CS616 - Optimization Algorithms

Chapter 1 - Introduction to Optimization

Dr. Mahdi Khemakhem

Department of Computer Science
College of Computer Engineering and Science
Prince Sattam bin Abdulaziz University

AY - 2025/2026

Outline

1. Optimization Problems
1.1 Definitions
1.2 Components
1.3 OP Requirements and Solving
1.4 Types of Optimization Problems
1.5 Application Areas
1.6 Key Takeaways

2. Linear Programming Model
2.1 Definitions
2.2 Characteristics
2.3 General Forms
2.4 First Simple LP Model
2.5 Linear Programming Theorem
2.6 Some Tips for Formulating LP Models
2.7 Key Takeaways

3. Examples of LP Problems
3.1 Scheduling Problem
3.2 Assignment Problem
3.3 Knapsack Problem

Outline (Contd.)

3.4 Bin Packing Problem
3.5 Multi-dimensional Bin Packing Problem
3.6 Traveling Salesman Problem
3.7 Key Takeaways

4. Optimization-Based Decision Making
4.1 Decision Making Steps
4.2 Classical Optimization Models
4.3 Different Families of Optimization Models
4.4 Key Takeaways

5. Complexity Theory
5.1 Complexity of Algorithms
5.2 Complexity of Problems
5.3 Key Takeaways

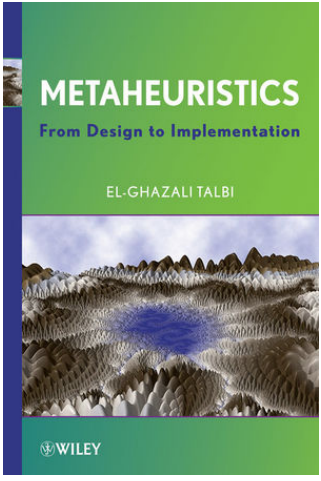
6. Optimization Methods
6.1 Exact Methods
6.2 Approximate Algorithms
6.3 Heuristics
6.4 Metaheuristics

Outline (Contd.)

6.5 Key Takeaways

Materials

- **Textbook:** "Metaheuristics: From Design to Implementation" by *El-Ghazali Talbi*
- $\Rightarrow$ **Read Chapter 1 (Sections 1.1, 1.2, and 1.3) for this chapter's material.**



Outline

1. Optimization Problems
1.1 Definitions
1.2 Components
1.3 OP Requirements and Solving
1.4 Types of Optimization Problems
1.5 Application Areas
1.6 Key Takeaways

2. Linear Programming Model

3. Examples of LP Problems

4. Optimization-Based Decision Making

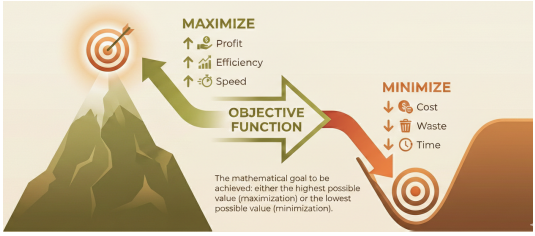
5. Complexity Theory

6. Optimization Methods

What is an Optimization Problem?

- An *optimization problem (OP)* is the problem of **finding the best solution** from all **feasible solutions**.
- An **OP** consists of **maximizing** or **minimizing** an **objective function** by **selecting** values of **decision variable(s)** while satisfying a set of **constraints**.
- Solving an **OP** consists of **finding** the **optimal solution(s)** (**best solution(s)**) in the **feasible region** according to the defined **objective function** and the **constraint(s)** set **limitations**.

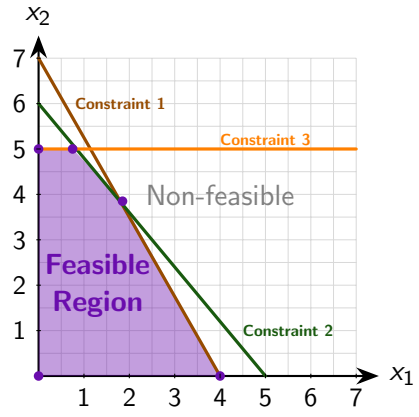
What is an Objective Function?



- An *objective function* is a **mathematical function**, defined in terms of **decision variable(s)**, that **needs to be optimized** (**maximized** or **minimized**).
- The **objective function** **quantifies** how **good** a **solution** is.
- The **objective function** **guides the search process** toward the **best solution(s)** in the **feasible region**.

What is a Feasible Region?

- The *feasible region* is the **set of all possible solutions** that **satisfy** the problem **constraints**.
- The **feasible region** is **defined** by the **constraints** of the **OP**.
- The **optimal solution(s)** to the **OP** **lie(s) within** the **feasible region**.



What are Constraints?

- *Constraints* are **limitations** or **restrictions** that **define** the **feasible region** of an **OP**.
- **Constraints** are **expressed** as **mathematical equations or inequalities** involving the **decision variable(s)**.

What is a Decision Variable?

- *Decision variable(s)* are the **unknowns** that **need to be determined** in order to **optimize** the **objective function**.
- **Decision variable(s)** represent the **choices** or **actions** that can be taken to influence the outcome of the **OP**.
- **Decision variable(s)** can be **continuous** (taking any value within a range) or **discrete** (taking specific values, such as integers).
- The **values** of the **decision variable(s)** **directly affect** the **objective function**.

Optimization Problem Requirements

Optimization Problem Requirements

An **optimization problem** definition **requires** prior definitions of:

1. A set of **decision variable(s)**.
2. An **objective function** to be **maximized** or **minimized** expressed using the decision variables.
3. A set of **constraints** expressed using the decision variables that **define the feasible region**.

Optimal Solution and Problem Solving

Optimal Solution Definition

- An *optimal solution* is a **solution** that **yields the best possible value** of the **objective function** while **satisfying all constraints**.
- The goal of solving an **optimization problem** is to **find** the **optimal solution(s)** within the **feasible region**.

Optimization Problem Solving

Solving an **optimization problem** consists of **finding** the **best solution(s)** in the **feasible region** according to the defined **objective function** and the **constraint(s)** set **limitations**.

Types of Optimization Problems (1/2)

Classification of Optimization Problems

Optimization problems can be classified into various types based on the nature of the **objective function**, **constraints**, and **decision variables**.

- Based on the **number of objective functions**, **optimization problems** can be categorized as:
  - **Single-Objective Optimization**: Involves optimizing a single **objective function**.
  - **Multi-Objective Optimization**: Involves optimizing multiple **objective functions** simultaneously, often requiring trade-offs between conflicting objectives.
- Based on the **nature of objective function and constraints**, **optimization problems** can be categorized as:
  - **Linear Programming (LP)**: **Objective function** and **constraints** are expressed as **linear equations or inequalities**.
  - **Non-Linear Programming (NLP)**: **Objective function** and/or **constraints** are expressed as **non-linear equations or inequalities**.

Types of Optimization Problems (2/2)

- Based on the **nature of decision variables**, **optimization problems** can be categorized as:
  - **Continuous Programming (CP)**: **Decision variables** can take any **real values** within a specified range.
  - **Integer Programming (IP)**: **Decision variables** are **restricted to integer values** only.
  - **Binary Programming (BP)**: **Decision variables** can only take **binary values** (0 or 1).
  - **Mixed-Integer Programming (MIP)**: **Decision variables** include both **continuous**, **integer** variables, and/or **binary** variables.

Focus of the Course

In this course, we will mainly focus on **Single Objective**, **Linear Programming (LP)** problems and their variants such as **CP**, **IP**, **BP**, and **MIP**, as they form the foundation for many optimization techniques and algorithms.

Application Areas of Optimization Problems

- **Optimization problems** are widely used in various fields and industries to improve decision-making and resource allocation.
- Some common application areas include:
  - **Computer Science and Artificial Intelligence**: Resource allocation, scheduling, and machine learning model optimization.
  - **Transportation and Logistics**: Route optimization, vehicle scheduling, and fleet management.
  - **Education**: Curriculum design, resource allocation, and scheduling.
  - **Supply Chain Management**: Optimizing inventory levels, transportation routes, and production schedules.
  - **Finance, Marketing, and Economics**: Portfolio optimization, risk management, and pricing strategies.
  - **Agriculture**: Crop planning, resource allocation, and supply chain optimization.
  - **Construction and Project Management**: Resource allocation, scheduling, and cost minimization.
  - **Manufacturing**: Production planning, scheduling, and quality control.
  - **Energy**: Optimal power generation, distribution, and consumption, as well as renewable energy integration.
  - **Telecommunications**: Network design, routing, and bandwidth allocation.
  - **Healthcare**: Resource allocation, treatment planning, and scheduling.

Key Takeaways

Key Takeaways

- An **optimization problem** involves finding the **best solution** from all **feasible solutions** by **optimizing** an **objective function** while satisfying a set of **constraints**.
- The main components of an **optimization problem** are **decision variable(s)**, **objective function**, **constraints**, and the **feasible region**.
- **Optimization problems** can be classified based on the number of **objective functions**, the nature of the **objective function and constraints**, and the nature of the **decision variables**.
- This course will primarily focus on **Single Objective**, **Linear Programming (LP)** problems and their variants.

Outline

1. Optimization Problems

2. Linear Programming Model
2.1 Definitions
2.2 Characteristics
2.3 General Forms
2.4 First Simple LP Model
2.5 Linear Programming Theorem
2.6 Some Tips for Formulating LP Models
2.7 Key Takeaways

3. Examples of LP Problems

4. Optimization-Based Decision Making

5. Complexity Theory

Linear Programming Model

Definition

A **Linear Programming (LP) model** is a **mathematical model** that **represents** an **optimization problem** (**minimization** or **maximization**) where both the **objective function** and the **constraints** are **linear**.

- The **basic components** of an **LP model** include:
  - **Data** (or **Parameters**)
  - **Decision Variables**
  - **Constraints**
  - **Objective Function(s)**
- **LP model** is **a commonly used model** in **mathematical programming**.
- **Constraints** may be **equalities** or **inequalities**.
- **Decision variables** are **non-negative** and can be **continuous** and/or **discrete** (integers or binary).

LP Model Characteristics (1/2)

The **Linear Programming (LP) model** has the following key characteristics:

1. **Objective Function**
   - An **LP model** must have a single **objective function** that needs to be **maximized** or **minimized**.
   - The **objective function** is expressed as a **linear combination** of the **decision variables**.
2. **Decision Variables**
   - An **LP model** must have one or more **decision variables** that represent the choices to be made.
   - The **decision variables** can be **continuous** (taking any value within a range) or **discrete** (taking specific values, such as integers).
3. **Constraints**
   - An **LP model** must have one or more **constraints** that define the **feasible region**.
   - The **constraints** are expressed as **linear equations or inequalities** involving the **decision variables**.
4. **Linearity**
   - Both the **objective function** and the **constraints** must be **linear**, meaning that they can only involve **addition**, **subtraction**, and **multiplication by constants**.
   - No **non-linear terms** (e.g., products of decision variables, exponents, logarithms) are allowed.

LP Model Characteristics (2/2)

5 Non-Negativity

- In many **LP models**, the **decision variables** are required to be **non-negative** (i.e., greater than or equal to zero).
- This reflects real-world scenarios where negative values may not make sense (e.g., quantities of products, resources).

6 Deterministic Nature

- **LP models** are typically **deterministic**, meaning that all parameters (coefficients in the objective function and constraints) are known with certainty.
- There is no consideration of uncertainty or randomness in the model.

7 Finiteness

- An **LP model** has a **finite number** of **decision variables** and **constraints**, making it computationally manageable.
- This allows for the application of various **solution techniques** to find the optimal solution efficiently.

LP Expanded Form

A Linear Programming model, with n **decision variables** and m **constraints**, can be **expandedly** formulated as follows:

$$
\begin{aligned}
&\text{Min (or Max) } Z = \sum_{i=1}^{n} c_i x_i \\
&\text{subject to (s.t)} \\
&\begin{cases}
\sum_{i=1}^{n} a_{1i} x_i \leq (\text{or } \geq \text{ or } =) b_1 \\
\sum_{i=1}^{n} a_{2i} x_i \leq (\text{or } \geq \text{ or } =) b_2 \\
\vdots \qquad\qquad \vdots \\
\sum_{i=1}^{n} a_{ji} x_i \leq (\text{or } \geq \text{ or } =) b_j \\
\vdots \qquad\qquad \vdots \\
\sum_{i=1}^{n} a_{mi} x_i \leq (\text{or } \geq \text{ or } =) b_m \\
\forall i \in \{1, 2, \ldots, n\}, x_i \in \mathbb{R}^+ (\text{or } \mathbb{N} \text{ or } \{0, 1\})
\end{cases}
\end{aligned}
$$

Where:

- Z: *Objective function* value.
- x_i: *Decision variable* i, for $i = 1, 2, \ldots, n$ that can be *continuous* (in $\mathbb{R}^+$), *integer* (in $\mathbb{N}$), or *binary* (in $\{0, 1\}$).
- c_i: *Coefficient* of *decision variable* x_i in the *objective function*.
- a_{ji}: *Coefficient* of *decision variable* x_i in *constraint* j, for $j = 1, 2, \ldots, m$.
- b_j: *Right-hand side* constant of *constraint* j.

LP Compact/Matrix Form

A Linear Programming model, with n **decision variables** and m **constraints**, can be **compactly** formulated as follows:

$$\begin{array}{l}\text{Min (or Max) } Z = c \cdot x \\ \text{subject to (s.t)} \\ \begin{cases} A \cdot x \leq b \\ \qquad \geq \\ \qquad = \\ \forall x_i \in x, x_i \in \mathbb{R}^+ (\text{or } \mathbb{N} \text{ or } \{0,1\}) \end{cases}\end{array}$$

with:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \; c = [c_1, c_2, \ldots, c_n], \; b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \; A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

where x is a *vector* of n *continuous* (or *integer*, or *binary*, or *mixed*) *decision variables*, and c and b (resp. A) are *coefficient vectors* (resp. *matrix*) of *coefficients*.

First Simple LP Model (1/3)

Problem statement: A manufacturing company produces two products, $Prod_1$ and $Prod_2$, using two raw materials, M_1 and M_2. The company aims to determine the optimal production quantities (number of tons) of each product to maximize its profit while adhering to the availability constraints of the raw materials.

The following data is provided:

- Each unit of $Prod_1$ requires 6 units of M_1 and 1 unit of M_2.
- Each unit of $Prod_2$ requires 4 units of M_1 and 2 units of M_2.
- The profit obtained from selling one unit of $Prod_1$ is \$5, while the profit from selling one unit of $Prod_2$ is \$4.
- The available quantities of raw materials are 24 units of M_1 and 6 units of M_2.

Data summary:

Raw Material	Usage for $Prod_1$	Usage for $Prod_2$	*Material Availability*
M_1	6	4	**24**
M_2	1	2	**6**
Profit	**\$5**	**\$4**	

First Simple LP Model (2/3)

To formulate the **LP model** for this problem, **we need to define the following components**:

- **Decision variables:**
  - Let X_1 be the number of units produced of $Prod_1$.
  - Let X_2 be the number of units produced of $Prod_2$.
- **Objective function:** Maximize the total profit Z from producing $Prod_1$ and $Prod_2$, given by $\mathbf{Max} Z = 5X_1 + 4X_2$
- **Constraints:**
  1. M_1 availability constraint: $6X_1 + 4X_2 \leq 24$
  2. M_2 availability constraint: $1X_1 + 2X_2 \leq 6$
  3. Non-negativity constraints: $X_1, X_2 \geq 0$

Remarks

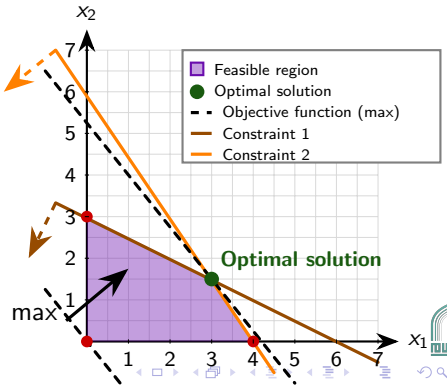
- The provided **LP** is a **continuous maximization** problem.
- **Feasible region** is defined by the **intersection** of all three **constraints**.
- **Optimal solution belongs** to the **feasible region** (**satisfies all constraints**) and **provides the highest** (**maximum**) **objective function** value.

First Simple LP Model (3/3)

The **LP model** for the given problem can be **formulated** as follows:

$$\text{Max } Z = 5X_1 + 4X_2$$
$$\text{(s.t)}$$
$$\begin{cases} 6X_1 + 4X_2 \leq 24 \\ 1X_1 + 2X_2 \leq 6 \\ X_1, X_2 \in \mathbb{R}^+ \end{cases}$$

$$x = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix},\ c = [5, 4],\ b = \begin{bmatrix} 24 \\ 6 \end{bmatrix},$$
$$A = \begin{bmatrix} 6 & 4 \\ 1 & 2 \end{bmatrix}$$



- The figure on the right illustrates the **graphical interpretation of the model**.
- Each **constraint** can be **represented by a line**.
- The **objective function** is an **infinity of parallel lines** that **move** in the direction of **maximization**.
- The **optimum solution** will **always lie** at one of the *extreme points* (**corner points of the feasible polygon**).
- The **optimal solution** is $(X_1 = 3, X_2 = 1.5)$ with a **maximum profit** of $Z = 21$.

Linear Programming Theorem

LP Theorem

If an **LP problem** has an **optimal solution**, then at least one **extreme point** (corner point) of the **feasible region** is an **optimal solution**.

- An **extreme point** is a **vertex** of the **feasible region** formed by the intersection of the **constraint lines**.
- The **feasible region** is a **convex polygon** in the case of two decision variables.
- The **optimal solution** can be found by one of the following methods:
  - Evaluating the **objective function** at each **extreme point** of the **feasible region**
  - Translating the **objective function** line (**in the direction of optimization**) until it reaches the **last extreme point** (or **the line joining the last two extreme points**) in the **feasible region**.
- If multiple **extreme points** yield the same optimal value, then the **LP problem** has **multiple optimal solutions**.

Some Tips for Formulating LP Models

- **Understand the problem:** Carefully read and analyze the problem statement to identify the key components, such as parameters, decision variables, objective function, and constraints. Then summarize the data provided.
- **Define decision variables:** To define decision variables you should ask yourself what are the unknowns you need to determine in order to solve the problem. Clearly define each decision variable and its domain (continuous, integer, binary).
- **Formulate the objective function:** Express the objective function in terms of the corresponding parameters and decision variables, ensuring it reflects the goal of the problem (maximization or minimization).
- **Identify constraints:** Determine the constraints that limit the feasible solutions, and express them as linear equations or inequalities involving the parameters and decision variables.
- **Check for linearity:** Ensure that both the objective function and constraints are linear, avoiding any non-linear terms.
- **Include non-negativity constraints:** If applicable, include non-negativity constraints for the decision variables to reflect real-world scenarios.
- **Review and validate:** Double-check the formulation for accuracy and completeness, ensuring that all aspects of the problem are represented correctly.

Key Takeaways

Key Takeaways

- A **Linear Programming model** is a mathematical representation of an optimization problem where both the **objective function** and **constraints** are **linear**.
- **LP models** consist of **decision variables**, **constraints**, and an **objective function** to be optimized (maximized or minimized).
- The **LP Theorem** states that if an optimal solution exists, it will be found at an **extreme point** (corner point) of the **feasible region**.
- **LP models** can be formulated in **expanded form** or **compact/matrix form**, depending on the complexity and size of the problem.

Outline

1. Optimization Problems

2. Linear Programming Model

3. Examples of LP Problems
3.1 Scheduling Problem
3.2 Assignment Problem
3.3 Knapsack Problem
3.4 Bin Packing Problem
3.5 Multi-dimensional Bin Packing Problem
3.6 Traveling Salesman Problem
3.7 Key Takeaways

4. Optimization-Based Decision Making

5. Complexity Theory

Scheduling Problem - Employees Shift Planning (1/2)

Problem statement: A hospital service director is responsible for organizing the nurses' schedules. A working day is divided into 12 time slots of two hours each. Staffing needs vary from one time slot to another. For example, few nurses are needed during the night, but the staff must be reinforced in the morning to ensure patient care. The table beside gives the staffing needs for each of the time slots. What is the minimum number of nurses needed to cover time slot requirements? It's important to know that each nurse works 8 hours a day and has a 2-hours break after 4 hours of work.

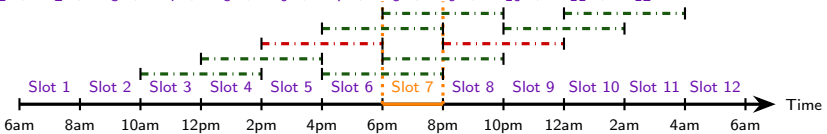
Data summary:

- Each nurse works for 8 hours (4 time slots) and takes a 2-hour break after 4 hours of work.
- The staffing needs for each time slot are provided in the table beside.

#	Time slot	Nurses min. num.
1	06 am - 08 am	35
2	08 am - 10 am	40
3	10 am - 12 pm	40
4	12 pm - 02 pm	35
5	02 pm - 04 pm	30
6	04 pm - 06 pm	30
7	06 pm - 08 pm	35
8	08 pm - 10 pm	30
9	10 pm - 12 am	20
10	12 am - 02 am	15
11	02 am - 04 am	15
12	04 am - 06 am	15

Scheduling - Employees Shift Planning (2/2)

- **Decision variables:** Unknown variables that determine the number of nurses starting their working shift at the beginning of each time slot. $X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9, X_{10}, X_{11}, X_{12} \in \mathbb{N}$
- **Objective function:** Minimize the total number of nurses.
  $MinZ = X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + X_8 + X_9 + X_{10} + X_{11} + X_{12}$



- **Constraints:**
  1. Time slot #7 requirement: Consider nurses that start their shift at 10am, at 12pm, at 4pm, and 6pm. Nurses that start their shift at 2pm are not considered because they are on break during slot #7. $X_7 + X_6 + X_4 + X_3 \geq 35$
  2. Remaining time slots #1, 2, 3, 4, 5, 6, 8, 9, 10, 11, and 12 requirements:
     $X_1 + X_{12} + X_{10} + X_9 \geq 35$, $X_2 + X_1 + X_{11} + X_{10} \geq 40$, $X_3 + X_2 + X_{12} + X_{11} \geq 40$,
     $X_4 + X_3 + X_1 + X_{12} \geq 35$, $X_5 + X_4 + X_2 + X_1 \geq 30$, $X_6 + X_5 + X_3 + X_2 \geq 30$, $X_8 + X_7 + X_5 + X_4 \geq 30$,
     $X_9 + X_8 + X_6 + X_5 \geq 20$, $X_{10} + X_9 + X_7 + X_6 \geq 15$, $X_{11} + X_{10} + X_8 + X_7 \geq 15$, $X_{12} + X_{11} + X_9 + X_8 \geq 15$

Assignment Problem - Military Operations (1/2)

Problem statement: During a military operation, an officer must choose the members of a patrol of minimum size among the different soldiers whose skills are described in the table below:

Soldier	Radio	Camouflage	Sniper	Physical endurance	Non-commissioned officer
A	✓	✓	6	5	✓
B		✓	8	7	
C	✓	✓	7	7	
D	✓		4	9	
E		✓	7	9	✓
F	✓		8	6	

The officer must respect the following conditions:

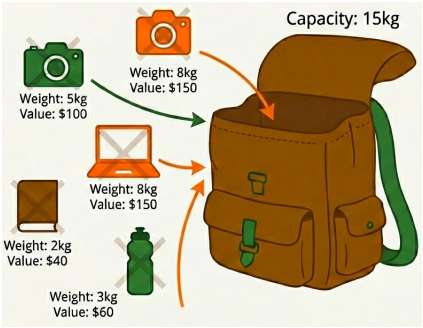
- *One and only one* non-commissioned officer is needed.
- *At least 1* Radio operator and *at least 1* Camouflage specialist are required.
- *At least two* radio operators will *not be part of the patrol* in order to remain available for other missions.
- Patrol members *must have an average sniping rate of at least 7* and *an average physical endurance rate of at least 7.5*.
- If soldier *A is part of the patrol*, then *neither B nor C should be part of it*.

Assignment Problem - Military Operations (2/2)

- **Decision variables:** Unknown binary variables that determine for each soldier (from A,B,...,F) if he will be a part of the patrol or not. $X_A, X_B, X_C, X_D, X_E, X_F \in \{0, 1\}$
- **Objective function:** Minimize the total number of soldiers in the patrol.
  $MinZ = X_A + X_B + X_C + X_D + X_E + X_F$
- **Constraints:**
  1. NCO requirement: One and only one NCO is needed. $X_A + X_E = 1$
  2. Radio requirement: At least 1 Radio operator. $X_A + X_C + X_D + X_F \geq 1$
  3. Radio requirement: At least two radio operators will not be part of the patrol $\rightarrow$ *At most two radio operators will be a part of the patrol*. $X_A + X_C + X_D + X_F \leq 2$
  4. Camouflage requirement: At least 1 Camouflage specialist. $X_A + X_B + X_C + X_E \geq 1$
  5. Sniping rate requirement: At least 7 of *average* sniping rate. $\frac{6X_A+8X_B+7X_C+4X_D+7X_E+8X_F}{X_A+X_B+X_C+X_D+X_E+X_F} \geq 7 \Rightarrow -1X_A + 1X_B + 0X_C - 3X_D + 0X_E + 1X_F \geq 0 \Rightarrow X_B - X_A - 3X_D + X_F \geq 0$
  6. Physical endurance rate requirement: At least 7.5 of *average* physical endurance rate. $\frac{5X_A+7X_B+7X_C+9X_D+9X_E+6X_F}{X_A+X_B+X_C+X_D+X_E+X_F} \geq 7.5 \Rightarrow -2.5X_A + 0.5X_B + 0.5X_C + 1.5X_D + 1.5X_E - 1.5X_F \geq 0$
  7. Co-existence requirement : If soldier *A is part of the patrol*, then *neither B nor C should be part of it*. $2X_A + X_B + X_C \leq 2$

Knapsack Problem (1/3)

The **Knapsack Problem (KP)** is a classic combinatorial optimization problem that can be described as follows: given a set of n objects, each with a weight w_i and a profit p_i, determine the subset of objects to include in a knapsack of capacity C such that the total profit is maximized without exceeding the knapsack's weight limit. Each object can either be included in the knapsack or not, leading to a binary decision for each object.



Knapsack Problem (2/3)

Decision variables: Binary variables $x_i \in \{0,1\}, \forall i \in \{1,2,\ldots,n\}$ indicating whether object i is included in the knapsack ($x_i = 1$) or not ($x_i = 0$).

Objective function: Maximize the total profit of the selected objects in the knapsack.

$MaxZ = \sum_{i=1}^{n} p_i x_i$

Constraints: The total weight of the selected objects must not exceed the knapsack capacity C.

$\sum_{i=1}^{n} w_i x_i \leq C$

General formulation of the 0-1 Knapsack Problem

$$\text{Max } Z = \sum_{i=1}^{n} p_i x_i$$

$$\text{(s.t)} \begin{cases} \sum_{i=1}^{n} w_i x_i \leq C \\ \forall i \in \{1,2,\ldots,n\}, x_i \in \{0,1\} \end{cases}$$

Model Analysis

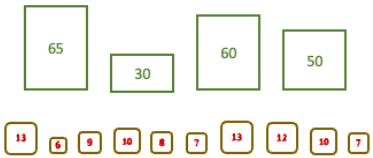
The model is a **Binary Integer Linear Programming (BILP)** with n **variables** and **1 constraint** (plus the binary constraints).

Knapsack Problem (3/3)

- The size of search space is 2^n, where n is the number of objects.
- The Knapsack Problem is **NP-hard**, meaning that there is no known polynomial-time algorithm to solve it optimally big instances. It can be encountered in various fields such as resource allocation, budget management, and logistics.
- Unfortunately, **enumerating all possible** subsets of objects to find the optimal solution is computationally infeasible for large n.
- The table shows the combinatorial explosion of the search space as the number of objects increases (*consider that each possibility is evaluated in* 10^{-6} *seconds*):

Number of objects (n)	Number of possible subsets	Approx. Time to enumerate (seconds)
10	1,024	0.001024
20	1,048,576	1.048576
30	1,073,741,824	1,073.741824
40	1,099,511,627,776	1,099,511.627776 = 12.7 days
50	1,125,899,906,842,624	1,125,899,906.842624 = 35.7 years

Bin Packing Problem (1/4)



Bin	Capacity
1	65
2	30
3	60
4	50

Item	Volume
1	13
2	6
3	9
4	10
5	8
6	7
7	13
8	12
9	10
10	7

The **Bin Packing Problem (BPP)** is a classic combinatorial optimization problem that can be described as follows: given a set of n items, each with a volume v_i, and a set of m bins, each with a maximum capacity c_i, determine the minimum number of bins required to pack all items such that the total volume of items in each bin does not exceed its capacity. Each item must be placed in exactly one bin. The tables beside illustrate an example of the Bin Packing Problem with $n = 10$ items and $m = 4$ bins.

Bin Packing Problem (2/4)

Specific formulation of the Bin Packing Problem

Decision variables:

- Binary variables $x_{ij} \in \{0,1\}, \forall i \in \{1,2,\ldots,10\}, \forall j \in \{1,2,3,4\}$ indicating whether item i is placed in bin j ($x_{ij} = 1$) or not ($x_{ij} = 0$).
- Binary variables $y_j \in \{0,1\}, \forall j \in \{1,2,3,4\}$ indicating whether bin j is used ($y_j = 1$) or not ($y_j = 0$).

Objective function: Minimize the total number of bins used to pack all items.
$MinZ = y_1 + y_2 + y_3 + y_4 = \sum_{j=1}^{4} y_j$.

Constraints:

1. Each item must be placed in exactly one bin. $\sum_{j=1}^{4} x_{ij} = 1, \forall i \in \{1,2,\ldots,10\}$
   - E.g., item 1: $x_{11} + x_{12} + x_{13} + x_{14} = 1$
   - E.g., item 2: $x_{21} + x_{22} + x_{23} + x_{24} = 1$
   - ...
   - E.g., item 10: $x_{10,1} + x_{10,2} + x_{10,3} + x_{10,4} = 1$
2. The total volume of items in each bin must not exceed its capacity. $\sum_{i=1}^{10} v_i x_{ij} \leq c_j y_j, \forall j \in \{1,2,3,4\}$
   - E.g., bin 1: $13x_{11} + 6x_{21} + 9x_{31} + 10x_{41} + 8x_{51} + 7x_{61} + 13x_{71} + 12x_{81} + 10x_{91} + 7x_{10,1} \leq 65y_1$
   - E.g., bin 2: $13x_{12} + 6x_{22} + 9x_{32} + 10x_{42} + 8x_{52} + 7x_{62} + 13x_{72} + 12x_{82} + 10x_{92} + 7x_{10,2} \leq 30y_2$
   - E.g., bin 3: $13x_{13} + 6x_{23} + 9x_{33} + 10x_{43} + 8x_{53} + 7x_{63} + 13x_{73} + 12x_{83} + 10x_{93} + 7x_{10,3} \leq 60y_3$
   - E.g., bin 4: $13x_{14} + 6x_{24} + 9x_{34} + 10x_{44} + 8x_{54} + 7x_{64} + 13x_{74} + 12x_{84} + 10x_{94} + 7x_{10,4} \leq 50y_4$

Bin Packing Problem (3/4)

General formulation of the Bin Packing Problem

Decision variables:

- Binary variables $x_{ij} \in \{0,1\}, \forall i \in \{1,2,\ldots,n\}, \forall j \in \{1,2,\ldots,m\}$ indicating whether item i is placed in bin j ($x_{ij}=1$) or not ($x_{ij}=0$).
- Binary variables $y_j \in \{0,1\}, \forall j \in \{1,2,\ldots,m\}$ indicating whether bin j is used ($y_j=1$) or not ($y_j=0$).

Objective function: Minimize the total number of bins used to pack all items. $MinZ = \sum_{j=1}^{m} y_j$.

Constraints:

1. Each item must be placed in exactly one bin. $\sum_{j=1}^{m} x_{ij} = 1, \forall i \in \{1,2,\ldots,n\}$
2. The total volume of items in each bin must not exceed its capacity. $\sum_{i=1}^{n} v_i x_{ij} \leq c_j y_j, \forall j \in \{1,2,\ldots,m\}$

$$\text{Min } Z = \sum_{j=1}^{m} y_j$$

$$\text{(s.t)} \begin{cases} \sum_{j=1}^{m} x_{ij} = 1, \forall i \in \{1,2,\ldots,n\} \\ \sum_{i=1}^{n} v_i x_{ij} \leq c_j y_j, \forall j \in \{1,2,\ldots,m\} \\ \forall i \in \{1,2,\ldots,n\}, x_{ij} \in \{0,1\}, \forall j \in \{1,2,\ldots,m\} \\ \forall j \in \{1,2,\ldots,m\}, y_j \in \{0,1\} \end{cases}$$

Model Analysis

The model is a **Binary Integer Linear Programming (BILP)** with $n \times m + m$ **variables** and $n+m$ **constraints** (plus the binary constraints).

Bin Packing Problem (4/4)

- The size of search space is $2^{n \times m}$, where n is the number of items and m is the number of bins.
- The Bin Packing Problem is **NP-hard** encountered in various real-world applications, such as resource allocation, logistics, and manufacturing.
- Unfortunately, **enumerating all possible** arrangements of items in bins to find the optimal solution is computationally infeasible for large n and m.
- The table shows the combinatorial explosion of the search space as the number of items and bins increases (*consider that each possibility is evaluated in* 10^{-6} *seconds*):

Number of items (n)	Number of bins (m)	Approx. Time to enumerate (seconds)
5	3	34.9
10	4	$1.0995 \times 10^{12} = 34,844$ years
15	5	$3.2768 \times 10^{30} = 1.04 \times 10^{23}$ years
20	6	$9.2234 \times 10^{48} = 2.92 \times 10^{41}$ years

Multi-dimensional BPP - Cloud Computing Application (1/6)

Cloud Computing: On-demand access to shared computing resources via the internet.

Key Challenge: Efficient allocation of **Virtual Machines (VMs)** to **Physical Machines (PMs)** in data centers.

Model: Multi-dimensional Bin Packing Problem (MBPP)

- **Items (VMs):** Have multiple resource demands (CPU, RAM, etc.)
- **Bins (PMs):** Have limited capacities for each resource (CPU, RAM, etc.)
- **Goal:** Minimize number of active PMs

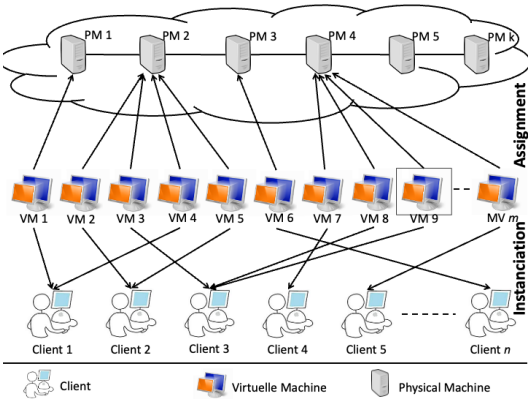
Multi-dimensional BPP - Cloud Computing Application (2/6)

Scenario: Multiple clients request cloud services requiring VMs deployed on PMs in a data center.

Constraint: Each PM has **limited resources** (CPU, RAM) that must be managed efficiently.

Example from figure beside:

- **Client 1:** Needs 2 VMs
  - $VM1 \rightarrow PM1$
  - $VM4 \rightarrow PM2$
- **Client 3:** Needs 3 VMs
  - $VM3 \rightarrow PM2$
  - $VM8, VM9 \rightarrow PM4$



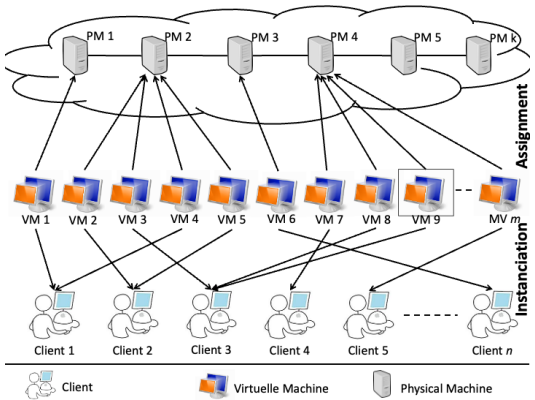
Multi-dimensional BPP - Cloud Computing Application (3/6)

Problem: VMs/PMs Deployement Problem **VMPMDP** fits the MBPP framework (**NP-hard**)

Goal: Deploy VMs onto PMs to **minimize active PMs** while satisfying resource constraints.

Given: Number of VMs and resource demands per VM is considered **given input data**.

- n clients with VMs and demands
- p PMs with capacities

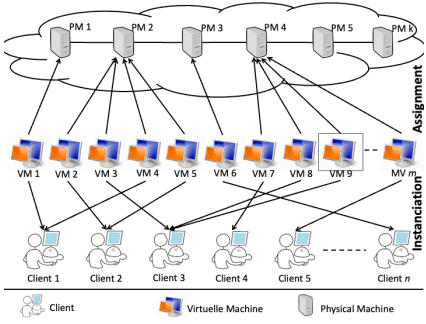


Multi-dimensional BPP - Cloud Computing Application (4/6)

Formal Definition of VMPMDP

Input Data

- **Clients:** $N = \{1, 2, \ldots, n\}$
- **VMs per client:** $M_i = \{1, 2, \ldots, m_i\}$ for each $i \in N$
- **Total VMs:** $m = \sum_{i=1}^{n} m_i$
- **Physical Machines:** $P = \{1, 2, \ldots, p\}$
- **VM demands:** VM $j \in M_i$ needs
  - CPU: $dcpu_{ij}$
  - RAM: $dram_{ij}$
- **PM capacities:** PM $k \in P$ has
  - CPU: $ccpu_k$
  - RAM: $cram_k$



Problem: Find optimal assignment **VM** $\rightarrow$ **PM** to minimize active PMs

Multi-dimensional BPP - Cloud Computing Application (5/6)

Mathematical Formulation

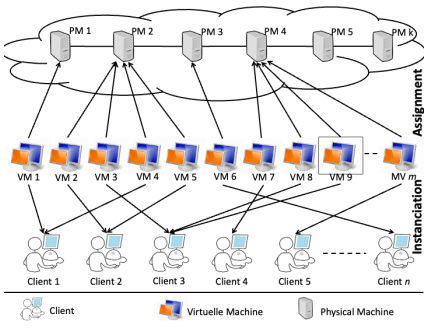
Decision Variables:

- $x_{ijk} \in \{0,1\}, \forall i \in N, j \in M_i, k \in P$, $x_{ijk} = 1$: VM j of client $i \rightarrow$ PM k, $x_{ijk} = 0$: otherwise
- $y_k \in \{0,1\}, \forall k \in P$, $y_k = 1$: PM k is active, $y_k = 0$: PM k is inactive

Objective: Minimize active PMs **Min** $Z = \sum_{k \in P} y_k$

Constraints:

- Each VM to exactly one PM $\sum_{k \in P} x_{ijk} = 1, \forall i \in N, j \in M_i$
- Don't exceed PM CPU capacity $\sum_{i \in N} \sum_{j \in M_i} dcpu_{ij} \cdot x_{ijk} \leq ccpu_k \cdot y_k, \forall k \in P$
- Don't exceed PM RAM capacity $\sum_{i \in N} \sum_{j \in M_i} dram_{ij} \cdot x_{ijk} \leq cram_k \cdot y_k, \forall k \in P$



Multi-dimensional BPP - Cloud Computing Application (6/6)

Complete VMPMDP Model

$$\text{Min } Z = \sum_{k \in P} y_k$$

$$\text{(s.t)} \begin{cases} \sum_{k \in P} x_{ijk} = 1, \forall i \in N, j \in M_i \\ \sum_{i \in N} \sum_{j \in M_i} dcpu_{ij} \cdot x_{ijk} \leq ccpu_k \cdot y_k, \forall k \in P \\ \sum_{i \in N} \sum_{j \in M_i} dram_{ij} \cdot x_{ijk} \leq cram_k \cdot y_k, \forall k \in P \\ x_{ijk} \in \{0,1\}, \forall i \in N, j \in M_i, k \in P \\ y_k \in \{0,1\}, \forall k \in P \end{cases}$$

Model Analysis

Type: Binary Integer Linear Programming (BILP) model.

- Variables: $m \times p + p$
  - $m \times p$ assignment variables x_{ijk}
  - p activation variables y_k
- Constraints: $m + 2p$
  - m assignment constraints
  - $2p$ capacity constraints

Key Insight

y_k variables link PM activation to assignments:

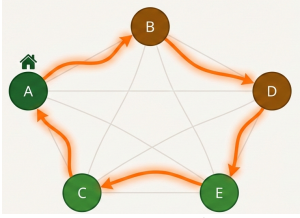
- If $y_k = 0 \rightarrow$ PM k inactive $\rightarrow$ no VMs assigned
- If $y_k = 1 \rightarrow$ PM k active $\rightarrow$ VMs can be assigned

Traveling Salesman Problem (1/8)

The **Traveling Salesman Problem (TSP)** is a classic combinatorial optimization problem that can be described as follows: given a graph $G = (V, E)$ where V is a set of n cities (or nodes) and E is a set of **edges or arcs** representing the distances between each pair of cities, the goal is to find the shortest **Hamiltonian cycle (circuit)** (a tour that visits each city exactly once and returns to the starting city) that minimizes the total travel distance. The distances between cities are represented by a distance matrix $D = [d_{ij}]$, where d_{ij} is the distance from city i to city j. It is assumed that there are no self-loops, i.e., $d_{ii} = 0, \forall i \in \{1, 2, \ldots, n\}$.

There are two versions of the TSP:

- **Symmetric TSP (STSP):** The distance between two cities is the same in both directions. $d_{ij} = d_{ji}, \forall i, j \in \{1, 2, \ldots, n\}, i \neq j$. In this case we use the term **undirected graph** since the **edges** have no direction.
- **Asymmetric TSP (ATSP):** The distance between two cities may differ depending on the direction of travel. $d_{ij} \neq d_{ji}, \forall i, j \in \{1, 2, \ldots, n\}, i \neq j$. In this case we use the term **directed graph** since the **arcs** have a direction.



Traveling Salesman Problem (2/8)

General formulation of the STSP

Decision variables: Binary variables $x_{ij} \in \{0,1\}, \forall i,j \in \{1,2,\ldots,n\}, i \neq j$ indicating whether the edge from city i to city j is included in the tour ($x_{ij}=1$) or not ($x_{ij}=0$).

Objective function: Minimize the total distance traveled in the tour. $MinZ = \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} d_{ij}x_{ij}$.

Constraints:

1. Flow conservation constraints: Each city must have degree 2 (entered and exited exactly once). $\sum_{j=1,j\neq i}^{n}(x_{ij}+x_{ji}) = 2, \forall i \in \{1,2,\ldots,n\}$.
2. Subtour elimination constraints: Prevent the formation of smaller cycles that do not include all cities. $\sum_{i\in S}\sum_{j\in S, j\neq i} x_{ij} \leq |S|-1, \forall S \subset \{1,2,\ldots,n\}, 2 \leq |S| \leq n-1$.

$$\text{Min } Z = \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n} d_{ij}x_{ij}$$

$$\text{(s.t)} \begin{cases} \sum_{j=1,j\neq i}^{n}(x_{ij}+x_{ji}) = 2, \forall i \in \{1,2,\ldots,n\} \\ \sum_{i\in S}\sum_{j\in S, j\neq i} x_{ij} \leq |S|-1, \forall S \subset \{1,2,\ldots,n\}, 2 \leq |S| \leq n-1 \\ \forall i,j \in \{1,2,\ldots,n\}, i \neq j, x_{ij} \in \{0,1\} \end{cases}$$

Model Analysis

This model is a **Binary Integer Linear Programming (BILP)** for **STSP** with $n \times (n-1)$ **variables** (using directed edge representation) and $n + 2^n - 2n - 1$ **constraints** (plus the binary constraints).

$\Rightarrow$ *The number of constraints is calculated as follows: there are n degree constraints (flow conservation) and* $\sum_{k=2}^{n-1}\binom{n}{k} = 2^n - 2n - 1$ *subtour elimination constraints.*

Traveling Salesman Problem (3/8)

Small example of the STSP with 4 cities

Consider 4 cities with the following distance matrix:

$$D = \begin{bmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 35 & 25 \\ 15 & 35 & 0 & 30 \\ 20 & 25 & 30 & 0 \end{bmatrix}$$

Decision variables: Binary variables $x_{ij} \in \{0,1\}, \forall i,j \in \{1,2,3,4\}, i \neq j$ indicating whether the edge from city i to city j is included in the tour ($x_{ij} = 1$) or not ($x_{ij} = 0$).

Objective function: Minimize the total distance traveled in the tour.

$MinZ = 10x_{12} + 15x_{13} + 20x_{14} + 10x_{21} + 35x_{23} + 25x_{24} + 15x_{31} + 30x_{32} + 30x_{34} + 20x_{41} + 25x_{42} + 15x_{43}$.

Constraints:

1. Flow conservation constraints: Each city must have degree 2 (sum of incident edges equals 2).
   $x_{12} + x_{13} + x_{14} + x_{21} + x_{31} + x_{41} = 2$,
   $x_{21} + x_{23} + x_{24} + x_{12} + x_{32} + x_{42} = 2$,
   $x_{31} + x_{32} + x_{34} + x_{13} + x_{23} + x_{43} = 2$,
   $x_{41} + x_{42} + x_{43} + x_{14} + x_{24} + x_{34} = 2$.
2. Subtour elimination constraints to prevent the formation of smaller cycles that do not include all cities. $x_{12} + x_{21} \leq 1$, $x_{13} + x_{31} \leq 1$, $x_{14} + x_{41} \leq 1$, $x_{23} + x_{32} \leq 1$, $x_{24} + x_{42} \leq 1$, $x_{34} + x_{43} \leq 1$, $x_{12} + x_{23} + x_{31} \leq 2$, $x_{12} + x_{24} + x_{41} \leq 2$, $x_{13} + x_{32} + x_{21} \leq 2$, $x_{13} + x_{34} + x_{41} \leq 2$, $x_{14} + x_{42} + x_{21} \leq 2$, $x_{14} + x_{43} + x_{31} \leq 2$

How to prevent subtours?

Consider a solution with two subtours: {1,2} and {3,4}.

- $x_{12} + x_{21} \leq 1$ (prevents the subtour {1,2})
- $x_{34} + x_{43} \leq 1$ (prevents the subtour {3,4})

Consider a solution with two subtours: {1,2,3} and {4}.

- $x_{12} + x_{23} + x_{31} \leq 2$ (prevents the subtour {1,2,3})

Traveling Salesman Problem (4/8)

General formulation of the ATSP

Decision variables:

- Binary variables $x_{ij} \in \{0,1\}, \forall i,j \in \{1,2,\ldots,n\}, i \neq j$ indicating whether the arc from city i to city j is included in the tour ($x_{ij} = 1$) or not ($x_{ij} = 0$).
- Continuous auxiliary variables $u_i, \forall i \in \{2,3,\ldots,n\}$ representing the position (order) of city i in the tour, with $1 \leq u_i \leq n-1$.

Objective function: Minimize the total distance traveled in the tour. $MinZ = \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} d_{ij} x_{ij}$.

Constraints:

1. Each city must be entered exactly once.
   $\sum_{i=1,i\neq j}^{n} x_{ij} = 1, \forall j \in \{1,2,\ldots,n\}$
2. Each city must be exited exactly once.
   $\sum_{j=1,j\neq i}^{n} x_{ij} = 1, \forall i \in \{1,2,\ldots,n\}$
3. Miller-Tucker-Zemlin (MTZ) subtour elimination constraints to prevent the formation of smaller cycles that do not include all cities.
   $u_i - u_j + n x_{ij} \leq n-1, \forall i,j \in \{2,3,\ldots,n\}, i \neq j$.

$$\text{Min } Z = \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} d_{ij} x_{ij}$$

(s.t)

$$\begin{cases} \sum_{i=1,i\neq j}^{n} x_{ij} = 1, \forall j \in \{1,2,\ldots,n\} \\ \sum_{j=1,j\neq i}^{n} x_{ij} = 1, \forall i \in \{1,2,\ldots,n\} \\ u_i - u_j + n x_{ij} \leq n-1, \forall i,j \in \{2,3,\ldots,n\}, i \neq j \\ \forall i,j \in \{1,2,\ldots,n\}, i \neq j, x_{ij} \in \{0,1\} \\ \forall i \in \{2,3,\ldots,n\}, 1 \leq u_i \leq n-1 \end{cases}$$

Model Analysis

This model is a **Mixed Integer Linear Programming (MILP)** for **ATSP** with $n \times (n-1)$ **binary variables** and $(n-1)$ **continuous variables**, and $2n+(n-1)(n-2)$ **constraints** (plus the variable bound constraints).

Traveling Salesman Problem (5/8)

Small example of the ATSP with 4 cities

Consider 4 cities with the following distance matrix:

$$D = \begin{bmatrix} 0 & 10 & 15 & 20 \\ 12 & 0 & 35 & 25 \\ 15 & 30 & 0 & 30 \\ 21 & 23 & 18 & 0 \end{bmatrix}$$

Decision variables:

- Binary variables $x_{ij} \in \{0,1\}, \forall i,j \in \{1,2,3,4\}, i \neq j$ indicating whether the arc from city i to city j is included in the tour ($x_{ij} = 1$) or not ($x_{ij} = 0$).
- Continuous auxiliary variables u_2, u_3, u_4 with $1 \leq u_i \leq 3$.

Objective function: Minimize the total distance traveled in the tour.

$MinZ = 10x_{12} + 15x_{13} + 20x_{14} + 12x_{21} + 35x_{23} + 25x_{24} + 15x_{31} + 30x_{32} + 30x_{34} + 21x_{41} + 23x_{42} + 18x_{43}$.

Constraints:

1. Each city must be entered exactly once.
   $x_{21} + x_{31} + x_{41} = 1$, $x_{12} + x_{32} + x_{42} = 1$,
   $x_{13} + x_{23} + x_{43} = 1$, $x_{14} + x_{24} + x_{34} = 1$.
2. Each city must be exited exactly once.
   $x_{12} + x_{13} + x_{14} = 1$, $x_{21} + x_{23} + x_{24} = 1$,
   $x_{31} + x_{32} + x_{34} = 1$, $x_{41} + x_{42} + x_{43} = 1$.
3. Miller-Tucker-Zemlin (MTZ) subtour elimination constraints to prevent the formation of smaller cycles that do not include all cities. $u_2 - u_3 + 4x_{23} \leq 3$, $u_2 - u_4 + 4x_{24} \leq 3$, $u_3 - u_2 + 4x_{32} \leq 3$, $u_3 - u_4 + 4x_{34} \leq 3$, $u_4 - u_2 + 4x_{42} \leq 3$, $u_4 - u_3 + 4x_{43} \leq 3$

How to prevent subtours using MTZ constraints?

Consider a solution with two subtours: {2,3} and {1,4} (city 1 is the depot).

- $u_2 - u_3 + 4x_{23} \leq 3$, $u_3 - u_2 + 4x_{32} \leq 3$ (prevents the subtour {2,3})

Note: MTZ constraints ensure city 1 is always the starting depot, so any cycle not including city 1 is automatically prevented by the u_i ordering.

- If $x_{23} = 1$, then $u_2 < u_3$, enforcing a strict ordering that prevents cycles among cities $\{2,3,4\}$.

Traveling Salesman Problem (6/8) - STSP Complexity

- The size of search space is $\frac{(n-1)!}{2}$, where n is the number of cities.
- For **STSP**, tours and their reverses are **identical** (e.g., $1\to2\to3\to1 = 1\to3\to2\to1$), so we divide by 2.
- The **Symmetric Traveling Salesman Problem (STSP)** is **NP-hard** problem encountered in various fields such as logistics, transportation, and manufacturing.

Number of cities (n)	Number of possible tours	Approx. Time to enumerate (seconds)
5	12	0.000012
10	181,440	0.18144
15	43,589,145,600/2 = 21,794,572,800	21,794.5728 = 6.05 hours
20	60,822,550,204,416,000	$6.08 \times 10^{16} = 1.93 \times 10^{9}$ years

- **STSP** benefits from symmetric structure: half the tours and fewer unique edges
- Modern solvers exploit symmetry to reduce computational burden

Traveling Salesman Problem (7/8) - ATSP Complexity

- The size of search space is $(n-1)!$, where n is the number of cities.
- For **ATSP**, tours and their reverses are **different** (e.g., $1\to2\to3\to1 \neq 1\to3\to2\to1$) due to asymmetric distances.
- The table shows the combinatorial explosion of the search space (*each tour evaluated in* 10^{-6} *seconds*):

Number of cities (n)	Number of possible tours	Approx. Time to enumerate (seconds)
5	24	0.000024
10	362,880	0.36288
15	43,589,145,600	43,589.1456 = 12.1 hours
20	121,645,100,408,832,000	$1.2165 \times 10^{17} = 3.86 \times 10^{9}$ years

- **ATSP** is generally harder to solve: lacks symmetric structure exploitation
- MTZ formulation allows polynomial constraints but introduces additional continuous variables

Traveling Salesman Problem (8/8) - STSP and ATSP Comparison

Comparison between STSP and ATSP Models

Characteristic	Symmetric TSP (STSP)	Asymmetric TSP (ATSP)
Distance Matrix	$d_{ij} = d_{ji}$ (symmetric)	$d_{ij} \neq d_{ji}$ (asymmetric)
Graph Type	Undirected graph (edges)	Directed graph (arcs)
Decision Variables	$n \times (n-1)$ binary variables x_{ij}	$n \times (n-1)$ binary variables $x_{ij} + (n-1)$ continuous variables u_i
Constraints	n flow conservation + $2^n - 2n - 1$ subtour elimination	$2n$ flow (in/out) + $(n-1)(n-2)$ MTZ constraints
Total Constraints	$2^n - n - 1$ (exponential)	$n^2 - n + 2$ (polynomial)
Subtour Elimination	Exponential number of constraints	Miller-Tucker-Zemlin (MTZ) constraints
Model Type	Binary Integer LP (BILP)	Mixed Integer LP (MILP)
Complexity	**NP-hard**	**NP-hard**
Solving Difficulty	Generally easier due to symmetry	Generally harder (fewer structural properties)

- **STSP** has search space of size $\frac{(n-1)!}{2}$ (tours and reverse are identical)
- **ATSP** has search space of size $(n-1)!$ (direction matters)
- **STSP** model has fewer constraints in practice but exponential growth
- **ATSP** model has polynomial constraints but more complex structure

Key Takeaways

Key Takeaways

- **Linear Programming problems** can model a wide variety of real-world optimization scenarios including **scheduling**, **assignment**, **resource allocation**, and **routing problems**.
- Common **LP problem types** include the **Knapsack Problem**, **Bin Packing Problem**, and **Traveling Salesman Problem**, each with distinct characteristics and complexities.
- Most practical **LP problems** are **NP-hard**, meaning they exhibit **exponential growth** in computational complexity as problem size increases.
- The **Symmetric TSP** and **Asymmetric TSP** differ in their distance matrices, decision variables, constraints, and solution approaches.

Outline

1. Optimization Problems

2. Linear Programming Model

3. Examples of LP Problems

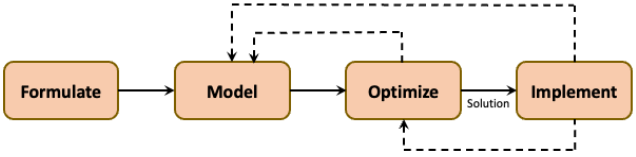
4. Optimization-Based Decision Making
4.1 Decision Making Steps
4.2 Classical Optimization Models
4.3 Different Families of Optimization Models
4.4 Key Takeaways

5. Complexity Theory

6. Optimization Methods

Decision Making Steps

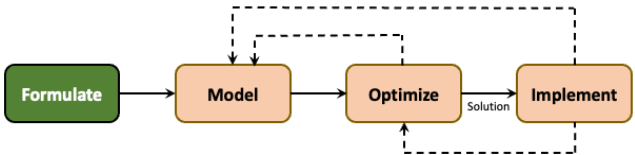
- As **scientists**, **engineers**, and **managers**, we always have to **take decisions**.
- **Decision making is everywhere**.
- As **the world becomes more and more complex and competitive**, **decision making** must be tackled in a rational and optimal way.
- **Decision making consists in the following steps**:



The **classical process** in **decision making**: **formulate**, **model**, **solve**, and **implement**.
⇒**In practice, this process may be iterated to improve the optimization model or algorithm until an acceptable solution is found.**

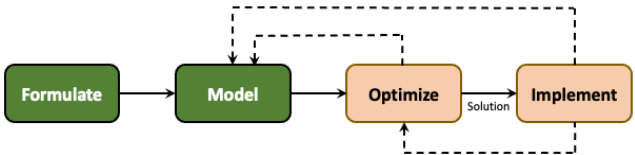
Step 1: Formulate the Problem

1. The **first step** consists of **formulating the problem**:
   - A **decision problem** is **identified**.
   - Then, an **initial statement of the problem** is **made**.
   - **This formulation may be imprecise**.
   - The **internal** and **external factors**, and the **objective(s) of the problem** are **outlined**.
   - Many **decision makers** may be involved in **formulating the problem**.



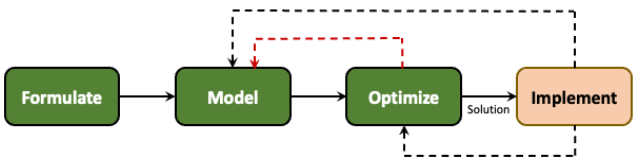
Step 2: Model the Problem

2. The **second step** consists of **modeling the problem**:
   - An **abstract mathematical model** is **built for the problem**.
   - **The modeler can be inspired by similar models in the literature.**
   - Usually, **models** we are solving **are simplifications of the reality**. They **involve approximations and sometimes they skip processes** that are **complex** to represent in a **mathematical model**.



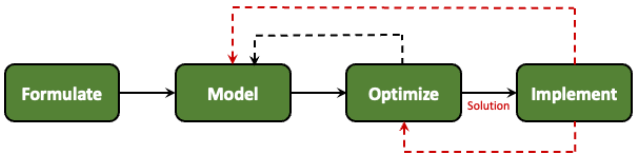
Step 3: Optimize the Problem

3. The **third step** consists of **optimizing the problem**:
   - The **solving procedure generates a "good" solution** for the problem.
   - The **solution** may be **optimal** or **sub-optimal**.
   - **Let us notice** that we are **finding a solution for an abstract model of the problem** and **not for the originally formulated real-life problem**.
   - The **algorithm designer** can **reuse state-of-the-art algorithms on similar problems or integrate the knowledge of this specific application into the algorithm**.



Step 4: Implement the Solution

4. The **fourth step** consists of **implementing the solution**:
   - The **obtained solution** is **tested practically** by the **decision maker** and is **implemented if it is "acceptable"**.
   - **Some practical knowledge may be introduced in the solution to be implemented**.
   - **If the solution is unacceptable**, the **model** and/or the **optimization algorithm** has **to be improved** and **the decision making process is repeated**.



Classical Optimization Models

- As mentioned, **optimization problems are encountered in many domains**: **science**, **engineering**, **management**, and **business**.
- An **optimization problem** may be defined by the **couple** (S, f), where S represents the **set of feasible solutions**[1], and $f : S \rightarrow \mathbb{R}$ **or** $\mathbb{N}$ the **objective function**[2] **to optimize**.
- The **objective function** f **assigns** to every solution $s \in S$ of the **search space** a **real or integer number indicating its value**.
- The **objective function** f **is used to define** a **total order relation between any pair of solutions in the search space**.

[1] *Sometimes named: Feasible Region, or Feasible Set, or Search Space, or Solution Space*
[2] *Sometimes named: Cost, Utility, or Fitness Function.*

Global Optimum

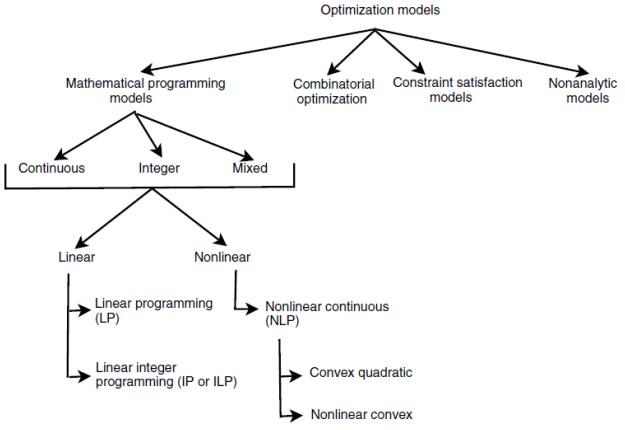
A solution $s^* \in S$ is a **global optimum** if it **has a better objective function value**[3] **than all solutions of the search space**, that is, $\forall s \in S, f(s^*) \leq f(s)$.

- The **main goal** in **solving** an **optimization problem** is to **find** a **global optimal solution** s^*.
- **Many global optimal solutions may exist** for a given problem.
- **Hence, to get more alternatives, the problem may also be defined as finding all global optimal solutions**.

[3] *We suppose without loss of generality a minimization problem. Maximizing an objective function f is equivalent to minimizing $-f$.*

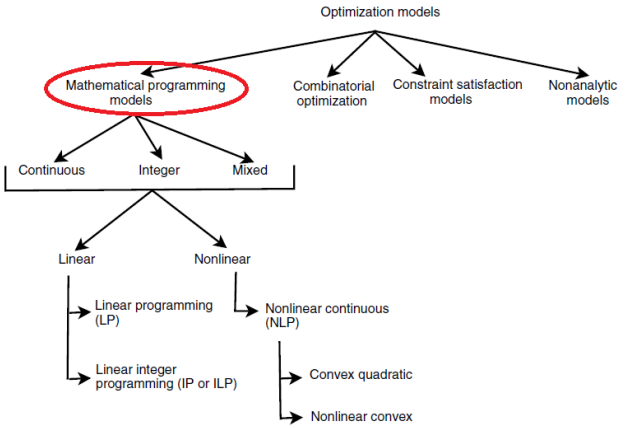
Different Families of Optimization Models

- Different **families of optimization models** are **used** in practice to **formulate** and **solve** **decision-making problems**.



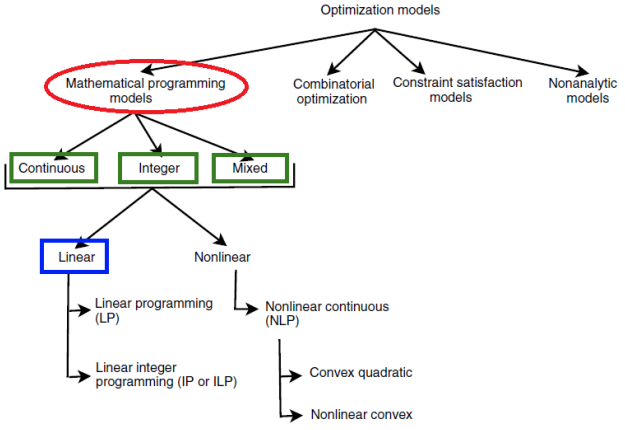
Different Families of Optimization Models

- Different **families of optimization models** are **used** in practice to **formulate** and **solve** **decision-making problems**.
- **The most successful models** are based on **mathematical programming**.



Different Families of Optimization Models

- Different **families of optimization models** are **used** in practice to **formulate** and **solve** **decision-making problems**.
- **The most successful models** are based on **mathematical programming**.
- **The most common mathematical programming models** are **linear** with **continuous**, or **integer**, or **mixed decision variables**.



Key Takeaways

Key Takeaways

- **Optimization-based decision making** involves four key steps: **formulate**, **model**, **solve**, and **implement**.
- An **optimization problem** can be defined by the couple (S, f), where S represents the **feasible solutions** and f is the **objective function** to optimize.
- A **global optimum** is a solution that has a better objective function value than all other solutions in the search space.
- The most successful optimization models are based on **mathematical programming**, particularly **linear programming** with continuous, integer, or mixed decision variables.

Outline

1. Optimization Problems

2. Linear Programming Model

3. Examples of LP Problems

4. Optimization-Based Decision Making

5. Complexity Theory
5.1 Complexity of Algorithms
5.2 Complexity of Problems
5.3 Key Takeaways

6. Optimization Methods

Complexity of Algorithms

- An **algorithm** needs **two important resources to solve a problem**: **time (CPU)** and **space (RAM)**.
- The **time complexity of an algorithm** is the **number of steps required to solve a problem of size** n.
- The **complexity** is generally **defined in terms of the worst-case analysis**.
- The **goal** in the determination of the **computational complexity of an algorithm** is **not to obtain an exact step count** but an **asymptotic bound on the step count**.
- The **Big-O notation makes use of asymptotic analysis**. **It is one of the most popular notations in the analysis of algorithms**.

Definition of Big-O notation

An algorithm has a complexity $f(n) = O(g(n))$ if there exist positive constants n_0 and c such that $\forall n > n_0, f(n) \leq cg(n)$.

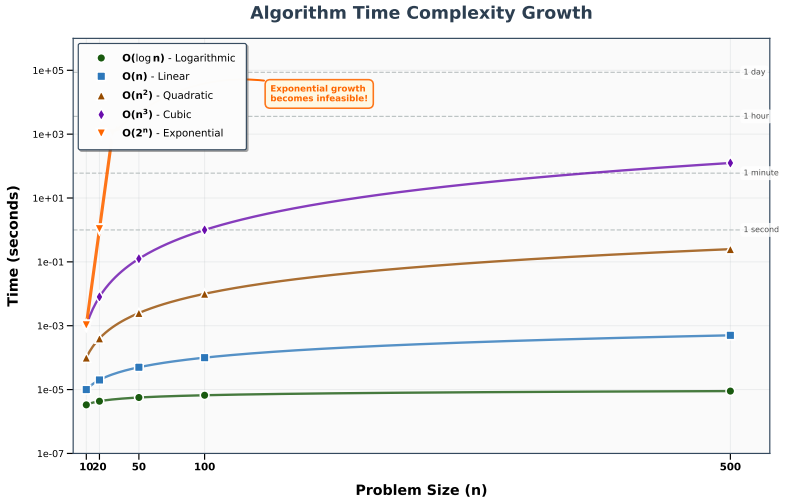
Search Time of Algorithms (1/2)

The below table illustrates how the **search time of an algorithm grows with the size of the problem using different time complexities of an optimization algorithm**.

- Consider that each step of the algorithm takes 10^{-6} seconds. The table shows clearly the **combinatorial explosion of exponential complexities compared to polynomial ones**.
- *In practice, we cannot wait some centuries to solve a problem.*
- *The algorithm shown in the last line of the table needs the age of universe to solve it in an exact manner using exhaustive search.*

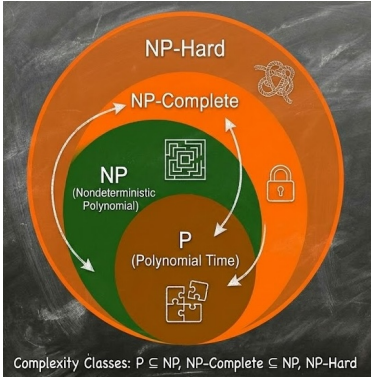
		Size				
Type of Algo.	Complexity	n= 10	n= 20	n= 30	n= 40	n= 50
Logarithmic	$O(\log n)$	0.000001 *s*	0.000001 *s*	0.000001 *s*	0.000002 *s*	0.000003 *s*
Polynomial						
Linear	$O(n)$	0.00001 *s*	0.00002 *s*	0.00003 *s*	0.00004 *s*	0.00005 *s*
Quadratic	$O(n^2)$	0.0001 *s*	0.0004 *s*	0.0009 *s*	0.0016 *s*	0.0025 *s*
Cubic	$O(n^3)$	0.001 *s*	0.008 *s*	0.027 *s*	0.064 *s*	0.125 *s*
Exponential	$O(2^n)$	0.001 *s*	1.0 *s*	17.9 *mn*	12.7 *days*	35.7 *years*
	$O(3^n)$	0.059 *s*	58.0 *mn*	6.5 *years*	3855 *centuries*	2×10^8 *centuries*

Search Time of Algorithms (2/2)



Complexity of Problems and Complexity Classes

- The **complexity of a problem** is the **complexity of the best algorithm** that solves it.
- **Tractable problems:** can be solved in polynomial time (easy).
- **Intractable problems:** require exponential time (hard).
- **P:** Problems solvable in polynomial time (e.g., sorting, search).
- **NP:** Problems verifiable in polynomial time, i.e., given a solution, we can check its correctness quickly (e.g., TSP, Knapsack).
- **NP-complete:** The hardest problems in NP, to which any NP problem can be reduced in polynomial time. If one NP-complete problem is solved in polynomial time, all NP problems can be.
- **NP-hard:** Most difficult problems, not necessarily in NP (e.g., Bin packing, Scheduling). No known polynomial-time solutions.
- **Heuristics/Metaheuristics** provide efficient solutions for NP-hard problems.



Key Takeaways

Key Takeaways

- The **time complexity** of an algorithm measures the number of steps required to solve a problem as a function of problem size n, typically expressed using **Big-O notation**.
- **Polynomial algorithms** (e.g., $O(n)$, $O(n^2)$, $O(n^3)$) remain practical for large problem sizes, while **exponential algorithms** (e.g., $O(2^n)$) quickly become computationally infeasible.
- Problems are classified into complexity classes: **P** (solvable in polynomial time), **NP** (verifiable in polynomial time), **NP-complete** (hardest NP problems), and **NP-hard** (most difficult problems).
- For **NP-hard problems**, **heuristics** and **metaheuristics** provide efficient approximate solutions when exact methods are impractical.

Outline

1. Optimization Problems

2. Linear Programming Model

3. Examples of LP Problems

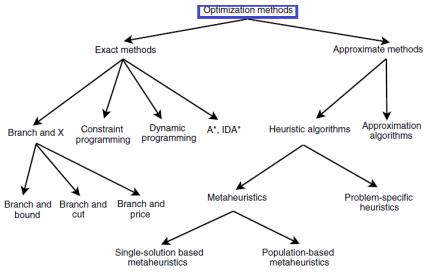
4. Optimization-Based Decision Making

5. Complexity Theory

6. Optimization Methods
6.1 Exact Methods
6.2 Approximate Algorithms
6.3 Heuristics
6.4 Metaheuristics
6.5 Key Takeaways

Optimization Methods

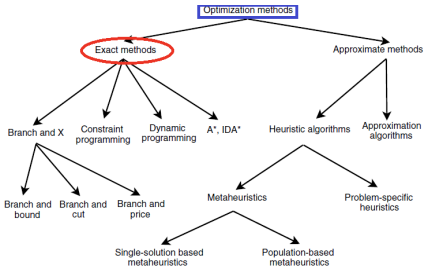
- **Following the complexity of the problem, it may be solved by an exact method or an approximate method.**



[a] In the artificial intelligence community, those algorithms are also named complete algorithms.

Optimization Methods

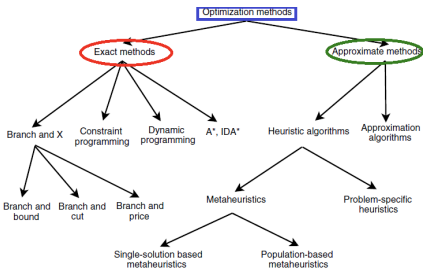
- **Following the complexity of the problem, it may be solved by an exact method** or an **approximate method**.
- **Exact methods**[a] **obtain optimal solutions and guarantee their optimality**.
- For **NP-complete** problems, exact algorithms are **nonpolynomial-time algorithms** (unless **P = NP**).



[a] In the artificial intelligence community, those algorithms are also named complete algorithms.

Optimization Methods

- **Following the complexity of the problem, it may be solved by an exact method** or an **approximate method**.
- **Exact methods**[a] **obtain optimal solutions and guarantee their optimality**.
- For **NP-complete** problems, exact algorithms are **nonpolynomial-time algorithms** (unless **P = NP**).
- **Approximate (or heuristic) methods generate high-quality solutions in a reasonable time for practical use, but there is no guarantee of finding a global optimal solution**.



[a] In the artificial intelligence community, those algorithms are also named complete algorithms.

Exact Methods (1/2)

In the class of **exact methods** we can find the following classical algorithms:

- **Dynamic programming** is based on the recursive division of a problem into simpler subproblems.
- **Branch and Bound** and **A*** algorithms are based on an implicit enumeration of all solutions of the considered optimization problem.
- **Constraint programming** is a language built around concepts of tree search and logical implications.

⇒**Exact methods** can be applied to **small instances** of **difficult problems** (see the below Table that shows the *order of magnitude of the maximal size of instances that state-of-the-art exact methods can solve to optimality*):

Optimization Problem	Quadratic Assignment	Flow-Shop Scheduling	Graph Coloring	Capacitated Vehicle Routing	Traveling Salesman Problem
Size of the instances	30 objects	100 jobs 20 machines	100 nodes	60 clients	100 cities

Exact Methods (2/2)

The **size of the instance** is **not the unique indicator that describes the difficulty of a problem**, but also **its structure**.

- For a given problem, **some small instances cannot be solved by an exact algorithm while some large instances may be solved exactly by the same algorithm**.
- The below table shows for some popular optimization problems *(e.g., SOP[4]: Sequential Ordering Problem; QAP[5]: Quadratic Assignment Problem; GC[6]: Graph Coloring)* **small instances that are not solved exactly and large instances solved exactly by state-of-the-art exact optimization methods**.

Optimization Problem	SOP	QAP	GC
Size of some unsolved instances	53	30	125
Size of some solved instances	70	36	561

[4] See https://arxiv.org/abs/1911.12427
[5] See https://en.wikipedia.org/wiki/Quadratic_assignment_problem
[6] See https://en.wikipedia.org/wiki/Graph_coloring

Approximate Algorithms

- In the class of approximate methods, two subclasses of algorithms may be distinguished:
  - **Approximation Algorithms**
  - **Heuristic Algorithms**
- Unlike **heuristics**, which usually find **reasonably "good" solutions in a reasonable time**, **approximation algorithms provide provable solution quality and provable run-time bounds**[7].

⇒**Heuristics** find "good" solutions **on large-size problem instances**.
They allow to obtain **acceptable performance** at **acceptable costs** in a wide range of problems.
In general, heuristics do not have an approximation guarantee on the obtained solutions.

[7] A provable solution quality means that the algorithm guarantees that the solution found is within a certain factor of the optimal solution. A provable run-time bound means that the algorithm's execution time can be bounded by a specific function of the input size.

Heuristics

Heuristics find "good" solutions **on large-size problem instances**.

- They allow to obtain **acceptable performance** at **acceptable costs** in a wide range of problems.
- **In general, heuristics do not have an approximation guarantee on the obtained solutions.**
- They may be classified into two families:
  - **Specific Heuristics** that are **tailored and designed to solve a specific problem and/or instance**.
  - **Metaheuristics** that are **general-purpose algorithms that can be applied to solve almost any optimization problem**. *They may be viewed as upper level general methodologies that can be used as a guiding strategy in designing underlying heuristics to solve specific optimization problems*.

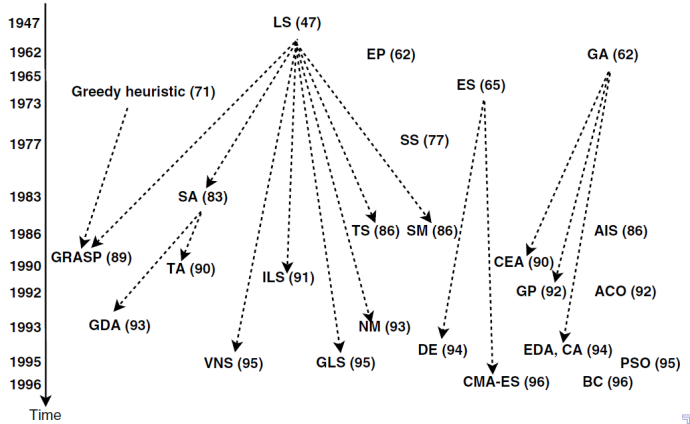
Metaheuristics (1/2)

Unlike **exact methods**, **Metaheuristics** allow to tackle large-size problem instances by delivering satisfactory solutions in a reasonable time.

- **There is no guarantee to find global optimal solutions or even bounded solutions.**
- **Metaheuristics have received more and more popularity in the past 30 years.**
- **Their use in many applications shows their efficiency and effectiveness to solve large and complex problems.**
- Application of Metaheuristics falls into a large number of areas; some them are:
  - **Combinatorial Optimization:** Traveling Salesman Problem, Vehicle Routing Problem, Job-Shop Scheduling Problem, Graph Coloring Problem, Knapsack Problem, etc.
  - **Continuous Optimization:** Function Optimization, Parameter Tuning, Machine Learning Hyperparameter Optimization, etc.
  - **Multi-objective Optimization:** Engineering Design, Portfolio Optimization, Supply Chain Management, etc.
  - **Real-world Applications:** Network Design, Resource Allocation, Bioinformatics, Robotics, etc.

Metaheuristics (2/2)

Optimization is everywhere; optimization problems are often complex; then **Metaheuristics** are everywhere. The below figure shows the **genealogy of the numerous Metaheuristics** in the literature.

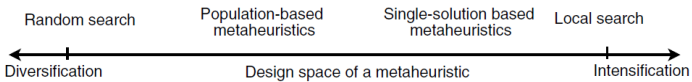


Categories of Metaheuristics (1/3)

In designing a **metaheuristic**, two contradictory criteria must be taken into account: **exploitation** of the best solutions found **(intensification)** and **exploration** of the search space **(diversification)**.

- **In intensification**, the promising regions are explored more thoroughly in the hope to find better solutions.
- **In diversification**, nonexplored regions must be visited to be sure that all regions of the search space are evenly explored and that the search is not confined to only a reduced number of regions.

$\Rightarrow$ In this design space, the extreme search algorithms in terms of the **exploration (resp. exploitation) are random search (resp. iterative improvement local search)**.



Categories of Metaheuristics (2/3)

Metaheuristics can be classified into two main categories:

- **Single-Solution Based Search Metaheuristics**: **intensification oriented**, at each iteration one selects the best neighboring solution that improves the current solution.
  - Local Search, Iterated Local Search, Guided Local Search.
  - Variable Neighborhood Search
  - Tabu Search
  - Simulated Annealing
- **Population-Based Search Metaheuristics**: **diversification oriented**, at each iteration, one generates a random solution in the search space.
  - Genetic Algorithms
  - Swarm Intelligence Algorithms
  - Scatter Search

Categories of Metaheuristics (3/3)

Many other classification criteria may be used for **Metaheuristics**:

- **Nature inspired versus nonnature inspired:** Many Metaheuristics are inspired by natural processes from biology (e.g., ants, bees colonies), or social science (e.g., particle swarm), or physics (e.g., simulated annealing).
- **Memory usage versus memoryless methods:** Some metaheuristic algorithms use a memory that contains some information extracted online during the search (e.g., short-term and long-term memories in tabu search).
- **Deterministic versus stochastic:** In stochastic Metaheuristics, some random rules are applied during the search where different final solutions may be obtained from the same initial solution.(e.g., simulated annealing, evolutionary algorithms).
- **Iterative versus greedy:** Greedy algorithms start from an empty solution, and at each step a decision variable of the problem is assigned until a complete solution is obtained. Most of the Metaheuristics are iterative algorithms that start with a complete solution (or population of solutions) and transform it at each iteration using some search operators.

Key Takeaways

Key Takeaways

- Optimization methods can be classified into **exact methods** (guarantee optimal solutions but are computationally expensive) and **approximate methods** (provide good solutions in reasonable time).
- **Exact methods** such as dynamic programming, branch and bound, and constraint programming are suitable for small instances of difficult problems.
- **Approximate methods** include **approximation algorithms** (with provable solution quality) and **heuristics** (find good solutions without guarantees).
- **Metaheuristics** are general-purpose algorithms that balance **intensification** (exploitation) and **diversification** (exploration), and can be classified as **single-solution based** or **population-based** methods.

End of Chapter 1