



SAPIENZA
UNIVERSITÀ DI ROMA

Ottimizzazione combinatoria

Volume 1 — Problemi di ottimizzazione combinatoria

I problemi di ottimizzazione combinatoria e le loro famiglie: problemi di zaino, di packing, di colorazione, di clique, di routing, di scheduling e di lot sizing. Per ogni problema: definizione, ipotesi, istanze, conteggio delle soluzioni, una prima soluzione a mano e un bound semplice.

Fabio Furini

fabiofurini.github.io

Dipartimento di Ingegneria informatica, automatica e gestionale — Sapienza Università di Roma

Versione online e materiale scaricabile:
fabiofurini.github.io/ottimizzazione-combinatoria/
github.com/fabiofurini/ottimizzazione-combinatoria

Anno accademico 2026–2027

Indice

1	Problemi di ottimizzazione combinatoria	4
1.1	Problemi di ottimizzazione combinatoria	5
1.2	Categorie di problemi di ottimizzazione combinatoria	6
1.3	Contare le soluzioni	10
1.4	Esplosione combinatoria e forza bruta	12
1.5	Bound primali, bound duali e gap	15
1.6	Cenni di complessità computazionale	17
1.7	Errori tipici	19
1.8	In sintesi	19
2	Problemi di zaino	21
2.1	Panoramica della famiglia	21
2.2	Il problema dello zaino 0/1	23
2.3	Il problema del subset sum	26
2.4	Il problema dello zaino limitato	28
2.5	Il problema dello zaino illimitato	31
2.6	Il problema dello zaino multidimensionale	33
2.7	Il problema degli zaini multipli	36
2.8	Il problema dello zaino a scelta multipla	39
2.9	Il problema dello zaino con setup	41
2.10	Osservazioni	44
2.11	Errori tipici	44
2.12	In sintesi	45
3	Problemi di packing	46
3.1	Panoramica della famiglia	47
3.2	Il problema del bin packing	47
3.3	Il problema del cutting stock	50
3.4	Osservazioni	53
3.5	Errori tipici	53
3.6	In sintesi	54

4	Problemi di colorazione	55
4.1	Panoramica della famiglia	56
4.2	Il problema della colorazione dei vertici	57
4.3	Osservazioni	61
4.4	Errori tipici	61
4.5	In sintesi	61
5	Problemi di clique	63
5.1	Panoramica della famiglia	64
5.2	Il problema della clique massima	64
5.3	Contare le soluzioni, una prima soluzione e i bound	67
5.4	Complessità	69
5.5	Varianti pesate	69
5.6	Osservazioni	71
5.7	Errori tipici	71
5.8	In sintesi	71
6	Problemi di routing	73
6.1	Panoramica della famiglia	74
6.2	Il problema del commesso viaggiatore	74
6.3	Contare i giri	76
6.4	Un primo giro a mano	77
6.5	Un lower bound combinatorio	78
6.6	Varianti: TSP metrico e TSP asimmetrico	79
6.7	Osservazioni	80
6.8	Errori tipici	81
6.9	In sintesi	81
7	Problemi di scheduling	82
7.1	Panoramica della famiglia	83
7.2	Scheduling su macchine parallele identiche	84
7.3	Contare le soluzioni	86
7.4	Un primo schedule a mano	86
7.5	Un lower bound combinatorio	87
7.6	Relazione con il problema del bin packing	88
7.7	Osservazioni	88
7.8	Errori tipici	88
7.9	In sintesi	89
8	Problemi di lot sizing	90
8.1	Panoramica della famiglia	90
8.2	Il problema del lot sizing capacitato	91
8.3	Contare le soluzioni	93

8.4	Un primo piano a mano	94
8.5	Un lower bound combinatorio	95
8.6	Varianti	95
8.7	Osservazioni	97
8.8	Errori tipici	97
8.9	In sintesi	98

Problemi di ottimizzazione combinatoria

Prerequisiti

- Dai *Preliminari*: problemi di ottimizzazione (regione ammissibile, funzione obiettivo, soluzione ottima e valore ottimo), coefficienti binomiali e partizioni di un insieme finito.
- Non serve alcuna conoscenza di programmazione intera: i bound di questo capitolo vengono da argomenti combinatori.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- riconoscere un problema di ottimizzazione combinatoria, la categoria delle sue decisioni e l'oggetto combinatorio delle sue soluzioni (un sottoinsieme, un ordinamento, una partizione, un assegnamento);
- contare le soluzioni candidate (2^n sottoinsiemi, $n!$ ordinamenti, $(n-1)!/2$ ordinamenti ciclici, numeri di Bell per le partizioni) e stimare il tempo di calcolo di un'enumerazione a forza bruta;
- definire bound primali, bound duali e gap, e usarli per certificare che una soluzione è ottima;
- collocare un problema nelle classi P, NP e NP-hard, e distinguere un algoritmo pseudo-polinomiale da uno polinomiale.

1.1 Problemi di ottimizzazione combinatoria

Definizione 1.1: problema di ottimizzazione combinatoria

Dati:

1. un insieme finito $\mathcal{U} = \{1, 2, \dots, n\}$, detto **insieme base**,
2. una famiglia $\mathcal{F} \subseteq 2^{\mathcal{U}}$ di sottoinsiemi di $\mathcal{U}$, detti **soluzioni ammissibili**,
3. una **funzione obiettivo** $f : \mathcal{F} \rightarrow \mathbb{R}$,

il **problema di ottimizzazione combinatoria** associato chiede di trovare una soluzione ammissibile $\tilde{S} \in \mathcal{F}$ tale che

$$f(\tilde{S}) \geq f(S), \quad \forall S \in \mathcal{F} \quad (\text{forma di massimo}),$$

oppure tale che

$$f(\tilde{S}) \leq f(S), \quad \forall S \in \mathcal{F} \quad (\text{forma di minimo}).$$

- La regione ammissibile $\mathcal{F}$ è un insieme **finito**, poiché $|2^{\mathcal{U}}| = 2^n$. Di conseguenza, una soluzione ottima esiste sempre quando $\mathcal{F} \neq \emptyset$.
- Nella maggior parte dei problemi la funzione obiettivo è **lineare** (additiva): ogni elemento $j \in \mathcal{U}$ ha un peso c_j e

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

Alcuni problemi hanno un obiettivo diverso, per esempio la più grande fra diverse somme, che è un massimo e non una somma.

- Un **problema** è una domanda generale con dati simbolici; un'**istanza** I del problema si ottiene fissando i numeri. Indichiamo con $z(I)$ il valore ottimo dell'istanza I .

Esempio 1.1: una piccola istanza

Consideriamo l'insieme base $\mathcal{U} = \{1, 2, 3, 4\}$, i pesi $\mathbf{c} = (7, 6, 4, 3)$ e l'obiettivo additivo $f(S) = \sum_{j \in S} c_j$, da massimizzare. Le soluzioni ammissibili sono gli 11 sottoinsiemi

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

Per esempio, $f(\{1, 3\}) = 7 + 4 = 11$ e $f(\{2, 3, 4\}) = 6 + 4 + 3 = 13$, mentre il sottoinsieme $\{1, 2\}$ non è ammissibile. Chiamiamo questa istanza [Esempio 1.1](#).

- Qui $\mathcal{F}$ è elencata. Nei problemi del corso non è mai elencata: è descritta da una regola, per esempio un vincolo di capacità, ed è troppo grande per essere scritta.

L'**ottimizzazione combinatoria** studia i problemi di ottimizzazione su **strutture discrete** e **oggetti combinatori**, come grafi, sottoinsiemi, permutazioni, sequenze, partizioni, coperture e impaccamenti.

- Le **caratteristiche principali** dei problemi di ottimizzazione combinatoria sono:
 1. l'insieme delle soluzioni ammissibili è **finito**;
 2. il numero di soluzioni ammissibili è tipicamente **enorme**: cresce in modo esponenziale (o più rapido) con la dimensione dell'istanza;
 3. trovare una soluzione ottima può essere **computazionalmente molto difficile** (Sezione 1.6);
 4. servono algoritmi specializzati per esplorare lo spazio delle soluzioni senza enumerarlo.

1.2 Categorie di problemi di ottimizzazione combinatoria

- Le decisioni di un problema di ottimizzazione combinatoria rientrano in cinque **categorie**. A ogni categoria sono associati un oggetto combinatorio, che è la forma di una soluzione, e un tipo naturale di variabili decisionali.

1.2.1 Categorie di decisioni

1. **IN/OUT (inclusione/esclusione)**. Ogni elemento è selezionato oppure no: una soluzione è un **sottoinsieme** degli elementi, descritto da variabili binarie $x_j \in \{0, 1\}$, con $x_j = 1$ se e solo se l'elemento j è selezionato.
 - Esempi: selezionare i progetti da finanziare con un budget limitato, scegliere i prodotti di un portafoglio, decidere quali clienti visitare in un percorso.
2. **Conteggio (quantità discrete)**. La decisione è quante unità di ogni elemento usare: una soluzione è un vettore di **quantità intere**, descritto da variabili intere $x_j \in \mathbb{Z}_{\geq 0}$.
 - Esempi: quante unità di un prodotto produrre, quanti operatori assegnare a un turno, quanti server attivare in un data center.
3. **Assegnamento**. Ogni elemento è assegnato a esattamente una fra diverse risorse distinte: una soluzione è una **funzione** dagli elementi alle risorse, descritta da variabili binarie $x_{ij} \in \{0, 1\}$, uguali a 1 se e solo se l'elemento j è assegnato alla risorsa i .
 - Esempi: assegnare compiti a macchine o persone, abbinare studenti a progetti, allocare veicoli ai percorsi di consegna.
4. **Ordinamento e sequenze**. La decisione è l'ordine in cui gli elementi sono lavorati o visitati: una soluzione è una **permutazione** $\sigma = (\sigma_1, \dots, \sigma_n)$ degli elementi, o un ordinamento ciclico per un giro.

- Esempi: l'ordine in cui sono lavorati i lavori, la sequenza delle operazioni su una macchina, l'ordine di visita dei clienti.

5. **Configurazione e partizionamento.** Gli elementi sono suddivisi in gruppi, ciascuno dei quali soddisfa una condizione come una capacità: una soluzione è una **partizione** degli elementi in gruppi intercambiabili.

- Esempi: suddividere un insieme di elementi in gruppi, impaccare oggetti in contenitori, tagliare materiali minimizzando lo sfrido.

Categoria	Oggetto combinatorio	Variabili naturali
IN/OUT	sottoinsieme	$x_j \in \{0, 1\}$
conteggio	vettore di quantità intere	$x_j \in \mathbb{Z}_{\geq 0}$
assegnamento	funzione (classi etichettate)	$x_{ij} \in \{0, 1\}$, una risorsa i per ogni j
ordinamento e sequenze	permutazione, ordinamento ciclico	$x_{ij} \in \{0, 1\}$, uguale a 1 se j segue i
configurazione e partizionamento	partizione (classi non etichettate)	$x_{ij} \in \{0, 1\}$, elemento j nel gruppo i

- La tabella riassume le cinque categorie, i loro oggetti combinatori e le loro variabili naturali.
- Con un insieme base opportuno ogni categoria rientra nella Definizione 1.1: per un assegnamento l'insieme base è l'insieme delle coppie (elemento, risorsa), per un ordinamento l'insieme delle coppie di elementi consecutivi.
- Assegnamento e partizionamento differiscono solo nelle risorse: risorse distinte danno classi etichettate, risorse identiche danno gruppi intercambiabili (Sezione 1.3).

La categoria delle decisioni determina l'oggetto combinatorio delle soluzioni, il numero di soluzioni candidate e le variabili decisionali naturali.

1.2.2 Aree applicative

- I problemi di ottimizzazione combinatoria nascono in molte aree applicative. Le principali, con le loro decisioni tipiche e alcuni esempi, sono:

Area	Decisioni tipiche	Esempi
pianificazione della produzione	che cosa, quando e quanto produrre, con quali risorse	lot sizing, pianificazione della capacità, bilanciamento delle linee di assemblaggio
scheduling	ordine e tempi delle operazioni sulle risorse	scheduling dei lavori sulle macchine, scheduling di progetti, scheduling della manutenzione
localizzazione di impianti	dove aprire gli impianti e quali clienti serve ciascuno	localizzazione di magazzini, di stabilimenti, di servizi di emergenza
packing	come sistemare oggetti in contenitori	selezione del carico, carico di più veicoli, carico dei pallet
taglio	come tagliare il materiale grezzo nei pezzi richiesti	taglio di barre e lastre, minimizzazione dello sfrido, taglio a ghigliottina
logistica e catena di fornitura	percorsi, flotte e livelli di scorta	instradamento di veicoli, gestione della flotta, gestione delle scorte
trasporti	equipaggi, veicoli e orari di un sistema di mobilità	turnazione degli equipaggi, instradamento degli aerei, orari ferroviari
gestione del personale	assegnamento di persone a compiti, turni e ruoli	pianificazione della forza lavoro, turni degli infermieri, pianificazione dei turni
progettazione di reti	quali nodi e archi di una rete attivare	reti di telecomunicazione, energetiche e di trasporto

- Un problema reale combina spesso diverse categorie. Per esempio, la localizzazione di impianti decide quali impianti aprire (IN/OUT) e assegna ogni cliente a un impianto aperto (assegnamento).

1.2.3 I problemi del corso per categoria

- La tabella colloca i problemi del corso nelle loro categorie; ogni nome rimanda alla sua pagina.

Problema	Soluzione	Categoria
problema dello zaino 0/1 (KP)	sottoinsieme di oggetti	IN/OUT
problema del subset sum (SSP)	sottoinsieme di oggetti	IN/OUT
problema dello zaino limitato (BKP)	numero di copie di ogni oggetto	conteggio
problema dello zaino illimitato (UKP)	numero di copie di ogni oggetto	conteggio
problema dello zaino multidimensionale (MDKP)	sottoinsieme di oggetti	IN/OUT
problema degli zaini multipli (MKP)	ogni oggetto in uno zaino o in nessuno	assegnamento
problema dello zaino a scelta multipla (MCKP)	un oggetto per classe	IN/OUT
problema dello zaino con setup (KPS)	sottoinsieme di classi e di oggetti	IN/OUT
problema del bin packing (BPP)	partizione degli oggetti nei contenitori	partizionamento
problema del cutting stock (CSP)	numero di rotoli tagliati con ogni schema	configurazione, conteggio
problema della colorazione dei vertici (VCP)	partizione dei vertici in insiemi stabili	partizionamento
problema della clique massima (MCP)	sottoinsieme di vertici	IN/OUT
problema dell'insieme stabile massimo (MSSP)	sottoinsieme di vertici	IN/OUT
problema del commesso viaggiatore (TSP)	ordinamento ciclico delle città	ordinamento
problema del commesso viaggiatore asimmetrico (ATSP)	ordinamento delle città su un giro	ordinamento
scheduling su macchine parallele identiche ($P C_{\max}$)	assegnamento dei lavori alle macchine	assegnamento
problema di lot sizing capacitato (CLS)	periodi di setup e quantità prodotte	IN/OUT, conteggio

1.3 Contare le soluzioni

- Prima di risolvere un problema ci chiediamo **quante** soluzioni candidate ha. La risposta dipende solo dall'oggetto combinatorio: sottoinsiemi, ordinamenti o partizioni.

1.3.1 Sottoinsiemi

Proposizione 1.2

Un insieme con n elementi ha 2^n sottoinsiemi.

Dimostrazione. Un sottoinsieme $S \subseteq \{1, 2, \dots, n\}$ è identificato dal vettore binario $\mathbf{x} \in \{0, 1\}^n$ con $x_j = 1$ se e solo se $j \in S$. Ognuna delle n componenti assume 2 valori in modo indipendente, quindi i vettori sono $2 \cdot 2 \cdots 2 = 2^n$. Equivalentemente, il numero di sottoinsiemi con k elementi è $\binom{n}{k}$, e abbiamo:

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

□

- Geometricamente, i 2^n vettori binari sono i vertici dell'ipercubo unitario $[0, 1]^n$: il quadrato per $n = 2$ (4 vertici) e il cubo per $n = 3$ (8 vertici).
- Ogni decisione IN/OUT su n elementi è un sottoinsieme: i 2^n sottoinsiemi sono le sue soluzioni candidate.

1.3.2 Ordinamenti e ordinamenti ciclici

- Gli n elementi di un insieme si possono ordinare in $n! = n \cdot (n - 1) \cdots 2 \cdot 1$ modi (**permutazioni**).
- Un **ordinamento ciclico** dispone gli n elementi lungo un cerchio, come in un giro attraverso n luoghi: l'elemento di partenza e il verso di percorrenza non contano.

Proposizione 1.3

Per $n \geq 3$ elementi, il numero di ordinamenti ciclici distinti, dove una sequenza, le sue rotazioni e la sua inversa sono lo stesso ordinamento ciclico, è

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

Dimostrazione. Poiché l'elemento di partenza non cambia un ordinamento ciclico, possiamo partire sempre dall'elemento 1: i restanti $n - 1$ elementi si possono ordinare in $(n - 1)!$ modi. Ogni ordinamento ciclico si ottiene due volte, una per ogni verso, per esempio $(1, 3, 5, 2, 4)$ e $(1, 4, 2, 5, 3)$. Quindi il numero di ordinamenti ciclici distinti è $(n - 1)!/2$. □

- Per $n = 4$ ci sono $3!/2 = 3$ ordinamenti ciclici: $(1, 2, 3, 4)$, $(1, 2, 4, 3)$ e $(1, 3, 2, 4)$. Ogni altra sequenza è una rotazione o un'inversa di uno di essi, per esempio $(2, 1, 4, 3)$ è l'inversa di $(1, 2, 3, 4)$ fatta partire dall'elemento 2.

1.3.3 Partizioni e numeri di Bell

- Una **partizione** di un insieme finito A è una collezione di sottoinsiemi non vuoti e a due a due disgiunti di A la cui unione è A . Le decisioni di raggruppamento, come la suddivisione di oggetti in gruppi, hanno partizioni come soluzioni.
- Il numero di partizioni di un insieme con n elementi è il **numero di Bell** $B(n)$, definito dalla ricorsione:

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

La ricorsione conta le partizioni in base al blocco che contiene l'elemento n : se questo blocco ha altri $n - 1 - k$ elementi, essi si possono scegliere in $\binom{n-1}{n-1-k} = \binom{n-1}{k}$ modi, e i restanti k elementi si partizionano in $B(k)$ modi.

- I primi numeri di Bell sono:

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

- Un **assegnamento** di n elementi a m etichette è una funzione dagli elementi alle etichette: ce ne sono m^n . Se le etichette sono intercambiabili (risorse identiche), due assegnamenti che differiscono solo per una ridenominazione delle etichette sono la stessa soluzione; le soluzioni distinte sono le partizioni degli elementi in al più m blocchi.

Esempio 1.2: quante soluzioni candidate

Per un insieme con $n = 5$ elementi, i numeri dei quattro oggetti combinatori sono:

oggetto combinatorio	numero
sottoinsiemi	$2^5 = 32$
ordinamenti (permutazioni)	$5! = 120$
ordinamenti ciclici	$4!/2 = 12$
partizioni	$B(5) = 52$
assegnamenti a 3 etichette	$3^5 = 243$
partizioni in al più 3 blocchi	$1 + 15 + 25 = 41$

L'ultima riga conta le partizioni di 5 elementi in 1, 2 o 3 blocchi (rispettivamente 1, 15 e 25); le altre 10 + 1 partizioni delle 52 hanno 4 o 5 blocchi.

- Questi numeri contano le soluzioni **candidate**. Quelle ammissibili possono essere molte meno: nell'Esempio 1.1 solo 11 dei 16 sottoinsiemi sono ammissibili.

1.4 Esplosione combinatoria e forza bruta

- Poiché la regione ammissibile è finita, un problema di ottimizzazione combinatoria si può sempre risolvere per **enumerazione completa**: elenchiamo tutte le soluzioni candidate, scartiamo quelle non ammissibili e teniamo la migliore. Questo algoritmo si chiama **forza bruta**.

Algoritmo

Input: un'istanza di un problema di ottimizzazione combinatoria in forma di massimo.

Output: una soluzione ottima e il valore ottimo.

Passo 0: **Inizializzazione.** Porre $\bar{S} \leftarrow$ nessuna e $LB \leftarrow -\infty$.

Passo 1: **Enumerazione.** Per ogni soluzione candidata S (sottoinsieme, ordinamento, partizione):

- (a) se S non è ammissibile, scartarla;
- (b) se S è ammissibile e $f(S) > LB$, porre $\bar{S} \leftarrow S$ e $LB \leftarrow f(S)$.

Passo 2: **Output.** Restituire $\tilde{S} = \bar{S}$ e $z(I) = LB$. STOP

Esempio 1.3: forza bruta sulla piccola istanza

Per l'istanza dell'Esempio 1.1 i $2^4 = 16$ sottoinsiemi di $\mathcal{U}$ sono:

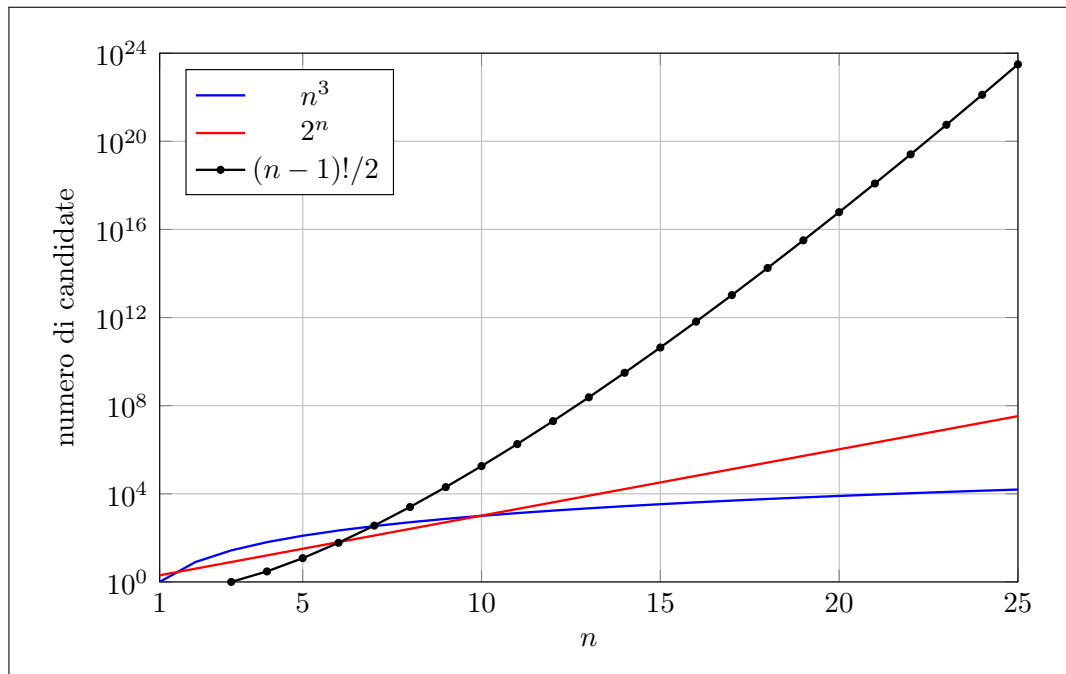
S	$f(S)$	ammissibile	S	$f(S)$	ammissibile
$\emptyset$	0	sì	$\{2, 3\}$	10	sì
$\{1\}$	7	sì	$\{2, 4\}$	9	sì
$\{2\}$	6	sì	$\{3, 4\}$	7	sì
$\{3\}$	4	sì	$\{1, 2, 3\}$	17	no
$\{4\}$	3	sì	$\{1, 2, 4\}$	16	no
$\{1, 2\}$	13	no	$\{1, 3, 4\}$	14	no
$\{1, 3\}$	11	sì	$\{2, 3, 4\}$	13	sì
$\{1, 4\}$	10	sì	$\{1, 2, 3,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

Fra gli 11 sottoinsiemi ammissibili, il migliore è $\tilde{S} = \{2, 3, 4\}$, con $z(I) = 13$. È l'unica soluzione ottima: anche il sottoinsieme $\{1, 2\}$ ha valore 13, ma non è ammissibile.

1.4.1 Crescita esponenziale

- La forza bruta funziona per le piccole istanze viste sopra, ma il numero di candidate cresce molto rapidamente con la dimensione n dell'istanza. La tabella seguente confronta la crescita polinomiale e quella esponenziale.

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 Tempi di calcolo

- Supponiamo che un calcolatore valuti 10^9 soluzioni candidate al secondo, e calcoliamo il tempo di un'enumerazione completa (un anno = 365.25 giorni):

n	2^n	tempo a 10^9 al secondo	tempo a 10^{12} al secondo
10	1 024	≈ 1.02 microsecondi	≈ 1.02 nanosecondi
20	$\approx 1.05 \cdot 10^6$	≈ 1.05 millisecondi	≈ 1.05 microsecondi
30	$\approx 1.07 \cdot 10^9$	≈ 1.07 secondi	≈ 1.07 millisecondi
40	$\approx 1.10 \cdot 10^{12}$	≈ 18.3 minuti	≈ 1.10 secondi
50	$\approx 1.13 \cdot 10^{15}$	≈ 13.0 giorni	≈ 18.8 minuti
60	$\approx 1.15 \cdot 10^{18}$	≈ 36.5 anni	≈ 13.3 giorni
70	$\approx 1.18 \cdot 10^{21}$	$\approx 3.74 \cdot 10^4$ anni	≈ 37.4 anni
80	$\approx 1.21 \cdot 10^{24}$	$\approx 3.83 \cdot 10^7$ anni	$\approx 3.83 \cdot 10^4$ anni

n	numero	ordinamenti ciclici: $(n-1)!/2$		partizioni: $B(n)$	
		10^9 al secondo	10^{12} al secondo	numero	10^9 al secondo
10	181 440	≈ 181 microsecondi	≈ 181 nanosecondi	115 975	≈ 116 microsecondi
15	$\approx 4.36 \cdot 10^{10}$	≈ 43.6 secondi	≈ 43.6 millisecondi	$\approx 1.38 \cdot 10^9$	≈ 1.38 secondi
20	$\approx 6.08 \cdot 10^{16}$	≈ 1.93 anni	≈ 16.9 ore	$\approx 5.17 \cdot 10^{13}$	≈ 14.4 ore
25	$\approx 3.10 \cdot 10^{23}$	$\approx 9.83 \cdot 10^6$ anni	$\approx 9.83 \cdot 10^3$ anni	$\approx 4.64 \cdot 10^{18}$	≈ 147 anni

- Un calcolatore 1000 volte più veloce non cambia il quadro: poiché $2^{10} = 1024 > 1000$, nello stesso tempo enumera i sottoinsiemi di soli 10 elementi in più, circa. Per gli ordinamenti ciclici il guadagno è ancora minore: passare da n a $n+1$ elementi moltiplica il loro numero per n , e $15 \cdot 16 \cdot 17 = 4080 > 1000$, quindi da 15 elementi in su il calcolatore più veloce guadagna meno di 3 elementi.

L'enumerazione completa è possibile solo per **istanze molto piccole**. Calcolatori più veloci non aiutano: un numero esponenziale di candidate vanifica qualunque accelerazione costante. Servono approcci più intelligenti, che **escludono** intere parti dello spazio delle soluzioni senza esplorarle.

1.5 Bound primali, bound duali e gap

- Gli approcci più intelligenti si basano su un'idea semplice: invece di calcolare direttamente $z(I)$, lo **racchiudiamo** fra due numeri. Uno viene da una soluzione ammissibile, l'altro da un argomento che vale per *tutte* le soluzioni ammissibili.

Definizione 1.4: bound primali e duali

Consideriamo un'istanza I di un problema di ottimizzazione combinatoria con soluzioni ammissibili $\mathcal{F}$, funzione obiettivo f e valore ottimo $z(I)$.

1. Un **bound primale** è il valore $f(\bar{S})$ di una soluzione ammissibile $\bar{S} \in \mathcal{F}$.
2. Un **bound duale** è un numero β tale che

$$f(S) \leq \beta, \quad \forall S \in \mathcal{F} \quad (\text{forma di massimo}),$$

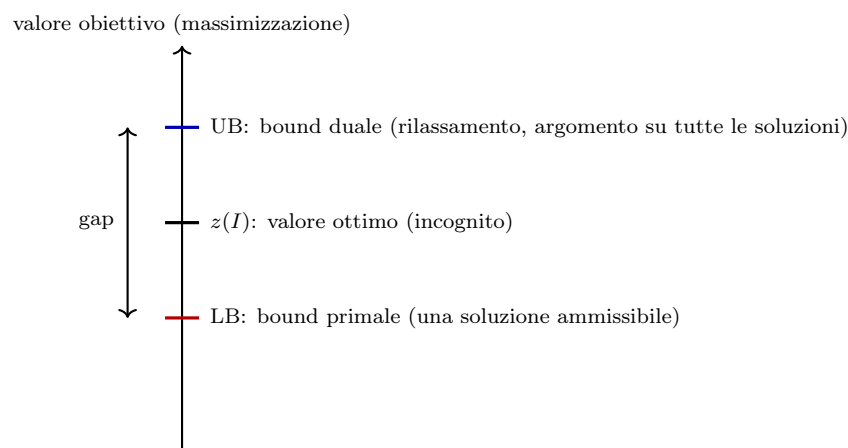
oppure tale che

$$f(S) \geq \beta, \quad \forall S \in \mathcal{F} \quad (\text{forma di minimo}).$$

- Nella **forma di massimo** un bound primale è un **lower bound** e un bound duale è un **upper bound**:

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

- Nella **forma di minimo** i ruoli si scambiano: un bound primale è un **upper bound** $\text{UB} = f(\bar{S})$ e un bound duale è un **lower bound** LB , con $\text{LB} \leq z(I) \leq \text{UB}$.
- I bound primali vengono da soluzioni ammissibili, per esempio trovate da **algoritmi euristici**; i bound duali vengono da **rilassamenti** (per esempio il rilassamento lineare) o da argomenti combinatori.



Definizione 1.5: gap

Siano $\text{LB} \leq z(I) \leq \text{UB}$ un lower bound e un upper bound di un'istanza I , uno primale e uno duale. Il **gap assoluto** è $\text{UB} - \text{LB}$ e, per $\text{LB} \neq 0$, il **gap relativo** (o semplicemente

gap) è

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

- Il denominatore è sempre il **lower bound**: nella forma di massimo è il valore della soluzione ammissibile, come nel gap di ottimalità del modulo *Programmazione lineare intera*, [Capitolo 2 «Bound primali e duali»](#); nella forma di minimo è il bound duale.
- Con questa scelta il gap non è mai minore dell'errore relativo $|f(\bar{S}) - z(I)|/z(I)$ della soluzione ammissibile $\bar{S}$: un gap del 5% garantisce una soluzione entro il 5% dall'ottimo.
- Se il gap è nullo, la soluzione ammissibile $\bar{S}$ è ottima e il bound duale è un **certificato** della sua ottimalità.

Osservazione 1.6

Se $f(S) \in \mathbb{Z}$ per ogni $S \in \mathcal{F}$, allora nella forma di massimo anche $\lfloor \beta \rfloor$ è un bound duale, e nella forma di minimo anche $\lceil \beta \rceil$ è un bound duale.

Dimostrazione. Nella forma di massimo, per ogni $S \in \mathcal{F}$ abbiamo $f(S) \leq \beta$ e $f(S) \in \mathbb{Z}$; il più grande intero non maggiore di β è $\lfloor \beta \rfloor$, quindi $f(S) \leq \lfloor \beta \rfloor$. La forma di minimo è analoga. $\square$

Esempio 1.4: bound sulla piccola istanza

Consideriamo di nuovo l'istanza dell'[Esempio 1.1](#) (massimizzazione), e supponiamo di non avere enumerato i suoi sottoinsiemi.

- *Bound primale.* Partiamo da $\emptyset$ ed esaminiamo gli elementi per peso non crescente, aggiungendo ciascuno se l'insieme resta ammissibile: $\{1\}$ è ammissibile, $\{1, 2\}$ no, $\{1, 3\}$ è ammissibile, $\{1, 3, 4\}$ no. La soluzione ammissibile $\bar{S} = \{1, 3\}$ dà $\text{LB} = 11$.
- *Un primo bound duale.* I pesi sono positivi, quindi $f(S) \leq f(\mathcal{U}) = 20$ per ogni $S \in \mathcal{F}$.
- *Un bound duale migliore.* Nessun sottoinsieme ammissibile contiene sia 1 sia 2, quindi un sottoinsieme ammissibile prende al più uno dei pesi 7 e 6:

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

Il gap assoluto è $14 - 11 = 3$ e il gap relativo è $\frac{3}{11} \approx 27.3\%$.

- *Un bound primale migliore.* La soluzione ammissibile $\{2, 3, 4\}$ dà $\text{LB} = 13$: il gap scende a $\frac{1}{13} \approx 7.7\%$.
- *Chiusura del gap.* L'unico sottoinsieme ammissibile con 3 elementi è $\{2, 3, 4\}$, e un sottoinsieme con al più 2 elementi ha valore al più $7 + 6 = 13$. Quindi $\text{UB} = 13 = \text{LB}$: il gap è nullo, e questo argomento è un certificato che $\{2, 3, 4\}$ è ottima, senza elencare i 16 sottoinsiemi.

Se invece un argomento desse il bound duale 13.5, anche $\lfloor 13.5 \rfloor = 13$ sarebbe un bound duale, poiché tutti i valori $f(S)$ sono interi (Osservazione 1.6).

- Nel resto del corso, ogni capitolo su un problema registra, per le sue istanze, i migliori bound primali e duali trovati fino a quel punto; ogni metodo migliora l'uno o l'altro lato.

1.6 Cenni di complessità computazionale

- La forza bruta richiede tempo esponenziale. Esiste un algoritmo che risolve un problema di ottimizzazione combinatoria in tempo polinomiale? La complessità computazionale classifica i problemi in base a questa domanda; qui diamo solo le definizioni che servono nel corso.
- La **dimensione** di un'istanza è il numero di bit necessari per scriverla. Per esempio, un'istanza con n numeri, il più grande dei quali è M , ha dimensione dell'ordine di $n \log_2 M$: un numero M si scrive con circa $\log_2 M$ bit, non con M simboli.
- Un algoritmo è **polinomiale** se il suo tempo di esecuzione è limitato da un polinomio nella dimensione dell'istanza.
- La **versione decisionale** di un problema di ottimizzazione pone una domanda sì/no, per esempio: “esiste una soluzione ammissibile di valore almeno k ?”.

Definizione 1.7: le classi P e NP

1. **P** è la classe dei problemi decisionali che si possono risolvere con un algoritmo polinomiale.
2. **NP** è la classe dei problemi decisionali le cui istanze “sì” ammettono un certificato che si può verificare in tempo polinomiale (per esempio, una soluzione ammissibile di cui ammissibilità e valore si calcolano in tempo polinomiale).
3. Un problema decisionale in NP è **NP-completo** se ogni problema in NP si può ridurre a esso in tempo polinomiale. Un problema di ottimizzazione è **NP-hard** se la sua versione decisionale è NP-completa (più in generale, se un algoritmo polinomiale per esso darebbe un algoritmo polinomiale per un problema NP-completo).

- Non è noto se $P = NP$. Non si conosce alcun algoritmo polinomiale per un problema NP-completo, ed è opinione diffusa che non ne esista nessuno: per un problema NP-hard non ci aspettiamo un algoritmo esatto polinomiale.

1.6.1 Algoritmi pseudo-polinomiali e problemi NP-hard in senso forte

- Alcuni problemi NP-hard si possono risolvere in tempo polinomiale in n e **nel valore** dei numeri dell'istanza. Per esempio, un algoritmo che richiede tempo $O(nM)$, dove M è il numero più grande dell'istanza, non è polinomiale: M è esponenziale nel suo numero di bit $\log_2 M$.

Definizione 1.8: algoritmo pseudo-polinomiale, problemi NP-hard in senso debole e forte

1. Un algoritmo è **pseudo-polinomiale** se il suo tempo di esecuzione è limitato da un polinomio nella dimensione dell'istanza e nel numero più grande che vi compare.
2. Un problema NP-hard è **NP-hard in senso forte** se resta NP-hard quando tutti i suoi numeri sono limitati da un polinomio nella dimensione dell'istanza (equivalentemente, quando i numeri sono scritti in unario). Altrimenti, se ammette un algoritmo pseudo-polinomiale, si dice **NP-hard in senso debole** (NP-hard in senso ordinario).

- Un problema NP-hard in senso forte non ha algoritmi pseudo-polinomiali, a meno che $P = NP$.

Che un problema sia NP-hard non significa che un'istanza non si possa risolvere: significa che non ci aspettiamo un algoritmo che risolva *ogni* istanza in tempo polinomiale. In pratica combiniamo **metodi esatti** che chiudono il gap fra bound primali e duali, **euristiche** che trovano rapidamente buone soluzioni, e **algoritmi di approssimazione** che garantiscono un gap limitato nel caso peggiore.

- I problemi dei prossimi capitoli sono tutti NP-hard: il problema dello zaino 0/1 è NP-hard in senso debole, mentre il bin packing, la colorazione dei vertici, la clique massima e il problema del commesso viaggiatore sono NP-hard in senso forte. Ogni capitolo discute la complessità del suo problema.

1.7 Errori tipici

Attenzione

- **Confondere soluzioni candidate e ammissibili.** I numeri 2^n , $(n-1)!/2$ e $B(n)$ contano le candidate; le soluzioni ammissibili possono essere molte meno, per esempio 11 sottoinsiemi su 16 nell'Esempio 1.1.
- **Scambiare i due versi dei bound.** In un problema di massimo un bound primale è un lower bound e un bound duale è un upper bound; in un problema di minimo i ruoli si scambiano.
- **Arrotondare un bound duale quando l'obiettivo non è intero.** L'arrotondamento dell'Osservazione 1.6 richiede $f(S) \in \mathbb{Z}$ per ogni soluzione ammissibile.
- **Contare due volte soluzioni equivalenti.** Un ordinamento ciclico e il suo inverso sono lo stesso ordinamento ciclico, e due assegnamenti che differiscono per una ridenominazione di etichette intercambiabili sono la stessa soluzione.
- **Leggere NP-hard come irrisolvibile.** Essere NP-hard è un'affermazione sul caso peggiore: molte istanze di dimensione pratica si risolvono, e il resto del corso mostra come.

1.8 In sintesi

In sintesi

- Un problema di ottimizzazione combinatoria ha un insieme base finito, una famiglia di sottoinsiemi ammissibili e una funzione obiettivo; una soluzione ottima esiste quando esiste una soluzione ammissibile.
- La categoria delle decisioni dà l'oggetto combinatorio di una soluzione: un sottoinsieme, un vettore di quantità intere, un assegnamento, un ordinamento o una partizione.
- Le soluzioni candidate sono 2^n sottoinsiemi, $n!$ ordinamenti, $(n-1)!/2$ ordinamenti ciclici e $B(n)$ partizioni: la forza bruta funziona solo su istanze molto piccole.
- Una soluzione ammissibile dà un bound primale, un rilassamento o un argomento combinatorio dà un bound duale; un gap nullo certifica che la soluzione ammissibile è ottima.
- I problemi del corso sono NP-hard: per essi non ci aspettiamo un algoritmo esatto polinomiale.

Quadro riassuntivo della piccola istanza.

Esempio 1.1 (max)	lower bound	upper bound	da
primi bound	11	14	scelta greedy, uno dei pesi 7 e 6
bound finali	13	13	soluzione $\{2, 3, 4\}$, argomento sulla cardinalità

Problemi di zaino

Prerequisiti

- Da questo modulo, [Capitolo 1 «Problemi di ottimizzazione combinatoria»](#): istanze, soluzioni ammissibili, bound primali e duali e gap.
- Dai *Preliminari*: sottoinsiemi di un insieme finito e il numero 2^n di sottoinsiemi.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- enunciare la definizione del problema dello zaino 0/1 e delle sue varianti, con istanze e soluzioni ammissibili;
- spiegare le ipotesi sui dati e perché si possono fare senza perdita di generalità;
- contare le soluzioni candidate e le soluzioni ammissibili di una piccola istanza;
- trovare a mano una soluzione ammissibile e calcolare un semplice bound duale combinatorio;
- distinguere le varianti: copie degli oggetti, più risorse, più zaini, classi di oggetti.

2.1 Panoramica della famiglia

- I **problemi di zaino** hanno gli stessi dati: un insieme $\mathcal{J} = \{1, 2, \dots, n\}$ di n oggetti, per ogni oggetto $j \in \mathcal{J}$ un *peso* $w_j \in \mathbb{Z}_{\geq 1}$ e un *profitto* $p_j \in \mathbb{Z}_{\geq 1}$, e uno o più zaini con una *capacità* in $\mathbb{Z}_{\geq 1}$.
- Una soluzione decide quali oggetti selezionare, o quante copie di ogni oggetto: il peso totale degli oggetti selezionati non deve superare la capacità dello zaino che li riceve.
- L'obiettivo è massimizzare il profitto totale degli oggetti selezionati.

Un problema di zaino seleziona oggetti di profitto totale massimo il cui peso totale rispetta le capacità. I membri della famiglia differiscono nel numero di copie degli oggetti, nel numero di capacità e di zaini, e nelle classi in cui gli oggetti sono raggruppati.

- La tabella elenca i membri della famiglia, con la sezione che definisce ciascuno di essi.

Problema	Sigla	Differenza rispetto al KP	Sezione
problema dello zaino 0/1	KP	una copia di ogni oggetto, uno zaino	2.2
problema del subset sum	SSP	$p_j = w_j$, peso totale esattamente uguale a c	2.3
problema dello zaino limitato	BKP	al più b_j copie dell'oggetto j	2.4
problema dello zaino illimitato	UKP	un numero illimitato di copie di ogni oggetto	2.5
problema dello zaino multidimensionale	MDKP	k capacità c_d e pesi w_{dj}	2.6
problema degli zaini multipli	MKP	m zaini con capacità c_k	2.7
problema dello zaino a scelta multipla	MCKP	classi $\mathcal{J}_h$, esattamente un oggetto per classe	2.8
problema dello zaino con setup	KPS	classi $\mathcal{J}_h$ con costi di setup f_h e pesi di setup s_h	2.9

2.1.1 Le varianti in breve

- Ogni problema della famiglia è illustrato su una piccola istanza, nella sua sezione della tabella precedente e con la notazione di quella sezione.
- La tabella riporta, per queste istanze, il numero di soluzioni candidate, il numero di soluzioni ammissibili e il valore ottimo, calcolati per enumerazione.
- Nel BKP e nell'UKP una soluzione candidata è il vettore dei numeri di copie di ogni oggetto, così che copie identiche non sono contate due volte.
- La soluzione ottima non è sempre unica: l'MKP, l'MCKP e il KPS hanno più soluzioni ottime con lo stesso valore, e nel SSP tre sottoinsiemi raggiungono il valore obiettivo.

Variante	Soluzioni candidate	Numero	Ammissibili	Ottimo	Soluzioni ottime
KP	sottoinsiemi di 6 oggetti	64	13	15	$\{1, 4\}$
SSP	sottoinsiemi di 6 oggetti	64	3	sì	$\{6\}, \{1, 4\}, \{2, 3\}$
BKP	copie $x_j \leq b_j$	144	18	18	due copie dell'oggetto 1 e l'oggetto 5
UKP	copie $x_j \leq \hat{b}_j$	320	22	24	quattro copie dell'oggetto 1
MDKP	sottoinsiemi di 4 oggetti	16	9	14	$\{3, 4\}$
MKP	ogni oggetto in uno zaino o in nessuno	243	109	30	$(S_1, S_2) = (\{1, 2\}, \{4, 5\}), (\{2, 4\}, \{1, 5\}), (\{4, 5\}, \{1, 2\})$
MCKP	un oggetto per classe	8	3	17	$\{1, 4, 5\}, \{2, 3, 5\}$
KPS	sottoinsiemi di 6 oggetti	64	40	16	$\{1, 2, 3\}, \{4, 5, 6\}, \{1, 2, 3, 6\}$

- Nell'MCKP solo 3 delle 8 scelte di un oggetto per classe entrano nello zaino: la condizione di ammissibilità fra le ipotesi della Sezione 2.8 non è una formalità.

2.2 Il problema dello zaino 0/1

Definizione 2.1: problema dello zaino 0/1

Dati:

1. uno zaino di *capacità* $c \in \mathbb{Z}_{\geq 1}$,
2. un insieme $\mathcal{I} = \{1, 2, \dots, n\}$ di n oggetti,
3. per ogni oggetto $j \in \mathcal{I}$, un *profitto* $p_j \in \mathbb{Z}_{\geq 1}$ e un *peso* $w_j \in \mathbb{Z}_{\geq 1}$,

il **problema dello zaino 0/1** (KP) chiede di decidere quali oggetti inserire nello zaino, in modo che il peso totale degli oggetti selezionati non superi la capacità dello zaino e il profitto totale degli oggetti selezionati sia massimo.

- Un'istanza del KP è specificata da:

1. due numeri interi:

il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,

la capacità dello zaino $c \in \mathbb{Z}_{\geq 1}$,

2. due vettori di numeri interi di lunghezza n :

$\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, che contiene i profitti degli oggetti,

$\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, che contiene i pesi degli oggetti.

- Una **soluzione ammissibile del KP** è un sottoinsieme di oggetti

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

tale che il peso totale degli oggetti selezionati non supera la capacità dello zaino, ossia

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

Di conseguenza, il KP si può enunciare in modo equivalente come il problema di selezionare un sottoinsieme di oggetti il cui peso totale rispetta la capacità dello zaino e il cui profitto totale è massimo.

Data un'istanza del KP, il KP si può enunciare formalmente come:

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

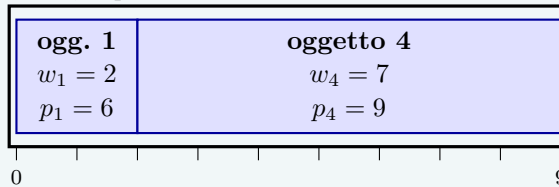
Esempio 2.1: istanza del KP

Consideriamo uno zaino di capacità $c = 9$ e un insieme $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ di 6 oggetti ($n = 6$), con i seguenti profitti e pesi:

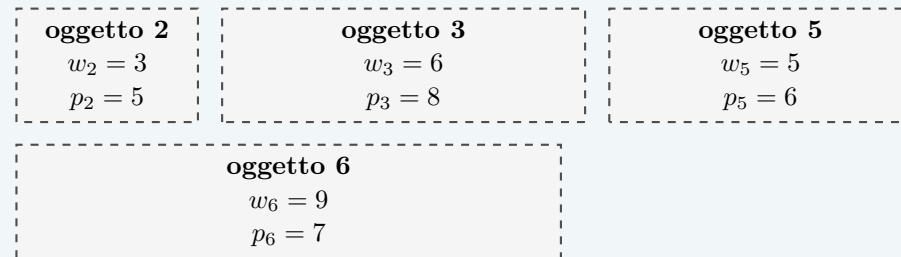
Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Profitti	p_j	6	5	8	9	6	7
Pesi	w_j	2	3	6	7	5	9

Nel disegno, lo zaino è una scatola di lunghezza $c = 9$ e ogni oggetto è un blocco di lunghezza w_j . Gli oggetti 1 e 4 riempiono lo zaino, con peso 9 e profitto 15; gli altri oggetti restano fuori (tratteggiati).

zaino, capacità $c = 9$



non selezionati



Chiamiamo questa istanza del KP [Esempio 2.1](#): è l'istanza di riferimento del capitolo.

2.2.1 Ipotesi

- Facciamo le seguenti ipotesi:
 - Il peso totale di tutti gli oggetti supera la capacità dello zaino (altrimenti si potrebbero selezionare tutti gli oggetti). Formalmente:

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

- Ogni oggetto, da solo, entra nello zaino (altrimenti non può essere selezionato). Formalmente:

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

2.2.2 Contare le soluzioni, una prima soluzione e un bound semplice

- **Soluzioni candidate.** Ogni sottoinsieme di $\mathcal{I}$ è una soluzione candidata, quindi ce ne sono 2^n (Capitolo 1 «Problemi di ottimizzazione combinatoria»). Per l'Esempio 2.1 abbiamo $2^6 = 64$.
- **Soluzioni ammissibili.** I tre oggetti più leggeri pesano $2+3+5 = 10 > 9$, quindi nessun sottoinsieme con tre o più oggetti è ammissibile. Le soluzioni ammissibili sono l'insieme vuoto, i 6 singoletti e le 6 coppie della tabella seguente: 13 soluzioni su 64.

coppia S	$\{1, 2\}$	$\{1, 3\}$	$\{1, 4\}$	$\{1, 5\}$	$\{2, 3\}$	$\{2, 5\}$
peso	5	8	9	7	9	8
profitto	11	14	15	12	13	11

- **Una prima soluzione, a occhio.** L'oggetto 4 ha il profitto massimo (9, peso 7). La capacità residua $9 - 7 = 2$ basta per l'oggetto 1 (peso 2, profitto 6). Il sottoinsieme $\{1, 4\}$ ha peso 9 e profitto 15: è un **bound primale**, $LB = 15 \leq z(I)$.

Osservazione 2.2: bound di cardinalità

Sia $\bar{k}$ il più grande intero tale che i $\bar{k}$ oggetti più leggeri pesano al più c . Ogni sottoinsieme ammissibile ha al più $\bar{k}$ oggetti, quindi $z(I)$ è al più la somma dei $\bar{k}$ profitti più grandi.

Dimostrazione. Un sottoinsieme con k oggetti pesa almeno quanto i k oggetti più leggeri. Se $k > \bar{k}$, i k oggetti più leggeri pesano più di c , quindi il sottoinsieme non è ammissibile. Il profitto di un sottoinsieme con al più $\bar{k}$ oggetti è al più la somma dei $\bar{k}$ profitti più grandi. $\square$

- **Un primo bound duale.** Nell'Esempio 2.1 abbiamo $\bar{k} = 2$, poiché $2 + 3 = 5 \leq 9$ e $2 + 3 + 5 = 10 > 9$. I due profitti più grandi sono 9 e 8, quindi $UB = 17$. La somma di tutti i profitti, 41, è un bound duale molto più debole.
- **Il gap.** Abbiamo $15 \leq z(I) \leq 17$: il gap assoluto è 2 e il gap relativo è $\frac{2}{15}$. Elencando i 13 sottoinsiemi ammissibili si vede che $\{1, 4\}$ è l'unica soluzione ottima e $z(I) = 15$; per istanze più grandi servono bound più forti, che vengono dai rilassamenti (modulo Programmazione lineare intera, Capitolo 2 «Bound primali e duali»).

2.3 Il problema del subset sum

Definizione 2.3: problema del subset sum

Dati:

1. un valore obiettivo $c \in \mathbb{Z}_{\geq 1}$,
2. un insieme $\mathcal{I} = \{1, 2, \dots, n\}$ di n oggetti,
3. per ogni oggetto $j \in \mathcal{I}$, un *peso* $w_j \in \mathbb{Z}_{\geq 1}$,

il **problema del subset sum** (SSP) chiede se esiste un sottoinsieme di oggetti il cui peso totale è esattamente uguale al valore obiettivo c .

- Un'istanza del SSP è specificata da:

1. due numeri interi:

il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,

il valore obiettivo $c \in \mathbb{Z}_{\geq 1}$,

2. un vettore di numeri interi di lunghezza n :

$\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, che contiene i pesi degli oggetti.

- Una **soluzione ammissibile del SSP** è un sottoinsieme di oggetti

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

tale che il peso totale degli oggetti selezionati è esattamente uguale al valore obiettivo, ossia

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

Di conseguenza, il SSP si può enunciare in modo equivalente come il problema di selezionare un sottoinsieme di oggetti il cui peso totale coincide esattamente con il valore obiettivo.

Data un'istanza del SSP, il SSP si può enunciare formalmente come:

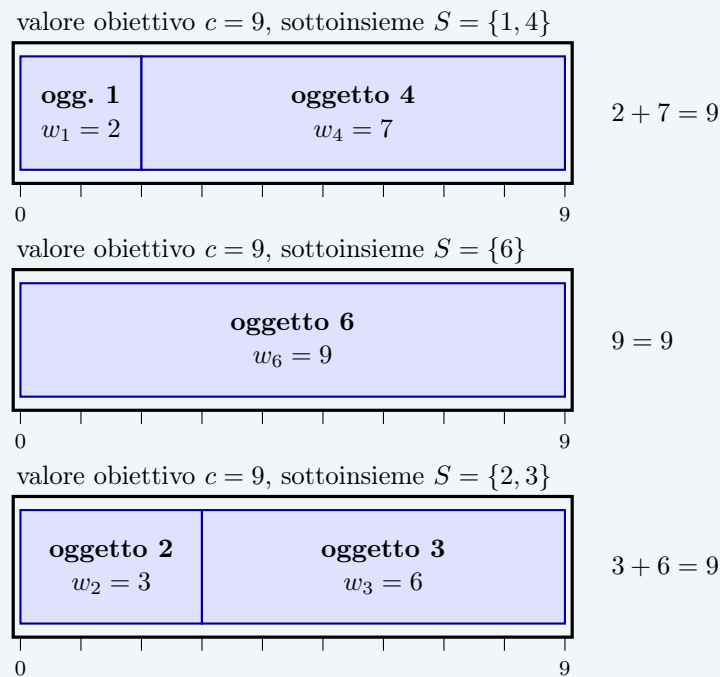
$$\text{trovare } S \subseteq \mathcal{I} \quad \text{tale che} \quad \sum_{j \in S} w_j = c. \quad (2.2)$$

Esempio 2.2: istanza del SSP

Consideriamo un valore obiettivo $c = 9$ e un insieme $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ di 6 oggetti ($n = 6$), con i seguenti pesi:

Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Pesi	w_j	2	3	6	7	5	9

Per esempio, il sottoinsieme $S = \{1, 4\}$ è ammissibile poiché $w_1 + w_4 = 2 + 7 = 9$. Anche il sottoinsieme $S = \{6\}$ è ammissibile poiché $w_6 = 9$. Il disegno mostra i tre sottoinsiemi i cui pesi riempiono esattamente la scatola di lunghezza $c = 9$: $\{1, 4\}$, $\{6\}$ e $\{2, 3\}$.



- Il SSP non è un **problema di ottimizzazione**, poiché la sua risposta è **sì** oppure **no**. Tuttavia, si può risolvere usando un problema di ottimizzazione.
- Un'istanza del SSP si può risolvere come un'istanza del KP ponendo il profitto di ogni oggetto uguale al suo peso, ossia

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

In questo caso, risolviamo il seguente KP:

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

Se $\sum_{j \in \tilde{S}} w_j = c$, la risposta al SSP è **sì**; altrimenti, se $\sum_{j \in \tilde{S}} w_j < c$, la risposta è **no**.

2.3.1 Ipotesi

- Facciamo le seguenti ipotesi:
 1. Il peso totale di tutti gli oggetti è almeno pari al valore obiettivo (altrimenti non può esistere alcun sottoinsieme ammissibile). Formalmente:

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

2. Il peso di ogni oggetto non supera il valore obiettivo (altrimenti l'oggetto non può appartenere ad alcun sottoinsieme ammissibile). Formalmente:

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

2.4 Il problema dello zaino limitato

Definizione 2.4: problema dello zaino limitato

Dati:

1. uno zaino di *capacità* $c \in \mathbb{Z}_{\geq 1}$,
2. un insieme $\mathcal{I} = \{1, 2, \dots, n\}$ di n oggetti,
3. per ogni oggetto $j \in \mathcal{I}$, un *profitto* $p_j \in \mathbb{Z}_{\geq 1}$, un *peso* $w_j \in \mathbb{Z}_{\geq 1}$ e un *numero di copie disponibili* $b_j \in \mathbb{Z}_{\geq 1}$,

il **problema dello zaino limitato** (BKP) chiede di decidere quante copie di ogni oggetto inserire nello zaino, in modo che il peso totale di tutte le copie selezionate non superi la capacità dello zaino, che nessun oggetto sia preso più volte delle sue copie disponibili e che il profitto totale di tutte le copie selezionate sia massimo.

- Un'istanza del BKP è specificata da:
 1. due numeri interi:
 - il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,
 - la capacità dello zaino $c \in \mathbb{Z}_{\geq 1}$,
 2. tre vettori di numeri interi di lunghezza n :
 - $\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, che contiene i profitti degli oggetti,
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, che contiene i pesi degli oggetti,
 - $\mathbf{b} \in \mathbb{Z}_{\geq 1}^n$, che contiene il numero di copie disponibili di ogni oggetto.
- Data un'istanza del BKP, definiamo l'**insieme esteso degli oggetti** come

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

dove ogni coppia (j, k) rappresenta la k -esima copia disponibile dell'oggetto j . La cardinalità dell'insieme esteso degli oggetti è

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

- Una **soluzione ammissibile del BKP** è un sottoinsieme di oggetti

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

tale che il peso totale degli oggetti selezionati non supera la capacità dello zaino, ossia

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

Di conseguenza, il BKP si può enunciare in modo equivalente come il problema di selezionare un sottoinsieme dell'insieme esteso degli oggetti il cui peso totale rispetta la capacità dello zaino e il cui profitto totale è massimo.

Data un'istanza del BKP, il BKP si può enunciare formalmente come:

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

Esempio 2.3: istanza del BKP

Consideriamo uno zaino di capacità $c = 9$ e un insieme $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ di 6 oggetti ($n = 6$), con i seguenti profitti, pesi e copie disponibili:

Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Profitti	p_j	6	5	8	9	6	7
Pesi	w_j	2	3	6	7	5	9
Copie disponibili	b_j	2	2	1	1	1	1

L'insieme esteso degli oggetti è:

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

Ogni copia (j, k) ha profitto p_j e peso w_j .

Etichetta della copia	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)
Profitto p_j	6	6	5	5	8	9	6	7
Peso w_j	2	2	3	3	6	7	5	9

Il disegno mostra una soluzione ottima: le due copie dell'oggetto 1 e la copia dell'oggetto 5, con peso $2 + 2 + 5 = 9$ e profitto $6 + 6 + 6 = 18$.

zaino, capacità $c = 9$

(1, 1)	(1, 2)	(5, 1)
$w_1 = 2$	$w_1 = 2$	$w_5 = 5$
$p_1 = 6$	$p_1 = 6$	$p_5 = 6$

peso $2 + 2 + 5 = 9$
 profitto $6 + 6 + 6 = 18$

copie non selezionate

$(2, 1)$ $w_2 = 3$ $p_2 = 5$	$(2, 2)$ $w_2 = 3$ $p_2 = 5$	$(4, 1)$ $w_4 = 7$ $p_4 = 9$
$(3, 1)$ $w_3 = 6$ $p_3 = 8$	$(6, 1)$ $w_6 = 9$ $p_6 = 7$	

2.4.1 Ipotesi

- Facciamo le seguenti ipotesi:
 - Il peso totale di tutte le copie disponibili di tutti gli oggetti supera la capacità dello zaino (altrimenti si potrebbero selezionare tutte le copie). Formalmente:

$$\sum_{j \in \mathcal{I}} b_j w_j > c$$
 - Ogni oggetto, da solo, entra nello zaino (altrimenti non può essere selezionato). Formalmente:

$$w_j \leq c, \quad \forall j \in \mathcal{I}$$
 - Il peso totale di tutte le copie disponibili di un singolo oggetto non supera la capacità dello zaino (altrimenti quell'oggetto non potrebbe mai essere preso nel numero massimo di copie consentito). Formalmente:

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

2.5 Il problema dello zaino illimitato

Definizione 2.5: problema dello zaino illimitato

Dati:

1. uno zaino di *capacità* $c \in \mathbb{Z}_{\geq 1}$,
2. un insieme $\mathcal{J} = \{1, 2, \dots, n\}$ di n oggetti,
3. per ogni oggetto $j \in \mathcal{J}$, disponibile in un numero illimitato di copie, un *profitto* $p_j \in \mathbb{Z}_{\geq 1}$ e un *peso* $w_j \in \mathbb{Z}_{\geq 1}$,

il **problema dello zaino illimitato** (UKP) chiede di decidere quante copie di ogni oggetto inserire nello zaino, in modo che il peso totale di tutte le copie selezionate non superi la capacità dello zaino e il profitto totale di tutte le copie selezionate sia massimo.

- Un'istanza dell'UKP è specificata da:

1. due numeri interi:

il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,
la capacità dello zaino $c \in \mathbb{Z}_{\geq 1}$,

2. due vettori di numeri interi di lunghezza n :

$\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, che contiene i profitti degli oggetti,
 $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, che contiene i pesi degli oggetti.

- Data un'istanza dell'UKP, definiamo prima, per ogni oggetto $j \in \mathcal{J}$, il **massimo numero ammissibile di copie**

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

ossia il massimo numero di copie dell'oggetto j che si possono inserire nello zaino senza superarne la capacità.

- Definiamo poi l'**insieme esteso degli oggetti** come

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

dove ogni coppia (j, k) rappresenta la k -esima copia dell'oggetto j . La cardinalità dell'insieme esteso degli oggetti è

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

- Una **soluzione ammissibile dell'UKP** è un sottoinsieme di oggetti

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

tale che il peso totale degli oggetti selezionati non supera la capacità dello zaino, ossia

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

Di conseguenza, l'UKP si può enunciare in modo equivalente come il problema di selezionare un sottoinsieme dell'insieme esteso degli oggetti il cui peso totale rispetta la capacità dello zaino e il cui profitto totale è massimo.

Data un'istanza dell'UKP, l'UKP si può enunciare formalmente come:

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

Esempio 2.4: istanza dell'UKP

Consideriamo uno zaino di capacità $c = 9$ e un insieme $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ di 6 oggetti ($n = 6$), con i seguenti profitti e pesi:

Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Profitti	p_j	6	5	8	9	6	7
Pesi	w_j	2	3	6	7	5	9

I massimi numeri ammissibili di copie dei singoli oggetti sono:

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

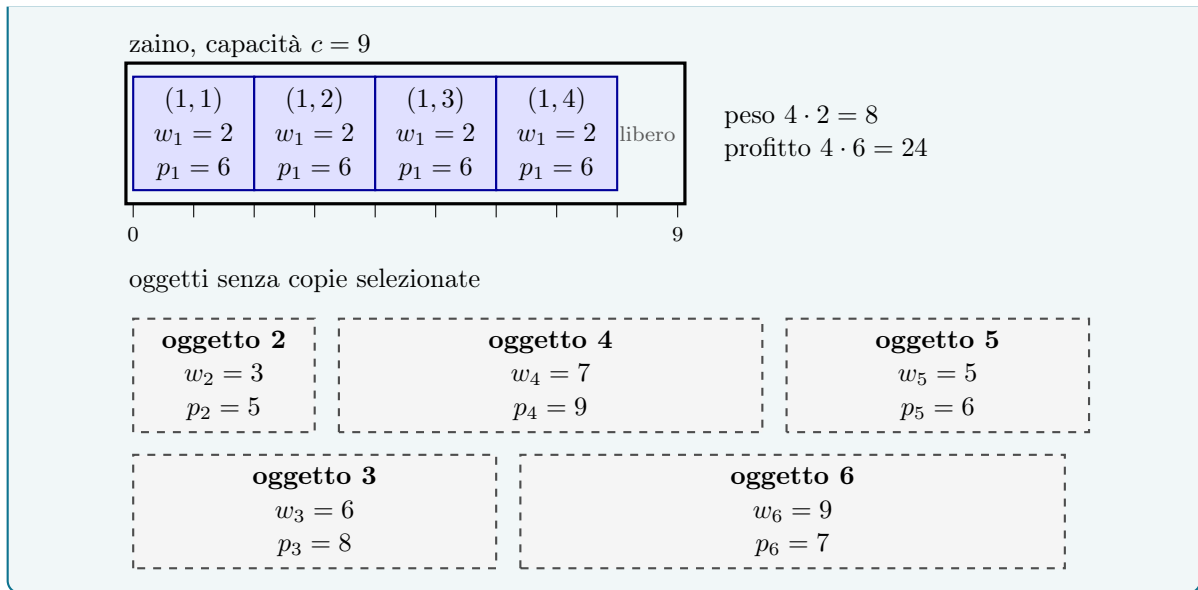
L'insieme esteso degli oggetti è:

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

Ogni copia (j, k) ha profitto p_j e peso w_j .

Etichetta della copia	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)
Profitto p_j	6	6	6	6	5	5	5	8	9	6	7
Peso w_j	2	2	2	2	3	3	3	6	7	5	9

Il disegno mostra una soluzione ottima: quattro copie dell'oggetto 1, con peso $4 \cdot 2 = 8$ e profitto $4 \cdot 6 = 24$; resta libera un'unità di capacità.



2.5.1 Ipotesi

- Facciamo le seguenti ipotesi:

- Il peso totale di tutte le copie possibili di tutti gli oggetti (fino al numero massimo che entra nello zaino) supera la capacità dello zaino (altrimenti si potrebbero selezionare tutte le copie). Formalmente:

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

- Ogni oggetto, da solo, entra nello zaino (altrimenti non può essere selezionato). Formalmente:

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

2.6 Il problema dello zaino multidimensionale

Definizione 2.6: problema dello zaino multidimensionale

Dati:

- uno zaino k -dimensionale,
- per ogni dimensione $d \in \underbrace{\{1, 2, \dots, k\}}_{=\mathcal{D}}$, una capacità $c_d \in \mathbb{Z}_{\geq 1}$,
- un insieme $\mathcal{I} = \{1, 2, \dots, n\}$ di n oggetti,
- per ogni oggetto $j \in \mathcal{I}$, un profitto $p_j \in \mathbb{Z}_{\geq 1}$,
- per ogni dimensione $d \in \mathcal{D}$ e ogni oggetto $j \in \mathcal{I}$, un peso $w_{dj} \in \mathbb{Z}_{\geq 1}$,

il **problema dello zaino multidimensionale** (MDKP) chiede di decidere quali oggetti inserire nello zaino, in modo che per ogni dimensione il peso totale degli oggetti selezionati

non superi la capacità corrispondente e il profitto totale degli oggetti selezionati sia massimo.

- Un'istanza dell'MDKP è specificata da:

1. due numeri interi:

il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,

il numero di dimensioni $k \in \mathbb{Z}_{\geq 1}$,

2. un vettore di numeri interi di lunghezza n :

$\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, che contiene i profitti degli oggetti,

3. un vettore di numeri interi di lunghezza k :

$\mathbf{c} \in \mathbb{Z}_{\geq 1}^k$, che contiene le capacità dello zaino per ogni dimensione,

4. una matrice di numeri interi con k righe e n colonne:

$\mathbf{W} \in \mathbb{Z}_{\geq 1}^{k \times n}$, che contiene i pesi degli oggetti per ogni dimensione.

- Una **soluzione ammissibile dell'MDKP** è un sottoinsieme di oggetti

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

tale che i vincoli di capacità sono soddisfatti in ogni dimensione, ossia

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

Di conseguenza, l'MDKP si può enunciare in modo equivalente come il problema di selezionare un sottoinsieme di oggetti il cui peso totale rispetta le capacità dello zaino in ogni dimensione e il cui profitto totale è massimo.

Data un'istanza dell'MDKP, l'MDKP si può enunciare formalmente come:

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

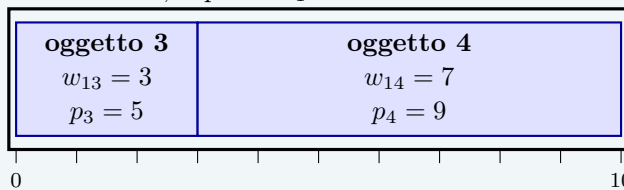
Esempio 2.5: istanza dell'MDKP

Consideriamo uno zaino 2-dimensionale ($k = 2$) con capacità $c_1 = 10$ e $c_2 = 12$, e un insieme $\mathcal{J} = \{1, 2, 3, 4\}$ di 4 oggetti ($n = 4$), con i seguenti profitti e pesi:

Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$
Profitti	p_j	6	7	5	9
Pesi (dim. 1)	w_{1j}	4	5	3	7
Pesi (dim. 2)	w_{2j}	6	4	5	3

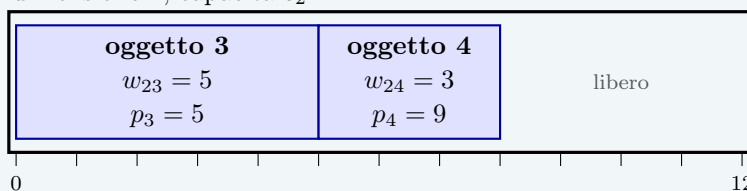
Il disegno ha una scatola per dimensione, di lunghezza $c_1 = 10$ e $c_2 = 12$: un oggetto selezionato occupa un blocco di lunghezza w_{1j} nella prima scatola e uno di lunghezza w_{2j} nella seconda. La soluzione ottima $\{3, 4\}$ riempie la prima dimensione e usa 8 unità della seconda, con profitto $5 + 9 = 14$.

dimensione 1, capacità $c_1 = 10$



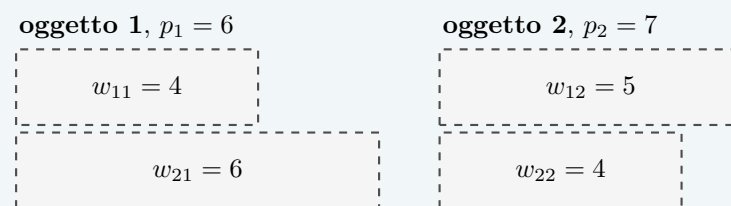
peso $3 + 7 = 10$

dimensione 2, capacità $c_2 = 12$



peso $5 + 3 = 8$

non selezionati (sopra: dimensione 1, sotto: dimensione 2)



2.6.1 Ipotesi

- Facciamo le seguenti ipotesi:

1. L'insieme di tutti gli oggetti *non* entra nello zaino (altrimenti selezionare tutti gli oggetti è ammissibile e ottimo). Formalmente,

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

2. Ogni oggetto, da solo, entra nello zaino in ogni dimensione (altrimenti non può mai essere selezionato). Formalmente,

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

2.7 Il problema degli zaini multipli

Definizione 2.7: problema degli zaini multipli

Dati:

1. $m \in \mathbb{Z}_{\geq 1}$ zaini,
2. per ogni zaino $k \in \underbrace{\{1, 2, \dots, m\}}_{=\mathcal{K}}$, una *capacità* $c_k \in \mathbb{Z}_{\geq 1}$,
3. un insieme $\mathcal{I} = \{1, 2, \dots, n\}$ di n oggetti,
4. per ogni oggetto $j \in \mathcal{I}$, un *profitto* $p_j \in \mathbb{Z}_{\geq 1}$ e un *peso* $w_j \in \mathbb{Z}_{\geq 1}$,

il **problema degli zaini multipli** (MKP) chiede di decidere quali oggetti inserire in ogni zaino, in modo che in ogni zaino il peso totale degli oggetti assegnati non superi la sua capacità e il profitto totale degli oggetti selezionati sia massimo.

- Un'istanza dell'MKP è specificata da:

1. due numeri interi:

il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,

il numero di zaini $m \in \mathbb{Z}_{\geq 1}$,

2. un vettore di numeri interi di lunghezza n :

$\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, che contiene i profitti degli oggetti,

3. un vettore di numeri interi di lunghezza n :

$\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, che contiene i pesi degli oggetti,

4. un vettore di numeri interi di lunghezza m :

$\mathbf{c} \in \mathbb{Z}_{\geq 1}^m$, che contiene le capacità degli zaini.

- Una **soluzione ammissibile dell'MKP** è una famiglia di sottoinsiemi di oggetti

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

tale che:

- (i) ogni oggetto è assegnato ad al più uno zaino, ossia

$$S_k \cap S_{k'} = \emptyset \text{ per ogni } k, k' \in \mathcal{K} \text{ con } k \neq k';$$

- (ii) il peso totale degli oggetti assegnati a ogni zaino rispetta la sua capacità, ossia

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

Di conseguenza, l'MKP si può enunciare in modo equivalente come il problema di selezionare sottoinsiemi di oggetti a due a due disgiunti, uno per ogni zaino, in modo che ogni capacità sia rispettata e il profitto totale sia massimo.

Data un'istanza dell'MKP, l'MKP si può enunciare formalmente come:

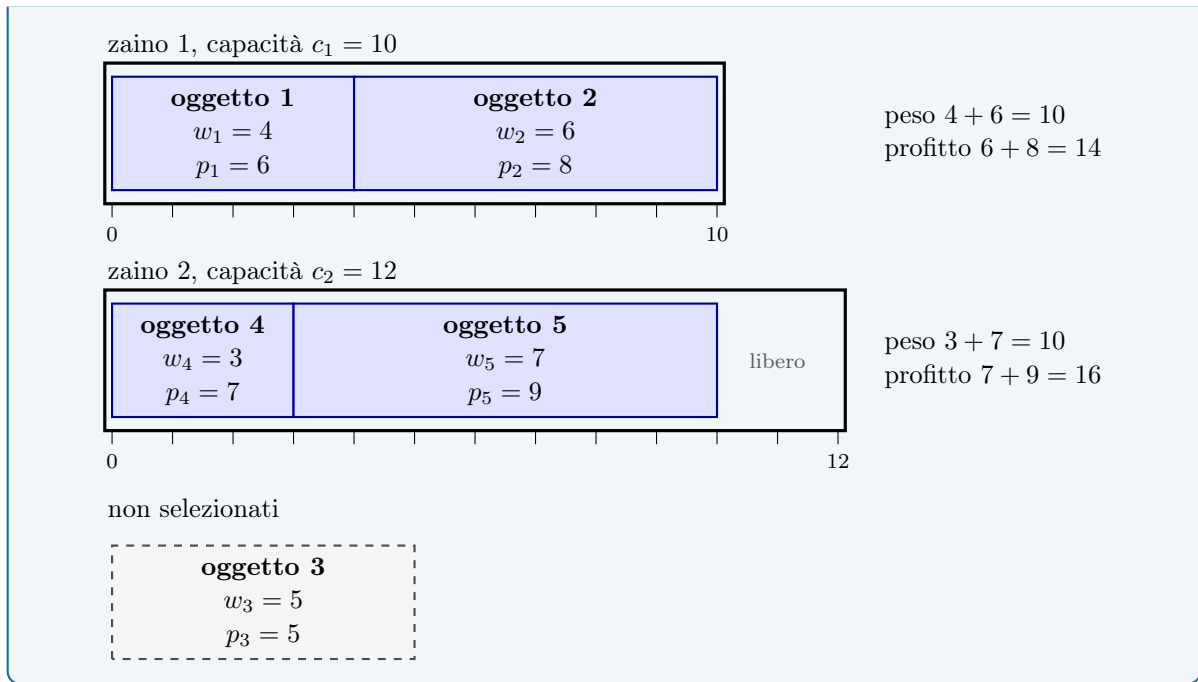
$$\tilde{\mathcal{S}} \in \arg \max_{\mathcal{S} = \{S_k\}_{k \in \mathcal{K}}} \left\{ \sum_{k \in \mathcal{K}} \sum_{j \in S_k} p_j : \begin{array}{l} S_k \subseteq \mathcal{I} \quad \forall k \in \mathcal{K}, \\ S_k \cap S_{k'} = \emptyset \quad \forall k, k' \in \mathcal{K} \text{ con } k \neq k',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

Esempio 2.6: istanza dell'MKP

Consideriamo $m = 2$ zaini con capacità $c_1 = 10$ e $c_2 = 12$, e un insieme $\mathcal{I} = \{1, 2, 3, 4, 5\}$ di 5 oggetti ($n = 5$), con i seguenti profitti e pesi:

Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
Profitti	p_j	6	8	5	7	9
Pesi	w_j	4	6	5	3	7

Il disegno mostra una soluzione ottima: $S_1 = \{1, 2\}$ nello zaino 1 e $S_2 = \{4, 5\}$ nello zaino 2, con profitto totale $14 + 16 = 30$; l'oggetto 3 resta fuori.



2.7.1 Ipotesi

- Facciamo le seguenti ipotesi:
 1. L'insieme di tutti gli oggetti *non* entra nel complesso degli zaini (altrimenti selezionare tutti gli oggetti è ammissibile e ottimo). Formalmente,

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

2. Ogni oggetto, da solo, entra in (almeno) uno zaino (altrimenti non può mai essere selezionato). Formalmente,

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

2.8 Il problema dello zaino a scelta multipla

Definizione 2.8: problema dello zaino a scelta multipla

Dati:

1. uno zaino di *capacità* $c \in \mathbb{Z}_{\geq 1}$,
2. un insieme $\mathcal{I} = \{1, 2, \dots, n\}$ di n oggetti,
3. una partizione dell'insieme degli oggetti in $q \in \mathbb{Z}_{\geq 2}$ classi,
4. per ogni classe $h \in \{1, 2, \dots, q\}$, il sottoinsieme $\mathcal{I}_h \subseteq \mathcal{I}$ degli oggetti di quella classe,
5. per ogni oggetto $j \in \mathcal{I}$, un *profitto* $p_j \in \mathbb{Z}_{\geq 1}$ e un *peso* $w_j \in \mathbb{Z}_{\geq 1}$,

il **problema dello zaino a scelta multipla** (MCKP) chiede di decidere quali oggetti inserire nello zaino, in modo che da ogni classe sia selezionato esattamente un oggetto, che il peso totale degli oggetti selezionati non superi la capacità dello zaino e che il profitto totale degli oggetti selezionati sia massimo.

- Un'istanza dell'MCKP è specificata da:

1. due numeri interi:
 - il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,
 - la capacità dello zaino $c \in \mathbb{Z}_{\geq 1}$,
2. un vettore di numeri interi di lunghezza n :
 - $\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, che contiene i profitti degli oggetti,
3. un vettore di numeri interi di lunghezza n :
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, che contiene i pesi degli oggetti,
4. un numero intero:
 - il numero di classi $q \in \mathbb{Z}_{\geq 2}$,
5. una famiglia di sottoinsiemi:
 - $\{\mathcal{I}_1, \mathcal{I}_2, \dots, \mathcal{I}_q\}$, una partizione di $\mathcal{I}$ in q classi, dove $\mathcal{I}_h \subseteq \mathcal{I}$ è l'insieme degli oggetti che appartengono alla classe h .

- Una **soluzione ammissibile** dell'MCKP è un sottoinsieme di oggetti

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

tale che:

- (i) da ogni classe è selezionato esattamente un oggetto, ossia

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

(ii) il peso totale degli oggetti selezionati non supera la capacità dello zaino, ossia

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

Di conseguenza, l'MCKP si può enunciare in modo equivalente come il problema di selezionare esattamente un oggetto da ogni classe in modo che la capacità dello zaino sia rispettata e il profitto totale sia massimo.

Data un'istanza dell'MCKP, l'MCKP si può enunciare formalmente come:

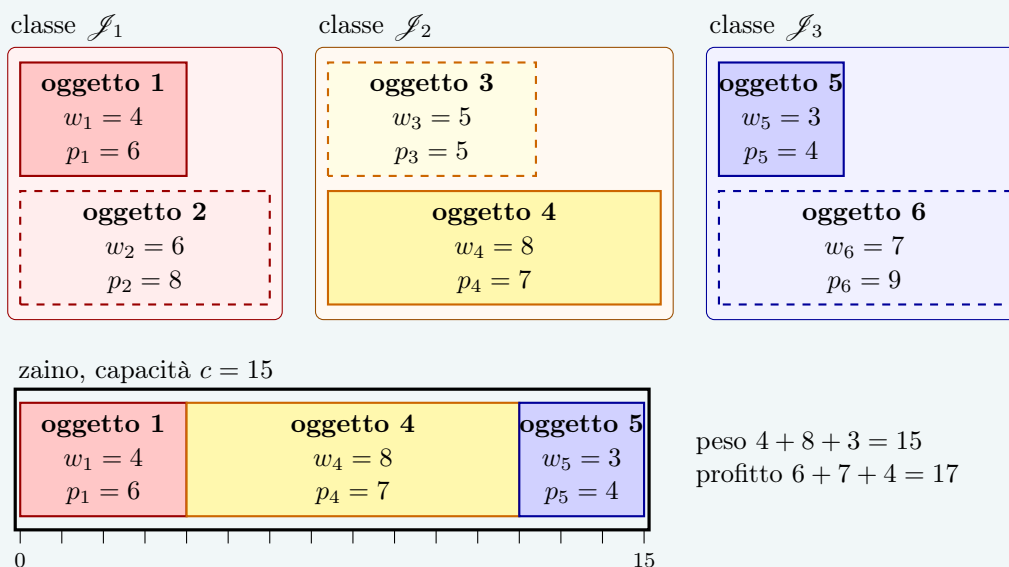
$$\tilde{S} \in \arg \max_{S \subseteq \mathcal{I}} \left\{ \sum_{j \in S} p_j : \sum_{j \in S} w_j \leq c, |S \cap \mathcal{J}_h| = 1 \ \forall h \in \{1, 2, \dots, q\} \right\}. \quad (2.10)$$

Esempio 2.7: istanza dell'MCKP

Consideriamo uno zaino di capacità $c = 15$ e un insieme $\mathcal{I} = \{1, 2, 3, 4, 5, 6\}$ di 6 oggetti ($n = 6$). Gli oggetti sono partizionati in $q = 3$ classi: $\mathcal{J}_1 = \{1, 2\}$, $\mathcal{J}_2 = \{3, 4\}$, $\mathcal{J}_3 = \{5, 6\}$. I profitti e i pesi sono i seguenti:

Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Profitti	p_j	6	8	5	7	4	9
Pesi	w_j	4	6	5	8	3	7

Nel disegno ogni classe è un gruppo colorato, e l'oggetto scelto in ogni classe è disegnato pieno, l'altro tratteggiato. Una soluzione ottima, $\{1, 4, 5\}$, riempie lo zaino, con peso $4 + 8 + 3 = 15$ e profitto $6 + 7 + 4 = 17$.



2.8.1 Ipotesi

- Facciamo le seguenti ipotesi:

1. Ogni classe contiene almeno un oggetto (altrimenti selezionare esattamente un oggetto da ogni classe è impossibile). Formalmente:

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

2. L'istanza è ammissibile, ossia esiste almeno una scelta di esattamente un oggetto per classe che entra nello zaino. Una condizione necessaria e sufficiente (e facile da verificare) è:

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

2.9 Il problema dello zaino con setup

Definizione 2.9: problema dello zaino con setup

Dati:

1. uno zaino di *capacità* $c \in \mathbb{Z}_{\geq 1}$,
2. un insieme $\mathcal{J} = \{1, 2, \dots, n\}$ di n oggetti,
3. una partizione dell'insieme degli oggetti in $q \in \mathbb{Z}_{\geq 2}$ classi,
4. per ogni classe $h \in \{1, 2, \dots, q\}$, il sottoinsieme $\mathcal{J}_h \subseteq \mathcal{J}$ degli oggetti di quella classe, con un *costo di setup* $f_h \in \mathbb{Z}_{\geq 1}$ e un *peso di setup* $s_h \in \mathbb{Z}_{\geq 1}$ associati,
5. per ogni oggetto $j \in \mathcal{J}$, un *ricavo* $p_j \in \mathbb{Z}_{\geq 1}$ e un *peso* $w_j \in \mathbb{Z}_{\geq 1}$,

il **problema dello zaino con setup** (KPS) chiede di decidere quali oggetti inserire nello zaino, in modo che il peso totale degli oggetti selezionati più i pesi di setup delle loro classi non superi la capacità dello zaino e il ricavo totale degli oggetti selezionati meno i costi di setup delle loro classi sia massimo.

- Un'istanza del KPS è specificata da:

1. due numeri interi:
 - il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,
 - la capacità dello zaino $c \in \mathbb{Z}_{\geq 1}$,
2. un vettore di numeri interi di lunghezza n :
 - $\mathbf{p} \in \mathbb{Z}_{\geq 1}^n$, che contiene i ricavi degli oggetti,
3. un vettore di numeri interi di lunghezza n :

- $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, che contiene i pesi degli oggetti,
- 4. un numero intero:
il numero di classi $q \in \mathbb{Z}_{\geq 2}$,
- 5. una famiglia di sottoinsiemi:
 $\{\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_q\}$, una partizione di $\mathcal{J}$ in q classi, dove $\mathcal{J}_h \subseteq \mathcal{J}$ è l'insieme degli oggetti che appartengono alla classe h ,
- 6. due vettori di numeri interi di lunghezza q :
 $\mathbf{f} \in \mathbb{Z}_{\geq 1}^q$, che contiene i costi di setup delle classi,
 $\mathbf{s} \in \mathbb{Z}_{\geq 1}^q$, che contiene i pesi di setup delle classi.

- Una **soluzione ammissibile del KPS** è un sottoinsieme di oggetti

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

Dato S , definiamo l'insieme delle **classi attivate** come

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

Il sottoinsieme S è ammissibile se il peso totale degli oggetti selezionati più i pesi di setup delle classi attivate non supera la capacità dello zaino, ossia

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

Di conseguenza, il KPS si può enunciare in modo equivalente come il problema di selezionare un sottoinsieme di oggetti che massimizza il ricavo totale degli oggetti selezionati meno i costi di setup delle classi attivate.

Data un'istanza del KPS, il KPS si può enunciare formalmente come:

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

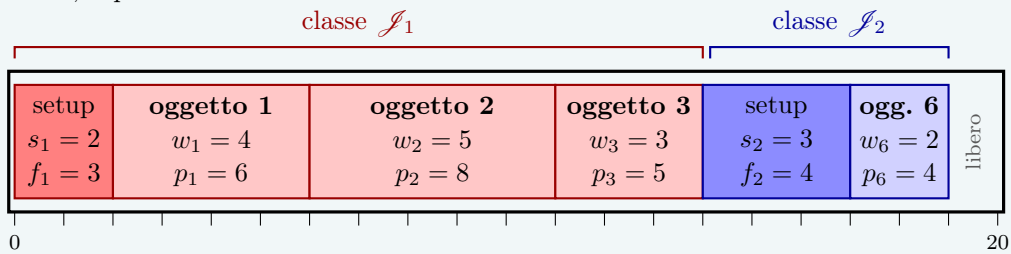
Esempio 2.8: istanza del KPS

Consideriamo uno zaino di capacità $c = 20$ e un insieme $\mathcal{J} = \{1, 2, 3, 4, 5, 6\}$ di 6 oggetti ($n = 6$). Gli oggetti sono partizionati in $q = 2$ classi: $\mathcal{J}_1 = \{1, 2, 3\}$ e $\mathcal{J}_2 = \{4, 5, 6\}$. Ogni classe ha un costo di setup e un peso di setup: $f_1 = 3$, $s_1 = 2$ per la classe 1, e $f_2 = 4$, $s_2 = 3$ per la classe 2. I ricavi e i pesi degli oggetti sono i seguenti:

Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
Ricavi	p_j	6	8	5	7	9	4
Pesi	w_j	4	5	3	6	7	2

Nel disegno le classi hanno due colori, e un blocco più scuro è il setup di una classe, con il suo peso s_h e il suo costo f_h . Una soluzione ottima, $\{1, 2, 3, 6\}$, attiva entrambe le classi: il peso è $(2 + 4 + 5 + 3) + (3 + 2) = 19 \leq 20$ e il valore è $(6 + 8 + 5 + 4) - (3 + 4) = 16$.

zaino, capacità $c = 20$



non selezionati

2.9.1 Ipotesi

- Facciamo le seguenti ipotesi:

- Ogni classe contiene almeno un oggetto. Formalmente:

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

- Ogni oggetto entra nello zaino insieme al peso di setup della sua classe (altrimenti non può mai essere selezionato). Formalmente:

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

2.10 Osservazioni

- Dantzig (1957) ordinò gli oggetti per efficienza p_j/w_j non crescente e usò questo ordine per risolvere la versione continua del KP; lo stesso ordine ritorna nei bound dei capitoli successivi.
- Karp (1972) inserì il problema del subset sum, chiamato knapsack nel suo articolo, fra i primi 21 problemi NP-completi. Il KP contiene il SSP come caso particolare $p_j = w_j$, quindi è NP-hard.
- Ibarra e Kim (1975) diedero il primo schema di approssimazione pienamente polinomiale (FPTAS) per il KP.

2.11 Errori tipici

Attenzione

- **Confondere il KP con il SSP.** Il SSP chiede un peso totale *esattamente* uguale a c e ha una risposta sì/no; è il KP con $p_j = w_j$, in cui la risposta è sì se il valore ottimo è c .
- **Fermarsi all'ordine di efficienza.** Prendere gli oggetti nell'ordine $1, 2, 3, \dots$ di p_j/w_j non crescente dà $\{1, 2\}$ con profitto 11, mentre la soluzione ottima dell'Esempio 2.1 è $\{1, 4\}$ con profitto 15.
- **Contare copie identiche come soluzioni diverse.** Nel BKP e nell'UKP le copie (j, k) dello stesso oggetto sono intercambiabili: sottoinsiemi diversi dell'insieme esteso degli oggetti possono dare lo stesso riempimento.
- **Scegliere al più un oggetto per classe.** Nell'MCKP si seleziona esattamente un oggetto per classe, quindi un'istanza può essere inammissibile; nel KPS una classe senza oggetti selezionati non costa nulla.
- **Dimenticare le ipotesi.** Un oggetto con $w_j > c$ non può mai essere selezionato, e se tutti gli oggetti entrano il problema è banale: entrambi i casi si eliminano prima di risolvere.

2.12 In sintesi

In sintesi

- Il KP seleziona un sottoinsieme di oggetti di peso al più c e profitto massimo; una soluzione è un sottoinsieme, e le candidate sono 2^n sottoinsiemi.
- Il SSP, il BKP, l'UKP, l'MDKP, l'MKP, l'MCKP e il KPS cambiano il numero di copie, di risorse, di zaini o la struttura degli oggetti; ciascuno ha le sue ipotesi sui dati.
- Una soluzione trovata a occhio è un bound primale; il bound di cardinalità è un bound duale che usa solo i pesi e i profitti.
- L'ordine di efficienza non basta a trovare una soluzione ottima, neppure su una piccola istanza.

Quadro riassuntivo dell'istanza di riferimento.

KP, Esempio 2.1 (max)	lower bound	upper bound	da
questo capitolo	15 (a occhio)	17 (bound di cardinalità)	definizione

Problemi di packing

Prerequisiti

- Da questo modulo, [Capitolo 1 «Problemi di ottimizzazione combinatoria»](#): istanze, soluzioni ammissibili, bound primali e duali, il gap e i numeri di Bell che contano le partizioni di un insieme.
- Da questo modulo, [Capitolo 2 «Problemi di zaino»](#): la forma delle definizioni e delle ipotesi sui dati.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- enunciare la definizione del problema del bin packing e del problema del cutting stock, con istanze e soluzioni ammissibili;
- spiegare le ipotesi sui dati;
- contare le soluzioni candidate di una piccola istanza e riconoscere che sono partizioni;
- trovare a mano una soluzione ammissibile e calcolare il bound sul peso;
- usare i due bound per certificare che una soluzione è ottima.

3.1 Panoramica della famiglia

- I **problemi di packing** hanno gli stessi dati: un numero infinito di contenitori identici di *capacità* $c \in \mathbb{Z}_{\geq 1}$, un insieme $\mathcal{J} = \{1, 2, \dots, n\}$ di n oggetti e, per ogni oggetto $j \in \mathcal{J}$, un *peso* $w_j \in \mathbb{Z}_{\geq 1}$.
- Una soluzione assegna ogni oggetto, o ogni copia richiesta di un oggetto, a un contenitore: il peso totale degli oggetti in un contenitore non deve superare la capacità c .
- L'obiettivo è minimizzare il numero di contenitori usati.

Un problema di packing sistema tutti gli oggetti nel minor numero di contenitori di capacità c . I membri della famiglia differiscono nel numero di copie di ogni oggetto da impaccare.

- La tabella elenca i membri della famiglia, con la sezione che definisce ciascuno di essi.

Problema	Sigla	Oggetti da impaccare	Sezione
problema del bin packing	BPP	una copia di ogni oggetto	3.2
problema del cutting stock	CSP	d_j copie dell'oggetto j	3.3

3.2 Il problema del bin packing

Definizione 3.1: problema del bin packing

Dati:

1. un numero infinito di contenitori, ciascuno di *capacità* $c \in \mathbb{Z}_{\geq 1}$,
2. un insieme $\mathcal{J} = \{1, 2, \dots, n\}$ di n oggetti,
3. per ogni oggetto $j \in \mathcal{J}$, un *peso* $w_j \in \mathbb{Z}_{\geq 1}$,

il **problema del bin packing** (BPP) chiede di decidere in quale contenitore sistemare ogni oggetto, in modo che il peso totale degli oggetti in ogni contenitore non superi la capacità del contenitore e il numero totale di contenitori usati sia minimo.

- Un'istanza del BPP è specificata da:
 1. due numeri interi:
 - la capacità dei contenitori $c \in \mathbb{Z}_{\geq 1}$,
 - il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,
 2. un vettore di numeri interi di lunghezza n :
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, che contiene i pesi degli oggetti.
- Una **soluzione ammissibile del BPP** è una famiglia di **schemi di carico**, ossia sottoinsiemi di oggetti ciascuno assegnato a un contenitore,

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

tale che:

(i) $\mathcal{P}$ forma una partizione di $\mathcal{I}$ in schemi di carico *non vuoti*, ossia

$$\begin{aligned} P &\neq \emptyset, \quad \text{per ogni } P \in \mathcal{P}, \\ P \cap P' &= \emptyset, \quad \text{per ogni } P, P' \in \mathcal{P} \text{ con } P \neq P', \\ \bigcup_{P \in \mathcal{P}} P &= \mathcal{I}; \end{aligned}$$

(ii) il peso totale di ogni schema di carico in $\mathcal{P}$ rispetta la capacità del contenitore, ossia

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

Di conseguenza, il BPP si può enunciare in modo equivalente come il problema di determinare una partizione dell'insieme degli oggetti in schemi di carico tale che il peso totale degli oggetti in ogni schema di carico non superi la capacità del contenitore e il numero di schemi di carico (contenitori) sia minimo.

Data un'istanza del BPP, il BPP si può enunciare formalmente come:

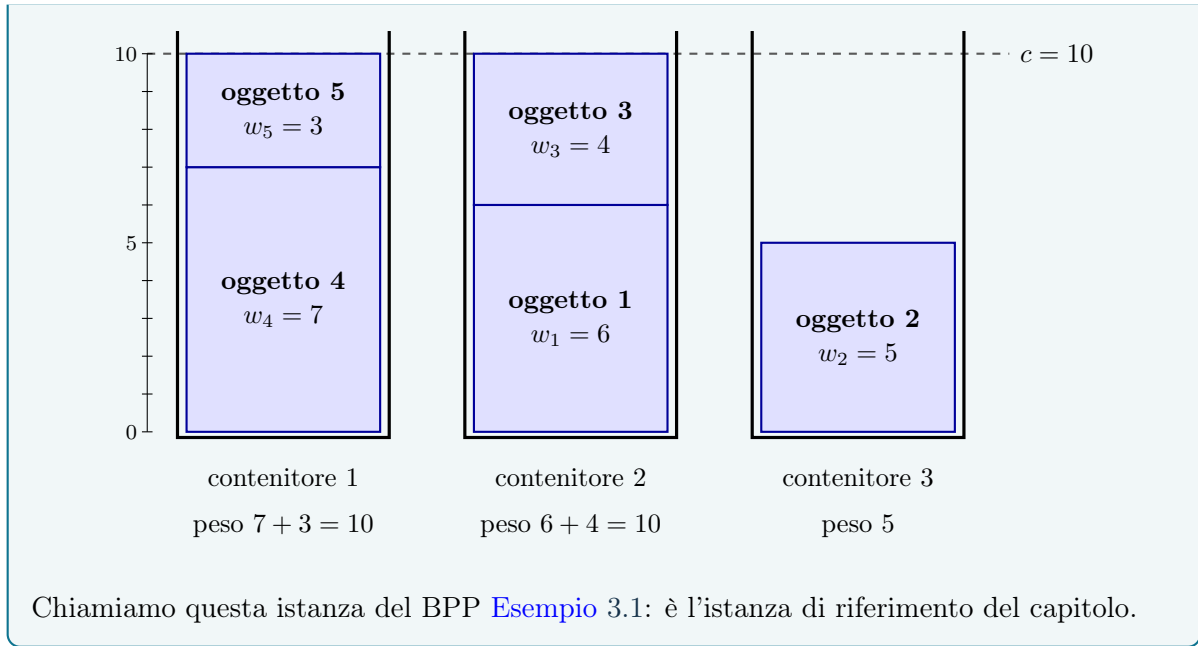
$$\tilde{\mathcal{P}} \in \arg \min_{\mathcal{P} \subseteq 2^{\mathcal{I}}} |\mathcal{P}| : \left\{ \begin{array}{l} P \neq \emptyset, \quad \forall P \in \mathcal{P}, \\ P \cap P' = \emptyset, \quad \forall P, P' \in \mathcal{P}, P \neq P', \\ \bigcup_{P \in \mathcal{P}} P = \mathcal{I}, \\ \sum_{j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

Esempio 3.1: istanze del BPP

Consideriamo contenitori di capacità $c = 10$ e un insieme $\mathcal{I} = \{1, 2, 3, 4, 5\}$ di 5 oggetti ($n = 5$), con i seguenti pesi:

Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
Pesi	w_j	6	5	4	7	3

Nel disegno ogni contenitore è una scatola di altezza $c = 10$ e ogni oggetto un blocco di altezza w_j . I tre contenitori contengono $\{4, 5\}$, $\{1, 3\}$ e $\{2\}$; mostriamo più avanti che 3 contenitori sono il minimo.



3.2.1 Ipotesi

- Facciamo le seguenti ipotesi:
 - Il peso totale di tutti gli oggetti supera la capacità di un contenitore (altrimenti tutti gli oggetti entrerebbero in un solo contenitore). Formalmente:

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

- Ogni oggetto, da solo, entra in un contenitore (altrimenti non può essere impaccato). Formalmente:

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

3.2.2 Contare le soluzioni, una prima soluzione e un bound semplice

- Soluzioni candidate.** Le soluzioni candidate sono le partizioni di $\mathcal{I}$: sono contate dai numeri di Bell ([Capitolo 1 «Problemi di ottimizzazione combinatoria»](#)). Per l'[Esempio 3.1](#) abbiamo $B(5) = 52$ partizioni.
- Soluzioni ammissibili.** Una partizione è ammissibile se ogni contenitore pesa al più 10. L'enumerazione dà 11 partizioni ammissibili: 1 con 5 contenitori (un oggetto per contenitore), 6 con 4 contenitori e 4 con 3 contenitori. Due contenitori non bastano, poiché il peso totale 25 supera $2 \cdot 10$.
- Una prima soluzione, a occhio.** Accoppiamo gli oggetti pesanti con quelli leggeri: l'oggetto 4 (peso 7) con l'oggetto 5 (peso 3), e l'oggetto 1 (peso 6) con l'oggetto 3 (peso 4); l'oggetto 2 (peso 5) resta da solo. La soluzione

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

usa 3 contenitori, di pesi 10, 10 e 5: è un **bound primale**, $UB = 3 \geq z(I)$.

Osservazione 3.2: bound sul peso

Ogni soluzione ammissibile usa almeno

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

contenitori, quindi questo valore è un bound duale.

Dimostrazione. Se una soluzione usa k contenitori, il peso totale che trasporta è al più kc . Il peso totale degli oggetti è $\sum_j w_j$, quindi $kc \geq \sum_j w_j$. Il numero k è intero, quindi $k \geq \lceil \sum_j w_j / c \rceil$. $\square$

- **Il bound duale e il gap.** Nell'Esempio 3.1 abbiamo $\sum_j w_j = 25$ e $c = 10$, quindi $LB = \lceil 25/10 \rceil = 3$. I due bound coincidono: il gap è nullo, la soluzione trovata a occhio è ottima e $z(I) = 3$, senza enumerare le partizioni.
- La soluzione ottima non è unica: ci sono 4 partizioni ottime,

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

3.3 Il problema del cutting stock

Definizione 3.3: problema del cutting stock

Dati:

1. un numero infinito di contenitori, ciascuno di capacità $c \in \mathbb{Z}_{\geq 1}$,
2. un insieme $\mathcal{J} = \{1, 2, \dots, n\}$ di n oggetti,
3. per ogni oggetto $j \in \mathcal{J}$, un peso $w_j \in \mathbb{Z}_{\geq 1}$ e un numero di copie richieste $d_j \in \mathbb{Z}_{\geq 1}$,

il **problema del cutting stock** (CSP) chiede di decidere in quale contenitore sistemare ogni copia di ogni oggetto, in modo che il peso totale delle copie in ogni contenitore non superi la capacità del contenitore, che il numero richiesto di copie di ogni oggetto sia sistemato nei contenitori e che il numero totale di contenitori usati sia minimo.

- Un'istanza del CSP è specificata da:
 1. due numeri interi:
 - la capacità dei contenitori $c \in \mathbb{Z}_{\geq 1}$,
 - il numero di oggetti $n \in \mathbb{Z}_{\geq 1}$,
 2. due vettori di numeri interi di lunghezza n :
 - $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$, che contiene i pesi degli oggetti,

$\mathbf{d} \in \mathbb{Z}_{\geq 1}^n$, che contiene il numero di copie richieste di ogni oggetto.

- Il problema si chiama *cutting stock* (taglio del materiale) perché, nella sua applicazione classica, i contenitori corrispondono a grandi rotoli di materiale (o lastre) che devono essere tagliati per produrre le copie richieste degli oggetti, dette anche domande degli oggetti.
- Data un'istanza del CSP, definiamo l'**insieme esteso degli oggetti** come

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

dove ogni coppia (j, k) rappresenta la k -esima copia richiesta dell'oggetto j . La cardinalità dell'insieme esteso degli oggetti è

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

- Una **soluzione ammissibile del CSP** è una famiglia di **schemi di carico**, ossia sottoinsiemi di copie ciascuno assegnato a un contenitore,

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

tale che:

- (i) $\mathcal{P}$ forma una partizione di $\hat{\mathcal{J}}$ in schemi di carico *non vuoti*, ossia

$$\begin{aligned} P &\neq \emptyset, \quad \text{per ogni } P \in \mathcal{P}, \\ P \cap P' &= \emptyset, \quad \text{per ogni } P, P' \in \mathcal{P} \text{ con } P \neq P', \\ \bigcup_{P \in \mathcal{P}} P &= \hat{\mathcal{J}}; \end{aligned}$$

- (ii) il peso totale delle copie in ogni schema di carico non supera la capacità del contenitore, ossia

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

Di conseguenza, il CSP si può enunciare in modo equivalente come il problema di determinare una partizione dell'insieme esteso degli oggetti in schemi di carico tale che il peso totale in ogni schema di carico non superi la capacità del contenitore e il numero di schemi di carico (contenitori) sia minimo.

Data un'istanza del CSP, il CSP si può enunciare formalmente come:

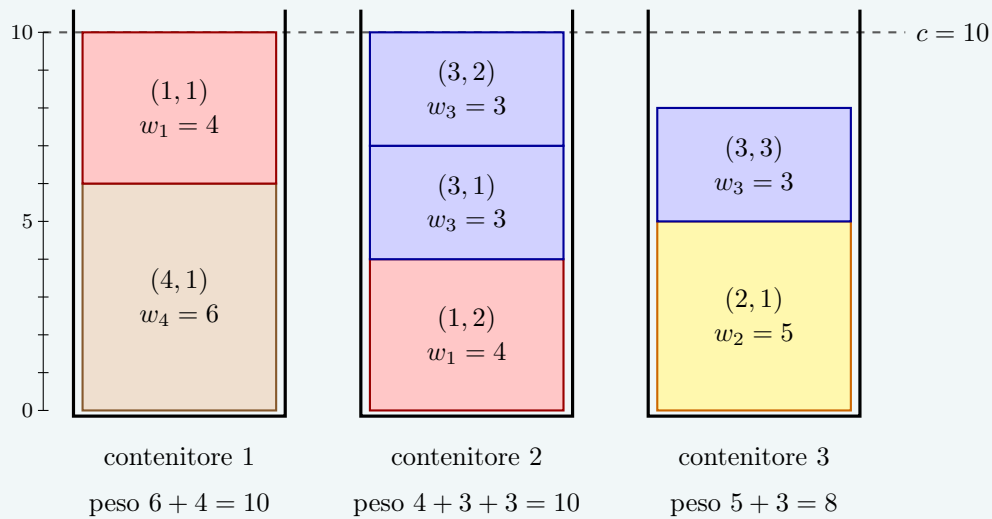
$$\tilde{\mathcal{P}} \in \arg \min_{\mathcal{P} \subseteq 2^{\hat{\mathcal{J}}}} \left\{ |\mathcal{P}| : \begin{aligned} &P \neq \emptyset, \quad \forall P \in \mathcal{P}, \\ &P \cap P' = \emptyset, \quad \forall P, P' \in \mathcal{P}, P \neq P', \\ &\bigcup_{P \in \mathcal{P}} P = \hat{\mathcal{J}},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

Esempio 3.2: istanze del CSP

Consideriamo contenitori di capacità $c = 10$ e un insieme $\mathcal{J} = \{1, 2, 3, 4\}$ di 4 oggetti ($n = 4$), con i seguenti pesi e copie richieste:

Oggetti		$j = 1$	$j = 2$	$j = 3$	$j = 4$
Pesi	w_j	4	5	3	6
Copie	d_j	2	1	3	1

Nel disegno le copie dello stesso oggetto hanno lo stesso colore, e la copia (j, k) è un blocco di altezza w_j . Le 7 copie entrano in 3 contenitori, di pesi 10, 10 e 8.



Chiamiamo questa istanza del CSP [Esempio 3.2](#): è l'istanza di riferimento del CSP nel capitolo.

3.3.1 Ipotesi

- Facciamo le seguenti ipotesi:
 - Il peso totale di tutte le copie richieste supera la capacità di un contenitore (altrimenti tutte le copie entrerebbero in un solo contenitore). Formalmente:

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

- Ogni oggetto, da solo, entra in un contenitore (altrimenti nessuna copia di quell'oggetto può essere impaccata). Formalmente:

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

3.3.2 Contare le soluzioni, una prima soluzione e un bound semplice

- **Soluzioni candidate.** L'istanza dell'Esempio 3.2 ha $2+1+3+1=7$ copie. Le partizioni dell'insieme esteso degli oggetti sono contate da $B(7) = 877$, dove copie identiche sono contate come oggetti diversi; 262 di esse sono ammissibili.
- **Una prima soluzione, a occhio.** Accoppiamo l'oggetto 4 (peso 6) con una copia dell'oggetto 1 (peso 4), mettiamo l'altra copia dell'oggetto 1 con due copie dell'oggetto 3 ($4+3+3$), e l'ultima copia dell'oggetto 3 con l'oggetto 2 ($3+5$):

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

usa 3 contenitori, di pesi 10, 10 e 8, quindi $UB = 3$.

- **Il bound duale e il gap.** Il bound sul peso del bin packing vale con i pesi di tutte le copie: $LB = \lceil \sum_j d_j w_j / c \rceil = \lceil 28/10 \rceil = 3$. Il gap è nullo, quindi la soluzione è ottima e $z(I) = 3$.

3.4 Osservazioni

- Il comportamento nel caso peggiore di semplici regole di packing, come First Fit, è stato analizzato da Garey, Graham e Ullman (1972) e da Johnson, Demers, Ullman, Garey e Graham (1974).
- Johnson et al. (1974) dimostrarono che First Fit usa al più $\frac{17}{10}$ del numero ottimo di contenitori, più 2.

3.5 Errori tipici

Attenzione

- **Credere che il bound sul peso sia sempre stretto.** Con tre oggetti di peso 6 e $c = 10$ il bound dà $\lceil 18/10 \rceil = 2$, ma due oggetti non entrano insieme, quindi servono 3 contenitori.
- **Contare i contenitori solo in base ai pesi.** Il bound sul peso ignora come gli oggetti si combinano; è un bound duale, e il bound primale richiede comunque una soluzione.
- **Elencare due volte lo stesso riempimento.** Nel problema del cutting stock le copie di un oggetto sono intercambiabili: partizioni che differiscono solo per lo scambio di copie identiche sono lo stesso riempimento.
- **Ignorare che l'ordine dei contenitori non conta.** Una soluzione è una partizione: $\{\{4,5\}, \{1,3\}, \{2\}\}$ e $\{\{2\}, \{4,5\}, \{1,3\}\}$ sono la stessa soluzione.
- **Dimenticare le ipotesi.** Un oggetto con $w_j > c$ non entra in alcun contenitore, e se il peso totale è al più c basta un contenitore.

3.6 In sintesi

In sintesi

- Il BPP partiziona gli oggetti nel numero minimo di contenitori di capacità c ; il problema del cutting stock è il BPP con più copie identiche di ogni oggetto, e il BPP è il caso $d_j = 1$.
- Una soluzione è una partizione, quindi le candidate sono contate dai numeri di Bell e crescono molto rapidamente.
- Una soluzione trovata a occhio è un bound primale; il bound sul peso $\lceil \sum_j w_j/c \rceil$ è un bound duale.
- Quando i due bound coincidono, il gap è nullo e la soluzione è certificata ottima.

Quadro riassuntivo delle istanze di riferimento.

istanza (min)	lower bound	upper bound	da
BPP, Esempio 3.1	3 (bound sul peso)	3 (a occhio)	questo capitolo
CSP, Esempio 3.2	3 (bound sul peso)	3 (a occhio)	questo capitolo

Problemi di colorazione

Prerequisiti

- Da questo modulo, [Capitolo 1 «Problemi di ottimizzazione combinatoria»](#): istanze, soluzioni ammissibili, bound primali e duali, il gap e i numeri di Bell che contano le partizioni di un insieme.
- Dal [Capitolo 3 «Problemi di packing»](#): il bound sul peso come esempio di bound duale.
- Non serve alcuna conoscenza di teoria dei grafi: le definizioni usate qui sono richiamate in una breve sezione.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- usare le nozioni di base sui grafi: intorno, grado, grafo indotto, grafo complementare, clique e insieme stabile;
- enunciare la definizione del problema della colorazione dei vertici e le ipotesi sul grafo;
- contare le soluzioni candidate di una piccola istanza e riconoscere che sono partizioni;
- trovare a mano una colorazione e calcolare il bound di clique e il bound degli insiemi stabili;
- spiegare perché il bound di clique non è sempre stretto.

4.1 Panoramica della famiglia

- I **problemi di colorazione** hanno gli stessi dati: un grafo semplice non orientato $G = (V, E)$, i cui lati sono le coppie di vertici in conflitto.
- Una soluzione assegna un colore a ogni vertice, in modo che i due estremi di ogni lato ricevano colori diversi: i vertici con lo stesso colore formano un insieme stabile di G .
- L'obiettivo è minimizzare il numero di colori usati.

Un problema di colorazione partiziona i vertici di un grafo nel minor numero di insiemi stabili. Le nozioni sui grafi della sottosezione seguente sono la notazione comune della famiglia.

- La tabella elenca i membri della famiglia, con la sezione che definisce ciascuno di essi.

Problema	Sigla	Soluzione	Sezione
problema della colorazione dei vertici	VCP	una partizione di V in insiemi stabili	4.2

4.1.1 Un breve richiamo sui grafi

Definizione 4.1: grafo, intorno, grafo indotto e grafo complementare

- Un **grafo** $G = (V, E)$ è dato da un insieme finito V di **vertici** e da un insieme E di **lati**, ciascuno dei quali è un sottoinsieme di 2 elementi di V . Scriviamo $n = |V|$ e $m = |E|$; i vertici sono etichettati $1, 2, \dots, n$. Tutti i grafi di questo capitolo sono semplici e non orientati.
- Due vertici distinti u e v sono **adiacenti** se $\{u, v\} \in E$. L'**intorno** di un vertice v è l'insieme $N(v) = \{u \in V : \{u, v\} \in E\}$, e il suo **grado** è $\deg(v) = |N(v)|$.
- Per un sottoinsieme $U \subseteq V$ il **grafo indotto** $G[U] = (U, E[U])$ ha i vertici di U e i lati di E con entrambi gli estremi in U .
- Il **grafo complementare** $\overline{G} = (V, \overline{E})$ ha gli stessi vertici di G , e due vertici distinti sono adiacenti in $\overline{G}$ se e solo se non sono adiacenti in G .

Definizione 4.2: clique e insieme stabile

Sia $G = (V, E)$ un grafo.

1. Una **clique** è un sottoinsieme $K \subseteq V$ di vertici a due a due adiacenti: $\{u, v\} \in E$ per ogni $u, v \in K$ con $u \neq v$.
2. Un **insieme stabile** (detto anche **insieme indipendente**) è un sottoinsieme $Q \subseteq V$ di vertici a due a due non adiacenti: $\{u, v\} \notin E$ per ogni $u, v \in Q$ con $u \neq v$.

Il **numero di clique** $\omega(G)$ è il massimo numero di vertici di una clique, e il **numero di stabilità** $\alpha(G)$ è il massimo numero di vertici di un insieme stabile.

- Un insieme stabile di G è una clique del grafo complementare $\overline{G}$, e viceversa. Un vertice da solo e l'insieme vuoto sono sia una clique sia un insieme stabile.

Una clique è un gruppo di vertici a due a due in *conflitto*; un insieme stabile è un gruppo di vertici senza conflitti. In una colorazione, i vertici di una clique richiedono colori diversi, e i vertici di un insieme stabile possono dividerne uno.

4.2 Il problema della colorazione dei vertici

Definizione 4.3: problema della colorazione dei vertici

Dato:

1. un *grafo* semplice $G = (V, E)$, dove

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

è l'insieme degli n vertici e

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

è l'insieme degli m lati,

il **problema della colorazione dei vertici** (VCP) chiede di decidere come assegnare un colore a ogni vertice, in modo che due vertici adiacenti non abbiano lo stesso colore e il numero totale di colori usati sia minimo.

- Un'istanza del **VCP** è specificata da:
 1. due numeri interi:
 - il numero di vertici $n \in \mathbb{Z}_{\geq 1}$,
 - il numero di lati $m \in \mathbb{Z}_{\geq 1}$,
 2. due vettori di numeri interi di lunghezza m :
 - $\mathbf{u} \in \mathbb{Z}_{\geq 1}^m$, che contiene il primo estremo dei lati,
 - $\mathbf{v} \in \mathbb{Z}_{\geq 1}^m$, che contiene il secondo estremo dei lati.
- Una **soluzione ammissibile del VCP** è una famiglia di **classi di colore**, ossia sottoinsiemi di vertici ciascuno assegnato a un colore,

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

tale che:

- (i) $\mathcal{Q}$ forma una partizione di V in classi di colore *non vuote*, ossia

$$\begin{aligned} Q &\neq \emptyset, \quad \text{per ogni } Q \in \mathcal{Q}, \\ Q \cap Q' &= \emptyset, \quad \text{per ogni } Q, Q' \in \mathcal{Q} \text{ con } Q \neq Q', \\ \bigcup_{Q \in \mathcal{Q}} Q &= V; \end{aligned}$$

(ii) ogni classe di colore in $\mathcal{Q}$ è un *insieme stabile*, ossia

$$\{u, v\} \notin E, \quad \text{per ogni } Q \in \mathcal{Q}, \text{ per ogni } u, v \in Q \text{ con } u \neq v.$$

Di conseguenza, il VCP si può enunciare in modo equivalente come il problema di determinare una partizione dell'insieme dei vertici in classi di colore tale che ogni classe di colore sia un insieme stabile (ossia i suoi vertici possano ricevere lo stesso colore) e il numero di classi di colore (colori) sia minimo.

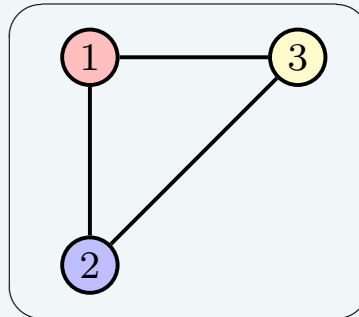
Data un'istanza del VCP, il VCP si può enunciare formalmente come:

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

- Il valore ottimo del VCP è il **numero cromatico** $\chi(G)$ del grafo: il numero minimo di colori (insiemi stabili) necessari per colorare tutti i vertici. Le classi di colore di una soluzione sono insiemi stabili, e il numero di classi è il numero di colori usati.

Esempio 4.1: istanza del VCP: un triangolo

Consideriamo il seguente grafo $G = (V, E)$ con 3 vertici ($n = 3$) e 3 lati ($m = 3$):



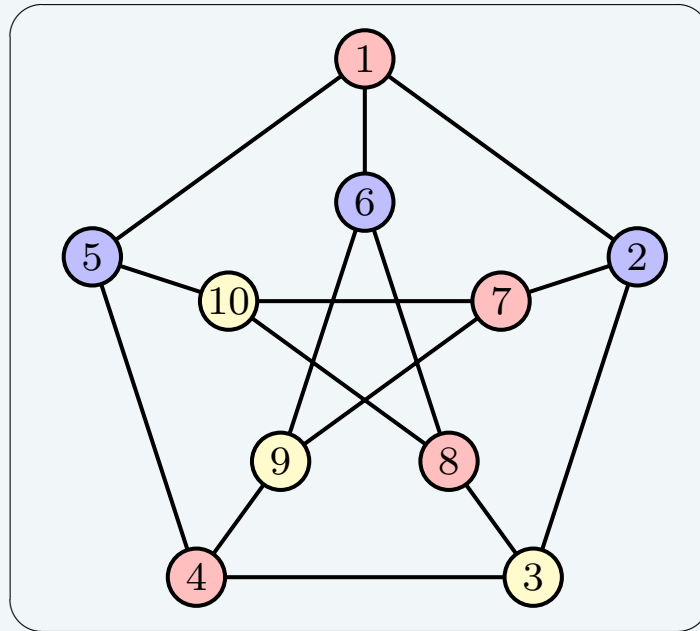
L'insieme dei vertici V e l'insieme dei lati E sono:

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

Chiamiamo questa istanza del VCP [Esempio 4.1](#).

Esempio 4.2: istanza del VCP: il grafo di Petersen

Consideriamo ora il seguente grafo $G = (V, E)$ con 10 vertici ($n = 10$) e 15 lati ($m = 15$):



L'insieme dei vertici V e l'insieme dei lati E sono:

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

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

Chiamiamo questa istanza del VCP [Esempio 4.2](#): è l'istanza di riferimento del capitolo.

4.2.1 Ipotesi

- Facciamo le seguenti ipotesi:

1. Il grafo non ha vertici isolati. Un vertice isolato può ricevere un qualunque colore usato dal resto del grafo; se il grafo non ha lati, il valore ottimo è uno. Formalmente:

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

2. Nessun vertice ha l'intorno contenuto nell'intorno di un altro vertice. Formalmente:

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

- La seconda ipotesi si può fare senza perdita di generalità, perché:

1. se $N(u) \subseteq N(v)$, il vertice u si può eliminare senza cambiare il numero cromatico: ogni colorazione di $G \setminus \{u\}$ si estende a u con il colore di v ;
2. i vertici u e v non sono adiacenti: altrimenti $v \in N(u) \subseteq N(v)$, il che è impossibile in un grafo semplice, quindi u e v possono condividere un colore.

4.2.2 Due bound semplici

Osservazione 4.4: bound di clique

Per ogni grafo $G = (V, E)$ abbiamo

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

Dimostrazione. Poiché tutti i vertici di una clique sono a due a due adiacenti, devono ricevere colori diversi in ogni soluzione ammissibile del VCP. Quindi servono almeno $\omega(G)$ colori diversi per colorare i vertici di una clique massima, da cui $\omega(G) \leq \chi(G)$. $\square$

Osservazione 4.5: bound degli insiemi stabili

Per ogni grafo $G = (V, E)$ abbiamo

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

Dimostrazione. In una soluzione ammissibile con k classi di colore, ogni classe è un insieme stabile, quindi ha al più $\alpha(G)$ vertici. Gli n vertici sono coperti dalle classi, quindi $n \leq k \alpha(G)$. Il numero k è intero, quindi $k \geq \lceil n/\alpha(G) \rceil$. $\square$

- Il bound di clique è utile quando il grafo ha una clique grande, e il bound degli insiemi stabili quando i suoi insiemi stabili sono piccoli. Entrambi sono bound duali: una clique e un insieme stabile si trovano senza conoscere $\chi(G)$.

4.2.3 Contare le soluzioni, una prima soluzione e i bound

- **Soluzioni candidate.** Le soluzioni candidate sono le partizioni di V , contate dai numeri di Bell (Capitolo 1 «Problemi di ottimizzazione combinatoria»). Per l'Esempio 4.1 abbiamo $B(3) = 5$ partizioni e una sola è ammissibile: tre singoletti, poiché ogni coppia di vertici è adiacente. Quindi $\chi(G) = 3$.
- Per l'Esempio 4.2 abbiamo $B(10) = 115\,975$ partizioni, e 7430 di esse sono ammissibili, cioè hanno come classi solo insiemi stabili.
- **Una prima soluzione, a occhio.** La colorazione mostrata nell'Esempio 4.2 usa 3 colori:

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

Nessun lato unisce due vertici della stessa classe, quindi è ammissibile: è un **bound primale**, $UB = 3 \geq \chi(G)$.

- **Bound duali sul grafo di Petersen.** Il grafo non ha triangoli, quindi $\omega(G) = 2$ e il bound di clique dà solo $LB = 2$: resta un gap di 1.
- I vertici 1, 2, 3, 4, 5 formano un ciclo con 5 vertici, e così 6, 7, 8, 9, 10 (i lati della figura). Un insieme stabile contiene al più 2 vertici di ogni ciclo, quindi $\alpha(G) \leq 4$; l'insieme $\{1, 4, 7, 8\}$ mostra che $\alpha(G) = 4$.

- Il bound degli insiemi stabili dà $\lceil 10/4 \rceil = 3$. I due bound coincidono: il gap è nullo e $\chi(G) = 3$.

4.3 Osservazioni

- Brooks (1941) dimostrò che un grafo connesso che non è né completo né un ciclo dispari si può colorare con al più Δ colori, dove Δ è il grado massimo.
- Karp (1972) incluse il numero cromatico fra i suoi 21 problemi NP-completi. Garey, Johnson e Stockmeyer (1976) dimostrarono che stabilire se 3 colori bastano resta NP-completo sui grafi planari di grado massimo 4.

4.4 Errori tipici

Attenzione

- **Confondere una clique con un insieme stabile.** In una clique tutte le coppie sono lati e i vertici richiedono colori diversi; in un insieme stabile nessuna coppia è un lato e i vertici possono condividere un colore.
- **Credere che il bound di clique sia stretto.** Non lo è: il grafo di Petersen ha $\omega(G) = 2$ e $\chi(G) = 3$, e lo stesso accade per un ciclo con 5 vertici.
- **Contare le permutazioni dei colori come soluzioni diverse.** Una soluzione è una partizione dei vertici in insiemi stabili: scambiare i nomi di due colori dà la stessa soluzione.
- **Usare una classe che non è un insieme stabile.** Una classe di colore con due vertici adiacenti rende la soluzione inammissibile, qualunque sia il numero di colori.
- **Dimenticare le ipotesi.** Un vertice isolato e un vertice con $N(u) \subseteq N(v)$ si possono eliminare prima di risolvere, senza cambiare il numero cromatico.

4.5 In sintesi

In sintesi

- Il VCP partiziona i vertici di un grafo nel numero minimo di insiemi stabili; il valore ottimo è il numero cromatico $\chi(G)$.
- Una clique richiede colori diversi e un insieme stabile può dividerne uno; le clique di G sono gli insiemi stabili del grafo complementare.
- Una colorazione trovata a occhio è un bound primale; il bound di clique $\omega(G)$ e il bound degli insiemi stabili $\lceil n/\alpha(G) \rceil$ sono bound duali.
- I bound non sempre si incontrano: sul grafo di Petersen il bound di clique lascia un gap, il bound degli insiemi stabili lo chiude.

Quadro riassuntivo delle istanze di riferimento.

istanza (min)	lower bound	upper bound	da
Esempio 4.1 (triangolo)	3 (bound di clique)	3 (a occhio)	questo capitolo
Esempio 4.2 (Petersen)	3 (bound degli insiemi stabili)	3 (a occhio)	questo capitolo

Problemi di clique

Prerequisiti

- Da questo modulo, il capitolo «[Problemi di ottimizzazione combinatoria](#)»: istanze, soluzioni ammissibili, bound primali e duali, e il gap.
- Da questo modulo, il capitolo «[Problemi di colorazione](#)»: grafi, intorni, gradi, il grafo complementare e il numero cromatico $\chi(G)$.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- enunciare il problema della clique massima e il problema dell'insieme stabile massimo, e passare dall'uno all'altro attraverso il grafo complementare;
- descrivere un'istanza e una soluzione ammissibile, e contare le soluzioni candidate di un piccolo grafo;
- trovare a mano una clique e calcolare quattro semplici upper bound sul numero di clique;
- enunciare le varianti pesate del problema e la sua complessità.

5.1 Panoramica della famiglia

- I **problemi di clique** hanno gli stessi dati: un grafo semplice non orientato $G = (V, E)$, con $V = \{1, 2, \dots, n\}$ e m lati, eventualmente con pesi sui vertici o sui lati.
- Una soluzione è una clique di G , ossia un sottoinsieme di vertici a due a due adiacenti.
- L'obiettivo è massimizzare la cardinalità della clique, o il suo peso totale.

Definizione 5.1: clique e numero di clique

Sia $G = (V, E)$ un grafo semplice non orientato con n vertici e m lati.

1. Una **clique** è un sottoinsieme $K \subseteq V$ di vertici a due a due adiacenti: $\{u, v\} \in E$ per ogni $u, v \in K$ con $u \neq v$. Equivalentemente, K induce un grafo completo.
2. Il **numero di clique** $\omega(G)$ è il massimo numero di vertici di una clique di G .

Un problema di clique seleziona una clique di cardinalità massima o di peso massimo. Sul grafo complementare $\overline{G}$ gli stessi problemi selezionano insiemi stabili.

- La tabella elenca i membri della famiglia, con la sezione che definisce ciascuno di essi.

Problema	Sigla	Obiettivo	Sezione
clique massima	MCP	massimo numero di vertici di una clique	5.2
insieme stabile massimo	MSSP	massimo numero di vertici di un insieme stabile	5.2
clique massima pesata sui vertici	VWMCP	massimo peso totale dei vertici	5.5
clique massima pesata sui lati	EWMCP	massimo peso totale dei lati	5.5

5.2 Il problema della clique massima

Definizione 5.2: problema della clique massima

Dato:

1. un *grafo* semplice $G = (V, E)$, dove

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

è l'insieme degli n *vertici* e

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

è l'insieme degli m *lati*,

il **problema della clique massima** (MCP) chiede di trovare una clique $K \subseteq V$ di G il cui numero di vertici sia massimo. Il suo valore ottimo è il numero di clique $\omega(G)$.

Definizione 5.3: insieme stabile e problema dell'insieme stabile massimo

Sia $G = (V, E)$ un grafo.

1. Un **insieme stabile** (detto anche **insieme indipendente**) è un sottoinsieme $Q \subseteq V$ di vertici a due a due non adiacenti: $\{u, v\} \notin E$ per ogni $u, v \in Q$ con $u \neq v$.
2. Il **problema dell'insieme stabile massimo** (MSSP) chiede di trovare un insieme stabile di G il cui numero di vertici sia massimo. Il suo valore ottimo è il **numero di stabilità** $\alpha(G)$.

Definizione 5.4: grafo complementare

Il **grafo complementare** di $G = (V, E)$ è il grafo $\bar{G} = (V, \bar{E})$ con gli stessi vertici e l'insieme dei lati

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

Le coppie in $\bar{E}$ sono i **non-lati** di G , e $|\bar{E}| = \binom{n}{2} - m$.

Proposizione 5.5: numero di clique e numero di stabilità

Un sottoinsieme $K \subseteq V$ è una clique di G se e solo se è un insieme stabile di $\bar{G}$. Quindi

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

Dimostrazione. Due vertici distinti u, v sono adiacenti in G se e solo se non sono adiacenti in $\bar{G}$. Quindi tutte le coppie di K sono lati di G esattamente quando nessuna coppia di K è un lato di $\bar{G}$. I due problemi hanno le stesse soluzioni ammissibili, quindi lo stesso valore ottimo. La seconda uguaglianza segue poiché $\bar{\bar{G}} = G$. $\square$

Il MCP su G e il MSSP su $\bar{G}$ sono lo stesso problema: una clique massima di G è un insieme stabile massimo di $\bar{G}$. Tutto ciò che impariamo su uno dei due problemi vale anche per l'altro.

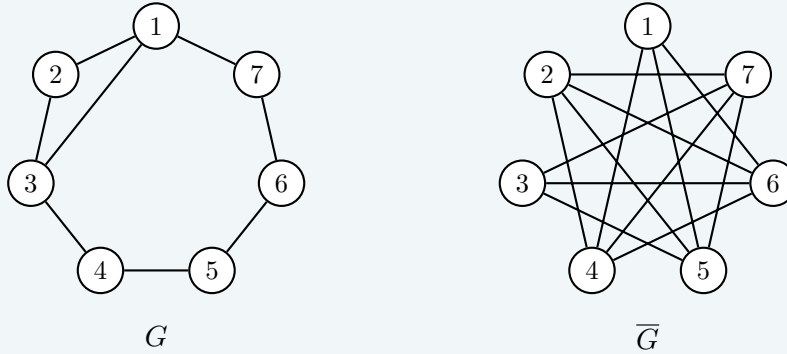
- Un'istanza del MCP è specificata da:
 1. due numeri interi: il numero di vertici $n \in \mathbb{Z}_{\geq 1}$ e il numero di lati $m \in \mathbb{Z}_{\geq 1}$;
 2. due vettori di numeri interi di lunghezza m : $\mathbf{u} \in \mathbb{Z}_{\geq 1}^m$ e $\mathbf{v} \in \mathbb{Z}_{\geq 1}^m$, con i due estremi dei lati.
- Una **soluzione ammissibile del MCP** è una clique $K \subseteq V$. L'insieme vuoto e un singolo vertice sono clique, quindi ogni istanza ha soluzioni ammissibili.

Data un'istanza del MCP, il MCP si può enunciare formalmente come:

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

Esempio 5.1: istanza del MCP: un ciclo con una corda

Consideriamo il grafo $G = (V, E)$ con 7 vertici ($n = 7$) e 8 lati ($m = 8$): il ciclo $1, 2, \dots, 7, 1$ e la corda $\{1, 3\}$. A destra, il suo grafo complementare $\bar{G}$, con i $\binom{7}{2} - 8 = 13$ non-lati di G .



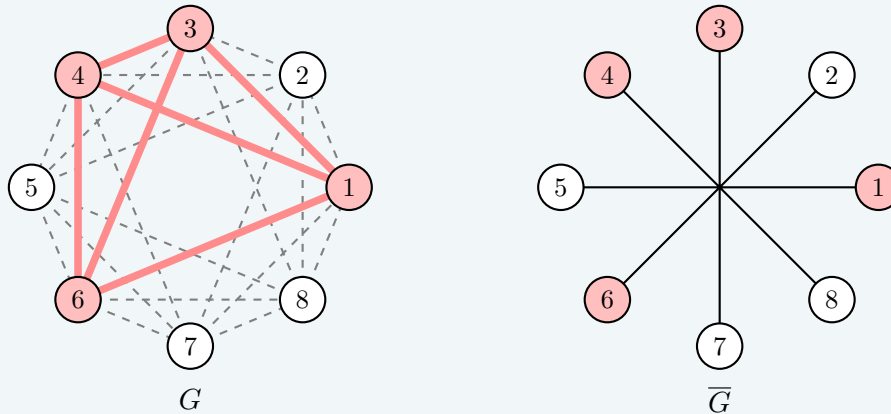
L'insieme dei vertici e l'insieme dei lati sono:

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

Chiamiamo questa istanza del MCP [Esempio 5.1](#): è l'istanza di riferimento del capitolo.

Esempio 5.2: istanza del MCP: un grafo denso con 8 vertici

Consideriamo il grafo $G = (V, E)$ con 8 vertici disposti su una circonferenza, in cui ogni vertice è adiacente a tutti gli altri tranne quello opposto. Ha $m = \binom{8}{2} - 4 = 24$ lati, e il suo complementare ha solo i 4 lati $\{1, 5\}, \{2, 6\}, \{3, 7\}, \{4, 8\}$. Una clique massima $\tilde{K} = \{1, 3, 4, 6\}$ è in rosso.



In $\bar{G}$ i vertici rossi formano un insieme stabile: prendono un vertice da ciascuno dei 4 lati. Chiamiamo questa istanza del MCP [Esempio 5.2](#).

- Il grafo di Petersen del [Capitolo 4 «Problemi di colorazione»](#) non ha triangoli, quindi le sue clique più grandi sono i suoi lati e $\omega(G) = 2$.

5.2.1 Ipotesi

- Consideriamo istanze non banali: il grafo ha almeno un lato e almeno un non-lato. Formalmente:

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

- L'ipotesi si può fare senza perdita di generalità: se il grafo non ha lati, ogni singolo vertice è una clique massima e $\omega(G) = 1$; se il grafo è completo, V stesso è una clique e $\omega(G) = n$.

5.3 Contare le soluzioni, una prima soluzione e i bound

- **Soluzioni candidate.** Le soluzioni candidate sono tutti i sottoinsiemi di V , in numero di 2^n . Per l'Esempio 5.1 abbiamo $2^7 = 128$ sottoinsiemi, e solo 17 di essi sono clique:

numero di vertici	0	1	2	3	totale
clique	1	7	8	1	17

L'unica clique con 3 vertici è $\{1, 2, 3\}$, quindi la soluzione ottima è unica e $\omega(G) = 3$.

- Per l'Esempio 5.2 abbiamo $2^8 = 256$ sottoinsiemi e 81 clique. Una clique prende al più un vertice da ciascuna coppia $\{1, 5\}, \{2, 6\}, \{3, 7\}, \{4, 8\}$: le clique massime ne prendono esattamente uno, quindi sono $2^4 = 16$ e $\omega(G) = 4$.
- **Una prima soluzione, a mano.** Partiamo da un vertice di grado massimo e aggiungiamo, uno alla volta, un vertice adiacente a tutti i vertici già scelti. Nell'Esempio 5.1, il vertice 1 ha grado 3; aggiungiamo il suo vicino 2, e poi 3, che è adiacente a entrambi.
- Nessun altro vertice è adiacente a 1, 2 e 3, quindi ci fermiamo con $K = \{1, 2, 3\}$. È un **bound primale**: $LB = 3 \leq \omega(G)$.

5.3.1 Upper bound semplici

- Il MCP è un problema di massimo: un bound duale è un **upper bound** su $\omega(G)$. Indichiamo con $\deg(v)$ il grado di un vertice e con Δ il grado massimo del grafo.

Osservazione 5.6: bound sul grado

Per ogni grafo $G = (V, E)$ abbiamo $\omega(G) \leq \Delta + 1$.

Dimostrazione. In una clique K , ogni vertice è adiacente agli altri $|K| - 1$ vertici di K , quindi il suo grado è almeno $|K| - 1$. Quindi $|K| - 1 \leq \Delta$ per ogni clique. $\square$

Osservazione 5.7: bound sui lati

Per ogni grafo $G = (V, E)$ abbiamo $\omega(G) (\omega(G) - 1) / 2 \leq m$.

Dimostrazione. Una clique con k vertici contiene come lati tutte le $\binom{k}{2} = k(k-1)/2$ coppie dei suoi vertici, e questi sono lati di G . $\square$

Osservazione 5.8: bound sul conteggio dei gradi

Sia $k^{\max}$ il massimo intero k tale che almeno k vertici di G abbiano grado almeno $k-1$. Allora $\omega(G) \leq k^{\max}$.

Dimostrazione. Una clique con k vertici ha k vertici, ciascuno di grado almeno $k-1$ (dimostrazione del bound sul grado). Quindi $k = \omega(G)$ soddisfa la condizione, e $\omega(G) \leq k^{\max}$. $\square$

Osservazione 5.9: bound di colorazione

Per ogni grafo $G = (V, E)$ abbiamo $\omega(G) \leq \chi(G)$. In particolare, il numero di colori di una qualunque colorazione ammissibile di G è un upper bound su $\omega(G)$.

Dimostrazione. I vertici di una clique sono a due a due adiacenti, quindi ricevono colori a due a due diversi: una colorazione con k colori ha $k \geq |K|$ per ogni clique K . È il bound di clique del Capitolo 4 «Problemi di colorazione», letto dal lato della clique. $\square$

- **Bound sull'Esempio 5.1.** I vertici 1 e 3 hanno grado 3, gli altri cinque hanno grado 2. Il bound sul grado dà $\Delta + 1 = 4$, e anche il bound sui lati dà 4, poiché $4 \cdot 3/2 = 6 \leq 8 < 10 = 5 \cdot 4/2$.
- Solo 2 vertici hanno grado almeno 3, quindi $k = 4$ non soddisfa la condizione del bound sul conteggio dei gradi, mentre $k = 3$ la soddisfa: $k^{\max} = 3$. La colorazione $\{1, 4, 6\}, \{2, 5, 7\}, \{3\}$ dà anch'essa 3. Il gap con $\text{LB} = 3$ è nullo.
- **Bound sull'Esempio 5.2.** Ogni vertice ha grado 6: il bound sul grado e il bound sul conteggio dei gradi danno 7, e il bound sui lati dà 7, poiché $7 \cdot 6/2 = 21 \leq 24 < 28$.
- Le quattro coppie $\{1, 5\}, \{2, 6\}, \{3, 7\}, \{4, 8\}$ sono insiemi stabili, quindi colorano il grafo con 4 colori: il bound di colorazione dà $4 = \omega(G)$. Sui grafi densi i bound basati sui gradi sono deboli, e una buona colorazione è molto migliore.

5.4 Complessità

- La **versione decisionale** del MCP chiede, dati un grafo G e un intero k , se G abbia una clique con k vertici. Karp (1972) la incluse fra i suoi 21 problemi NP-completi, quindi il MCP è NP-difficile. Per la Proposizione 5.5, lo stesso vale per il MSSP.
- Elencare tutti i sottoinsiemi di V e controllare le coppie di ciascuno richiede $O(2^n n^2)$ passi. Per un k fissato, i $\binom{n}{k}$ sottoinsiemi con k vertici si controllano in $O(n^k k^2)$ passi: un polinomio per ogni k fissato, ma l'esponente cresce con la cardinalità della clique.
- Il MCP è anche difficile da approssimare: Håstad (1999) dimostrò che, per ogni $\varepsilon > 0$, nessun algoritmo polinomiale garantisce una clique con almeno $\omega(G)/n^{1-\varepsilon}$ vertici, a meno che ogni problema in NP si possa risolvere in tempo polinomiale randomizzato.

5.5 Varianti pesate

- Sia $\mathbf{w} \in \mathbb{Z}_{\geq 1}^n$ un **vettore dei pesi dei vertici**, con un peso w_v per ogni vertice $v \in V$, e sia $\mathbf{r} \in \mathbb{Z}_{\geq 1}^m$ un **vettore dei pesi dei lati**, con un peso r_{uv} per ogni lato $\{u, v\} \in E$. Per una clique K definiamo

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

dove $E[K]$ è l'insieme dei lati con entrambi gli estremi in K .

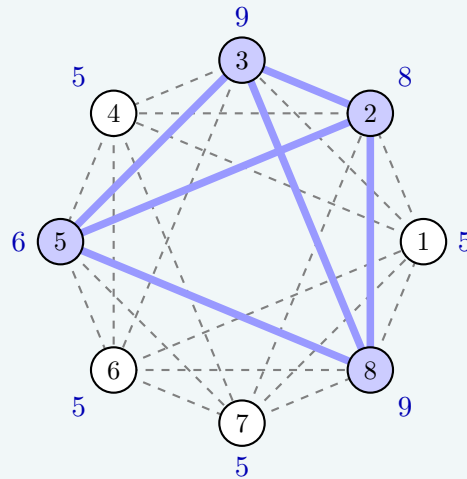
Definizione 5.10: problemi della clique massima pesata

1. Dati un grafo G e un vettore dei pesi dei vertici $\mathbf{w}$, il **problema della clique massima pesata sui vertici** (VWMCP) chiede di trovare una clique K di G con peso totale dei vertici $w(K)$ massimo.
2. Dati un grafo G e un vettore dei pesi dei lati $\mathbf{r}$, il **problema della clique massima pesata sui lati** (EWMCP) chiede di trovare una clique K di G con peso totale dei lati $r(K)$ massimo.

- Con tutti i pesi $w_v = 1$ il primo problema è il MCP. Il problema pesato sui lati premia le clique con molti lati pesanti, e con $r_{uv} = 1$ è di nuovo il MCP, poiché $r(K) = \binom{|K|}{2}$ cresce con $|K|$.

Esempio 5.3: problema della clique massima pesata sui vertici

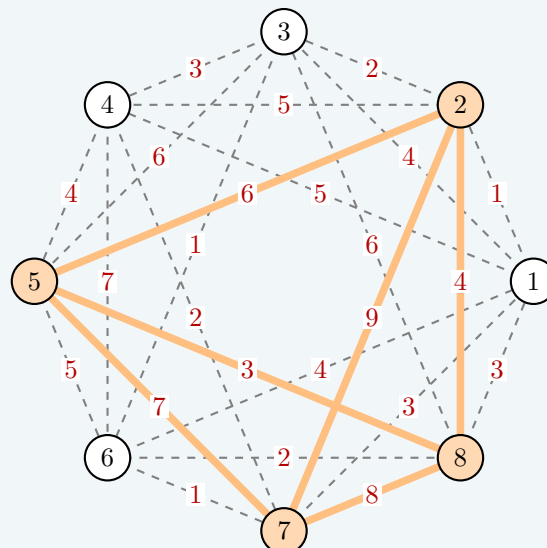
Aggiungiamo al grafo dell'Esempio 5.2 i pesi dei vertici in blu. La clique massima pesata sui vertici è $\{2, 3, 5, 8\}$, con peso $8 + 9 + 6 + 9 = 32$: è unica, e non è la clique massima in rosso dell'Esempio 5.2, il cui peso è $5 + 9 + 5 + 5 = 24$.



Una clique prende al più un vertice da ciascuna coppia $\{1, 5\}$, $\{2, 6\}$, $\{3, 7\}$, $\{4, 8\}$; prendere il più pesante di ciascuna coppia dà $6 + 8 + 9 + 9 = 32$, e questi quattro vertici formano una clique.

Esempio 5.4: problema della clique massima pesata sui lati

Aggiungiamo al grafo dell'Esempio 5.2 i pesi dei lati in rosso. La clique massima pesata sui lati è $\{2, 5, 7, 8\}$, in arancione: i suoi sei lati pesano $6 + 4 + 9 + 7 + 3 + 8 = 37$, e l'enumerazione delle 16 clique massime mostra che è l'unica soluzione ottima.



5.6 Osservazioni

- In una colorazione, ogni classe di colore è un insieme stabile: il MSSP è il problema della classe di colore più grande, e il numero di clique è il bound duale del Capitolo 4 «Problemi di colorazione».
- Applicazioni delle clique e degli insiemi stabili:
 - gruppi di membri a due a due connessi nelle reti sociali;
 - gruppi di attività o di esami a due a due compatibili, nello scheduling e nella pianificazione degli orari;
 - sottostrutture comuni a due molecole, in chimica e in biologia;
 - codici a correzione d'errore, come grandi insiemi di parole a due a due lontane.

5.7 Errori tipici

Attenzione

- **Confondere una clique massimale con una clique massima.** Una clique è massimale quando non le si può aggiungere alcun vertice; è massima quando nessuna clique è più grande. Nell'Esempio 5.1, $\{4, 5\}$ è massimale ma non massima.
- **Costruire il grafo complementare con i lati di G .** I lati di $\overline{G}$ sono esattamente le coppie che *non* sono lati di G : sono $\binom{n}{2} - m$.
- **Usare un lower bound come upper bound.** Una clique trovata a mano dà $LB \leq \omega(G)$; i bound sui gradi e una colorazione danno $UB \geq \omega(G)$. In questo capitolo il problema è di massimo.
- **Credere che il bound di colorazione sia stretto.** Sul ciclo con 5 vertici $\omega(G) = 2$ e $\chi(G) = 3$.

5.8 In sintesi

In sintesi

- Il MCP cerca il più grande insieme di vertici a due a due adiacenti; il suo valore ottimo è il numero di clique $\omega(G)$.
- Una clique di G è un insieme stabile di $\overline{G}$: il MCP su G è il MSSP su $\overline{G}$, e $\omega(G) = \alpha(\overline{G})$.
- Una clique trovata a mano è un bound primale; il bound sul grado, il bound sui lati, il bound sul conteggio dei gradi e il numero di colori di una colorazione sono bound duali.
- Il MCP è NP-difficile e difficile da approssimare; le sue varianti pesate pesano i vertici o i lati.

Quadro riassuntivo delle istanze di riferimento.

istanza (max)	lower bound		upper bound	da
Esempio 5.1 (ciclo con corda)	3 (a mano)	3 (conteggio dei gradi, colorazione)		questo capitolo
Esempio 5.2 (grafo denso)	4 (a mano)	4 (colorazione)		questo capitolo

Problemi di routing

Prerequisiti

- Da questo modulo, [Capitolo 1 «Problemi di ottimizzazione combinatoria»](#): che cos'è un problema di ottimizzazione combinatoria, come esplode il numero delle soluzioni, e le nozioni di bound primale, bound duale e gap.
- Dallo stesso capitolo, l'istanza di riferimento del TSP con 5 città, che questo capitolo studia in dettaglio.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- enunciare il TSP, il TSP asimmetrico e il TSP metrico, e descrivere un'istanza e un giro;
- calcolare la lunghezza di un giro e contare i giri di un'istanza;
- spiegare perché elencare tutti i giri è senza speranza già con 20 città;
- costruire a mano un primo giro con la nearest neighbour rule;
- calcolare un lower bound sulla lunghezza ottima a partire dai due archi più corti di ogni città, e misurare il gap rispetto a un giro.

6.1 Panoramica della famiglia

- I **problemi di routing** di questo capitolo hanno gli stessi dati: un insieme $\mathcal{V} = \{1, 2, \dots, n\}$ di n città e, per ogni coppia di città distinte $i, j \in \mathcal{V}$, una *distanza* $d_{ij} \in \mathbb{Z}_{\geq 1}$.
- Una soluzione è un **giro**, una permutazione σ delle città letta come un percorso chiuso che visita ogni città esattamente una volta; la sua **lunghezza** $L(\sigma)$ è la somma delle distanze percorse.
- L'obiettivo è minimizzare la lunghezza del giro.

Un problema di routing trova un giro più corto attraverso tutte le città. I membri della famiglia differiscono nelle proprietà delle distanze: simmetriche, metriche o asimmetriche.

- La tabella elenca i membri della famiglia, con la sezione che definisce ciascuno di essi.

Problema	Sigla	Distanze	Sezione
commesso viaggiatore	TSP	simmetriche, $d_{ij} = d_{ji}$	6.2
commesso viaggiatore metrico	TSP metrico	simmetriche, con la disuguaglianza triangolare	6.6
commesso viaggiatore asimmetrico	ATSP	d_{ij} può differire da d_{ji}	6.6

6.2 Il problema del commesso viaggiatore

Definizione 6.1: Problema del commesso viaggiatore

Dati:

1. un insieme $\mathcal{V} = \{1, 2, \dots, n\}$ di n città,
2. per ogni coppia di città distinte $i, j \in \mathcal{V}$, una *distanza* $d_{ij} \in \mathbb{Z}_{\geq 1}$, con $d_{ij} = d_{ji}$,

il **problema del commesso viaggiatore** (TSP) chiede di trovare un percorso chiuso che parte da una città, visita ogni altra città esattamente una volta e ritorna alla città di partenza, in modo che la distanza totale percorsa sia minima.

- Un'istanza del TSP è specificata da:
 1. un numero intero:
 - il numero di città $n \in \mathbb{Z}_{\geq 3}$,
 2. una matrice simmetrica di numeri interi di dimensione $n \times n$:
 - $D = (d_{ij})$, che contiene le distanze; gli elementi diagonali non si usano.

- Una **soluzione ammissibile del TSP**, detta **giro**, è una permutazione delle città

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

letta come il percorso chiuso $\sigma_1 \rightarrow \sigma_2 \rightarrow \dots \rightarrow \sigma_n \rightarrow \sigma_1$. La sua **lunghezza** è

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

dove l'ultimo termine è la distanza percorsa per ritornare alla città di partenza.

- Permutazioni diverse possono descrivere lo stesso giro: la città di partenza può essere una qualunque delle n città e, poiché le distanze sono simmetriche, il giro può essere percorso in entrambi i versi.

Data un'istanza del TSP, il TSP si può enunciare formalmente come:

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

- Indichiamo con $z(I)$ la lunghezza di un giro ottimo $\tilde{\sigma}$ dell'istanza I .

6.2.1 Ipotesi

- Facciamo le seguenti ipotesi:
 1. Ci sono almeno tre città (con una o due città c'è un solo percorso). Formalmente:

$$n \geq 3$$
 2. Le distanze sono simmetriche (andare da i a j costa quanto andare da j a i). Formalmente:

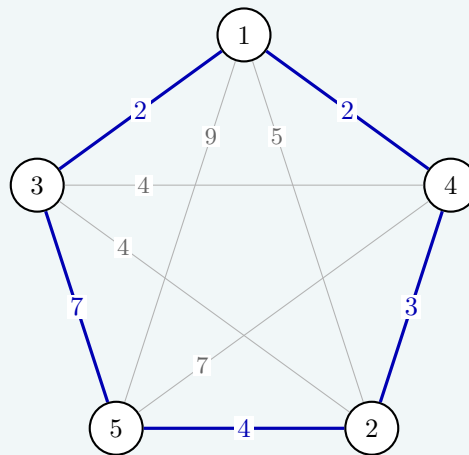
$$d_{ij} = d_{ji}, \quad \forall i, j \in \mathcal{V}, i \neq j$$

Esempio 6.1: Istanza del TSP

Consideriamo un insieme $\mathcal{V} = \{1, 2, 3, 4, 5\}$ di 5 città ($n = 5$), con le seguenti distanze:

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	—

L'istanza si può disegnare come un grafo completo, con un nodo per ogni città e un arco, etichettato con la distanza, per ogni coppia di città. Gli archi in grassetto formano il giro $\sigma = (1, 3, 5, 2, 4)$, e le città sono disegnate nell'ordine del giro:



La sua lunghezza è $L(\sigma) = d_{13} + d_{35} + d_{52} + d_{24} + d_{41} = 2 + 7 + 4 + 3 + 2 = 18$.

6.3 Contare i giri

Proposizione 6.2

Un'istanza del TSP con $n \geq 3$ città ha

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

giri distinti.

- Le permutazioni delle città sono $n!$. Ogni giro è descritto da n permutazioni, una per ogni possibile città di partenza, quindi ci sono $n!/n = (n-1)!$ ordinamenti ciclici delle città.
- Le distanze sono simmetriche, quindi il giro percorso all'indietro ha la stessa lunghezza e non è una nuova soluzione: ogni giro è contato due volte, e dividiamo per 2.

Esempio 6.2: enumerazione dei giri

L'istanza dell'Esempio 6.1 ha $\frac{(5-1)!}{2} = 12$ giri distinti, che si possono enumerare:

giro	L	giro	L	giro	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

Quindi il giro $\sigma = (1, 3, 5, 2, 4)$ è l'unico giro ottimo, e $z(I) = 18$.

- Il numero di giri cresce più velocemente di qualunque funzione esponenziale:

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

- Un calcolatore che valuta 10^9 giri al secondo impiega circa $6.1 \cdot 10^7$ secondi, ossia circa due anni, per elencare i giri di 20 città. L'enumerazione è quindi utile solo come verifica su istanze piccolissime, e gli algoritmi di questo corso devono fare di meglio.

6.4 Un primo giro a mano

- Il modo più semplice per costruire un giro è la **nearest neighbour rule**: si parte da una città e, finché ci sono città non visitate, ci si sposta nella città non visitata più vicina; alla fine si ritorna alla città di partenza. In caso di pareggio prendiamo la città di indice minimo.

Esempio 6.3: nearest neighbour dalla città 1

Sull'istanza dell'Esempio 6.1, partendo dalla città 1:

- da 1 le città più vicine sono 3 e 4, entrambe a distanza 2: andiamo in 3;
- da 3 le città non visitate 2 e 4 sono entrambe a distanza 4: andiamo in 2;
- da 2 la città non visitata più vicina è 4, a distanza 3; poi resta solo 5, a distanza $d_{45} = 7$;
- ritorniamo da 5 a 1 a distanza $d_{51} = 9$.

Il giro è $\sigma = (1, 3, 2, 4, 5)$, di lunghezza $2 + 4 + 3 + 7 + 9 = 25$.

- Gli ultimi archi del giro nearest neighbour sono forzati, e possono essere molto lunghi: qui l'arco finale $\{5, 1\}$ costa 9. Il giro di lunghezza 25 è peggiore di quello ottimo, di lunghezza 18, di 7.

- Dal solo giro nearest neighbour non possiamo dire quanto siamo lontani dall'ottimo: serve un lower bound.

6.5 Un lower bound combinatorio

- Per una città i , siano $a_i \leq b_i$ le due distanze più piccole da i alle altre città.

Osservazione 6.3

Per ogni istanza simmetrica del TSP, la lunghezza ottima soddisfa

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

- In un giro, in ogni città si entra con un arco e se ne esce con un altro, quindi ogni città ha esattamente due archi del giro, lunghi almeno a_i e b_i rispettivamente.
- Sommando su tutte le città, ogni arco del giro è contato due volte, una per ciascuna delle sue città estreme: quindi $2L(\sigma) \geq \sum_i (a_i + b_i)$ per ogni giro σ .
- Le distanze sono intere, quindi lo è anche $L(\sigma)$, e possiamo arrotondare il bound per eccesso.

Esempio 6.4: il bound sull'istanza di riferimento

Per l'istanza dell'Esempio 6.1 le due distanze più piccole di ogni città sono:

città i	1	2	3	4	5	somma
a_i	2	3	2	2	4	
b_i	2	4	4	3	7	
$a_i + b_i$	4	7	6	5	11	33

Quindi $z(I) \geq \lceil 33/2 \rceil = 17$. Insieme al giro nearest neighbour otteniamo

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

un gap assoluto di 8, ossia un gap relativo di $\frac{8}{17} \approx 47.1\%$. Il giro $(1, 3, 5, 2, 4)$ di lunghezza 18 ne chiude la maggior parte, e solo l'enumerazione dei 12 giri dimostra che 18 è ottimo.

6.6 Varianti: TSP metrico e TSP asimmetrico

6.6.1 Il problema del commesso viaggiatore metrico

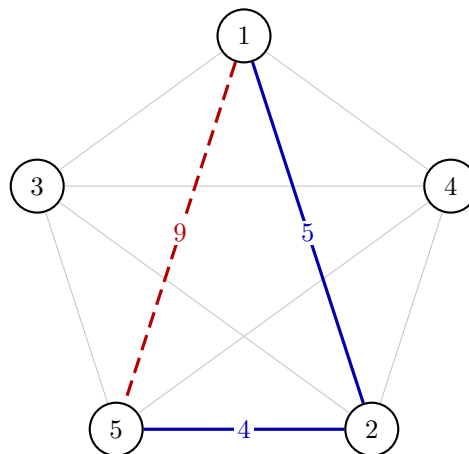
Definizione 6.4: TSP metrico

Un'istanza del TSP è **metrica** se le sue distanze soddisfano la **disuguaglianza triangolare**

$$d_{ij} \leq d_{ik} + d_{kj} \quad \text{per ogni terna di città distinte } i, j, k \in \mathcal{V}.$$

Il **problema del commesso viaggiatore metrico** (TSP metrico) è il TSP ristretto alle istanze metriche.

- La disuguaglianza triangolare dice che andare direttamente da i a j non è mai più lungo che passare per una terza città k .
- L'istanza dell'Esempio 6.1 è metrica: tutte le $5 \cdot 4 \cdot 3 = 60$ disuguaglianze triangolari valgono, alcune all'uguaglianza, per esempio $d_{15} = 9 = d_{12} + d_{25} = 5 + 4$. Nel disegno, l'arco diretto $\{1, 5\}$ (tratteggiato) è lungo quanto il cammino da 1 a 5 che passa per la città 2.



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

- Le istanze ottenute da punti nel piano, con d_{ij} uguale alla distanza euclidea fra i punti, sono metriche (quando le distanze non sono arrotondate).
- Diversi algoritmi euristici per il TSP si comportano bene, e alcuni di essi hanno una garanzia di prestazione, solo sulle istanze metriche.

6.6.2 Il problema del commesso viaggiatore asimmetrico

Definizione 6.5: Problema del commesso viaggiatore asimmetrico

Dati:

1. un insieme $\mathcal{V} = \{1, 2, \dots, n\}$ di n città,
2. per ogni coppia ordinata di città distinte $i, j \in \mathcal{V}$, una distanza $d_{ij} \in \mathbb{Z}_{\geq 1}$ dalla città i alla città j , dove d_{ij} può differire da d_{ji} ,

il **problema del commesso viaggiatore asimmetrico** (ATSP) chiede di trovare un percorso chiuso che parte da una città, visita ogni altra città esattamente una volta e ritorna alla città di partenza, in modo che la distanza totale percorsa sia minima.

- Le distanze asimmetriche nascono, per esempio, quando si percorrono strade a senso unico o in salita e in discesa.
- La definizione di giro e della sua lunghezza $L(\sigma)$ non cambia, ma ora il verso conta: un giro e lo stesso giro percorso all'indietro sono soluzioni diverse.
- Il TSP è il caso particolare dell'ATSP in cui $d_{ij} = d_{ji}$ per ogni i, j .
- Un'istanza dell'ATSP ha $\frac{n!}{n} = (n-1)!$ giri distinti, il doppio del conteggio della [Proposizione 6.2](#): la città di partenza continua a non cambiare il giro, ma il verso sì, quindi non dividiamo per 2.
- L'istanza dell'[Esempio 6.1](#), letta come istanza dell'ATSP, ha $4! = 24$ giri invece di 12.

6.7 Osservazioni

- L'istanza più grande risolta all'ottimo, pla85900, ha 85 900 punti che vengono da un problema di progettazione di chip. È stata risolta nel 2005–2006 con il codice Concorde; il giro ottimo, di lunghezza 142 382 641, è stato certificato nel 2009 da Applegate, Bixby, Chvátal, Cook, Espinoza, Goycoolea e Helsgaun.
- Sulle mappe stradali di paesi reali, il record del 2025 è l'istanza korea81998, con 81 998 bar della Corea del Sud.
- Il TSP è fortemente NP-difficile ([Capitolo 1 «Problemi di ottimizzazione combinatoria»](#)). Christofides (1976) ha dato un algoritmo di approssimazione con rapporto $\frac{3}{2}$ per il TSP metrico.

6.8 Errori tipici

Attenzione

- **Contare $n!$ giri.** La città di partenza non cambia il giro e, nel caso simmetrico, nemmeno il verso: il numero di giri è $(n-1)!/2$.
- **Dimenticare l'arco di ritorno.** La lunghezza di un giro comprende l'ultima distanza $d_{\sigma_n \sigma_1}$, dall'ultima città di nuovo alla prima.
- **Scambiare due cicli per un giro.** Due cicli disgiunti che insieme visitano ogni città una volta non sono un giro: un giro è un unico ciclo attraverso tutte le città.
- **Leggere il bound come un giro.** Il valore 17 del bound dei due archi più corti non è la lunghezza di un giro; dice solo che nessun giro è più corto.
- **Usare la disuguaglianza triangolare senza verificarla.** Le garanzie degli algoritmi euristici valgono solo per le istanze metriche.

6.9 In sintesi

In sintesi

- Il TSP chiede un percorso chiuso più corto attraverso tutte le città; una soluzione è un ordinamento ciclico delle città, detto giro.
- Un'istanza simmetrica con n città ha $(n-1)!/2$ giri, una asimmetrica $(n-1)!$; elencarli è possibile solo per n molto piccolo.
- La nearest neighbour rule dà rapidamente un giro, senza garanzie; i due archi più corti di ogni città danno un lower bound sulla lunghezza ottima.
- Il TSP metrico, con la disuguaglianza triangolare, è il caso in cui esistono garanzie di approssimazione.

Cruscotto dell'istanza del TSP (minimizzazione: il lower bound è un bound duale, l'upper bound è la lunghezza di un giro):

Capitolo	Lower bound	Upper bound	Da
Questo capitolo	17	25	due archi più corti; nearest neighbour dalla città 1
Questo capitolo, enumerazione	18	18	i 12 giri

Problemi di scheduling

Prerequisiti

- Da questo modulo, [Capitolo 1 «Problemi di ottimizzazione combinatoria»](#): che cos'è un problema di ottimizzazione combinatoria, il numero di soluzioni candidate, i bound primali e duali e le partizioni di un insieme (numeri di Bell).
- Da questo modulo, [Capitolo 3 «Problemi di packing»](#): il problema del bin packing.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- classificare un problema di scheduling con la notazione a tre campi ed enunciare il problema $P || C_{\max}$;
- descrivere uno schedule tramite gli insiemi di lavori delle macchine e calcolare carichi e makespan;
- contare gli assegnamenti e gli schedule distinti di un'istanza;
- costruire a mano un primo schedule, assegnando ogni lavoro alla macchina meno carica;
- calcolare un lower bound sul makespan ottimo e usarlo per dimostrare che uno schedule è ottimo.

7.1 Panoramica della famiglia

- I **problemi di scheduling** di questo capitolo hanno gli stessi dati: un insieme $\mathcal{J} = \{1, 2, \dots, n\}$ di n lavori, m macchine parallele e, per ogni lavoro, un *tempo di processamento*.
- Una soluzione assegna ogni lavoro a una macchina: i lavori J_h della macchina h sono processati uno dopo l'altro, e la macchina li completa all'istante L_h , il suo **carico**.
- L'obiettivo è minimizzare il **makespan** $C_{\max} = \max_h L_h$, l'istante di completamento dell'ultimo lavoro.

Un problema di scheduling su macchine parallele ripartisce i lavori fra le macchine in modo che l'ultima macchina finisca il prima possibile. I membri della famiglia differiscono per l'ambiente delle macchine, descritto dalla notazione a tre campi.

- La tabella elenca i membri della famiglia, con la sezione che definisce ciascuno di essi.

Problema	Notazione	Macchine	Sezione
macchine parallele identiche	$P \parallel C_{\max}$	ogni lavoro richiede t_j su ogni macchina	7.2
macchine con velocità diverse	$Q \parallel C_{\max}$	la macchina h ha una propria velocità	7.1.1
macchine parallele scorrelate	$R \parallel C_{\max}$	il lavoro j richiede t_{jh} sulla macchina h	7.1.1

7.1.1 La notazione a tre campi

- I problemi di scheduling si classificano con la **notazione a tre campi** $\alpha | \beta | \gamma$ di Graham, Lawler, Lenstra e Rinnooy Kan, dove:
 1. α descrive l'**ambiente delle macchine**: 1 per una sola macchina, P per macchine parallele identiche, Q per macchine parallele con velocità diverse, R per macchine parallele scorrelate;
 2. β descrive le **caratteristiche dei lavori**, come le date di rilascio r_j o i vincoli di precedenza *prec*; un campo vuoto indica che non ce ne sono;
 3. γ descrive la **funzione obiettivo**, come il makespan $C_{\max}$ o la somma dei tempi di completamento $\sum_j C_j$.
- In questa notazione, $P \parallel C_{\max}$ è il problema di schedulare lavori senza caratteristiche aggiuntive su macchine parallele identiche, minimizzando il makespan.

7.2 Scheduling su macchine parallele identiche

Definizione 7.1: Scheduling su macchine parallele identiche

Dati:

1. un insieme di m macchine identiche,
2. un insieme $\mathcal{J} = \{1, 2, \dots, n\}$ di n lavori,
3. per ogni lavoro $j \in \mathcal{J}$, un tempo di processamento $t_j \in \mathbb{Z}_{\geq 1}$,

il problema $P \parallel C_{\max}$ chiede di assegnare ogni lavoro a una macchina, in modo da minimizzare il tempo necessario per completare tutti i lavori, cioè il massimo tempo di processamento totale assegnato a una macchina.

- Il nome $P \parallel C_{\max}$ deriva dalla notazione a tre campi dei problemi di scheduling (Sezione 7.1.1).
- Un'istanza di $P \parallel C_{\max}$ è specificata da:
 1. due numeri interi:
 - il numero di macchine $m \in \mathbb{Z}_{\geq 1}$,
 - il numero di lavori $n \in \mathbb{Z}_{\geq 1}$,
 2. un vettore di numeri interi di lunghezza n :
 - $\mathbf{t} \in \mathbb{Z}_{\geq 1}^n$, che contiene i tempi di processamento.
- Una **soluzione ammissibile** di $P \parallel C_{\max}$ è un assegnamento dei lavori alle macchine, che memorizziamo in un vettore $\text{mach} = (\text{mach}_1, \text{mach}_2, \dots, \text{mach}_n)$:

$$\text{mach}_j = h \iff \text{il lavoro } j \text{ è assegnato alla macchina } h.$$

L'insieme dei lavori assegnati alla macchina h è $J_h = \{j \in \mathcal{J} : \text{mach}_j = h\}$, e il suo **carico** è

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

Ogni macchina processa i suoi lavori uno dopo l'altro, senza tempi morti, quindi la macchina h completa i suoi lavori all'istante L_h . Il **makespan** della soluzione è l'istante di completamento dell'ultimo lavoro,

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

- L'ordine in cui una macchina processa i suoi lavori non cambia il suo carico: una soluzione è descritta completamente dagli insiemi $J_1, J_2, \dots, J_m$, che formano una partizione di $\mathcal{J}$ (alcuni di essi possono essere vuoti).

Data un'istanza di $P \parallel C_{\max}$, il problema si può enunciare formalmente come:

$$\min \left\{ \max_{h \in \{1, \dots, m\}} \sum_{j \in J_h} t_j : J_1, J_2, \dots, J_m \text{ è una partizione di } \mathcal{J} \right\}. \quad (7.1)$$

- Indichiamo con $z(I)$ il makespan ottimo dell'istanza I .

7.2.1 Ipotesi

- Facciamo la seguente ipotesi:
 1. Ci sono più lavori che macchine (altrimenti ogni lavoro ha una macchina tutta per sé, e il makespan ottimo è $\max_j t_j$). Formalmente:

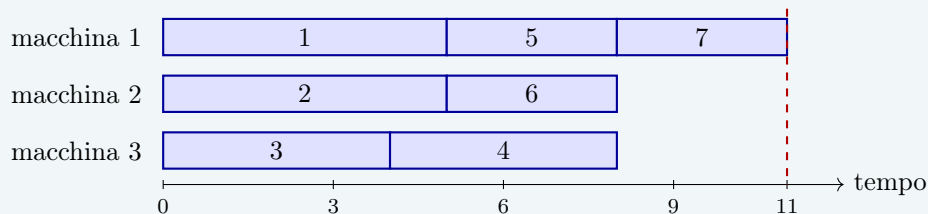
$$n > m$$

Esempio 7.1: istanza di $P \parallel C_{\max}$

Consideriamo $m = 3$ macchine e un insieme $\mathcal{J} = \{1, 2, \dots, 7\}$ di 7 lavori ($n = 7$), con i seguenti tempi di processamento:

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

Chiamiamo questa istanza [Esempio 7.1](#): è l'istanza di riferimento del capitolo. L'assegnamento $\text{mach} = (1, 2, 3, 3, 1, 2, 1)$ dà $J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$, con carichi $L_1 = 11$, $L_2 = 8$, $L_3 = 8$ e makespan 11:



7.3 Contare le soluzioni

- Ogni lavoro può essere assegnato a una qualunque delle m macchine, indipendentemente dagli altri: ci sono m^n assegnamenti mach.
- Le macchine sono identiche, quindi due assegnamenti che differiscono solo per i nomi delle macchine descrivono lo stesso schedule. Gli schedule distinti sono le partizioni di $\mathcal{J}$ in al più m insiemi non vuoti, e il loro numero è

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

dove $S(n, k)$ è il numero di Stirling di seconda specie, cioè il numero di partizioni di n elementi in esattamente k insiemi non vuoti.

Esempio 7.2: contare gli schedule

Per l'istanza dell'Esempio 7.1 ci sono $3^7 = 2187$ assegnamenti. I numeri di Stirling sono $S(7, 1) = 1$, $S(7, 2) = 63$ e $S(7, 3) = 301$, quindi gli schedule distinti sono

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

Per enumerazione, gli schedule con il makespan minimo, 9, sono esattamente due:

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

che corrispondono a $3! \cdot 2 = 12$ dei 2187 assegnamenti. Lo schedule ottimo quindi non è unico.

- La crescita è esponenziale in n : con 3 macchine, 20 lavori hanno $3^{20} \approx 3.5 \cdot 10^9$ assegnamenti, e 30 lavori ne hanno $3^{30} \approx 2.1 \cdot 10^{14}$.

7.4 Un primo schedule a mano

- Un modo semplice per costruire uno schedule è prendere i lavori uno alla volta, nell'ordine $1, 2, \dots, n$, e assegnare ogni lavoro alla macchina che in quel momento ha il carico minimo; in caso di pareggio prendiamo la macchina con l'indice minimo.

Esempio 7.3: assegnare ogni lavoro alla macchina meno carica

Sull'istanza dell'Esempio 7.1 i carichi delle tre macchine evolvono come segue:

lavoro j (t_j)	macchina scelta	carichi (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)

Otteniamo l'assegnamento $\text{mach} = (1, 2, 3, 3, 1, 2, 1)$ dell'esempio, con makespan 11.

- Quando arriva l'ultimo lavoro tutte le macchine hanno carico 8: qualunque macchina lo riceva, il carico massimo diventa $8 + 3 = 11$.
- Non sappiamo ancora se 11 è lontano dal makespan ottimo: ci serve un lower bound.

7.5 Un lower bound combinatorio

Osservazione 7.2

Per ogni istanza di $P \parallel C_{\max}$, il makespan ottimo $z(I)$ soddisfa

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

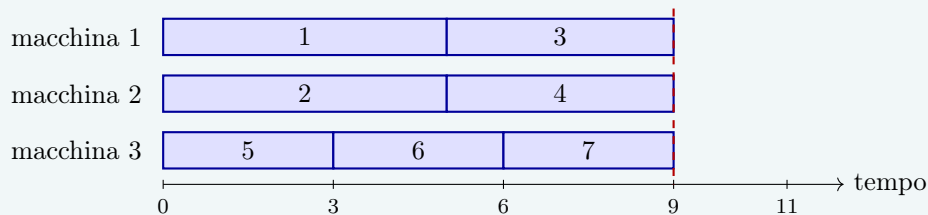
- La macchina che processa il lavoro più lungo ha carico almeno $\max_j t_j$.
- Gli m carichi sommano a $\sum_j t_j$, quindi il maggiore è almeno pari alla loro media $\frac{1}{m} \sum_j t_j$ e, essendo intero, almeno pari al suo arrotondamento per eccesso.

Esempio 7.4: il bound sull'istanza di riferimento

Sull'istanza dell'Esempio 7.1 il bound è $\max\{5, \lceil 27/3 \rceil\} = 9$. Insieme alle schedule della Sezione 7.4,

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

L'assegnamento $\text{mach} = (1, 2, 1, 2, 3, 3, 3)$ dà $J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$, tutti con carico 9, e makespan 9:



Raggiunge il lower bound, quindi è ottimo: $z(I) = 9$. Le tre macchine finiscono insieme, e non resta alcun tempo morto prima del makespan.

- Il bound non è sempre raggiunto. Con $m = 2$ macchine e tre lavori con tempi di processamento $(4, 3, 3)$ dà $\max\{4, \lceil 10/2 \rceil\} = 5$, ma lo schedule migliore è $\{1\}, \{2, 3\}$, con makespan 6.

7.6 Relazione con il problema del bin packing

- Dato un valore T , esiste uno schedule con makespan al più T se e solo se i lavori, visti come oggetti di peso t_j , possono essere inseriti in m contenitori di capacità T : le macchine sono i contenitori e i carichi sono i pesi totali dei pattern di riempimento.
- I due problemi fissano e minimizzano quantità opposte:

	numero di contenitori / macchine	capacità / makespan
BPP	minimizzato	fissato (c)
$P \parallel C_{\max}$	fissato (m)	minimizzato

- Per questo i due problemi condividono molte idee: diversi algoritmi euristici per uno dei due hanno un corrispondente per l'altro.

7.7 Osservazioni

- Nel 1966 R. L. Graham ha dimostrato che qualunque lista di lavori, processata in modo greedy, dà un makespan al più $2 - 1/m$ volte quello ottimo.
- Nel 1969 ha dimostrato il rapporto migliore $4/3 - 1/(3m)$ per la lista ordinata per tempo di processamento non crescente (la regola detta LPT).

7.8 Errori tipici

Attenzione

- **Contare m^n schedule diversi.** Le macchine sono identiche: rinominarle dà lo stesso schedule, quindi gli schedule distinti sono le partizioni dei lavori in al più m insiemi.
- **Confondere makespan e carico totale.** Il makespan è il carico massimo di una macchina, non la somma dei tempi di processamento e non il carico medio.
- **Dimenticare l'arrotondamento per eccesso.** I tempi di processamento sono interi, e quindi anche il makespan: il carico medio $27/3$ si arrotonda per eccesso quando non è intero.
- **Supporre che il bound sia raggiunto.** Il bound della Sezione 7.5 è un bound duale: uno schedule con quel makespan può non esistere, come mostra l'istanza $(4, 3, 3)$ su

due macchine.

- **Credere che lo schedule ottimo sia unico.** L'istanza dell'Esempio 7.1 ha due schedule ottimi e dodici assegnamenti ottimi.

7.9 In sintesi

In sintesi

- Il problema $P || C_{\max}$ chiede di assegnare lavori a macchine identiche minimizzando il carico massimo, il makespan; una soluzione è una partizione dei lavori in al più m insiemi.
- Ci sono m^n assegnamenti, ma solo $\sum_{k=1}^m S(n, k)$ schedule distinti.
- Dare ogni lavoro alla macchina meno carica costruisce rapidamente uno schedule; il massimo fra il lavoro più lungo e il carico medio arrotondato è un lower bound sul makespan ottimo.
- Quando uno schedule raggiunge il lower bound è ottimo; altrimenti il gap resta aperto.
- Il problema è lo speculare del bin packing: qui il numero di macchine è fissato e la capacità è minimizzata.

Quadro dei bound dell'istanza di $P || C_{\max}$ (minimizzazione: il lower bound è un bound duale, l'upper bound è il makespan di uno schedule):

Capitolo	Lower bound	Upper bound	Da
Questo capitolo	9	11	lavoro più lungo e carico medio; macchina meno carica
Questo capitolo, schedule migliorato	9	9	$J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$

Problemi di lot sizing

Prerequisiti

- Da questo modulo, [Capitolo 1 «Problemi di ottimizzazione combinatoria»](#): che cos'è un problema di ottimizzazione combinatoria, il numero di soluzioni candidate, e i bound primali e duali.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- descrivere un piano di produzione tramite le sue quantità, scorte e setup, e calcolarne il costo;
- enunciare il problema del lot sizing capacitato e indicarne le principali varianti;
- stabilire se un piano è ammissibile, e contare le scelte di setup ammissibili di un'istanza;
- costruire a mano un primo piano, accorpendo le domande di periodi consecutivi;
- calcolare un lower bound sul costo dal numero di setup richiesto dalla domanda totale.

8.1 Panoramica della famiglia

- I **problemi di lot sizing** hanno gli stessi dati: un insieme $T = \{1, 2, \dots, m\}$ di m periodi di tempo e, per ogni periodo $t \in T$, una *domanda* d_t , una *capacità* produttiva e tre costi: il costo di setup sc_t , il costo variabile di produzione vc_t e il costo di mantenimento a scorta hc_t .
- Una soluzione è un **piano di produzione**: la quantità prodotta in ogni periodo, che determina la scorta alla fine di ciascun periodo e i periodi con un setup.
- L'obiettivo è soddisfare tutte le domande al minimo costo totale di setup, produzione e mantenimento a scorta.

Un problema di lot sizing bilancia i costi di setup, pagati in ogni periodo con produzione, e i costi di mantenimento della scorta. I membri della famiglia differiscono per le capacità, per il numero di prodotti e per il modo in cui la domanda e la capacità si possono modificare.

- La tabella elenca i membri della famiglia, con la sezione che definisce ciascuno di essi.

Problema	Sigla	Differenza rispetto al CLS	Sezione
problema del lot sizing capacitato	CLS	un prodotto, capacità cap_t	8.2
problema del lot sizing senza capacità	ULS	nessuna capacità vincolante	8.6
più prodotti e tempi di setup	MLS	prodotti che condividono il tempo di ogni periodo	8.6
scorta iniziale	LS-SI	la scorta iniziale è una decisione	8.6
straordinario	LS-OT	capacità aggiuntiva a un costo più alto	8.6
domanda non soddisfatta	LS-UD	parte della domanda persa con una penalità	8.6

8.2 Il problema del lot sizing capacitato

Definizione 8.1: problema del lot sizing capacitato

Dati:

1. un insieme $T = \{1, 2, \dots, m\}$ di m periodi di tempo,
2. per ogni periodo $t \in T$, una domanda $d_t \in \mathbb{Z}_{\geq 1}$ e una capacità $cap_t \in \mathbb{Z}_{\geq 1}$, entrambe misurate in unità di un prodotto,
3. per ogni periodo $t \in T$, tre costi interi non negativi: il costo di setup sc_t , il costo variabile di produzione vc_t per unità prodotta e il costo di mantenimento a scorta hc_t per unità tenuta in scorta alla fine del periodo,

il **problema del lot sizing capacitato** (CLS) chiede di decidere quanto produrre in ogni periodo, in modo che ogni domanda sia soddisfatta con ciò che si produce e ciò che è in scorta, le capacità siano rispettate, e il costo totale di setup, produzione e mantenimento a scorta sia minimo.

- Un'istanza del CLS è specificata da:
 1. un numero intero:

il numero di periodi $m \in \mathbb{Z}_{\geq 1}$,
 2. cinque vettori di numeri interi di lunghezza m :

$d, cap \in \mathbb{Z}_{\geq 1}^m$, che contengono domande e capacità,
 $sc, vc, hc \in \mathbb{Z}_{\geq 0}^m$, che contengono i costi.

- Una **soluzione ammissibile del CLS** è un **piano di produzione**, ossia un vettore di quantità intere

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

dove x_t è la quantità prodotta nel periodo t . La scorta rimasta alla fine del periodo t è

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

perché la scorta del periodo $t-1$ più la produzione del periodo t deve coprire la domanda del periodo t , e ciò che resta è la scorta del periodo t . Il piano è **ammissibile** se

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

- C'è un **setup** nel periodo t quando $x_t > 0$: in questo caso scriviamo $y_t = 1$, e $y_t = 0$ altrimenti. Il **costo** di un piano è

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

Data un'istanza del CLS, il CLS si può enunciare formalmente come:

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

- Indichiamo con $z(I)$ il costo di un piano ottimo $\tilde{\mathbf{x}}$ dell'istanza I .

Esempio 8.1: istanza del CLS

Consideriamo $m = 5$ periodi, con i seguenti dati, tutti interi:

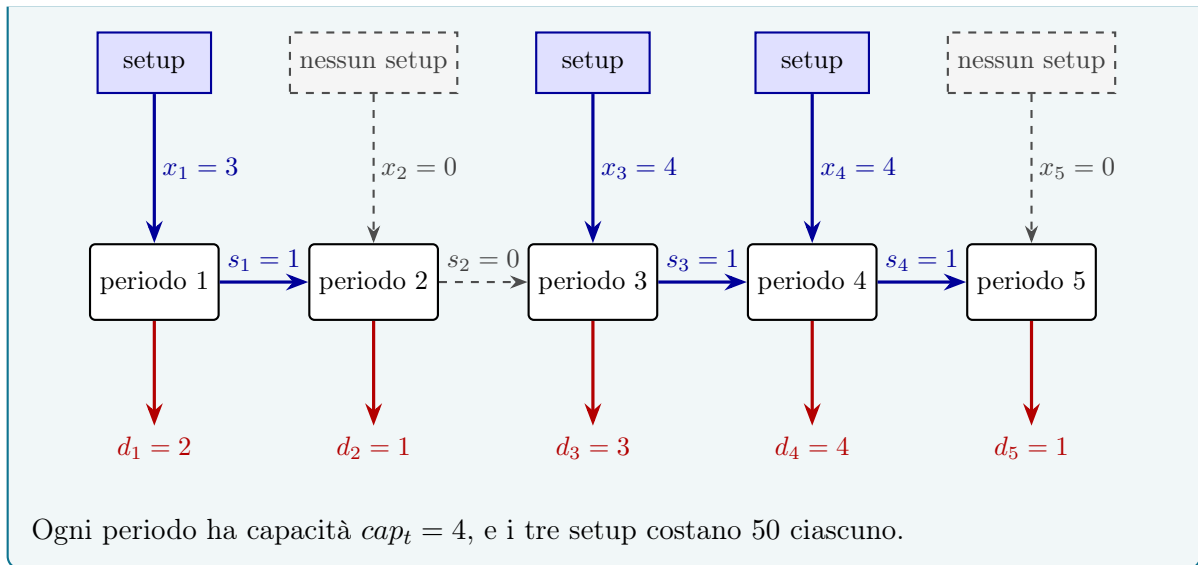
periodo t	1	2	3	4	5
domanda d_t	2	1	3	4	1
capacità cap_t	4	4	4	4	4
costo di setup sc_t	50	50	50	50	50
costo di produzione vc_t	0	0	0	0	0
costo di mantenimento hc_t	10	10	10	10	10

La domanda totale è $2+1+3+4+1 = 11$. Chiamiamo questa istanza del CLS [Esempio 8.1](#): è l'istanza di riferimento del capitolo.

Il piano $\mathbf{x} = (3, 0, 4, 4, 0)$ è ammissibile: le scorte sono $s = (1, 0, 1, 1, 0)$, tutte non negative, con $s_5 = 0$, e ogni quantità è al più 4. Ci sono 3 setup, nei periodi 1, 3 e 4:

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

Il piano è disegnato qui sotto, periodo per periodo da sinistra a destra: la produzione x_t entra in un periodo dall'alto, attraverso un setup (linea continua) oppure non entra affatto (tratteggio); la scorta s_t passa al periodo successivo; la domanda d_t esce dal periodo verso il basso.



8.2.1 Ipotesi

- Facciamo le seguenti ipotesi:
 - La scorta è vuota all'inizio e alla fine dell'orizzonte. Lasciare scorta alla fine aggiunge solo costo, quindi nessun piano ottimo ne ha bisogno; una scorta iniziale è una variante del problema (Sezione 8.6).
 - Le capacità permettono di soddisfare le domande, periodo dopo periodo (altrimenti non esiste alcun piano ammissibile). Formalmente:

$$\sum_{u=1}^t cap_u \geq \sum_{u=1}^t d_u, \quad \forall t \in T$$

- Le quantità sono intere. Con dati interi, una volta fissati i periodi con un setup, ciò che resta è un problema di flusso di costo minimo, che ha una soluzione ottima intera: restringere le quantità agli interi non cambia il costo ottimo.

8.3 Contare le soluzioni

- La parte combinatoria della decisione è la scelta dei periodi con un setup: ci sono 2^m sottoinsiemi di periodi. Non tutti sono ammissibili, perché i periodi con un setup devono avere capacità sufficiente per produrre tutto.
- Per una data scelta dei periodi di setup possono esistere molti piani di produzione, poiché le quantità si possono distribuire in molti modi: il numero di piani ammissibili è maggiore del numero di scelte di setup ammissibili.

Esempio 8.2: contare i piani

Per l'istanza dell'Esempio 8.1 ci sono $2^5 = 32$ sottoinsiemi di periodi. Sia k_t il numero di setup nei periodi $1, \dots, t$: le domande cumulate 2, 3, 6, 10, 11 richiedono, con capacità 4,

almeno un setup nei periodi 1–2, due nei periodi 1–3 e tre nei periodi 1–4. Poiché $d_1 > 0$ c'è un setup nel periodo 1, e nei periodi 2, 3, 4 servono almeno due setup, di cui uno nei periodi 2 o 3:

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

Il periodo 5 è libero, quindi le scelte di setup ammissibili sono $4 \cdot 2 = 8$ su 32.

Per enumerazione, a queste scelte di setup corrispondono 94 piani ammissibili. I costi di sei di essi sono:

piano	costo	piano	costo	piano	costo
(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

Il piano (3, 0, 4, 4, 0) è l'unico piano ottimo, e $z(I) = 180$.

- Senza le capacità la stessa istanza ha $2^{m-1} = 16$ piani che producono solo quando la scorta è vuota, e il costo ottimo è 170: qui la capacità costa 10.
- Il piano ottimo (3, 0, 4, 4, 0) produce nel periodo 4 con una scorta di 1 unità, quindi la proprietà di Wagner e Whitin non vale quando le capacità sono vincolanti.
- Il numero di scelte di setup cresce come 2^m : con $m = 52$ settimane supera $4 \cdot 10^{15}$.

8.4 Un primo piano a mano

- Il piano ammissibile più semplice è il **lot-for-lot**: in ogni periodo produciamo esattamente la domanda di quel periodo. Non c'è scorta, e ci sono m setup; è ammissibile quando ogni domanda rientra nella capacità del suo periodo.
- Un piano si migliora **accorpando**: la domanda di un periodo si produce in un periodo precedente, il che risparmia un setup e costa la scorta delle unità anticipate.

Esempio 8.3: dal lot-for-lot a un piano migliore

Sull'istanza dell'Esempio 8.1:

1. il piano lot-for-lot (2, 1, 3, 4, 1) ha 5 setup e nessuna scorta: costo $5 \cdot 50 = 250$;
2. produrre la domanda del periodo 2 nel periodo 1 risparmia un setup (-50) e tiene 1 unità in scorta per un periodo ($+10$): il piano (3, 0, 3, 4, 1) costa 210;
3. il periodo 4 non può produrre anche la domanda del periodo 5, perché $4+1 > cap_4 = 4$; ma il periodo 3 può anticipare un'unità del periodo 4, così che il periodo 4 abbia spazio per la domanda del periodo 5: il piano (3, 0, 4, 4, 0) risparmia un setup (-50) e tiene in scorta due unità in più ($+20$).

Il costo è $210 - 50 + 20 = 180$.

- Abbiamo trovato a mano il piano di costo 180, ma non sappiamo ancora dire se esista un piano più economico: ci serve un lower bound.

8.5 Un lower bound combinatorio

- Sia $D = \sum_{t \in T} d_t$ la domanda totale. Sia k il minimo numero di periodi le cui capacità sommano almeno a D , che si trova prendendo i periodi con le capacità più grandi.

Osservazione 8.2

Per ogni istanza del CLS, ogni piano ammissibile ha almeno k setup, e

$$z(I) \geq \text{la somma dei } k \text{ costi di setup più piccoli.}$$

- Il piano deve produrre D unità nei periodi con un setup, e un periodo senza setup non produce nulla: le capacità dei periodi con un setup sommano almeno a D , quindi questi periodi sono almeno k .
- I costi di produzione e di mantenimento a scorta sono non negativi: trascurandoli, e pagando solo k setup, i più economici, otteniamo un bound sul costo di ogni piano.

Esempio 8.4: il bound sull'istanza di riferimento

Per l'istanza dell'Esempio 8.1 abbiamo $D = 11$ e tutte le capacità uguali a 4, quindi $k = \lceil 11/4 \rceil = 3$. Tutti i costi di setup valgono 50, quindi

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

Insieme ai piani della sezione precedente otteniamo

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

un gap assoluto di 30, ossia un gap relativo di $\frac{30}{150} = 20\%$. L'enumerazione dei 94 piani dimostra che 180 è ottimo.

8.6 Varianti

- Ogni variante mantiene i dati del CLS (periodi, domande, costi) e cambia una caratteristica del problema. Le definizioni qui sotto riportano solo ciò che cambia.

8.6.1 Il problema del lot sizing senza capacità

Definizione 8.3: problema del lot sizing senza capacità

Il **problema del lot sizing senza capacità** (ULS) è il CLS in cui le capacità non sono vincolanti, per esempio $cap_t \geq \sum_{u \in T} d_u$ per ogni $t \in T$.

- Il ULS si risolve in tempo polinomiale con la programmazione dinamica.

8.6.2 Più prodotti e tempi di setup

Definizione 8.4: lot sizing con più prodotti e tempi di setup

Nel **problema del lot sizing con più prodotti e tempi di setup** (MLS) un insieme P di prodotti condivide la risorsa di ogni periodo: d_{pt} è la domanda del prodotto $p \in P$ nel periodo t , e cap_t è il tempo disponibile nel periodo t . Produrre un'unità di p richiede vt_{pt} unità di tempo, e un setup di p costa sc_{pt} e richiede st_{pt} unità di tempo.

8.6.3 Scorta iniziale

Definizione 8.5: lot sizing con scorta iniziale

Nel **problema del lot sizing con scorta iniziale** (LS-SI) la quantità si_p del prodotto p disponibile all'inizio dell'orizzonte è una decisione, acquistata al costo fc_p per unità.

8.6.4 Straordinario

Definizione 8.6: lot sizing con straordinario

Nel **problema del lot sizing con straordinario** (LS-OT) la capacità non è fissa: il tempo di lavoro in orario normale costa cw_t per unità fino a cap_t^N , e lo straordinario costa co_t per unità fino a cap_t^O .

8.6.5 Domanda non soddisfatta

Definizione 8.7: lot sizing con domanda non soddisfatta

Nel **problema del lot sizing con domanda non soddisfatta** (LS-UD) una parte della domanda può restare non soddisfatta, con una penalità cu_{pt} per unità.

8.7 Osservazioni

- Nel 1958 Wagner e Whitin proposero un algoritmo di programmazione dinamica per il caso senza capacità e con domanda variabile nel tempo. L'algoritmo si basa su una proprietà dei piani ottimi: si produce solo quando la scorta è vuota.
- Con le capacità la proprietà non vale più, e il problema è NP-difficile ([Capitolo 1 «Problemi di ottimizzazione combinatoria»](#)).

8.8 Errori tipici

Attenzione

- **Produrre in ritardo.** La produzione di un periodo non può servire la domanda di un periodo precedente: la scorta $s_t(\mathbf{x})$ deve essere non negativa in ogni periodo.
- **Addebitare un setup quando non si produce nulla.** Un costo di setup si paga solo nei periodi con $x_t > 0$.
- **Dimenticare la scorta degli ultimi periodi.** Il costo di mantenimento si paga sulla scorta alla fine di ogni periodo, compresi i periodi in cui il piano non produce nulla.
- **Credere che si produca solo con scorta vuota.** La proprietà di Wagner e Whitin vale senza capacità; con le capacità il piano ottimo dell'[Esempio 8.1](#) la viola.
- **Contare 2^m piani ammissibili.** I 2^m sottoinsiemi di periodi sono le scelte di setup: alcune sono inammissibili, e a ciascuna ammissibile corrispondono uno o più piani.

8.9 In sintesi

In sintesi

- Il problema del lot sizing capacitato chiede le quantità da produrre in ogni periodo, con capacità, setup e scorte, al minimo costo totale; una soluzione è un piano di produzione.
- La parte combinatoria è la scelta dei periodi con un setup: ci sono 2^m scelte, solo alcune ammissibili, e a ciascuna corrispondono uno o più piani.
- Il lot-for-lot dà rapidamente un piano, accorpare le domande lo migliora; il numero di setup richiesto dalla domanda totale dà un lower bound.
- Con le capacità, i piani ottimi possono produrre mentre la scorta è positiva; senza capacità non lo fanno mai.
- Le varianti aggiungono prodotti, tempi di setup, scorta iniziale, straordinario o domanda non soddisfatta; i loro modelli sono nel [Capitolo 15 «Formulazioni dei problemi di lot sizing»](#).

Quadro riassuntivo dell'istanza del CLS (minimizzazione: il lower bound è un bound duale, l'upper bound è il costo di un piano):

Capitolo	Lower bound	Upper bound	Da
Questo capitolo	150	250	almeno tre setup; lot-for-lot
Questo capitolo, accor- pando	150	180	il piano (3, 0, 4, 4, 0)
Questo capitolo, enume- razione	180	180	i 94 piani ammissibili