



SAPIENZA
UNIVERSITÀ DI ROMA

Combinatorial Optimization

Volume 1 — Combinatorial optimization problems

Combinatorial optimization problems and their families: knapsack, packing, coloring, clique, routing, scheduling and lot sizing problems. For each problem: definition, assumptions, instances, counting the solutions, a first solution by hand and a simple bound.

Fabio Furini

fabiofurini.github.io

Department of Computer, Control and Management Engineering — Sapienza University of Rome

Online version and downloadable material:
fabiofurini.github.io/combinatorial-optimization/
github.com/fabiofurini/combinatorial-optimization

Academic year 2026–2027

Contents

1	Combinatorial optimization problems	4
1.1	Combinatorial optimization problems	5
1.2	Categories of combinatorial optimization problems	6
1.3	Counting the solutions	10
1.4	Combinatorial explosion and brute force	12
1.5	Primal bounds, dual bounds and gap	15
1.6	A glimpse of computational complexity	17
1.7	Typical mistakes	18
1.8	In summary	19
2	Knapsack problems	20
2.1	Overview of the family	21
2.2	The 0/1 knapsack problem	22
2.3	The subset sum problem	25
2.4	The bounded knapsack problem	27
2.5	The unbounded knapsack problem	30
2.6	The multidimensional knapsack problem	32
2.7	The multiple knapsack problem	35
2.8	The multiple-choice knapsack problem	38
2.9	The knapsack problem with setups	40
2.10	Remarks	43
2.11	Typical mistakes	43
2.12	In summary	44
3	Packing problems	45
3.1	Overview of the family	46
3.2	The bin packing problem	46
3.3	The cutting stock problem	49
3.4	Remarks	52
3.5	Typical mistakes	52
3.6	In summary	53

4	Coloring problems	54
4.1	Overview of the family	55
4.2	The vertex coloring problem	56
4.3	Remarks	60
4.4	Typical mistakes	60
4.5	In summary	61
5	Clique problems	62
5.1	Overview of the family	62
5.2	The maximum clique problem	63
5.3	Counting the solutions, a first solution and the bounds	66
5.4	Complexity	68
5.5	Weighted variants	68
5.6	Remarks	70
5.7	Typical mistakes	70
5.8	In summary	70
6	Routing problems	72
6.1	Overview of the family	73
6.2	The travelling salesman problem	73
6.3	Counting the tours	75
6.4	A first tour by hand	76
6.5	A combinatorial lower bound	77
6.6	Variants: metric and asymmetric TSP	78
6.7	Remarks	79
6.8	Typical mistakes	80
6.9	In summary	80
7	Scheduling problems	81
7.1	Overview of the family	82
7.2	Scheduling on identical parallel machines	83
7.3	Counting the solutions	85
7.4	A first schedule by hand	85
7.5	A combinatorial lower bound	86
7.6	Relation with the bin packing problem	87
7.7	Remarks	87
7.8	Typical mistakes	87
7.9	In summary	88
8	Lot sizing problems	89
8.1	Overview of the family	89
8.2	The capacitated lot sizing problem	90
8.3	Counting the solutions	92

8.4	A first plan by hand	93
8.5	A combinatorial lower bound	94
8.6	Variants	94
8.7	Remarks	96
8.8	Typical mistakes	96
8.9	In summary	97

Combinatorial optimization problems

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.

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.

1.1 Combinatorial optimization problems

Definition 1.1: 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

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

- 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.

- 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 .

Example 1.1: a small instance

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 this instance [Example 1.1](#).

- Here $\mathcal{F}$ is listed. In the problems of the course it is never listed: it is described by a rule, for example a capacity constraint, and it is far too large to be written down.

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

- 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** (Section 1.6);
 4. specialized algorithms are needed to explore the solution space without enumerating it.

1.2 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.

1.2.1 Categories of decisions

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.
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.

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.

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

- The table summarizes the five categories, their combinatorial objects and their natural variables.
- With a suitable ground set every category fits Definition 1.1: 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 (Section 1.3).

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

1.2.2 Application areas

- 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
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
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

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

1.2.3 The problems of the course by category

- 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
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
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 ($P C_{\max}$)	assignment of the jobs to the machines	assignment
capacitated lot sizing problem (CLS)	setup periods and quantities produced	IN/OUT, counting

1.3 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.

1.3.1 Subsets

Proposition 1.2

A set with n elements has 2^n subsets.

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:

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

□

- Geometrically, the 2^n binary vectors are the vertices of the unit hypercube $[0, 1]^n$: the square for $n = 2$ (4 vertices) and 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.

1.3.2 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.

Proposition 1.3

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

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

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 $(1, 3, 5, 2, 4)$ and $(1, 4, 2, 5, 3)$. Hence the number of distinct cyclic orderings is $(n - 1)!/2$. □

- For $n = 4$ there are $3!/2 = 3$ cyclic orderings: $(1, 2, 3, 4)$, $(1, 2, 4, 3)$ and $(1, 3, 2, 4)$. 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.

1.3.3 Partitions and Bell numbers

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

$$B(0) = 1, \quad B(n) = \sum_{k=0}^{n-1} \binom{n-1}{k} B(k), \quad \forall n \geq 1. \quad (1.1)$$

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, and the remaining k elements are partitioned in $B(k)$ ways.

- 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

- 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 1.2: how many candidate solutions

For a set with $n = 5$ elements, the numbers of the four 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.

- These numbers count the **candidate** solutions. The feasible ones can be far fewer: in [Example 1.1](#) only 11 of the 16 subsets are feasible.

1.4 Combinatorial explosion and brute force

- Since the feasible region is finite, a combinatorial optimization problem can always be solved by **complete enumeration**: we list all candidate solutions, discard the infeasible ones and keep the best. This algorithm is called **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 $\tilde{S} \leftarrow \text{none}$ 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 $\tilde{S} \leftarrow S$ and $\text{LB} \leftarrow f(S)$.

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

Example 1.3: brute force on the small instance

For the instance of [Example 1.1](#) the $2^4 = 16$ subsets of $\mathcal{U}$ are:

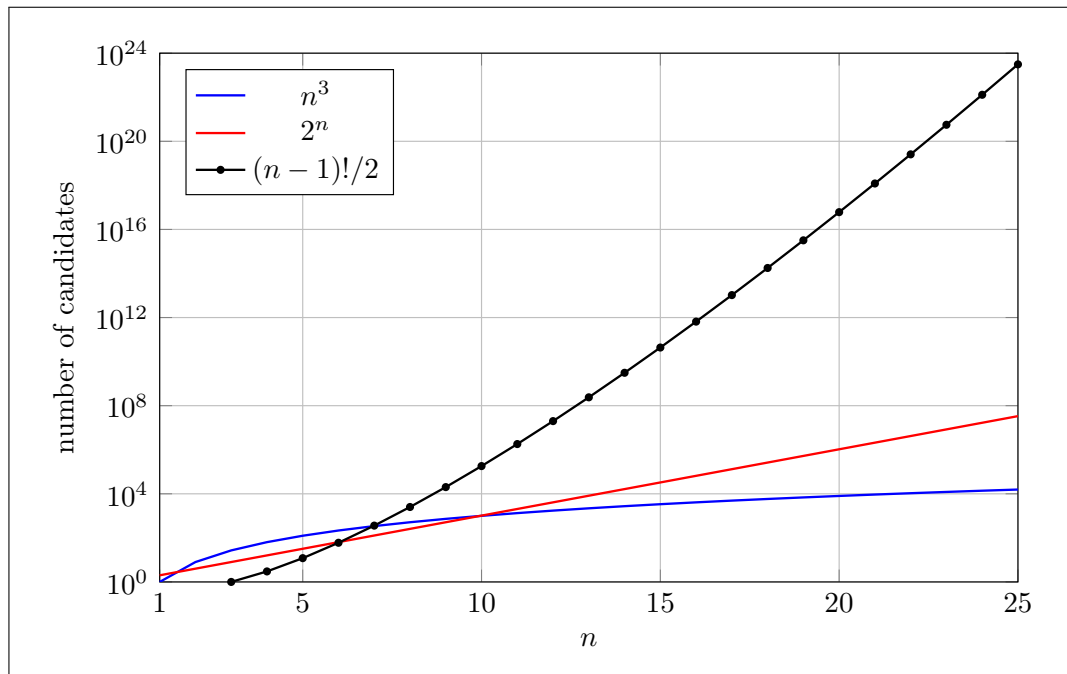
S	$f(S)$	feasible	S	$f(S)$	feasible
$\emptyset$	0	yes	$\{2, 3\}$	10	yes
$\{1\}$	7	yes	$\{2, 4\}$	9	yes
$\{2\}$	6	yes	$\{3, 4\}$	7	yes
$\{3\}$	4	yes	$\{1, 2, 3\}$	17	no
$\{4\}$	3	yes	$\{1, 2, 4\}$	16	no
$\{1, 2\}$	13	no	$\{1, 3, 4\}$	14	no
$\{1, 3\}$	11	yes	$\{2, 3, 4\}$	13	yes
$\{1, 4\}$	10	yes	$\{1,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

Among the 11 feasible subsets, the best one is $\tilde{S} = \{2, 3, 4\}$, with $z(I) = 13$. It is the only optimal solution: the subset $\{1, 2\}$ also has value 13, but it is not feasible.

1.4.1 Exponential growth

- Brute force works for the small instances above, but the number of candidates grows very quickly with the size n of the instance. The following 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	$\approx 1.07 \cdot 10^9$	$\approx 4.42 \cdot 10^{30}$	$\approx 8.47 \cdot 10^{23}$



1.4.2 Computing times

- 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

n	number	cyclic orderings: $(n-1)!/2$		partitions: $B(n)$	
		10^9 per second	10^{12} per second	number	10^9 per second
10	181 440	≈ 181 microseconds	≈ 181 nanoseconds	115 975	≈ 116 microseconds
15	$\approx 4.36 \cdot 10^{10}$	≈ 43.6 seconds	≈ 43.6 milliseconds	$\approx 1.38 \cdot 10^9$	≈ 1.38 seconds
20	$\approx 6.08 \cdot 10^{16}$	≈ 1.93 years	≈ 16.9 hours	$\approx 5.17 \cdot 10^{13}$	≈ 14.4 hours
25	$\approx 3.10 \cdot 10^{23}$	$\approx 9.83 \cdot 10^6$ years	$\approx 9.83 \cdot 10^3$ years	$\approx 4.64 \cdot 10^{18}$	≈ 147 years

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

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.

1.5 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.

Definition 1.4: primal and dual bounds

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

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

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

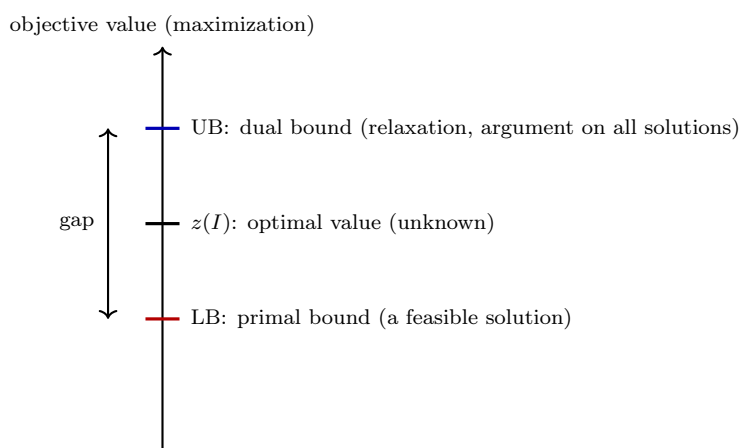
or such that

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

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



Definition 1.5: gap

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

gap) is

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

- 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 2 «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.

Observation 1.6

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.

Proof. In the maximization form, for every $S \in \mathcal{F}$ we have $f(S) \leq \beta$ and $f(S) \in \mathbb{Z}$; the largest integer not greater than β is $\lfloor \beta \rfloor$, hence $f(S) \leq \lfloor \beta \rfloor$. The minimization form is analogous. $\square$

Example 1.4: bounds on the small instance

We consider again the instance of Example 1.1 (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 $\text{LB} = 11$.
- *A first dual bound.* The weights are positive, so $f(S) \leq f(\mathcal{U}) = 20$ for every $S \in \mathcal{F}$.
- *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 $\frac{1}{13} \approx 7.7\%$.
- *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 $\text{UB} = 13 = \text{LB}$: 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 (Observation 1.6).

- 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.

1.6 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.
- The **size** of an instance is the number of bits needed to write it. For example, 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 1.7: 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).

- 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.

1.6.1 Pseudo-polynomial algorithms and strong NP-hardness

- 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 $O(nM)$ 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 1.8: 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-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, and **approximation algorithms** that guarantee a bounded gap in the worst case.

- The problems of the next chapters are all NP-hard: the 0/1 knapsack problem is weakly NP-hard, while bin packing, vertex coloring, maximum clique and the travelling salesman problem are strongly NP-hard. Each chapter discusses the complexity of its problem.

1.7 Typical mistakes

Warning

- **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 [Example 1.1](#).
- **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.
- **Rounding a dual bound when the objective is not integer.** The rounding of Observation 1.6 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.

- **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.

1.8 In summary

Summary

- 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.
- 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.

Dashboard of the small instance.

Example 1.1 (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

Knapsack problems

Prerequisites

- From this module, [Chapter 1 «Combinatorial optimization problems»](#): instances, feasible solutions, primal and dual bounds and the gap.
- From the *Preliminaries*: subsets of a finite set and the number 2^n of subsets.

Objectives

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

- state the definition of the 0/1 knapsack problem and of its variants, with instances and feasible solutions;
- explain the assumptions on the data and why they can be made without loss of generality;
- count the candidate solutions and the feasible solutions of a small instance;
- find a feasible solution by hand and compute a simple combinatorial dual bound;
- tell the variants apart: copies of the items, several resources, several knapsacks, classes of items.

2.1 Overview of the family

- The **knapsack problems** share the same data: a set $\mathcal{I} = \{1, 2, \dots, n\}$ of n items, for each item $j \in \mathcal{I}$ a *weight* $w_j \in \mathbb{Z}_{\geq 1}$ and a *profit* $p_j \in \mathbb{Z}_{\geq 1}$, and one or more knapsacks with a *capacity* in $\mathbb{Z}_{\geq 1}$.
- A solution decides which items are selected, or how many copies of each item: the total weight of the selected items must not exceed the capacity of the knapsack that receives them.
- The objective is to maximize the total profit of the selected items.

A knapsack problem selects items of maximum total profit whose total weight respects the capacities. The members of the family differ in the number of copies of the items, in the number of capacities and of knapsacks, and in the classes into which the items are grouped.

- The table lists the members of the family, with the section that defines each of them.

Problem	Acronym	Difference with respect to the KP	Section
0/1 knapsack problem	KP	one copy of each item, one knapsack	2.2
subset sum problem	SSP	$p_j = w_j$, total weight exactly equal to c	2.3
bounded knapsack problem	BKP	at most b_j copies of item j	2.4
unbounded knapsack problem	UKP	an unlimited number of copies of each item	2.5
multidimensional knapsack problem	MDKP	k capacities c_d and weights w_{dj}	2.6
multiple knapsack problem	MKP	m knapsacks with capacities c_k	2.7
multiple-choice knapsack problem	MCKP	classes $\mathcal{I}_h$, exactly one item per class	2.8
knapsack problem with setups	KPS	classes $\mathcal{I}_h$ with setup costs f_h and setup weights s_h	2.9

2.1.1 The variants at a glance

- Each problem of the family is illustrated on a small instance, in its section of the table above and with the notation of that section.
- The table lists, for these instances, the number of candidate solutions, the number of feasible solutions and the optimal value, computed by enumeration.
- In the BKP and in the UKP a candidate solution is the vector of the numbers of copies of every item, so that identical copies are not counted twice.
- The optimal solution is not always unique: the MKP, the MCKP and the KPS have several optimal solutions with the same value, and in the SSP three subsets reach the target.

Variant	Candidate solutions	Number	Feasible	Optimum	Optimal solutions
KP	subsets of 6 items	64	13	15	$\{1, 4\}$
SSP	subsets of 6 items	64	3	yes	$\{6\}, \{1, 4\}, \{2, 3\}$
BKP	copies $x_j \leq b_j$	144	18	18	two copies of item 1 and item 5
UKP	copies $x_j \leq \hat{b}_j$	320	22	24	four copies of item 1
MDKP	subsets of 4 items	16	9	14	$\{3, 4\}$
MKP	each item in a knapsack or none	243	109	30	$(S_1, S_2) = (\{1, 2\}, \{4, 5\}), (\{2, 4\}, \{1, 5\}), (\{4, 5\}, \{1, 2\})$
MCKP	one item per class	8	3	17	$\{1, 4, 5\}, \{2, 3, 5\}$
KPS	subsets of 6 items	64	40	16	$\{1, 2, 3\}, \{4, 5, 6\}, \{1, 2, 3, 6\}$

- In the MCKP only 3 of the 8 choices of one item per class fit in the knapsack: the feasibility condition among the assumptions of Section 2.8 is not a formality.

2.2 The 0/1 knapsack problem

Definition 2.1: 0/1 Knapsack Problem

Given:

1. a knapsack of *capacity* $c \in \mathbb{Z}_{\geq 1}$,
2. a set $\mathcal{I} = \{1, 2, \dots, n\}$ of n items,
3. for each item $j \in \mathcal{I}$, a *profit* $p_j \in \mathbb{Z}_{\geq 1}$ and a *weight* $w_j \in \mathbb{Z}_{\geq 1}$,

the **0/1 Knapsack Problem** (KP) asks to decide which items to insert into the knapsack, so that the total weight of the selected items does not exceed the knapsack capacity, and the total profit of the selected items is maximized.

- A **KP instance** is specified by:
 1. two integer numbers:
 - the number of items $n \in \mathbb{Z}_{\geq 1}$,
 - the knapsack capacity $c \in \mathbb{Z}_{\geq 1}$,
 2. two vectors of integer numbers of length n :
 - $\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, containing the item profits,
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, containing the item weights.

- A **KP feasible solution** is a subset of items

$$S \subseteq \mathcal{I}$$

such that the total weight of the selected items does not exceed the knapsack capacity, namely

$$\sum_{j \in S} w_j \leq c.$$

Accordingly, the KP can be equivalently stated as the problem of selecting a subset of items whose total weight respects the knapsack capacity, and whose total profit is maximized.

Given a KP instance, the KP can be formally stated as:

$$\tilde{S} \in \arg \max_{S \subseteq \mathcal{J}} \left\{ \sum_{j \in S} p_j : \sum_{j \in S} w_j \leq c \right\}. \quad (2.1)$$

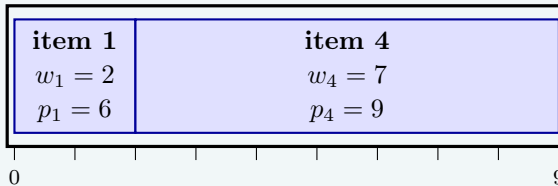
Example 2.1: KP instance

We consider a knapsack of capacity $c = 9$ and a set $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ of 6 items ($n = 6$), with the following profits and weights:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Profits	p_j	6	5	8	9	6	7
Weights	w_j	2	3	6	7	5	9

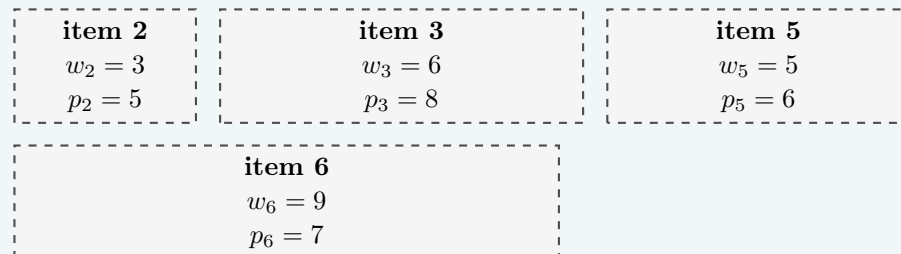
In the drawing, the knapsack is a box of length $c = 9$ and every item is a block of length w_j . The items 1 and 4 fill the knapsack, with weight 9 and profit 15; the other items stay out (dashed).

knapsack, capacity $c = 9$



weight $2 + 7 = 9$
profit $6 + 9 = 15$

not selected



We call this KP instance [Example 2.1](#): it is the reference instance of the chapter.

2.2.1 Assumptions

- We make the following assumptions:

1. The total weight of all items exceeds the knapsack capacity (otherwise, all items could be selected). Formally:

$$\sum_{j \in \mathcal{I}} w_j > c$$

2. Each item individually fits into the knapsack (otherwise, it cannot be selected). Formally:

$$w_j \leq c, \quad \forall j \in \mathcal{I}$$

2.2.2 Counting the solutions, a first solution and a simple bound

- **Candidate solutions.** Every subset of $\mathcal{I}$ is a candidate solution, so there are 2^n of them (Chapter 1 «Combinatorial optimization problems»). For Example 2.1 we have $2^6 = 64$.
- **Feasible solutions.** The three lightest items weigh $2 + 3 + 5 = 10 > 9$, so no subset with three or more items is feasible. The feasible solutions are the empty set, the 6 singletons and the 6 pairs of the following table: 13 solutions out of 64.

pair S	$\{1, 2\}$	$\{1, 3\}$	$\{1, 4\}$	$\{1, 5\}$	$\{2, 3\}$	$\{2, 5\}$
weight	5	8	9	7	9	8
profit	11	14	15	12	13	11

- **A first solution, by eye.** Item 4 has the largest profit (9, weight 7). The residual capacity $9 - 7 = 2$ is enough for item 1 (weight 2, profit 6). The subset $\{1, 4\}$ has weight 9 and profit 15: it is a **primal bound**, $LB = 15 \leq z(I)$.

Observation 2.2: cardinality bound

Let $\bar{k}$ be the largest integer such that the $\bar{k}$ lightest items weigh at most c . Every feasible subset has at most $\bar{k}$ items, hence $z(I)$ is at most the sum of the $\bar{k}$ largest profits.

Proof. A subset with k items weighs at least as much as the k lightest items. If $k > \bar{k}$, the k lightest items weigh more than c , so the subset is not feasible. The profit of a subset with at most $\bar{k}$ items is at most the sum of the $\bar{k}$ largest profits. $\square$

- **A first dual bound.** In Example 2.1 we have $\bar{k} = 2$, since $2 + 3 = 5 \leq 9$ and $2 + 3 + 5 = 10 > 9$. The two largest profits are 9 and 8, so $UB = 17$. The sum of all the profits, 41, is a much weaker dual bound.
- **The gap.** We have $15 \leq z(I) \leq 17$: the absolute gap is 2 and the relative gap is $\frac{2}{15}$. Listing the 13 feasible subsets shows that $\{1, 4\}$ is the unique optimal solution and $z(I) = 15$; for larger instances we need stronger bounds, which come from relaxations (module *Integer linear programming*, Chapter 2 «Primal and dual bounds»).

2.3 The subset sum problem

Definition 2.3: Subset Sum Problem

Given:

1. a target value $c \in \mathbb{Z}_{\geq 1}$,
2. a set $\mathcal{I} = \{1, 2, \dots, n\}$ of n items,
3. for each item $j \in \mathcal{I}$, a *weight* $w_j \in \mathbb{Z}_{\geq 1}$,

the **Subset Sum Problem** (SSP) asks whether there exists a subset of items whose total weight is exactly equal to the target value c .

- An **SSP instance** is specified by:
 1. two integer numbers:
 - the number of items $n \in \mathbb{Z}_{\geq 1}$,
 - the target value $c \in \mathbb{Z}_{\geq 1}$,
 2. one vector of integer numbers of length n :
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, containing the item weights.

- An **SSP feasible solution** is a subset of items

$$S \subseteq \mathcal{I}$$

such that the total weight of the selected items is exactly equal to the target value, namely

$$\sum_{j \in S} w_j = c.$$

Accordingly, the SSP can be equivalently stated as the problem of selecting a subset of items whose total weight matches the target value exactly.

Given an SSP instance, the SSP can be formally stated as:

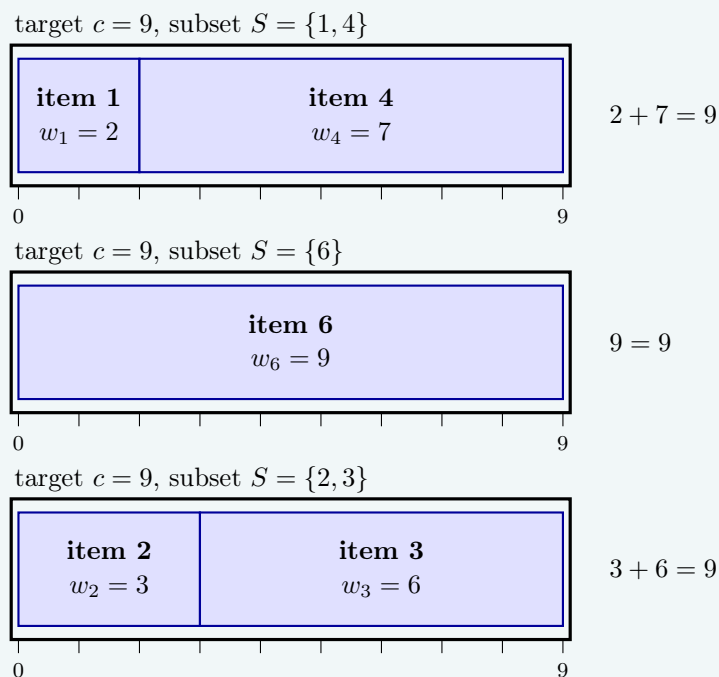
$$\text{find } S \subseteq \mathcal{I} \quad \text{such that} \quad \sum_{j \in S} w_j = c. \quad (2.2)$$

Example 2.2: SSP instance

We consider a target value $c = 9$ and a set $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ of 6 items ($n = 6$), with the following weights:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Weights	w_j	2	3	6	7	5	9

For instance, the subset $S = \{1, 4\}$ is feasible since $w_1 + w_4 = 2 + 7 = 9$. Also, the subset $S = \{6\}$ is feasible since $w_6 = 9$. The drawing shows the three subsets whose weights fill the box of length $c = 9$ exactly: $\{1, 4\}$, $\{6\}$ and $\{2, 3\}$.



- The SSP is not an **optimization problem**, since its answer is either **yes** or **no**. However, it can be solved by using an optimization problem.
- An SSP instance can be solved as a KP instance by setting the profit of each item equal to its weight, namely

$$p_j = w_j, \quad \forall j \in \mathcal{J}.$$

In this case, we solve the following KP:

$$\tilde{S} \in \arg \max_{S \subseteq \mathcal{J}} \left\{ \sum_{j \in S} w_j : \sum_{j \in S} w_j \leq c \right\}.$$

If $\sum_{j \in \tilde{S}} w_j = c$, then the answer to the SSP is **yes**; otherwise, if $\sum_{j \in \tilde{S}} w_j < c$, the answer is **no**.

2.3.1 Assumptions

- We make the following assumptions:
 1. The total weight of all items is at least the target value (otherwise, no feasible subset can exist). Formally:

$$\sum_{j \in \mathcal{I}} w_j \geq c$$

2. Each item weight is not larger than the target value (otherwise, it cannot belong to any feasible subset). Formally:

$$w_j \leq c, \quad \forall j \in \mathcal{I}$$

2.4 The bounded knapsack problem

Definition 2.4: Bounded Knapsack Problem

Given:

1. a knapsack of *capacity* $c \in \mathbb{Z}_{\geq 1}$,
2. a set $\mathcal{I} = \{1, 2, \dots, n\}$ of n items,
3. for each item $j \in \mathcal{I}$, a *profit* $p_j \in \mathbb{Z}_{\geq 1}$, a *weight* $w_j \in \mathbb{Z}_{\geq 1}$, and a *number of available copies* $b_j \in \mathbb{Z}_{\geq 1}$,

the **Bounded Knapsack Problem** (BKP) asks to decide how many copies of each item to insert into the knapsack, so that the total weight of all selected item copies does not exceed the knapsack capacity, each item is taken no more than its available copies, and the total profit of all selected item copies is maximized.

- A **BKP instance** is specified by:
 1. two integer numbers:
 - the number of items $n \in \mathbb{Z}_{\geq 1}$,
 - the knapsack capacity $c \in \mathbb{Z}_{\geq 1}$,
 2. three vectors of integer numbers of length n :
 - $\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, containing the item profits,
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, containing the item weights,
 - $\mathbf{b} \in \mathbb{Z}_{\geq 1}^n$, containing the number of available copies of each item.

- Given a BKP instance, we define the **extended set of items** as

$$\hat{\mathcal{I}} = \{ (j, k) : j \in \mathcal{I}, k \in \{1, 2, \dots, b_j\} \}, \quad (2.3)$$

where each pair (j, k) represents the k -th available copy of item j . The cardinality of the extended set of items is

$$|\hat{\mathcal{I}}| = \sum_{j \in \mathcal{I}} b_j.$$

- A **BKP feasible solution** is a subset of items

$$S \subseteq \mathcal{I}$$

such that the total weight of the selected items does not exceed the knapsack capacity, namely

$$\sum_{(j,k) \in S} w_j \leq c.$$

Accordingly, the BKP can be equivalently stated as the problem of selecting a subset of the extended set of items whose total weight respects the knapsack capacity, and whose total profit is maximized.

Given a BKP instance, the BKP can be formally stated as:

$$\tilde{S} \in \arg \max_{S \subseteq \tilde{\mathcal{I}}} \left\{ \sum_{(j,k) \in S} p_j : \sum_{(j,k) \in S} w_j \leq c \right\}. \quad (2.4)$$

Example 2.3: BKP instance

We consider a knapsack of capacity $c = 9$ and a set $\mathcal{I} = \{1, 2, 3, 4, 5, 6\}$ of 6 items ($n = 6$), with the following profits, weights, and available copies:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Profits	p_j	6	5	8	9	6	7
Weights	w_j	2	3	6	7	5	9
Available copies	b_j	2	2	1	1	1	1

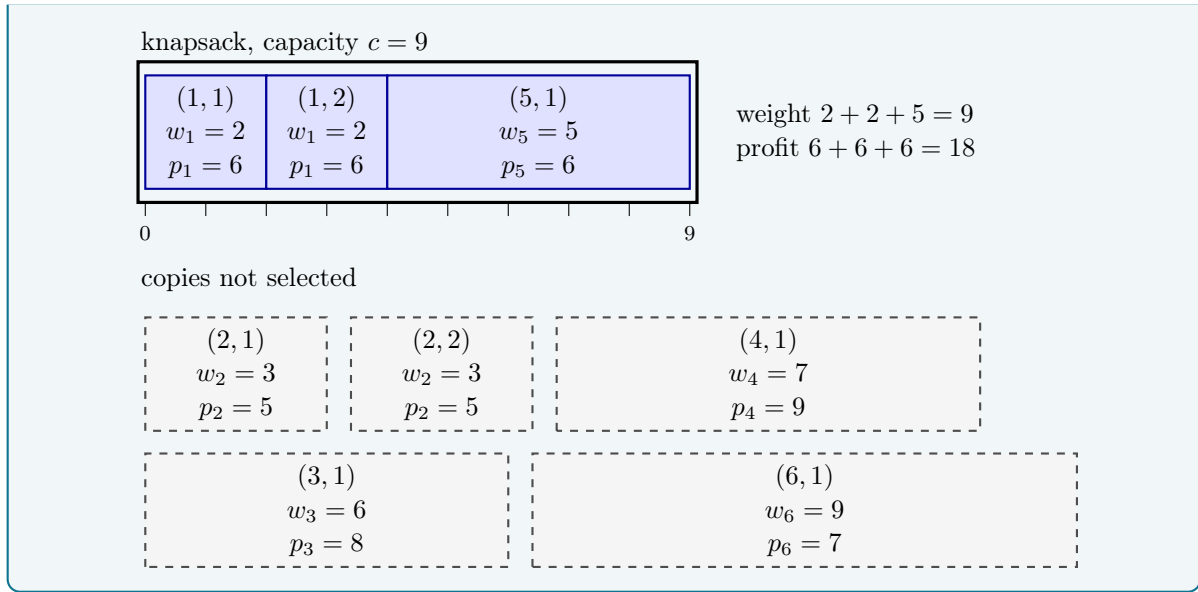
The extended set of items is:

$$\tilde{\mathcal{I}} = \{ (1,1), (1,2), (2,1), (2,2), (3,1), (4,1), (5,1), (6,1) \}.$$

Each copy (j,k) has profit p_j and weight w_j .

Copy label	1	2	3	4	5	6	7	8
(j,k)	(1,1)	(1,2)	(2,1)	(2,2)	(3,1)	(4,1)	(5,1)	(6,1)
Profit p_j	6	6	5	5	8	9	6	7
Weight w_j	2	2	3	3	6	7	5	9

The drawing shows an optimal solution: the two copies of item 1 and the copy of item 5, with weight $2 + 2 + 5 = 9$ and profit $6 + 6 + 6 = 18$.



2.4.1 Assumptions

- We make the following assumptions:

1. The total weight of all available copies of all items exceeds the knapsack capacity (otherwise, all copies could be selected). Formally:

$$\sum_{j \in \mathcal{I}} b_j w_j > c$$

2. Each item individually fits into the knapsack (otherwise, it cannot be selected). Formally:

$$w_j \leq c, \quad \forall j \in \mathcal{I}$$

3. The total weight of all available copies of any single item does not exceed the knapsack capacity (otherwise, that item could never be taken in the maximum allowed number of copies). Formally:

$$b_j w_j \leq c, \quad \forall j \in \mathcal{I}$$

2.5 The unbounded knapsack problem

Definition 2.5: Unbounded Knapsack Problem

Given:

1. a knapsack of *capacity* $c \in \mathbb{Z}_{\geq 1}$,
2. a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n items,
3. for each item $j \in \mathcal{J}$, which is available in an unlimited number of copies, a *profit* $p_j \in \mathbb{Z}_{\geq 1}$ and a *weight* $w_j \in \mathbb{Z}_{\geq 1}$,

the **Unbounded Knapsack Problem** (UKP) asks to decide how many copies of each item to insert into the knapsack, so that the total weight of all selected item copies does not exceed the knapsack capacity and the total profit of all selected item copies is maximized.

- A **UKP instance** is specified by:

1. two integer numbers:
 - the number of items $n \in \mathbb{Z}_{\geq 1}$,
 - the knapsack capacity $c \in \mathbb{Z}_{\geq 1}$,
2. two vectors of integer numbers of length n :
 - $\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, containing the item profits,
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, containing the item weights.

- Given a UKP instance, we first define for each item $j \in \mathcal{J}$ the **maximum feasible number of copies**

$$\hat{b}_j = \left\lfloor \frac{c}{w_j} \right\rfloor, \quad (2.5)$$

that is, the largest number of copies of item j that can be inserted into the knapsack without exceeding its capacity.

- We then define the **extended set of items** as

$$\hat{\mathcal{J}} = \{ (j, k) : j \in \mathcal{J}, k \in \{1, 2, \dots, \hat{b}_j\} \}, \quad (2.6)$$

where each pair (j, k) represents the k -th copy of item j . The cardinality of the extended set of items is

$$|\hat{\mathcal{J}}| = \sum_{j \in \mathcal{J}} \hat{b}_j.$$

- A **UKP feasible solution** is a subset of items

$$S \subseteq \hat{\mathcal{J}}$$

such that the total weight of the selected items does not exceed the knapsack capacity, namely

$$\sum_{(j,k) \in S} w_j \leq c.$$

Accordingly, the UKP can be equivalently stated as the problem of selecting a subset of the extended set of items whose total weight respects the knapsack capacity, and whose total profit is maximized.

Given a UKP instance, the UKP can be formally stated as:

$$\tilde{S} \in \arg \max_{S \subseteq \hat{\mathcal{J}}} \left\{ \sum_{(j,k) \in S} p_j : \sum_{(j,k) \in S} w_j \leq c \right\}. \quad (2.7)$$

Example 2.4: UKP instance

We consider a knapsack of capacity $c = 9$ and a set $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ of 6 items ($n = 6$), with the following profits and weights:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Profits	p_j	6	5	8	9	6	7
Weights	w_j	2	3	6	7	5	9

The maximum feasible number of copies of each item are:

$$\hat{b}_1 = \underbrace{\lfloor \frac{9}{2} \rfloor}_{=4}, \quad \hat{b}_2 = \underbrace{\lfloor \frac{9}{3} \rfloor}_{=3}, \quad \hat{b}_3 = \underbrace{\lfloor \frac{9}{6} \rfloor}_{=1}, \quad \hat{b}_4 = \underbrace{\lfloor \frac{9}{7} \rfloor}_{=1}, \quad \hat{b}_5 = \underbrace{\lfloor \frac{9}{5} \rfloor}_{=1}, \quad \hat{b}_6 = \underbrace{\lfloor \frac{9}{9} \rfloor}_{=1}.$$

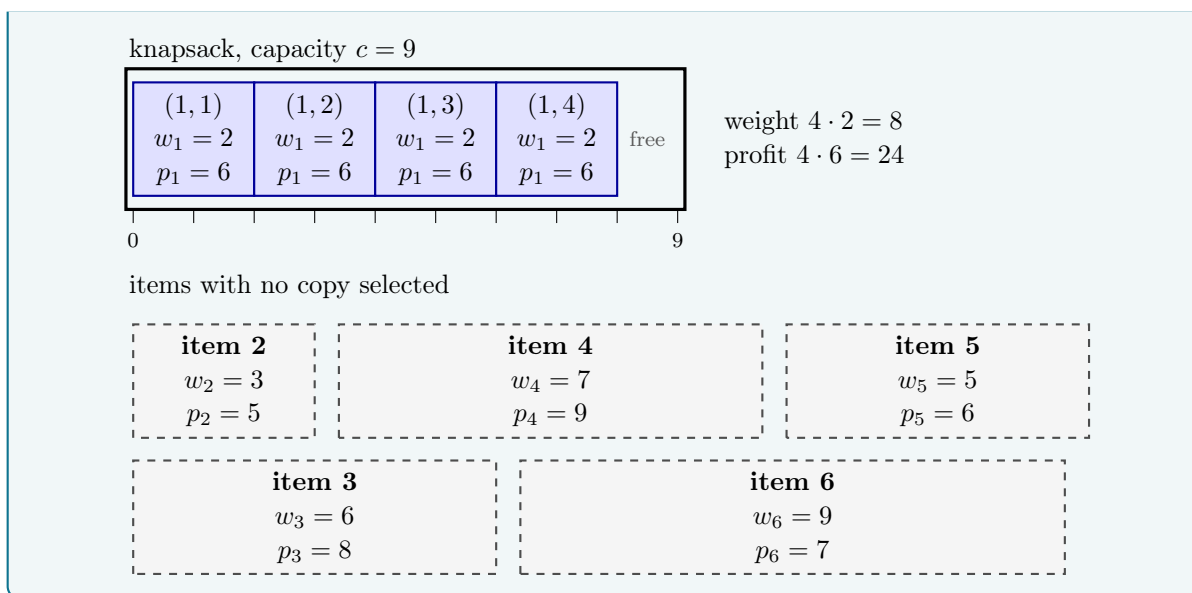
The extended set of items is:

$$\hat{\mathcal{J}} = \{ (1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (3, 1), (4, 1), (5, 1), (6, 1) \}.$$

Each copy (j, k) has profit p_j and weight w_j .

Copy label	1	2	3	4	5	6	7	8	9	10	11
(j, k)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(2, 1)	(2, 2)	(2, 3)	(3, 1)	(4, 1)	(5, 1)	(6, 1)
Profit p_j	6	6	6	6	5	5	5	8	9	6	7
Weight w_j	2	2	2	2	3	3	3	6	7	5	9

The drawing shows an optimal solution: four copies of item 1, with weight $4 \cdot 2 = 8$ and profit $4 \cdot 6 = 24$; one unit of capacity stays free.



2.5.1 Assumptions

- We make the following assumptions:
 1. The total weight of all possible copies of all items (up to the maximum number that can fit in the knapsack) exceeds the knapsack capacity (otherwise, all copies could be selected). Formally:

$$\sum_{j \in \mathcal{I}} \hat{b}_j w_j > c$$

2. Each item individually fits into the knapsack (otherwise, it cannot be selected). Formally:

$$w_j \leq c, \quad \forall j \in \mathcal{I}$$

2.6 The multidimensional knapsack problem

Definition 2.6: Multidimensional Knapsack Problem

Given:

1. a k -dimensional knapsack,
2. for each dimension $d \in \underbrace{\{1, 2, \dots, k\}}_{=\mathcal{D}}$, a *capacity* $c_d \in \mathbb{Z}_{\geq 1}$,
3. a set $\mathcal{I} = \{1, 2, \dots, n\}$ of n items,
4. for each item $j \in \mathcal{I}$, a *profit* $p_j \in \mathbb{Z}_{\geq 1}$,
5. for each dimension $d \in \mathcal{D}$ and each item $j \in \mathcal{I}$, a *weight* $w_{dj} \in \mathbb{Z}_{\geq 1}$,

the **Multidimensional Knapsack Problem** (MDKP) asks to decide which items to insert into the knapsack, so that for every dimension the total weight of the selected items

does not exceed the corresponding capacity, and the total profit of the selected items is maximized.

- An **MDKP instance** is specified by:

1. two integer numbers:

the number of items $n \in \mathbb{Z}_{\geq 1}$,

the number of dimensions $k \in \mathbb{Z}_{\geq 1}$,

2. one vector of integer numbers of length n :

$\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, containing the item profits,

3. one vector of integer numbers of length k :

$\mathbf{c} \in \mathbb{Z}_{\geq 1}^k$, containing the knapsack capacities for each dimension,

4. one matrix of integer numbers with k rows and n columns:

$\mathbf{W} \in \mathbb{Z}_{\geq 1}^{k \times n}$, containing the weights of the items for each dimension.

- An **MDKP feasible solution** is a subset of items

$$S \subseteq \mathcal{I}$$

such that the capacity constraints are satisfied in every dimension, namely

$$\sum_{j \in S} w_{dj} \leq c_d \quad \forall d \in \mathcal{D}.$$

Accordingly, the MDKP can be equivalently stated as the problem of selecting a subset of items whose total weight respects the knapsack capacities in every dimension, and whose total profit is maximized.

Given an MDKP instance, the MDKP can be formally stated as:

$$\tilde{S} \in \arg \max_{S \subseteq \mathcal{I}} \left\{ \sum_{j \in S} p_j \quad : \quad \sum_{j \in S} w_{dj} \leq c_d \quad \forall d \in \mathcal{D} \right\}. \quad (2.8)$$

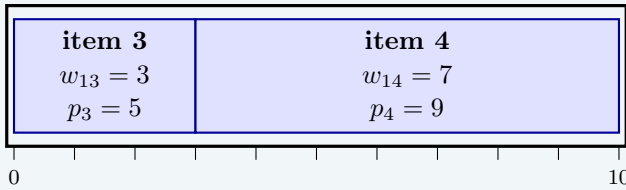
Example 2.5: MDKP instance

We consider a 2-dimensional knapsack ($k = 2$) with capacities $c_1 = 10$ and $c_2 = 12$, and a set $\mathcal{J} = \{1, 2, 3, 4\}$ of 4 items ($n = 4$), with the following profits and weights:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$
Profits	p_j	6	7	5	9
Weights (dim. 1)	w_{1j}	4	5	3	7
Weights (dim. 2)	w_{2j}	6	4	5	3

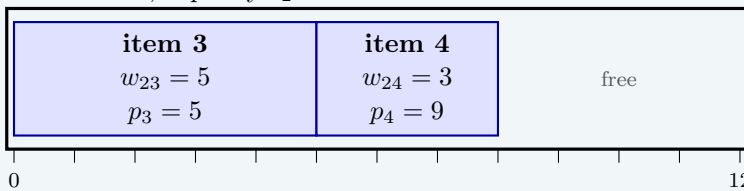
The drawing has one box per dimension, of length $c_1 = 10$ and $c_2 = 12$: a selected item occupies a block of length w_{1j} in the first box and one of length w_{2j} in the second. The optimal solution $\{3, 4\}$ fills the first dimension and uses 8 units of the second, with profit $5 + 9 = 14$.

dimension 1, capacity $c_1 = 10$



weight $3 + 7 = 10$

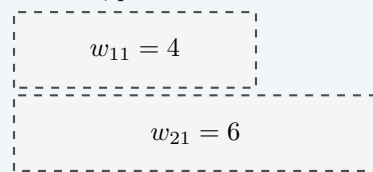
dimension 2, capacity $c_2 = 12$



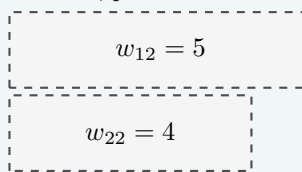
weight $5 + 3 = 8$

not selected (top: dimension 1, bottom: dimension 2)

item 1, $p_1 = 6$



item 2, $p_2 = 7$



2.6.1 Assumptions

- We make the following assumptions:
 1. The set of all items does *not* fit in the knapsack (otherwise, selecting all items is feasible and optimal). Formally,

$$\exists d \in \mathcal{D} \text{ such that } \sum_{j \in \mathcal{I}} w_{dj} > c_d.$$

2. Each item individually fits in the knapsack in every dimension (otherwise, it can never be selected). Formally,

$$w_{dj} \leq c_d, \quad \forall d \in \mathcal{D}, \forall j \in \mathcal{I}.$$

2.7 The multiple knapsack problem

Definition 2.7: Multiple Knapsack Problem

Given:

1. $m \in \mathbb{Z}_{\geq 1}$ knapsacks,
2. for each knapsack $k \in \underbrace{\{1, 2, \dots, m\}}_{=\mathcal{K}}$, a *capacity* $c_k \in \mathbb{Z}_{\geq 1}$,
3. a set $\mathcal{I} = \{1, 2, \dots, n\}$ of n items,
4. for each item $j \in \mathcal{I}$, a *profit* $p_j \in \mathbb{Z}_{\geq 1}$ and a *weight* $w_j \in \mathbb{Z}_{\geq 1}$,

the **Multiple Knapsack Problem** (MKP) asks to decide which items to insert into each knapsack, so that in every knapsack the total weight of the assigned items does not exceed its capacity, and the total profit of the selected items is maximized.

- An **MKP instance** is specified by:
 1. two integer numbers:
 - the number of items $n \in \mathbb{Z}_{\geq 1}$,
 - the number of knapsacks $m \in \mathbb{Z}_{\geq 1}$,
 2. one vector of integer numbers of length n :
 - $\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, containing the item profits,
 3. one vector of integer numbers of length n :
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, containing the item weights,
 4. one vector of integer numbers of length m :
 - $\mathbf{c} \in \mathbb{Z}_{\geq 1}^m$, containing the capacities of the knapsacks.

- An **MKP feasible solution** is a family of subsets of items

$$\mathcal{S} = \{S_k\}_{k \in \mathcal{K}}, \quad S_k \subseteq \mathcal{I} \text{ for all } k \in \mathcal{K},$$

such that:

- (i) each item is assigned to at most one knapsack, namely

$$S_k \cap S_{k'} = \emptyset \text{ for all } k, k' \in \mathcal{K} \text{ with } k \neq k';$$

- (ii) the total weight of the items assigned to each knapsack respects its capacity, namely

$$\sum_{j \in S_k} w_j \leq c_k \text{ for all } k \in \mathcal{K}.$$

Accordingly, the MKP can be equivalently stated as the problem of selecting pairwise disjoint subsets of items, one subset for each knapsack, such that every knapsack capacity is respected and the total profit is maximized.

Given an MKP instance, the MKP can be formally stated as:

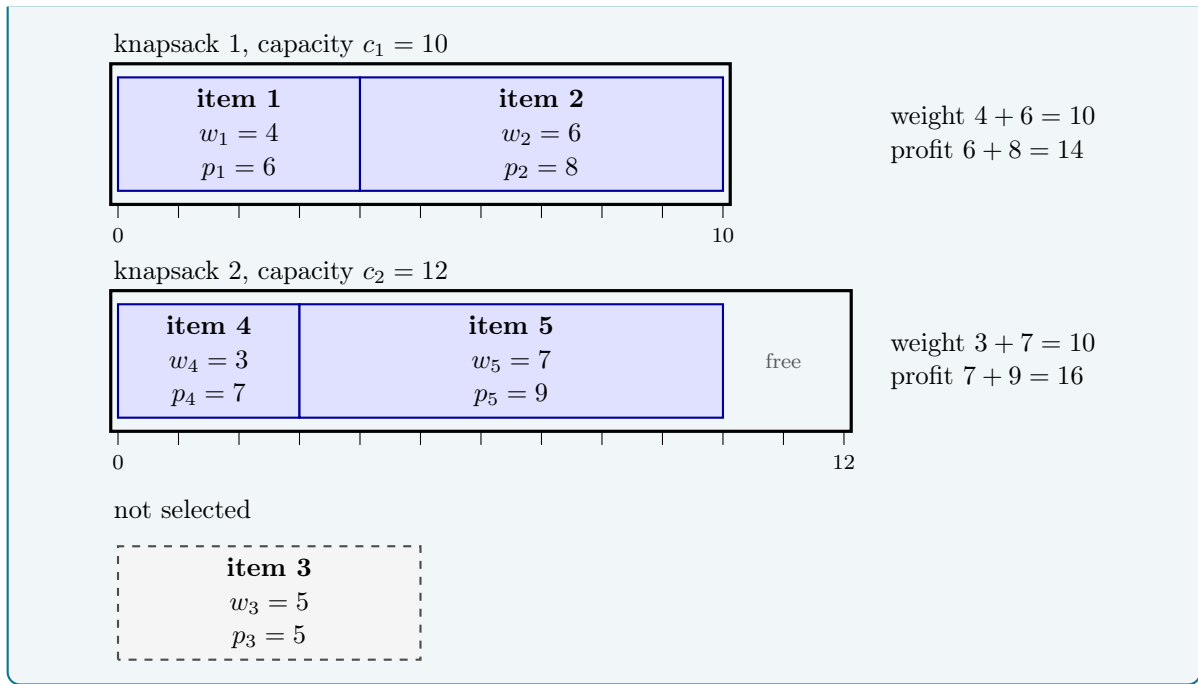
$$\tilde{\mathcal{S}} \in \arg \max_{\mathcal{S} = \{S_k\}_{k \in \mathcal{K}}} \left\{ \sum_{k \in \mathcal{K}} \sum_{j \in S_k} p_j : \begin{array}{l} S_k \subseteq \mathcal{I} \quad \forall k \in \mathcal{K}, \\ S_k \cap S_{k'} = \emptyset \quad \forall k, k' \in \mathcal{K} \text{ with } k \neq k', \\ \sum_{j \in S_k} w_j \leq c_k \quad \forall k \in \mathcal{K} \end{array} \right\}. \quad (2.9)$$

Example 2.6: MKP instance

We consider $m = 2$ knapsacks with capacities $c_1 = 10$ and $c_2 = 12$, and a set $\mathcal{I} = \{1, 2, 3, 4, 5\}$ of 5 items ($n = 5$), with the following profits and weights:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
Profits	p_j	6	8	5	7	9
Weights	w_j	4	6	5	3	7

The drawing shows an optimal solution: $S_1 = \{1, 2\}$ in knapsack 1 and $S_2 = \{4, 5\}$ in knapsack 2, with total profit $14 + 16 = 30$; item 3 stays out.



2.7.1 Assumptions

- We make the following assumptions:
 1. The set of all items does *not* fit in the knapsacks altogether (otherwise, selecting all items is feasible and optimal). Formally,

$$\sum_{j \in \mathcal{I}} w_j > \sum_{k \in \mathcal{K}} c_k.$$

2. Each item individually fits in (at least) one knapsack (otherwise, it can never be selected). Formally,

$$\forall j \in \mathcal{I} \quad \exists k \in \mathcal{K} \text{ such that } w_j \leq c_k.$$

2.8 The multiple-choice knapsack problem

Definition 2.8: Multiple-Choice Knapsack Problem

Given:

1. a knapsack of *capacity* $c \in \mathbb{Z}_{\geq 1}$,
2. a set $\mathcal{I} = \{1, 2, \dots, n\}$ of n items,
3. the set of items is partitioned into $q \in \mathbb{Z}_{\geq 2}$ classes,
4. for each class $h \in \{1, 2, \dots, q\}$, the subset $\mathcal{I}_h \subseteq \mathcal{I}$ contains the items of that class,
5. for each item $j \in \mathcal{I}$, a *profit* $p_j \in \mathbb{Z}_{\geq 1}$ and a *weight* $w_j \in \mathbb{Z}_{\geq 1}$,

the **Multiple-Choice Knapsack Problem** (MCKP) asks to decide which items to insert into the knapsack, so that exactly one item is selected from each class, the total weight of the selected items does not exceed the knapsack capacity, and the total profit of the selected items is maximized.

- An **MCKP instance** is specified by:

1. two integer numbers:
 - the number of items $n \in \mathbb{Z}_{\geq 1}$,
 - the knapsack capacity $c \in \mathbb{Z}_{\geq 1}$,
2. one vector of integer numbers of length n :
 - $\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, containing the item profits,
3. one vector of integer numbers of length n :
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, containing the item weights,
4. one integer number:
 - the number of classes $q \in \mathbb{Z}_{\geq 2}$,
5. one family of subsets:
 - $\{\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_q\}$, a partition of $\mathcal{I}$ into q classes, where $\mathcal{I}_h \subseteq \mathcal{I}$ is the set of items belonging to class h .

- An **MCKP feasible solution** is a subset of items

$$S \subseteq \mathcal{I}$$

such that:

- (i) exactly one item is selected from each class, namely

$$|S \cap \mathcal{I}_h| = 1 \quad \text{for all } h \in \{1, 2, \dots, q\};$$

(ii) the total weight of the selected items does not exceed the knapsack capacity, namely

$$\sum_{j \in S} w_j \leq c.$$

Accordingly, the MCKP can be equivalently stated as the problem of selecting exactly one item from each class such that the knapsack capacity is respected and the total profit is maximized.

Given an MCKP instance, the MCKP can be formally stated as:

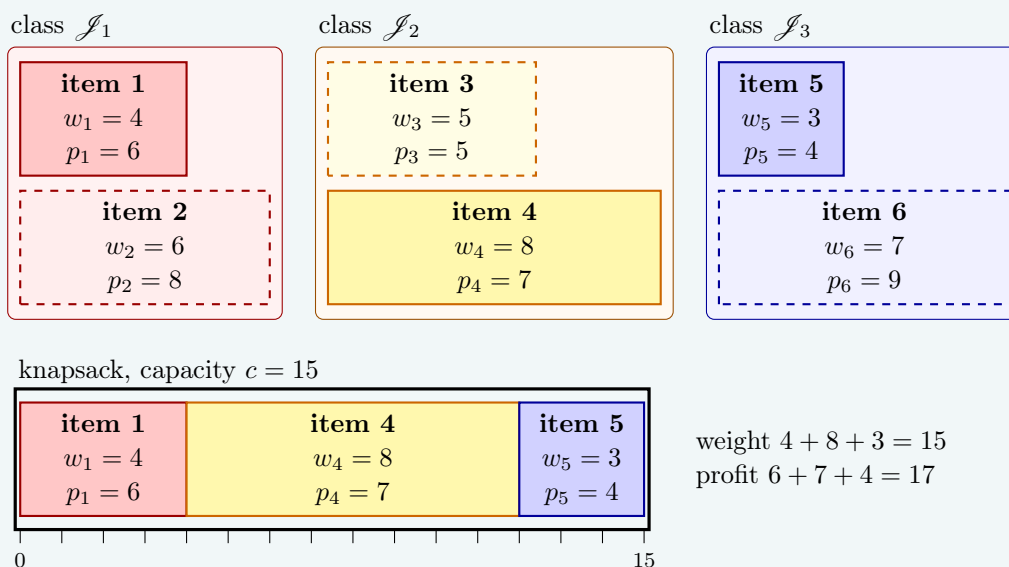
$$\tilde{S} \in \arg \max_{S \subseteq \mathcal{I}} \left\{ \sum_{j \in S} p_j : \sum_{j \in S} w_j \leq c, |S \cap \mathcal{J}_h| = 1 \ \forall h \in \{1, 2, \dots, q\} \right\}. \quad (2.10)$$

Example 2.7: MCKP instance

We consider a knapsack of capacity $c = 15$ and a set $\mathcal{I} = \{1, 2, 3, 4, 5, 6\}$ of 6 items ($n = 6$). The items are partitioned into $q = 3$ classes: $\mathcal{J}_1 = \{1, 2\}$, $\mathcal{J}_2 = \{3, 4\}$, $\mathcal{J}_3 = \{5, 6\}$. The profits and weights are given as follows:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Profits	p_j	6	8	5	7	4	9
Weights	w_j	4	6	5	8	3	7

In the drawing every class is a coloured group, and the item chosen in each class is drawn solid, the other one dashed. An optimal solution, $\{1, 4, 5\}$, fills the knapsack, with weight $4 + 8 + 3 = 15$ and profit $6 + 7 + 4 = 17$.



2.8.1 Assumptions

- We make the following assumptions:
 1. Each class contains at least one item (otherwise, selecting exactly one item from each class is impossible). Formally:

$$\mathcal{J}_h \neq \emptyset \quad \forall h \in \{1, \dots, q\}.$$

2. The instance is feasible, i.e., there exists at least one choice of exactly one item per class that fits in the knapsack. A necessary and sufficient (and easy-to-check) condition is:

$$\sum_{h=1}^q \min_{j \in \mathcal{J}_h} w_j \leq c.$$

2.9 The knapsack problem with setups

Definition 2.9: Knapsack Problem with Setups

Given:

1. a knapsack of *capacity* $c \in \mathbb{Z}_{\geq 1}$,
2. a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n items,
3. the set of items is partitioned into $q \in \mathbb{Z}_{\geq 2}$ classes,
4. for each class $h \in \{1, 2, \dots, q\}$, the subset $\mathcal{J}_h \subseteq \mathcal{J}$ contains the items of that class, with an associated *setup cost* $f_h \in \mathbb{Z}_{\geq 1}$ and a *setup weight* $s_h \in \mathbb{Z}_{\geq 1}$,
5. for each item $j \in \mathcal{J}$, a *revenue* $p_j \in \mathbb{Z}_{\geq 1}$ and a *weight* $w_j \in \mathbb{Z}_{\geq 1}$,

the **Knapsack Problem with Setups** (KPS) asks to decide which items to insert into the knapsack, so that the total weight of the selected items plus the setup weights of their classes does not exceed the knapsack capacity, and the total revenue of the selected items minus the setup costs of their classes is maximized.

- A **KPS instance** is specified by:
 1. two integer numbers:
 - the number of items $n \in \mathbb{Z}_{\geq 1}$,
 - the knapsack capacity $c \in \mathbb{Z}_{\geq 1}$,
 2. one vector of integer numbers of length n :
 - $\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, containing the item revenues,
 3. one vector of integer numbers of length n :

- $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, containing the item weights,
- 4. one integer number:
the number of classes $q \in \mathbb{Z}_{\geq 2}$,
- 5. one family of subsets:
 $\{\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_q\}$, a partition of $\mathcal{I}$ into q classes, where $\mathcal{J}_h \subseteq \mathcal{I}$ is the set of items belonging to class h ,
- 6. two vectors of integer numbers of length q :
 $\mathbf{f} \in \mathbb{Z}_{\geq 1}^q$, containing the setup costs of the classes,
 $\mathbf{s} \in \mathbb{Z}_{\geq 1}^q$, containing the setup weights of the classes.

- A **KPS feasible solution** is a subset of items

$$S \subseteq \mathcal{I}.$$

Given S , define the set of **activated classes** as

$$\mathcal{H}(S) := \{h \in \{1, 2, \dots, q\} : S \cap \mathcal{J}_h \neq \emptyset\}.$$

The subset S is feasible if the total weight of the selected items plus the setup weights of the activated classes does not exceed the knapsack capacity, namely

$$\sum_{j \in S} w_j + \sum_{h \in \mathcal{H}(S)} s_h \leq c.$$

Accordingly, the KPS can be equivalently stated as the problem of selecting a subset of items so as to maximize the total revenue of the selected items minus the setup costs of their activated classes.

Given a KPS instance, the KPS can be formally stated as:

$$\tilde{S} \in \arg \max_{S \subseteq \mathcal{I}} \left\{ \sum_{j \in S} p_j - \sum_{h \in \mathcal{H}(S)} f_h : \sum_{j \in S} w_j + \sum_{h \in \mathcal{H}(S)} s_h \leq c \right\}. \quad (2.11)$$

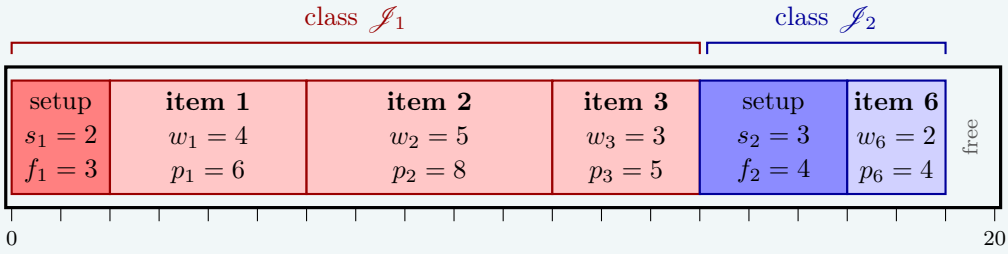
Example 2.8: KPS instance

We consider a knapsack of capacity $c = 20$ and a set $\mathcal{I} = \{1, 2, 3, 4, 5, 6\}$ of 6 items ($n = 6$). The items are partitioned into $q = 2$ classes: $\mathcal{I}_1 = \{1, 2, 3\}$ and $\mathcal{I}_2 = \{4, 5, 6\}$. Each class has a setup cost and a setup weight: $f_1 = 3$, $s_1 = 2$ for class 1, and $f_2 = 4$, $s_2 = 3$ for class 2. The item revenues and weights are given as follows:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Revenues	p_j	6	8	5	7	9	4
Weights	w_j	4	5	3	6	7	2

In the drawing the classes have two colours, and a darker block is the setup of a class, with its weight s_h and its cost f_h . An optimal solution, $\{1, 2, 3, 6\}$, activates both classes: the weight is $(2 + 4 + 5 + 3) + (3 + 2) = 19 \leq 20$ and the value is $(6 + 8 + 5 + 4) - (3 + 4) = 16$.

knapsack, capacity $c = 20$



not selected



2.9.1 Assumptions

- We make the following assumptions:

1. Each class contains at least one item. Formally:

$$\mathcal{I}_h \neq \emptyset \quad \forall h \in \{1, 2, \dots, q\}.$$

2. Each item can fit into the knapsack together with the setup weight of its class (otherwise, it can never be selected). Formally:

$$w_j + s_h \leq c \quad \forall h \in \{1, 2, \dots, q\}, \forall j \in \mathcal{I}_h.$$

2.10 Remarks

- Dantzig (1957) sorted the items by non-increasing efficiency p_j/w_j and used this order to solve the continuous version of the KP; the same order returns in the bounds of the following chapters.
- Karp (1972) put the subset sum problem, called knapsack in his paper, among the first 21 NP-complete problems. The KP contains the SSP as the special case $p_j = w_j$, so it is NP-hard.
- Ibarra and Kim (1975) gave the first fully polynomial-time approximation scheme for the KP.

2.11 Typical mistakes

Warning

- **Confusing the KP with the SSP.** The SSP asks for a total weight *exactly* equal to c and has a yes/no answer; it is the KP with $p_j = w_j$, where the answer is yes if the optimal value is c .
- **Stopping at the order of efficiency.** Taking the items in the order $1, 2, 3, \dots$ of non-increasing p_j/w_j gives $\{1, 2\}$ with profit 11, while the optimal solution of [Example 2.1](#) is $\{1, 4\}$ with profit 15.
- **Counting identical copies as different solutions.** In the BKP and in the UKP the copies (j, k) of the same item are interchangeable: different subsets of the extended set of items can give the same packing.
- **Choosing at most one item per class.** In the MCKP exactly one item per class is selected, so an instance can be infeasible; in the KPS a class with no selected item costs nothing.
- **Forgetting the assumptions.** An item with $w_j > c$ can never be selected, and if all the items fit the problem is trivial: both cases are removed before solving.

2.12 In summary

Summary

- The KP selects a subset of items of weight at most c and maximum profit; a solution is a subset, and 2^n subsets are candidates.
- The SSP, the BKP, the UKP, the MDKP, the MKP, the MCKP and the KPS change the number of copies, of resources, of knapsacks or the structure of the items; each one has its assumptions on the data.
- A solution found by eye is a primal bound; the cardinality bound is a dual bound that uses only the weights and the profits.
- The order of efficiency is not enough to find an optimal solution, even on a small instance.

Dashboard of the reference instance.

KP, Example 2.1 (max)	lower bound	upper bound	from
this chapter	15 (by eye)	17 (cardinality bound)	definition

Packing problems

Prerequisites

- From this module, [Chapter 1 «Combinatorial optimization problems»](#): instances, feasible solutions, primal and dual bounds, the gap, and the Bell numbers that count the partitions of a set.
- From this module, [Chapter 2 «Knapsack problems»](#): the form of the definitions and of the assumptions on the data.

Objectives

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

- state the definition of the bin packing problem and of the cutting stock problem, with instances and feasible solutions;
- explain the assumptions on the data;
- count the candidate solutions of a small instance and recognize that they are partitions;
- find a feasible solution by hand and compute the weight bound;
- use the two bounds to certify that a solution is optimal.

3.1 Overview of the family

- The **packing problems** share the same data: an infinite number of identical bins of *capacity* $c \in \mathbb{Z}_{\geq 1}$, a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n items and, for each item $j \in \mathcal{J}$, a *weight* $w_j \in \mathbb{Z}_{\geq 1}$.
- A solution assigns every item, or every required copy of an item, to a bin: the total weight of the items in a bin must not exceed the capacity c .
- The objective is to minimize the number of bins used.

A packing problem places all the items into the fewest bins of capacity c . The members of the family differ in the number of copies of each item that must be packed.

- The table lists the members of the family, with the section that defines each of them.

Problem	Acronym	Items to pack	Section
bin packing problem	BPP	one copy of each item	3.2
cutting stock problem	CSP	d_j copies of item j	3.3

3.2 The bin packing problem

Definition 3.1: Bin Packing Problem

Given:

1. an infinite number of bins, each with *capacity* $c \in \mathbb{Z}_{\geq 1}$,
2. a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n items,
3. for each item $j \in \mathcal{J}$, a *weight* $w_j \in \mathbb{Z}_{\geq 1}$,

the **Bin Packing Problem** (BPP) asks to decide in which bin to place each item, so that the total weight of the items in any bin does not exceed the bin capacity, and the total number of bins used is minimized.

- A **BPP instance** is specified by:
 1. two integer numbers:
 - the bin capacity $c \in \mathbb{Z}_{\geq 1}$,
 - the number of items $n \in \mathbb{Z}_{\geq 1}$,
 2. one vector of integer numbers of length n :
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, containing the item weights.
- A **BPP feasible solution** is a family of **packing patterns**, i.e., subsets of items each assigned to one bin,

$$\mathcal{P} \subseteq 2^{\mathcal{J}}$$

such that:

(i) $\mathcal{P}$ forms a partition of $\mathcal{I}$ into *nonempty* packing patterns, namely

$$\begin{aligned} P &\neq \emptyset, \quad \text{for all } P \in \mathcal{P}, \\ P \cap P' &= \emptyset, \quad \text{for all } P, P' \in \mathcal{P} \text{ with } P \neq P', \\ \bigcup_{P \in \mathcal{P}} P &= \mathcal{I}; \end{aligned}$$

(ii) the total weight of every packing pattern in $\mathcal{P}$ respects the bin capacity, namely

$$\sum_{j \in P} w_j \leq c, \quad \text{for all } P \in \mathcal{P}.$$

Accordingly, the BPP can be equivalently stated as the problem of determining a partition of the item set into packing patterns such that the total weight of the items in each packing pattern does not exceed the bin capacity, and the number of packing patterns (bins) is minimized.

Given a BPP instance, the BPP can be formally stated as:

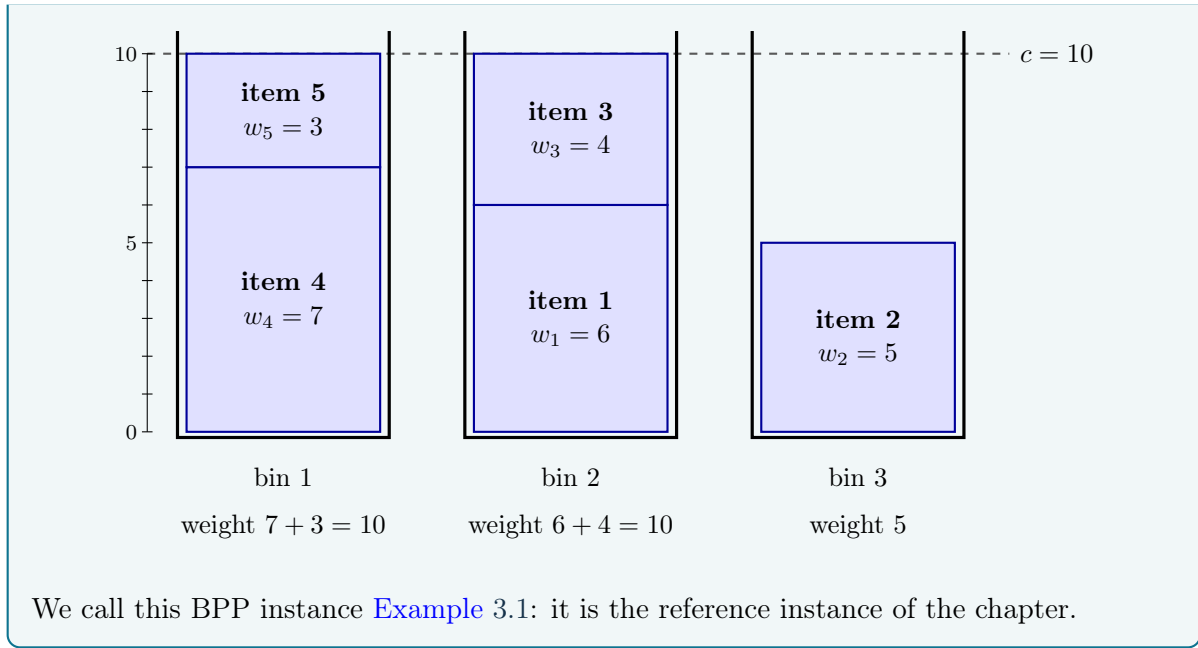
$$\tilde{\mathcal{P}} \in \arg \min_{\mathcal{P} \subseteq 2^{\mathcal{I}}} \left\{ |\mathcal{P}| : \begin{aligned} &P \neq \emptyset, \quad \forall P \in \mathcal{P}, \\ &P \cap P' = \emptyset, \quad \forall P, P' \in \mathcal{P}, P \neq P', \\ &\bigcup_{P \in \mathcal{P}} P = \mathcal{I}, \\ &\sum_{j \in P} w_j \leq c, \quad \forall P \in \mathcal{P} \end{aligned} \right\}. \quad (3.1)$$

Example 3.1: BPP instances

We consider a bin of capacity $c = 10$ and a set $\mathcal{I} = \{1, 2, 3, 4, 5\}$ of 5 items ($n = 5$), with the following item weights:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
Weights	w_j	6	5	4	7	3

In the drawing every bin is a box of height $c = 10$ and every item a block of height w_j . The three bins hold $\{4, 5\}$, $\{1, 3\}$ and $\{2\}$; we show below that 3 bins are the minimum.



3.2.1 Assumptions

- We make the following assumptions:
 1. The total weight of all items exceeds the bin capacity (otherwise, all items could fit in a single bin). Formally:

$$\sum_{j \in \mathcal{I}} w_j > c$$

2. Each item individually fits into a single bin (otherwise, it cannot be packed at all). Formally:

$$w_j \leq c, \quad \forall j \in \mathcal{I}$$

3.2.2 Counting the solutions, a first solution and a simple bound

- **Candidate solutions.** The candidate solutions are the partitions of $\mathcal{I}$: they are counted by the Bell numbers ([Chapter 1 «Combinatorial optimization problems»](#)). For [Example 3.1](#) we have $B(5) = 52$ partitions.
- **Feasible solutions.** A partition is feasible if every bin weighs at most 10. Enumeration gives 11 feasible partitions: 1 with 5 bins (one item per bin), 6 with 4 bins and 4 with 3 bins. Two bins are not enough, since the total weight 25 exceeds $2 \cdot 10$.
- **A first solution, by eye.** We pair the heavy items with the light ones: item 4 (weight 7) with item 5 (weight 3), and item 1 (weight 6) with item 3 (weight 4); item 2 (weight 5) stays alone. The solution

$$\mathcal{P} = \{ \{4, 5\}, \{1, 3\}, \{2\} \}$$

uses 3 bins, of weights 10, 10 and 5: it is a **primal bound**, $UB = 3 \geq z(I)$.

Observation 3.2: weight bound

Every feasible solution uses at least

$$\left\lceil \frac{\sum_{j \in \mathcal{J}} w_j}{c} \right\rceil$$

bins, so this value is a dual bound.

Proof. If a solution uses k bins, the total weight it carries is at most $k c$. The total weight of the items is $\sum_j w_j$, hence $k c \geq \sum_j w_j$. The number k is an integer, so $k \geq \lceil \sum_j w_j / c \rceil$. $\square$

- **The dual bound and the gap.** In [Example 3.1](#) we have $\sum_j w_j = 25$ and $c = 10$, so $\text{LB} = \lceil 25/10 \rceil = 3$. The two bounds coincide: the gap is zero, the solution found by eye is optimal and $z(I) = 3$, without enumerating the partitions.
- The optimal solution is not unique: there are 4 optimal partitions,

$$\begin{aligned} & \{\{1\}, \{2, 3\}, \{4, 5\}\}, \quad \{\{2\}, \{1, 3\}, \{4, 5\}\}, \\ & \{\{4\}, \{1, 5\}, \{2, 3\}\}, \quad \{\{4\}, \{1, 3\}, \{2, 5\}\}. \end{aligned}$$

3.3 The cutting stock problem

Definition 3.3: Cutting Stock Problem

Given:

1. an infinite number of bins, each with *capacity* $c \in \mathbb{Z}_{\geq 1}$,
2. a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n items,
3. for each item $j \in \mathcal{J}$, a *weight* $w_j \in \mathbb{Z}_{\geq 1}$ and a *number of required copies* $d_j \in \mathbb{Z}_{\geq 1}$,

the **Cutting Stock Problem** (CSP) asks to decide in which bin to place each copy of every item, so that the total weight of all item copies in any bin does not exceed the bin capacity, the required number of copies of each item is placed into the bins, and the total number of bins used is minimized.

- A **CSP instance** is specified by:
 1. two integer numbers:
 - the bin capacity $c \in \mathbb{Z}_{\geq 1}$,
 - the number of items $n \in \mathbb{Z}_{\geq 1}$,
 2. two vectors of integer numbers of length n :
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, containing the item weights,

$\mathbf{d} \in \mathbb{Z}_{\geq 1}^n$, containing the required number of copies for each item.

- The problem is called the *Cutting Stock Problem* because, in its classical application, the bins correspond to large stock rolls (or sheets of material) that must be cut in order to produce the required copies of the items, also known as item demands.
- Given a CSP instance, we define the **extended set of items** as

$$\hat{\mathcal{J}} = \{ (j, k) : j \in \mathcal{J}, k \in \{1, 2, \dots, d_j\} \}, \quad (3.2)$$

where each pair (j, k) represents the k -th required copy of item j . The cardinality of the extended set of items is

$$|\hat{\mathcal{J}}| = \sum_{j \in \mathcal{J}} d_j.$$

- A **CSP feasible solution** is a family of **packing patterns**, i.e., subsets of item copies each assigned to one bin,

$$\mathcal{P} \subseteq 2^{\hat{\mathcal{J}}}$$

such that:

- (i) $\mathcal{P}$ forms a partition of $\hat{\mathcal{J}}$ into *nonempty* packing patterns, namely

$$\begin{aligned} P &\neq \emptyset, \quad \text{for all } P \in \mathcal{P}, \\ P \cap P' &= \emptyset, \quad \text{for all } P, P' \in \mathcal{P} \text{ with } P \neq P', \\ \bigcup_{P \in \mathcal{P}} P &= \hat{\mathcal{J}}; \end{aligned}$$

- (ii) the total weight of the item copies in every packing pattern does not exceed the bin capacity, namely

$$\sum_{(j,k) \in P} w_j \leq c, \quad \text{for all } P \in \mathcal{P}.$$

Accordingly, the CSP can be equivalently stated as the problem of determining a partition of the extended item set into packing patterns such that the total weight in each packing pattern does not exceed the bin capacity, and the number of packing patterns (bins) is minimized.

Given a CSP instance, the CSP can be formally stated as:

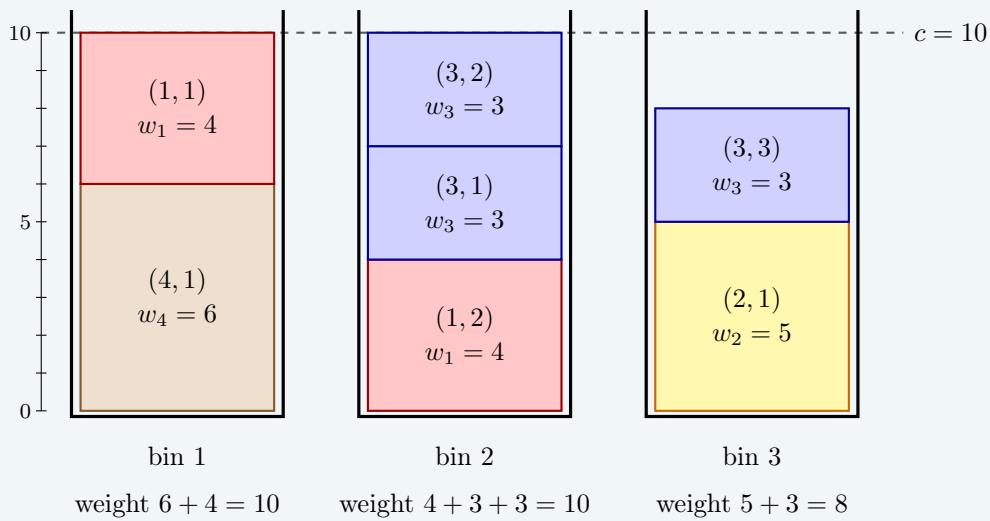
$$\tilde{\mathcal{P}} \in \arg \min_{\mathcal{P} \subseteq 2^{\hat{\mathcal{J}}}} \left\{ |\mathcal{P}| : \begin{aligned} &P \neq \emptyset, \quad \forall P \in \mathcal{P}, \\ &P \cap P' = \emptyset, \quad \forall P, P' \in \mathcal{P}, P \neq P', \\ &\bigcup_{P \in \mathcal{P}} P = \hat{\mathcal{J}}, \\ &\sum_{(j,k) \in P} w_j \leq c, \quad \forall P \in \mathcal{P} \end{aligned} \right\}. \quad (3.3)$$

Example 3.2: CSP instances

We consider bins of capacity $c = 10$ and a set $\mathcal{I} = \{1, 2, 3, 4\}$ of 4 items ($n = 4$), with the following weights and required copies:

Items		$j = 1$	$j = 2$	$j = 3$	$j = 4$
Weights	w_j	4	5	3	6
Copies	d_j	2	1	3	1

In the drawing the copies of the same item have the same colour, and the copy (j, k) is a block of height w_j . The 7 copies fit in 3 bins, of weights 10, 10 and 8.



We call this CSP instance [Example 3.2](#): it is the reference CSP instance of the chapter.

3.3.1 Assumptions

- We make the following assumptions:
 1. The total weight of all required item copies exceeds the bin capacity (otherwise, all copies could fit in a single bin). Formally:

$$\sum_{j \in \mathcal{I}} d_j w_j > c.$$

2. Each item individually fits into a single bin (otherwise, no copy of that item can be packed at all). Formally:

$$w_j \leq c, \quad \forall j \in \mathcal{I}.$$

3.3.2 Counting the solutions, a first solution and a simple bound

- **Candidate solutions.** The instance of [Example 3.2](#) has $2 + 1 + 3 + 1 = 7$ copies. The partitions of the extended set of items are counted by $B(7) = 877$, where identical copies are counted as different items; 262 of them are feasible.
- **A first solution, by eye.** We pair item 4 (weight 6) with a copy of item 1 (weight 4), put the other copy of item 1 with two copies of item 3 ($4 + 3 + 3$), and the last copy of item 3 with item 2 ($3 + 5$):

$$\mathcal{P} = \{ \{(4, 1), (1, 1)\}, \{(1, 2), (3, 1), (3, 2)\}, \{(2, 1), (3, 3)\} \}$$

uses 3 bins, of weights 10, 10 and 8, hence $UB = 3$.

- **The dual bound and the gap.** The weight bound of the bin packing problem holds with the weights of all the copies: $LB = \lceil \sum_j d_j w_j / c \rceil = \lceil 28/10 \rceil = 3$. The gap is zero, so the solution is optimal and $z(I) = 3$.

3.4 Remarks

- The worst-case behaviour of simple packing rules, such as First Fit, was analysed by Garey, Graham and Ullman (1972) and by Johnson, Demers, Ullman, Garey and Graham (1974).
- Johnson et al. (1974) proved that First Fit uses at most $\frac{17}{10}$ of the optimal number of bins, plus 2.

3.5 Typical mistakes

Warning

- **Believing that the weight bound is always tight.** With three items of weight 6 and $c = 10$ the bound gives $\lceil 18/10 \rceil = 2$, but no two items fit together, so 3 bins are needed.
- **Counting the bins by the weights alone.** The weight bound ignores how the items fit together; it is a dual bound, and the primal bound still needs a solution.
- **Listing the same packing twice.** In the cutting stock problem the copies of an item are interchangeable: partitions that differ only by the exchange of identical copies are the same packing.
- **Ignoring the order of the bins.** A solution is a partition: $\{\{4, 5\}, \{1, 3\}, \{2\}\}$ and $\{\{2\}, \{4, 5\}, \{1, 3\}\}$ are the same solution.
- **Forgetting the assumptions.** An item with $w_j > c$ fits in no bin, and if the total weight is at most c one bin is enough.

3.6 In summary

Summary

- The BPP partitions the items into the minimum number of bins of capacity c ; the cutting stock problem is the BPP with several identical copies of every item, and the BPP is the case $d_j = 1$.
- A solution is a partition, so the candidates are counted by the Bell numbers and grow very fast.
- A solution found by eye is a primal bound; the weight bound $\lceil \sum_j w_j / c \rceil$ is a dual bound.
- When the two bounds coincide, the gap is zero and the solution is certified optimal.

Dashboard of the reference instances.

instance (min)	lower bound	upper bound	from
BPP, Example 3.1	3 (weight bound)	3 (by eye)	this chapter
CSP, Example 3.2	3 (weight bound)	3 (by eye)	this chapter

Coloring problems

Prerequisites

- From this module, [Chapter 1 «Combinatorial optimization problems»](#): instances, feasible solutions, primal and dual bounds, the gap, and the Bell numbers that count the partitions of a set.
- From [Chapter 3 «Packing problems»](#): the weight bound as an example of a dual bound.
- No knowledge of graph theory is needed: the definitions used here are recalled in a short section.

Objectives

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

- use the basic notions on graphs: neighbourhood, degree, induced graph, complement graph, clique and stable set;
- state the definition of the vertex coloring problem and the assumptions on the graph;
- count the candidate solutions of a small instance and recognize that they are partitions;
- find a coloring by hand and compute the clique bound and the stable set bound;
- explain why the clique bound is not always tight.

4.1 Overview of the family

- The **coloring problems** share the same data: a simple undirected graph $G = (V, E)$, whose edges are the pairs of vertices in conflict.
- A solution assigns a color to every vertex, so that the two endpoints of every edge receive different colors: the vertices with the same color form a stable set of G .
- The objective is to minimize the number of colors used.

A coloring problem partitions the vertices of a graph into the fewest stable sets. The notions on graphs of the next subsection are the common notation of the family.

- The table lists the members of the family, with the section that defines each of them.

Problem	Acronym	Solution	Section
vertex coloring problem	VCP	a partition of V into stable sets	4.2

4.1.1 A short review of graphs

Definition 4.1: graph, neighbourhood, induced graph and complement graph

- A **graph** $G = (V, E)$ is given by a finite set V of **vertices** and a set E of **edges**, each edge being a 2-element subset of V . We write $n = |V|$ and $m = |E|$; the vertices are labelled $1, 2, \dots, n$. All the graphs of this chapter are simple and undirected.
- Two distinct vertices u and v are **adjacent** if $\{u, v\} \in E$. The **neighbourhood** of a vertex v is the set $N(v) = \{u \in V : \{u, v\} \in E\}$, and its **degree** is $\deg(v) = |N(v)|$.
- For a subset $U \subseteq V$ the **induced graph** $G[U] = (U, E[U])$ has the vertices of U and the edges of E with both endpoints in U .
- The **complement graph** $\overline{G} = (V, \overline{E})$ has the same vertices as G , and two distinct vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in G .

Definition 4.2: clique and stable set

Let $G = (V, E)$ be a graph.

1. A **clique** is a subset $K \subseteq V$ of pairwise adjacent vertices: $\{u, v\} \in E$ for all $u, v \in K$ with $u \neq v$.
2. A **stable set** (also called **independent set**) is a subset $Q \subseteq V$ of pairwise non-adjacent vertices: $\{u, v\} \notin E$ for all $u, v \in Q$ with $u \neq v$.

The **clique number** $\omega(G)$ is the largest number of vertices of a clique, and the **stability number** $\alpha(G)$ is the largest number of vertices of a stable set.

- A stable set of G is a clique of the complement graph $\overline{G}$, and vice versa. A vertex alone and the empty set are both a clique and a stable set.

A clique is a group of vertices that pairwise *conflict*; a stable set is a group of vertices with no conflict. In a coloring, the vertices of a clique need different colors, and the vertices of a stable set can share one.

4.2 The vertex coloring problem

Definition 4.3: Vertex Coloring Problem

Given:

1. a simple *graph* $G = (V, E)$, where

$$V = \{1, 2, \dots, n\}$$

is the set of n *vertices*, and

$$E \subseteq \{ \{u, v\} : u, v \in V, u \neq v \}$$

is the set of m *edges*,

the **Vertex Coloring Problem** (VCP) asks to decide how to assign a color to each vertex, so that no two adjacent vertices share the same color, and the total number of colors used is minimized.

- A **VCP instance** is specified by:

1. two integer numbers:
 - the number of vertices $n \in \mathbb{Z}_{\geq 1}$,
 - the number of edges $m \in \mathbb{Z}_{\geq 1}$,
2. two vectors of integer numbers of length m :
 - $\mathbf{u} \in \mathbb{Z}_{\geq 1}^m$, containing the first endpoint of the edges,
 - $\mathbf{v} \in \mathbb{Z}_{\geq 1}^m$, containing the second endpoint of the edges.

- A **VCP feasible solution** is a family of **color classes**, i.e., subsets of vertices each assigned to one color,

$$\mathcal{Q} \subseteq 2^V$$

such that:

- (i) $\mathcal{Q}$ forms a partition of V into *nonempty* color classes, namely

$$\begin{aligned} Q &\neq \emptyset, \quad \text{for all } Q \in \mathcal{Q}, \\ Q \cap Q' &= \emptyset, \quad \text{for all } Q, Q' \in \mathcal{Q} \text{ with } Q \neq Q', \\ \bigcup_{Q \in \mathcal{Q}} Q &= V; \end{aligned}$$

- (ii) each color class in $\mathcal{Q}$ is a *stable set*, namely

$$\{u, v\} \notin E, \quad \text{for all } Q \in \mathcal{Q}, \text{ for all } u, v \in Q \text{ with } u \neq v.$$

Accordingly, the VCP can be equivalently stated as the problem of determining a partition of the vertex set into color classes such that each color class is a stable set (i.e., its vertices can receive the same color), and the number of color classes (colors) is minimized.

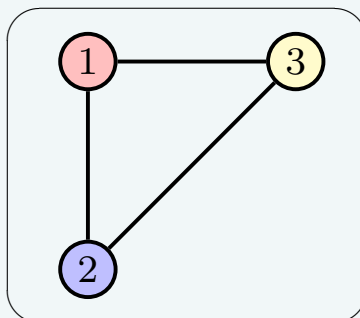
Given a VCP instance, the VCP can be formally stated as:

$$\tilde{\mathcal{Q}} \in \arg \min_{\mathcal{Q} \subseteq 2^V} \left\{ |\mathcal{Q}| : \begin{array}{l} Q \neq \emptyset, \quad \forall Q \in \mathcal{Q}, \\ Q \cap Q' = \emptyset, \quad \forall Q, Q' \in \mathcal{Q} \text{ with } Q \neq Q', \\ \bigcup_{Q \in \mathcal{Q}} Q = V, \\ \{u, v\} \notin E, \quad \forall Q \in \mathcal{Q}, \forall u, v \in Q \text{ with } u \neq v \end{array} \right\}. \quad (4.1)$$

- The optimal value of the VCP is the **chromatic number** $\chi(G)$ of the graph: the minimum number of colors (stable sets) needed to color all the vertices. The color classes of a solution are stable sets, and the number of classes is the number of colors used.

Example 4.1: VCP instance: a triangle

We consider the following graph $G = (V, E)$ with 3 vertices ($n = 3$) and 3 edges ($m = 3$):



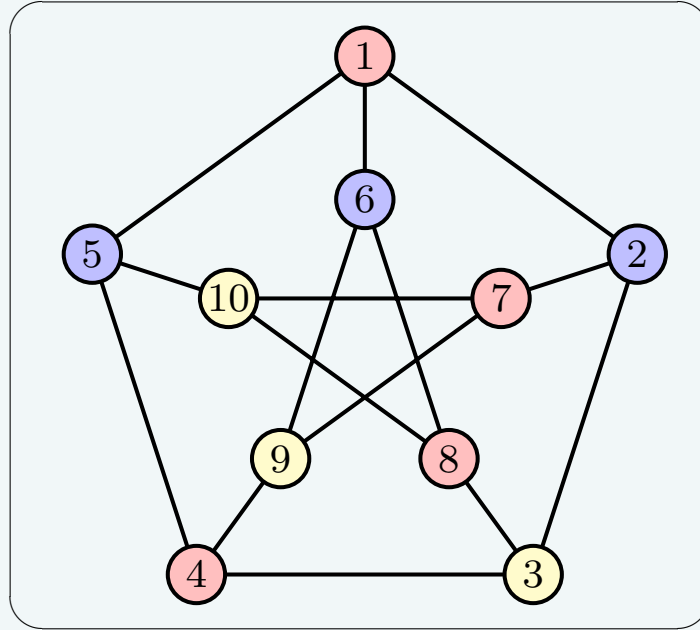
The vertex set V and the edge set E are:

$$V = \{ 1, 2, 3 \}, \quad E = \{ \{1, 2\}, \{1, 3\}, \{2, 3\} \}.$$

We call this VCP instance [Example 4.1](#).

Example 4.2: VCP instance: the Petersen graph

We consider now the following graph $G = (V, E)$ with 10 vertices ($n = 10$) and 15 edges ($m = 15$):



The vertex set V and the edge set E are:

$$V = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \},$$

$$E = \{ \{1, 2\}, \{1, 5\}, \{1, 6\}, \{2, 3\}, \{2, 7\}, \\ \{3, 4\}, \{3, 8\}, \{4, 5\}, \{4, 9\}, \{5, 10\}, \\ \{6, 8\}, \{6, 9\}, \{7, 9\}, \{7, 10\}, \{8, 10\} \}.$$

We call this VCP instance [Example 4.2](#): it is the reference instance of the chapter.

4.2.1 Assumptions

- We make the following assumptions:

1. The graph has no isolated vertices. An isolated vertex can receive any color used by the rest of the graph; if the graph has no edges, the optimal value is one. Formally:

$$\deg(v) \geq 1, \quad \forall v \in V.$$

2. No vertex has its neighbourhood contained in the neighbourhood of another vertex. Formally:

$$N(u) \not\subseteq N(v), \quad \forall u, v \in V, u \neq v.$$

- The second assumption can be made without loss of generality, because:

1. if $N(u) \subseteq N(v)$, the vertex u can be removed without changing the chromatic number: every coloring of $G \setminus \{u\}$ extends to u with the color of v ;

2. the vertices u and v are not adjacent: otherwise $v \in N(u) \subseteq N(v)$, which is impossible in a simple graph, so u and v can share a color.

4.2.2 Two simple bounds

Observation 4.4: clique bound

For every graph $G = (V, E)$ we have

$$\omega(G) \leq \chi(G).$$

Proof. Since all vertices in any clique are pairwise adjacent, they must all receive different colors in any VCP feasible solution. Thus, at least $\omega(G)$ different colors are needed to color the vertices of any maximum clique, implying $\omega(G) \leq \chi(G)$. $\square$

Observation 4.5: stable set bound

For every graph $G = (V, E)$ we have

$$\left\lceil \frac{n}{\alpha(G)} \right\rceil \leq \chi(G).$$

Proof. In a feasible solution with k color classes, every class is a stable set, so it has at most $\alpha(G)$ vertices. The n vertices are covered by the classes, hence $n \leq k \alpha(G)$. The number k is an integer, so $k \geq \lceil n/\alpha(G) \rceil$. $\square$

- The clique bound is useful when the graph has a large clique, and the stable set bound when its stable sets are small. Both are dual bounds: a clique and a stable set are found without knowing $\chi(G)$.

4.2.3 Counting the solutions, a first solution and the bounds

- **Candidate solutions.** The candidate solutions are the partitions of V , counted by the Bell numbers (Chapter 1 «Combinatorial optimization problems»). For [Example 4.1](#) we have $B(3) = 5$ partitions and only one is feasible: three singletons, since every pair of vertices is adjacent. Hence $\chi(G) = 3$.
- For [Example 4.2](#) we have $B(10) = 115\,975$ partitions, and 7 430 of them are feasible, that is, have only stable sets as classes.
- **A first solution, by eye.** The coloring shown in [Example 4.2](#) uses 3 colors:

$$\mathcal{Q} = \{ \{1, 4, 7, 8\}, \{2, 5, 6\}, \{3, 9, 10\} \}.$$

No edge joins two vertices of the same class, so it is feasible: it is a **primal bound**, $UB = 3 \geq \chi(G)$.

- **Dual bounds on the Petersen graph.** The graph has no triangle, so $\omega(G) = 2$ and the clique bound gives only $LB = 2$: a gap of 1 remains.

- The vertices 1, 2, 3, 4, 5 form a cycle with 5 vertices, and so do 6, 7, 8, 9, 10 (the edges of the figure). A stable set contains at most 2 vertices of each cycle, so $\alpha(G) \leq 4$; the set $\{1, 4, 7, 8\}$ shows that $\alpha(G) = 4$.
- The stable set bound gives $\lceil 10/4 \rceil = 3$. The two bounds coincide: the gap is zero and $\chi(G) = 3$.

4.3 Remarks

- Brooks (1941) proved that a connected graph that is neither complete nor an odd cycle can be colored with at most Δ colors, where Δ is the largest degree.
- Karp (1972) included the chromatic number among his 21 NP-complete problems. Garey, Johnson and Stockmeyer (1976) proved that deciding whether 3 colors are enough remains NP-complete on planar graphs of maximum degree 4.

4.4 Typical mistakes

Warning

- **Confusing a clique with a stable set.** In a clique all the pairs are edges and the vertices need different colors; in a stable set no pair is an edge and the vertices can share a color.
- **Believing that the clique bound is tight.** It is not: the Petersen graph has $\omega(G) = 2$ and $\chi(G) = 3$, and the same happens for a cycle with 5 vertices.
- **Counting the permutations of the colors as different solutions.** A solution is a partition of the vertices into stable sets: exchanging the names of two colors gives the same solution.
- **Using a class that is not a stable set.** A color class with two adjacent vertices makes the solution infeasible, whatever the number of colors.
- **Forgetting the assumptions.** An isolated vertex and a vertex with $N(u) \subseteq N(v)$ can be removed before solving, without changing the chromatic number.

4.5 In summary

Summary

- The VCP partitions the vertices of a graph into the minimum number of stable sets; the optimal value is the chromatic number $\chi(G)$.
- A clique needs different colors and a stable set can share one; the cliques of G are the stable sets of the complement graph.
- A coloring found by eye is a primal bound; the clique bound $\omega(G)$ and the stable set bound $\lceil n/\alpha(G) \rceil$ are dual bounds.
- The bounds do not always meet: on the Petersen graph the clique bound leaves a gap, the stable set bound closes it.

Dashboard of the reference instances.

instance (min)	lower bound	upper bound	from
Example 4.1 (triangle)	3 (clique bound)	3 (by eye)	this chapter
Example 4.2 (Petersen)	3 (stable set bound)	3 (by eye)	this chapter

Clique problems

Prerequisites

- From this module, the chapter «[Combinatorial optimization problems](#)»: instances, feasible solutions, primal and dual bounds, and the gap.
- From this module, the chapter «[Coloring problems](#)»: graphs, neighbourhoods, degrees, the complement graph and the chromatic number $\chi(G)$.

Objectives

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

- state the maximum clique problem and the maximum stable set problem, and pass from one to the other through the complement graph;
- describe an instance and a feasible solution, and count the candidate solutions of a small graph;
- find a clique by hand and compute four simple upper bounds on the clique number;
- state the weighted variants of the problem and its complexity.

5.1 Overview of the family

- The **clique problems** share the same data: a simple undirected graph $G = (V, E)$, with $V = \{1, 2, \dots, n\}$ and m edges, possibly with weights on the vertices or on the edges.
- A solution is a clique of G , that is, a subset of pairwise adjacent vertices.
- The objective is to maximize the size of the clique, or its total weight.

Definition 5.1: clique and clique number

Let $G = (V, E)$ be a simple undirected graph with n vertices and m edges.

1. A **clique** is a subset $K \subseteq V$ of pairwise adjacent vertices: $\{u, v\} \in E$ for all $u, v \in K$ with $u \neq v$. Equivalently, K induces a complete graph.

2. The **clique number** $\omega(G)$ is the largest number of vertices of a clique of G .

A clique problem selects a clique of maximum size or of maximum weight. On the complement graph $\overline{G}$ the same problems select stable sets.

- The table lists the members of the family, with the section that defines each of them.

Problem	Acronym	Objective	Section
maximum clique problem	MCP	maximum number of vertices of a clique	5.2
maximum stable set problem	MSSP	maximum number of vertices of a stable set	5.2
vertex-weighted maximum clique problem	VWMCP	maximum total vertex weight	5.5
edge-weighted maximum clique problem	EWMCP	maximum total edge weight	5.5

5.2 The maximum clique problem

Definition 5.2: Maximum Clique Problem

Given:

1. a simple graph $G = (V, E)$, where

$$V = \{1, 2, \dots, n\}$$

is the set of n vertices, and

$$E \subseteq \{ \{u, v\} : u, v \in V, u \neq v \}$$

is the set of m edges,

the **Maximum Clique Problem** (MCP) asks to find a clique $K \subseteq V$ of G whose number of vertices is maximum. Its optimal value is the clique number $\omega(G)$.

Definition 5.3: stable set and Maximum Stable Set Problem

Let $G = (V, E)$ be a graph.

1. A **stable set** (also called **independent set**) is a subset $Q \subseteq V$ of pairwise non-adjacent vertices: $\{u, v\} \notin E$ for all $u, v \in Q$ with $u \neq v$.
2. The **Maximum Stable Set Problem** (MSSP) asks to find a stable set of G whose number of vertices is maximum. Its optimal value is the **stability number** $\alpha(G)$.

Definition 5.4: complement graph

The **complement graph** of $G = (V, E)$ is the graph $\overline{G} = (V, \overline{E})$ with the same vertices

and the edge set

$$\bar{E} = \{ \{u, v\} : u, v \in V, u \neq v, \{u, v\} \notin E \}.$$

The pairs in $\bar{E}$ are the **non-edges** of G , and $|\bar{E}| = \binom{n}{2} - m$.

Proposition 5.5: clique number and stability number

A subset $K \subseteq V$ is a clique of G if and only if it is a stable set of $\bar{G}$. Hence

$$\omega(G) = \alpha(\bar{G}) \quad \text{and} \quad \alpha(G) = \omega(\bar{G}).$$

Proof. Two distinct vertices u, v are adjacent in G if and only if they are not adjacent in $\bar{G}$. So all the pairs of K are edges of G exactly when no pair of K is an edge of $\bar{G}$. The two problems have the same feasible solutions, hence the same optimal value. The second equality follows since $\bar{\bar{G}} = G$. $\square$

The MCP on G and the MSSP on $\bar{G}$ are the same problem: a maximum clique of G is a maximum stable set of $\bar{G}$. Whatever we learn on one of the two problems holds for the other one.

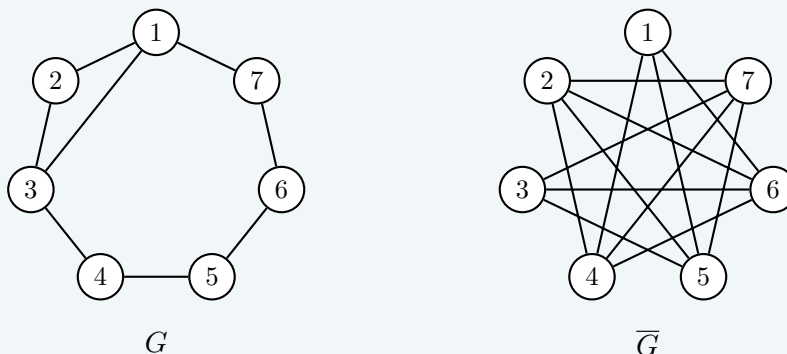
- An **MCP instance** is specified by:
 1. two integer numbers: the number of vertices $n \in \mathbb{Z}_{\geq 1}$ and the number of edges $m \in \mathbb{Z}_{\geq 1}$;
 2. two vectors of integer numbers of length m : $\mathbf{u} \in \mathbb{Z}_{\geq 1}^m$ and $\mathbf{v} \in \mathbb{Z}_{\geq 1}^m$, with the two endpoints of the edges.
- An **MCP feasible solution** is a clique $K \subseteq V$. The empty set and a single vertex are cliques, so every instance has feasible solutions.

Given an MCP instance, the MCP can be formally stated as:

$$\tilde{K} \in \arg \max_{K \subseteq V} \left\{ |K| : \{u, v\} \in E, \quad \forall u, v \in K \text{ with } u \neq v \right\}, \quad |\tilde{K}| = \omega(G). \quad (5.1)$$

Example 5.1: MCP instance: a cycle with a chord

We consider the graph $G = (V, E)$ with 7 vertices ($n = 7$) and 8 edges ($m = 8$): the cycle $1, 2, \dots, 7, 1$ and the chord $\{1, 3\}$. On the right, its complement graph $\overline{G}$, with the $\binom{7}{2} - 8 = 13$ non-edges of G .



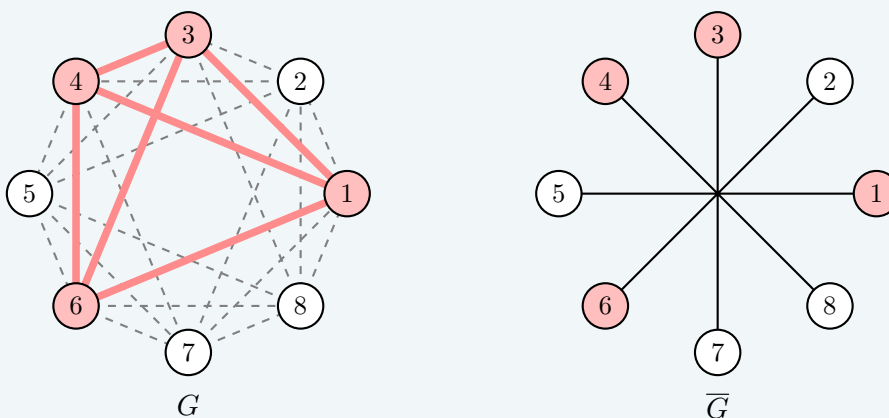
The vertex set and the edge set are:

$$V = \{1, \dots, 7\}, \quad E = \{\{1, 2\}, \{1, 3\}, \{1, 7\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}, \{6, 7\}\}.$$

We call this MCP instance [Example 5.1](#): it is the reference instance of the chapter.

Example 5.2: MCP instance: a dense graph with 8 vertices

We consider the graph $G = (V, E)$ with 8 vertices placed on a circle, where every vertex is adjacent to all the others except the opposite one. It has $m = \binom{8}{2} - 4 = 24$ edges, and its complement has only the 4 edges $\{1, 5\}, \{2, 6\}, \{3, 7\}, \{4, 8\}$. A maximum clique $\tilde{K} = \{1, 3, 4, 6\}$ is in red.



In $\overline{G}$ the red vertices form a stable set: they take one vertex from each of the 4 edges. We call this MCP instance [Example 5.2](#).

- The Petersen graph of [Chapter 4 «Coloring problems»](#) has no triangle, so its largest cliques are its edges and $\omega(G) = 2$.

5.2.1 Assumptions

- We consider nontrivial instances: the graph has at least one edge and at least one non-edge. Formally:

$$1 \leq m < \binom{n}{2}.$$

- The assumption can be made without loss of generality: if the graph has no edge, every single vertex is a maximum clique and $\omega(G) = 1$; if the graph is complete, V itself is a clique and $\omega(G) = n$.

5.3 Counting the solutions, a first solution and the bounds

- **Candidate solutions.** The candidate solutions are the subsets of V : there are 2^n of them. For [Example 5.1](#) we have $2^7 = 128$ subsets, and only 17 of them are cliques:

number of vertices	0	1	2	3	total
cliques	1	7	8	1	17

The only clique with 3 vertices is $\{1, 2, 3\}$, so the optimal solution is unique and $\omega(G) = 3$.

- For [Example 5.2](#) we have $2^8 = 256$ subsets and 81 cliques. A clique takes at most one vertex from each pair $\{1, 5\}, \{2, 6\}, \{3, 7\}, \{4, 8\}$: the maximum cliques take exactly one, so there are $2^4 = 16$ of them and $\omega(G) = 4$.
- **A first solution, by hand.** We start from a vertex of maximum degree and add, one at a time, a vertex adjacent to all the vertices already chosen. In [Example 5.1](#), vertex 1 has degree 3; we add its neighbour 2, and then 3, which is adjacent to both.
- No other vertex is adjacent to 1, 2 and 3, so we stop with $K = \{1, 2, 3\}$. It is a **primal bound**: $\text{LB} = 3 \leq \omega(G)$.

5.3.1 Simple upper bounds

- The MCP is a maximization problem: a dual bound is an **upper bound** on $\omega(G)$. We denote by $\deg(v)$ the degree of a vertex and by Δ the maximum degree of the graph.

Observation 5.6: degree bound

For every graph $G = (V, E)$ we have $\omega(G) \leq \Delta + 1$.

Proof. In a clique K , every vertex is adjacent to the other $|K| - 1$ vertices of K , so its degree is at least $|K| - 1$. Hence $|K| - 1 \leq \Delta$ for every clique. $\square$

Observation 5.7: edge bound

For every graph $G = (V, E)$ we have $\omega(G) (\omega(G) - 1) / 2 \leq m$.

Proof. A clique with k vertices contains all the $\binom{k}{2} = k(k-1)/2$ pairs of its vertices as edges, and these are edges of G . $\square$

Observation 5.8: degree count bound

Let $k^{\max}$ be the largest integer k such that at least k vertices of G have degree at least $k-1$. Then $\omega(G) \leq k^{\max}$.

Proof. A clique with k vertices has k vertices, each of degree at least $k-1$ (proof of the degree bound). So $k = \omega(G)$ satisfies the condition, and $\omega(G) \leq k^{\max}$. $\square$

Observation 5.9: coloring bound

For every graph $G = (V, E)$ we have $\omega(G) \leq \chi(G)$. In particular, the number of colors of any feasible coloring of G is an upper bound on $\omega(G)$.

Proof. The vertices of a clique are pairwise adjacent, so they take pairwise different colors: a coloring with k colors has $k \geq |K|$ for every clique K . This is the clique bound of Chapter 4 «Coloring problems», read from the side of the clique. $\square$

- **Bounds on Example 5.1.** Vertices 1 and 3 have degree 3, the other five have degree 2. The degree bound gives $\Delta + 1 = 4$, and the edge bound gives 4 too, since $4 \cdot 3/2 = 6 \leq 8 < 10 = 5 \cdot 4/2$.
- Only 2 vertices have degree at least 3, so $k = 4$ fails the condition of the degree count bound, while $k = 3$ satisfies it: $k^{\max} = 3$. The coloring $\{1, 4, 6\}, \{2, 5, 7\}, \{3\}$ gives 3 as well. The gap with LB = 3 is zero.
- **Bounds on Example 5.2.** Every vertex has degree 6: the degree bound and the degree count bound give 7, and the edge bound gives 7, since $7 \cdot 6/2 = 21 \leq 24 < 28$.
- The four pairs $\{1, 5\}, \{2, 6\}, \{3, 7\}, \{4, 8\}$ are stable sets, so they color the graph with 4 colors: the coloring bound gives $4 = \omega(G)$. On dense graphs the bounds based on degrees are weak, and a good coloring is much better.

5.4 Complexity

- The **decision version** of the MCP asks, for a graph G and an integer k , whether G has a clique with k vertices. Karp (1972) included it among his 21 NP-complete problems, so the MCP is NP-hard. By Proposition 5.5, the same holds for the MSSP.
- Listing all the subsets of V and checking the pairs of each one takes $O(2^n n^2)$ steps. For a *fixed* k , the $\binom{n}{k}$ subsets with k vertices are checked in $O(n^k k^2)$ steps: a polynomial for every fixed k , but the exponent grows with the size of the clique.
- The MCP is also hard to approximate: Håstad (1999) proved that, for every $\varepsilon > 0$, no polynomial algorithm guarantees a clique with at least $\omega(G)/n^{1-\varepsilon}$ vertices, unless every problem in NP can be solved in randomized polynomial time.

5.5 Weighted variants

- Let $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$ be a **vertex-weight vector**, with a weight w_v for each vertex $v \in V$, and let $\mathbf{r} \in \mathbb{Z}_{\geq 1}^m$ be an **edge-weight vector**, with a weight r_{uv} for each edge $\{u, v\} \in E$. For a clique K we define

$$w(K) = \sum_{v \in K} w_v \quad \text{and} \quad r(K) = \sum_{\{u,v\} \in E[K]} r_{uv},$$

where $E[K]$ is the set of the edges with both endpoints in K .

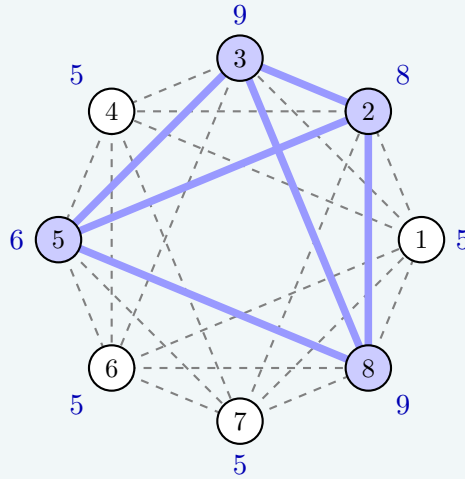
Definition 5.10: weighted maximum clique problems

1. Given a graph G and a vertex-weight vector $\mathbf{w}$, the **vertex-weighted maximum clique problem** (VWMCP) asks to find a clique K of G with maximum total vertex weight $w(K)$.
2. Given a graph G and an edge-weight vector $\mathbf{r}$, the **edge-weighted maximum clique problem** (EWMCP) asks to find a clique K of G with maximum total edge weight $r(K)$.

- With all the weights $w_v = 1$ the first problem is the MCP. The edge-weighted problem rewards the cliques with many heavy edges, and with $r_{uv} = 1$ it is again the MCP, since $r(K) = \binom{|K|}{2}$ grows with $|K|$.

Example 5.3: vertex-weighted maximum clique problem

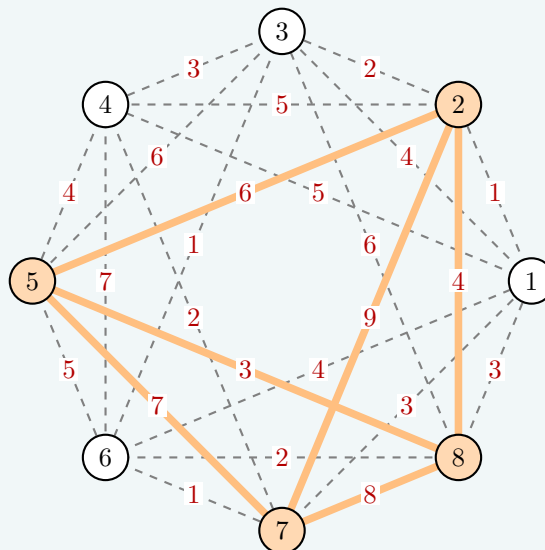
We add to the graph of [Example 5.2](#) the vertex weights in blue. The vertex-weighted maximum clique is $\{2, 3, 5, 8\}$, with weight $8 + 9 + 6 + 9 = 32$: it is unique, and it is not the maximum clique in red of [Example 5.2](#), whose weight is $5 + 9 + 5 + 5 = 24$.



A clique takes at most one vertex from each pair $\{1, 5\}$, $\{2, 6\}$, $\{3, 7\}$, $\{4, 8\}$; taking the heavier one of each pair gives $6 + 8 + 9 + 9 = 32$, and these four vertices form a clique.

Example 5.4: edge-weighted maximum clique problem

We add to the graph of [Example 5.2](#) the edge weights in red. The edge-weighted maximum clique is $\{2, 5, 7, 8\}$, in orange: its six edges weigh $6 + 4 + 9 + 7 + 3 + 8 = 37$, and the enumeration of the 16 maximum cliques shows that it is the unique optimal solution.



5.6 Remarks

- In a coloring, every color class is a stable set: the MSSP is the problem of the largest color class, and the clique number is the dual bound of Chapter 4 «Coloring problems».
- Applications of cliques and stable sets:
 - groups of pairwise connected members in social networks;
 - groups of pairwise compatible tasks or exams, in scheduling and in the planning of timetables;
 - common substructures of two molecules, in chemistry and in biology;
 - codes that correct errors, as large sets of words pairwise far apart.

5.7 Typical mistakes

Warning

- **Confusing a maximal clique with a maximum clique.** A clique is maximal when no vertex can be added to it; it is maximum when no clique is larger. In [Example 5.1](#), $\{4, 5\}$ is maximal but not maximum.
- **Building the complement graph with the edges of G .** The edges of $\overline{G}$ are exactly the pairs that are *not* edges of G : there are $\binom{n}{2} - m$ of them.
- **Using a lower bound as an upper bound.** A clique found by hand gives $\text{LB} \leq \omega(G)$; the degree bounds and a coloring give $\text{UB} \geq \omega(G)$. In this chapter the problem is a maximization.
- **Believing that the coloring bound is tight.** On the cycle with 5 vertices $\omega(G) = 2$ and $\chi(G) = 3$.

5.8 In summary

Summary

- The MCP looks for the largest set of pairwise adjacent vertices; its optimal value is the clique number $\omega(G)$.
- A clique of G is a stable set of $\overline{G}$: the MCP on G is the MSSP on $\overline{G}$, and $\omega(G) = \alpha(\overline{G})$.
- A clique found by hand is a primal bound; the degree bound, the edge bound, the degree count bound and the number of colors of a coloring are dual bounds.
- The MCP is NP-hard and hard to approximate; its weighted variants weigh the vertices or the edges.

Dashboard of the reference instances.

instance (max)	lower bound	upper bound	from
Example 5.1 (cycle with a chord)	3 (by hand)	3 (degree count, coloring)	this chapter
Example 5.2 (dense graph)	4 (by hand)	4 (coloring)	this chapter

Routing problems

Prerequisites

- From this module, [Chapter 1 «Combinatorial optimization problems»](#): what a combinatorial optimization problem is, how the number of solutions explodes, and the notions of primal bound, dual bound and gap.
- From the same chapter, the reference TSP instance with 5 cities, which this chapter studies in detail.

Objectives

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

- state the TSP, the asymmetric TSP and the metric TSP, and describe an instance and a tour;
- compute the length of a tour, and count the tours of an instance;
- explain why listing all the tours is hopeless already for 20 cities;
- build a first tour by hand with the nearest neighbour rule;
- compute a lower bound on the optimal length from the two shortest edges of every city, and measure the gap with a tour.

6.1 Overview of the family

- The **routing problems** of this chapter share the same data: a set $\mathcal{V} = \{1, 2, \dots, n\}$ of n cities and, for each pair of distinct cities $i, j \in \mathcal{V}$, a *distance* $d_{ij} \in \mathbb{Z}_{\geq 1}$.
- A solution is a **tour**, a permutation σ of the cities read as a closed route that visits every city exactly once; its **length** $L(\sigma)$ is the sum of the distances travelled.
- The objective is to minimize the length of the tour.

A routing problem finds a shortest tour through all the cities. The members of the family differ in the properties of the distances: symmetric, metric or asymmetric.

- The table lists the members of the family, with the section that defines each of them.

Problem	Acronym	Distances	Section
travelling salesman problem	TSP	symmetric, $d_{ij} = d_{ji}$	6.2
metric travelling salesman problem	metric TSP	symmetric, with the triangle inequality	6.6
asymmetric travelling salesman problem	ATSP	d_{ij} may differ from d_{ji}	6.6

6.2 The travelling salesman problem

Definition 6.1: Travelling salesman problem

Given:

1. a set $\mathcal{V} = \{1, 2, \dots, n\}$ of n cities,
2. for each pair of distinct cities $i, j \in \mathcal{V}$, a *distance* $d_{ij} \in \mathbb{Z}_{\geq 1}$, with $d_{ij} = d_{ji}$,

the **travelling salesman problem** (TSP) asks to find a closed route that starts from a city, visits every other city exactly once and returns to the starting city, so that the total distance travelled is minimized.

- A **TSP instance** is specified by:
 1. one integer number:
the number of cities $n \in \mathbb{Z}_{\geq 3}$,
 2. one symmetric matrix of integer numbers of size $n \times n$:
 $D = (d_{ij})$, containing the distances; the diagonal entries are not used.

- A **TSP feasible solution**, called a **tour**, is a permutation of the cities

$$\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n),$$

read as the closed route $\sigma_1 \rightarrow \sigma_2 \rightarrow \dots \rightarrow \sigma_n \rightarrow \sigma_1$. Its **length** is

$$L(\sigma) = \sum_{k=1}^{n-1} d_{\sigma_k \sigma_{k+1}} + d_{\sigma_n \sigma_1},$$

where the last term is the distance travelled to return to the starting city.

- Different permutations may describe the same tour: the starting city can be any of the n cities, and, since the distances are symmetric, the tour can be travelled in either direction.

Given a TSP instance, the TSP can be formally stated as:

$$\tilde{\sigma} \in \arg \min_{\sigma} \left\{ L(\sigma) = \sum_{k=1}^{n-1} d_{\sigma_k \sigma_{k+1}} + d_{\sigma_n \sigma_1} : \sigma \text{ is a permutation of } \mathcal{V} \right\}. \quad (6.1)$$

- We denote by $z(I)$ the length of an optimal tour $\tilde{\sigma}$ of the instance I .

6.2.1 Assumptions

- We make the following assumptions:
 1. There are at least three cities (with one or two cities there is only one route). Formally:

$$n \geq 3$$
 2. The distances are symmetric (travelling from i to j costs as much as travelling from j to i). Formally:

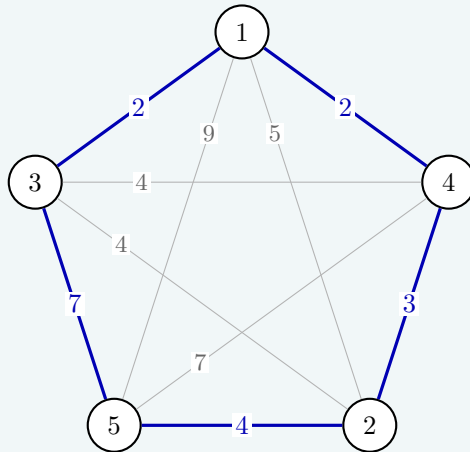
$$d_{ij} = d_{ji}, \quad \forall i, j \in \mathcal{V}, \quad i \neq j$$

Example 6.1: TSP instance

We consider a set $\mathcal{V} = \{1, 2, 3, 4, 5\}$ of 5 cities ($n = 5$), with the following distances:

d_{ij}	1	2	3	4	5
1	—	5	2	2	9
2	5	—	4	3	4
3	2	4	—	4	7
4	2	3	4	—	7
5	9	4	7	7	—

The instance can be drawn as a complete graph, with one node per city and one edge, labelled with the distance, per pair of cities. The bold edges form the tour $\sigma = (1, 3, 5, 2, 4)$, and the cities are drawn in the order of the tour:



Its length is $L(\sigma) = d_{13} + d_{35} + d_{52} + d_{24} + d_{41} = 2 + 7 + 4 + 3 + 2 = 18$.

6.3 Counting the tours

Proposition 6.2

A TSP instance with $n \geq 3$ cities has

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

distinct tours.

- There are $n!$ permutations of the cities. Each tour is described by n permutations, one for every possible starting city, so there are $n!/n = (n-1)!$ cyclic orderings of the cities.
- The distances are symmetric, so the tour travelled backwards has the same length and is not a new solution: every tour is counted twice, and we divide by 2.

Example 6.2: enumeration of the tours

The instance of [Example 6.1](#) has $\frac{(5-1)!}{2} = 12$ distinct tours, which can be enumerated:

tour	L	tour	L	tour	L
(1, 3, 5, 2, 4)	18	(1, 3, 4, 2, 5)	22	(1, 2, 3, 5, 4)	25
(1, 3, 2, 5, 4)	19	(1, 4, 3, 2, 5)	23	(1, 3, 2, 4, 5)	25
(1, 2, 5, 3, 4)	22	(1, 2, 4, 5, 3)	24	(1, 4, 2, 3, 5)	25
(1, 2, 5, 4, 3)	22			(1, 2, 4, 3, 5)	28
				(1, 2, 3, 4, 5)	29

Hence the tour $\sigma = (1, 3, 5, 2, 4)$ is the unique optimal tour, and $z(I) = 18$.

- The number of tours grows faster than any exponential function:

n	5	10	15	20
$\frac{(n-1)!}{2}$	12	181 440	$\approx 4.4 \cdot 10^{10}$	$\approx 6.1 \cdot 10^{16}$

- A computer that evaluates 10^9 tours per second needs about $6.1 \cdot 10^7$ seconds, that is about two years, to list the tours of 20 cities. Enumeration is therefore useful only as a check on tiny instances, and the algorithms of this course must do better.

6.4 A first tour by hand

- The simplest way to build a tour is the **nearest neighbour rule**: start from a city and, as long as there are unvisited cities, move to the closest unvisited one; at the end, return to the starting city. In case of a tie we take the city with the smallest index.

Example 6.3: nearest neighbour from city 1

On the instance of [Example 6.1](#), starting from city 1:

1. from 1 the closest cities are 3 and 4, both at distance 2: we go to 3;
2. from 3 the unvisited cities 2 and 4 are both at distance 4: we go to 2;
3. from 2 the closest unvisited city is 4, at distance 3; then only 5 is left, at distance $d_{45} = 7$;
4. we return from 5 to 1 at distance $d_{51} = 9$.

The tour is $\sigma = (1, 3, 2, 4, 5)$, of length $2 + 4 + 3 + 7 + 9 = 25$.

- The last edges of the nearest neighbour tour are forced, and may be very long: here the final edge $\{5, 1\}$ costs 9. The tour of length 25 is worse than the optimal one, of length 18, by 7.
- From the nearest neighbour tour alone we cannot tell how far we are from the optimum: we need a lower bound.

6.5 A combinatorial lower bound

- For a city i , let $a_i \leq b_i$ be the two smallest distances from i to the other cities.

Observation 6.3

For every symmetric TSP instance, the optimal length satisfies

$$z(I) \geq \left\lceil \frac{1}{2} \sum_{i \in \mathcal{V}} (a_i + b_i) \right\rceil.$$

- In a tour, every city is entered by one edge and left by another one, so it has exactly two edges of the tour, each at least as long as a_i and b_i respectively.
- Summing over all the cities, every edge of the tour is counted twice, once for each of its end cities: hence $2L(\sigma) \geq \sum_i (a_i + b_i)$ for every tour σ .
- The distances are integer, so is $L(\sigma)$, and we can round the bound up.

Example 6.4: the bound on the reference instance

For the instance of [Example 6.1](#) the two smallest distances of each city are:

city i	1	2	3	4	5	sum
a_i	2	3	2	2	4	
b_i	2	4	4	3	7	
$a_i + b_i$	4	7	6	5	11	33

Hence $z(I) \geq \lceil 33/2 \rceil = 17$. Together with the nearest neighbour tour we obtain

$$17 \leq z(I) \leq 25,$$

an absolute gap of 8, that is a relative gap of $\frac{8}{17} \approx 47.1\%$. The tour $(1, 3, 5, 2, 4)$ of length 18 closes most of it, and only the enumeration of the 12 tours proves that 18 is optimal.

6.6 Variants: metric and asymmetric TSP

6.6.1 The metric travelling salesman problem

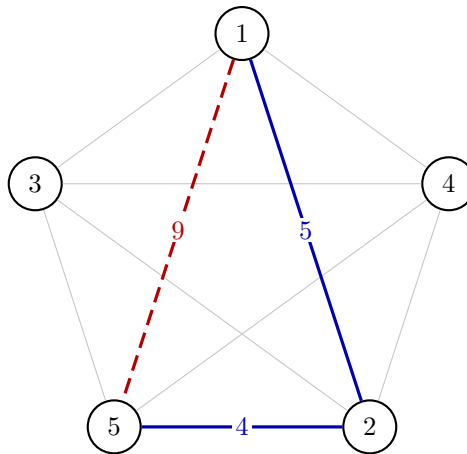
Definition 6.4: Metric TSP

A TSP instance is **metric** if its distances satisfy the **triangle inequality**

$$d_{ij} \leq d_{ik} + d_{kj} \quad \text{for all distinct } i, j, k \in \mathcal{V}.$$

The **metric travelling salesman problem** (metric TSP) is the TSP restricted to metric instances.

- The triangle inequality says that going directly from i to j is never longer than going through a third city k .
- The instance of [Example 6.1](#) is metric: all the $5 \cdot 4 \cdot 3 = 60$ triangle inequalities hold, some of them with equality, e.g., $d_{15} = 9 = d_{12} + d_{25} = 5 + 4$. In the drawing, the direct edge $\{1, 5\}$ (dashed) is as long as the path from 1 to 5 through city 2.



$$d_{15} = 9 \leq d_{12} + d_{25} = 5 + 4$$

- Instances obtained from points in the plane, with d_{ij} equal to the Euclidean distance between the points, are metric (when the distances are not rounded).
- Several heuristic algorithms for the TSP behave well, and some of them have a performance guarantee, only on metric instances.

6.6.2 The asymmetric travelling salesman problem

Definition 6.5: Asymmetric travelling salesman problem

Given:

1. a set $\mathcal{V} = \{1, 2, \dots, n\}$ of n cities,
2. for each ordered pair of distinct cities $i, j \in \mathcal{V}$, a distance $d_{ij} \in \mathbb{Z}_{\geq 1}$ from city i to city j , where d_{ij} may differ from d_{ji} ,

the **asymmetric travelling salesman problem** (ATSP) asks to find a closed route that starts from a city, visits every other city exactly once and returns to the starting city, so that the total distance travelled is minimized.

- Asymmetric distances arise, for example, when travelling along one-way streets or uphill and downhill.
- The definition of tour and of its length $L(\sigma)$ is unchanged, but the direction now matters: a tour and the same tour travelled backwards are different solutions.
- The TSP is the special case of the ATSP in which $d_{ij} = d_{ji}$ for all i, j .
- An ATSP instance has $\frac{n!}{n} = (n-1)!$ distinct tours, twice the count of [Proposition 6.2](#): the starting city still does not change the tour, but the direction does, so we do not divide by 2.
- The instance of [Example 6.1](#), read as an ATSP instance, has $4! = 24$ tours instead of 12.

6.7 Remarks

- The largest instance solved to optimality, pla85900, has 85 900 points from a chip design problem. It was solved in 2005–2006 with the Concorde code; the optimal tour, of length 142 382 641, was certified in 2009 by Applegate, Bixby, Chvátal, Cook, Espinoza, Goycoolea and Helsgaun.
- On the road maps of real countries, the record of 2025 is the instance korea81998, with 81 998 bars of South Korea.
- The TSP is strongly NP-hard ([Chapter 1 «Combinatorial optimization problems»](#)). Christofides (1976) gave an approximation algorithm with ratio $\frac{3}{2}$ for the metric TSP.

6.8 Typical mistakes

Warning

- **Counting $n!$ tours.** The starting city does not change the tour, and in the symmetric case neither does the direction: the number of tours is $(n-1)!/2$.
- **Forgetting the return edge.** The length of a tour includes the last distance $d_{\sigma_n \sigma_1}$, from the last city back to the first one.
- **Taking two cycles for a tour.** Two disjoint cycles that together visit every city once are not a tour: a tour is a single cycle through all the cities.
- **Reading the bound as a tour.** The value 17 of the two-shortest-edges bound is not the length of a tour; it only says that no tour is shorter.
- **Using the triangle inequality without checking it.** The guarantees of the heuristic algorithms hold only for metric instances.

6.9 In summary

Summary

- The TSP asks for a shortest closed route through all the cities; a solution is a cyclic ordering of the cities, called a tour.
- A symmetric instance with n cities has $(n-1)!/2$ tours, an asymmetric one $(n-1)!$; listing them is possible only for very small n .
- The nearest neighbour rule gives a tour quickly, with no guarantee; the two shortest edges of every city give a lower bound on the optimal length.
- The metric TSP, with the triangle inequality, is the case where approximation guarantees exist.

Dashboard of the TSP instance (minimization: the lower bound is a dual bound, the upper bound is the length of a tour):

Chapter	Lower bound	Upper bound	From
This chapter	17	25	two shortest edges; nearest neighbour from city 1
This chapter, enumeration	18	18	the 12 tours

Scheduling problems

Prerequisites

- From this module, [Chapter 1 «Combinatorial optimization problems»](#): what a combinatorial optimization problem is, the number of candidate solutions, primal and dual bounds, and the partitions of a set (Bell numbers).
- From this module, [Chapter 3 «Packing problems»](#): the bin packing problem.

Objectives

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

- classify a scheduling problem with the three-field notation, and state the problem $P \parallel C_{\max}$;
- describe a schedule by the sets of jobs of the machines, and compute loads and makespan;
- count the assignments and the distinct schedules of an instance;
- build a first schedule by hand, assigning every job to the least loaded machine;
- compute a lower bound on the optimal makespan and use it to prove that a schedule is optimal.

7.1 Overview of the family

- The **scheduling problems** of this chapter share the same data: a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n jobs, m parallel machines and, for each job, a *processing time*.
- A solution assigns every job to a machine: the jobs J_h of machine h are processed one after the other, and the machine completes them at the time L_h , its **load**.
- The objective is to minimize the **makespan** $C_{\max} = \max_h L_h$, the completion time of the last job.

A parallel machine scheduling problem partitions the jobs among the machines so that the last machine finishes as early as possible. The members of the family differ in the machine environment, described by the three-field notation.

- The table lists the members of the family, with the section that defines each of them.

Problem	Notation	Machines	Section
identical parallel machines	$P \parallel C_{\max}$	every job takes t_j on every machine	7.2
machines with different speeds	$Q \parallel C_{\max}$	machine h has its own speed	7.1.1
unrelated parallel machines	$R \parallel C_{\max}$	job j takes t_{jh} on machine h	7.1.1

7.1.1 The three-field notation

- Scheduling problems are classified with the **three-field notation** $\alpha | \beta | \gamma$ of Graham, Lawler, Lenstra and Rinnooy Kan, where:
 1. α describes the **machine environment**: 1 for a single machine, P for identical parallel machines, Q for parallel machines with different speeds, R for unrelated parallel machines;
 2. β describes the **job characteristics**, such as release dates r_j or precedence constraints $prec$; an empty field means that there are none;
 3. γ describes the **objective function**, such as the makespan $C_{\max}$ or the total completion time $\sum_j C_j$.
- In this notation, $P \parallel C_{\max}$ is the problem of scheduling jobs without additional characteristics on identical parallel machines, minimizing the makespan.

7.2 Scheduling on identical parallel machines

Definition 7.1: Scheduling on identical parallel machines

Given:

1. a set of m identical *machines*,
2. a set $\mathcal{J} = \{1, 2, \dots, n\}$ of n *jobs*,
3. for each job $j \in \mathcal{J}$, a *processing time* $t_j \in \mathbb{Z}_{\geq 1}$,

the problem $P \parallel C_{\max}$ asks to assign each job to a machine, so that the time needed to complete all the jobs, i.e., the largest total processing time assigned to a machine, is minimized.

- The name $P \parallel C_{\max}$ comes from the three-field notation of scheduling problems (Section 7.1.1).
- A $P \parallel C_{\max}$ **instance** is specified by:
 1. two integer numbers:
 - the number of machines $m \in \mathbb{Z}_{\geq 1}$,
 - the number of jobs $n \in \mathbb{Z}_{\geq 1}$,
 2. one vector of integer numbers of length n :
 - $\mathbf{t} \in \mathbb{Z}_{\geq 1}^n$, containing the processing times.
- A $P \parallel C_{\max}$ **feasible solution** is an assignment of the jobs to the machines, which we store in a vector $\text{mach} = (\text{mach}_1, \text{mach}_2, \dots, \text{mach}_n)$:

$$\text{mach}_j = h \iff \text{job } j \text{ is assigned to machine } h.$$

The set of jobs assigned to machine h is $J_h = \{j \in \mathcal{J} : \text{mach}_j = h\}$, and its **load** is

$$L_h = \sum_{j \in J_h} t_j.$$

Each machine processes its jobs one after the other, without idle times, so machine h completes its jobs at time L_h . The **makespan** of the solution is the completion time of the last job,

$$C_{\max} = \max_{h \in \{1, 2, \dots, m\}} L_h.$$

- The order in which a machine processes its jobs does not change its load: a solution is fully described by the sets $J_1, J_2, \dots, J_m$, which form a partition of $\mathcal{J}$ (some of them may be empty).

Given a $P \parallel C_{\max}$ instance, the problem can be formally stated as:

$$\min \left\{ \max_{h \in \{1, \dots, m\}} \sum_{j \in J_h} t_j : J_1, J_2, \dots, J_m \text{ is a partition of } \mathcal{J} \right\}. \quad (7.1)$$

- We denote by $z(I)$ the optimal makespan of the instance I .

7.2.1 Assumptions

- We make the following assumption:
 1. There are more jobs than machines (otherwise each job gets a machine of its own, and the optimal makespan is $\max_j t_j$). Formally:

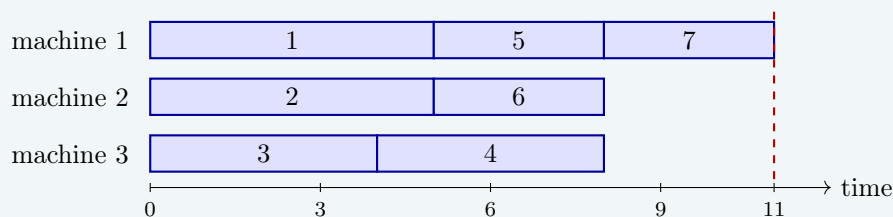
$$n > m$$

Example 7.1: $P \parallel C_{\max}$ instance

We consider $m = 3$ machines and a set $\mathcal{J} = \{1, 2, \dots, 7\}$ of 7 jobs ($n = 7$), with the following processing times:

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$	$j = 7$
t_j	5	5	4	4	3	3	3

We call this instance [Example 7.1](#): it is the reference instance of the chapter. The assignment $\text{mach} = (1, 2, 3, 3, 1, 2, 1)$ gives $J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$, with loads $L_1 = 11$, $L_2 = 8$, $L_3 = 8$ and makespan 11:



7.3 Counting the solutions

- Every job can be assigned to any of the m machines, independently of the others: there are m^n assignments mach.
- The machines are identical, so two assignments that differ only by the names of the machines describe the same schedule. The distinct schedules are the partitions of $\mathcal{J}$ into at most m non-empty sets, and their number is

$$\sum_{k=1}^m S(n, k),$$

where $S(n, k)$ is the Stirling number of the second kind, the number of partitions of n elements into exactly k non-empty sets.

Example 7.2: counting the schedules

For the instance of [Example 7.1](#) there are $3^7 = 2187$ assignments. The Stirling numbers are $S(7, 1) = 1$, $S(7, 2) = 63$ and $S(7, 3) = 301$, so the distinct schedules are

$$1 + 63 + 301 = 365.$$

By enumeration, the schedules with the smallest makespan, 9, are exactly two:

$$\{1, 3\}, \{2, 4\}, \{5, 6, 7\} \quad \text{and} \quad \{1, 4\}, \{2, 3\}, \{5, 6, 7\},$$

which give $3! \cdot 2 = 12$ of the 2187 assignments. The optimal schedule is therefore not unique.

- The growth is exponential in n : with 3 machines, 20 jobs have $3^{20} \approx 3.5 \cdot 10^9$ assignments, and 30 jobs have $3^{30} \approx 2.1 \cdot 10^{14}$.

7.4 A first schedule by hand

- A simple way to build a schedule is to take the jobs one by one, in the order $1, 2, \dots, n$, and to assign each job to the machine with the smallest load at that moment; in case of a tie we take the machine with the smallest index.

Example 7.3: assigning every job to the least loaded machine

On the instance of [Example 7.1](#) the loads of the three machines evolve as follows:

job j (t_j)	machine chosen	loads (L_1, L_2, L_3)
1 (5)	1	(5, 0, 0)
2 (5)	2	(5, 5, 0)
3 (4)	3	(5, 5, 4)
4 (4)	3	(5, 5, 8)
5 (3)	1	(8, 5, 8)
6 (3)	2	(8, 8, 8)
7 (3)	1	(11, 8, 8)

We obtain the assignment $\text{mach} = (1, 2, 3, 3, 1, 2, 1)$ of the example, with makespan 11.

- When the last job arrives all the machines have load 8: whichever machine receives it, the largest load becomes $8 + 3 = 11$.
- We do not know yet whether 11 is far from the optimal makespan: we need a lower bound.

7.5 A combinatorial lower bound

Observation 7.2

For every $P \parallel C_{\max}$ instance, the optimal makespan $z(I)$ satisfies

$$z(I) \geq \max \left\{ \max_{j \in \mathcal{J}} t_j, \left\lceil \frac{\sum_{j \in \mathcal{J}} t_j}{m} \right\rceil \right\}.$$

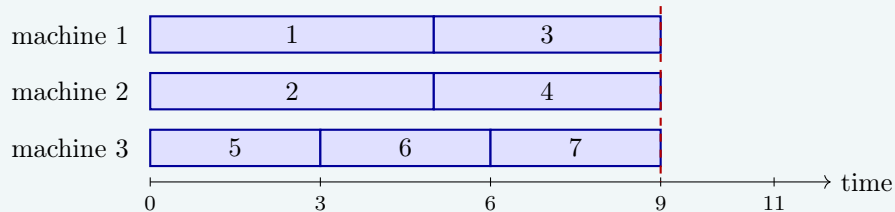
- The machine processing the longest job has load at least $\max_j t_j$.
- The m loads sum up to $\sum_j t_j$, so the largest one is at least their average $\frac{1}{m} \sum_j t_j$, and, being an integer, at least its ceiling.

Example 7.4: the bound on the reference instance

On the instance of [Example 7.1](#) the bound is $\max\{5, \lceil 27/3 \rceil\} = 9$. Together with the schedule of Section 7.4,

$$9 \leq z(I) \leq 11.$$

The assignment $\text{mach} = (1, 2, 1, 2, 3, 3, 3)$ gives $J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$, all with load 9, and makespan 9:



It attains the lower bound, so it is optimal: $z(I) = 9$. The three machines finish together,

and no idle time is left before the makespan.

- The bound is not always attained. With $m = 2$ machines and three jobs of processing times $(4, 3, 3)$ it gives $\max\{4, \lceil 10/2 \rceil\} = 5$, but the best schedule is $\{1\}, \{2, 3\}$, with makespan 6.

7.6 Relation with the bin packing problem

- Given a value T , a schedule with makespan at most T exists if and only if the jobs, seen as items of weight t_j , can be packed into m bins of capacity T : the machines are the bins and the loads are the total weights of the packing patterns.
- The two problems fix and minimize opposite quantities:

	number of bins / machines	capacity / makespan
BPP	minimized	fixed (c)
$P \parallel C_{\max}$	fixed (m)	minimized

- For this reason the two problems share many ideas: several heuristic algorithms for one of them have a counterpart for the other.

7.7 Remarks

- In 1966 R. L. Graham proved that any list of jobs, processed greedily, gives a makespan at most $2 - 1/m$ times the optimal one.
- In 1969 he proved the better ratio $4/3 - 1/(3m)$ for the list sorted by non-increasing processing time (the rule called LPT).

7.8 Typical mistakes

Warning

- **Counting m^n different schedules.** The machines are identical: renaming them gives the same schedule, so the distinct schedules are the partitions of the jobs into at most m sets.
- **Confusing makespan and total load.** The makespan is the largest load of a machine, not the sum of the processing times and not the average load.
- **Forgetting the ceiling.** The processing times are integer, so is the makespan: the average load $27/3$ is rounded up when it is not an integer.
- **Assuming that the bound is attained.** The bound of Section 7.5 is a dual bound: a schedule with that makespan may not exist, as the instance $(4, 3, 3)$ on two machines shows.

- **Believing that the optimal schedule is unique.** The instance of [Example 7.1](#) has two optimal schedules, and twelve optimal assignments.

7.9 In summary

Summary

- The problem $P \parallel C_{\max}$ asks to assign jobs to identical machines minimizing the largest load, the makespan; a solution is a partition of the jobs into at most m sets.
- There are m^n assignments, but only $\sum_{k=1}^m S(n, k)$ distinct schedules.
- Giving each job to the least loaded machine builds a schedule quickly; the maximum between the longest job and the rounded average load is a lower bound on the optimal makespan.
- When a schedule attains the lower bound it is optimal; otherwise the gap remains open.
- The problem is the mirror of bin packing: here the number of machines is fixed and the capacity is minimized.

Dashboard of the $P \parallel C_{\max}$ instance (minimization: the lower bound is a dual bound, the upper bound is the makespan of a schedule):

Chapter	Lower bound	Upper bound	From
This chapter	9	11	longest job and average load; least loaded machine
This chapter, improved schedule	9	9	$J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$

Lot sizing problems

Prerequisites

- From this module, [Chapter 1 «Combinatorial optimization problems»](#): what a combinatorial optimization problem is, the number of candidate solutions, and primal and dual bounds.

Objectives

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

- describe a production plan by its quantities, stocks and setups, and compute its cost;
- state the capacitated lot sizing problem and name its main variants;
- decide whether a plan is feasible, and count the feasible setup choices of an instance;
- build a first plan by hand, merging the demands of consecutive periods;
- compute a lower bound on the cost from the number of setups that the total demand requires.

8.1 Overview of the family

- The **lot sizing problems** share the same data: a set $T = \{1, 2, \dots, m\}$ of m time periods and, for each period $t \in T$, a *demand* d_t , a production *capacity* and three costs: the setup cost sc_t , the variable production cost vc_t and the holding cost hc_t .
- A solution is a **production plan**: the quantity produced in every period, which determines the stock at the end of each period and the periods with a setup.
- The objective is to satisfy all the demands at the minimum total cost of setups, production and holding.

A lot sizing problem balances the setup costs, paid in every period with production, against the holding costs of the stock. The members of the family differ in the capacities, in the number of products and in the way the demand and the capacity can be adjusted.

- The table lists the members of the family, with the section that defines each of them.

Problem	Acronym	Difference with respect to the CLS	Section
capacitated lot sizing problem	CLS	one product, capacities cap_t	8.2
uncapacitated lot sizing problem	ULS	no binding capacity	8.6
several products and setup times	MLS	products sharing the time of each period	8.6
initial stock	LS-SI	the initial stock is a decision	8.6
overtime	LS-OT	extra capacity at a higher cost	8.6
unsatisfied demand	LS-UD	part of the demand lost at a penalty	8.6

8.2 The capacitated lot sizing problem

Definition 8.1: Capacitated lot sizing problem

Given:

1. a set $T = \{1, 2, \dots, m\}$ of m time periods,
2. for each period $t \in T$, a demand $d_t \in \mathbb{Z}_{\geq 1}$ and a capacity $cap_t \in \mathbb{Z}_{\geq 1}$, both counted in units of one product,
3. for each period $t \in T$, three non-negative integer costs: the *setup cost* sc_t , the *variable production cost* vc_t per unit produced and the *holding cost* hc_t per unit kept in stock at the end of the period,

the **capacitated lot sizing problem** (CLS) asks to decide how much to produce in every period, so that every demand is satisfied with what is produced and what is in stock, the capacities are respected, and the total cost of setups, production and holding is minimized.

- A **CLS instance** is specified by:
 1. one integer number:
the number of periods $m \in \mathbb{Z}_{\geq 1}$,
 2. five vectors of integer numbers of length m :
 $\mathbf{d}$, $\mathbf{cap} \in \mathbb{Z}_{\geq 1}^m$, containing demands and capacities,
 $\mathbf{sc}$, $\mathbf{vc}$, $\mathbf{hc} \in \mathbb{Z}_{\geq 0}^m$, containing the costs.

- A **CLS feasible solution** is a **production plan**, that is a vector of integer quantities

$$\mathbf{x} = (x_1, x_2, \dots, x_m) \in \mathbb{Z}_{\geq 0}^m,$$

where x_t is the quantity produced in period t . The stock left at the end of period t is

$$s_t(\mathbf{x}) = \sum_{u=1}^t (x_u - d_u),$$

because the stock of period $t - 1$ plus the production of period t must cover the demand of period t , and what remains is the stock of period t . The plan is **feasible** if

$$x_t \leq \text{cap}_t \quad \text{and} \quad s_t(\mathbf{x}) \geq 0 \quad \forall t \in T, \quad s_m(\mathbf{x}) = 0.$$

- There is a **setup** in period t when $x_t > 0$: we write $y_t = 1$ in this case and $y_t = 0$ otherwise. The **cost** of a plan is

$$c(\mathbf{x}) = \sum_{t \in T} (sc_t y_t + vc_t x_t + hc_t s_t(\mathbf{x})).$$

Given a CLS instance, the CLS can be formally stated as:

$$\tilde{\mathbf{x}} \in \arg \min_{\mathbf{x}} \{c(\mathbf{x}) : x_t \leq \text{cap}_t, s_t(\mathbf{x}) \geq 0, \forall t \in T, s_m(\mathbf{x}) = 0, \mathbf{x} \in \mathbb{Z}_{\geq 0}^m\}. \quad (8.1)$$

- We denote by $z(I)$ the cost of an optimal plan $\tilde{\mathbf{x}}$ of the instance I .

Example 8.1: CLS instance

We consider $m = 5$ periods, with the following data, all integer:

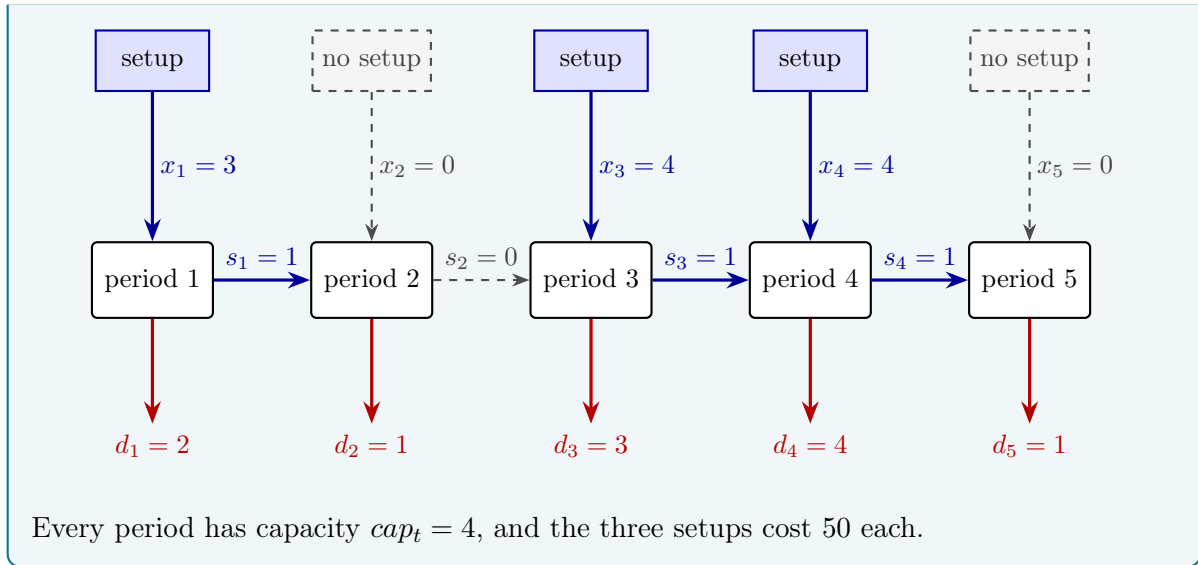
period t	1	2	3	4	5
demand d_t	2	1	3	4	1
capacity cap_t	4	4	4	4	4
setup cost sc_t	50	50	50	50	50
production cost vc_t	0	0	0	0	0
holding cost hc_t	10	10	10	10	10

The total demand is $2 + 1 + 3 + 4 + 1 = 11$. We call this CLS instance [Example 8.1](#): it is the reference instance of the chapter.

The plan $\mathbf{x} = (3, 0, 4, 4, 0)$ is feasible: the stocks are $s = (1, 0, 1, 1, 0)$, all non-negative, with $s_5 = 0$, and every quantity is at most 4. There are 3 setups, in the periods 1, 3 and 4:

$$c(\mathbf{x}) = 3 \cdot 50 + 10 \cdot (1 + 0 + 1 + 1 + 0) = 150 + 30 = 180.$$

The plan is drawn below, period by period from left to right: the production x_t enters a period from above, through a setup (solid) or not at all (dashed); the stock s_t passes to the next period; the demand d_t leaves the period from below.



8.2.1 Assumptions

- We make the following assumptions:
 1. The stock is empty at the beginning and at the end of the horizon. Leaving stock at the end only adds cost, so no optimal plan needs it; an initial stock is a variant of the problem (Section 8.6).
 2. The capacities allow to satisfy the demands, period after period (otherwise there is no feasible plan). Formally:

$$\sum_{u=1}^t cap_u \geq \sum_{u=1}^t d_u, \quad \forall t \in T$$
 3. The quantities are integer. With integer data, once the periods with a setup are fixed, what remains is a minimum cost flow problem, which has an integer optimal solution: restricting the quantities to integers does not change the optimal cost.

8.3 Counting the solutions

- The combinatorial part of the decision is the choice of the periods with a setup: there are 2^m subsets of periods. Not all of them are feasible, because the periods with a setup must have enough capacity to produce everything.
- For a given choice of the setup periods there may be many production plans, since the quantities can be distributed in many ways: the number of feasible plans is larger than the number of feasible setup choices.

Example 8.2: counting the plans

For the instance of [Example 8.1](#) there are $2^5 = 32$ subsets of periods. Let k_t be the number of setups in the periods $1, \dots, t$: the cumulative demands 2, 3, 6, 10, 11 require, with capacity

4, at least one setup in the periods 1–2, two in the periods 1–3 and three in the periods 1–4. Since $d_1 > 0$ there is a setup in period 1, and in the periods 2, 3, 4 we need at least two setups, one of which in the periods 2 or 3:

$$(y_2, y_3, y_4) \in \{(1, 1, 0), (1, 0, 1), (0, 1, 1), (1, 1, 1)\}.$$

The period 5 is free, so the feasible setup choices are $4 \cdot 2 = 8$ out of 32.

By enumeration, these setup choices carry 94 feasible plans. The costs of six of them are:

plan	cost	plan	cost	plan	cost
(3, 0, 4, 4, 0)	180	(4, 0, 3, 4, 0)	200	(2, 1, 4, 4, 0)	220
(3, 0, 3, 4, 1)	210	(4, 0, 4, 3, 0)	210	(2, 1, 3, 4, 1)	250

The plan (3, 0, 4, 4, 0) is the unique optimal plan, and $z(I) = 180$.

- Without the capacities the same instance has $2^{m-1} = 16$ plans that produce only when the stock is empty, and the optimal cost is 170: here the capacity costs 10.
- The optimal plan (3, 0, 4, 4, 0) produces in period 4 with a stock of 1 unit, so the property of Wagner and Whitin does not hold when the capacities are binding.
- The number of setup choices grows as 2^m : with $m = 52$ weeks it is more than $4 \cdot 10^{15}$.

8.4 A first plan by hand

- The simplest feasible plan is **lot-for-lot**: in every period we produce exactly the demand of that period. There is no stock, and there are m setups; it is feasible when every demand fits in the capacity of its period.
- A plan is improved by **merging**: the demand of a period is produced in an earlier period, which saves a setup and costs the stock of the units anticipated.

Example 8.3: from lot-for-lot to a better plan

On the instance of [Example 8.1](#):

1. the lot-for-lot plan (2, 1, 3, 4, 1) has 5 setups and no stock: cost $5 \cdot 50 = 250$;
2. producing the demand of period 2 in period 1 saves one setup (-50) and keeps 1 unit for one period ($+10$): the plan (3, 0, 3, 4, 1) costs 210;
3. period 4 cannot produce the demand of period 5 as well, because $4 + 1 > cap_4 = 4$; but period 3 can anticipate one unit of period 4, so that period 4 has room for the demand of period 5: the plan (3, 0, 4, 4, 0) saves a setup (-50) and keeps two more units ($+20$).

The cost is $210 - 50 + 20 = 180$.

- We have found the plan of cost 180 by hand, but we cannot yet tell whether a cheaper plan exists: we need a lower bound.

8.5 A combinatorial lower bound

- Let $D = \sum_{t \in T} d_t$ be the total demand. Let k be the smallest number of periods whose capacities sum up to at least D , which is found by taking the periods with the largest capacities.

Observation 8.2

For every CLS instance, every feasible plan has at least k setups, and

$$z(I) \geq \text{the sum of the } k \text{ smallest setup costs.}$$

- The plan must produce D units in the periods with a setup, and a period without setup produces nothing: the capacities of the periods with a setup sum up to at least D , hence they are at least k .
- The costs of production and holding are non-negative: dropping them, and paying only k setups, the cheapest ones, gives a bound on the cost of every plan.

Example 8.4: the bound on the reference instance

For the instance of [Example 8.1](#) we have $D = 11$ and all capacities equal to 4, so $k = \lceil 11/4 \rceil = 3$. All the setup costs are 50, hence

$$z(I) \geq 3 \cdot 50 = 150.$$

Together with the plans of the previous section we obtain

$$150 \leq z(I) \leq 180,$$

an absolute gap of 30, that is a relative gap of $\frac{30}{150} = 20\%$. The enumeration of the 94 plans proves that 180 is optimal.

8.6 Variants

- Each variant keeps the data of the CLS (periods, demands, costs) and changes one feature of the problem. The definitions below give only what changes.

8.6.1 The uncapacitated lot sizing problem

Definition 8.3: Uncapacitated lot sizing problem

The **uncapacitated lot sizing problem** (ULS) is the CLS in which the capacities are not binding, for example $cap_t \geq \sum_{u \in T} d_u$ for every $t \in T$.

- The ULS is solved in polynomial time by dynamic programming.

8.6.2 Several products and setup times

Definition 8.4: Lot sizing with several products and setup times

In the **lot sizing problem with several products and setup times** (MLS) a set P of products shares the resource of each period: d_{pt} is the demand of product $p \in P$ in period t , and cap_t is the time available in period t . Producing a unit of p takes vt_{pt} units of time, and a setup of p costs sc_{pt} and takes st_{pt} units of time.

8.6.3 Initial stock

Definition 8.5: Lot sizing with initial stock

In the **lot sizing problem with initial stock** (LS-SI) the quantity si_p of product p available at the beginning of the horizon is a decision, bought at the cost fc_p per unit.

8.6.4 Overtime

Definition 8.6: Lot sizing with overtime

In the **lot sizing problem with overtime** (LS-OT) the capacity is not fixed: working time in normal hours costs cw_t per unit up to cap_t^N , and overtime costs co_t per unit up to cap_t^O .

8.6.5 Unsatisfied demand

Definition 8.7: Lot sizing with unsatisfied demand

In the **lot sizing problem with unsatisfied demand** (LS-UD) a part of the demand may be left unsatisfied, at a penalty cu_{pt} per unit.

8.7 Remarks

- In 1958 Wagner and Whitin gave a dynamic programming algorithm for the case without capacity and with demand that varies over time. It rests on a property of the optimal plans: one produces only when the stock is empty.
- With capacities the property fails, and the problem is NP-hard ([Chapter 1 «Combinatorial optimization problems»](#)).

8.8 Typical mistakes

Warning

- **Producing late.** The production of a period cannot serve the demand of an earlier period: the stock $s_t(\mathbf{x})$ must be non-negative in every period.
- **Charging a setup when nothing is produced.** A setup cost is paid only in the periods with $x_t > 0$.
- **Forgetting the stock of the last periods.** The holding cost is paid on the stock at the end of every period, including the periods in which the plan produces nothing.
- **Believing that one produces only with empty stock.** The property of Wagner and Whitin holds without capacities; with capacities the optimal plan of [Example 8.1](#) violates it.
- **Counting 2^m feasible plans.** The 2^m subsets of periods are the setup choices: some are infeasible, and each feasible one carries one or more plans.

8.9 In summary

Summary

- The capacitated lot sizing problem asks for the production quantities of every period, with capacities, setups and stock, at the minimum total cost; a solution is a production plan.
- The combinatorial part is the choice of the periods with a setup: there are 2^m choices, only some of them feasible, and each carries one or more plans.
- Lot-for-lot gives a plan quickly, merging the demands improves it; the number of setups that the total demand requires gives a lower bound.
- With capacities, optimal plans may produce while the stock is positive; without capacities they never do.
- The variants add products, setup times, initial stock, overtime or unsatisfied demand; their models are in [Chapter 15 «Formulations of lot sizing problems»](#).

Dashboard of the CLS instance (minimization: the lower bound is a dual bound, the upper bound is the cost of a plan):

Chapter	Lower bound	Upper bound	From
This chapter	150	250	three setups at least; lot-for-lot
This chapter, merging	150	180	the plan (3, 0, 4, 4, 0)
This chapter, enumeration	180	180	the 94 feasible plans