

Combinatorial optimization problems



Outline

Why combinatorial optimization?

Combinatorial optimization problems

Categories of combinatorial optimization problems

Counting the solutions

Combinatorial explosion and brute force

Primal bounds, dual bounds and gap

A glimpse of computational complexity

Typical mistakes

In summary

Learning objectives

At the end of this chapter you will be able to:

- ▶ recognize a combinatorial optimization problem, the category of its decisions and the combinatorial object behind its solutions (a subset, an ordering, a partition, an assignment);
- ▶ count the candidate solutions (2^n subsets, $n!$ orderings, $(n-1)!/2$ cyclic orderings, Bell numbers for the partitions) and estimate the computing time of a brute force enumeration;
- ▶ define primal bounds, dual bounds and the gap, and use them to certify that a solution is optimal;
- ▶ place a problem in the classes P, NP and NP-hard, and tell a pseudo-polynomial algorithm from a polynomial one.

Prerequisites

- ▶ From the *Preliminaries*: optimization problems (feasible region, objective function, optimal solution and optimal value), binomial coefficients and partitions of a finite set
- ▶ No knowledge of integer programming is needed: the bounds of this chapter come from combinatorial arguments

Why combinatorial optimization?

Introduction: Why combinatorial optimization?

- ▶ Many decisions do not choose a number, but a **discrete object**: a set of projects, an order of visit, a way to split items among containers
- ▶ In this section we see where these decisions arise and why they matter in practice
- ▶ We meet the six problems around which the course is built, each one through a short story

Optimization problems: a recall

- ▶ We recall that, given a set F and a function f defined on F , the associated **optimization problem** P in maximization (minimization) form asks to find a point $\tilde{\mathbf{x}}(P) \in F$ such that

$$f(\tilde{\mathbf{x}}(P)) \geq f(\mathbf{x}) \quad (\leq), \quad \forall \mathbf{x} \in F$$

- ▶ The value $z(P) = f(\tilde{\mathbf{x}}(P))$ is the **optimal value**
- ▶ In many decision problems the feasible solutions are not points of a continuum but **discrete objects**:
 - ▶ a set of projects to fund;
 - ▶ an order in which to visit customers;
 - ▶ a way to split items among containers.

Where the problems arise (1/4)

- Combinatorial optimization problems arise whenever decisions involve **selecting**, **ordering**, **grouping** or **assigning** discrete objects. The main application areas, with typical problems:

Area	Typical problems
Production planning	what, when and how much to produce: demand satisfaction over several periods, lot sizing , capacity planning, assembly line balancing
Scheduling	order and timing of jobs on machines (makespan, tardiness), project scheduling, maintenance scheduling

Where the problems arise (2/4)

Area	Typical problems
Facility location	where to open warehouses, plants, emergency services, and which clients each one serves
Packing	load items into vehicles, containers or pallets; select the most valuable cargo for a single container
Cutting	cut bars, sheets, rolls or panels into the required pieces with minimum waste (steel, glass, textile, timber, paper)
Logistics	vehicle routing, fleet management, inventory management, distribution network design

Where the problems arise (3/4)

Area	Typical problems
Transportation	crew scheduling, aircraft routing, train timetabling, cargo loading
Personnel	workforce and shift scheduling, nurse rostering, crew pairing
Network design	telecommunication, energy, transportation and supply chain networks
Finance	portfolio selection with cardinality constraints, index tracking, capital budgeting

Where the problems arise (4/4)

Area	Typical problems
Telecommunications	frequency assignment, routing, capacity expansion, bandwidth allocation
Energy	unit commitment (which plants to switch on), generation scheduling, grid operation
Healthcare	surgery scheduling, radiotherapy planning, kidney exchange and organ matching, bed management
Other areas	crop planning, sports scheduling, allocation of virtual machines to servers, patrol planning

A tool for decision making

Combinatorial optimization is a fundamental tool for decision making.

The parcels we receive, the electricity we use and the flights we take are planned by solving combinatorial optimization problems.

The six problems of the course

The course is built around **six problems**.

Each one is introduced here with a short story;
the formal definitions and the variants come later in the
course.

Six stories (1/3)

The 0/1 knapsack problem

- ▶ A company has a budget of c euros and n candidate projects; project j costs w_j and yields a profit p_j
- ▶ Which projects should be funded?

The bin packing problem

- ▶ Parcels of different weights must be shipped with identical trucks of capacity c
- ▶ How many trucks are needed?
- ▶ The same problem arises when virtual machines are placed on identical servers

Six stories (2/3)

The vertex coloring problem

- ▶ Six exams must be scheduled in the exam session. Two exams that share at least one student cannot take place on the same day
- ▶ What is the minimum number of days?

The travelling salesman problem

- ▶ A courier leaves the depot, must visit $n - 1$ customers once each and come back
- ▶ Which tour is the shortest?

Scheduling on identical parallel machines

- ▶ Seven print jobs must be sent to three identical printers. Each job runs on one printer without interruption
- ▶ How should the jobs be distributed so that all of them are finished as early as possible?

The capacitated lot sizing problem

- ▶ A factory knows the demand of its product for the next m periods (weeks)
- ▶ Producing in a period requires a **setup** (a fixed cost to prepare the line) and is limited by the capacity of the line; what is produced in advance is kept in stock, at a holding cost
- ▶ In which periods should the factory produce, and how much?

Summary

Discrete decisions:

- ▶ The feasible solutions are discrete objects: a set of projects, an order of visit, a split of items among containers

Applications:

- ▶ Selecting, ordering, grouping and assigning: production, scheduling, location, packing, cutting, logistics, transportation, personnel, networks, finance, telecommunications, energy, healthcare

Six problems:

- ▶ Knapsack, bin packing, vertex coloring, travelling salesman, scheduling on parallel machines, lot sizing

Combinatorial optimization problems

Introduction: Combinatorial optimization problems

- ▶ In this section we give the formal definition of a **combinatorial optimization problem**: a ground set, a family of feasible solutions and an objective function
- ▶ We distinguish a **problem** from its **instances**, and we introduce a small instance that we use throughout the chapter
- ▶ We list the main characteristics of these problems: easy to state, often very hard to solve

Definition 1: combinatorial optimization problem

Given:

1. a *finite* set $\mathcal{U} = \{1, 2, \dots, n\}$, called the **ground set**,
2. a family $\mathcal{F} \subseteq 2^{\mathcal{U}}$ of subsets of $\mathcal{U}$, called the **feasible solutions**,
3. an **objective function** $f : \mathcal{F} \rightarrow \mathbb{R}$,

the associated **combinatorial optimization problem** asks to find a feasible solution $\tilde{S} \in \mathcal{F}$ such that

$$f(\tilde{S}) \geq f(S), \quad \forall S \in \mathcal{F} \quad (\text{maximization form}),$$

or such that

$$f(\tilde{S}) \leq f(S), \quad \forall S \in \mathcal{F} \quad (\text{minimization form}).$$

Feasible region and objective function

- ▶ The feasible region $\mathcal{F}$ is a **finite** set, since $|2^{\mathcal{U}}| = 2^n$
- ▶ Accordingly, an optimal solution always exists whenever $\mathcal{F} \neq \emptyset$
- ▶ In most problems the objective function is **linear** (additive): each element $j \in \mathcal{U}$ has a weight c_j and

$$f(S) = \sum_{j \in S} c_j$$

- ▶ Some problems have a different objective, for example the largest of several sums, which is a maximum and not a sum

Problems and instances

- ▶ A **problem** is a general question with symbolic data
- ▶ An **instance** I of the problem is obtained by fixing the numbers
- ▶ We denote by $z(I)$ the optimal value of the instance I

We now fix the numbers of a small instance:
we will use it throughout the chapter.

Example

- ▶ We consider the ground set $\mathcal{U} = \{1, 2, 3, 4\}$, the weights $\mathbf{c} = (7, 6, 4, 3)$ and the additive objective $f(S) = \sum_{j \in S} c_j$, to be maximized
- ▶ The feasible solutions are the 11 subsets

$$\mathcal{F} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 3\}, \{1, 4\}, \\ \{2, 3\}, \{2, 4\}, \{3, 4\}, \{2, 3, 4\}\}$$

- ▶ For example, $f(\{1, 3\}) = 7 + 4 = 11$ and $f(\{2, 3, 4\}) = 6 + 4 + 3 = 13$, while the subset $\{1, 2\}$ is not feasible
- ▶ We call it the **small instance**

How the feasible solutions are given

- ▶ In the small instance $\mathcal{F}$ is **listed**
- ▶ In the problems of the course it is never listed: it is described by a **rule**, for example a capacity constraint
- ▶ The family $\mathcal{F}$ is far too large to be written down

What is combinatorial optimization

Combinatorial optimization is the study of optimization problems on **discrete structures** and **combinatorial objects**, such as graphs, subsets, permutations, sequences, partitions, coverings and packings.

Main characteristics

The **main characteristics** of combinatorial optimization problems are:

1. the set of feasible solutions is **finite**;
2. the number of feasible solutions is typically **enormous**: it grows exponentially (or faster) with the size of the instance;
3. finding an optimal solution can be **computationally very difficult** (see the glimpse of computational complexity later in this chapter);
4. specialized algorithms are needed to explore the solution space without enumerating it.

Summary

Definition:

- ▶ A finite ground set $\mathcal{U}$, a family of feasible solutions $\mathcal{F} \subseteq 2^{\mathcal{U}}$, an objective function f , in maximization or minimization form
- ▶ An optimal solution exists whenever $\mathcal{F} \neq \emptyset$; the objective is usually additive

Problems and instances:

- ▶ An instance I fixes the numbers; $z(I)$ is its optimal value
- ▶ The small instance: $\mathcal{U} = \{1, 2, 3, 4\}$, $\mathbf{c} = (7, 6, 4, 3)$, 11 feasible subsets

Characteristics:

- ▶ Finite but enormous feasible region: specialized algorithms are needed

Categories of combinatorial optimization problems

Introduction: Categories of combinatorial optimization problems

- ▶ The decisions of a combinatorial optimization problem fall into five **categories**
- ▶ Each category is associated with a **combinatorial object**, which is the form of a solution, and with a natural type of **decision variables**
- ▶ We then see the main application areas and we place the problems of the course in their categories

Categories of decisions (1/3)

1. **IN/OUT (inclusion/exclusion).** Each element is either selected or not: a solution is a **subset** of the elements, described by binary variables $x_j \in \{0, 1\}$, with $x_j = 1$ if and only if element j is selected.
 - ▶ Examples: select the projects to fund with a limited budget, choose the products of a portfolio, decide which clients to visit in a route.
2. **Counting (discrete quantities).** The decision is how many units of each element to use: a solution is a vector of **integer quantities**, described by integer variables $x_j \in \mathbb{Z}_{\geq 0}$.
 - ▶ Examples: how many units of a product to produce, how many operators to assign to a shift, how many servers to activate in a data center.

Categories of decisions (2/3)

3. **Assignment.** Each element is assigned to exactly one of several distinct resources: a solution is a **function** from the elements to the resources, described by binary variables $x_{ij} \in \{0, 1\}$, equal to 1 if and only if element j is assigned to resource i .
 - ▶ Examples: assign tasks to machines or people, match students to projects, allocate vehicles to delivery routes.
4. **Ordering and sequences.** The decision is the order in which the elements are processed or visited: a solution is a **permutation** $\sigma = (\sigma_1, \dots, \sigma_n)$ of the elements, or a cyclic ordering for a round trip.
 - ▶ Examples: the order in which jobs are processed, the sequence of operations on a machine, the order of visit of the customers.

Categories of decisions (3/3)

5. **Configuration and partitioning.** The elements are subdivided into groups, each one satisfying a condition such as a capacity: a solution is a **partition** of the elements into interchangeable groups.
- ▶ Examples: subdivide a set of elements into groups, pack objects into bins, cut materials minimizing waste.

Let us collect the five categories in a table.

Categories, objects and variables

Category	Combinatorial object	Natural variables
IN/OUT	subset	$x_j \in \{0, 1\}$
counting	vector of integer quantities	$x_j \in \mathbb{Z}_{\geq 0}$
assignment	function (labelled classes)	$x_{ij} \in \{0, 1\}$, one resource i for each j
ordering and sequences	permutation, cyclic ordering	$x_{ij} \in \{0, 1\}$, equal to 1 if j follows i
configuration and partitioning	partition (unlabelled classes)	$x_{ij} \in \{0, 1\}$, element j in group i

Reading the table

- ▶ The table summarizes the five categories, their combinatorial objects and their natural variables
- ▶ With a suitable ground set every category fits the definition of a combinatorial optimization problem:
 - ▶ for an assignment the ground set is the set of pairs (element, resource);
 - ▶ for an ordering the set of pairs of consecutive elements.
- ▶ Assignment and partitioning differ only in the resources: distinct resources give labelled classes, identical resources give interchangeable groups (we count both in the next section)

The role of the category

The **category** of the decisions determines
the **combinatorial object** of the solutions,
the **number of candidate solutions**
and the natural **decision variables**.

Application areas (1/3)

- Combinatorial optimization problems arise in many application areas. The main ones, with their typical decisions and some examples, are:

Area	Typical decisions	Examples
production planning	what, when and how much to produce, with which resources	lot sizing, capacity planning, assembly line balancing
scheduling	order and timing of the operations on the resources	job scheduling on machines, project scheduling, maintenance scheduling

Application areas (2/3)

Area	Typical decisions	Examples
facility location	where to open facilities and which clients each one serves	warehouse location, plant location, emergency service location
packing	how to fit items into containers	cargo selection, multi-vehicle loading, pallet loading
cutting	how to cut stock material into the required pieces	cutting of bars and sheets, trim loss minimization, guillotine cutting
logistics and supply chain	routes, fleets and inventory levels	vehicle routing, fleet management, inventory management

Application areas (3/3)

Area	Typical decisions	Examples
transportation	crews, vehicles and timetables of a mobility system	crew scheduling, aircraft routing, train timetabling
personnel management	assignment of people to tasks, shifts and roles	workforce scheduling, nurse rostering, shift scheduling
network design	which nodes and arcs of a network to activate	telecommunication, energy and transportation networks

Combining categories

- ▶ A real problem often combines several categories
- ▶ For example, **facility location**:
 - ▶ decides which facilities to open (IN/OUT);
 - ▶ assigns each client to an open facility (assignment).

We now place the problems of the course in their categories.

The problems of the course by category (1/3)

The table places the problems of the course in their categories; each name links to its page.

Problem	Solution	Category
0/1 knapsack problem (KP)	subset of items	IN/OUT
subset sum problem (SSP)	subset of items	IN/OUT
bounded knapsack problem (BKP)	number of copies of each item	counting
unbounded knapsack problem (UKP)	number of copies of each item	counting
multidimensional knapsack problem (MDKP)	subset of items	IN/OUT

The problems of the course by category (2/3)

Problem	Solution	Category
multiple knapsack problem (MKP)	each item in a knapsack or none	assignment
multiple-choice knapsack problem (MCKP)	one item per class	IN/OUT
knapsack problem with setups (KPS)	subset of classes and of items	IN/OUT
bin packing problem (BPP)	partition of the items into bins	partitioning
cutting stock problem (CSP)	number of rolls cut with each pattern	configuration, counting
vertex coloring problem (VCP)	partition of the vertices into stable sets	partitioning

The problems of the course by category (3/3)

Problem	Solution	Category
maximum clique problem (MCP)	subset of vertices	IN/OUT
maximum stable set problem (MSSP)	subset of vertices	IN/OUT
travelling salesman problem (TSP)	cyclic ordering of the cities	ordering
asymmetric travelling salesman problem (ATSP)	ordering of the cities on a round trip	ordering
scheduling on identical parallel machines ($P \parallel C_{\max}$)	assignment of the jobs to the machines	assignment
capacitated lot sizing problem (CLS)	setup periods and quantities produced	IN/OUT, counting

Summary

Five categories of decisions:

- ▶ IN/OUT (subset), counting (integer quantities), assignment (function), ordering (permutation, cyclic ordering), configuration and partitioning (partition)
- ▶ The category gives the combinatorial object, the number of candidates and the natural variables

Assignment versus partitioning:

- ▶ Distinct resources give labelled classes, identical resources give interchangeable groups

Applications and problems:

- ▶ A real problem often combines several categories, as in facility location

Counting the solutions



Introduction: Counting the solutions

- ▶ Before solving a problem we ask **how many** candidate solutions it has
- ▶ The answer depends only on the combinatorial object: subsets, orderings or partitions
- ▶ These numbers tell us how far a complete enumeration can go, and why it soon becomes impossible

Proposition 1

A set with n elements has 2^n subsets.

- ▶ The proof identifies each subset with a binary vector

Proof

- ▶ A subset $S \subseteq \{1, 2, \dots, n\}$ is identified by the binary vector $\mathbf{x} \in \{0, 1\}^n$ with $x_j = 1$ if and only if $j \in S$
- ▶ Each of the n components takes 2 values independently, hence there are

$$2 \cdot 2 \cdots 2 = 2^n$$

vectors

- ▶ Equivalently, the number of subsets with k elements is $\binom{n}{k}$, and we have:

$$\sum_{k=0}^n \binom{n}{k} = 2^n$$



Subsets and the unit hypercube

- ▶ Geometrically, the 2^n binary vectors are the vertices of the unit hypercube $[0, 1]^n$:
 - ▶ the square for $n = 2$ (4 vertices);
 - ▶ the cube for $n = 3$ (8 vertices).
- ▶ Every IN/OUT decision on n elements is a subset: the 2^n subsets are its candidate solutions

Orderings and cyclic orderings

- ▶ The n elements of a set can be ordered in

$$n! = n \cdot (n - 1) \cdots 2 \cdot 1$$

ways (**permutations**)

- ▶ A **cyclic ordering** places the n elements around a circle, as in a round trip through n places: the starting element and the direction of travel do not matter

Counting the cyclic orderings

Proposition 2

For $n \geq 3$ elements, the number of distinct cyclic orderings, where a sequence, its rotations and its reverse are the same cyclic ordering, is

$$\frac{(n-1)!}{2}.$$

- The proof fixes the starting element and then removes the double count due to the direction

Proof

- ▶ Since the starting element does not change a cyclic ordering, we can always start from element 1
- ▶ The remaining $n - 1$ elements can be ordered in $(n - 1)!$ ways
- ▶ Each cyclic ordering is obtained twice, once for each direction, for example

$$(1, 3, 5, 2, 4) \quad \text{and} \quad (1, 4, 2, 5, 3)$$

- ▶ Hence the number of distinct cyclic orderings is $(n - 1)!/2$



Example

- ▶ For $n = 4$ there are $3!/2 = 3$ cyclic orderings:

$$(1, 2, 3, 4), \quad (1, 2, 4, 3), \quad (1, 3, 2, 4)$$

- ▶ Every other sequence is a rotation or a reverse of one of them
- ▶ For example, $(2, 1, 4, 3)$ is the reverse of $(1, 2, 3, 4)$ started from element 2

Partitions

- ▶ A **partition** of a finite set A is a collection of non-empty, pairwise disjoint subsets of A whose union is A
- ▶ Grouping decisions, such as splitting objects into groups, have partitions as solutions

How many partitions does a set with n elements have?

Bell numbers

- ▶ The number of partitions of a set with n elements is the **Bell number** $B(n)$, defined by the recursion:

$$B(0) = 1, \quad B(n) = \sum_{k=0}^{n-1} \binom{n-1}{k} B(k), \quad \forall n \geq 1$$

- ▶ The recursion counts the partitions by the block that contains element n :
 - ▶ if this block has $n - 1 - k$ further elements, they can be chosen in

$$\binom{n-1}{n-1-k} = \binom{n-1}{k}$$

ways;

- ▶ the remaining k elements are partitioned in $B(k)$ ways.

The first Bell numbers

- ▶ The first Bell numbers are:

n	0	1	2	3	4	5	6	7	8	9	10
$B(n)$	1	1	2	5	15	52	203	877	4140	21147	115975

- ▶ Like 2^n and $(n-1)!/2$, the Bell numbers grow very quickly with n

Assignments and interchangeable labels

- ▶ An **assignment** of n elements to m labels is a function from the elements to the labels: there are m^n of them
- ▶ If the labels are interchangeable (identical resources), two assignments that differ only by a renaming of the labels are the same solution
- ▶ The distinct solutions are the **partitions** of the elements into at most m blocks

Example

For a set with $n = 5$ elements, the numbers of the combinatorial objects are:

combinatorial object	number
subsets	$2^5 = 32$
orderings (permutations)	$5! = 120$
cyclic orderings	$4!/2 = 12$
partitions	$B(5) = 52$
assignments to 3 labels	$3^5 = 243$
partitions into at most 3 blocks	$1 + 15 + 25 = 41$

- The last line counts the partitions of 5 elements into 1, 2 or 3 blocks (1, 15 and 25 of them); the other 10 + 1 partitions of the 52 have 4 or 5 blocks

Candidate and feasible solutions

- ▶ These numbers count the **candidate** solutions
- ▶ The **feasible** ones can be far fewer: in the small instance only 11 of the 16 subsets are feasible

The number of candidates is the work of a complete enumeration:
this is the topic of the next section.

Summary

Counting formulas:

- ▶ 2^n subsets of n elements (the vertices of the unit hypercube)
- ▶ $n!$ orderings and, for $n \geq 3$, $(n-1)!/2$ cyclic orderings
- ▶ $B(n)$ partitions, computed by the recursion of the Bell numbers
- ▶ m^n assignments to m labels; with interchangeable labels, the partitions into at most m blocks

Candidates versus feasible solutions:

- ▶ The formulas count candidates: the feasible solutions can be far fewer

Combinatorial explosion and brute force

Introduction: Combinatorial explosion and brute force

- ▶ A finite feasible region can always be explored completely: this is the **brute force** algorithm
- ▶ We describe it and we run it on the small instance
- ▶ Then we measure how quickly the number of candidates grows, and how long a complete enumeration takes on a computer: this is the **combinatorial explosion**

Complete enumeration

- ▶ Since the feasible region is finite, a combinatorial optimization problem can always be solved by **complete enumeration**:
 - ▶ we list all candidate solutions;
 - ▶ we discard the infeasible ones;
 - ▶ we keep the best.
- ▶ This algorithm is called **brute force**

The brute force algorithm

Input: An instance of a combinatorial optimization problem in maximization form.

Output: An optimal solution and the optimal value.

Step 0: Initialization. Set $\bar{S} \leftarrow \text{none}$ and $\text{LB} \leftarrow -\infty$.

Step 1: Enumeration. For each candidate solution S (subset, ordering, partition):

- (a) if S is not feasible, discard it;
- (b) if S is feasible and $f(S) > \text{LB}$, set $\bar{S} \leftarrow S$ and $\text{LB} \leftarrow f(S)$.

Step 2: Output. Return $\tilde{S} = \bar{S}$ and $z(I) = \text{LB}$. STOP

Example (1/2)

For the small instance the $2^4 = 16$ subsets of $\mathcal{U}$ are:

S	$f(S)$	feasible	S	$f(S)$	feasible
$\emptyset$	0	yes	$\{2, 3\}$	10	yes
$\{1\}$	7	yes	$\{2, 4\}$	9	yes
$\{2\}$	6	yes	$\{3, 4\}$	7	yes
$\{3\}$	4	yes	$\{1, 2, 3\}$	17	no
$\{4\}$	3	yes	$\{1, 2, 4\}$	16	no
$\{1, 2\}$	13	no	$\{1, 3, 4\}$	14	no
$\{1, 3\}$	11	yes	$\{2, 3, 4\}$	13	yes
$\{1, 4\}$	10	yes	$\{1, 2, 3, 4\}$	20	no

Example (2/2)

- ▶ Among the 11 feasible subsets, the best one is

$$\tilde{S} = \{2, 3, 4\}, \quad z(I) = 13$$

- ▶ It is the only optimal solution: the subset $\{1, 2\}$ also has value 13, but it is not feasible

Brute force works for this small instance.

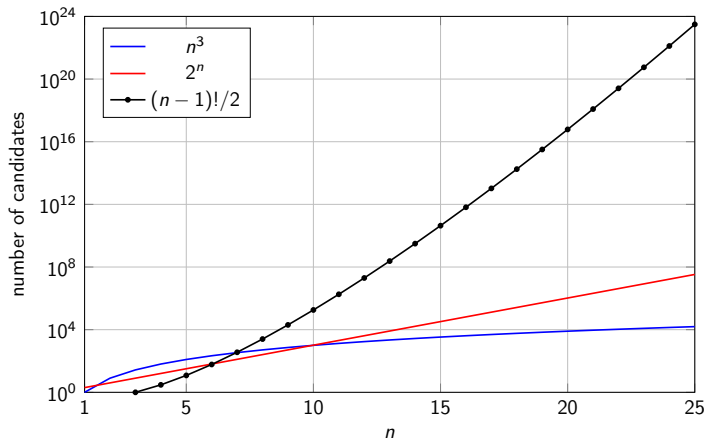
How does the number of candidates grow with n ?

Exponential growth

- The number of candidates grows very quickly with the size n of the instance. The table compares polynomial and exponential growth:

n	n^2	n^3	2^n	$(n-1)!/2$	$B(n)$
5	25	125	32	12	52
10	100	1 000	1 024	181 440	115 975
15	225	3 375	32 768	$\approx 4.36 \cdot 10^{10}$	$\approx 1.38 \cdot 10^9$
20	400	8 000	1 048 576	$\approx 6.08 \cdot 10^{16}$	$\approx 5.17 \cdot 10^{13}$
25	625	15 625	$\approx 3.36 \cdot 10^7$	$\approx 3.10 \cdot 10^{23}$	$\approx 4.64 \cdot 10^{18}$
30	900	27 000	$\approx 1.07 \cdot 10^9$	$\approx 4.42 \cdot 10^{30}$	$\approx 8.47 \cdot 10^{23}$

Exponential growth: logarithmic scale



Computing times: subsets

- We assume that a computer evaluates 10^9 candidate solutions per second, and we compute the time of a complete enumeration (one year = 365.25 days):

n	2^n	time at 10^9 per second	time at 10^{12} per second
10	1 024	≈ 1.02 microseconds	≈ 1.02 nanoseconds
20	$\approx 1.05 \cdot 10^6$	≈ 1.05 milliseconds	≈ 1.05 microseconds
30	$\approx 1.07 \cdot 10^9$	≈ 1.07 seconds	≈ 1.07 milliseconds
40	$\approx 1.10 \cdot 10^{12}$	≈ 18.3 minutes	≈ 1.10 seconds
50	$\approx 1.13 \cdot 10^{15}$	≈ 13.0 days	≈ 18.8 minutes
60	$\approx 1.15 \cdot 10^{18}$	≈ 36.5 years	≈ 13.3 days
70	$\approx 1.18 \cdot 10^{21}$	$\approx 3.74 \cdot 10^4$ years	≈ 37.4 years
80	$\approx 1.21 \cdot 10^{24}$	$\approx 3.83 \cdot 10^7$ years	$\approx 3.83 \cdot 10^4$ years

Computing times: cyclic orderings and partitions

cyclic orderings: $(n - 1)!/2$			
n	number	10^9 per second	10^{12} per second
10	181 440	≈ 181 microseconds	≈ 181 nanoseconds
15	$\approx 4.36 \cdot 10^{10}$	≈ 43.6 seconds	≈ 43.6 milliseconds
20	$\approx 6.08 \cdot 10^{16}$	≈ 1.93 years	≈ 16.9 hours
25	$\approx 3.10 \cdot 10^{23}$	$\approx 9.83 \cdot 10^6$ years	$\approx 9.83 \cdot 10^3$ years

partitions: $B(n)$		
n	number	10^9 per second
10	115 975	≈ 116 microseconds
15	$\approx 1.38 \cdot 10^9$	≈ 1.38 seconds
20	$\approx 5.17 \cdot 10^{13}$	≈ 14.4 hours
25	$\approx 4.64 \cdot 10^{18}$	≈ 147 years

A computer 1000 times faster

- ▶ A computer 1000 times faster does not change the picture
- ▶ **Subsets:** since $2^{10} = 1024 > 1000$, it enumerates in the same time the subsets of only about 10 more elements
- ▶ **Cyclic orderings:** the gain is even smaller
 - ▶ going from n to $n + 1$ elements multiplies their number by n ;
 - ▶ $15 \cdot 16 \cdot 17 = 4080 > 1000$, so from 15 elements the faster computer gains fewer than 3 elements.

Enumeration is not enough

- ▶ Complete enumeration is possible only for **very small instances**
- ▶ Faster computers do not help: an exponential number of candidates defeats any constant speed-up
- ▶ We need smarter approaches, which **exclude** whole parts of the solution space without exploring them

Summary

Brute force:

- ▶ Lists all candidates, discards the infeasible ones, keeps the best: it is exact
- ▶ On the small instance it examines 16 subsets and finds $\tilde{S} = \{2, 3, 4\}$, $z(I) = 13$

Combinatorial explosion:

- ▶ 2^n , $(n-1)!/2$ and $B(n)$ grow much faster than any polynomial
- ▶ At 10^9 candidates per second, 2^{60} subsets take about 36.5 years
- ▶ A constant speed-up gains only a few elements

Primal bounds, dual bounds and gap

Introduction: Primal bounds, dual bounds and gap

- ▶ The smarter approaches rest on a simple idea: instead of computing $z(I)$ directly, we **enclose** it between two numbers
- ▶ One comes from a feasible solution, the other from an argument that holds for *all* feasible solutions
- ▶ The distance between the two numbers, the **gap**, measures how far we are from a proof of optimality

Definition 2: primal and dual bounds

Consider an instance I of a combinatorial optimization problem with feasible solutions $\mathcal{F}$, objective function f and optimal value $z(I)$.

1. A **primal bound** is the value $f(\bar{S})$ of a feasible solution $\bar{S} \in \mathcal{F}$.
2. A **dual bound** is a number β such that

$$f(S) \leq \beta, \quad \forall S \in \mathcal{F} \quad (\text{maximization form}),$$

or such that

$$f(S) \geq \beta, \quad \forall S \in \mathcal{F} \quad (\text{minimization form}).$$

Lower and upper bounds

- ▶ In the **maximization form** a primal bound is a **lower bound** and a dual bound is an **upper bound**:

$$\text{LB} = f(\bar{S}) \leq z(I) \leq \text{UB}$$

- ▶ In the **minimization form** the roles are exchanged: a primal bound is an **upper bound** $\text{UB} = f(\bar{S})$ and a dual bound is a **lower bound** LB , with

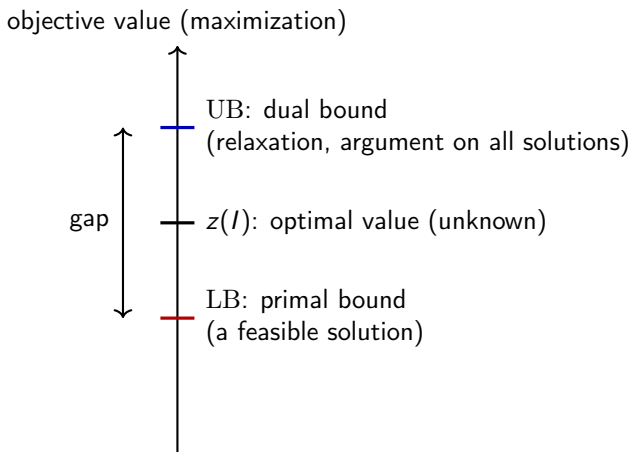
$$\text{LB} \leq z(I) \leq \text{UB}$$

Where the bounds come from

- ▶ Primal bounds come from feasible solutions, for example found by **heuristic algorithms**
- ▶ Dual bounds come from **relaxations** (for example the linear programming relaxation) or from combinatorial arguments

The next figure shows the two bounds in the maximization form:
the optimal value is enclosed between them.

Bounds in the maximization form



Definition 3: gap

Let $LB \leq z(I) \leq UB$ be a lower and an upper bound of an instance I , one primal and one dual. The **absolute gap** is $UB - LB$ and, for $LB \neq 0$, the **relative gap** (or simply **gap**) is

$$gap(I) = \frac{UB - LB}{|LB|}.$$

Reading the gap

- ▶ The denominator is always the **lower bound**:
 - ▶ in the maximization form it is the value of the feasible solution, as in the optimality gap of the module *Integer linear programming*, chapter «[Primal and dual bounds](#)»;
 - ▶ in the minimization form it is the dual bound.
- ▶ With this choice the gap is never smaller than the relative error $|f(\bar{S}) - z(I)|/z(I)$ of the feasible solution $\bar{S}$: a gap of 5% guarantees a solution within 5% of the optimum
- ▶ If the gap is zero, the feasible solution $\bar{S}$ is optimal and the dual bound is a **certificate** of its optimality

Integer objective values

Observation 1

If $f(S) \in \mathbb{Z}$ for every $S \in \mathcal{F}$, then in the maximization form $\lfloor \beta \rfloor$ is also a dual bound, and in the minimization form $\lceil \beta \rceil$ is also a dual bound.

- The proof uses only the definition of the integer part

Proof

- ▶ In the maximization form, for every $S \in \mathcal{F}$ we have

$$f(S) \leq \beta \quad \text{and} \quad f(S) \in \mathbb{Z}$$

- ▶ The largest integer not greater than β is $\lfloor \beta \rfloor$, hence

$$f(S) \leq \lfloor \beta \rfloor, \quad \forall S \in \mathcal{F}$$

- ▶ The minimization form is analogous



Example (1/3)

We consider again the small instance (maximization), and we suppose that we have not enumerated its subsets.

- ▶ *Primal bound.* We start from $\emptyset$ and examine the elements by non-increasing weight, adding each one if the set stays feasible:
 - ▶ $\{1\}$ is feasible, $\{1, 2\}$ is not;
 - ▶ $\{1, 3\}$ is feasible, $\{1, 3, 4\}$ is not.

The feasible solution $\bar{S} = \{1, 3\}$ gives $LB = 11$

- ▶ *A first dual bound.* The weights are positive, so

$$f(S) \leq f(\mathcal{U}) = 20, \quad \forall S \in \mathcal{F}$$

Example (2/3)

- ▶ *A better dual bound.* No feasible subset contains both 1 and 2, so a feasible subset takes at most one of the weights 7 and 6:

$$f(S) \leq \max\{7, 6\} + 4 + 3 = 14, \quad \forall S \in \mathcal{F}$$

The absolute gap is $14 - 11 = 3$ and the relative gap is $\frac{3}{11} \approx 27.3\%$

- ▶ *A better primal bound.* The feasible solution $\{2, 3, 4\}$ gives $\text{LB} = 13$: the gap drops to

$$\frac{1}{13} \approx 7.7\%$$

Example (3/3)

- ▶ *Closing the gap.* The only feasible subset with 3 elements is $\{2, 3, 4\}$, and a subset with at most 2 elements has value at most $7 + 6 = 13$. Hence

$$UB = 13 = LB$$

- ▶ The gap is zero, and this argument is a certificate that $\{2, 3, 4\}$ is optimal, without listing the 16 subsets
- ▶ If an argument gave instead the dual bound 13.5, then $\lfloor 13.5 \rfloor = 13$ would also be a dual bound, since all values $f(S)$ are integer (by the observation on integer objective values)

Bounds in the rest of the course

- ▶ In the rest of the course, every chapter on a problem records, for its instances, the best primal and dual bounds found so far
- ▶ Every method improves one side or the other
- ▶ On the small instance we went from $LB = 11$, $UB = 14$ to $LB = UB = 13$



Summary

Primal and dual bounds:

- ▶ A primal bound is the value of a feasible solution; a dual bound holds for all feasible solutions
- ▶ Maximization: $LB = f(\bar{S}) \leq z(I) \leq UB$; minimization: the roles are exchanged

Gap:

- ▶ Absolute gap $UB - LB$, relative gap $(UB - LB)/|LB|$
- ▶ A zero gap certifies that the feasible solution is optimal
- ▶ With integer objective values a dual bound can be rounded

Small instance: from $LB = 11$, $UB = 14$ to $LB = UB = 13$

A glimpse of computational complexity

Introduction: A glimpse of computational complexity

- ▶ Brute force takes exponential time. Is there an algorithm that solves a combinatorial optimization problem in polynomial time?
- ▶ Computational complexity classifies problems according to this question
- ▶ Here we give only the definitions needed in the course

Size of an instance and polynomial algorithms

- ▶ The **size** of an instance is the number of bits needed to write it
 - ▶ an instance with n numbers, the largest of which is M , has size of the order of $n \log_2 M$;
 - ▶ a number M is written with about $\log_2 M$ bits, not with M symbols.
- ▶ An algorithm is **polynomial** if its running time is bounded by a polynomial in the size of the instance
- ▶ The **decision version** of an optimization problem asks a yes/no question, for example: “is there a feasible solution with value at least k ?”

Definition 4: the classes P and NP

1. **P** is the class of decision problems that can be solved by a polynomial algorithm.
2. **NP** is the class of decision problems whose “yes” instances admit a certificate that can be checked in polynomial time (for example, a feasible solution whose feasibility and value are computed in polynomial time).
3. A decision problem in NP is **NP-complete** if every problem in NP can be reduced to it in polynomial time. An optimization problem is **NP-hard** if its decision version is NP-complete (more generally, if a polynomial algorithm for it would give a polynomial algorithm for an NP-complete problem).

Is P equal to NP ?

- ▶ It is not known whether $P = NP$
- ▶ No polynomial algorithm is known for any NP -complete problem, and it is widely believed that none exists
- ▶ For an NP -hard problem we do not expect a polynomial exact algorithm

Polynomial in the values of the numbers

- ▶ Some NP-hard problems can be solved in time polynomial in n **and in the value** of the numbers of the instance
- ▶ For example, an algorithm that takes $O(n M)$ time, where M is the largest number of the instance, is not polynomial: M is exponential in its number of bits $\log_2 M$

Definition 5: pseudo-polynomial algorithm, weak and strong NP-hardness

1. An algorithm is **pseudo-polynomial** if its running time is bounded by a polynomial in the size of the instance and in the largest number appearing in it.
2. An NP-hard problem is **strongly NP-hard** if it remains NP-hard when all its numbers are bounded by a polynomial in the size of the instance (equivalently, when the numbers are written in unary). Otherwise, if it admits a pseudo-polynomial algorithm, it is called **weakly NP-hard** (NP-hard in the ordinary sense).

- A strongly NP-hard problem has no pseudo-polynomial algorithm, unless $P = NP$

NP-hard does not mean unsolvable

- ▶ NP-hardness does not mean that an instance cannot be solved: it means that no algorithm is expected to solve *every* instance in polynomial time
- ▶ In practice we combine:
 - ▶ **exact methods** that close the gap between primal and dual bounds;
 - ▶ **heuristics** that find good solutions quickly;
 - ▶ **approximation algorithms** that guarantee a bounded gap in the worst case.

The complexity of the problems of the course

- ▶ The problems studied later in the course are all NP-hard:
 - ▶ the 0/1 knapsack problem is **weakly** NP-hard;
 - ▶ bin packing, vertex coloring, maximum clique and the travelling salesman problem are **strongly** NP-hard.
- ▶ Each chapter discusses the complexity of its problem

Summary

Size and polynomial algorithms:

- ▶ The size of an instance is its number of bits: a number M takes about $\log_2 M$ bits

Classes:

- ▶ P, NP, NP-complete; an optimization problem is NP-hard if its decision version is NP-complete
- ▶ Pseudo-polynomial algorithms; weakly and strongly NP-hard problems

In practice:

- ▶ Exact methods, heuristics and approximation algorithms, combined

Typical mistakes

Introduction: Typical mistakes

- ▶ In this section we collect the typical mistakes on the notions of the chapter
- ▶ They concern the counting of the solutions, the direction and the rounding of the bounds, and the meaning of NP-hardness
- ▶ Recognizing them helps to read correctly the rest of the course

Typical mistakes (1/3)

Observation 2: typical mistakes (1/3)

- ▶ ***Confusing candidate and feasible solutions.*** The numbers 2^n , $(n-1)!/2$ and $B(n)$ count the candidates; the feasible solutions can be far fewer, for example 11 subsets out of 16 in the small instance.
- ▶ ***Exchanging the two directions of the bounds.*** In a maximization problem a primal bound is a lower bound and a dual bound is an upper bound; in a minimization problem the roles are exchanged.

Typical mistakes (2/3)

Observation 3: typical mistakes (2/3)

- ▶ ***Rounding a dual bound when the objective is not integer.** The rounding of the observation on integer objective values needs $f(S) \in \mathbb{Z}$ for every feasible solution.*
- ▶ ***Counting equivalent solutions twice.** A cyclic ordering and its reverse are the same cyclic ordering, and two assignments that differ by a renaming of interchangeable labels are the same solution.*

Typical mistakes (3/3)

Observation 4: typical mistakes (3/3)

- ▶ *Reading NP-hard as unsolvable. NP-hardness is a statement on the worst case: many instances of practical size are solved, and the rest of the course shows how.*

Summary

Counting:

- ▶ Candidates are not feasible solutions; equivalent solutions are counted once

Bounds:

- ▶ Primal and dual exchange their direction between maximization and minimization
- ▶ A dual bound is rounded only if every objective value is integer

Complexity:

- ▶ NP-hard is a statement on the worst case, not on every instance

In summary

Introduction: In summary

- ▶ We collect the key ideas of the chapter: definition, categories, counting, bounds and complexity
- ▶ We close with the dashboard of the small instance, which records its primal and dual bounds
- ▶ Every chapter on a problem will keep such a dashboard for its instances

Summary (1/3)

- ▶ A combinatorial optimization problem has a finite ground set, a family of feasible subsets and an objective function; an optimal solution exists whenever a feasible solution exists
- ▶ The category of the decisions gives the combinatorial object of a solution: a subset, a vector of integer quantities, an assignment, an ordering or a partition
- ▶ The candidate solutions are 2^n subsets, $n!$ orderings, $(n-1)!/2$ cyclic orderings and $B(n)$ partitions: brute force works only on very small instances

Summary (2/3)

- ▶ A feasible solution gives a primal bound, a relaxation or a combinatorial argument gives a dual bound; a zero gap certifies that the feasible solution is optimal
- ▶ The problems of the course are NP-hard: no polynomial exact algorithm is expected for them



Summary (3/3)

Dashboard of the small instance.

small instance (max)	lower bound	upper bound	from
first bounds	11	14	greedy choice, one of the weights 7 and 6
final bounds	13	13	solution $\{2, 3, 4\}$, argument by cardinality