



SAPIENZA
UNIVERSITÀ DI ROMA

Combinatorial Optimization

Catalogue of combinatorial problems

All the problems defined in the course, grouped by family: for each one its definition, its assumptions and what the course does on it, technique by technique.

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	Knapsack problems	3
1.1	The 0/1 knapsack problem (KP)	4
1.2	Subset sum problem (SSP)	6
1.3	Bounded knapsack problem (BKP)	7
1.4	Unbounded knapsack problem (UKP)	8
1.5	Multidimensional knapsack problem (MDKP)	9
1.6	Multiple knapsack problem (MKP)	10
1.7	Multiple-choice knapsack problem (MCKP)	11
1.8	Knapsack problem with setups (KPS)	12
2	Packing problems	14
2.1	The bin packing problem (BPP)	14
2.2	Cutting stock problem (CSP)	16
3	Coloring problems	17
3.1	The vertex coloring problem (VCP)	17
4	Clique problems	20
4.1	The maximum clique problem (MCP)	20
4.2	Maximum stable set problem (MSSP)	22
4.3	Vertex-weighted maximum clique problem (VWMCP)	22
4.4	Edge-weighted maximum clique problem (EWMCP)	23
5	Routing problems	24
5.1	The travelling salesman problem (TSP)	24
5.2	Metric travelling salesman problem (metric TSP)	26
5.3	Asymmetric travelling salesman problem (ATSP)	27
5.4	Hamiltonian cycle problem (HCP)	27
6	Scheduling problems	28
6.1	Scheduling on identical parallel machines ($P C_{\max}$)	28
6.2	Scheduling on a fixed number of identical machines ($Pm C_{\max}$)	30
6.3	Scheduling on parallel machines with different speeds ($Q C_{\max}$)	30
6.4	Scheduling on unrelated parallel machines ($R C_{\max}$)	31

7	Lot sizing problems	32
7.1	Capacitated lot sizing (CLS)	33
7.2	Uncapacitated lot sizing problem (ULS)	34
7.3	Capacitated lot sizing with constant capacity (CLS-C)	35
7.4	Lot sizing with several products and setup times (MLS)	35
7.5	Lot sizing with initial stock (LS-SI)	36
7.6	Lot sizing with overtime (LS-OT)	36
7.7	Lot sizing with unsatisfied demand (LS-UD)	37
8	Miscellaneous problems	38
8.1	Rod cutting problem (RCP)	38
8.2	Coin change problem (CCP)	39

Knapsack problems

- Each item j has a profit p_j and a weight w_j ; a solution selects a subset, or a multiset, of the items whose weights satisfy one or more capacity constraints.
- The objective is to maximize the total profit of the selected items; in the subset sum problem the total weight must be equal to a target value.
- The members differ in the number of copies of each item, in the number of capacities and knapsacks, and in additional constraints on the selected items.

Problem	Acronym	Difference
0/1 knapsack problem	KP	The central problem of the family.
Subset sum problem	SSP	The SSP is solved as the KP with $p_j = w_j$ for all $j \in \mathcal{J}$: the answer is yes if and only if the optimal value of this KP equals c .
Bounded knapsack problem	BKP	The BKP reduces to the KP when $b_j = 1$ for every item, and to the UKP when every bound b_j is large enough not to be binding.
Unbounded knapsack problem	UKP	The UKP is the BKP with $b_j = \hat{b}_j = \lfloor c/w_j \rfloor$: each item can be taken as many times as it fits.
Multidimensional knapsack problem	MDKP	The MDKP is the KP with k capacity constraints, one for each resource; with $k = 1$ it is the KP.
Multiple knapsack problem	MKP	The MKP is the KP with m knapsacks; every feasible solution of the MKP is feasible for the KP with capacity $\sum_{k \in \mathcal{K}} c_k$, which gives a dual bound.
Multiple-choice knapsack problem	MCKP	The MCKP is the KP with the additional constraint that exactly one item of every class is selected, so an instance can be infeasible.
Knapsack problem with setups	KPS	The KPS is the KP in which selecting the first item of a class costs the setup f_h and consumes the setup weight s_h ; a class with no selected item costs nothing.

1.1 The 0/1 knapsack problem (KP)

Definition 1.1: The 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.

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:

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

1.1.1 What the course does on this problem

Technique	Where
Definition	Ch. 2 «Knapsack problems»
Formulations	Ch. 9 «Formulations of knapsack problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 17 «DP for the 0/1 knapsack problem»
Heuristics	Ch. 29 «Heuristic algorithms: definitions»; Ch. 30 «Constructive heuristics»; Ch. 31 «Local search»; Ch. 32 «Metaheuristics»
MIP heuristics	Ch. 33 «MIP heuristics for the classical problems»
Approximation	Ch. 34 «Principles of approximation algorithms»; Ch. 35 «Approximation algorithms for the knapsack problem»
Ad hoc algorithms	Ch. 41 «Knapsack: branch-and-bound on the critical item»
Exercises	Ch. 2 (6 exercises); Ch. 9 (5 exercises); Exercise 16.1; Ch. 17 (5 exercises); Ch. 29 (2 exercises); Ch. 30 (3 exercises); Ch. 31 (3 exercises); Ch. 32 (5 exercises); Ch. 33 (4 exercises); Exercise 34.4; Ch. 35 (15 exercises); Ch. 41 (4 exercises)

Properties

- Weakly NP-hard: dynamic programming solves it in $\Theta(nc)$ steps, a pseudo-polynomial time (Ch. 1).
- Cardinality bound: if at most $\bar{k}$ items fit, $z(I)$ is at most the sum of the $\bar{k}$ largest profits (Ch. 2).
- Sorting by efficiency is not enough: on the instance of Ch. 2 it gives profit 11, while the optimum is 15 (Ch. 2).

Formulations

- The model ILP_{KP} has one binary variable per item and one capacity constraint (Ch. 9).
- The relaxation LP_{KP} is solved by sorting the items by efficiency: the items before the split item s are taken, s only in part (Ch. 9).
- $z(LP_{KP}) \leq 2z(ILP_{KP})$, and the ratio 2 is approached (Ch. 9).
- Rounding down $z(LP_{KP})$ gives the Dantzig bound UB_D ; the Martello–Toth bound UB_{MT} treats the split item out and in (Ch. 9).

Dynamic programming

- The recurrence $v(j, k)$ over items and capacities takes $\Theta(nc)$ steps (Ch. 17).
- The lists of non-dominated states keep only the reachable states, at most $\min\{2^j, c + 1\}$ per stage (Ch. 17).
- The DP by profits takes $\Theta(nP)$ steps; it is the core of the FPTAS (Ch. 17).
- The UKP and the BKP have their own recurrences (Ch. 18, Ch. 19).
- So do the SSP, the MCKP and the KPS, in $\Theta(nc)$ steps; for the MDKP and the MKP the table grows exponentially with the number of capacities (Ch. 20, Ch. 21, Ch. 22, Ch. 23, Ch. 24).

Heuristics

- Greedy by profit, by weight and by efficiency: on the four-item instance of Ch. 29 (optimum 17) they give 17, 10 and 16 (Ch. 30).
- Flip and swap local search: first and best improvement reach 17 along different paths (Ch. 31).
- A penalty on the excess weight lets the search cross infeasible solutions; with $\alpha > \max_j p_j$ its local optima are feasible (Ch. 31).
- Simulated annealing, GRASP, genetic and memetic algorithms; on KP20 the memetic algorithm reaches the optimum 521 in every run (Ch. 32).

MIP heuristics

- Rounding down the LP solution is always feasible, since the model has a single packing constraint (Ch. 33).
- On the instance of Ch. 29 fractional diving ends at 16 and RINS fails, while local branching and proximity search reach and prove the optimum 17 (Ch. 33).

Approximation

- The greedy algorithm has no constant guarantee; Ext-Greedy is a 2-approximation, and the bound is tight (Ch. 35).
- H^ℓ guarantees $\frac{\ell+2}{\ell+1}$ in $O(n^{\ell+1})$ time, which gives a PTAS (Ch. 35).
- The Basic-FPTAS scales the profits and guarantees $\frac{1}{1-\varepsilon}$ in $O(n^2/\varepsilon)$ time (Ch. 35).
- No polynomial algorithm has a fixed absolute guarantee, unless it solves the KP exactly (Ch. 35).

Ad hoc algorithms

- The LP of a node is solved by the Dantzig formulas, without the simplex algorithm, and has one fractional variable: the split item, on which it branches (Ch. 41).

1.2 Subset sum problem (SSP)

Definition 1.2: 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 .

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:

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

1.2.1 What the course does on this problem

Technique	Where
Definition	Ch. 2 «Knapsack problems»
Formulations	Ch. 9 «Formulations of knapsack problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 20 «DP for the subset sum problem»
Exercises	Exercise 2.2; Exercise 9.6; Exercise 16.2; Ch. 20 (5 exercises)

1.3 Bounded knapsack problem (BKP)

Definition 1.3: 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.

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:

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

1.3.1 What the course does on this problem

Technique	Where
Definition	Ch. 2 «Knapsack problems»
Formulations	Ch. 9 «Formulations of knapsack problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 19 «DP for the bounded knapsack problem»
Exercises	Exercise 2.3; Exercise 9.7; Ch. 19 (5 exercises)

1.4 Unbounded knapsack problem (UKP)

Definition 1.4: 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.

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:

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

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

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

1.4.1 What the course does on this problem

Technique	Where
Definition	Ch. 2 «Knapsack problems»
Formulations	Ch. 9 «Formulations of knapsack problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 18 «DP for the unbounded knapsack problem»; Ch. 27 «DP for the coin change problem»; Ch. 28 «DP for the rod cutting problem»
Exercises	Exercise 2.3; Exercise 9.8; Ch. 18 (5 exercises)

1.5 Multidimensional knapsack problem (MDKP)

Definition 1.5: 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.

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,

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

1.5.1 What the course does on this problem

Technique	Where
Definition	Ch. 2 «Knapsack problems»
Formulations	Ch. 9 «Formulations of knapsack problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 22 «DP for the multidimensional knapsack problem»
Exercises	Exercise 9.9; Ch. 22 (5 exercises)

1.6 Multiple knapsack problem (MKP)

Definition 1.6: 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.

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,

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

1.6.1 What the course does on this problem

Technique	Where
Definition	Ch. 2 «Knapsack problems»
Formulations	Ch. 9 «Formulations of knapsack problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 24 «DP for the multiple knapsack problem»
Exercises	Exercise 2.4; Exercise 9.10; Ch. 24 (5 exercises)

1.7 Multiple-choice knapsack problem (MCKP)

Definition 1.7: 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.

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:

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

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:

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

1.7.1 What the course does on this problem

Technique	Where
Definition	Ch. 2 «Knapsack problems»
Formulations	Ch. 9 «Formulations of knapsack problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 21 «DP for the multiple-choice knapsack problem»
Exercises	Exercise 2.5; Exercise 9.11; Exercise 16.5; Ch. 21 (5 exercises)

1.8 Knapsack problem with setups (KPS)

Definition 1.8: 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.

Assumptions

- We make the following assumptions:

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

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

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

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

1.8.1 What the course does on this problem

Technique	Where
Definition	Ch. 2 «Knapsack problems»
Formulations	Ch. 9 «Formulations of knapsack problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 23 «DP for the knapsack problem with setups»
Exercises	Exercise 9.12; Ch. 23 (5 exercises)

Packing problems

- Each item j has a weight w_j ; a solution is a partition of the items into bins of capacity c , such that the total weight of the items in every bin is at most c .
- The objective is to minimize the number of bins; in the cutting stock problem the items are grouped into types with a demand d_j .

Problem	Acronym	Difference
Bin packing problem	BPP	The central problem of the family.
Cutting stock problem	CSP	The CSP is the BPP with d_j identical copies of every item j , and the BPP is the case $d_j = 1$ for all $j \in \mathcal{J}$.

2.1 The bin packing problem (BPP)

Definition 2.1: The 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.

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:

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

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

Formally:

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

2.1.1 What the course does on this problem

Technique	Where
Definition	Ch. 3 «Packing problems»
Formulations	Ch. 10 «Formulations of packing problems»
Heuristics	Ch. 29 «Heuristic algorithms: definitions»; Ch. 30 «Constructive heuristics»; Ch. 31 «Local search»; Ch. 32 «Metaheuristics»
MIP heuristics	Ch. 33 «MIP heuristics for the classical problems»
Approximation	Ch. 36 «Approximation algorithms for bin packing»
Exercises	Ch. 3 (6 exercises); Exercise 7.5; Ch. 10 (5 exercises); Exercise 29.2; Ch. 30 (3 exercises); Ch. 31 (3 exercises); Ch. 32 (2 exercises); Exercise 34.3; Ch. 36 (12 exercises)

Properties

- Strongly NP-hard: no pseudo-polynomial algorithm exists, unless $P = NP$ (Ch. 1).
- Weight bound $LB = \lceil \sum_j w_j / c \rceil$; it is not always tight: three items of weight 6 with $c = 10$ need 3 bins, not 2 (Ch. 3).

Formulations

- The model ILP_{BPP} has mn assignment variables x_{jb} and m activation variables y_b (Ch. 10).
- Both LP relaxations have value $\sum_j w_j / c$: rounded up, the weight bound again (Ch. 10).
- The ratio of the LP bound to the optimum is at least $\frac{1}{2}$, and $\frac{1}{2}$ is approached (Ch. 10).

Heuristics

- First Fit Decreasing and Best Fit Decreasing find the optimum 3 on the instance of Ch. 29; $FFD(I) \leq \frac{11}{9} z(I) + \frac{6}{9}$, tight (Ch. 30).
- The number of bins is flat, so move and swap local search uses the auxiliary objective $\max \sum_b L_b^2$ (Ch. 31).
- VND with move and swap reaches the optimum in 89 of 100 runs on an instance with $n = 12$ (Ch. 31).
- Large neighbourhood search: on BPP24 the exact repair with $q \geq 3$ reaches the optimum 9 in every run (Ch. 32).

MIP heuristics

- Rounding and fractional diving fail on the instance of [Ch. 29](#): the LP spreads an item over several bins ([Ch. 33](#)).
- The feasibility pump always finds a packing, but it may leave empty bins open ([Ch. 33](#)).
- On an instance with $n = 60$ the whole difficulty is primal: the heuristics of Gurobi decide the solution time ([Ch. 33](#)).

Approximation

- Every Any Fit algorithm is half-full, hence a 2-approximation ([Ch. 36](#)).
- $r(NF) = r(WF) = 2$ and $r(FF) = r(BF) = \frac{17}{10}$ ([Ch. 36](#)).

2.2 Cutting stock problem (CSP)

Definition 2.2: 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.

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:

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

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

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

2.2.1 What the course does on this problem

Technique	Where
Definition	Ch. 3 «Packing problems»
Formulations	Ch. 10 «Formulations of packing problems»
Exercises	Exercise 3.4 ; Exercise 10.6

Coloring problems

- Given an undirected graph $G = (V, E)$, a solution assigns a color to every vertex so that adjacent vertices receive different colors.
- Equivalently, a solution is a partition of V into stable sets; the minimum number of colors is the chromatic number $\chi(G)$.

Problem	Acronym	Difference
Vertex coloring problem	VCP	The central problem of the family.

3.1 The vertex coloring problem (VCP)

Definition 3.1: The 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.

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.

3.1.1 What the course does on this problem

Technique	Where
Definition	Ch. 4 «Coloring problems»
Formulations	Ch. 11 «Formulations of coloring problems»
Heuristics	Ch. 30 «Constructive heuristics»
Limits of approximation	Ch. 37 «Vertex coloring: limits of approximation»
Ad hoc algorithms	Ch. 42 «Vertex coloring: the DSATUR branch-and-bound»
Exercises	Ch. 4 (5 exercises) ; Ch. 11 (5 exercises) ; Ch. 30 (3 exercises) ; Ch. 37 (5 exercises) ; Ch. 42 (5 exercises)

Properties

- Strongly NP-hard: deciding whether $\chi(G) \leq 3$ is NP-complete ([Ch. 1](#)).
- Clique bound $\omega(G) \leq \chi(G)$ and stable set bound $\lceil n/\alpha(G) \rceil \leq \chi(G)$ ([Ch. 4](#)).
- The bounds can miss $\chi(G)$: on the Petersen graph $\omega(G) = 2 < \chi(G) = 3$ ([Ch. 4](#)).
- Brooks: $\chi(G) \leq \Delta$ for a connected graph that is neither complete nor an odd cycle ([Ch. 4](#)).

Formulations

- The model ILP_{VCP} has kn variables x_{vc} and k variables y_c , and $k!$ equivalent copies of every coloring ([Ch. 11](#)).
- Both LP relaxations have value 2 for every graph with an edge ([Ch. 11](#)).
- On the complete graph with n vertices the gap is $n - 2$ and the ratio $\frac{2}{n}$ tends to 0 ([Ch. 11](#)).

Heuristics

- The greedy coloring uses at most $\Delta(G) + 1$ colors, and the order of the vertices matters ([Ch. 30](#)).
- *DSATUR* colors first the vertex that sees the most colors; it is exact on bipartite graphs ([Ch. 30](#)).

Approximation

- Not approximable: no polynomial algorithm guarantees a ratio below $\frac{4}{3}$ unless $P = NP$, and not even $n^{1-\varepsilon}$ (Zuckerman) ([Ch. 37](#)).
- Special cases: bipartite and interval graphs exactly, planar graphs within 2 colors of the optimum ([Ch. 37](#)).

Ad hoc algorithms

- The DSATUR branch-and-bound branches on the most saturated vertex, with one child per feasible color and one new color, and prunes a node when $q(k) \geq \text{UB}$ ([Ch. 42](#)).
- On the Petersen graph it proves $\chi(G) = 3$ with 6 nodes ([Ch. 42](#)).

Clique problems

- Given an undirected graph $G = (V, E)$, a clique is a subset of pairwise adjacent vertices; the objective is to maximize its cardinality $\omega(G)$, or a weight of its vertices or edges.
- A stable set of G is a clique of the complement graph $\bar{G}$, so the maximum stable set problem belongs to the same family.

Problem	Acronym	Difference
Maximum clique problem	MCP	The central problem of the family.
Maximum stable set problem	MSSP	A subset is a clique of G if and only if it is a stable set of $\bar{G}$: $\omega(G) = \alpha(\bar{G})$ and $\alpha(G) = \omega(\bar{G})$.
Vertex-weighted maximum clique problem	VWMCP	With all the weights $w_v = 1$ it is the MCP.
Edge-weighted maximum clique problem	EWMCP	With $r_{uv} = 1$ it is again the MCP, since $r(K) = \binom{ K }{2}$ grows with $ K $.

4.1 The maximum clique problem (MCP)

Definition 4.1: The 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 $\omega(G)$.

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 $\omega(G) = n$.

4.1.1 What the course does on this problem

Technique	Where
Definition	Ch. 5 «Clique problems»
Formulations	Ch. 12 «Formulations of clique problems»
Limits of approximation	Ch. 38 «Maximum clique: limits of approximation»
Exercises	Ch. 4 (2 exercises); Ch. 5 (5 exercises); Ch. 12 (5 exercises); Ch. 38 (4 exercises)

Properties

- NP-hard: deciding whether a clique with k vertices exists is one of the 21 NP-complete problems of Karp (Ch. 5).
- Upper bounds: degree $\Delta + 1$, edges, degree count $k^{\max}$ and coloring $\chi(G)$ (Ch. 5).
- A first clique by hand: start from a vertex of maximum degree and add vertices adjacent to all those chosen (Ch. 5).

Formulations

- The model ILP_{MCP} has one binary variable per vertex and one constraint $x_u + x_v \leq 1$ per non-edge (Ch. 12).
- $x_v = \frac{1}{2}$ for all vertices is feasible, so $z(\text{LP}_{\text{MCP}}) \geq n/2$: the LP bound can be arbitrarily weak (Ch. 12).
- Clique inequalities on the stable sets of G strengthen the model, and the dual of their relaxation is a VCP relaxation: $\omega(G) \leq z(\text{LP}_{\text{MCP}}^{\text{CL}}) \leq \chi(G)$ (Ch. 12).

Approximation

- Not approximable: for every $\varepsilon > 0$ no polynomial algorithm guarantees a ratio within $n^{1-\varepsilon}$, unless $P = NP$ (Zuckerman) (Ch. 38).
- Special cases: planar graphs and graphs of bounded degree, solved exactly by enumeration (Ch. 38).

4.2 Maximum stable set problem (MSSP)

Definition 4.2: 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** $\alpha(G)$.

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 $\omega(G) = n$.

4.2.1 What the course does on this problem

Technique	Where
Definition	Ch. 5 «Clique problems»
Formulations	Ch. 12 «Formulations of clique problems»
Limits of approximation	Ch. 38 «Maximum clique: limits of approximation»
Exercises	Exercise 4.3; Ch. 5 (2 exercises); Exercise 12.6; Ch. 38 (2 exercises)

4.3 Vertex-weighted maximum clique problem (VWMCP)

Definition 4.3: Vertex-weighted maximum clique problem

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 $r(K)$.

4.3.1 What the course does on this problem

Technique	Where
Definition	Ch. 5 «Clique problems»
Formulations	Ch. 12 «Formulations of clique problems»
Exercises	Exercise 5.5; Exercise 12.7

4.4 Edge-weighted maximum clique problem (EWMCP)

Definition 4.4: Edge-weighted maximum clique problem

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 $r(K)$.

4.4.1 What the course does on this problem

Technique	Where
Definition	Ch. 5 «Clique problems»
Formulations	Ch. 12 «Formulations of clique problems»
Exercises	Exercise 12.8

Routing problems

- Given n cities and the distance d_{ij} between every ordered pair, a solution is a tour that visits every city exactly once and returns to the first one.
- The objective is to minimize the total length of the tour; the members differ in the properties of the distances, symmetric, metric or asymmetric.
- The Hamiltonian cycle problem is the decision problem on a graph: it asks whether a tour made of edges of the graph exists.

Problem	Acronym	Difference
Travelling salesman problem	TSP	The central problem of the family.
Metric travelling salesman problem	metric TSP	The special case of the TSP with the triangle inequality, where approximation guarantees exist.
Asymmetric travelling salesman problem	ATSP	The TSP is the special case of the ATSP in which $d_{ij} = d_{ji}$ for all i, j ; an ATSP instance has $(n - 1)!$ tours.
Hamiltonian cycle problem	HCP	With $d_{ij} = 1$ on the edges and a large distance elsewhere, G is Hamiltonian if and only if $z(I_G) = n$: the TSP contains it.

5.1 The travelling salesman problem (TSP)

Definition 5.1: The 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.

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:

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

5.1.1 What the course does on this problem

Technique	Where
Definition	Ch. 6 «Routing problems»
Formulations	Ch. 13 «Formulations of routing problems»
Heuristics	Ch. 29 «Heuristic algorithms: definitions»; Ch. 30 «Constructive heuristics»; Ch. 31 «Local search»; Ch. 32 «Metaheuristics»
MIP heuristics	Ch. 33 «MIP heuristics for the classical problems»
Approximation	Ch. 39 «The TSP: the general case and the metric case»
Exercises	Ch. 6 (7 exercises); Ch. 13 (5 exercises); Exercise 29.3; Ch. 30 (3 exercises); Ch. 31 (3 exercises); Ch. 32 (7 exercises); Ch. 39 (3 exercises)

Properties

- Strongly NP-hard: it contains the Hamiltonian cycle problem (Ch. 1).
- Lower bound $\lceil \frac{1}{2} \sum_i (a_i + b_i) \rceil$, with a_i, b_i the two shortest edges at city i (Ch. 6).
- Nearest neighbour from city 1 gives 25 on the five-city instance of Ch. 6, whose optimum is 18 (Ch. 6).

Formulations

- The MTZ model eliminates the subtours with $n - 1$ position variables u_i ; the DFJ model with $2^{n-1} - n$ subtour constraints, added only when violated (Ch. 13).
- The MTZ relaxation is weak; on the instance of Ch. 6 the bounds are 16 (assignment), 17 (MTZ) and 18 (DFJ) (Ch. 13).

Heuristics

- Multi-start nearest neighbour and the insertion heuristics NI, FI and CI all reach 18 on the instance of Ch. 6 (Ch. 30).
- 2-opt evaluates each of its $\frac{n(n-3)}{2}$ moves in $O(1)$ and turns the tour of length 25 into an optimal one in two moves (Ch. 31).
- Or-opt and VND escape the local optima of 2-opt (Ch. 31).

- Simulated annealing, tabu search, iterated local search, GRASP and LNS are compared on TSP20 (optimum 291) (Ch. 32).

MIP heuristics

- With the subtour constraints the LP optimum of the instance of Ch. 6 is integer: rounding gives 18, gap 0% (Ch. 33).

Approximation

- Metric TSP: the double tree algorithm is a 2-approximation, tight (Ch. 39).
- Christofides is a $\frac{3}{2}$ -approximation, tight (Ch. 39).
- General TSP: no constant ratio, unless $P = NP$ (Ch. 39).

5.2 Metric travelling salesman problem (metric TSP)

Definition 5.2: Metric travelling salesman problem

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.

5.2.1 What the course does on this problem

Technique	Where
Definition	Ch. 6 «Routing problems»
Formulations	Ch. 13 «Formulations of routing problems»
Heuristics	Ch. 30 «Constructive heuristics»
Approximation	Ch. 39 «The TSP: the general case and the metric case»
Exercises	Exercise 6.4; Ch. 39 (2 exercises)

5.3 Asymmetric travelling salesman problem (ATSP)

Definition 5.3: Asymmetric travelling salesman problem

In the **asymmetric travelling salesman problem** (ATSP) the distances are not required to be symmetric: d_{ij} may differ from d_{ji} . The definition of tour and of its length $L(\sigma)$ is unchanged, but a tour and the same tour travelled backwards are different solutions.

5.3.1 What the course does on this problem

Technique	Where
Definition	Ch. 6 «Routing problems»
Formulations	Ch. 13 «Formulations of routing problems»
Exercises	Exercise 6.5 ; Exercise 13.6

5.4 Hamiltonian cycle problem (HCP)

Definition 5.4: Hamiltonian cycle problem

The **Hamiltonian cycle problem** asks whether a graph $G = (V, E)$ has a cycle through all its vertices.

5.4.1 What the course does on this problem

Technique	Where
Definition	This page
Approximation	Ch. 39 «The TSP: the general case and the metric case»
Exercises	Exercise 39.3

Scheduling problems

- Given n jobs with processing times and m parallel machines, a solution assigns every job to one machine, where it is processed without interruption.
- The objective is to minimize the makespan $C_{\max}$, the completion time of the last job; in the three-field notation $\alpha \parallel C_{\max}$ the field α describes the machines: identical (P), a fixed number of identical machines (Pm), with different speeds (Q) or unrelated (R).

Problem	Acronym	Difference
Scheduling on identical parallel machines	$P \parallel C_{\max}$	The central problem of the family.
Scheduling on a fixed number of identical machines	$Pm \parallel C_{\max}$	Special case of $P \parallel C_{\max}$ with m fixed; with $m = 2$ it asks for a split of the jobs into two sets of loads as equal as possible.
Scheduling on parallel machines with different speeds	$Q \parallel C_{\max}$	Generalizes $P \parallel C_{\max}$, which is the case of machines with equal speeds.
Scheduling on unrelated parallel machines	$R \parallel C_{\max}$	Generalizes $Q \parallel C_{\max}$ and $P \parallel C_{\max}$.

6.1 Scheduling on identical parallel machines ($P \parallel C_{\max}$)

Definition 6.1: 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.

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:

$$n > m$$

6.1.1 What the course does on this problem

Technique	Where
Definition	Ch. 7 «Scheduling problems»
Formulations	Ch. 14 «Formulations of scheduling problems»
Heuristics	Ch. 29 «Heuristic algorithms: definitions»; Ch. 30 «Constructive heuristics»; Ch. 31 «Local search»; Ch. 32 «Metaheuristics»
MIP heuristics	Ch. 33 «MIP heuristics for the classical problems»
Approximation	Ch. 34 «Principles of approximation algorithms»; Ch. 40 «Approximation for $P C_{\max}$ »
Exercises	Ch. 7 (5 exercises); Ch. 14 (5 exercises); Ch. 29 (2 exercises); Ch. 30 (3 exercises); Ch. 31 (3 exercises); Ch. 32 (3 exercises); Exercise 33.5; Exercise 34.2; Ch. 40 (3 exercises)

Properties

- Strongly NP-hard when m is part of the input; weakly NP-hard for a fixed number of machines (Ch. 1).
- Lower bound $\max\{\max_j t_j, \lceil \sum_j t_j / m \rceil\}$; it is not always attained: $(4, 3, 3)$ on two machines gives 5, the optimum is 6 (Ch. 7).
- A makespan at most T exists if and only if the jobs fit into m bins of capacity T (Ch. 7).

Formulations

- The model $ILP_{P||C_{\max}}$ has nm binary variables x_{jh} and the continuous variable $C_{\max}$ (Ch. 14).
- The LP bound is $\sum_j t_j / m$; its ratio to the optimum is at least $1/m$, and $1/m$ is approached (Ch. 14).
- Valid inequalities strengthen the model; load ordering breaks the symmetry only when the loads are distinct (Ch. 14).

Heuristics

- On the instance of Ch. 7 (three machines, optimum 9) LIST with the shortest jobs first gives 12 and LPT gives 11 (Ch. 30).
- The LPT solution is a local optimum for move, while two swaps reach 9; the auxiliary objective $\sum_h L_h^2$ gives a slope to the plateaus (Ch. 31).

- Tabu search and VNS reach 9 on the same instance (Ch. 32).

MIP heuristics

- On the instance of Ch. 7 fractional diving ends at 10, guided diving from LPT finds 9, the feasibility pump returns 11 (Ch. 33).
- RINS, local branching and proximity search, started from LPT, reach 9; proximity search also proves it optimal (Ch. 33).

Approximation

- LIST is a $(2 - \frac{1}{m})$ -approximation for every list, tight (Ch. 40).
- LPT is a $(\frac{4}{3} - \frac{1}{3m})$ -approximation, tight on Graham's family; for $m = 3$ it is the instance of Ch. 7 (Ch. 40).

6.2 Scheduling on a fixed number of identical machines ($Pm \parallel C_{\max}$)

Definition 6.2: Scheduling on a fixed number of identical machines

The problem $P \parallel C_{\max}$ in which the number m of machines is a constant, for example $m = 2$, and only the jobs $\mathcal{J}$ and the processing times $t_j \in \mathbb{Z}_{\geq 1}$ form the input.

6.2.1 What the course does on this problem

Technique	Where
Definition	This page
Dynamic programming	Ch. 16 «Principles of dynamic programming»
Exercises	Exercise 16.3

6.3 Scheduling on parallel machines with different speeds ($Q \parallel C_{\max}$)

Definition 6.3: Scheduling on parallel machines with different speeds

In the three-field notation $\alpha | \beta | \gamma$, the machine environment $\alpha = Q$ denotes parallel machines with different speeds; $Q \parallel C_{\max}$ assigns jobs without additional characteristics to such machines, minimizing the makespan $C_{\max}$.

6.3.1 What the course does on this problem

Technique	Where
Definition	Ch. 7 «Scheduling problems»

6.4 Scheduling on unrelated parallel machines ($R||C_{\max}$)

Definition 6.4: Scheduling on unrelated parallel machines

In the three-field notation $\alpha|\beta|\gamma$, the machine environment $\alpha = R$ denotes unrelated parallel machines; $R||C_{\max}$ assigns jobs without additional characteristics to such machines, minimizing the makespan $C_{\max}$.

6.4.1 What the course does on this problem

Technique	Where
Definition	Ch. 7 «Scheduling problems»

Lot sizing problems

- Given a set $T = \{1, 2, \dots, m\}$ of periods with a demand d_t in every period $t \in T$, a solution fixes the periods with a setup and the quantity produced in each of them, so that every demand is satisfied.
- The objective is to minimize the total cost of setups, production and inventory holding; the members differ in the capacities, in the number of products and in the treatment of initial stock, overtime and unsatisfied demand.

Problem	Acronym	Difference
Capacitated lot sizing	CLS	The central problem of the family.
Uncapacitated lot sizing problem	ULS	Special case of the CLS in which the capacities are not binding, for example $cap_t \geq \sum_u d_u$.
Capacitated lot sizing with constant capacity	CLS-C	Special case of the CLS; the zero-inventory ordering property is replaced by a weaker one: between two periods with empty inventory at most one period produces a quantity strictly between 0 and C .
Lot sizing with several products and setup times	MLS	Generalizes the CLS, which is the case of one product with $vt_t = 1$ and no setup time.
Lot sizing with initial stock	LS-SI	Removes the assumption of an empty initial stock of the CLS.
Lot sizing with overtime	LS-OT	Replaces the fixed capacity cap_t of the CLS by two levels of capacity with different costs.
Lot sizing with unsatisfied demand	LS-UD	Relaxes the demand constraints of the CLS, with a penalty in the objective.

7.1 Capacitated lot sizing (CLS)

Definition 7.1: Capacitated lot sizing

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.

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.
 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.

7.1.1 What the course does on this problem

Technique	Where
Definition	Ch. 8 «Lot sizing problems»
Formulations	Ch. 15 «Formulations of lot sizing problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 26 «DP for the capacitated lot sizing problem»
MIP heuristics	Ch. 33 «MIP heuristics for the classical problems»
Exercises	Ch. 8 (5 exercises); Ch. 15 (5 exercises); Ch. 26 (2 exercises)

Properties

- NP-hard with general capacities; without capacities (ULS) it is polynomial (Ch. 1).
- Lower bound: the sum of the k smallest setup costs, with k the fewest periods whose capacities cover the total demand (Ch. 8).
- Lot-for-lot and merging by hand: $250 \rightarrow 210 \rightarrow 180$ on the instance of Ch. 8, where 180 is optimal (Ch. 8).

Formulations

- Big-M formulation: the strong value $M_t = \min\{cap_t, D_{t,m}\}$ raises the LP bound on the instance of Ch. 8 from 99.65 to 157.5 (Ch. 15).
- The shortest-path formulation gives the bound 170, the strongest of the chapter (Ch. 15).

Dynamic programming

- Without capacities an optimal plan produces only with empty stock (Wagner and Whitin): a DP over the last production run solves the ULS in $\Theta(m^2)$ steps (Ch. 25).
- With capacities the state is (period, inventory): $O(mDC)$ steps, pseudo-polynomial (Ch. 26).

MIP heuristics

- On the uncapacitated instance of the MIP course relax-and-fix finds the optimum 170, while the least unit cost heuristic gives 200 (Ch. 33).
- On a capacitated instance relax-and-fix stops at 170, and fix-and-optimize reaches the optimum 160 (Ch. 33).

7.2 Uncapacitated lot sizing problem (ULS)

Definition 7.2: Uncapacitated lot sizing problem

Given:

1. a set $T = \{1, 2, \dots, m\}$ of $m \in \mathbb{Z}_{\geq 1}$ time periods,
2. for each period $t \in T$, a demand $d_t \in \mathbb{Z}_{\geq 1}$,
3. for each period $t \in T$, a setup cost $sc_t \in \mathbb{Z}_{\geq 0}$, a variable production cost $vc_t \in \mathbb{Z}_{\geq 0}$ per unit, and a holding cost $hc_t \in \mathbb{Z}_{\geq 0}$ per unit in stock at the end of period t ,

the **Uncapacitated Lot-Sizing Problem** (ULS) asks to decide how many units to produce in each period, so that, starting with an empty inventory and ending with an empty inventory, the demand of every period is satisfied, and the total cost of setups, production, and holding is minimized.

Assumptions

- We make the following assumptions:
 1. There is a single product, and the stock is empty at the beginning and at the end of the horizon: the ULS is the CLS of a single product without capacities and with no initial stock.
 2. 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.

7.2.1 What the course does on this problem

Technique	Where
Definition	This page
Formulations	Ch. 15 «Formulations of lot sizing problems»
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 25 «DP for the uncapacitated lot sizing problem»
MIP heuristics	Ch. 33 «MIP heuristics for the classical problems»
Exercises	Exercise 8.4; Exercise 15.6; Ch. 25 (3 exercises)

7.3 Capacitated lot sizing with constant capacity (CLS-C)

Definition 7.3: Capacitated lot sizing with constant capacity

The CLS in which the capacity is the same in all periods, $cap_t = C$ for all $t \in T$.

7.3.1 What the course does on this problem

Technique	Where
Definition	This page
Dynamic programming	Ch. 26 «DP for the capacitated lot sizing problem»

7.4 Lot sizing with several products and setup times (MLS)

Definition 7.4: Lot sizing with several products and setup times

A set P of products shares the resource of each period: d_{pt} , sc_{pt} , vc_{pt} and hc_{pt} are the data of product $p \in P$ in period $t \in T$; producing a unit of p takes the time $vt_{pt} \in \mathbb{Z}_{\geq 1}$, a setup takes the time $st_{pt} \in \mathbb{Z}_{\geq 0}$, and the capacity cap_t is a time.

7.4.1 What the course does on this problem

Technique	Where
Definition	This page
Formulations	Ch. 15 «Formulations of lot sizing problems»
Exercises	Exercise 15.7

7.5 Lot sizing with initial stock (LS-SI)

Definition 7.5: Lot sizing with initial stock

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; the balance of the first period becomes $si_p + x_{p1} = d_{p1} + s_{p1}$.

7.5.1 What the course does on this problem

Technique	Where
Definition	Ch. 8 «Lot sizing problems»
Formulations	Ch. 15 «Formulations of lot sizing problems»

7.6 Lot sizing with overtime (LS-OT)

Definition 7.6: Lot sizing with overtime

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 .

7.6.1 What the course does on this problem

Technique	Where
Definition	Ch. 8 «Lot sizing problems»
Formulations	Ch. 15 «Formulations of lot sizing problems»
Exercises	Exercise 15.8

7.7 Lot sizing with unsatisfied demand (LS-UD)

Definition 7.7: Lot sizing with unsatisfied demand

A part of the demand may be left unsatisfied, at a penalty cu_{pt} per unit.

7.7.1 What the course does on this problem

Technique	Where
Definition	Ch. 8 «Lot sizing problems»
Formulations	Ch. 15 «Formulations of lot sizing problems»
Exercises	Exercise 15.9

Miscellaneous problems

- In the rod cutting problem a rod of length n is cut into pieces of integer length ℓ with profits p_ℓ ; the total length of the pieces equals n and their total profit is maximized.
- In the coin change problem an amount c is returned with coins of the denominations $w_1, \dots, w_n$, each in unlimited supply; the total value equals c and the number of coins is minimized.

Problem	Acronym	Difference
Rod cutting problem	RCP	An unbounded knapsack with one item type per piece length ℓ , weight ℓ , profit p_ℓ and capacity n , where the total length must be exactly n .
Coin change problem	CCP	Like the UKP it has unlimited copies of every item type, but it asks for exactly c and minimizes the number of coins, while the UKP asks for at most c and maximizes the profit.

8.1 Rod cutting problem (RCP)

Definition 8.1: Rod cutting problem

Given:

1. a rod of length $n \in \mathbb{Z}_{\geq 1}$,
2. a set $\{1, 2, \dots, n\}$ of n possible piece lengths,
3. for each piece length $\ell \in \{1, 2, \dots, n\}$, a profit $p_\ell \in \mathbb{Z}_{\geq 1}$,

the **Rod Cutting Problem** (RCP) asks to decide how many copies of each piece length to cut, so that the total length of the obtained pieces is equal to the rod length, and the total profit of the obtained pieces is maximized.

8.1.1 What the course does on this problem

Technique	Where
Definition	This page
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 28 «DP for the rod cutting problem»
Exercises	Ch. 28 (5 exercises)

8.2 Coin change problem (CCP)

Definition 8.2: Coin change problem

Given:

1. a vector $w \in \mathbb{Z}_{\geq 1}^n$ of n denominations, each available in unlimited supply,
2. an integer amount of change $c \in \mathbb{Z}_{\geq 0}$ to return,

the **Coin Change Problem** (CCP) asks to decide how many coins of each denomination to use so that their total value is exactly equal to c , and the total number of coins used is minimized. If no combination of coins returns exactly c , the instance is *infeasible*.

8.2.1 What the course does on this problem

Technique	Where
Definition	This page
Dynamic programming	Ch. 16 «Principles of dynamic programming»; Ch. 27 «DP for the coin change problem»
Exercises	Ch. 27 (5 exercises)