Euristiche costruttive

Indice

Bin packing: le regole di inserimento

$P||C_{\max}$: la regola del meno carico

Set covering: il completamento più economico

Zaino: la regola del rapporto migliore

TSP: il vicino più vicino

Lot sizing: il costo unitario minimo

Ricerca locale, e che cosa non dà

Quando un'euristica costruttiva fallisce

Che cosa dà, e che cosa non dà

Il quadro delle euristiche

Obiettivi (1/2)

Alla fine di questo capitolo sarai in grado di:

- descrivere un'euristica costruttiva con i suoi tre ingredienti e dire da quale lato il suo valore è un bound;
- eseguire a mano next-fit, first-fit, best-fit, LPT, le regole del covering, del rapporto e del nearest neighbour;

Obiettivi (2/2)

Alla fine di questo capitolo sarai in grado di:

- calcolare il gap certificato di una soluzione euristica con un bound duale;
- riconoscere quando un'euristica si blocca e perché allora non dà alcun bound.

Prerequisiti

- Da questo modulo, Capitolo 2 «Rilassamenti, dualità e bound»: il bound duale, il bound dall'altro lato con cui confrontare un'euristica
- Da questo modulo, Capitolo 3 «Dal modello a Python/Gurobi»: i modelli di bin packing, makespan e TSP

Prima di risolvere, una soluzione

- Un modello MILP si può risolvere all'ottimo, ma prima conviene avere in mano *una* soluzione ammissibile
- Serve a due cose:
  - dà un **bound primale**: un numero che l'ottimo non può superare (in un minimo) o sotto cui non può stare (in un massimo)
  - dà un punto di partenza al solver, e un termine di confronto per capire se il modello dice cose sensate

Che cos'è un'euristica costruttiva

- Costruisce una soluzione **aggiungendo un elemento per volta**, senza mai tornare indietro
- Non garantisce l'ottimo, e in generale non lo trova
- Non garantisce nemmeno di arrivare a una soluzione ammissibile: può bloccarsi a metà, con un elemento che non entra da nessuna parte — proprio perché indietro non torna
- Quello che garantisce è di **fermarsi in fretta**

Che cosa deve produrre un'euristica (1/2)

1. uno **pseudocodice** leggibile, con l'ordine di scansione, il criterio di scelta, la regola di spareggio e il caso di fallimento scritti esplicitamente
2. la corrispondente **funzione Python**, riga per riga (in `python/euristiche.py`)
3. la **traccia** dell'esecuzione su un'istanza

Che cosa deve produrre un'euristica (2/2)

4. il **controllo di ammissibilità** della soluzione prodotta: vincoli, domini *e* interezza
5. il **bound** che ne esce, con il nome giusto: UB in un problema di minimo, LB in uno di massimo

Il punto 4 non è una formalità: una soluzione che soddisfa i vincoli lineari ma ha una componente frazionaria è ammissibile per il *rilassamento*, non per il MILP, e il suo valore non è un bound primale.

I tre ingredienti

Ogni euristica costruttiva si descrive dicendo tre cose, e solo tre:

1. un **ordine** in cui si guardano gli elementi da collocare
2. una **regola di inserimento**, che dice dove va l'elemento corrente
3. un **controllo di ammissibilità**, che dice quando un inserimento non si può fare

Cambiando uno solo dei tre si ottiene un'euristica diversa, con un valore diverso.

Osservazione 1: Il lato del bound dipende dall'obiettivo, non dall'euristica

Un'euristica produce una soluzione ammissibile, e il suo valore è raggiungibile.

In un problema di ***minimo*** *quel valore è un bound superiore:* $z(MILP) \leq UB$.

In un problema di ***massimo*** *è un bound inferiore:* $LB \leq z(MILP)$.

Chiamare UB il risultato di un'euristica costruttiva su un problema di massimo è l'errore di segno più frequente del corso.

Bin packing: le regole di inserimento

Introduzione: bin packing

- Il bin packing è il primo classico: degli oggetti vanno messi in contenitori tutti uguali, di capacità limitata, usandone il meno possibile
- Qui non lo risolviamo: lo *costruiamo*, una scelta per volta
- Vedremo tre regole di inserimento — next-fit, first-fit, best-fit — e come, sulla stessa istanza, diano risultati molto diversi

Next-fit, first-fit, best-fit: lo pseudocodice (1/2)

Input: n, pesi $w[1..n]$, capacità C, criterio γ.
Output: i contenitori usati.

- si parte senza nessun contenitore aperto
- per $j := 1$ a n:
  - si sceglie b^* secondo la regola
  - se nessun contenitore aperto va bene, se ne apre uno nuovo
  - si mette j in b^* e si aggiorna $res[b^*] \leftarrow res[b^*] - w[j]$

Next-fit, first-fit, best-fit: lo pseudocodice (2/2)

La regola di scelta di b^*:

- **next-fit**: solo il contenitore corrente, poi se ne apre uno nuovo
- **first-fit**: il primo contenitore aperto in cui l'oggetto ci sta
- **best-fit**: fra i contenitori in cui ci sta, quello di $\gamma(j, b, res)$ minimo

L'ordine e i pareggi

- Tutte e tre scandiscono gli oggetti **nell'ordine dato**
- Cambiare l'ordine cambia il risultato, e questo va detto quando si riporta un valore
- I pareggi nel criterio γ si rompono sull'indice più piccolo, così l'esecuzione è riproducibile

L'istanza

- Sei oggetti di peso $w = (4, 4, 5, 3, 2, 3)$, in contenitori di capacità $C = 7$
- Sono gli stessi contenitori del modello del capitolo su gurobipy, con due oggetti in più: con quattro oggetti le tre regole rispondono tutte "tre contenitori" e non si distinguono

Euristica	come riempie	UB	$z(MILP)$	gap
next-fit	[4] [4] [5] [3+2] [3]	5	3	66,7%
first-fit	[4+3] [4+2] [5] [3]	4	3	33,3%
best-fit sul riempimento	[4+3] [4+3] [5+2]	3	3	0,0%

Il bound elementare

- Il peso totale è $4 + 4 + 5 + 3 + 2 + 3 = 21$ e un contenitore ne porta 7
- Quindi servono almeno

$$\left\lceil \frac{21}{7} \right\rceil = 3 \quad \text{contenitori}$$

- Il best-fit ne usa tre, e li riempie tutti e tre esatti: $LB = UB = 3$
- L'ottimo è **dimostrato** senza mai chiamare il solver

Dove le tre regole si separano (1/2)

Next-fit:

- chiude un contenitore appena un oggetto non ci sta, e non torna mai indietro
- l'ultimo oggetto, di peso 3, entrerebbe esatto nel primo contenitore, che ha ancora 3 di spazio
- ma quel contenitore non lo guarda più: cinque contenitori invece di tre

Dove le tre regole si separano (2/2)

Fra **first-fit** e **best-fit** la differenza sta tutta sull'oggetto di peso 2:

- il first-fit lo mette nel *primo* che lo accoglie, il secondo contenitore, che resta con 1 di spazio ormai inutile
- il best-fit lo mette dove entra *esatto*, il terzo, che aveva 2
- così il 3 finale trova ancora posto nel secondo, e i contenitori restano tre

Riepilogo: bin packing

Le tre regole:

- next-fit guarda solo il contenitore corrente, first-fit il primo che accoglie, best-fit quello che si riempie meglio
- sulla stessa istanza danno 5, 4 e 3 contenitori

Che cosa portarsi via:

- l'ordine di scansione fa parte dell'euristica, va sempre dichiarato
- il bound elementare $\lceil \sum_j w_j / C \rceil$ costa niente e qui chiude il problema

$P||C_{max}$: la regola del meno carico

Introduzione: lo scheduling su macchine identiche

- Il secondo classico è lo scheduling su macchine identiche, in notazione standard $P||C_{\max}$
- n lavori di durata t_j su k macchine uguali, minimizzando l'istante in cui finisce l'ultima
- Vedremo la regola naturale (*list scheduling*) e l'ordine che la rende buona, da cui nasce l'**LPT**

L'idea

- Il *makespan* è il carico della macchina più carica
- **List scheduling**: il lavoro corrente va sulla macchina in quel momento meno carica
- L'ordine in cui si guardano i lavori decide il risultato: quello che funziona meglio è per **durata decrescente**
- La regola che ne esce ha un nome suo: **LPT** (*longest processing time first*)

Perché l'ordine decrescente

Lasciare per ultimi i lavori lunghi li rende impossibili da sistemare:
quando arrivano, tutte le macchine sono già cariche
e il lungo si somma per intero al carico più basso.

LPT: lo pseudocodice

Input: n, k, durate $t[1..n]$.
Output: un'assegnazione x e il suo makespan.

- $L[m] \leftarrow 0$ per ogni m (carichi correnti)
- per j in ordine di $t[j]$ *decrescente*:
  - $m^* \leftarrow \arg\min_m L[m]$ (pareggi: l'indice più piccolo)
  - $x[j][m^*] \leftarrow 1$; $L[m^*] \leftarrow L[m^*] + t[j]$
- si restituiscono x e $\max_m L[m]$

Esempio: sette lavori su tre macchine (1/2)

$t = (5, 5, 4, 4, 3, 3, 3)$, $k = 3$, totale 27. La traccia:

- **Passi 1–3.** I lavori 5, 5, 4 vanno sulle tre macchine vuote: $L = (5, 5, 4)$
- **Passo 4.** Lavoro 4: il carico minimo è la macchina 3, che passa a 8. $L = (5, 5, 8)$
- **Passi 5–6.** I due lavori da 3 vanno sulle macchine 1 e 2: $L = (8, 8, 8)$
- **Passo 7.** L'ultimo lavoro da 3 trova tutti i carichi pari a 8; per la regola dei pareggi va sulla macchina 1, che arriva a 11

Esempio: sette lavori su tre macchine (2/2)

- Makespan dell'LPT: $UB = 11$, con carichi $(11, 8, 8)$

Il bound elementare. Il makespan è almeno

$$\max\left(\max_j t_j,\ \frac{\sum_j t_j}{k}\right) = \max\left(5,\ \frac{27}{3}\right) = 9,$$

- nessuna macchina può fare meno del lavoro più lungo, e almeno una deve portare la media
- l'ottimo è proprio $z(MILP) = 9$, raggiunto con $\{5, 4\}$, $\{5, 4\}$, $\{3, 3, 3\}$: l'LPT sbaglia del 22,2%

Due bound gratis, da confrontare

- $\max_j t_j$ e $\sum_j t_j/k$ sono due bound inferiori che si calcolano senza risolvere niente
- Il migliore dei due è già spesso vicino all'ottimo
- Un bound "ovvio" che nessuno scrive è un bound sprecato: il bound duale serve quando quelli ovvi non bastano, non al loro posto

Riepilogo: $P||C_{max}$

La regola:

- list scheduling: ogni lavoro sulla macchina meno carica
- ordine per durata decrescente: è l'LPT

Sull'istanza di sette lavori:

- $UB = 11$ dall'LPT, $LB = 9$ dai due bound elementari, $z(MILP) = 9$
- i bound che si calcolano a mano vanno sempre scritti: sono gratis

Set covering: il completamento più economico

Introduzione: set covering

- Nel set covering si scelgono elementi di costo noto, in modo che ogni zona sia coperta da almeno uno di essi
- La regola costruttiva naturale non guarda il costo assoluto, ma il **costo per zona nuova coperta**
- Vedremo l'euristica, la traccia sull'istanza delle quattro squadre e il confronto con il bound duale

Euristica costruttiva di copertura: lo pseudocodice

Input: costi $c[1..n]$, insiemi $S[1..m]$ (chi copre la zona i).
Output: un insieme di elementi che copre tutte le zone.

- $scoperte \leftarrow \{1, 2, \ldots, m\}$; $\quad y[j] \leftarrow 0$ per ogni j
- finché $scoperte \neq \emptyset$:
  - per ogni j non ancora scelto:
    $nuove(j) \leftarrow |\{i \in scoperte : j \in S_i\}|$
  - se $nuove(j) = 0$ per ogni j: "nessuna soluzione trovata"
  - $j^* \leftarrow \arg\min_{j:nuove(j)>0} c[j]/nuove(j)$
  - $y[j^*] \leftarrow 1$; $\quad scoperte \leftarrow scoperte \setminus \{i : j^* \in S_i\}$

Il criterio

Il criterio è il **costo per zona nuova coperta**, non il costo assoluto.

Prendere l'elemento più economico senza guardare quanto copre
porta a soluzioni molto peggiori.

Esempio: le quattro squadre (1/2)

Costi $c = (4, 3, 5, 3)$, sei zone (le sei coppie di quartieri).

- **Passo 1.** Tutte e sei le zone sono scoperte; i rapporti sono $4/3$, $3/3 = 1$, $5/3$, $3/3 = 1$; pareggio fra gli elementi 2 e 4, si sceglie il 2, che copre le zone 1, 2 e 5
- **Passo 2.** Restano le zone 3, 4, 6; i rapporti sono $4/2 = 2$, $5/2$, $3/2$; si sceglie l'elemento 4, che copre le zone 4 e 6
- **Passo 3.** Resta la zona 3; i rapporti sono $4/1$ e $5/1$; si sceglie l'elemento 1

Esempio: le quattro squadre (2/2)

- Soluzione $\{1, 2, 4\}$, di costo $UB = 10$
- Qui è l'ottimo: $z(MILP) = 10$
- Il bound duale costruito nel capitolo sui bound duali vale 7: il **gap certificato** è del 30%, anche se l'euristica era già ottima
- Il gap certificato misura quello che *sappiamo*, non quello che abbiamo ottenuto

Riepilogo: set covering

La regola:

- si sceglie ogni volta l'elemento di rapporto $c[j]/nuove(j)$ minimo
- ci si ferma quando non resta nessuna zona scoperta

Sull'istanza delle quattro squadre:

- $UB = 10$, che è l'ottimo, ma il bound duale disponibile è 7
- un'euristica ottima con un gap certificato del 30%: il gap dice quanto si sa, non quanto si è perso

Zaino: la regola del rapporto migliore

Introduzione: lo zaino

- Nello zaino gli oggetti hanno un *valore* oltre a un peso, e la risorsa è una sola
- È un problema di **massimo**: il bound che ne esce si chiama LB, non UB
- La regola costruttiva naturale guarda il rapporto valore/peso: vedremo che cosa produce e dove sbaglia

Euristica per rapporto valore/peso: lo pseudocodice

Input: valori $p[1..n]$, pesi $w[1..n]$, capacità C.
Output: un sottoinsieme ammissibile.

- $residuo \leftarrow C$; $\quad y[j] \leftarrow 0$ per ogni j
- per j in ordine di $p[j]/w[j]$ *decrescente*:
  - se $w[j] \leq residuo$: $\quad y[j] \leftarrow 1$; $\quad residuo \leftarrow residuo - w[j]$
- si restituisce y

Esempio: quattro oggetti

Istanza: $p = (10, 7, 6, 4)$, $w = (5, 4, 3, 3)$, $C = 9$.

- I rapporti sono 2, 7/4, 2, 4/3
- Si prende l'oggetto 1 e poi l'oggetto 3 (peso complessivo 8), mentre il 2 e il 4 non entrano più
- Valore della soluzione: 16

Poiché il problema è di *massimo*,

$$LB = 16 \leq z(MILP) = 17,$$

con un gap del 5,9%.

Perché sbaglia

- L'ottimo prende gli oggetti 1 e 2, riempiendo lo zaino esattamente $(5 + 4 = 9)$
- L'euristica del rapporto sbaglia proprio perché l'oggetto 3, migliore come rapporto, lascia un residuo **inutilizzabile**
- Il rapporto valore/peso è un criterio locale: non vede che cosa resta da riempire dopo

Riepilogo: zaino

La regola:

- si scandiscono gli oggetti per rapporto $p[j]/w[j]$ decrescente e si prende ogni oggetto che ci sta

Sull'istanza di quattro oggetti:

- $LB = 16 \leq z(MILP) = 17$: in un massimo il bound dell'euristica è *inferiore*
- l'errore nasce dal residuo che l'oggetto 3 lascia inutilizzabile

TSP: il vicino più vicino

Introduzione: il commesso viaggiatore

- Date n città e le distanze $d_{ij} \in \mathbb{Q}_{>0}$ fra ogni coppia, si cerca il giro più corto che le visiti tutte una volta sola e torni al punto di partenza
- È il problema su cui la costruzione passo per passo si vede meglio, perché la soluzione è una sequenza: l'ordine *è* la soluzione
- Vedremo la regola del vicino più vicino e l'effetto della città di partenza

Il vicino più vicino

- Si parte da una città, e ogni volta si va alla più vicina fra quelle non ancora visitate
- Quando non ne restano, si torna alla partenza
- È **ammissibile per costruzione**: ogni città entra una volta sola, e l'ultimo arco chiude il giro
- Ed è veloce, perché a ogni passo guarda solo le distanze dalla città corrente

Esempio: cinque città (1/3)

Le distanze, simmetriche:

	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	—

Esempio: cinque città (2/3)

Partendo dalla città 1:

- la più vicina è la 3 (distanza 2); da lì la 2 (4); da lì la 4 (3); resta la 5 (7); e si torna alla 1, che costa 9
- il giro $1 \to 3 \to 2 \to 4 \to 5 \to 1$ è lungo 25

L'ultimo arco è quello che si paga: la regola sceglie bene finché ha scelta, e all'ultimo passo non ne ha più.

Esempio: cinque città (3/3)

Cambiando la città di partenza cambia il giro, e cambia la lunghezza:

partenza	giro	lunghezza
1	$1 \to 3 \to 2 \to 4 \to 5 \to 1$	25
2	$2 \to 4 \to 1 \to 3 \to 5 \to 2$	18
3	$3 \to 1 \to 4 \to 2 \to 5 \to 3$	18
4	$4 \to 1 \to 3 \to 2 \to 5 \to 4$	19
5	$5 \to 2 \to 4 \to 1 \to 3 \to 5$	18

Con cinque città i giri distinti sono $(5-1)!/2 = 12$: l'ottimo, per enumerazione, è $1 \to 3 \to 5 \to 2 \to 4 \to 1$, lungo 18.

Due cose da portare via

- L'euristica dà *una* soluzione ammissibile, quindi un bound superiore — qui $z(MILP) \leq 25$ — e nient'altro
  - che 18 sia l'ottimo lo sappiamo per enumerazione, non perché l'euristica lo dica
- Far girare la stessa regola da tutte le partenze e tenere il giro migliore costa n volte tanto e dà un bound superiore migliore
  - è la forma più semplice di *multi-start*, e resta un bound da un lato solo

Riepilogo: TSP

La regola:

- dalla città corrente si va sempre alla più vicina non visitata, poi si chiude il giro
- ammissibile per costruzione, ma l'ultimo arco è quello che si paga

Sull'istanza di cinque città:

- dalla città 1 il giro vale 25, cioè il 38,9% sopra l'ottimo 18
- cambiando partenza il valore cambia: il multi-start migliora il bound, ma resta un bound solo

Lot sizing: il costo unitario minimo

Introduzione: lot sizing

- Su T periodi, con domanda d_t da soddisfare, un costo fisso f ogni volta che si lancia una produzione e un costo h per ogni unità tenuta in magazzino per un periodo
- Si decide in quali periodi produrre e quanto
- La tensione è tutta lì: **lanciare di rado costa magazzino, lanciare spesso costa setup**

La regola del costo unitario minimo

- Dal primo periodo con domanda si prova a coprire $1, 2, 3, \ldots$ periodi consecutivi
- Per ciascuna scelta si calcola il **costo medio per unità** prodotta — setup più magazzino, diviso la quantità
- Si prende la copertura con il costo medio più basso
- Poi si riparte dal primo periodo rimasto scoperto

Least unit cost: lo pseudocodice

Input: domande $d[1..T]$, costo di lancio f, costo unitario di magazzino h.

- $t \leftarrow 1$; finché $t \leq T$:
  - si saltano i periodi con $d[t] = 0$
  - per $k := 1$ a $T - t + 1$:

$$q_k \leftarrow \sum_{s=t}^{t+k-1} d[s], \qquad c_k \leftarrow \frac{f + h \sum_{s=t}^{t+k-1} (s-t)\, d[s]}{q_k}$$

  - $k^* \leftarrow \arg\min_k c_k$ (il costo medio per unità più basso)
  - si produce q_{k^*} nel periodo t; $t \leftarrow t + k^*$

Osservazione 2: Questa non è la procedura di Wagner–Whitin

Wagner–Whitin è un algoritmo ***esatto*** *di programmazione dinamica per il modello di lot sizing senza capacità: risolve quel modello all'ottimo in tempo polinomiale.*

La procedura qui sopra è un'euristica, e il suo valore è solo un bound. Chiamarla "euristica costruttiva di Wagner–Whitin" confonde due cose diverse.

Esempio: cinque periodi

Domande $d = (20, 10, 30, 40, 10)$, lancio $f = 50$, magazzino $h = 1$.

- **Passo 1.** Dal periodo 1 conviene coprire 2 periodi (quantità 30, costo unitario 2)
- **Passo 2.** Dal periodo 3 conviene coprire 2 periodi (quantità 70, costo unitario $\approx 1{,}286$)
- **Passo 3.** Dal periodo 5 resta solo quel periodo (quantità 10, costo unitario 5)

Costo della soluzione euristica: $UB = 200$. L'ottimo del MILP corrispondente è $z(MILP) = 170$, un gap del 17,6%.

Riepilogo: lot sizing

La regola:

- si copre il blocco di periodi con il costo medio per unità più basso, poi si riparte
- il compromesso fra setup e magazzino è tutto nel rapporto c_k

Sull'istanza di cinque periodi:

- $UB = 200$ contro $z(MILP) = 170$: gap del 17,6%
- Wagner–Whitin darebbe 170, ma è un algoritmo esatto, non questa euristica

Ricerca locale, e che cosa non dà

Introduzione: la ricerca locale

- Dopo aver costruito una soluzione, il passo successivo naturale è provare a migliorarla con piccole modifiche
- Vedremo che cos'è un ottimo locale e perché dipende dalla famiglia di mosse scelta
- E soprattutto che cosa la ricerca locale **non** dà: nessun bound dal lato ottimistico

Un passo oltre la costruzione

- Una **ricerca locale** parte da una soluzione ammissibile e prova mosse elementari (sposta un lavoro, scambia due oggetti), accettando quelle che migliorano
- Si ferma in un **ottimo locale**: una soluzione che nessuna mossa della famiglia scelta migliora
- "Locale" è relativo alla famiglia di mosse: cambiando le mosse cambia l'ottimo locale

Sull'esempio dell'LPT

Soluzione LPT: $L = (11, 8, 8)$, makespan 11.

- La mossa "sposta un lavoro su un'altra macchina" non migliora nulla
- Spostare uno dei due lavori da 3 dalla macchina 1 porta il suo carico a 8, ma alza a 11 quello della macchina che lo riceve
- La ricerca locale si ferma a 11, mentre l'ottimo è 9: per arrivarci serve uno *scambio* fra due macchine, non un semplice spostamento

Osservazione 3: Un ottimo locale non è un bound migliore

La ricerca locale restituisce una soluzione ammissibile, quindi un bound dal lato pessimistico, e nient'altro.

Non dimostra niente sull'ottimo, non produce nessun bound dal lato ottimistico, e il fatto che si sia fermata non significa che sia arrivata.

L'unico modo per sapere quanto sia buona è confrontarla con un bound dall'altro lato.

Riepilogo: ricerca locale

Che cos'è:

- si parte da una soluzione ammissibile e si accettano le mosse che migliorano
- ci si ferma in un ottimo locale, relativo alla famiglia di mosse scelta

Che cosa non dà:

- nessuna dimostrazione sull'ottimo, nessun bound dal lato ottimistico
- sull'LPT si ferma a 11 mentre l'ottimo è 9: servono gli scambi, non gli spostamenti

Quando un'euristica costruttiva fallisce

Introduzione: il fallimento

- Un'euristica costruttiva può arrivare a un punto in cui l'elemento corrente non si può collocare da nessuna parte
- Non è un errore di programmazione: è la conseguenza del **non tornare mai indietro**
- Vedremo un'istanza dove succede, e che cosa si può (e non si può) concludere

Osservazione 4: "Nessuna soluzione trovata" non è "nessuna soluzione esiste"

Tre lavori di durata $(3, 3, 2)$ *su due macchine con disponibilità* $(5, 3)$. *Il next-fit:*

- *il lavoro 1 va sulla macchina 1 (residuo* 2*);*
- *il lavoro 2 non ci sta e passa alla macchina 2 (residuo* 0*);*
- *il lavoro 3 non ci sta e non ci sono altre macchine:* ***fallimento****.*

Ma il problema è ammissibile: i lavori 2 e 3 stanno insieme sulla macchina 1 $(3 + 2 = 5)$ *e il lavoro 1 sulla macchina 2* $(3 \leq 3)$.

Un'euristica costruttiva è miope

- Decide una cosa alla volta e non torna indietro
- Il suo fallimento è un'informazione **sull'euristica**, non sul problema
- Per dimostrare che un modello è inammissibile serve il solver (`Status = INFEASIBLE`) o una dimostrazione, non un'euristica costruttiva andata male

Osservazione 5: Un'euristica che fallisce non dà nessun bound

Se l'euristica non restituisce una soluzione, non si è ottenuto niente: non c'è alcun bound primale. Le due cose da fare sono, nell'ordine:

1. *cambiare l'ordine — quello per dimensione decrescente fallisce molto meno spesso di quello naturale, perché sistema per primi gli oggetti difficili;*
2. *cambiare la regola di inserimento.*

Se anche così non si trova nulla

La domanda diventa se il problema sia ammissibile,
e questa è una domanda **per il modello**, non per l'euristica.

Riepilogo: il fallimento

Che cosa significa:

- l'euristica è miope: decide una cosa per volta e non torna indietro
- il suo fallimento parla dell'euristica, non dell'ammissibilità del problema

Che cosa fare:

- prima cambiare l'ordine (dimensione decrescente), poi la regola di inserimento
- l'inammissibilità si dimostra con il solver o con un argomento, mai con un'euristica

Che cosa dà, e che cosa non dà

Introduzione: il conto esatto

- Fin qui abbiamo visto sei euristiche su sei problemi diversi
- Ora facciamo il conto esatto di che cosa si sa dopo averne eseguita una
- E definiamo il **gap certificato**, l'unica misura di qualità disponibile *prima* di risolvere il MILP

Il conto esatto di che cosa si sa

Sia $\bar{z}$ il valore della soluzione trovata da un'euristica costruttiva.

- In un problema di **minimo** si sa che

$$z(MILP) \leq \bar{z}, \qquad \text{cioè} \quad UB = \bar{z}$$

- In un problema di **massimo** si sa che

$$z(MILP) \geq \bar{z}, \qquad \text{cioè} \quad LB = \bar{z}$$

Quello che non si sa

- In nessuno dei due casi si sa *quanto* $\bar{z}$ sia lontano dall'ottimo
- Per saperlo serve un bound dall'altro lato
- E quello non viene da una soluzione ammissibile: viene da un **rilassamento** o da un **argomento combinatorio**

Il gap certificato

Con entrambi i bound in mano si scrive il **gap certificato**: in assoluto $UB - LB$, e in percentuale

$$\frac{UB - LB}{|UB|} \text{ in un minimo,} \qquad \frac{UB - LB}{|LB|} \text{ in un massimo.}$$

- In un minimo l'incumbent è UB, in un massimo è LB
- Al denominatore sta sempre la soluzione che si ha in mano, mai $z(MILP)$
- È questo che lo rende l'unica misura di qualità disponibile *prima* di risolvere il MILP

Perché costruire l'euristica a mano, se c'è il solver

Un solver moderno trova soluzioni ammissibili da solo, spesso migliori. Costruirne una a mano serve comunque:

- obbliga a capire che cosa rende una soluzione ammissibile, e questo è esattamente il contenuto dei vincoli
- dà un valore di riferimento indipendente dal solver: se il solver restituisce di peggio, c'è un errore nel modello
- nei problemi grandi il solver non arriva all'ottimo, e la qualità della soluzione finale dipende da quella iniziale

Riepilogo: che cosa dà un'euristica

Quello che dà:

- un bound primale: $UB = \bar{z}$ in un minimo, $LB = \bar{z}$ in un massimo
- una soluzione che si può realizzare davvero

Quello che non dà:

- nessuna informazione sulla distanza dall'ottimo
- il gap certificato richiede un bound dall'altro lato, che viene da un rilassamento o da un argomento combinatorio

Il quadro delle euristiche

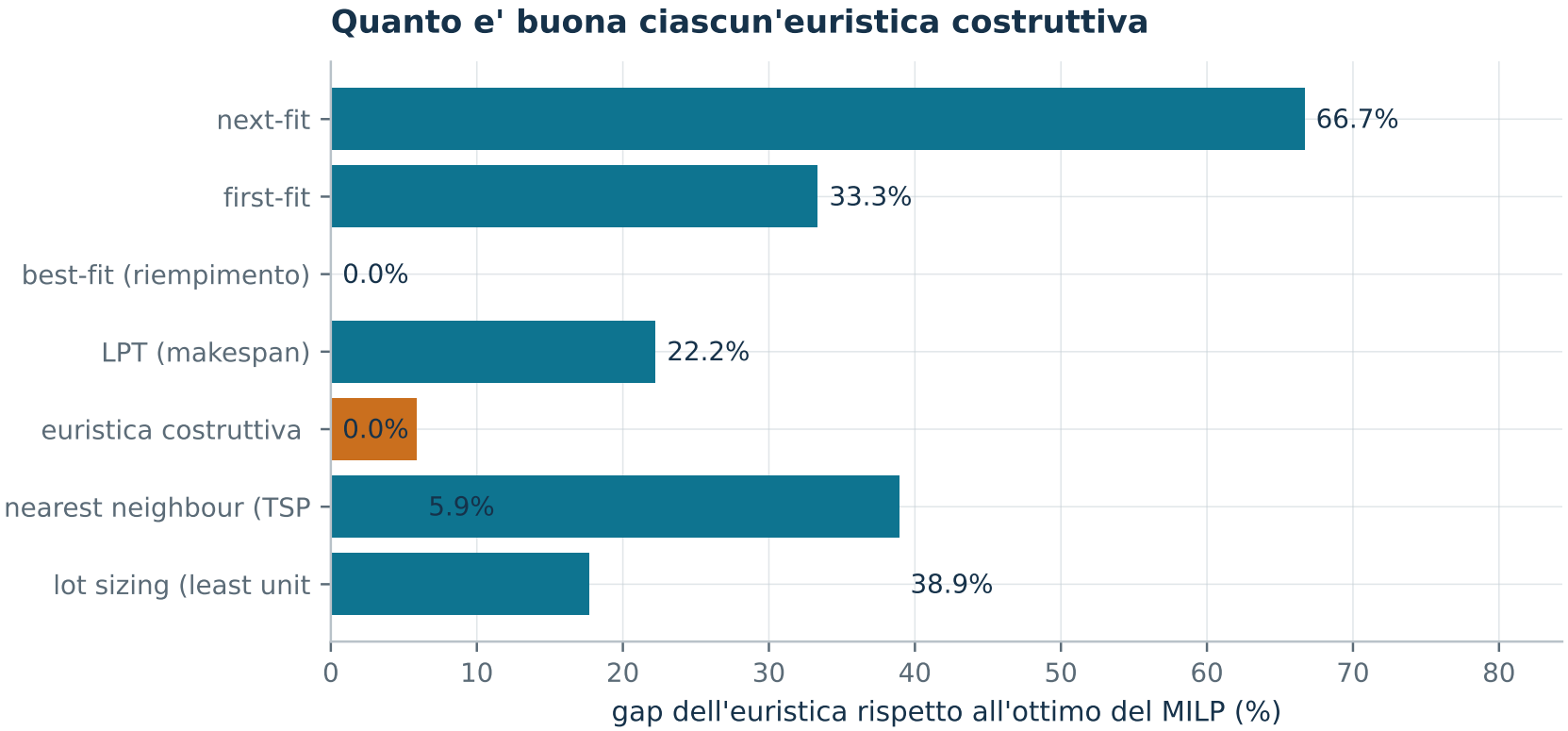
Introduzione: il quadro d'insieme

- Mettiamo in una sola tabella tutte le euristiche del capitolo, con il loro valore, l'ottimo del MILP e il gap
- Serve a vedere, tutto insieme, quanto siano diverse fra loro
- E a capire perché il corso chiede **sempre due bound**

La tabella riassuntiva

Euristica	Verso	valore	$z(MILP)$	gap
next-fit (bin packing)	min (UB)	5	3	66,7%
first-fit (bin packing)	min (UB)	4	3	33,3%
best-fit sul riempimento	min (UB)	3	3	0,0%
LPT (makespan)	min (UB)	11	9	22,2%
copertura (set covering)	min (UB)	10	10	0,0%
rapporto (zaino)	max (LB)	16	17	5,9%
nearest neighbour (TSP)	min (UB)	25	18	38,9%
least unit cost (lot sizing)	min (UB)	200	170	17,6%

Il gap di ciascuna euristica



Petrolio per i problemi di minimo, arancione per quelli di massimo.

Che cosa si impara da questa tabella

- Due euristiche trovano l'ottimo e sei no, e *prima* di risolvere il MILP non c'è modo di sapere quali
- Un gap del 0% e uno del 67% si distinguono soltanto *dopo*
- È esattamente per questo che il corso chiede sempre due bound: un'euristica da sola dice quanto costa una soluzione che si può realizzare, non quanto si sta perdendo

Riepilogo: il quadro

La tabella:

- otto euristiche su sei problemi, gap da 0% a 66,7%
- due trovano l'ottimo, ma non si può sapere quali prima di risolvere

La morale:

- un'euristica da sola non misura la propria qualità
- servono sempre due bound, uno per lato

Riepilogo finale (1/3)

Che cos'è un'euristica costruttiva:

- costruisce una soluzione un elemento per volta, senza tornare indietro
- si descrive con tre ingredienti: ordine, regola di inserimento, controllo di ammissibilità

Il lato del bound:

- dipende dall'obiettivo, non dall'euristica: UB in un minimo, LB in un massimo

Riepilogo finale (2/3)

Le regole viste:

- bin packing: next-fit, first-fit, best-fit
- $P||C_{\max}$: list scheduling per durata decrescente (LPT)
- set covering: costo per zona nuova coperta
- zaino: rapporto valore/peso decrescente
- TSP: vicino più vicino
- lot sizing: costo medio per unità minimo

Riepilogo finale (3/3)

I limiti:

- un'euristica può fallire: fallimento non significa inammissibilità
- la ricerca locale si ferma in un ottimo locale, e non dà nessun bound in più

Il conto finale:

- un'euristica dà un bound da un lato solo
- il gap certificato ($UB - LB$) richiede un bound dall'altro lato
- è l'unica misura di qualità disponibile prima di risolvere il MILP