are some possible repair operators for the VRP?



# Adaptive large neighborhood search

Given:

- A set of destroy operators  $\Omega^-$
- A set of repair operators  $\Omega^+$

## Question

How do we choose a destroy and repair operator in an iteration?

# Adaptive large neighborhood search

Given:

- A set of destroy operators  $\Omega^-$
- A set of repair operators  $\Omega^+$

## Question

How do we choose a destroy and repair operator in an iteration?

- We use two vectors to give the operators weights

## Adaptive large neighborhood search

**Given:**

- A set of destroy operators  $\Omega^-$
  - A set of repair operators  $\Omega^+$

## Question

How do we choose a destroy and repair operator in an iteration?

- We use two vectors to give the operators weights
  - $\rho^- \in \mathbb{R}^{|\Omega^-|}$  for the destroy operators

# Adaptive large neighborhood search

Given:

- A set of destroy operators  $\Omega^-$
- A set of repair operators  $\Omega^+$

## Question

How do we choose a destroy and repair operator in an iteration?

- We use two vectors to give the operators weights
- $\rho^- \in \mathbb{R}^{|\Omega^-|}$  for the destroy operators
- $\rho^+ \in \mathbb{R}^{|\Omega^+|}$  for the repair operators

# Adaptive large neighborhood search

Given:

- A set of destroy operators  $\Omega^-$
- A set of repair operators  $\Omega^+$

## Question

How do we choose a destroy and repair operator in an iteration?

- We use two vectors to give the operators weights
- $\rho^- \in \mathbb{R}^{|\Omega^-|}$  for the destroy operators
- $\rho^+ \in \mathbb{R}^{|\Omega^+|}$  for the repair operators
- Initially all methods have the same weight.

# Adaptive large neighborhood search

Given:

- A set of destroy operators  $\Omega^-$
- A set of repair operators  $\Omega^+$

## Question

How do we choose a destroy and repair operator in an iteration?

- We use two vectors to give the operators weights
- $\rho^- \in \mathbb{R}^{|\Omega^-|}$  for the destroy operators
- $\rho^+ \in \mathbb{R}^{|\Omega^+|}$  for the repair operators
- Initially all methods have the same weight.
- We use the *roulette wheel* principle to select the operators.

# Adaptive large neighborhood search

Given:

- A set of destroy operators  $\Omega^-$
- A set of repair operators  $\Omega^+$

## Question

How do we choose a destroy and repair operator in an iteration?

- We use two vectors to give the operators weights
- $\rho^- \in \mathbb{R}^{|\Omega^-|}$  for the destroy operators
- $\rho^+ \in \mathbb{R}^{|\Omega^+|}$  for the repair operators
- Initially all methods have the same weight.
- We use the *roulette wheel* principle to select the operators.
- The weights are adjusted dynamically at the search progresses.

We assume the same symbols and math of Pisinger und Røpke (2010)

# ALNS: Operator selection

Probability destroy operator  $i$  selected:

$$\phi_i^- = \frac{\rho_i^-}{\sum_{j=1}^{|\Omega^-|} \rho_j^-} \quad (1)$$

Probability repair operator  $i$  selected:

$$\phi_i^+ = \frac{\rho_i^+}{\sum_{j=1}^{|\Omega^+|} \rho_j^+} \quad (2)$$

# ALNS: Update (For Minimization)

- We have a candidate solution  $s'$ , an old solution  $s$  and a best solution  $s^*$

$$\Psi = \max \begin{cases} \omega_1 & \text{if } f(s') < f(s^*) \\ \omega_2 & \text{if } f(s') < f(s) \\ \omega_3 & \text{if } s' \text{ accepted} \\ \omega_4 & \text{if } s' \text{ not accepted} \end{cases}$$

Normally:  $\omega_1 \geq \omega_2 \geq \omega_3 \geq \omega_4$  (Parameters)

# ALNS: Update (For Minimization)

- We have a candidate solution  $s'$ , an old solution  $s$  and a best solution  $s^*$

$$\Psi = \max \begin{cases} \omega_1 & \text{if } f(s') < f(s^*) \\ \omega_2 & \text{if } f(s') < f(s) \\ \omega_3 & \text{if } s' \text{ accepted} \\ \omega_4 & \text{if } s' \text{ not accepted} \end{cases}$$

Normally:  $\omega_1 \geq \omega_2 \geq \omega_3 \geq \omega_4$  (Parameters)

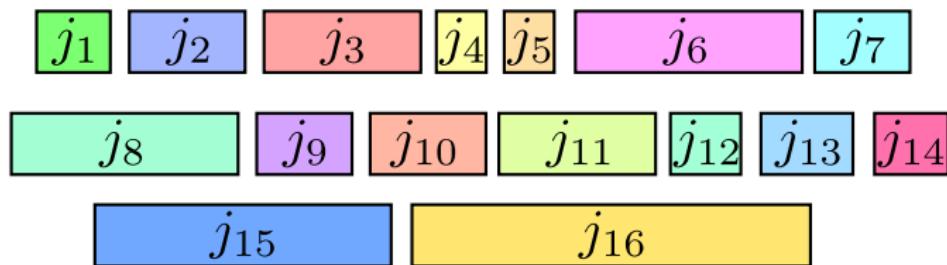
Destroy operator update:  $\rho_i^- \leftarrow \lambda \rho_i^- + (1 - \lambda) \Psi$   
Repair operator update:  $\rho_i^+ \leftarrow \lambda \rho_i^+ + (1 - \lambda) \Psi$   
 $\lambda \in [0, 1]$

# ALNS for the job shop scheduling problem

Machine



Jobs



# ALNS for the job shop scheduling problem

Machine

0	$j_5$	$j_6$	$j_8$	$j_7$	
1	$j_{11}$	$j_9$	$j_2$	$j_{15}$	
2	$j_{10}$	$j_4$	$j_3$	$j_1$	
3		$j_{16}$	$j_{12}$	$j_{13}$	$j_{14}$

# ALNS for the job shop scheduling problem

Machine

0	$j_5$	$j_6$		$j_8$	$j_7$	
1	$j_{11}$	$j_9$	$j_2$		$j_{15}$	
2	$j_{10}$	$j_4$	$j_3$	$j_1$		
3		$j_{16}$		$j_{12}$	$j_{13}$	$j_{14}$

$$f(s^*)$$

# ALNS for the job shop scheduling problem

Machine

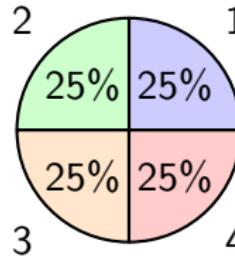
0	$j_5$	$j_6$	$j_8$	$j_7$	
1	$j_{11}$	$j_9$	$j_2$	$j_{15}$	
2	$j_{10}$	$j_4$	$j_3$	$j_1$	
3		$j_{16}$	$j_{12}$	$j_{13}$	$j_{14}$

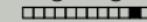
$$f(s^*)$$

Step 1: Choose a destroy operator

$$\Omega^- = \{RndJobSubset, RndMachineSubset, BigJobs, FullMachines\}$$

$$\rho^- = \{1, 1, 1, 1\}$$





# ALNS for the job shop scheduling problem

Machine

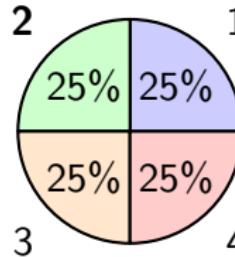
0	$j_5$	$j_6$	$j_8$	$j_7$	
1	$j_{11}$	$j_9$	$j_2$	$j_{15}$	
2	$j_{10}$	$j_4$	$j_3$	$j_1$	
3		$j_{16}$	$j_{12}$	$j_{13}$	$j_{14}$

$$f(s^*)$$

Step 1: Choose a destroy operator

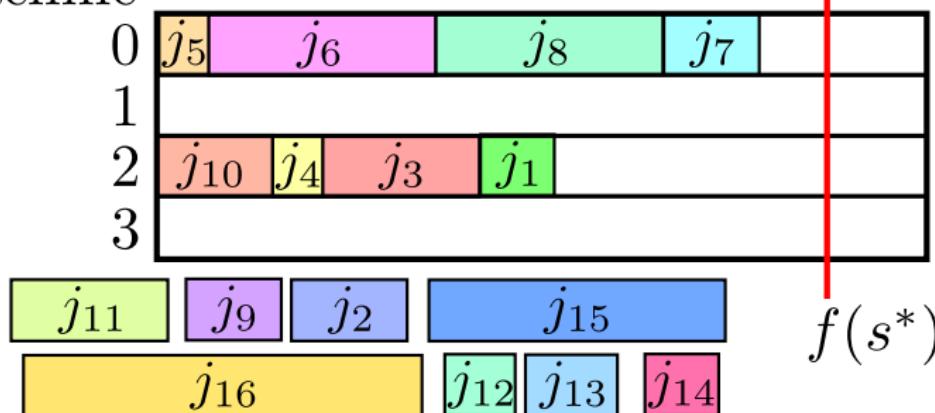
$$\Omega^- = \{RndJobSubset, \textbf{RndMachineSubset}, BigJobs, FullMachines\}$$

$$\rho^- = \{1, 1, 1, 1\}$$



# ALNS for the job shop scheduling problem

Machine



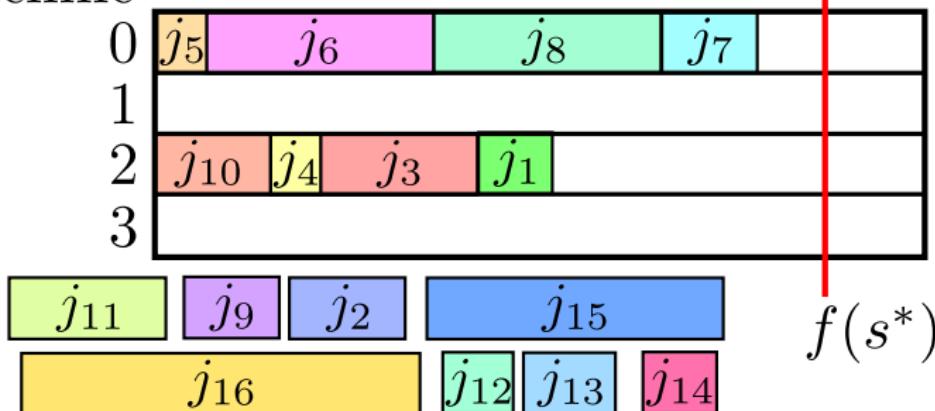
Step 2: Choose a repair operator

$$\Omega^+ = \{SmJobFirstFill, LgJobFirstFill, RandomFill\}$$

$$\rho^+ = \{1, 1, 1\}$$

# ALNS for the job shop scheduling problem

Machine

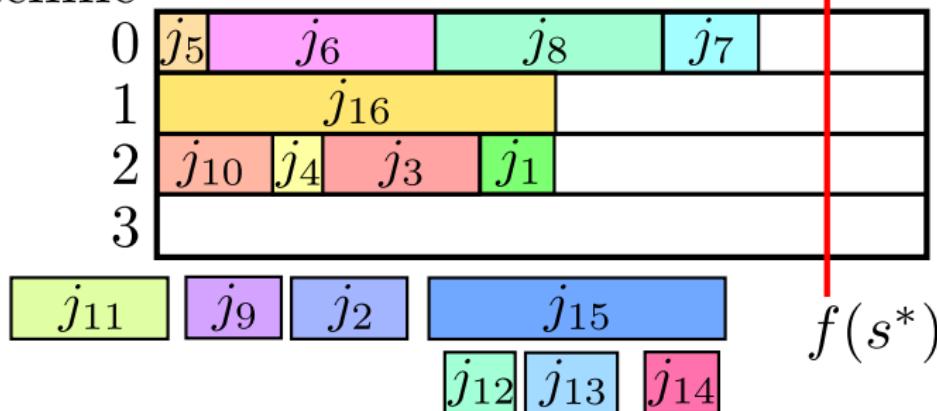


Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, \mathbf{LgJobFirstFill}, RandomFill\} \\ \rho^+ &= \{1, 1, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine

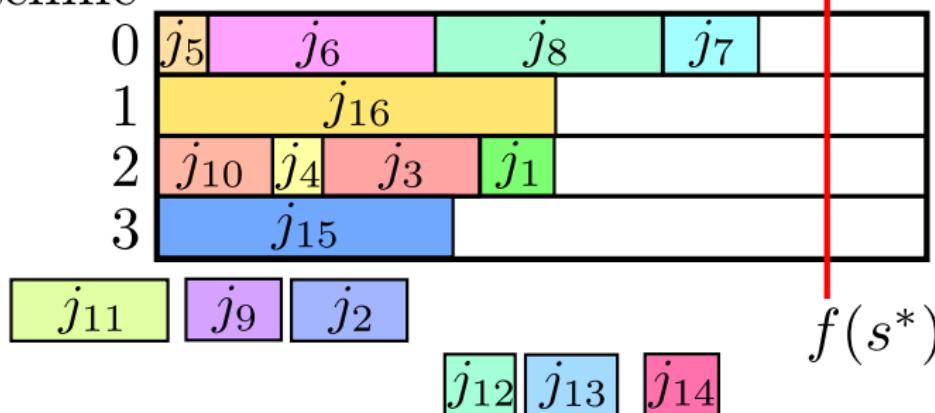


Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, \textbf{LgJobFirstFill}, RandomFill\} \\ \rho^+ &= \{1, 1, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine

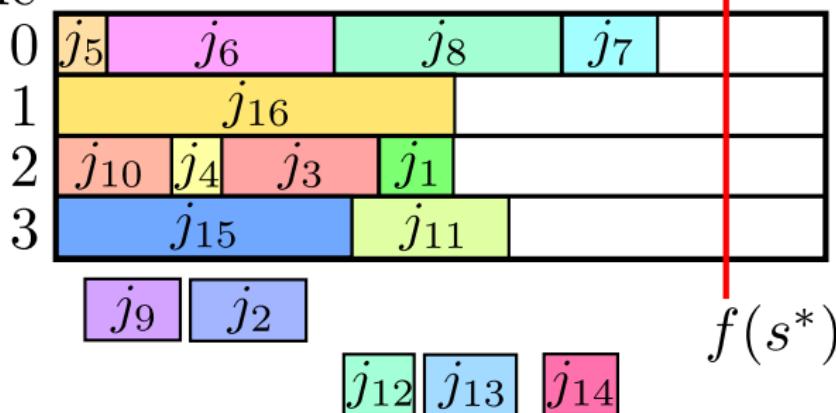


Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, \textbf{LgJobFirstFill}, RandomFill\} \\ \rho^+ &= \{1, 1, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine



Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, \mathbf{LgJobFirstFill}, RandomFill\} \\ \rho^+ &= \{1, 1, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine

0	$j_5$	$j_6$	$j_8$	$j_7$	
1		$j_{16}$	$j_2$	$j_{12}$	
2	$j_{10}$	$j_4$	$j_3$	$j_1$	$j_9$
3		$j_{15}$	$j_{11}$	$j_{13}$	

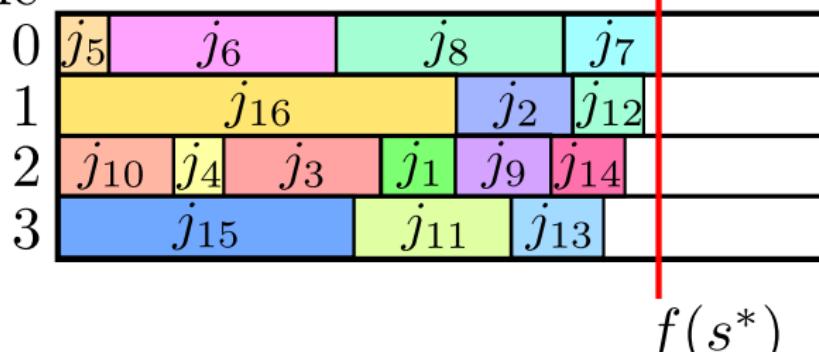
$$f(s^*)$$

Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, \textbf{LgJobFirstFill}, RandomFill\} \\ \rho^+ &= \{1, 1, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine



Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, \textbf{LgJobFirstFill}, RandomFill\} \\ \rho^+ &= \{1, 1, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine

0	$j_5$	$j_6$	$j_8$	$j_7$	
1		$j_{16}$	$j_2$	$j_{12}$	
2	$j_{10}$	$j_4$	$j_3$	$j_1$	$j_9$
3		$j_{15}$	$j_{11}$	$j_{13}$	

$$f(s^*)$$

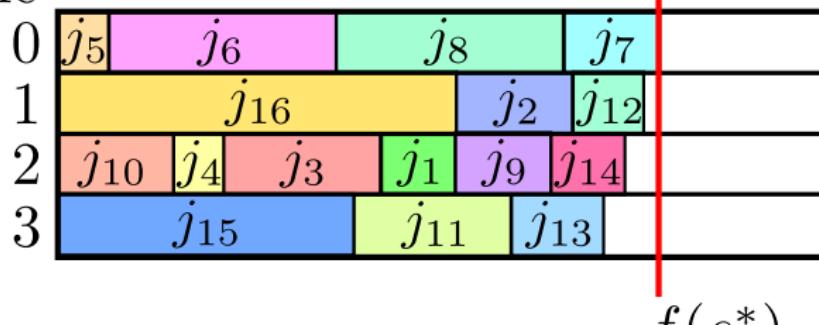
Step 3:Accept solution and update  $\Omega^-$  and  $\Omega^+$

$$\Omega^- \leftarrow \{1, \lambda(1) + (1 - \lambda)\omega_1, 1, 1\}$$

$$\Omega^+ \leftarrow \{1, \lambda(1) + (1 - \lambda)\omega_1, 1\}$$

# ALNS for the job shop scheduling problem

Machine



Step 3: Accept solution and update  $\Omega^-$  and  $\Omega^+$

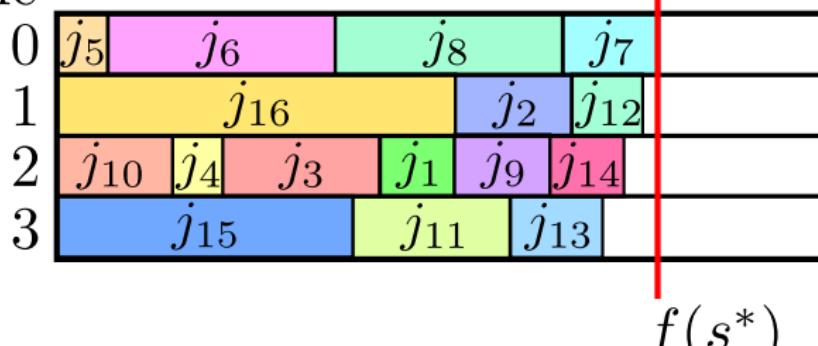
$$\Omega^- \leftarrow \{1, \lambda(1) + (1 - \lambda)\omega_1, 1, 1\}$$

$$\Omega^+ \leftarrow \{1, \lambda(1) + (1 - \lambda)\omega_1, 1\}$$

Sei  $\lambda = 0.8$  und  $\omega_1 = 5$

# ALNS for the job shop scheduling problem

Machine



Step 3: Accept solution and update  $\Omega^-$  and  $\Omega^+$

$$\Omega^- \leftarrow \{1, 1.8, 1, 1\}$$

$$\Omega^+ \leftarrow \{1, 1.8, 1\}$$

Sei  $\lambda = 0.8$  und  $\omega_1 = 5$

# ALNS for the job shop scheduling problem

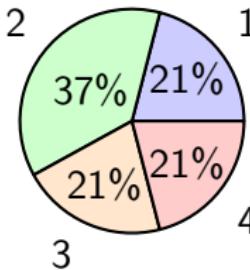
Machine

0	$j_5$	$j_6$	$j_8$	$j_7$	
1		$j_{16}$	$j_2$	$j_{12}$	
2	$j_{10}$	$j_4$	$j_3$	$j_1$	$j_9$
3		$j_{15}$	$j_{11}$	$j_{13}$	

Step 4: Go to Step 1: Choose a destroy operator  $f(s^*)$

$$\Omega^- = \{RndJobSubset, RndMachineSubset, BigJobs, FullMachines\}$$

$$\rho^- = \{1, 1.8, 1, 1\}$$



# ALNS for the job shop scheduling problem

Machine

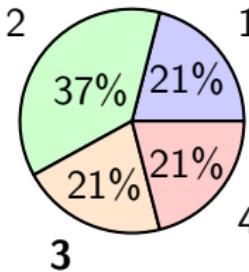
0	$j_5$	$j_6$	$j_8$	$j_7$	
1		$j_{16}$	$j_2$	$j_{12}$	
2	$j_{10}$	$j_4$	$j_3$	$j_1$	$j_9$
3		$j_{15}$	$j_{11}$	$j_{13}$	

$$f(s^*)$$

Step 4: Go to Step 1: Choose a destroy operator

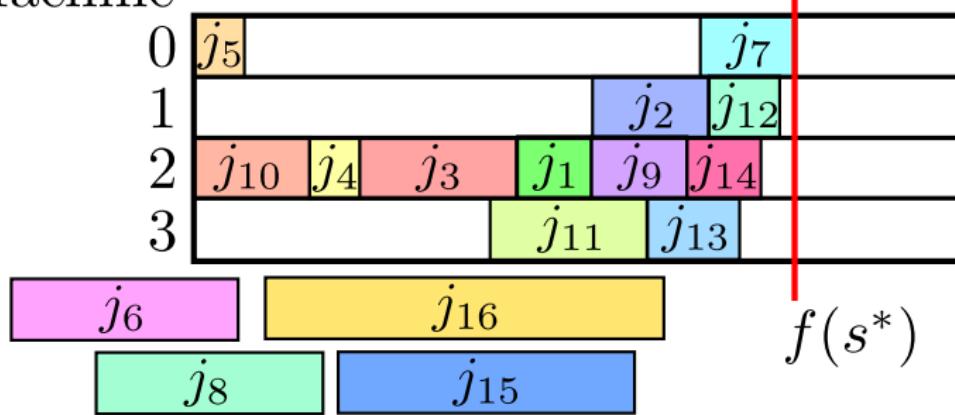
$$\Omega^- = \{RndJobSubset, RndMachineSubset, \textbf{BigJobs}, FullMachines\}$$

$$\rho^- = \{1, 1.8, 1, 1\}$$



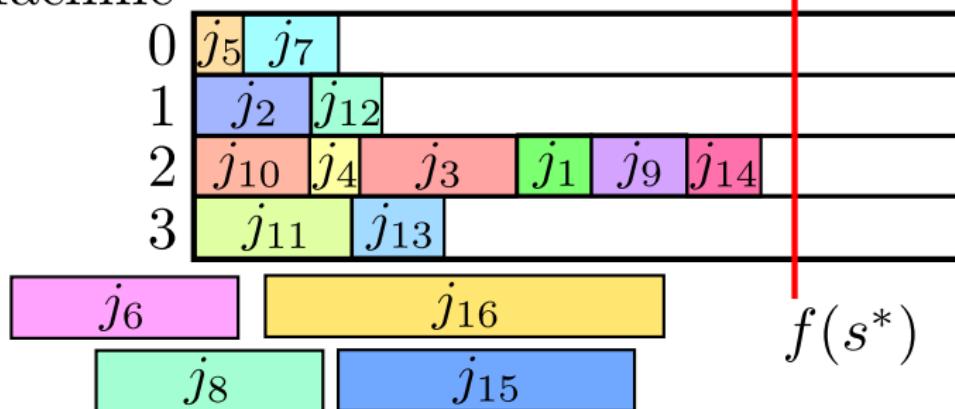
## ALNS for the job shop scheduling problem

Machine



# ALNS for the job shop scheduling problem

Machine

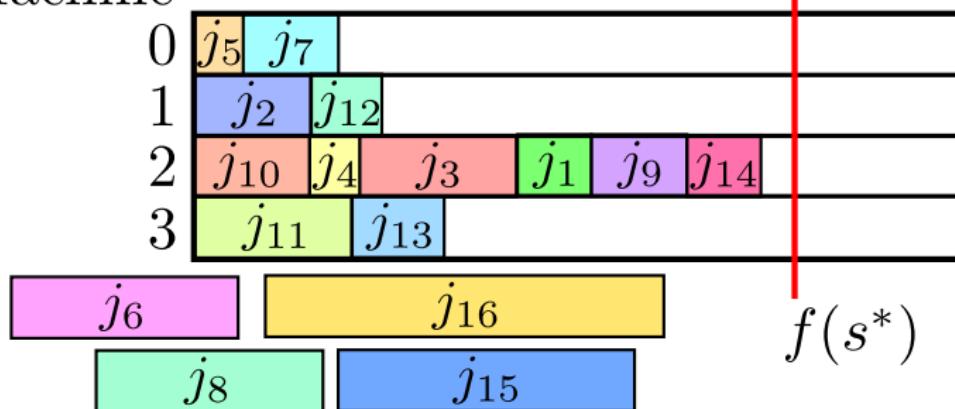


Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, LgJobFirstFill, RandomFill\} \\ \rho^+ &= \{1, 1.8, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine

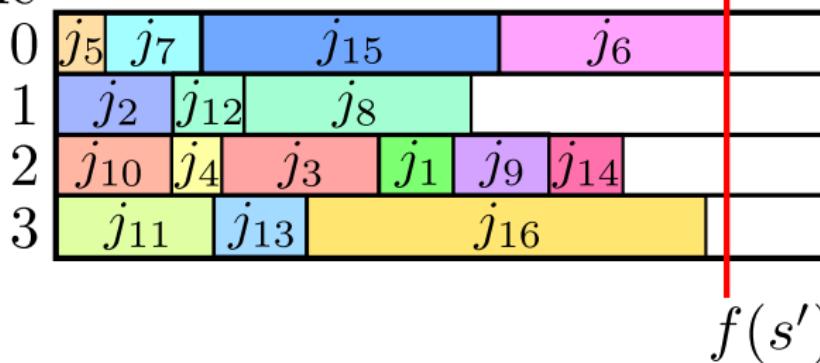


Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, LgJobFirstFill, \textbf{RandomFill}\} \\ \rho^+ &= \{1, 1.8, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine

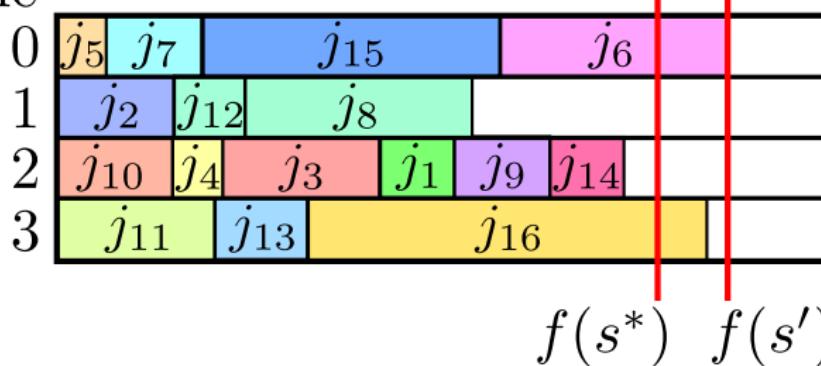


Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, LgJobFirstFill, \textbf{RandomFill}\} \\ \rho^+ &= \{1, 1.8, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine

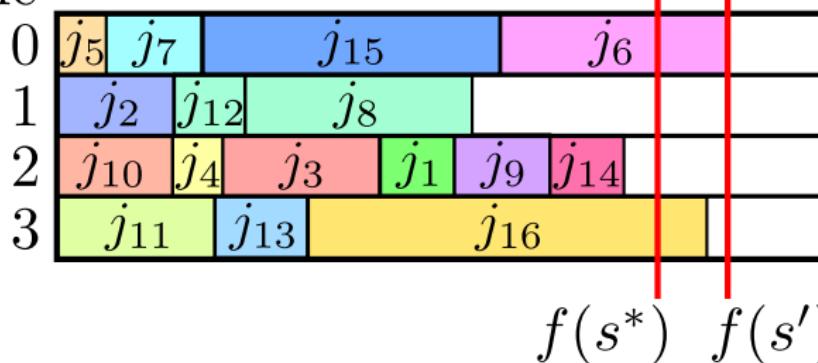


Step 2: Choose a repair operator

$$\begin{aligned}\Omega^+ &= \{SmJobFirstFill, LgJobFirstFill, \textbf{RandomFill}\} \\ \rho^+ &= \{1, 1.8, 1\}\end{aligned}$$

# ALNS for the job shop scheduling problem

Machine



Step 3: Accept solution and update  $\Omega^-$  and  $\Omega^+$

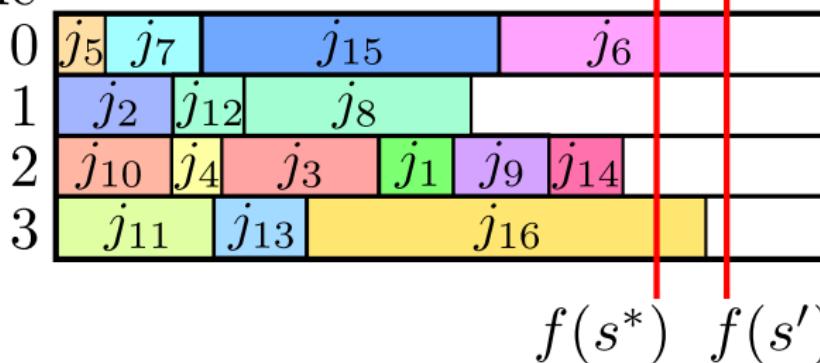
$$\Omega^- \leftarrow \{1, 1.8, \lambda(1) + (\lambda - 1)\omega_3, 1\}$$

$$\Omega^+ \leftarrow \{1, 1.8, \lambda(1) + (\lambda - 1)\omega_3\}$$

$$\lambda = 0.8 \text{ and } \omega_3 = 1$$

# ALNS for the job shop scheduling problem

Machine



Step 3: Accept solution and update  $\Omega^-$  and  $\Omega^+$

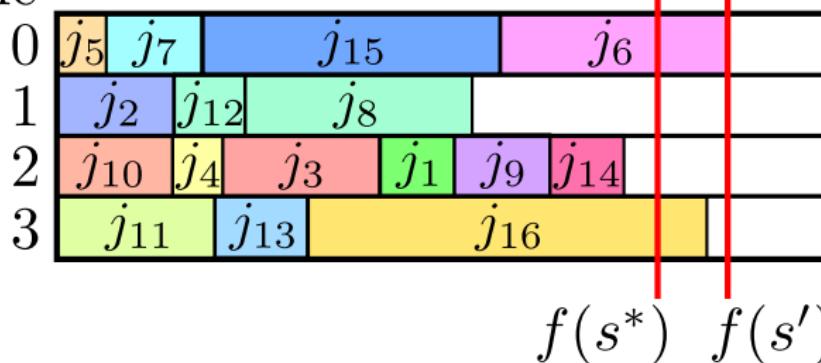
$$\Omega^- \leftarrow \{1, 1.8, \lambda(1) + (\lambda - 1)\omega_3, 1\}$$

$$\Omega^+ \leftarrow \{1, 1.8, \lambda(1) + (\lambda - 1)\omega_3\}$$

Note: If we had not accepted the solution in this step, we would use  $\omega_4$  instead of  $\omega_3$ .

# ALNS for the job shop scheduling problem

Machine



Step 3: Accept solution and update  $\Omega^-$  and  $\Omega^+$

$$\Omega^- \leftarrow \{1, 1.8, 1, 1\}$$

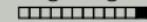
$$\Omega^+ \leftarrow \{1, 1.8, 1\}$$

# Destroy & repair: considerations

- Destroy: “degree of destruction”

# Destroy & repair: considerations

- Destroy: “degree of destruction”
  - Little destroyed: intensification



# Destroy & repair: considerations

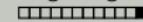
- Destroy: “degree of destruction”
  - Little destroyed: intensification
  - More destroyed: diversification

# Destroy & repair: considerations

- Destroy: “degree of destruction”
  - Little destroyed: intensification
  - More destroyed: diversification
  - Can destroy specific structures of the problem

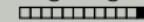
# Destroy & repair: considerations

- Destroy: “degree of destruction”
  - Little destroyed: intensification
  - More destroyed: diversification
  - Can destroy specific structures of the problem
- Repair



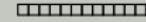
# Destroy & repair: considerations

- Destroy: “degree of destruction”
  - Little destroyed: intensification
  - More destroyed: diversification
  - Can destroy specific structures of the problem
- Repair
  - Can rebuild specific structures of the problem



# Destroy & repair: considerations

- Destroy: “degree of destruction”
  - Little destroyed: intensification
  - More destroyed: diversification
  - Can destroy specific structures of the problem
- Repair
  - Can rebuild specific structures of the problem
  - Look for the optimal solution in the destroyed area?



# When does LNS work?

The usual question

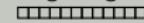
For which problems should we use LNS?

# When does LNS work?

The usual question

For which problems should we use LNS?

- Problems with large neighborhoods!

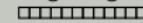


# When does LNS work?

The usual question

For which problems should we use LNS?

- Problems with large neighborhoods!
- Routing problems (especially VRP variants)



# When does LNS work?

The usual question

For which problems should we use LNS?

- Problems with large neighborhoods!
- Routing problems (especially VRP variants)
- Scheduling problems

## Literature

# Literature

- Prescott-Gagnon, Eric, Guy Desaulniers, and Louis-Martin Rousseau. "A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows." *Networks* 54.4 (2009): 190-204.
- Pisinger, David, and Stefan Røpke. "Large neighborhood search." *Handbook of metaheuristics*. Springer US, 2010. 399-419.
- Hansen, Pierre, and Nenad Mladenović. "Variable neighborhood search." *Handbook of metaheuristics*. Springer US, 2010. 61-86.

# Thank you for your attention!

Leuphana University of Lüneburg  
Business information systems, in particular Operations Research  
Prof. Dr. Lin Xie  
Universitätsallee 1  
Building 4, Room 314  
21335 Lüneburg  
Phone +49 4131 677 2305  
Fax +49 4131 677 1749  
[xie@leuphana.de](mailto:xie@leuphana.de)

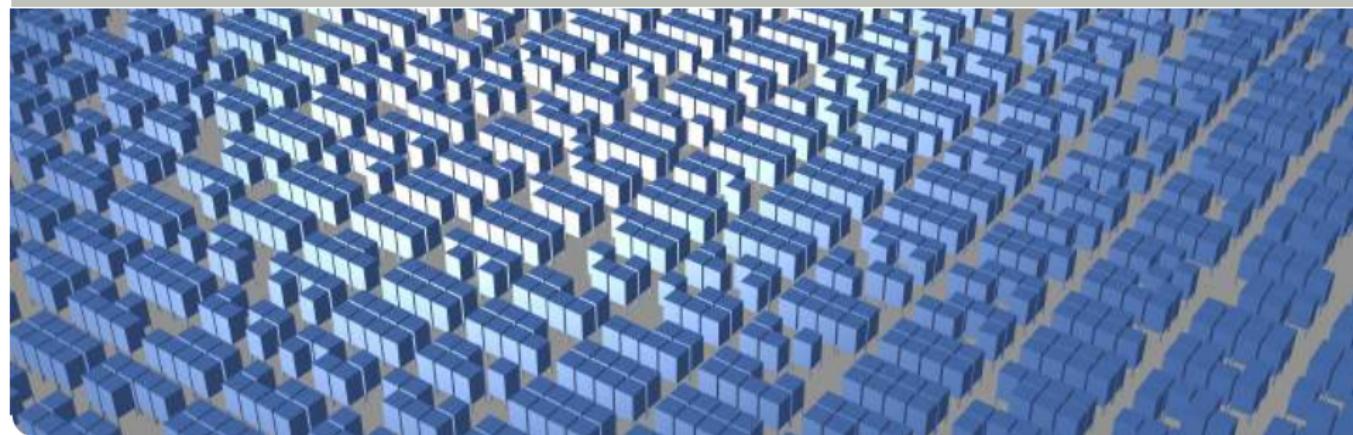


# Analysing Networks with OR-Methods

Part 10 – Genetic Algorithm

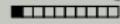
Lin Xie | 22.06.2021

PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH

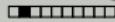


# Agenda

- 1** GA Overview
- 2** Guiding example: 1-max
  - 1** Selection
  - 2** Recombination
  - 3** Mutation
- 3** Finding the right genome representation
- 4** Practical considerations



# Genetic Algorithms



# Genetic Algorithms (GA)

## Definition

Genetic algorithms simulate the evolution of a population over several generations.

# Genetic Algorithms (GA)

## Definition

Genetic algorithms simulate the evolution of a population over several generations.

GAs work as follows:

- 1 Generate initial solutions
- 2 Loop:

- 
- 1 Fitness evaluation
  - 2 Selection
  - 3 Generate successors through *recombination* of genomes.
  - 4 *Mutation* of the successors.

potential termination criteria

# Genetic Algorithms (GA)

## Definition

Genetic algorithms simulate the evolution of a population over several generations.

GAs work as follows:

- 1 Generate initial solutions
- 2 Loop:
  - 1 Fitness evaluation
  - 2 Selection
  - 3 Generate successors through *recombination* of genomes.
  - 4 *Mutation* of the successors.

## Question

What is a potential termination criterion for a GA?

many options

given number of iterations  
check relationship: best solution object. fits  
Last 10 no improvement  
terminate

# GA Example: 1-max

- The genome: bit string with length 8.
- Fitness function:  $f(x) = \underline{\text{number of 1s}}$  in the bit string.
- Objective:  $\max f(x)$
- e.g.:  $f(1\underset{1}{1}\underset{2}{1}0\underset{3}{1}\underset{4}{1}\underset{5}{1}1\underset{6}{0}) = 6$

# GA Example: 1-max: Initialization

Step 1: Initialize the population

4 parents

(Fitness: 4)

(Fitness: 5)

(Fitness: 3)

(Fitness: 3)

# GA Example: 1-max: Selection

## Schritt 2: Selection (Roulette Wheel Selection)

- We need 2 genomes to generate successors.
- Many options exist to determine the successors.

A (4)

B (5)

C (3)

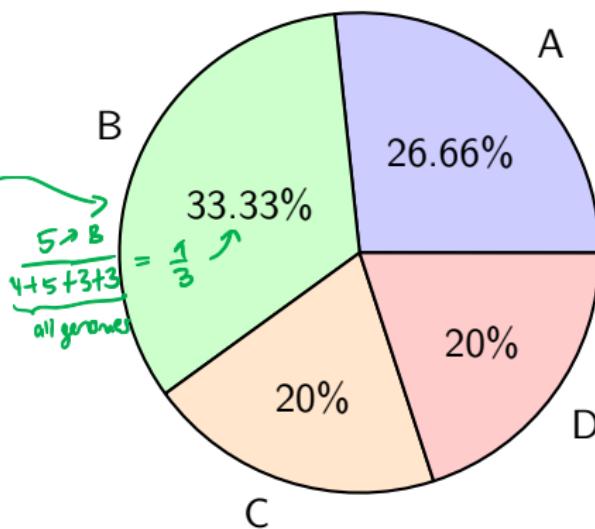
D (3)

# GA Example: 1-max: Selection

## Schritt 2: Selection (Roulette Wheel Selection)

- We need 2 genomes to generate successors.
- Many options exist to determine the successors.

A		(4)
B		(5)
C		(3)
D		(3)

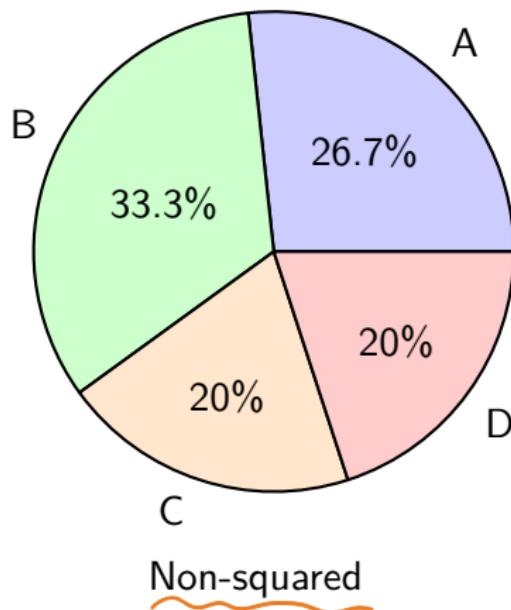


We spin the roulette wheel twice to get two genomes.

# GA Example: 1-max: Fitness Scaling

- We can square the fitness function before we create the probability distribution.

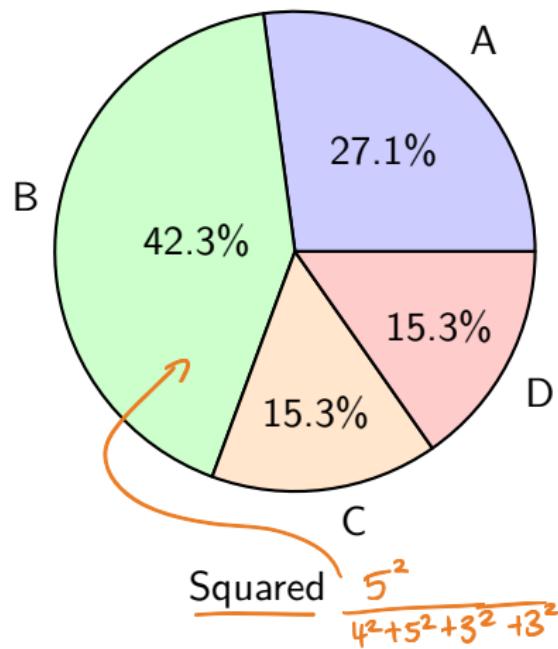
A		(4)
B		(5)
C		(3)
D		(3)



# GA Example: 1-max: Fitness Scaling

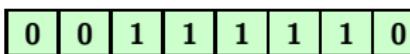
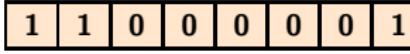
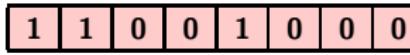
- We can square the fitness function before we create the probability distribution.

A	<table border="1"> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> </table>	0	1	1	1	0	0	0	1	(4)
0	1	1	1	0	0	0	1			
B	<table border="1"> <tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> </table>	0	0	1	1	1	1	1	0	(5)
0	0	1	1	1	1	1	0			
C	<table border="1"> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> </table>	1	1	0	0	0	0	0	1	(3)
1	1	0	0	0	0	0	1			
D	<table border="1"> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> </table>	1	1	0	0	1	0	0	0	(3)
1	1	0	0	1	0	0	0			



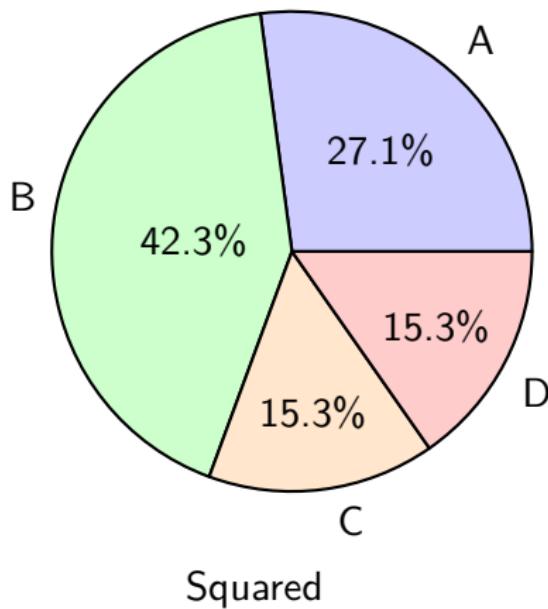
# GA Example: 1-max: Fitness Scaling

- We can square the fitness function before we create the probability distribution.

A		(4)
B		(5)
C		(3)
D		(3)



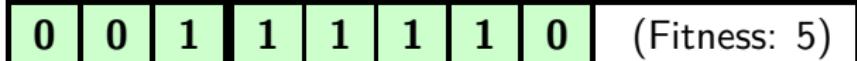
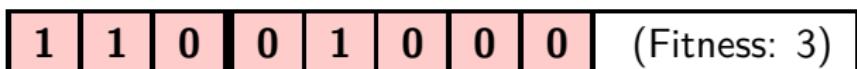
**Advantage:** More probability mass on good solutions (more intensification)



# GA Example: 1-max: Recombination

- We selected these two genomes with the roulette wheel.
- Recombination is performed through a “crossover” operation.
- Here: single point crossover.

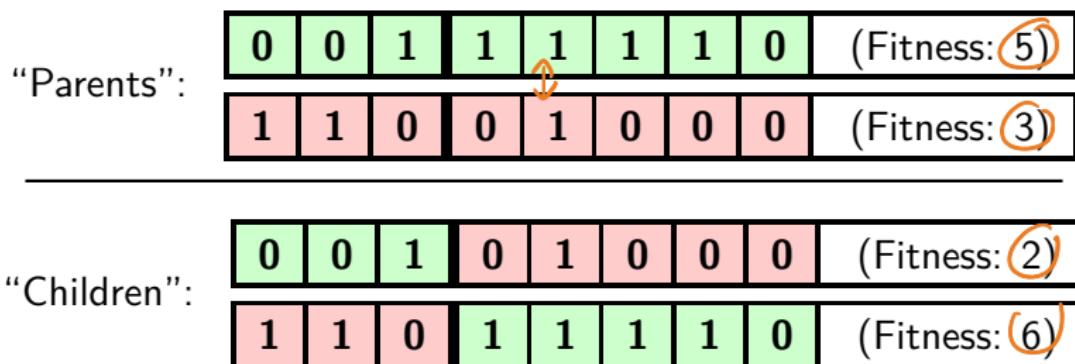
“Parents”:

	(Fitness: 5)
	(Fitness: 3)

*exchange both up to this point*

# GA Example: 1-max: Recombination

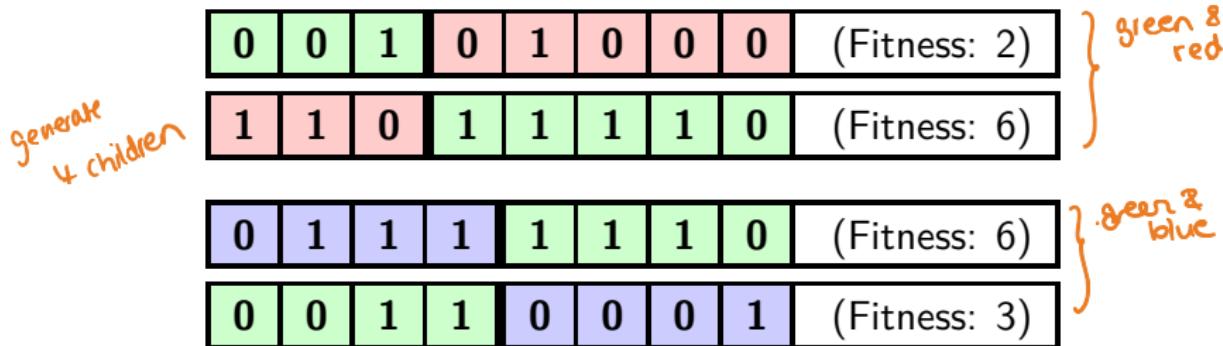
- We selected these two genomes with the roulette wheel.
- Recombination is performed through a “crossover” operation.
- Here: single point crossover.



# GA Example: 1-max: Recombination

1) wheel to get parent  
2) combination to get children

- Recombination repeats until  $n$  children have been generated.



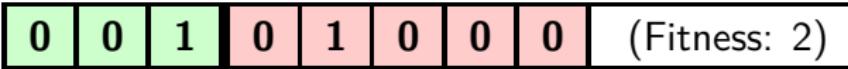
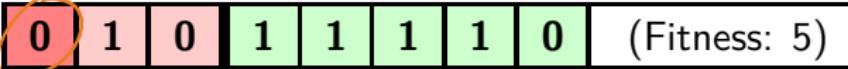
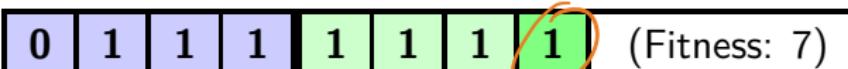
- There are many recombination operators, some of which are problem specific.

Use single point crossover

*m = possibility*

## GA Example: 1-max: Mutation

- With probability  $m$  per bit, invert the bit.

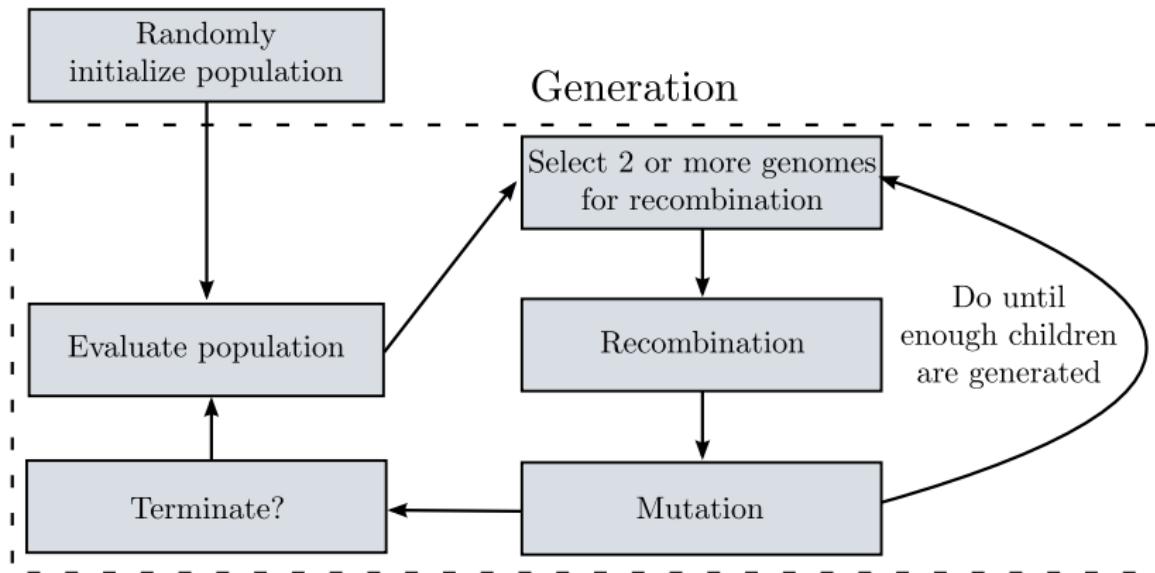





Randomly  
Chosen



- The generation is now complete.
- We repeat the process again for the next generation:  
Evaluation, Selection, Recombination, **Mutation**.

# GA Overview



## A closer look at recombination

# Recombination

## Question

Is it possible to find a 1-max optimum for the following genomes only with single point crossover?

A  (Fitness: 3)

B  (Fitness: 5)

C  (Fitness: 3)

D  (Fitness: 3)

Not possible to change to 1

# Recombination

## Question

Is it possible to find a 1-max optimum for the following genomes only with single point crossover?

A	0	1	0	1	0	0	0	1	(Fitness: 3)
B	0	0	1	1	0	1	1	0	(Fitness: 5)
C	1	1	0	0	0	0	0	1	(Fitness: 3)
D	1	1	0	0	0	1	0	0	(Fitness: 3)

If change to 1  
= Mutation

## Answer

No: The bit in position 5 cannot become 1.

1) Mutation  
2) or better cross over

# Recombination

## Question

Is it possible to find a 1-max optimum for the following genomes only with single point crossover?

A (Fitness: 3)

B (Fitness: 5)

C (Fitness: 3)

D (Fitness: 3)

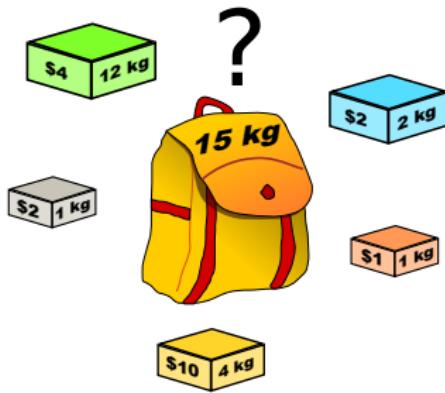
## Answer

No: The bit in position 5 cannot become 1.

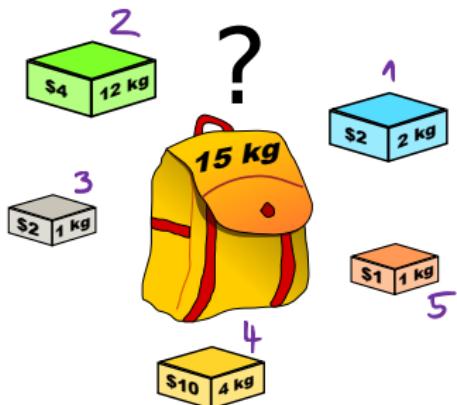
- This means that mutation is important (and/or a better recombination operator).

# Genome Representations

# Genome Representation: Knapsack Problem

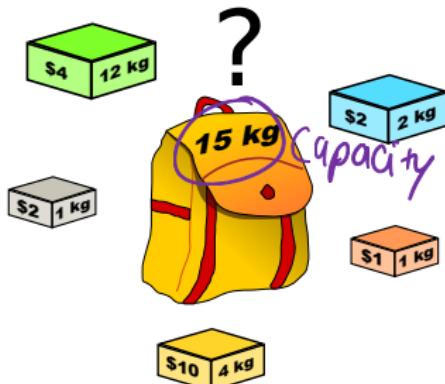


# Genome Representation: Knapsack Problem



A possible genome: Bit string (1 = take item, 0 = don't take item).

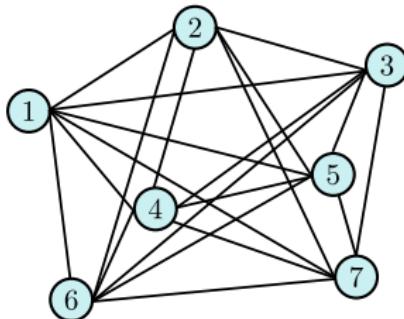
# Genome Representation: Knapsack Problem



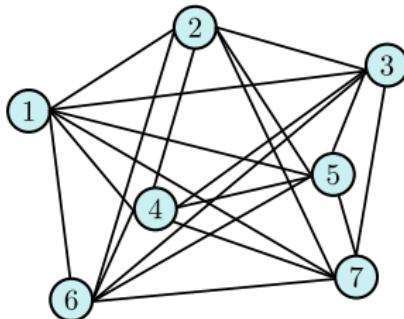
A possible genome: Bit string (1 = take item, 0 = don't take item).

Item	1	2	3	4	5
Individual 1	0	1	0	0	1
Individual 2	1	0	1	0	0
Individual 3	1	0	1	1	1
Individual 4	0	1	1	0	0

# Genome Representation: Travelling Salesman Problem

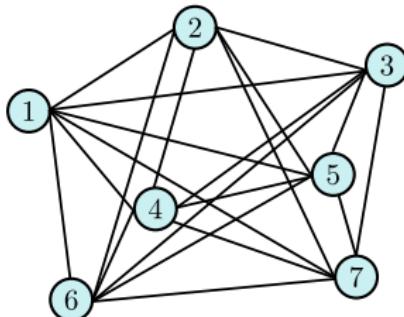


# Genome Representation: Travelling Salesman Problem



A possible genome: Permutation of the nodes.

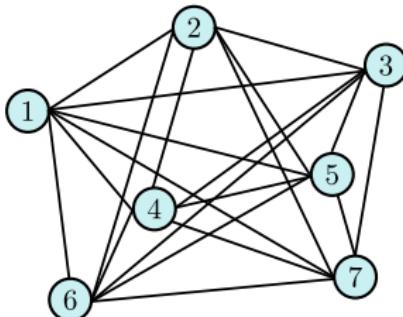
# Genome Representation: Travelling Salesman Problem



A possible genome: Permutation of the nodes.

Visit number:	1	2	3	4	5	6	7
Individual 1	1	2	5	3	7	6	4
Individual 2	1	6	4	7	5	2	3
Individual 3	1	7	4	2	5	3	6
Individual 4	1	2	3	7	6	4	5

# Genome Representation: Travelling Salesman Problem



A possible genome: Permutation of the nodes.

Visit number:	1	2	3	4	5	6	7
Individual 1	1	2	5	3	7	6	4
Individual 2	1	6	4	7	5	2	3
Individual 3	1	7	4	2	5	3	6
Individual 4	1	2	3	7	6	4	5

**Are there any issues with such a genome?**

# Genome Representation: Travelling Salesman Problem

Consider single point crossover using the following individuals:

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3

# Genome Representation: Travelling Salesman Problem

Consider single point crossover using the following individuals:

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3

# Genome Representation: Travelling Salesman Problem

Consider single point crossover using the following individuals:

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3

Child 1	1	2	5	7	5	2	3
Child 2	1	6	4	3	7	6	4

# Genome Representation: Travelling Salesman Problem

Consider single point crossover using the following individuals:

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3

Child 1	1	2	5	7	5	2	3
Child 2	1	6	4	3	7	6	4

**The child tours visit the same node twice and omit a node!**

# Genome Representation: Travelling Salesman Problem

Consider single point crossover using the following individuals:

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3
Child 1	1	2	5	7	5	2	3
Child 2	1	6	4	3	7	6	4

**The child tours visit the same node twice and omit a node!**

Our options:

- 1 Allow infeasible solutions and penalize them.
- 2 Use a repair procedure to fix the infeasibility.
- 3 Change our crossover operator to prevent infeasibility.
- 4 Change our genome representation to prevent infeasibility.

# TSP: The partially-mapped crossover (PMX) operator

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3

Child 1	1	2	5	3	7	6	4
Child 2	1	6	4	7	5	2	3

# TSP: The partially-mapped crossover (PMX) operator

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3

Child 1	1	2	5	3	7	6	4
Child 2	1	6	4	7	5	2	3

- 1 Select two points in the genome

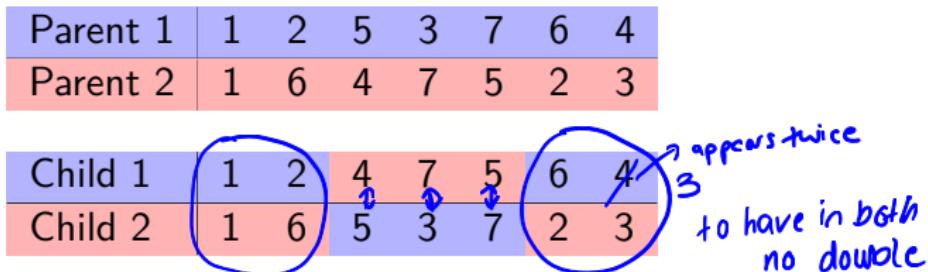
# TSP: The partially-mapped crossover (PMX) operator

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3

Child 1	1	2	4	7	5	6	4
Child 2	1	6	5	3	7	2	3

- 1 Select two points in the genome
- 2 Swap the elements between the points between the genomes.

# TSP: The partially-mapped crossover (PMX) operator



- 1 Select two points in the genome
- 2 Swap the elements between the points between the genomes.
- 3 Use the following mapping to repair non-copied portions of the genome:

$$\begin{array}{l} 5 \leftrightarrow 4 \\ 3 \leftrightarrow 7 \\ 7 \leftrightarrow 5 \end{array}$$

# TSP: The partially-mapped crossover (PMX) operator

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3
Child 1	1	2	4	7	5	6	4
Child 2	1	6	5	3	7	2	3

- 1 Select two points in the genome
- 2 Swap the elements between the points between the genomes.
- 3 Use the following mapping to repair non-copied portions of the genome:  
 $5 \leftrightarrow 4$ ,  $3 \leftrightarrow 7$ ,  $7 \leftrightarrow 5$

# TSP: The partially-mapped crossover (PMX) operator

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3
Child 1	1	2	4	7	5	6	3
Child 2	1	6	5	3	7	2	3

- 1 Select two points in the genome
- 2 Swap the elements between the points between the genomes.
- 3 Use the following mapping to repair non-copied portions of the genome:  
 $5 \leftrightarrow 4, 3 \leftrightarrow 7, 7 \leftrightarrow 5$

Note:  $4$  in child 1 becomes  $5$ , which has the mapping rule  $7 \leftrightarrow 5$  so it is then changed to  $7$ , which has the mapping rule  $3 \leftrightarrow 7$  meaning the final value for this element of the genome is  $3$ !

# TSP: The partially-mapped crossover (PMX) operator

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3
Child 1	1	2	4	7	5	6	3
Child 2	1	6	5	3	7	2	3

- 1 Select two points in the genome
- 2 Swap the elements between the points between the genomes.
- 3 Use the following mapping to repair non-copied portions of the genome:  
 $5 \leftrightarrow 4$ ,  $3 \leftrightarrow 7$ ,  $7 \leftrightarrow 5$

# TSP: The partially-mapped crossover (PMX) operator

Parent 1	1	2	5	3	7	6	4
Parent 2	1	6	4	7	5	2	3
Child 1	1	2	4	7	5	6	3
Child 2	1	6	5	3	7	2	4

- 1 Select two points in the genome
- 2 Swap the elements between the points between the genomes.
- 3 Use the following mapping to repair non-copied portions of the genome:  
 $5 \leftrightarrow 4$ ,  $3 \leftrightarrow 7$ ,  $7 \leftrightarrow 5$

# TSP: Alternative Genome Representation

- We use for example  $L = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7)$  as the ordered list of cities.

# TSP: Alternative Genome Representation

*Readjust List index*

- We use for example  $L = (1 \cancel{2} \cancel{3} 4 \cancel{5} \cancel{6} \cancel{7})$  as the ordered list of cities.

City	1	2	5	3	7	6	4
Parent 1	1	1	3	1	3	2	1
Parent 2	1	5	3	4	3	1	1
	1	6	4	7	5	2	3

# TSP: Alternative Genome Representation

$$L(e_i) = 1$$

- We use for example  $L = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7)$  as the ordered list of cities.

$\nwarrow$

Parent 1	1	1	3	1	3	2	1
Parent 2	1	5	3	4	3	1	1

- We create a path based on the following algorithm, where  $e_i$  is the genome element at position  $i$  in the genome.

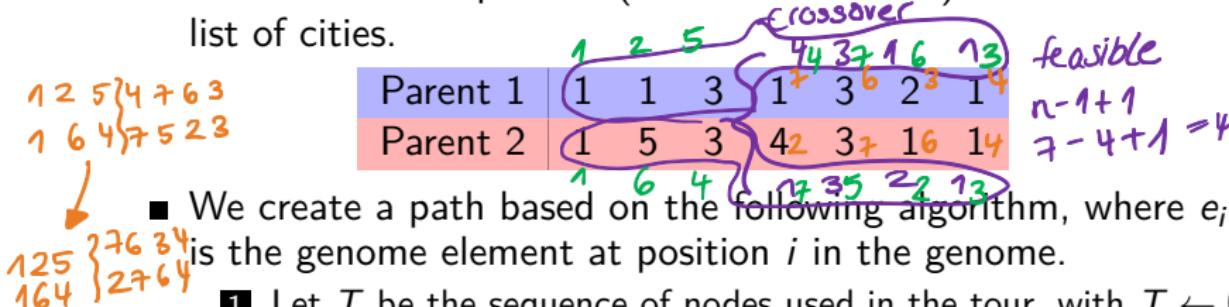
- Let  $T$  be the sequence of nodes used in the tour, with  $T \leftarrow \emptyset$ .
- for  $i = 1 \dots n$ :
- $T \leftarrow T \cup L[e_i]$ ,  $L = L \setminus L[e_i]$

$\hookrightarrow$  Indexing automatic

after Crossover:

# TSP: Alternative Genome Representation

- We use for example  $L = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7)$  as the ordered list of cities.



- We create a path based on the following algorithm, where  $e_i$  is the genome element at position  $i$  in the genome.

- Let  $T$  be the sequence of nodes used in the tour, with  $T \leftarrow \emptyset$ .
- for  $i = 1 \dots n$ :
- $T \leftarrow T \cup L[e_i]$ ,  $L = L \setminus L[e_i]$

## Questions

- Does single-point crossover return feasible solutions with this representation? *yes, because  
Index cannot be larger than 4?*

# TSP: Alternative Genome Representation

- We use for example  $L = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7)$  as the ordered list of cities.

Parent 1	1	1	3	1	3	2	1
Parent 2	1	5	3	4	3	1	1

- We create a path based on the following algorithm, where  $e_i$  is the genome element at position  $i$  in the genome.
  - Let  $T$  be the sequence of nodes used in the tour, with  $T \leftarrow \emptyset$ .
  - for  $i = 1 \dots n$ :
  - $T \leftarrow T \cup L[e_i]$ ,  $L = L \setminus L[e_i]$

## Questions

- Does single-point crossover return feasible solutions with this representation?
- Are there any downsides to using single-point crossover with this representation?

# Summary GA

- 1 GAs have a *population* of solutions

# Summary GA

- 1 GAs have a *population* of solutions
- 2 Through recombination and mutation, the population evolves  
and removes bad members from its ranks  
*and remain good members*

# Summary GA

- 1 GAs have a *population* of solutions
- 2 Through recombination and mutation, the population evolves and removes bad members from its ranks
- 3 GAs provide lots of randomness and are rather effective when a **good genome representation is possible.**

## Practical Considerations

# Practical Considerations

## ■ Genome

- The structure of the problem needs to be present
- If possible, should remain feasible under recombination (or be easy to repair)
- Does not have to be a bit string!

# Practical Considerations

## ■ Genome

- The structure of the problem needs to be present
- If possible, should remain feasible under recombination (or be easy to repair)
- Does not have to be a bit string!

## ■ Population

- Diversity is critical in the initial population
- Normally between 25 and 100 (problem and GA dependent)
- Elitism: Keep the best member or  $x\%$  of the population in each generation.

# Practical Considerations

## ■ Genome

- The structure of the problem needs to be present
- If possible, should remain feasible under recombination (or be easy to repair)
- Does not have to be a bit string!

## ■ Population

- Diversity is critical in the initial population
- Normally between 25 and 100 (problem and GA dependent)
- Elitism: Keep the best member or  $x\%$  of the population in each generation.

## ■ Recombination

- Problem specific operators often help.
- Feasibility must be considered!

↳ after recombination  
genomes should remain feasible

# Practical Considerations

## ■ Genome

- The structure of the problem needs to be present
- If possible, should remain feasible under recombination (or be easy to repair)
- Does not have to be a bit string!

## ■ Population

- Diversity is critical in the initial population
- Normally between 25 and 100 (problem and GA dependent)
- Elitism: Keep the best member or  $x\%$  of the population in each generation.

## ■ Recombination

- Problem specific operators often help.
- Feasibility must be considered!

## ■ Termination Criterion

- One possibility: Measure the average fitness of the population over  $n$  generations ( $n \approx 5$ )

# Practical Considerations

## ■ Genome

- The structure of the problem needs to be present
- If possible, should remain feasible under recombination (or be easy to repair)
- Does not have to be a bit string!

## ■ Population

- Diversity is critical in the initial population
- Normally between 25 and 100 (problem and GA dependent)
- Elitism: Keep the best member or  $x\%$  of the population in each generation.

## ■ Recombination

- Problem specific operators often help.
- Feasibility must be considered!

## ■ Termination Criterion

- One possibility: Measure the average fitness of the population over  $n$  generations ( $n \approx 5$ )

## ■ Mutation

- Is absolutely necessary. Never forget this!

each time or after crossover

# When do GAs work best?

## Question

See above.

When have different parents  $\rightarrow$  diversity  
Reasonable genome representation

# When do GAs work best?

## Question

See above.

- Difficult to answer.
- GAs are good when diversity and randomness are important for finding good solutions.
- Only operates on problems in which the genome "makes sense"

# Literature

# Literature

- PMX Operator: Originally described in Goldberg, David E., and Robert Lingle (1985). "Alleles, loci, and the traveling salesman problem." Proceedings of the first international conference on genetic algorithms and their applications.
- But see also for PMX: Larrañaga, Pedro, et al. (1999) "Genetic algorithms for the travelling salesman problem: A review of representations and operators." Artificial Intelligence Review 13.2: 129-170.

# Thank you for your attention!

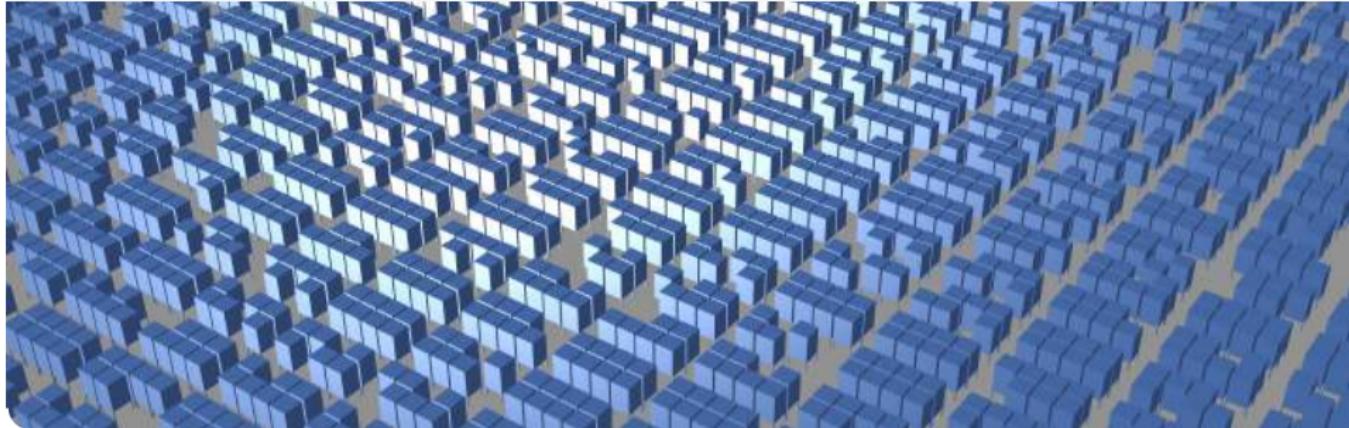
Leuphana University of Lüneburg  
Business information systems, in particular Operations Research  
Prof. Dr. Lin Xie  
Universitätsallee 1  
Building 4, Room 314  
21335 Lüneburg  
Phone +49 4131 677 2305  
Fax +49 4131 677 1749  
[xie@leuphana.de](mailto:xie@leuphana.de)

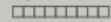
# Analysing Networks with OR-Methods

Part 11 – Matheuristics, Hybrid Metaheuristics

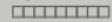
Lin Xie | 29.06.2021

PROF. DR. LIN XIE – BUSINESS INFORMATION SYSTEMS, IN PARTICULAR OPERATIONS RESEARCH

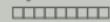




# Matheuristics



# Matheuristic



# Matheuristic

## Math:

- Exact approaches (MIP, LP, etc.)
- Provable optimality
- Lower/upper bounds



# Matheuristic

## Math:

- Exact approaches (MIP, LP, etc.)
- Provable optimality
- Lower/upper bounds

## Heuristic:

- No guarantees
- Good solutions fast



# Matheuristic

## Math:

- Exact approaches (MIP, LP, etc.)
- Provable optimality
- Lower/upper bounds

## Heuristic:

- No guarantees
- Good solutions fast

Matheuristics combine the power of exact solution techniques within a metaheuristic framework.

# Decomposition approaches

## Idea

Decompose a large problem and solve one or more smaller or more constrained MIPs

# Decomposition approaches

## Idea

Decompose a large problem and solve one or more smaller or more constrained MIPs

Decomposition is well suited to exact approaches:

- Exact methods sometimes faster than heuristics for finding good solutions on small problems
- Mathematical modelling usually easier and has less development time than some heuristics (delta evaluation, etc.)

# Fix-and-resolve LNS

Consider **Large Neighborhood Search** for a MIP:

$$\max \mathbf{c}^T \mathbf{x}$$

$$\text{s.t. } \mathbf{Ax} \leq \mathbf{b}$$

$$\mathbf{x} \geq 0$$

# Fix-and-resolve LNS

Consider **Large Neighborhood Search** for a MIP:

$$\max \mathbf{c}^T \mathbf{x}$$

$$\text{s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{b}$$

$$\mathbf{x} \geq 0$$

Combined destroy and repair operator: “fix-and-resolve”:

Given a partial solution to the problem  $\bar{s}$ , fix the corresponding subset of  $x$  variables to the values in  $\bar{s}$ .

$$c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4 \dots$$

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 \dots \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 \dots \leq b_2$$

$$\begin{matrix} \vdots & \vdots \end{matrix}$$

# Fix-and-resolve LNS

Consider **Large Neighborhood Search** for a MIP:

$$\max \mathbf{c}^T \mathbf{x}$$

$$\text{s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{b}$$

$$\mathbf{x} \geq 0$$

Combined destroy and repair operator: “fix-and-resolve”:

Given a partial solution to the problem  $\bar{s}$ , fix the corresponding subset of  $x$  variables to the values in  $\bar{s}$ .

$$\begin{array}{ll}
 c_1x_1 & + c_2x_2 + c_3x_3 + c_4x_4 \dots \\
 a_{11}x_1 & + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 \dots \leq b_1 \\
 a_{21}x_1 & + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 \dots \leq b_2 \\
 & \vdots & \vdots \\
 & & 
 \end{array}$$

$x_2 = \bar{s}_{x_2}$        $x_4 = \bar{s}_{x_4}$

# Decoupling

Isolates parts of a problem and solves them using an exact algorithm.

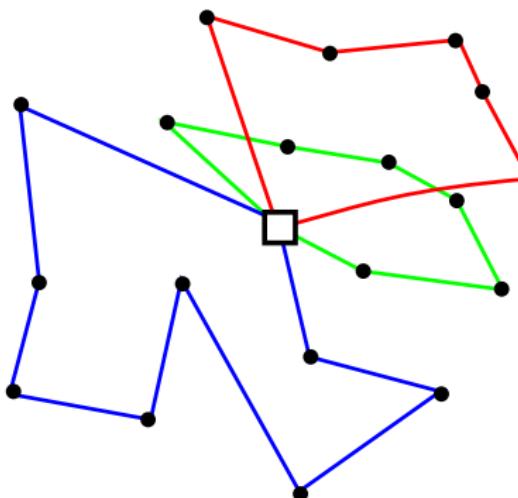
Key difference to fix and resolve: much smaller problems

Idea: Make a self-contained MIP; Links to other parts of the problem included as constants.

# Decoupling

Isolates parts of a problem and solves them using an exact algorithm.

Key difference to fix and resolve: much smaller problems

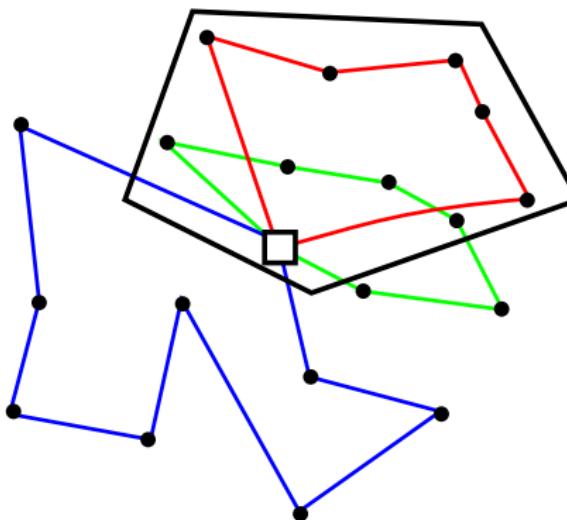


Idea: Make a self-contained MIP; Links to other parts of the problem included as constants.

# Decoupling

Isolates parts of a problem and solves them using an exact algorithm.

Key difference to fix and resolve: much smaller problems



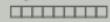
Idea: Make a self-contained MIP; Links to other parts of the problem included as constants.



# Using MIP information

MIPs (and LPs) contain information a heuristic may not have:

- Upper/lower bounds on the solution value
- Dual variable values



# Using MIP information

MIPs (and LPs) contain information a heuristic may not have:

- Upper/lower bounds on the solution value
- Dual variable values

## Usage:

- Determine which heuristic to apply next
- Use within a heuristic (bound branching options, etc.)
- Use duals as a gradient

# Hybrid Metaheuristics

# Characteristics of Metaheuristics

Feature	SA	TS	ILS	VNS/LNS	GA
Population					
Memory					
Multiple neighborhoods					
Solution construction					
Nature-inspired					

+: feature present

⊓: partially present

-: not present

# Hybrid metaheuristics

(Paraphrased) The no free lunch theorem suggests that no single approach is dominant across all instances of a problem

# Tabu Search + Random Walk

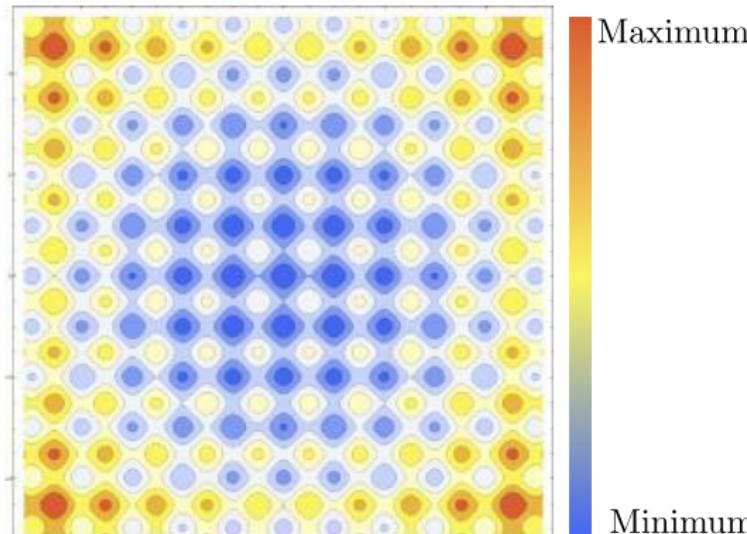
Restart options once a tabu search (or any metaheuristic) terminates:

- Random solution
- ILS perturbation
- Clear tabu list and keep searching

# Tabu Search + Random Walk

Restart options once a tabu search (or any metaheuristic) terminates:

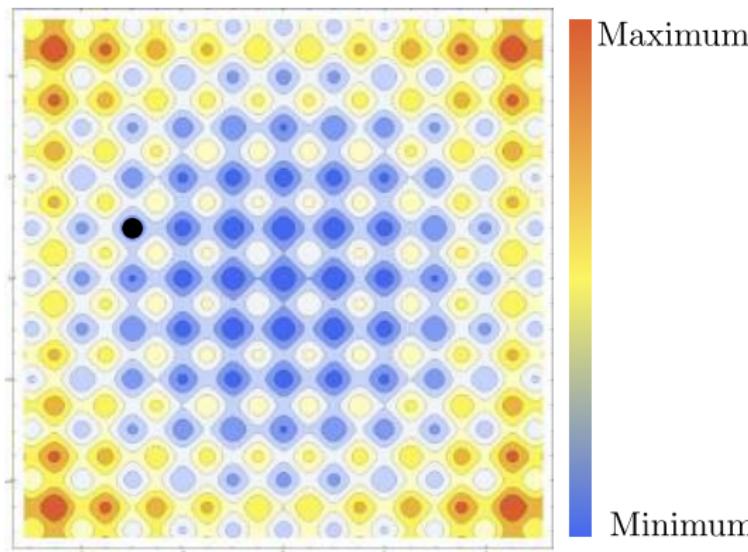
- Random solution
- ILS perturbation
- Clear tabu list and keep searching



# Tabu Search + Random Walk

A random walk is a perturbation based restart strategy:

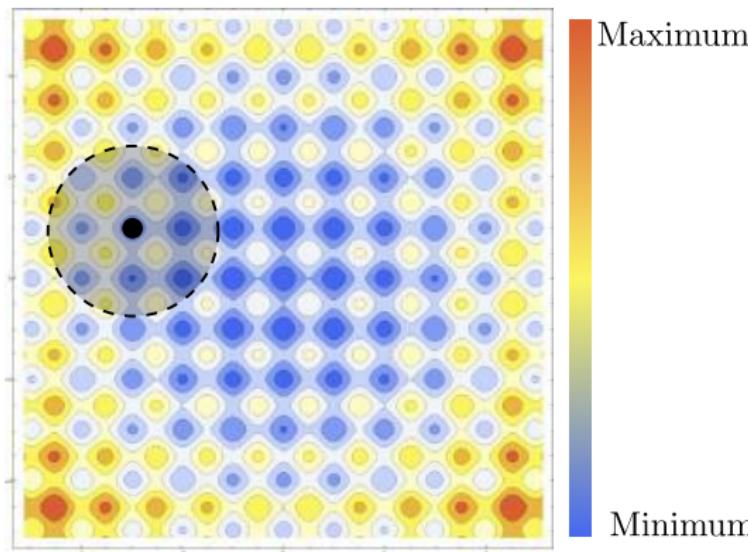
- Moves away from current solution, but not too far
- Problem independent (mostly)
- Problem: Too random?



# Tabu Search + Random Walk

A random walk is a perturbation based restart strategy:

- Moves away from current solution, but not too far
- Problem independent (mostly)
- Problem: Too random?

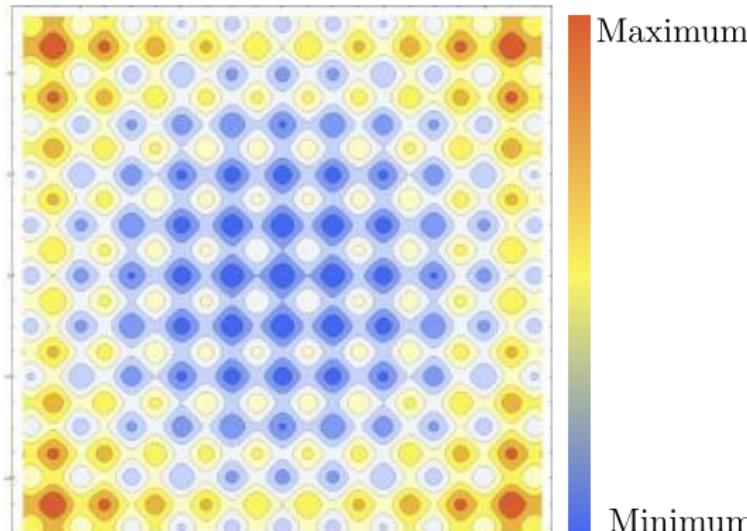


# Tabu Search + Simulated Annealing

Alternative to random walk: Simulated annealing

Introduced in Voß and Fink (2012). Hybridizing Reactive Tabu Search with Simulated Annealing.

- Focuses perturbation on good areas of the search space, but with randomness

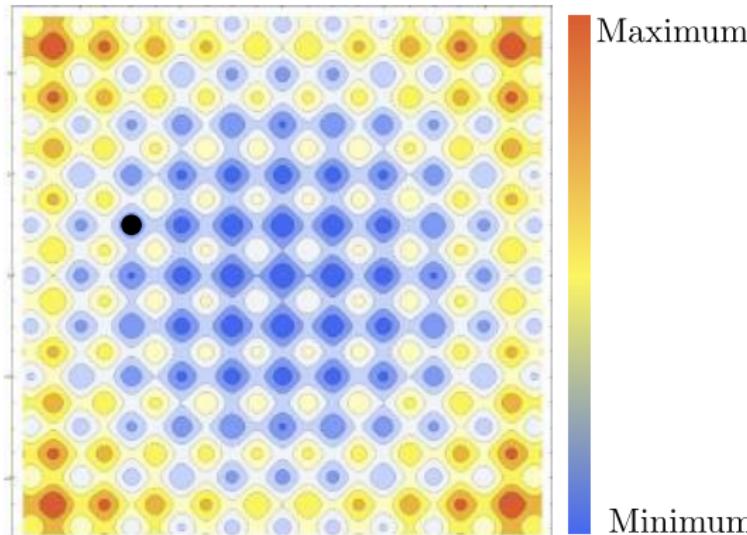


# Tabu Search + Simulated Annealing

Alternative to random walk: Simulated annealing

Introduced in Voß and Fink (2012). Hybridizing Reactive Tabu Search with Simulated Annealing.

- Focuses perturbation on good areas of the search space, but with randomness

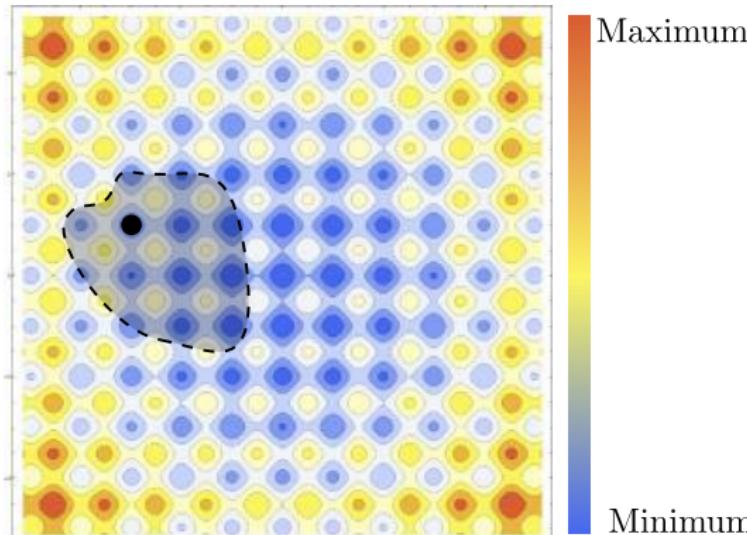


# Tabu Search + Simulated Annealing

Alternative to random walk: Simulated annealing

Introduced in Voß and Fink (2012). Hybridizing Reactive Tabu Search with Simulated Annealing.

- Focuses perturbation on good areas of the search space, but with randomness



# Genetic algorithms with local intensification: Memetic algorithms

## Problems with GAs:

- Sometimes too much diversity; search stagnates
- Large neighborhoods; nearby good solutions may be missed

# Genetic algorithms with local intensification: Memetic algorithms

## Problems with GAs:

- Sometimes too much diversity; search stagnates
- Large neighborhoods; nearby good solutions may be missed

**Solution:** Use improvement procedures on a selected subset of the GA population

# Genetic algorithms with local intensification: Memetic algorithms

## Problems with GAs:

- Sometimes too much diversity; search stagnates
- Large neighborhoods; nearby good solutions may be missed

**Solution:** Use improvement procedures on a selected subset of the GA population

**Example:** Iterative improvement until convergence on the best 5% of a GA population

# Memetic algorithms?

- Broad topic with lots of details; has not caught on widely in OR.
- Read about them on your own if you are interested

# Exercise

## Task

Design a hybrid metaheuristic using metaheuristics we've talked about in the course. Argue why the hybrid might offer better performance than its individual components.

# Literature

- V. Maniezzo, T. Stützle, S. Voß (2009). Matheuristics: Hybridizing Metaheuristics and Mathematical Programming
- R. Bent and P. Van Hentenryck (2007). Randomized Adaptive Spatial Decoupling For Large-Scale Vehicle Routing with Time Windows.

# Thank you for your attention!

Leuphana University of Lüneburg  
Business information systems, in particular Operations Research  
Prof. Dr. Lin Xie  
Universitätsallee 1  
Building 4, Room 314  
21335 Lüneburg  
Phone +49 4131 677 2305  
Fax +49 4131 677 1749  
[xie@leuphana.de](mailto:xie@leuphana.de)