

SAPIENZA
UNIVERSITÀ DI ROMA

Algoritmi di approssimazione

Volume 1 — Algoritmi di approssimazione

Algoritmi di approssimazione: rapporti di prestazione e schemi di approssimazione, con algoritmi e analisi del caso peggiore per lo zaino, il bin packing, il TSP metrico e lo scheduling su macchine parallele, e i limiti dell'approssimazione per la colorazione dei vertici, la clique massima e il TSP generale.

Fabio Furini

fabiofurini.github.io

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

Versione online e materiale scaricabile:
fabiofurini.github.io/algoritmi-di-approssimazione/
github.com/fabiofurini/algoritmi-di-approssimazione

Anno accademico 2026–2027

Indice

1 Principi degli algoritmi di approssimazione **4**

1.1 Algoritmi di approssimazione 5
1.2 Schemi di approssimazione 8
1.3 Errori tipici 10
1.4 In sintesi 11

Problemi di zaino **12**

2 Algoritmi di approssimazione per lo zaino **13**

2.1 Notazione 14
2.2 Oggetto critico e capacità residua 14
2.3 L'algoritmo greedy 15
2.4 L'algoritmo Ext-Greedy 21
2.5 L'algoritmo H^2 23
2.6 L'algoritmo H^ℓ 26
2.7 L'algoritmo Basic-FPTAS 28
2.8 Garanzie di prestazione 31
2.9 Garanzia di prestazione assoluta 43
2.10 Errori tipici 44
2.11 In sintesi 45
2.12 Implementazione in `C++` 45

Problemi di packing **56**

3 Algoritmi di approssimazione per il bin packing **57**

3.1 Notazione 58
3.2 Next Fit 58
3.3 La classe Any Fit 61
3.4 Proprietà di riempimento a metà 71
3.5 Garanzie di prestazione 74
3.6 Errori tipici 81
3.7 In sintesi 82

3.8 Implementazione in C++ 82

Problemi di colorazione **91**

4 Colorazione dei vertici: limiti dell'approssimazione **92**
4.1 Notazione 93
4.2 Il VCP non è approssimabile 93
4.3 Casi particolari 95
4.4 Esempi 99
4.5 Fonti 100
4.6 Errori tipici 100
4.7 In sintesi 101

Problemi di clique **102**

5 Clique massima: limiti dell'approssimazione **103**
5.1 Notazione 104
5.2 Il MCP non è approssimabile 104
5.3 Casi particolari 106
5.4 Esempio 108
5.5 Fonti 108
5.6 Errori tipici 109
5.7 In sintesi 109

Problemi di routing **110**

6 TSP: il caso generale e il caso metrico **111**
6.1 Il TSP generale non è approssimabile 112
6.2 Un caso particolare: il TSP metrico 113
6.3 L'algoritmo del doppio albero e l'algoritmo di Christofides 113
6.4 Proprietà strutturali 116
6.5 Garanzie di prestazione 118
6.6 Fonti 121
6.7 Errori tipici 121
6.8 In sintesi 121

Problemi di scheduling **123**

7 Approssimazione per $P\|C_{\max}$ **124**
7.1 Notazione 125
7.2 Gli algoritmi $LIST$ e LPT 125

7.3 Una proprietà del list scheduling 128
7.4 Garanzie di prestazione 128
7.5 Fonti 132
7.6 Errori tipici 132
7.7 In sintesi 132

Capitolo 1

Principi degli algoritmi di approssimazione

Prerequisiti

- Dal modulo *Algoritmi euristici*, il capitolo «Algoritmi euristici: definizioni»: algoritmi euristici, soluzioni ammissibili e valore della soluzione restituita da un algoritmo euristico.
- Dal modulo *Problemi di ottimizzazione combinatoria*, i capitoli sui problemi di ottimizzazione combinatoria: istanze, soluzioni ottime e valore ottimo $z(I)$ di un'istanza I.

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- calcolare il rapporto di prestazione, l'errore additivo e l'errore relativo di un algoritmo euristico su un'istanza, per problemi di minimo e di massimo;
- stabilire quando un numero è una garanzia di prestazione relativa o additiva, e quando è tight;
- dimostrare che una garanzia è tight esibendo un'istanza, o una successione di istanze;
- definire uno schema di approssimazione, un PTAS e un FPTAS, e tradurre l'accuratezza ε in una garanzia di prestazione.

1.1 Algoritmi di approssimazione

Definizione 1.1: Valore euristico

Sia $\mathscr{A}$ un **algoritmo euristico** per un **problema di ottimizzazione** $\mathscr{P}$. Per ogni **istanza** I di $\mathscr{P}$:

1. $z(I)$ è il **valore ottimo** di $\mathscr{P}$ su I (supposto positivo);
2. $\mathscr{A}(I)$ è il **valore euristico**, cioè il valore obiettivo della soluzione ammissibile restituita da $\mathscr{A}$ su I (supposto positivo).

- Supponiamo che $\mathscr{A}$ restituisca sempre una **soluzione ammissibile**. Quindi, per ogni istanza I,

$$\begin{cases} \mathscr{A}(I) \geq z(I) \quad \text{e} \quad \dfrac{\mathscr{A}(I)}{z(I)} \geq 1 & \text{se } \mathscr{P} \text{ è un problema di minimo,} \\ \mathscr{A}(I) \leq z(I) \quad \text{e} \quad \dfrac{z(I)}{\mathscr{A}(I)} \geq 1 & \text{se } \mathscr{P} \text{ è un problema di massimo.} \end{cases}$$

In generale, un algoritmo euristico può a volte generare soluzioni di valore quasi ottimo, ma in altri casi può produrre soluzioni sensibilmente peggiori. Di conseguenza, ci interessa "misurare" (quando possibile) la qualità delle soluzioni fornite da un algoritmo euristico.

- Adottiamo il punto di vista del **comportamento nel caso peggiore**: valutiamo un algoritmo euristico guardando il massimo scostamento possibile, su tutte le istanze, fra il valore euristico e il valore ottimo.

- Ci sono due modi standard di misurare questo scostamento:

  1. **Garanzia di prestazione relativa** (garanzia moltiplicativa): in questo caso misuriamo la qualità della soluzione euristica guardando l'*errore relativo*, cioè quanto il valore euristico dista dall'ottimo *in termini relativi*, usando rapporti.

  2. **Garanzia di prestazione assoluta** (garanzia additiva): in questo caso misuriamo la qualità della soluzione euristica guardando l'*errore assoluto*, cioè la differenza fra il valore euristico e il valore ottimo.

1.1.1 Garanzia di prestazione relativa

Definizione 1.2: Rapporto di prestazione

Il **rapporto di prestazione** di $\mathscr{A}$ su un'istanza I è

$$\begin{cases} \dfrac{\mathscr{A}(I)}{z(I)} & \text{se } \mathscr{P} \text{ è un problema di minimo,} \\ \dfrac{z(I)}{\mathscr{A}(I)} & \text{se } \mathscr{P} \text{ è un problema di massimo.} \end{cases}$$

- Nell'ipotesi di ammissibilità, il rapporto di prestazione è almeno 1 per ogni istanza I.

Definizione 1.3: Algoritmo di α-approssimazione

Un algoritmo euristico $\mathscr{A}$ per un problema di ottimizzazione $\mathscr{P}$ è un **algoritmo di α-approssimazione** se esiste una costante $\alpha \in [1, +\infty)$ tale che, per ogni istanza I di $\mathscr{P}$,

$$\begin{cases} \dfrac{\mathscr{A}(I)}{z(I)} \leq \alpha & \text{se } \mathscr{P} \text{ è un problema di minimo,} \\ \dfrac{z(I)}{\mathscr{A}(I)} \leq \alpha & \text{se } \mathscr{P} \text{ è un problema di massimo.} \end{cases}$$

Ogni costante α di questo tipo si chiama **garanzia di prestazione relativa** di $\mathscr{A}$.

- Una garanzia di prestazione relativa α per $\mathscr{A}$ è **tight** se nessuna costante minore $\alpha' < \alpha$ è anch'essa una garanzia di prestazione relativa per $\mathscr{A}$ (cioè tale che le stesse disuguaglianze valgano per tutte le istanze con α' al posto di α).
- Che una garanzia sia tight si dimostra di solito esibendo un'istanza I per cui il rapporto di prestazione raggiunge α, cioè

$$\frac{\mathscr{A}(I)}{z(I)} = \alpha \quad \text{(minimo)} \qquad \text{oppure} \qquad \frac{z(I)}{\mathscr{A}(I)} = \alpha \quad \text{(massimo)},$$

  oppure una successione di istanze $\{I_n\}$ i cui rapporti di prestazione si avvicinano arbitrariamente ad α.

Definizione 1.4: Rapporto di prestazione nel caso peggiore

Il **rapporto di prestazione nel caso peggiore** di $\mathscr{A}$ è

$$r(\mathscr{A}) = \begin{cases} \sup\left\{ \dfrac{\mathscr{A}(I)}{z(I)} : I \text{ è un'istanza di } \mathscr{P} \right\} & \text{se } \mathscr{P} \text{ è un problema di minimo,} \\ \sup\left\{ \dfrac{z(I)}{\mathscr{A}(I)} : I \text{ è un'istanza di } \mathscr{P} \right\} & \text{se } \mathscr{P} \text{ è un problema di massimo.} \end{cases}$$

- Il valore $r(\mathscr{A})$ è la costante minima α per cui $\mathscr{A}$ è un algoritmo di α-approssimazione. La garanzia di prestazione relativa α è tight se e solo se $\alpha = r(\mathscr{A})$.
- Se $r(\mathscr{A}) < \infty$, allora $\mathscr{A}$ ammette una garanzia di prestazione relativa finita (equivalentemente, ha un rapporto di prestazione limitato). Se $r(\mathscr{A}) = \infty$, allora $\mathscr{A}$ non ha alcuna garanzia relativa nel caso peggiore.
- Terminologia usata in letteratura
  - una garanzia di prestazione relativa finita α si chiama anche **fattore di approssimazione** (o **garanzia di approssimazione**)
  - un **algoritmo di α-approssimazione** si chiama anche **algoritmo di approssimazione a fattore costante**
  - $r(\mathscr{A})$ si chiama anche **rapporto di approssimazione (nel caso peggiore)** di $\mathscr{A}$

1.1.2 Garanzia di prestazione additiva

- L'**errore additivo** (o **errore assoluto**) di $\mathscr{A}$ su un'istanza I è

$$\begin{cases} \mathscr{A}(I) - z(I) & \text{se } \mathscr{P} \text{ è un problema di minimo,} \\ z(I) - \mathscr{A}(I) & \text{se } \mathscr{P} \text{ è un problema di massimo.} \end{cases}$$

  Nell'ipotesi di ammissibilità, questa quantità è non negativa per ogni istanza I.

Definizione 1.5: Algoritmo di approssimazione β-additivo

Un algoritmo euristico $\mathscr{A}$ per un problema di ottimizzazione $\mathscr{P}$ è un **algoritmo di approssimazione β-additivo** se esiste una costante $\beta \in [0, +\infty)$ tale che, per ogni istanza I di $\mathscr{P}$,

$$\begin{cases} \mathscr{A}(I) - z(I) \leq \beta & \text{se } \mathscr{P} \text{ è un problema di minimo,} \\ z(I) - \mathscr{A}(I) \leq \beta & \text{se } \mathscr{P} \text{ è un problema di massimo.} \end{cases}$$

Ogni costante β di questo tipo si chiama **garanzia di prestazione additiva** di $\mathscr{A}$.

- Una garanzia di prestazione additiva β per $\mathscr{A}$ è **tight** se nessuna costante minore $\beta' < \beta$ è anch'essa una garanzia di prestazione additiva per $\mathscr{A}$.
- Che una garanzia sia tight si dimostra di solito esibendo un'istanza I per cui l'errore additivo raggiunge β, cioè

$$\mathscr{A}(I) - z(I) = \beta \ \text{ (minimo)} \qquad \text{oppure} \qquad z(I) - \mathscr{A}(I) = \beta \ \text{ (massimo)},$$

  oppure una successione di istanze $\{I_n\}$ i cui errori additivi si avvicinano arbitrariamente a β.

Definizione 1.6: Errore additivo nel caso peggiore

L'**errore additivo nel caso peggiore** di $\mathscr{A}$ è

$$e(\mathscr{A}) = \begin{cases} \sup\{\mathscr{A}(I) - z(I) : I \text{ istanza di } \mathscr{P}\} & \text{se } \mathscr{P} \text{ è un problema di minimo,} \\ \sup\{z(I) - \mathscr{A}(I) : I \text{ istanza di } \mathscr{P}\} & \text{se } \mathscr{P} \text{ è un problema di massimo.} \end{cases}$$

- Il valore $e(\mathscr{A})$ è la costante minima β per cui $\mathscr{A}$ è un algoritmo di approssimazione β-additivo. La garanzia di prestazione additiva β è tight se e solo se $\beta = e(\mathscr{A})$.
- Se $e(\mathscr{A}) < \infty$, allora $\mathscr{A}$ ammette una garanzia di prestazione additiva finita (equivalentemente, ha errore additivo limitato). Se $e(\mathscr{A}) = \infty$, allora $\mathscr{A}$ non ha alcuna garanzia additiva (assoluta) nel caso peggiore.
- Terminologia usata in letteratura
  - una garanzia di prestazione additiva finita β si chiama anche **garanzia di prestazione assoluta** o **garanzia di approssimazione additiva**
  - $e(\mathscr{A})$ si chiama anche **errore assoluto (additivo) nel caso peggiore** di $\mathscr{A}$

1.2 Schemi di approssimazione

- L'**errore relativo** (detto anche **errore percentuale**) di $\mathscr{A}$ su un'istanza I è

$$\begin{cases} \dfrac{\mathscr{A}(I) - z(I)}{z(I)} & \text{se } \mathscr{P} \text{ è un problema di minimo,} \\ \dfrac{z(I) - \mathscr{A}(I)}{z(I)} & \text{se } \mathscr{P} \text{ è un problema di massimo.} \end{cases}$$

  Nell'ipotesi di ammissibilità, questa quantità è non negativa per ogni istanza I, ed è uguale a 0 se e solo se $\mathscr{A}(I) = z(I)$.
- L'errore relativo divide per il valore ottimo $z(I)$, e non va confuso con il gap del capitolo «Ottimizzazione combinatoria: definizioni» del modulo *Problemi di ottimizzazione combinatoria*, che divide per il lower bound e non è mai minore (capitolo «Algoritmi euristici: definizioni» del modulo *Algoritmi euristici*).

Definizione 1.7: Schema di approssimazione

Uno **schema di approssimazione** per un problema di ottimizzazione $\mathscr{P}$ è una famiglia di algoritmi euristici $\{\mathscr{A}_\varepsilon\}_{\varepsilon \in (0,1)}$ tale che, per ogni $\varepsilon \in (0,1)$ e per ogni istanza I di $\mathscr{P}$,

$$\begin{cases} \dfrac{\mathscr{A}_\varepsilon(I) - z(I)}{z(I)} \leq \varepsilon & \text{se } \mathscr{P} \text{ è un problema di minimo,} \\ \dfrac{z(I) - \mathscr{A}_\varepsilon(I)}{z(I)} \leq \varepsilon & \text{se } \mathscr{P} \text{ è un problema di massimo.} \end{cases}$$

- Per i problemi di minimo,

$$\frac{\mathscr{A}_\varepsilon(I) - z(I)}{z(I)} \leq \varepsilon \qquad \text{per ogni istanza } I \text{ di } \mathscr{P}.$$

implica

$$\mathscr{A}_\varepsilon(I) - z(I) \leq \varepsilon\, z(I) \quad \Longleftrightarrow \quad \mathscr{A}_\varepsilon(I) \leq (1+\varepsilon)\, z(I).$$

Dividendo per $z(I) > 0$,

$$\frac{\mathscr{A}_\varepsilon(I)}{z(I)} \leq 1 + \varepsilon \qquad \text{per ogni istanza } I \text{ di } \mathscr{P}.$$

Quindi $\mathscr{A}_\varepsilon$ è un algoritmo di $(1+\varepsilon)$-approssimazione.

- Per i problemi di massimo,

$$\frac{z(I) - \mathscr{A}_\varepsilon(I)}{z(I)} \leq \varepsilon \qquad \text{per ogni istanza } I \text{ di } \mathscr{P}.$$

implica

$$z(I) - \mathscr{A}_\varepsilon(I) \leq \varepsilon\, z(I) \quad \Longleftrightarrow \quad \mathscr{A}_\varepsilon(I) \geq (1-\varepsilon)\, z(I).$$

Poiché $0 < \varepsilon < 1$ e $z(I) > 0$, dividendo per $(1-\varepsilon)z(I)$ si ottiene

$$\frac{z(I)}{\mathscr{A}_\varepsilon(I)} \leq \frac{1}{1-\varepsilon} \qquad \text{per ogni istanza } I \text{ di } \mathscr{P}.$$

Quindi $\mathscr{A}_\varepsilon$ è un algoritmo di $\left(\frac{1}{1-\varepsilon}\right)$-approssimazione.

È naturale che il tempo di esecuzione di uno schema di approssimazione cresca al diminuire di ε: più piccolo è l'errore ammesso, più alto è il tempo di esecuzione richiesto.

- La quantità $\frac{1}{\varepsilon}$ si chiama spesso **accuratezza**. Infatti, dalle disuguaglianze che definiscono uno schema di approssimazione otteniamo

$$\frac{\mathscr{A}_\varepsilon(I) - z(I)}{z(I)} \leq \varepsilon \implies \mathscr{A}_\varepsilon(I) - z(I) \leq \varepsilon\, z(I) \implies \frac{z(I)}{\mathscr{A}_\varepsilon(I) - z(I)} \geq \frac{1}{\varepsilon} \qquad \text{(minimo)},$$

$$\frac{z(I) - \mathscr{A}_\varepsilon(I)}{z(I)} \leq \varepsilon \implies z(I) - \mathscr{A}_\varepsilon(I) \leq \varepsilon\, z(I) \implies \frac{z(I)}{z(I) - \mathscr{A}_\varepsilon(I)} \geq \frac{1}{\varepsilon} \qquad \text{(massimo)}.$$

Nel caso particolare in cui $\mathscr{A}_\varepsilon(I) = z(I)$, i denominatori sopra sono nulli; in questo caso interpretiamo le frazioni corrispondenti come $+\infty$, e le disuguaglianze valgono banalmente.

- Non tutti gli schemi di approssimazione sono ugualmente utili in pratica: li distinguiamo in base a come il tempo di esecuzione dipende dalla dimensione dell'input e dall'accuratezza $\frac{1}{\varepsilon}$.

Definizione 1.8: Schema di approssimazione polinomiale (PTAS)

Uno schema di approssimazione $\{\mathscr{A}_\varepsilon\}_{\varepsilon\in(0,1)}$ è uno **schema di approssimazione polinomiale** se, per ogni $\varepsilon \in (0,1)$ fissato, il tempo di esecuzione di $\mathscr{A}_\varepsilon$ è polinomiale nella dimensione dell'istanza.

- Quindi il tempo di esecuzione è polinomiale nella dimensione dell'input, ma può crescere molto rapidamente al diminuire di ε (cioè al crescere dell'accuratezza richiesta).

Definizione 1.9: Schema di approssimazione completamente polinomiale (FPTAS)

Uno schema di approssimazione $\{\mathscr{A}_\varepsilon\}_{\varepsilon\in(0,1)}$ è uno **schema di approssimazione completamente polinomiale** se il tempo di esecuzione di $\mathscr{A}_\varepsilon$ è polinomiale sia nella dimensione dell'istanza sia nell'accuratezza $\frac{1}{\varepsilon}$.

1.3 Errori tipici

Attenzione

- **Invertire il rapporto.** Il rapporto di prestazione è $\frac{\mathscr{A}(I)}{z(I)}$ per un problema di minimo e $\frac{z(I)}{\mathscr{A}(I)}$ per un problema di massimo: in entrambi i casi è almeno 1.
- **Dimostrare una garanzia con esempi.** Un'istanza su cui il rapporto vale α mostra che nessuna costante minore di α è una garanzia, ma non dimostra che α lo sia: per questo serve una dimostrazione per *ogni* istanza.
- **Dire tight una garanzia senza un'istanza.** Una garanzia α è tight solo se qualche istanza, o una successione di istanze, raggiunge α al limite.
- **Confondere ε e α.** Uno schema di approssimazione con errore relativo ε è un algoritmo di $(1+\varepsilon)$-approssimazione per un problema di minimo, ma un algoritmo di $\frac{1}{1-\varepsilon}$-approssimazione per un problema di massimo.
- **PTAS e FPTAS.** Il tempo di esecuzione di un PTAS può crescere come $n^{1/\varepsilon}$; solo per un FPTAS è polinomiale anche in $\frac{1}{\varepsilon}$.

1.4 In sintesi

In sintesi

- Una garanzia di prestazione relativa limita il rapporto $\mathscr{A}(I)/z(I)$ (minimo) o $z(I)/\mathscr{A}(I)$ (massimo) su tutte le istanze; una garanzia additiva limita la differenza.
- Una garanzia è tight se è raggiunta da un'istanza, o avvicinata da una successione di istanze; la garanzia minima è il rapporto nel caso peggiore $r(\mathscr{A})$, o l'errore additivo nel caso peggiore $e(\mathscr{A})$.
- Uno schema di approssimazione ha errore relativo al più ε per ogni $\varepsilon \in (0, 1)$; è un PTAS se il tempo di esecuzione è polinomiale nella dimensione dell'istanza per ogni ε fissato, e un FPTAS se è polinomiale anche in $\frac{1}{\varepsilon}$.

Quadro riassuntivo. Una garanzia limita, su ogni istanza, la distanza fra una soluzione euristica (un lower bound su $z(I)$ per un problema di massimo, un upper bound per un problema di minimo) e il valore ottimo. I quadri riassuntivi dei capitoli di questo volume raccolgono, per le istanze di riferimento, i valori degli algoritmi euristici e le loro garanzie; per il VCP, il MCP e il TSP generale non esiste alcuna garanzia costante, a meno che $P = NP$.

Problemi di zaino

Capitolo 2

Algoritmi di approssimazione per lo zaino

Prerequisiti

- Da questo modulo, il capitolo «Principi degli algoritmi di approssimazione»: rapporto di prestazione, garanzie di prestazione relative e assolute, tightness, schemi di approssimazione, PTAS e FPTAS.
- Dal modulo *Problemi di ottimizzazione combinatoria*, i capitoli «Problemi di zaino» e «Formulazioni dei problemi di zaino»: il KP, il rilassamento lineare $\mathrm{LP}_{\mathrm{KP}}$ e il bound di Dantzig, che usa l'oggetto critico.
- Dal modulo *Algoritmi euristici*, il capitolo «Euristiche costruttive»: l'algoritmo greedy.
- Dal modulo *Programmazione dinamica*, il capitolo «PD per il problema dello zaino 0/1»: la programmazione dinamica sui profitti, usata dall'algoritmo Basic-FPTAS.

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- eseguire a mano gli algoritmi greedy, Ext-Greedy, H^2, H^ℓ e Basic-FPTAS su un'istanza del KP;
- spiegare perché l'algoritmo greedy non ha una garanzia costante e perché Ext-Greedy ha garanzia 2;
- dimostrare le garanzie $\frac{4}{3}$ di H^2, $\frac{\ell+2}{\ell+1}$ di H^ℓ e $\frac{1}{1-\varepsilon}$ dell'algoritmo Basic-FPTAS;
- confrontare i tempi di calcolo di H^ℓ e dell'algoritmo Basic-FPTAS, e dire perché solo il secondo è un FPTAS;
- spiegare perché una garanzia di prestazione assoluta per il KP richiede un algoritmo esatto.

Il problema di questo capitolo è il problema dello zaino 0/1 (KP): la sua definizione e le sue ipotesi sono nella sua pagina.

2.1 Notazione

- Come nella pagina del problema, c è la capacità dello zaino e $\mathscr{I} = \{1, 2, \ldots, n\}$ l'insieme degli oggetti; in questo capitolo l'oggetto $i \in \mathscr{I}$ ha profitto p_i e peso w_i.

- Indichiamo con $z(I)$ il **valore ottimo** del KP sull'istanza I, e con $\mathscr{A}(I)$ il profitto della soluzione ammissibile restituita da un algoritmo euristico $\mathscr{A}$. Poiché ogni algoritmo euristico di questo capitolo restituisce una soluzione ammissibile,

$$\mathscr{A}(I) \leq z(I) \qquad \text{e} \qquad \frac{z(I)}{\mathscr{A}(I)} \geq 1$$

  per ogni istanza I del KP.

- Supponiamo sempre che gli oggetti siano ordinati per efficienza non crescente

$$\frac{p_i}{w_i} \geq \frac{p_{i+1}}{w_{i+1}} \qquad \text{per ogni} \qquad i \in \{1, 2, \ldots, n-1\},$$

  il che richiede tempo $O(n \log n)$, una funzione polinomiale del numero n di oggetti.

2.2 Oggetto critico e capacità residua

Definizione 2.1: oggetto critico e capacità residua

Dato un insieme $\mathscr{I} = \{1, 2, \ldots, n\}$ di n oggetti ordinati per efficienza non crescente, l'**oggetto critico** è l'oggetto

$$s \in \mathscr{I} \quad \text{tale che} \quad \sum_{i=1}^{s-1} w_i \leq c \quad \text{e} \quad \sum_{i=1}^{s} w_i > c \tag{2.1}$$

e la **capacità residua** è

$$rc = c - \sum_{i=1}^{s-1} w_i \tag{2.2}$$

- L'oggetto critico è il primo che non entra per intero nello zaino se tutti i precedenti sono selezionati.

- La capacità residua è la capacità dello zaino che rimane quando tutti gli oggetti che precedono l'oggetto critico sono selezionati.

- Ordinare gli oggetti per efficienza non crescente richiede tempo $O(n \log n)$, e l'oggetto critico si trova poi scorrendoli. L'oggetto critico si può trovare anche in tempo $O(n)$, senza ordinare, con una ricerca della mediana pesata: l'ordinamento è comodo, non necessario.

Esempio 2.1: oggetto critico e capacità residua

Consideriamo uno zaino di capacità $c = 9$ e un insieme $\mathscr{I} = \{1, 2, 3, 4, 5, 6, 7\}$ di 7 oggetti ($n = 7$), con i seguenti profitti e pesi:

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$
p_i	6	5	8	9	6	7	3
w_i	2	3	6	7	5	9	4

Le efficienze degli oggetti sono:

$$\frac{p_1}{w_1} = \frac{6}{2}, \quad \frac{p_2}{w_2} = \frac{5}{3}, \quad \frac{p_3}{w_3} = \frac{8}{6}, \quad \frac{p_4}{w_4} = \frac{9}{7}, \quad \frac{p_5}{w_5} = \frac{6}{5}, \quad \frac{p_6}{w_6} = \frac{7}{9}, \quad \frac{p_7}{w_7} = \frac{3}{4}.$$

Quindi gli oggetti sono ordinati per efficienza non crescente: (1, 2, 3, 4, 5, 6, 7). L'oggetto critico è $s = 3$ poiché:

$$\sum_{i=1}^{2} w_i = 2 + 3 = 5 \leq 9 \quad \text{e} \quad \sum_{i=1}^{3} w_i = 2 + 3 + 6 = 11 > 9$$

e la capacità residua è:

$$rc = c - \sum_{i=1}^{2} w_i = 9 - 2 - 3 = 4$$

2.3 L'algoritmo greedy

- Supponiamo che gli oggetti siano ordinati per efficienza non crescente

$$\frac{p_i}{w_i} \geq \frac{p_{i+1}}{w_{i+1}} \quad \text{per ogni} \quad i \in \{1, 2, \ldots, n-1\}$$

L'algoritmo *greedy* (indicato con GR) è un'euristica costruttiva per il KP. Costruisce una soluzione ammissibile esaminando gli oggetti in ordine di *efficienza* non crescente, dove l'efficienza è il rapporto fra profitto e peso. Partendo da un sottoinsieme vuoto di oggetti, ogni oggetto viene aggiunto alla soluzione se e solo se il vincolo di capacità non viene violato.

- Lo pseudocodice di GR è il seguente:

Algoritmo

Algorithm: Greedy

Input: istanza I del KP con n oggetti, profitti $p[1..n]$, pesi $w[1..n]$, capacità c
Output: soluzione ammissibile $x[1..n]$ del KP

```
1  Function Greedy(n, p, w, c)
2      x := nuovo vettore[1..n]                          // oggetti selezionati
3      for i := 1 to n do
4          x[i] ← 0                  // all'inizio l'oggetto i non è selezionato
5      end
6      rc := c                                           // capacità residua
7      for i := 1 to n do
8          if w[i] ≤ rc then
9              x[i] ← 1                                  // selezionare l'oggetto i
10             rc ← rc − w[i]                // aggiornare la capacità residua
11         end
12     end
13     return x
```

- Dato l'oggetto critico s, GR seleziona tutti gli oggetti che precedono l'oggetto critico, cioè

$$\{1, 2, \ldots, s-1\} \subseteq S \qquad \text{e} \qquad GR(I) \geq \sum_{i=1}^{s-1} p_i.$$

  Inoltre, l'oggetto s non viene selezionato da GR, mentre GR può selezionare anche alcuni oggetti che seguono l'oggetto critico.

Esempio 2.2: Esecuzione di GR

Per l'istanza del KP dell'Esempio 2.1, gli oggetti sono ordinati per efficienza non crescente:

$$(1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7).$$

L'esecuzione di GR procede come segue:

1. L'oggetto 1 (peso $w_1 = 2$) viene selezionato. La capacità disponibile rc diventa $9 - 2 = 7$.
2. L'oggetto 2 (peso $w_2 = 3$) viene selezionato. La capacità disponibile rc diventa $7 - 3 = 4$.
3. L'oggetto 3 (peso $w_3 = 6$) non viene selezionato perché non entra nella capacità disponibile $rc = 4$.
4. L'oggetto 4 (peso $w_4 = 7$) non viene selezionato perché non entra nella capacità disponibile $rc = 4$.
5. L'oggetto 5 (peso $w_5 = 5$) non viene selezionato perché non entra nella capacità disponibile $rc = 4$.

6. L'oggetto 6 (peso $w_6 = 9$) non viene selezionato perché non entra nella capacità disponibile $rc = 4$.

7. L'oggetto 7 (peso $w_7 = 4$) viene selezionato. La capacità disponibile rc diventa $4 - 4 = 0$.

Quindi GR restituisce la soluzione ammissibile:

$$S = \{1, 2, 7\}.$$

Il profitto totale della soluzione ammissibile è:

$$GR(I) = \sum_{i \in S} p_i = p_1 + p_2 + p_7 = 6 + 5 + 3 = 14.$$

Pertanto, 14 è un lower bound valido sul valore ottimo $z(I)$:

$$14 \leq z(I).$$

In questa istanza l'oggetto critico è $s = 3$. Di conseguenza, GR seleziona tutti gli oggetti che precedono l'oggetto critico, cioè gli oggetti 1 e 2, non seleziona l'oggetto 3 (l'oggetto critico), e poi seleziona anche l'oggetto 7, che segue l'oggetto critico.

Esempio 2.3: Esecuzione di GR

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

	$i = 1$	$i = 2$	$i = 3$	$i = 4$
p_i	6	8	19	4
w_i	2	3	9	5

Le efficienze degli oggetti sono:

$$\frac{p_1}{w_1} = \frac{6}{2}, \quad \frac{p_2}{w_2} = \frac{8}{3}, \quad \frac{p_3}{w_3} = \frac{19}{9}, \quad \frac{p_4}{w_4} = \frac{4}{5}.$$

Quindi gli oggetti sono ordinati per efficienza non crescente: (1, 2, 3, 4). L'oggetto critico è $s = 3$ poiché:

$$\sum_{i=1}^{2} w_i = 2 + 3 = 5 \leq 10 \quad \text{e} \quad \sum_{i=1}^{3} w_i = 2 + 3 + 9 = 14 > 10$$

e la capacità residua è:

$$rc = c - \sum_{i=1}^{2} w_i = 10 - 2 - 3 = 5.$$

L'esecuzione di GR procede come segue:

1. L'oggetto 1 (peso $w_1 = 2$) viene selezionato. La capacità disponibile rc diventa $10 - 2 = 8$.

2. L'oggetto 2 (peso $w_2 = 3$) viene selezionato. La capacità disponibile rc diventa $8 - 3 = 5$.

3. L'oggetto 3 (peso $w_3 = 9$) non viene selezionato perché non entra nella capacità disponibile $rc = 5$.

4. L'oggetto 4 (peso $w_4 = 5$) viene selezionato. La capacità disponibile rc diventa $5 - 5 = 0$.

Quindi GR restituisce la soluzione ammissibile:

$$S = \{1, 2, 4\}.$$

Il profitto totale della soluzione ammissibile è:

$$GR(I) = \sum_{i \in S} p_i = p_1 + p_2 + p_4 = 6 + 8 + 4 = 18.$$

Pertanto, 18 è un lower bound valido sul valore ottimo $z(I)$:

$$18 \leq z(I).$$

In questa istanza l'oggetto critico è $s = 3$. Di conseguenza, GR seleziona tutti gli oggetti che precedono l'oggetto critico, cioè gli oggetti 1 e 2, non seleziona l'oggetto 3 (l'oggetto critico), e poi seleziona l'oggetto 4, che segue l'oggetto critico. L'oggetto 3 ha il profitto massimo ($p_3 = 19$) e selezionarlo da solo è ottimo, ma GR non lo seleziona.

Esempio 2.4: Esecuzione di *GR*

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

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
p_i	4	6	5	12	3
w_i	1	2	2	6	2

Le efficienze degli oggetti sono:

$$\frac{p_1}{w_1} = \frac{4}{1} = 4, \quad \frac{p_2}{w_2} = \frac{6}{2} = 3, \quad \frac{p_3}{w_3} = \frac{5}{2} = 2.5, \quad \frac{p_4}{w_4} = \frac{12}{6} = 2, \quad \frac{p_5}{w_5} = \frac{3}{2} = 1.5.$$

Quindi gli oggetti sono ordinati per efficienza non crescente: (1, 2, 3, 4, 5). L'oggetto critico è $s = 4$ poiché:

$$\sum_{i=1}^{3} w_i = 1 + 2 + 2 = 5 \leq 10 \quad \text{e} \quad \sum_{i=1}^{4} w_i = 1 + 2 + 2 + 6 = 11 > 10$$

e la capacità residua è:

$$rc = c - \sum_{i=1}^{3} w_i = 10 - 5 = 5.$$

L'esecuzione di GR procede come segue:

1. L'oggetto 1 (peso $w_1 = 1$) viene selezionato. La capacità disponibile rc diventa $10 - 1 = 9$.
2. L'oggetto 2 (peso $w_2 = 2$) viene selezionato. La capacità disponibile rc diventa $9 - 2 = 7$.
3. L'oggetto 3 (peso $w_3 = 2$) viene selezionato. La capacità disponibile rc diventa $7 - 2 = 5$.
4. L'oggetto 4 (peso $w_4 = 6$) non viene selezionato perché non entra nella capacità disponibile $rc = 5$.
5. L'oggetto 5 (peso $w_5 = 2$) viene selezionato. La capacità disponibile rc diventa $5 - 2 = 3$.

Quindi GR restituisce la soluzione ammissibile:

$$S = \{1, 2, 3, 5\}.$$

Il profitto totale della soluzione ammissibile è:

$$GR(I) = \sum_{i \in S} p_i = p_1 + p_2 + p_3 + p_5 = 4 + 6 + 5 + 3 = 18.$$

Pertanto, 18 è un lower bound valido sul valore ottimo $z(I)$:

$$18 \leq z(I).$$

In questa istanza l'oggetto critico è $s = 4$. Di conseguenza, GR seleziona gli oggetti 1, 2 e 3 (che precedono l'oggetto critico), salta l'oggetto 4 (l'oggetto critico) e poi seleziona l'oggetto 5. La soluzione ottima è $\tilde{S} = \{2, 3, 4\}$ di valore $z(I) = 6 + 5 + 12 = 23$, migliore della soluzione trovata da GR.

Esempio 2.5: Esecuzione di *GR*

Consideriamo uno zaino di capacità $c = 21$ e un insieme $\mathscr{I} = \{1, 2, 3, 4, 5, 6\}$ di 6 oggetti ($n = 6$), con i seguenti profitti e pesi:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
p_i	4	6	4	3	2	20
w_i	1	2	2	2	2	21

Le efficienze degli oggetti sono:

$$\frac{p_1}{w_1} = \frac{4}{1} = 4, \quad \frac{p_2}{w_2} = \frac{6}{2} = 3, \quad \frac{p_3}{w_3} = \frac{4}{2} = 2, \quad \frac{p_4}{w_4} = \frac{3}{2} = 1.5, \quad \frac{p_5}{w_5} = \frac{2}{2} = 1, \quad \frac{p_6}{w_6} = \frac{20}{21}.$$

Quindi gli oggetti sono ordinati per efficienza non crescente: (1, 2, 3, 4, 5, 6). L'oggetto critico è $s = 6$ poiché:

$$\sum_{i=1}^{5} w_i = 1 + 2 + 2 + 2 + 2 = 9 \leq 21 \quad \text{e} \quad \sum_{i=1}^{6} w_i = 1 + 2 + 2 + 2 + 2 + 21 = 30 > 21$$

e la capacità residua è:

$$rc = c - \sum_{i=1}^{5} w_i = 21 - 9 = 12.$$

L'esecuzione di GR procede come segue:

1. L'oggetto 1 (peso $w_1 = 1$) viene selezionato. La capacità disponibile rc diventa $21 - 1 = 20$.
2. L'oggetto 2 (peso $w_2 = 2$) viene selezionato. La capacità disponibile rc diventa $20 - 2 = 18$.
3. L'oggetto 3 (peso $w_3 = 2$) viene selezionato. La capacità disponibile rc diventa $18 - 2 = 16$.
4. L'oggetto 4 (peso $w_4 = 2$) viene selezionato. La capacità disponibile rc diventa $16 - 2 = 14$.
5. L'oggetto 5 (peso $w_5 = 2$) viene selezionato. La capacità disponibile rc diventa $14 - 2 = 12$.
6. L'oggetto 6 (peso $w_6 = 21$) non viene selezionato perché non entra nella capacità disponibile $rc = 12$.

Quindi GR restituisce la soluzione ammissibile:

$$S = \{1, 2, 3, 4, 5\}.$$

Il profitto totale della soluzione ammissibile è:

$$GR(I) = \sum_{i \in S} p_i = p_1 + p_2 + p_3 + p_4 + p_5 = 4 + 6 + 4 + 3 + 2 = 19.$$

Pertanto, 19 è un lower bound valido sul valore ottimo $z(I)$:

$$19 \leq z(I).$$

In questa istanza l'oggetto di profitto massimo è l'oggetto 6, con $p_6 = 20$, che compare per ultimo nell'ordine per efficienza. L'algoritmo greedy GR seleziona tutti gli oggetti da 1 a 5 (che precedono l'oggetto critico), ma non seleziona l'oggetto 6 (l'oggetto critico stesso) perché il suo peso $w_6 = 21$ supera la capacità residua $rc = 12$. Da solo l'oggetto 6 entrerebbe nello zaino (poiché $w_6 = 21 \leq c = 21$) e darebbe un profitto di 20, migliore della soluzione greedy. La soluzione ottima è $\tilde{S} = \{6\}$ di valore $z(I) = 20$.

2.4 L'algoritmo Ext-Greedy

L'algoritmo *Ext-Greedy* (indicato con GR^e) è un'euristica costruttiva per il KP. Prima calcola una soluzione ammissibile eseguendo l'algoritmo greedy GR. Poi confronta questa soluzione con la migliore soluzione ammissibile formata da un solo oggetto. Infine restituisce la migliore delle due soluzioni.

- Lo pseudocodice di GR^e è il seguente:

Algoritmo

Algorithm: Extended Greedy

Input: istanza I del KP con n oggetti, profitti $p[1..n]$, pesi $w[1..n]$, capacità c
Output: soluzione ammissibile $x[1..n]$ del KP

```
1  Function ExtendedGreedy(n, p, w, c)
2      x := Greedy(n, p, w, c)                          // soluzione greedy
3      z := 0                                  // valore della soluzione greedy
4      for i := 1 to n do
5          if x[i] = 1 then
6              z ← z + p[i]      // sommare il profitto di un oggetto selezionato
7          end
8      end
9      i_m := 1                                        // miglior oggetto singolo
10     p_max := p[1]
11     for i := 2 to n do
12         if p[i] > p_max then
13             i_m ← i                  // aggiornare il miglior oggetto singolo
14             p_max ← p[i]                          // aggiornare il suo profitto
15         end
16     end
17     x̄ := nuovo vettore[1..n]        // migliore soluzione con un solo oggetto
18     for i := 1 to n do
19         x̄[i] ← 0
20     end
21     x̄[i_m] ← 1                                  // selezionare solo l'oggetto i_m
22     if z < p_max then
23         for i := 1 to n do
24             x[i] ← x̄[i]                  // copiare componente per componente
25         end
26     end
27     return x
```

- Sia $p_{\max}$ il **profitto massimo**

$$p_{\max} = \max \{\ p_i :\ i \in \mathscr{I}\ \}.$$

e sia $\bar{S}$ la **migliore soluzione ammissibile con un solo oggetto**

$$\bar{S} = \{\ i_m\ \} \quad \text{e} \quad i_m \in \arg\max \{\ p_i :\ i \in \mathscr{I}\ \}$$

Esempio 2.6: Esecuzione di GR^e

Partiamo dalla soluzione ammissibile calcolata da GR sull'istanza del KP dell'Esempio 2.1 (Esempio 2.2):

$$S = \{1, 2, 7\}, \qquad GR(I) = \sum_{i \in S} p_i = p_1 + p_2 + p_7 = 6 + 5 + 3 = 14.$$

Calcoliamo ora il profitto massimo di un singolo oggetto:

$$p_{\max} = \max \{\ p_i :\ i \in \mathscr{I}\ \} = p_4 = 9, \qquad \bar{S} = \{4\}.$$

Poiché $14 \geq 9$, GR^e mantiene la soluzione greedy e restituisce

$$S = \{1, 2, 7\}.$$

Esempio 2.7: Esecuzione di GR^e

Partiamo dalla soluzione ammissibile calcolata da GR sull'istanza del KP dell'Esempio 2.3:

$$S = \{1, 2, 4\}, \qquad GR(I) = \sum_{i \in S} p_i = p_1 + p_2 + p_4 = 6 + 8 + 4 = 18.$$

Calcoliamo ora il profitto massimo di un singolo oggetto:

$$p_{\max} = \max \{\ p_i :\ i \in \mathscr{I}\ \} = p_3 = 19, \qquad \bar{S} = \{3\}.$$

Poiché $18 < 19$, GR^e sostituisce la soluzione greedy con la migliore soluzione ammissibile con un solo oggetto e restituisce

$$S = \{3\}.$$

- Dato l'oggetto critico s, abbiamo

$$GR^e(I) \geq GR(I) \geq \sum_{i=1}^{s-1} p_i.$$

Inoltre, per definizione di GR^e, abbiamo

$$GR^e(I) \geq p_{\max}$$

- Per ogni istanza I del KP, i valori obiettivo soddisfano

$$GR(I) \;\leq\; GR^e(I) \;\leq\; z(I).$$

Questo vale perché GR^e restituisce la migliore fra la soluzione greedy e la migliore soluzione ammissibile con un solo oggetto, quindi non può essere peggiore di GR, e il valore di ogni soluzione ammissibile è al più il valore ottimo $z(I)$.

2.5 L'algoritmo H^2

L'algoritmo H^2 è un'euristica costruttiva per il KP; è l'algoritmo chiamato $G^{3/4}$ in letteratura. Enumera tutte le coppie ammissibili di oggetti (i,k) e "scommette" che i due oggetti appartengano a una soluzione ottima. Per ogni coppia, fissa i e k nello zaino, costruisce un'istanza residua con la capacità rimanente e tenendo solo gli oggetti di profitto al più $\min\{p_i, p_k\}$, e poi completa la soluzione parziale eseguendo l'algoritmo Ext-Greedy GR^e sull'istanza residua. Infine restituisce la migliore soluzione trovata su tutte le coppie, confrontandola anche con la migliore soluzione ammissibile con un solo oggetto.

Algoritmo

Algorithm: H^2

Input: istanza I del KP con insieme di oggetti $\mathscr{I} = \{1, 2, \ldots, n\}$, profitti $p[1..n]$, pesi $w[1..n]$, capacità c

Output: valore ammissibile $H^2(I)$ del KP

Function *H2*$(\mathscr{I}, p, w, c)$
  $p_{\max} := \max\{p[j] : j \in \mathscr{I}\}$
  $lb := p_{\max}$ `// miglior valore trovato`
  $\mathscr{P} := \{(i,k) \in \mathscr{I} \times \mathscr{I} : i < k,\ w[i] + w[k] \leq c\}$
  foreach (i,k) ***in*** $\mathscr{P}$ **do**
    $rc := c - w[i] - w[k]$ `// capacità residua`
    $\mathscr{I}(i,k) := \{j \in \mathscr{I} \setminus \{i,k\} : p[j] \leq \min\{p[i], p[k]\},\ w[j] \leq rc\}$
    I_{ik} := istanza del KP ristretta a $\mathscr{I}(i,k)$ con capacità rc
    $z_{res} := GR^e(I_{ik})$ `// valore di` GR^e `sull'istanza residua`
    $lb_{ik} := p[i] + p[k] + z_{res}$
    if $lb_{ik} > lb$ **then**
      $lb \leftarrow lb_{ik}$
    end
  end
  return lb

Definizione 2.2: coppie ordinate ammissibili di oggetti

Data un'istanza del KP con insieme di oggetti $\mathscr{I} = \{1, 2, \ldots, n\}$ e capacità dello zaino c, l'**insieme di tutte le coppie ordinate ammissibili di oggetti** è

$$\mathscr{P} = \left\{ \ (i,k) \in \mathscr{I} \times \mathscr{I} : \ i < k, \quad w_i + w_k \leq c \ \right\}. \tag{2.3}$$

- L'algoritmo H^2 si basa sulla seguente idea: una soluzione ottima o è formata da un solo oggetto, quello di profitto massimo, o contiene (almeno) due oggetti di profitto "grande". L'algoritmo quindi *scommette* sui due oggetti di profitto maggiore di una soluzione ottima, enumerando tutte le coppie ammissibili in $\mathscr{P}$, e completa ogni scommessa su un'istanza residua.

- L'algoritmo H^2 considera le coppie di oggetti $(i,k) \in \mathscr{P}$, una alla volta, ed esegue tre operazioni:

  1. La **prima operazione** consiste nell'inserire gli oggetti i e k nello zaino. Si ottiene così una **soluzione parziale** di profitto $p_i + p_k$ e una **capacità residua** $c - w_i - w_k$.
  2. La **seconda operazione** è la costruzione di un'**istanza residua** (indicata con $I(i,k)$) che contiene solo gli oggetti di profitto minore o uguale ai profitti dei due oggetti inseriti. L'**insieme residuo di oggetti** è:

  $$\mathscr{I}(i,k) = \left\{ j \in \mathscr{I} \setminus \{i,k\} : \quad p_j \leq \min\{p_i, p_k\}, \quad w_j \leq c - w_i - w_k \right\}$$

     Vengono scartati anche gli oggetti di peso maggiore della capacità residua.
  3. La **terza operazione** consiste nel completare la soluzione parziale eseguendo l'algoritmo GR^e sull'istanza residua $I(i,k)$.

- Per ogni coppia di oggetti $(i,k) \in \mathscr{P}$, abbiamo il lower bound:

$$lb(i,k) = p_i + p_k + GR^e\big(I(i,k)\big)$$

  il lower bound globale prodotto dall'algoritmo H^2 è:

$$H^2(I) = \max\left\{ \ \max_{(i,k)\in\mathscr{P}} \left\{ \ lb(i,k) \ \right\}, \ \ p_{\max} \right\}$$

Questo algoritmo euristico si può vedere come una "scommessa" sui due oggetti di profitto maggiore di una soluzione ottima (o sull'oggetto di profitto maggiore).

Esempio 2.8: Algoritmo H^2

Consideriamo un insieme $\mathscr{I} = \{1, 2, 3, 4, 5, 6, 7\}$ di 7 oggetti ($n = 7$) con i seguenti profitti e pesi:

oggetto	p_j	w_j	$\frac{p_j}{w_j}$	efficienza
$j=1$	6	2	$\frac{6}{2}$	$=3$
$j=2$	5	3	$\frac{5}{3}$	≈ 1.66
$j=3$	8	6	$\frac{8}{6}$	≈ 1.33
$j=4$	9	7	$\frac{9}{7}$	≈ 1.28
$j=5$	6	5	$\frac{6}{5}$	$=1.2$
$j=6$	7	9	$\frac{7}{9}$	≈ 0.77
$j=7$	3	4	$\frac{3}{4}$	$=0.75$

e una capacità dello zaino $c = 12$, per cui $p_{\max} = 9$. La soluzione ottima è $\tilde{S} = \{1, 2, 4\}$, di peso $2 + 3 + 7 = 12$ e valore $z(I) = 6 + 5 + 9 = 20$. L'oggetto critico è $s = 4$, e GR^e restituisce la soluzione $\{1, 2, 3\}$ con $GR^e(I) = 6 + 5 + 8 = 19$.

Esecuzione di H^2 (inizializzazione $lb = p_{\max} = 9$):

- Coppia $(1, 2)$: la capacità residua è $rc = 12-2-3 = 7$, $\mathscr{I}(1, 2) = \{7\}$ e $GR^e\big(I(1, 2)\big) = 3$, quindi $lb(1, 2) = 6 + 5 + 3 = 14 > lb$ e lb viene aggiornato a 14.
- Coppia $(1, 3)$: la capacità residua è $rc = 12 - 2 - 6 = 4$, $\mathscr{I}(1, 3) = \{2, 7\}$ e $GR^e\big(I(1, 3)\big) = 5$, quindi $lb(1, 3) = 6 + 8 + 5 = 19 > lb$ e lb viene aggiornato a 19.
- Coppia $(1, 4)$: la capacità residua è $rc = 12-2-7 = 3$, $\mathscr{I}(1, 4) = \{2\}$ e $GR^e\big(I(1, 4)\big) = 5$, quindi $lb(1, 4) = 6 + 9 + 5 = 20 > lb$ e lb viene aggiornato a 20.
- Coppia $(1, 5)$: la capacità residua è $rc = 12 - 2 - 5 = 5$, $\mathscr{I}(1, 5) = \{2, 7\}$ e $GR^e\big(I(1, 5)\big) = 5$, quindi $lb(1, 5) = 6 + 6 + 5 = 17 < lb$.
- Coppia $(1, 6)$: la capacità residua è $rc = 12 - 2 - 9 = 1$, $\mathscr{I}(1, 6) = \varnothing$, quindi $lb(1, 6) = 6 + 7 = 13 < lb$.
- Coppia $(1, 7)$: la capacità residua è $rc = 12 - 2 - 4 = 6$, $\mathscr{I}(1, 7) = \varnothing$, quindi $lb(1, 7) = 6 + 3 = 9 < lb$.
- Le coppie (i, k) con $i \geq 2$ si trattano allo stesso modo, e nessuna dà un valore maggiore di 20. Quindi $H^2(I) = 20 = z(I)$: su questa istanza H^2 trova la soluzione ottima, che GR^e non trova.

Proposizione 2.3

L'algoritmo H^2 si può calcolare in tempo $O(n^3)$, e in tempo $O(n^2)$ con un'implementazione più accurata.

- Non diamo una dimostrazione formale, ma solo le idee principali.
- Il tempo di calcolo dell'algoritmo H^2 è determinato essenzialmente dall'esecuzione dell'algoritmo GR^e, che richiede tempo $O(n)$, per ogni coppia di oggetti. Il numero di esecuzioni di GR^e è $\binom{n}{2}$ che è in $O(n^2)$, quindi il tempo di calcolo totale è $O(n^3)$.

- Con un'implementazione più accurata si può ridurre a $O(n^2)$.

2.6 L'algoritmo H^ℓ

L'algoritmo H^ℓ è un'euristica costruttiva per il KP. Prima calcola la migliore soluzione ammissibile formata da al più $\ell - 1$ oggetti. Poi enumera tutti gli insiemi ammissibili L di esattamente ℓ oggetti e "scommette" che siano gli ℓ oggetti di profitto maggiore di una soluzione ottima: per ogni insieme, inserisce gli oggetti di L, costruisce un'istanza residua con la capacità rimanente e gli oggetti di profitto al più $\min\{p_i : i \in L\}$, e completa la soluzione parziale eseguendo l'algoritmo Ext-Greedy GR^e sull'istanza residua. Infine restituisce la migliore soluzione trovata.

Algoritmo

Algorithm: H^ℓ

Input: istanza I del KP con insieme di oggetti $\mathscr{I} = \{1, 2, \ldots, n\}$, profitti $p[1..n]$, pesi $w[1..n]$, capacità c, intero ℓ con $2 \le \ell \le n$

Output: valore ammissibile $H^\ell(I)$ del KP

Function $Hl(\mathscr{I}, p, w, c, \ell)$
  $lb := \max\left\{\sum_{j\in S} p[j] : S \in \widehat{T}(\ell)\right\}$ `// insiemi ammissibili di al più` $\ell - 1$ `oggetti`
  foreach L ***in*** $T(\ell)$ `// insiemi ammissibili di esattamente` ℓ `oggetti`
  do
    $rc := c - \sum_{i\in L} w[i]$ `// capacità residua`
    $\mathscr{I}(L) := \left\{j \in \mathscr{I} \setminus L : p[j] \le \min_{i\in L} p[i],\ w[j] \le rc\right\}$
    $lb_L := \sum_{i\in L} p[i] + GR^e\big(I(L)\big)$ `// completare la scommessa`
    if $lb_L > lb$ **then**
      $lb \leftarrow lb_L$
    end
  end
  return lb

- L'algoritmo H^2 scommette sui *due* oggetti di profitto maggiore di una soluzione ottima. L'algoritmo H^ℓ generalizza questa idea: scommette sugli ℓ oggetti di profitto maggiore, per un intero fissato ℓ con $2 \le \ell \le n$. Per $\ell = 2$ è esattamente l'algoritmo H^2.

- Definiamo le famiglie di tutti gli insiemi ammissibili di esattamente ℓ oggetti e di al più $\ell - 1$ oggetti:

$$T(\ell) = \left\{ L \subseteq \mathscr{I} : \sum_{j\in L} w_j \le c,\ \ |L| = \ell \right\}$$

e

$$\widehat{T}(\ell) = \left\{ L \subseteq \mathscr{I} : \quad \sum_{j \in L} w_j \leq c, \quad |L| \leq \ell - 1 \right\}$$

- Per ogni insieme $L \in T(\ell)$ l'algoritmo esegue tre operazioni:
  1. inserisce nello zaino tutti gli oggetti $i \in L$, ottenendo una **soluzione parziale** di profitto $\sum_{i \in L} p_i$ e una **capacità residua** $c - \sum_{i \in L} w_i$;
  2. costruisce l'**istanza residua** $I(L)$, il cui **insieme residuo di oggetti** è
  $$\mathscr{I}(L) = \left\{ j \in \mathscr{I} \setminus L : \quad p_j \leq \min\{p_i : i \in L\}, \quad w_j \leq c - \sum_{i \in L} w_i \right\};$$
  3. completa la soluzione parziale eseguendo GR^e su $I(L)$, ottenendo il valore
  $$lb(L) = \sum_{i \in L} p_i + GR^e\big(I(L)\big).$$

- Il valore restituito dall'algoritmo è
$$H^\ell(I) = \max \left\{ \max_{L \in T(\ell)} lb(L), \quad \max_{S \in \widehat{T}(\ell)} \sum_{j \in S} p_j \right\}.$$

Esempio 2.9: Esecuzione di H^ℓ

Consideriamo uno zaino di capacità $c = 12$ e un insieme $\mathscr{I} = \{1, 2, 3, 4, 5\}$ di 5 oggetti ($n = 5$), con i seguenti profitti e pesi:

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
p_i	2	3	3	3	3
w_i	1	3	3	3	3

Gli oggetti sono ordinati per efficienza non crescente, poiché $\frac{2}{1} = 2 > \frac{3}{3} = 1$. La soluzione ottima inserisce i quattro oggetti di peso 3, con $z(I) = 12$.

Con $\ell = 2$, cioè l'algoritmo H^2 della Sezione 2.5. Una coppia di due oggetti di profitto 3, per esempio $L = \{2, 3\}$, lascia capacità residua 6 e l'insieme residuo $\mathscr{I}(L) = \{1, 4, 5\}$: GR^e inserisce l'oggetto 1 e l'oggetto 4, di profitto $2 + 3 = 5$, quindi $lb(L) = 3 + 3 + 5 = 11$. Una coppia che contiene l'oggetto 1 lascia un insieme residuo vuoto, poiché nessun altro oggetto ha profitto al più 2. Quindi $H^2(I) = 11 < 12$.

Con $\ell = 3$. Un insieme di tre oggetti di profitto 3, per esempio $L = \{2, 3, 4\}$, lascia capacità residua 3 e l'insieme residuo $\mathscr{I}(L) = \{1, 5\}$: GR^e restituisce la migliore fra l'oggetto 1 da solo (profitto 2, dopo il quale l'oggetto 5 non entra più) e l'oggetto 5 da solo (profitto 3), quindi $lb(L) = 9 + 3 = 12$. Quindi $H^3(I) = 12 = z(I)$.

Scommettere su un oggetto in più basta per trovare la soluzione ottima su questa istanza, sulla quale H^2 non la trova.

Osservazione 2.4

L'algoritmo H^ℓ si può calcolare in tempo $O(n^{\ell+1})$.

- Il tempo di calcolo è determinato dalle esecuzioni di GR^e, che richiede tempo $O(n)$, una per ogni insieme in $T(\ell)$. Questi insiemi sono al più $\binom{n}{\ell}$, che è in $O(n^\ell)$, quindi il tempo di calcolo totale è $O(n^{\ell+1})$. Con un'implementazione più accurata si può ridurre a $O(n^\ell)$.
- Oltre alla memorizzazione dei dati, non si usa memoria aggiuntiva significativa, e lo spazio totale richiesto è solo $O(n)$.

2.7 L'algoritmo Basic-FPTAS

- L'algoritmo H^ℓ enumera $O(n^\ell)$ insiemi di oggetti, un numero che cresce molto rapidamente con ℓ. L'algoritmo Basic-FPTAS segue una strada del tutto diversa: invece di scommettere sugli oggetti, arrotonda i profitti e risolve esattamente l'istanza arrotondata.

L'algoritmo *Basic-FPTAS* (indicato con FP^ε) sostituisce i profitti degli oggetti con profitti scalati, che assumono pochi valori distinti, e poi risolve l'istanza scalata *all'ottimo* con la programmazione dinamica sui profitti. La soluzione ottima dell'istanza scalata viene restituita come soluzione ammissibile dell'istanza originale, in generale non ottima.

Algoritmo

Algorithm: Basic-FPTAS

Input: istanza I del KP con n oggetti, profitti $p[1..n]$, pesi $w[1..n]$, capacità c, accuratezza $\varepsilon \in (0,1)$

Output: soluzione ammissibile $x[1..n]$ del KP

Function *BasicFPTAS*$(n, p, w, c, \varepsilon)$
  $lb := GR^e(I)$ `// valore della soluzione Ext-Greedy, un lower bound su` $z(I)$
  $K := \varepsilon \cdot lb/n$ `// fattore di scala`
  for $j := 1$ **to** n **do**
    $p'[j] \leftarrow \lfloor p[j]/K \rfloor$ `// profitto scalato`
  end
  $Q := \lfloor 2n/\varepsilon \rfloor$ `// upper bound sul valore ottimo dell'istanza scalata`
  $x := DPProfits(n, p', w, c, Q)$ `// istanza scalata risolta all'ottimo, tabella troncata a` Q
  return x

- Il **fattore di scala** è
$$K = \frac{\varepsilon\, GR^e(I)}{n},$$
e i **profitti scalati** si ottengono arrotondando per difetto i profitti a un multiplo di K:
$$p'_j = \left\lfloor \frac{p_j}{K} \right\rfloor \qquad \text{per ogni} \qquad j \in \mathscr{I}.$$
L'istanza scalata I' ha gli stessi pesi e la stessa capacità di I, quindi I e I' hanno esattamente le stesse soluzioni ammissibili.

- **Si scalano i profitti, non i pesi.** Arrotondare i pesi per difetto farebbe sembrare ammissibili soluzioni inammissibili, e arrotondarli per eccesso scarterebbe soluzioni ammissibili: in entrambi i casi l'istanza scalata risponderebbe a una domanda diversa. Arrotondare i profitti per difetto, invece, lascia intatto l'insieme delle soluzioni ammissibili e altera solo la funzione obiettivo.

- **L'istanza scalata si risolve con la programmazione dinamica sui profitti**, il secondo algoritmo di PD del capitolo «PD per il problema dello zaino 0/1» del modulo *Programmazione dinamica*: $u(j,q)$ è il peso totale minimo di un sottoinsieme di $\{1,2,\ldots,j\}$ di profitto totale esattamente q, e il valore ottimo è il massimo q con $u(n,q) \le c$.

- La tabella è troncata a $Q = \lfloor 2n/\varepsilon \rfloor$. Questo valore non è minore del valore ottimo dell'istanza scalata, che è al più $\frac{z(I)}{K} \le \frac{2\,GR^e(I)}{K} = \frac{2n}{\varepsilon}$ per la garanzia $z(I) \le 2\,GR^e(I)$ di GR^e. La tabella ha dimensione $n \times Q$, e il tempo di calcolo è $O(n\,Q) = O(n^2/\varepsilon)$.

Esempio 2.10: Esecuzione di FP^{ε}

Consideriamo uno zaino di capacità $c = 8$ e un insieme $\mathscr{I} = \{1, 2, 3, 4\}$ di 4 oggetti ($n = 4$), con i seguenti profitti e pesi:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$
p_i	19	19	14	14
w_i	2	4	3	3

Gli oggetti sono ordinati per efficienza non crescente, poiché $\frac{19}{2} = 9.5 > \frac{19}{4} = 4.75 > \frac{14}{3} \approx 4.67$. Eseguiamo FP^{ε} con $\varepsilon = \frac{1}{2}$.

Lower bound. GR inserisce l'oggetto 1 (capacità residua $8-2=6$) e l'oggetto 2 (capacità residua $6-4=2$), mentre gli oggetti 3 e 4 non entrano, quindi $GR(I) = 19+19 = 38$. Il miglior oggetto singolo ha profitto $19 \leq 38$, quindi

$$GR^{e}(I) = 38.$$

Scalatura. Il fattore di scala e i profitti scalati sono

$$K = \frac{\varepsilon\, GR^{e}(I)}{n} = \frac{\frac{1}{2} \cdot 38}{4} = 4.75, \qquad p'_1 = p'_2 = \left\lfloor \frac{19}{4.75} \right\rfloor = 4, \qquad p'_3 = p'_4 = \left\lfloor \frac{14}{4.75} \right\rfloor = 2.$$

Il profitto scalato totale è $P' = 4+4+2+2 = 12$, non maggiore di $Q = \lfloor 2n/\varepsilon \rfloor = 16$, quindi qui il troncamento non interviene.

Programmazione dinamica sui profitti. Sull'istanza scalata, i pesi minimi $u(4, q)$ sono

q	0	2	4	6	8	10	12
$u(4,q)$	0	3	2	5	6	9	12

Il massimo q con $u(4, q) \leq c = 8$ è $q = 8$, ottenuto con gli oggetti $\{1, 2\}$ di peso totale 6.

Soluzione restituita. FP^{ε} restituisce $S = \{1, 2\}$, di profitto effettivo

$$FP^{\varepsilon}(I) = 19 + 19 = 38.$$

La soluzione ottima è $\tilde{S} = \{1, 3, 4\}$, di peso $2+3+3 = 8$ e valore $z(I) = 19+14+14 = 47$: la scalatura ha reso gli oggetti 1 e 2 buoni quanto gli oggetti 1, 3 e 4 insieme. Il rapporto di prestazione è

$$\frac{z(I)}{FP^{\varepsilon}(I)} = \frac{47}{38} \approx 1.24.$$

2.8 Garanzie di prestazione

- Determiniamo ora il rapporto di prestazione nel caso peggiore degli algoritmi delle sezioni precedenti. Ricordiamo che, essendo il KP un problema di massimo, il rapporto di prestazione di un algoritmo $\mathscr{A}$ su un'istanza I è $z(I)/\mathscr{A}(I) \geq 1$, e che $r(\mathscr{A})$ è il suo estremo superiore su tutte le istanze.

2.8.1 Il rapporto di prestazione nel caso peggiore di GR

Proposizione 2.5

Esistono istanze I del KP tali che:

$$\frac{z(I)}{GR(I)} \to +\infty \qquad \text{per } c \to +\infty. \tag{2.4}$$

Dimostrazione. Fissiamo un intero $\vartheta > 2$ e consideriamo l'istanza I_ϑ del KP con capacità dello zaino $c = \vartheta$ e l'insieme di oggetti $\mathscr{I} = \{1, 2\}$, dove i profitti e i pesi degli oggetti sono

$$p_1 = 2, \quad w_1 = 1, \qquad \text{e} \qquad p_2 = \vartheta, \quad w_2 = \vartheta.$$

Una soluzione ottima inserisce l'oggetto 2, quindi

$$z(I_\vartheta) = \vartheta.$$

Poiché

$$\frac{p_1}{w_1} = 2 > \frac{p_2}{w_2} = \frac{\vartheta}{\vartheta} = 1,$$

GR inserisce per primo l'oggetto 1, lasciando capacità disponibile $\vartheta - 1$, quindi l'oggetto 2 non si può aggiungere. Quindi

$$GR(I_\vartheta) = 2.$$

Pertanto,

$$\lim_{\vartheta \to +\infty} \frac{z(I_\vartheta)}{GR(I_\vartheta)} = \lim_{\vartheta \to +\infty} \frac{\vartheta}{2} = +\infty.$$

□

- La proposizione mostra che non esiste una costante $\alpha \in [1, +\infty)$ tale che

$$\frac{z(I)}{GR(I)} \leq \alpha \qquad \text{per ogni istanza } I.$$

  Equivalentemente, il rapporto $\frac{z(I)}{GR(I)}$ può diventare grande a piacere con una scelta opportuna dell'istanza.

In altre parole, non esiste una costante $\alpha \in [1, +\infty)$ tale che GR sia un algoritmo di α-approssimazione, e il rapporto di prestazione nel caso peggiore di GR è illimitato:

$$r(GR) = +\infty.$$

- Un difetto fondamentale di GR è che può non selezionare l'oggetto di profitto massimo (anche quando quel solo oggetto è ottimo). In particolare, dando la priorità all'efficienza p_i/w_i, GR può riempire lo zaino con oggetti di efficienza alta e lasciare una capacità disponibile insufficiente per l'oggetto di profitto massimo.

2.8.2 Il rapporto di prestazione nel caso peggiore di GR^e

Proposizione 2.6

Per ogni istanza I del KP, abbiamo:

$$\frac{z(I)}{GR^e(I)} \leq 2. \tag{2.5}$$

Dimostrazione. Supponiamo che gli oggetti siano ordinati per efficienza non crescente. Siano s l'oggetto critico e rc la capacità residua. Abbiamo

$$\underbrace{\sum_{i=1}^{s-1} p_i \; + \; p_s \frac{rc}{w_s}}_{= z\left(\mathrm{LP_{KP}}\right)} \; \leq \; \sum_{i=1}^{s-1} p_i \; + \; p_s \qquad \text{poiché} \qquad rc < w_s,$$

dove $z\left(\mathrm{LP_{KP}}\right)$ è il valore ottimo del rilassamento lineare della formulazione PLI naturale del KP. Di conseguenza,

$$z(I) \leq z\left(\mathrm{LP_{KP}}\right) \leq \sum_{i=1}^{s-1} p_i + p_s \leq \sum_{i=1}^{s-1} p_i + p_{\max},$$

poiché per definizione di $p_{\max}$ abbiamo $p_{\max} \geq p_s$.

Inoltre,

$$GR^e(I) \geq GR(I) \geq \sum_{i=1}^{s-1} p_i \quad \text{e} \quad GR^e(I) \geq p_{\max}.$$

Sommando queste due disuguaglianze otteniamo

$$2\,GR^e(I) = GR^e(I) + GR^e(I) \; \geq \; \sum_{i=1}^{s-1} p_i + p_{\max}.$$

Pertanto,

$$z(I) \leq \sum_{i=1}^{s-1} p_i + p_{\max} \; \leq \; 2\,GR^e(I).$$

Dividendo per $GR^e(I) > 0$ otteniamo

$$\frac{z(I)}{GR^e(I)} \leq 2.$$

□

Proposizione 2.7

Esistono istanze I del KP tali che:

$$\frac{z(I)}{GR^e(I)} \to 2 \quad \text{per} \quad c \to +\infty. \tag{2.6}$$

Dimostrazione. Fissiamo un intero $\vartheta > 2$ e consideriamo l'istanza I_ϑ del KP con capacità $c = 2\vartheta$ e 3 oggetti:

$$p_1 = 2,\ w_1 = 1, \qquad p_2 = \vartheta,\ w_2 = \vartheta, \qquad p_3 = \vartheta,\ w_3 = \vartheta.$$

Gli oggetti 2 e 3 entrano insieme nello zaino poiché $w_2 + w_3 = 2\vartheta = c$; quindi una soluzione ottima inserisce gli oggetti 2 e 3 e

$$z(I_\vartheta) = 2\vartheta.$$

Inoltre,

$$\frac{p_1}{w_1} = 2 > \frac{p_2}{w_2} = \frac{p_3}{w_3} = 1,$$

quindi GR inserisce per primo l'oggetto 1. Dopo l'inserimento dell'oggetto 1, la capacità disponibile è $2\vartheta - 1$, quindi si può aggiungere esattamente uno degli oggetti 2 e 3. Pertanto GR restituisce una soluzione ammissibile di valore $\vartheta + 2$. Poiché $p_{\max} = \max\{p_1, p_2, p_3\} = \vartheta$, abbiamo $\vartheta + 2 > p_{\max}$, e quindi

$$GR^e(I_\vartheta) = \vartheta + 2.$$

Di conseguenza,

$$\lim_{\vartheta \to +\infty} \frac{z(I_\vartheta)}{GR^e(I_\vartheta)} = \lim_{\vartheta \to +\infty} \frac{2\vartheta}{\vartheta + 2} = \lim_{\vartheta \to +\infty} \frac{2}{1 + \frac{2}{\vartheta}} = 2.$$

□

Dalla Proposizione 2.6 e dalla Proposizione 2.7 segue che il rapporto di prestazione nel caso peggiore di GR^e è due, cioè abbiamo

$$r(GR^e) = 2$$

e GR^e è un algoritmo di 2-approssimazione.

- Un punto di forza di GR^e è che attenua il difetto principale di GR: anche se l'inserimento

greedy perde un oggetto molto redditizio a causa dell'ordine per efficienza, GR^e confronta esplicitamente la soluzione greedy con la migliore soluzione ammissibile con un solo oggetto e restituisce la migliore, garantendo così un fattore di approssimazione costante.

- Tuttavia, la garanzia $r(GR^e) = 2$ è tight: esistono istanze in cui GR^e è ingannato dall'efficienza e, dopo aver selezionato all'inizio un oggetto di efficienza alta, non ha più capacità sufficiente per inserire insieme i due oggetti di profitto maggiore (che formano la soluzione ottima), con un rapporto vicino a piacere a 2.

- Una soluzione che nel caso peggiore vale solo la metà della soluzione ottima non è un risultato molto soddisfacente.
- Conviene quindi cercare un algoritmo di approssimazione migliore, eventualmente al prezzo di un tempo di calcolo maggiore.

- Nell'esempio di caso peggiore precedente GR^e commette l'"errore" di escludere l'oggetto 3, che ha un profitto molto alto. Per evitare questo errore servono algoritmi euristici più avanzati.

2.8.3 Il rapporto di prestazione nel caso peggiore di H^2

Proposizione 2.8

Per ogni istanza I del KP, abbiamo:

$$\frac{z(I)}{H^2(I)} \leq \frac{4}{3}. \tag{2.7}$$

Dimostrazione. Se $z(I)$ è dato da un solo oggetto, abbiamo $H^2(I) = z(I)$.

Altrimenti consideriamo una coppia ammissibile $(\tilde{\imath}, \tilde{k}) \in \mathscr{P}$ che contribuisce con il profitto più alto in una soluzione ottima. L'istanza residua è $I(\tilde{\imath}, \tilde{k})$ e l'insieme residuo di oggetti è $\mathscr{I}(\tilde{\imath}, \tilde{k})$. L'algoritmo H^2 considera questa coppia di oggetti durante la sua esecuzione e applica GR^e all'istanza residua $I(\tilde{\imath}, \tilde{k})$.

La coppia contiene i due oggetti di profitto maggiore di una soluzione ottima, quindi ogni altro oggetto di quella soluzione ha profitto al più $\min\{p_{\tilde{\imath}}, p_{\tilde{k}}\}$ ed entra nella capacità residua: appartiene a $\mathscr{I}(\tilde{\imath}, \tilde{k})$. Il valore ottimo si può quindi scrivere come:

$$z(I) = z\big(I(\tilde{\imath}, \tilde{k})\big) + p_{\tilde{\imath}} + p_{\tilde{k}} \tag{2.8}$$

dove $z\big(I(\tilde{\imath}, \tilde{k})\big)$ è il valore ottimo dell'istanza residua $I(\tilde{\imath}, \tilde{k})$.

Dalle operazioni di H^2, abbiamo:

$$H^2(I) \quad \geq \quad p_{\tilde{\imath}} + p_{\tilde{k}} + GR^e\big(I(\tilde{\imath}, \tilde{k})\big) \tag{2.9}$$

e poiché GR^e è un algoritmo di 2-approssimazione, cioè $GR^e(I) \geq \frac{1}{2}\, z(I)$, abbiamo:

$$GR^e\big(I(\tilde{\imath},\tilde{k})\big) \quad \geq \quad \frac{1}{2}\, z\big(I(\tilde{\imath},\tilde{k})\big)$$

Di conseguenza:

$$H^2(I) \geq p_{\tilde{\imath}} + p_{\tilde{k}} + \frac{1}{2}\, z\big(I(\tilde{\imath},\tilde{k})\big)$$

Inoltre, per la (2.8), abbiamo:

$$H^2(I) \geq p_{\tilde{\imath}} + p_{\tilde{k}} + \frac{1}{2}\big(z(I) - p_{\tilde{\imath}} - p_{\tilde{k}}\big) = \frac{1}{2}\big(z(I) + p_{\tilde{\imath}} + p_{\tilde{k}}\big)$$

Consideriamo due casi possibili:

- Caso i):

  $$p_{\tilde{\imath}} + p_{\tilde{k}} \geq \frac{1}{2}\, z(I)$$

  In questo caso abbiamo:

  $$\begin{aligned} H^2(I) &\geq \frac{1}{2}\big(z(I) + p_{\tilde{\imath}} + p_{\tilde{k}}\big) \\ &\geq \frac{1}{2}\left(z(I) + \frac{1}{2}\, z(I)\right) = \frac{3}{4}\, z(I) \end{aligned}$$

  e di conseguenza

  $$H^2(I) \geq \frac{3}{4}\, z(I)$$

- Caso ii):

  $$p_{\tilde{\imath}} + p_{\tilde{k}} < \frac{1}{2}\, z(I)$$

  Poiché i loro profitti sommano a meno di $\frac{1}{2}\, z(I)$, almeno uno dei due oggetti $\tilde{\imath}$, $\tilde{k}$ ha profitto minore di $\frac{1}{4}\, z(I)$.

  Dal rilassamento $\mathrm{LP_{KP}}$ dell'istanza residua $I(\tilde{\imath},\tilde{k})$, abbiamo:

  $$\begin{aligned} z\big(I(\tilde{\imath},\tilde{k})\big) &\leq z\big(\mathrm{LP_{KP}}\big) \\ &\leq GR^e\big(I(\tilde{\imath},\tilde{k})\big) + p_s \\ &\leq GR^e\big(I(\tilde{\imath},\tilde{k})\big) + \frac{1}{4}\, z(I) \end{aligned}$$

  dove s è l'oggetto critico di $I(\tilde{\imath},\tilde{k})$ e $p_s < \frac{1}{4} z(I)$ poiché tutti i suoi oggetti hanno profitto al più $\min\{p_{\tilde{\imath}}, p_{\tilde{k}}\} < \frac{1}{4} z(I)$ (per costruzione dell'istanza residua). Se tutti gli oggetti dell'istanza residua entrano nella capacità residua non c'è oggetto critico: allora GR^e li inserisce tutti, $z\big(I(\tilde{\imath},\tilde{k})\big) = GR^e\big(I(\tilde{\imath},\tilde{k})\big)$, e le disuguaglianze

valgono con $p_s := 0$.

Abbiamo:

$$z(I) = p_{\tilde{\imath}} + p_{\tilde{k}} + z\big(I(\tilde{\imath}, \tilde{k})\big)$$

$$\leq \underbrace{p_{\tilde{\imath}} + p_{\tilde{k}} + GR^e\big(I(\tilde{\imath}, \tilde{k})\big)}_{\leq H^2(I),\ \text{eq.}(2.9)} + \frac{1}{4}\, z(I)$$

$$\leq H^2(I) + \frac{1}{4}\, z(I)$$

e di conseguenza:

$$H^2(I) \geq \frac{3}{4}\, z(I)$$

In entrambi i casi $H^2(I) \geq \frac{3}{4}\, z(I)$, cioè, dividendo per $H^2(I) > 0$,

$$\frac{z(I)}{H^2(I)} \leq \frac{4}{3}.$$

□

Proposizione 2.9

Esistono istanze I_M del KP tali che:

$$\frac{z(I_M)}{H^2(I_M)} \to \frac{4}{3} \qquad \text{per} \qquad M \to +\infty. \tag{2.10}$$

Dimostrazione. Consideriamo l'istanza I_M del KP con capacità $c = 4\,M$ e 5 oggetti:

$$p_1 = 2, \quad w_1 = 1, \quad \text{e} \quad p_2 = p_3 = p_4 = p_5 = M, \quad w_2 = w_3 = w_4 = w_5 = M,$$

dove $M \in \mathbb{Z}_{\geq 1}, M > 2$ è un numero intero. Una soluzione ottima inserisce i quattro oggetti grandi, poiché $w_2 + w_3 + w_4 + w_5 = 4\,M = c$, quindi

$$z(I_M) = 4\,M.$$

Le efficienze sono

$$\frac{p_1}{w_1} = 2 \quad > \quad \frac{p_2}{w_2} = \frac{p_3}{w_3} = \frac{p_4}{w_4} = \frac{p_5}{w_5} = \frac{M}{M} = 1$$

quindi l'oggetto 1 viene per primo nell'ordine per efficienza.

Consideriamo i due tipi possibili di coppia esaminati da H^2:

1. una coppia di due oggetti grandi: la capacità residua è $2\,M$ e l'istanza residua mantiene l'oggetto 1 e i due oggetti grandi rimanenti, quindi GR^e inserisce l'oggetto 1 e un oggetto grande, per un valore totale di $3\,M + 2$;

2. una coppia formata dall'oggetto 1 e da un oggetto grande: l'istanza residua è vuota, perché nessun oggetto ha profitto al più $\min\{2, M\} = 2$, per un valore totale di $M + 2$.

Il primo tipo di coppia dà il valore maggiore, quindi

$$H^2(I_M) = 3\,M + 2$$

Di conseguenza,

$$\lim_{M\to\infty} \frac{z(I_M)}{H^2(I_M)} = \lim_{M\to\infty} \frac{4\,M}{3\,M+2} = \lim_{M\to\infty} \frac{4}{3+\frac{2}{M}} = \frac{4}{3}.$$

□

Dalla Proposizione 2.8 e dalla Proposizione 2.9 segue che il rapporto di prestazione nel caso peggiore di H^2 è quattro terzi, cioè abbiamo

$$r(H^2) = \frac{4}{3}$$

e H^2 è un algoritmo di $\frac{4}{3}$-approssimazione. Scommettere su due oggetti invece che su uno fa scendere il rapporto dal 2 di GR^e a $\frac{4}{3}$.

2.8.4 Il rapporto di prestazione nel caso peggiore di H^ℓ

Proposizione 2.10

Per ogni istanza I del KP, abbiamo:

$$\frac{z(I)}{H^\ell(I)} \leq \frac{\ell+2}{\ell+1}. \tag{2.11}$$

Dimostrazione. Se una soluzione ottima contiene al più $\ell - 1$ oggetti, l'algoritmo la trova fra gli insiemi di $\widehat{T}(\ell)$, e $H^\ell(I) = z(I)$.

Altrimenti, sia $\tilde{L} \in T(\ell)$ l'insieme degli ℓ oggetti di profitto maggiore di una soluzione ottima. L'algoritmo considera $\tilde{L}$ durante la sua esecuzione e applica GR^e all'istanza

residua $I(\tilde{L})$. Tutti gli altri oggetti della soluzione ottima appartengono a $\mathscr{I}(\tilde{L})$, quindi

$$z(I) = \sum_{i \in \tilde{L}} p_i + z\big(I(\tilde{L})\big), \tag{2.12}$$

dove $z\big(I(\tilde{L})\big)$ è il valore ottimo dell'istanza residua. Dalle operazioni di H^ℓ, e poiché GR^e è un algoritmo di 2-approssimazione,

$$H^\ell(I) \;\geq\; \sum_{i \in \tilde{L}} p_i + GR^e\big(I(\tilde{L})\big) \;\geq\; \sum_{i \in \tilde{L}} p_i + \frac{1}{2}\, z\big(I(\tilde{L})\big) \;=\; \frac{1}{2}\Big(z(I) + \sum_{i \in \tilde{L}} p_i\Big),$$

dove l'ultima uguaglianza segue dalla (2.12). Consideriamo due casi.

- Caso i): $\displaystyle\sum_{i \in \tilde{L}} p_i \geq \frac{\ell}{\ell+2}\, z(I)$. Allora

$$H^\ell(I) \;\geq\; \frac{1}{2}\left(z(I) + \frac{\ell}{\ell+2}\, z(I)\right) = \frac{\ell+1}{\ell+2}\, z(I).$$

- Caso ii): $\displaystyle\sum_{i \in \tilde{L}} p_i < \frac{\ell}{\ell+2}\, z(I)$. Allora almeno un oggetto di $\tilde{L}$ ha profitto minore di $\frac{1}{\ell+2}\, z(I)$, e lo stesso vale per tutti gli oggetti dell'istanza residua, i cui profitti sono al più $\min\{p_i : i \in \tilde{L}\}$. Sia s l'oggetto critico di $I(\tilde{L})$ (se non esiste, GR^e inserisce tutti gli oggetti di $I(\tilde{L})$, e le disuguaglianze seguenti valgono con $p_s := 0$). Il rilassamento lineare dell'istanza residua dà

$$z\big(I(\tilde{L})\big) \;\leq\; z\big(\mathrm{LP}_{\mathrm{KP}}\big) \;\leq\; GR^e\big(I(\tilde{L})\big) + p_s \;<\; GR^e\big(I(\tilde{L})\big) + \frac{1}{\ell+2}\, z(I),$$

e quindi, per la (2.12),

$$z(I) \;<\; \sum_{i \in \tilde{L}} p_i + GR^e\big(I(\tilde{L})\big) + \frac{1}{\ell+2}\, z(I) \;\leq\; H^\ell(I) + \frac{1}{\ell+2}\, z(I),$$

cioè $H^\ell(I) > \frac{\ell+1}{\ell+2}\, z(I)$.

In entrambi i casi $H^\ell(I) \geq \frac{\ell+1}{\ell+2}\, z(I)$, e dividendo per $H^\ell(I) > 0$ si ottiene l'enunciato. □

Proposizione 2.11

Esistono istanze I_M del KP tali che:

$$\frac{z(I_M)}{H^\ell(I_M)} \;\to\; \frac{\ell+2}{\ell+1} \qquad \text{per} \qquad M \to +\infty. \tag{2.13}$$

Dimostrazione. Fissiamo un intero $M > 2$ e consideriamo l'istanza I_M del KP con capacità $c = (\ell + 2)\, M$ e $n = \ell + 3$ oggetti, dove

$$p_1 = 2, \quad w_1 = 1, \quad \text{e} \quad p_j = w_j = M \quad \text{per} \quad j \in \{2, 3, \ldots, n\}.$$

Una soluzione ottima inserisce gli $\ell + 2$ oggetti di peso M, che riempiono lo zaino, quindi $z(I_M) = (\ell + 2)\, M$. L'oggetto 1 viene per primo nell'ordine per efficienza, poiché $\frac{p_1}{w_1} = 2 > 1$. Gli insiemi L esaminati da H^ℓ sono di due tipi:

1. L formato da ℓ oggetti di profitto M: la capacità residua è $2M$, l'istanza residua mantiene l'oggetto 1 e i due oggetti rimanenti di profitto M, e GR^e inserisce l'oggetto 1 e uno di essi, per un valore totale di $\ell M + M + 2 = (\ell + 1)\, M + 2$;
2. L che contiene l'oggetto 1 e $\ell - 1$ oggetti di profitto M: l'istanza residua è vuota, poiché nessun oggetto ha profitto al più $\min\{2, M\} = 2$, per un valore totale di $(\ell - 1)\, M + 2$.

Anche gli insiemi di al più $\ell - 1$ oggetti danno al più $(\ell - 1)\, M + 2$. Quindi $H^\ell(I_M) = (\ell + 1)\, M + 2$ e

$$\lim_{M \to \infty} \frac{z(I_M)}{H^\ell(I_M)} = \lim_{M \to \infty} \frac{(\ell + 2)\, M}{(\ell + 1)\, M + 2} = \lim_{M \to \infty} \frac{\ell + 2}{\ell + 1 + \frac{2}{M}} = \frac{\ell + 2}{\ell + 1}.$$

Per $\ell = 2$ è l'istanza usata per H^2. □

Dalla Proposizione 2.10 e dalla Proposizione 2.11 segue che il rapporto di prestazione nel caso peggiore di H^ℓ è

$$r(H^\ell) = \frac{\ell + 2}{\ell + 1}$$

e H^ℓ è un algoritmo di $\frac{\ell+2}{\ell+1}$-approssimazione. Per $\ell = 2$ ritroviamo $r(H^2) = \frac{4}{3}$, e il rapporto tende a 1 al crescere di ℓ.

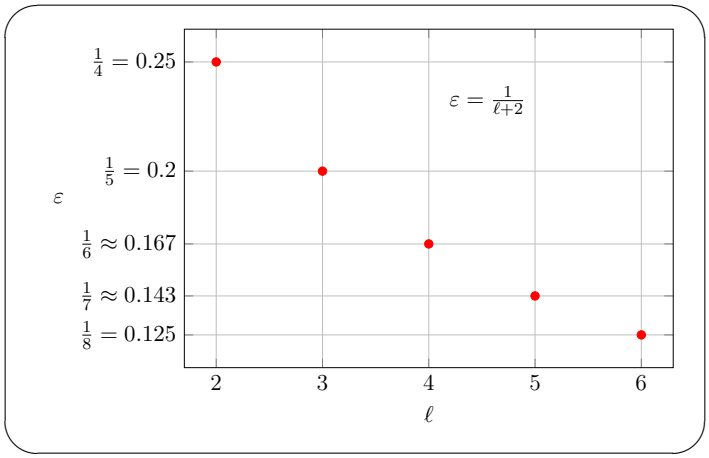
2.8.5 Schemi di approssimazione

- Richiamiamo le definizioni del capitolo «Principi degli algoritmi di approssimazione». L'**errore relativo** di un algoritmo $\mathscr{A}$ su un'istanza I del KP è $\frac{z(I) - \mathscr{A}(I)}{z(I)}$. Uno **schema di approssimazione** è una famiglia di algoritmi $\{\mathscr{A}_\varepsilon\}_{\varepsilon \in (0,1)}$ tale che, per ogni ε e ogni istanza I, l'errore relativo di $\mathscr{A}_\varepsilon$ è al più ε; per un problema di massimo questo significa che $\mathscr{A}_\varepsilon$ è un algoritmo di $\frac{1}{1-\varepsilon}$-approssimazione. Lo schema è un **PTAS** se, per ogni ε fissato, il tempo di calcolo di $\mathscr{A}_\varepsilon$ è polinomiale nella dimensione dell'istanza, e un **FPTAS** se è polinomiale sia nella dimensione dell'istanza sia nell'accuratezza $\frac{1}{\varepsilon}$.

- Per la Proposizione 2.10, $H^\ell(I) \geq \frac{\ell+1}{\ell+2}\, z(I)$, quindi l'errore relativo di H^ℓ è al più

$$1 - \frac{\ell + 1}{\ell + 2} = \frac{1}{\ell + 2},$$

che tende a 0 al crescere di ℓ:



- La relazione si può leggere al contrario: dato l'errore relativo ε da raggiungere, basta scegliere ℓ con $\frac{1}{\ell+2} \leq \varepsilon$, cioè $\ell \geq \frac{1}{\varepsilon} - 2$.

Scegliendo

$$\ell(\varepsilon) = \min\left\{\max\left\{\left\lceil\frac{1}{\varepsilon}\right\rceil - 2,\ 2\right\},\ n\right\}$$

otteniamo l'algoritmo H^ε, cioè H^ℓ con $\ell = \ell(\varepsilon)$, il cui errore relativo è al più ε per ogni $\varepsilon \in (0, 1)$.

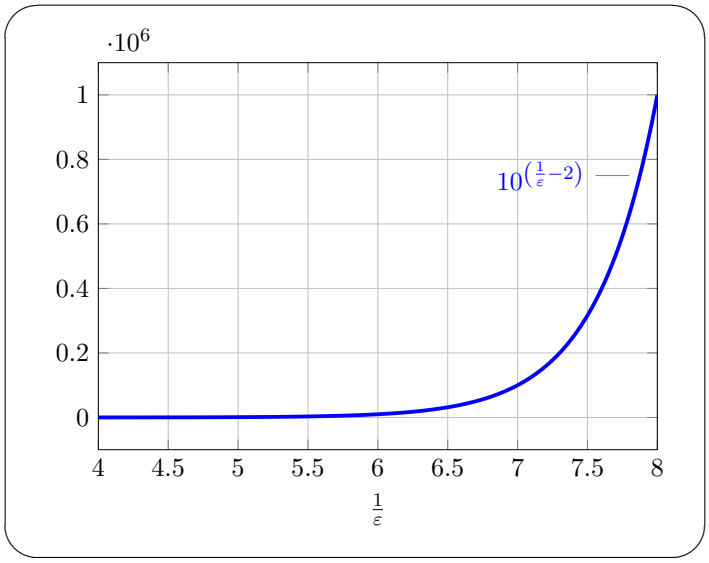
- Per $\varepsilon \geq \frac{1}{4}$ abbiamo $\ell(\varepsilon) = 2$, e H^ε è l'algoritmo H^2. La dimensione ℓ degli insiemi scommessi non può superare n, e con $\ell = n$ l'algoritmo enumera tutte le soluzioni ammissibili ed è esatto.

Osservazione 2.12

La famiglia $\{H^\varepsilon\}_{\varepsilon\in(0,1)}$ è un PTAS per il problema dello zaino 0/1.

- Per ogni ε fissato, $\ell(\varepsilon)$ è una costante, e per l'Osservazione 2.4 il tempo di calcolo $O\big(n^{\ell(\varepsilon)+1}\big)$ è polinomiale nella dimensione n dell'istanza.

- Tuttavia, H^ε **non** è un FPTAS: l'esponente $\ell(\varepsilon) = \left\lceil\frac{1}{\varepsilon}\right\rceil - 2$ cresce con l'accuratezza $\frac{1}{\varepsilon}$, quindi il tempo di calcolo non è polinomiale in $\frac{1}{\varepsilon}$. Il grafico mostra, per $n = 10$ oggetti e $4 \leq \frac{1}{\varepsilon} \leq 8$, il numero $n^{\frac{1}{\varepsilon}-2}$, un lower bound su $n^{\ell(\varepsilon)}$ (il limite $\ell(\varepsilon) \leq n$ non interviene, poiché $\frac{1}{\varepsilon} \leq 12$):



- L'algoritmo H^ε è un metodo semplice e abbastanza facile da implementare, ma il suo tempo di calcolo esplode al crescere dell'accuratezza richiesta. Anche per un errore relativo moderato $\varepsilon = \frac{1}{10}$, cioè uno scostamento del 10 per cento dal valore ottimo, abbiamo $\ell(\varepsilon) = 8$ e un tempo di calcolo $O(n^9)$, o comunque $O(n^8)$ con l'implementazione migliorata.

2.8.6 Il rapporto di prestazione nel caso peggiore dell'algoritmo Basic-FPTAS

Proposizione 2.13

Per ogni istanza I del KP e ogni $\varepsilon \in (0, 1)$, abbiamo:

$$\frac{z(I)}{FP^\varepsilon(I)} \leq \frac{1}{1-\varepsilon}. \tag{2.14}$$

Dimostrazione. Siano $\tilde{S}$ una soluzione ottima di I e S la soluzione restituita da FP^ε, cioè una soluzione ottima dell'istanza scalata I'. Poiché I e I' hanno le stesse soluzioni ammissibili, $\tilde{S}$ è ammissibile anche per I', e per l'ottimalità di S per I',

$$\sum_{j \in S} p'_j \;\geq\; \sum_{j \in \tilde{S}} p'_j.$$

Per definizione dei profitti scalati, $K\,p'_j \leq p_j$ e $K\,p'_j > p_j - K$ per ogni oggetto j. Pertanto

$$FP^\varepsilon(I) = \sum_{j \in S} p_j \;\geq\; K \sum_{j \in S} p'_j \;\geq\; K \sum_{j \in \tilde{S}} p'_j \;>\; \sum_{j \in \tilde{S}} (p_j - K) \;=\; z(I) - |\tilde{S}|\,K.$$

Poiché una soluzione ottima contiene al più n oggetti e $K = \varepsilon\, GR^e(I)/n$,

$$|\tilde{S}|\,K \;\leq\; n\,K \;=\; \varepsilon\, GR^e(I) \;\leq\; \varepsilon\, z(I),$$

dove l'ultima disuguaglianza vale perché $GR^e(I)$ è il valore di una soluzione ammissibile. Combinando le due relazioni,

$$FP^\varepsilon(I) \;\geq\; z(I) - \varepsilon\, z(I) \;=\; (1-\varepsilon)\, z(I),$$

e dividendo per $FP^\varepsilon(I) > 0$ si ottiene l'enunciato. Sull'istanza dell'Esempio 2.10, con $\varepsilon = \frac{1}{2}$, il rapporto è $\frac{47}{38} \approx 1.24$, ben al di sotto del bound $\frac{1}{1-\varepsilon} = 2$. □

Osservazione 2.14

L'algoritmo FP^ε si può calcolare in tempo $O\big(n^2/\varepsilon\big)$.

Dimostrazione. Sia $\tilde{q}$ il valore ottimo dell'istanza scalata I'. Ogni soluzione ammissibile di I' è ammissibile per I, quindi

$$\tilde{q} \;\leq\; \frac{z(I)}{K} \;\leq\; \frac{2\,GR^e(I)}{K} \;=\; \frac{2\,GR^e(I)\,n}{\varepsilon\,GR^e(I)} \;=\; \frac{2n}{\varepsilon},$$

dove $z(I) \leq 2\,GR^e(I)$ è esattamente la garanzia di prestazione relativa di GR^e. Poiché $\tilde{q}$ è un intero, $\tilde{q} \leq Q = \lfloor 2n/\varepsilon \rfloor$, e troncare a Q la tabella della programmazione dinamica non cambia il risultato.
La tabella ha $n+1$ righe e $Q+1$ colonne, quindi la programmazione dinamica richiede tempo $O(n\,Q) = O(n^2/\varepsilon)$, mentre la scalatura e la chiamata a GR^e richiedono tempo $O(n)$ e $O(n \log n)$.
Senza il troncamento, la tabella arriverebbe al profitto scalato totale $P' = \sum_j p'_j$. Poiché $p_j \leq GR^e(I)$ dà $p'_j \leq \frac{n}{\varepsilon}$, abbiamo solo $P' \leq \frac{n^2}{\varepsilon}$, valore che si può raggiungere, e il tempo di calcolo sarebbe $O(n\,P') = O(n^3/\varepsilon)$. □

Dalla Proposizione 2.13 e dall'Osservazione 2.14 segue che la famiglia $\{FP^\varepsilon\}_{\varepsilon\in(0,1)}$ è uno **schema di approssimazione** il cui tempo di calcolo è polinomiale sia nella dimensione n dell'istanza sia nell'accuratezza $\frac{1}{\varepsilon}$, cioè un **FPTAS** per il problema dello zaino 0/1.

- La garanzia di GR^e si usa due volte, con due ruoli diversi: $GR^e(I) \leq z(I)$ fissa il fattore di scala, e $z(I) \leq 2\,GR^e(I)$ limita la dimensione della tabella della programmazione dinamica. Un'euristica migliore al posto di GR^e accorcerebbe la tabella, non migliorerebbe la garanzia.

2.9 Garanzia di prestazione assoluta

- Per le garanzie di prestazione assolute del problema dello zaino 0/1 esiste purtroppo solo un risultato negativo.
- Il teorema seguente afferma che, se un algoritmo di approssimazione avesse una garanzia di prestazione assoluta k per un certo k fissato, allora questo algoritmo si potrebbe usare per risolvere all'ottimo qualsiasi istanza del problema dello zaino 0/1.

Sia $f(n, c)$ una funzione che descrive il tempo di calcolo di un algoritmo per il KP con n oggetti e capacità c.

Teorema 2.15

Sia $\mathscr{A}$ un algoritmo di approssimazione per il KP con garanzia di prestazione assoluta k e tempo di calcolo in $O\big(f(n,c)\big)$ per ogni istanza con n oggetti e capacità c. Allora $\mathscr{A}$ calcola anche la soluzione ottima di ogni istanza di questo tipo in tempo $O(f(n,c))$.

Dimostrazione. Consideriamo un algoritmo di approssimazione $\mathscr{A}$ per il KP con garanzia di prestazione assoluta k e sia I data da

$$p_1, \ldots, p_n, \quad w_1, \ldots, w_n, \quad c.$$

Costruiamo la nuova istanza I' con

$$p'_1, \ldots, p'_n, \quad w'_1, \ldots, w'_n, \quad c'$$

tale che

$$p'_j = (k+1)\, p_j, \quad w'_j = w_j, \quad \forall j \in \mathscr{I}, \quad \text{e} \quad c' = c.$$

- Le due istanze I e I' hanno esattamente lo stesso insieme di soluzioni ammissibili.
- Poiché il valore della funzione obiettivo di una soluzione di I' è esattamente $k+1$ volte il valore della funzione obiettivo della stessa soluzione per I, le due istanze hanno anche le stesse soluzioni ottime.

Applicando l'algoritmo $\mathscr{A}$ all'istanza I' otteniamo una soluzione di valore $\mathscr{A}(I')$ con

$$z(I') - \mathscr{A}(I') \leq k$$

per la garanzia di prestazione assoluta di $\mathscr{A}$.

Interpretando l'output di $\mathscr{A}$ come soluzione di I otteniamo una soluzione ammissibile di valore $\mathscr{A}(I)$, e

$$\mathscr{A}(I') = (k+1)\, \mathscr{A}(I) \qquad \text{e} \qquad z(I') = (k+1)\, z(I)$$

Sostituendo nella disuguaglianza precedente otteniamo

$$(k+1)\, z(I) - (k+1)\, \mathscr{A}(I) \leq k \iff z(I) - \mathscr{A}(I) \leq \frac{k}{k+1}$$

Ma poiché tutti i profitti sono interi, anche $z(I) - \mathscr{A}(I)$ deve essere intero, quindi questa disuguaglianza implica

$$z(I) - \mathscr{A}(I) \leq \left\lfloor \frac{k}{k+1} \right\rfloor \Longrightarrow z(I) - \mathscr{A}(I) \leq 0 \Longrightarrow z(I) = \mathscr{A}(I)$$

Poiché la soluzione approssimata di I' è stata calcolata in tempo $O(f(n, c))$, una soluzione ottima di I si può costruire nello stesso tempo. □

- Questo teorema indica che un algoritmo con garanzia di prestazione assoluta fissata per il KP avrebbe lo stesso tempo di calcolo sfavorevole di un algoritmo esatto.

  È molto improbabile che esista un algoritmo esatto per il KP con tempo di calcolo polinomiale in n.

- Quindi anche gli algoritmi di approssimazione con garanzia di prestazione assoluta fissata sono destinati ad avere un tempo di calcolo pseudo-polinomiale, che dipende in modo sfavorevole dalla grandezza dei dati.

2.10 Errori tipici

Attenzione

- **Fermare l'algoritmo greedy all'oggetto critico.** GR salta l'oggetto critico s ma continua a scorrere gli oggetti, e può selezionare oggetti che seguono s (Esempio 2.2).
- **Confondere i_m e s.** In GR^e l'oggetto i_m è l'oggetto di profitto massimo $p_{\max}$, non l'oggetto critico s: i due coincidono solo per caso.
- **Dimenticare di ordinare per efficienza.** Tutti gli algoritmi di questo capitolo suppongono gli oggetti ordinati per $\frac{p_j}{w_j}$ non crescente; l'oggetto critico e la soluzione greedy dipendono dall'ordine.
- **Chiamare H^ℓ un FPTAS.** Il suo tempo di calcolo $O(n^{\ell+1})$ ha l'esponente $\ell(\varepsilon) = \lceil \frac{1}{\varepsilon} \rceil - 2$ che cresce con $\frac{1}{\varepsilon}$: H^ε è un PTAS, non un FPTAS.
- **Scalare i pesi.** L'algoritmo Basic-FPTAS arrotonda per difetto solo i profitti; i pesi e la capacità restano invariati, così che le soluzioni ammissibili sono le stesse.
- **Contare l'intera tabella della programmazione dinamica.** Il bound $O(n^2/\varepsilon)$ vale perché la tabella è troncata a $\lfloor 2n/\varepsilon \rfloor$; il profitto scalato totale può arrivare a $\frac{n^2}{\varepsilon}$.

2.11 In sintesi

In sintesi

- GR non ha una garanzia costante, GR^e (il migliore fra GR e il miglior oggetto singolo) ha $r(GR^e) = 2$, H^2 ha $r(H^2) = \frac{4}{3}$ e H^ℓ ha $r(H^\ell) = \frac{\ell+2}{\ell+1}$.
- $\{H^\varepsilon\}$ è un PTAS con tempo di calcolo $O(n^{\ell+1})$; l'algoritmo Basic-FPTAS arrotonda per difetto i profitti, risolve esattamente l'istanza scalata con la programmazione dinamica ed è un FPTAS: garantisce $\frac{1}{1-\varepsilon}$ in tempo $O(n^2/\varepsilon)$.
- Non esiste un algoritmo con garanzia di prestazione assoluta fissata, a meno che non risolva esattamente il KP.

Quadro riassuntivo dell'istanza del KP dell'Esempio 2.1 ($c = 9$, $p = (6, 5, 8, 9, 6, 7, 3)$, $w = (2, 3, 6, 7, 5, 9, 4)$). I bound su $z(I)$ trovati finora:

Fonte	Lower bound	Upper bound	Da
Bound di Dantzig	–	16	$\lfloor \frac{49}{3} \rfloor$, con $s = 3$ e $rc = 4$
Questo capitolo, GR e GR^e	14	–	$S = \{1, 2, 7\}$
Questo capitolo, H^2 e H^3	15	–	$S = \{1, 4\}$
Questo capitolo, FP^ε con $\varepsilon = \frac{1}{2}$	15	–	$K = 1$, l'istanza scalata è quella originale
Bound migliori	15	16	UB − LB = 1; il valore ottimo è 15

2.12 Implementazione in C++

- In questa sezione gli oggetti sono indicizzati a partire da 0. La logica dell'algoritmo non cambia.
- Segue un'implementazione in C++ dell'**algoritmo greedy**.

```
// items are assumed to be sorted by non-increasing efficiency
// x[i] stores the selection decision for item i
// the function returns the profit of the solution x

int GreedyKP(int n, int c, int* w, int* p, int* x)
{
    for (int i = 0; i < n; ++i)
    {
        x[i] = 0;                       // item i is initially not selected
    }
    int rc = c;                         // residual capacity

    for (int i = 0; i < n; ++i)
```

```
14      {
15          if (w[i] <= rc)
16          {
17              x[i] = 1;                   // select item i
18              rc = rc - w[i];             // update residual capacity
19          }
20      }
21
22      // profit of the solution x, returned for convenience
23      int z = 0;
24      for (int i = 0; i < n; ++i)
25      {
26          if (x[i] == 1) z = z + p[i];
27      }
28      return z;
29  }
```

Questa è l'esecuzione passo per passo dell'implementazione C++ di GR sull'istanza del KP dell'Esempio 2.1.

Inizializzazione

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	-	-	-	-	-	-	-

$rc = 9$

1. Iterazione 1 – L'oggetto 0 viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	-	-	-	-	-	-

$rc = 7$

2. Iterazione 2 – L'oggetto 1 viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	-	-	-	-	-

$rc = 4$

3. Iterazione 3 – L'oggetto 2 non viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	-	-	-	-

$rc = 4$

4. Iterazione 4 – L'oggetto 3 non viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	-	-	-

$rc = 4$

5. Iterazione 5 – L'oggetto 4 non viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	0	-	-

$rc = 4$

6. Iterazione 6 – L'oggetto 5 non viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	0	0	-

$rc = 4$

7. Iterazione 7 – L'oggetto 6 viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	0	0	1

$rc = 0$

Il valore calcolato è

$$z = 14$$

Questa è l'esecuzione passo per passo dell'implementazione C++ di GR sull'istanza del KP dell'Esempio 2.3.

Inizializzazione

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$
	-	-	-	-

$rc = 10$

1. Iterazione 1 – L'oggetto 0 viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$
	1	-	-	-

$rc = 8$

2. Iterazione 2 – L'oggetto 1 viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$
	1	1	-	-

$rc = 5$

3. Iterazione 3 – L'oggetto 2 non viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$
	1	1	0	-

$rc = 5$

4. Iterazione 4 – L'oggetto 3 viene selezionato

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$
	1	1	0	1

$rc = 0$

Il valore calcolato è

$$z = 18$$

- Segue un'implementazione in C++ dell'**algoritmo greedy esteso**.

```cpp
// items are assumed to be sorted by non-increasing efficiency
// x[i] stores the selection decision for item i
// the function returns the profit of the solution x

int ExtendedGreedyKP(int n, int c, int* w, int* p, int* x)
{
    GreedyKP(n, c, w, p, x);              // greedy solution

    int z = 0;                            // value of the greedy solution
    for (int i = 0; i < n; ++i)
    {
        if (x[i] == 1)
        {
            z = z + p[i];                 // add the profit of a selected item
        }
    }

    int im = 0;                           // best single item
    int pm = p[0];
    for (int i = 1; i < n; ++i)
    {
        if (p[i] > pm)
        {
            im = i;                       // update best single item
            pm = p[i];                    // update its profit
        }
    }

    int* xhat = new int[n];               // best single-item solution
    for (int i = 0; i < n; ++i)
    {
        xhat[i] = 0;
    }
    xhat[im] = 1;                         // select only item im

    if (z < pm)
    {
        for (int i = 0; i < n; ++i)
```

```
39          {
40              x[i] = xhat[i];              // copy component by component
41          }
42          z = pm;
43      }
44
45      delete[] xhat;
46      return z;
47  }
```

- Questa è l'esecuzione passo per passo dell'implementazione C++ di GR^e sull'istanza del KP dell'Esempio 2.1.

Inizializzazione

La soluzione calcolata da GR è

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	0	0	1

Il valore calcolato è

$$z = 14$$

Il profitto massimo è:

$$p_{\max} = 9,$$

e l'oggetto di profitto massimo è (indici a partire da 0)

$$i_m = 3.$$

Poiché $14 \geq 9$, GR^e mantiene la soluzione greedy

- Questa è l'esecuzione passo per passo dell'implementazione C++ di GR^e sull'istanza del KP dell'Esempio 2.3.

Inizializzazione

La soluzione calcolata da GR è

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$
	1	1	0	1

Il valore calcolato è

$$z = 18$$

Il profitto massimo è:

$$p_{\max} = 19,$$

e l'oggetto di profitto massimo è (indici a partire da 0)

$$i_m = 2.$$

Poiché $18 < 19$, GR^e sostituisce la soluzione greedy con la migliore soluzione ammissibile con un solo oggetto, cioè

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$
	0	0	1	0

Il valore aggiornato è

$$z = 19.$$

- Segue un'implementazione in `C++` dell'**algoritmo** H^2. Chiama `ExtendedGreedyKP` sull'istanza residua di ogni coppia ammissibile, esattamente come lo pseudocodice, e mantiene la migliore soluzione trovata.

```
// items are assumed to be sorted by non-increasing efficiency
// the function returns H2(I), the value of the best solution found;
// in addition, x stores that solution, which the pseudo-code does not record

int H2KP(int n, int c, int* w, int* p, int* x)
{
    int im = 0;
    for (int j = 1; j < n; ++j)
    {
        if (p[j] > p[im]) im = j;
    }
    int pmax = p[im];
    int lb = pmax;                                  // best value found
    for (int j = 0; j < n; ++j) x[j] = 0;           // record the solution
    x[im] = 1;

    // residual instance I_ik: items, profits, weights and its solution
    int* idx = new int[n];
    int* pr  = new int[n];
    int* wr  = new int[n];
    int* xr  = new int[n];

    for (int i = 0; i < n; ++i)
    {
        for (int k = i + 1; k < n; ++k)
        {
            if (w[i] + w[k] > c) continue;          // the pair (i,k) is not in
    P

            int cbar = c - w[i] - w[k];             // residual capacity
            int pmin = (p[i] < p[k] ? p[i] : p[k]);

            // residual set I(i,k)
            int m = 0;
            for (int j = 0; j < n; ++j)
            {
                if (j != i && j != k && p[j] <= pmin && w[j] <= cbar)
                {
                    idx[m] = j; pr[m] = p[j]; wr[m] = w[j]; m++;
                }
            }

            // value of GR^e on the residual instance
            int zres = (m > 0) ? ExtendedGreedyKP(m, cbar, wr, pr, xr) : 0;
```

```
            int lbik = p[i] + p[k] + zres;

            if (lbik > lb)
            {
                lb = lbik;
                for (int j = 0; j < n; ++j) x[j] = 0;   // record the solution
                x[i] = 1; x[k] = 1;
                for (int j = 0; j < m; ++j) if (xr[j] == 1) x[idx[j]] = 1;
            }
        }
    }

    delete[] idx; delete[] pr; delete[] wr; delete[] xr;
    return lb;
}
```

- Sull'istanza del KP dell'Esempio 2.8, con $n = 7$ oggetti e capacità $c = 12$, la funzione restituisce $lb = 20$ con $x = (1, 1, 0, 1, 0, 0, 0)$, cioè la soluzione $S = \{1, 2, 4\}$ già trovata in quell'esempio, che è ottima per quell'istanza.
- I due cicli annidati su i e k eseguono $O(n^2)$ iterazioni, e ogni iterazione costruisce l'istanza residua ed esegue `ExtendedGreedyKP` in tempo $O(n)$: è il tempo di calcolo $O(n^3)$ della Proposizione 2.3.
- Segue un'implementazione in `C++` dell'**algoritmo** H^ℓ. La funzione `NextSubset` enumera i sottoinsiemi di esattamente r oggetti in ordine lessicografico; la fase preliminare esamina i sottoinsiemi di al più $\ell - 1$ oggetti, e ogni sottoinsieme di esattamente ℓ oggetti viene completato da `ExtendedGreedyKP` sulla sua istanza residua, come nello pseudocodice.

```
// items are assumed to be sorted by non-increasing efficiency
// l is the number of guessed items, with 2 <= l <= n
// the function returns H^l(I), the value of the best solution found;
// in addition, x stores that solution, which the pseudo-code does not record

// next subset of r indices of {0,...,n-1} in lexicographic order (false at the end)
static bool NextSubset(int* L, int r, int n)
{
    int t = r - 1;
    while (t >= 0 && L[t] == n - r + t) t--;
    if (t < 0) return false;
    L[t]++;
    for (int u = t + 1; u < r; ++u) L[u] = L[u - 1] + 1;
    return true;
}

int HlKP(int n, int c, int* w, int* p, int l, int* x)
{
    int* L    = new int[l];
    char* inL = new char[n];
    int* idx  = new int[n];
    int* pr   = new int[n];
```

```
    int* wr   = new int[n];
    int* xr   = new int[n];

    // lb := best value of the feasible sets of at most l-1 items
    int lb = 0;
    for (int j = 0; j < n; ++j) x[j] = 0;
    for (int r = 1; r <= l - 1; ++r)
    {
        for (int t = 0; t < r; ++t) L[t] = t;
        do
        {
            int wS = 0, pS = 0;
            for (int t = 0; t < r; ++t) { wS += w[L[t]]; pS += p[L[t]]; }
            if (wS <= c && pS > lb)
            {
                lb = pS;
                for (int j = 0; j < n; ++j) x[j] = 0;   // record the solution
                for (int t = 0; t < r; ++t) x[L[t]] = 1;
            }
        } while (NextSubset(L, r, n));
    }

    // foreach L in T(l): the feasible sets of exactly l items
    for (int t = 0; t < l; ++t) L[t] = t;
    do
    {
        int wL = 0, pL = 0, pmin = p[L[0]];
        for (int j = 0; j < n; ++j) inL[j] = 0;
        for (int t = 0; t < l; ++t)
        {
            wL += w[L[t]]; pL += p[L[t]]; inL[L[t]] = 1;
            if (p[L[t]] < pmin) pmin = p[L[t]];
        }
        if (wL > c) continue;                          // L is not in T(l)

        int cbar = c - wL;                             // residual capacity

        // residual set I(L)
        int m = 0;
        for (int j = 0; j < n; ++j)
        {
            if (!inL[j] && p[j] <= pmin && w[j] <= cbar)
            {
                idx[m] = j; pr[m] = p[j]; wr[m] = w[j]; m++;
            }
        }

        // complete the guess
        int lbL = pL + ((m > 0) ? ExtendedGreedyKP(m, cbar, wr, pr, xr) : 0);

        if (lbL > lb)
        {
            lb = lbL;
            for (int j = 0; j < n; ++j) x[j] = inL[j];  // record the solution
            for (int j = 0; j < m; ++j) if (xr[j] == 1) x[idx[j]] = 1;
        }
    } while (NextSubset(L, l, n));
```

```
80
81      delete[] L; delete[] inL; delete[] idx; delete[] pr; delete[] wr; delete[]
        xr;
82      return lb;
83  }
```

- Segue un'implementazione in `C++` della **programmazione dinamica sui profitti**, il secondo algoritmo di PD del capitolo «PD per il problema dello zaino 0/1» del modulo *Programmazione dinamica*, usato dall'algoritmo Basic-FPTAS. Con gli indici a partire da 0 del codice, la cella `u[i][q]` contiene il peso totale minimo di un sottoinsieme dei primi i oggetti di profitto totale esattamente q, e una soluzione ottima viene ricostruita all'indietro. Il parametro `qmax` tronca la tabella: si calcolano solo i profitti $q \leq \min\{P, \texttt{qmax}\}$, dove P è il profitto totale.

```
1  // dynamic programming by profits (second DP algorithm of the chapter "DP for
       the 0/1 knapsack problem")
2  // u(i,q) = minimum total weight of a subset of {0,...,i-1} with total profit
       exactly q
3  // the table is truncated at the profit qmax: only the values q = 0,...,min{P,
       qmax} are computed,
4  // where P is the total profit; qmax must be at least the optimal value of the
       instance
5  // the function returns the optimal value, i.e., the largest q with u(n,q) <= c
       ,
6  // and stores an optimal solution in x
7
8  int DPProfits(int n, int c, int* w, int* p, int qmax, int* x)
9  {
10     const int INF = 2147483647;
11     int P = 0;                                      // total profit
12     for (int i = 0; i < n; ++i) P += p[i];
13     if (P > qmax) P = qmax;                         // truncation of the table
14
15     // table u[i][q], i = 0,...,n and q = 0,...,P (P = min{total profit, qmax})
16     int** u = new int*[n + 1];
17     for (int i = 0; i <= n; ++i) u[i] = new int[P + 1];
18
19     u[0][0] = 0;
20     for (int q = 1; q <= P; ++q) u[0][q] = INF; // no item: only q = 0 is
       attained
21
22     for (int i = 1; i <= n; ++i)
23     {
24         for (int q = 0; q <= P; ++q)
25         {
26             u[i][q] = u[i - 1][q];                  // item i-1 is not selected
27             if (p[i - 1] <= q && u[i - 1][q - p[i - 1]] != INF
28                 && w[i - 1] + u[i - 1][q - p[i - 1]] < u[i][q])
29             {
30                 u[i][q] = w[i - 1] + u[i - 1][q - p[i - 1]];   // item i-1 is
       selected
31             }
32         }
33     }
```

```
34
35      // optimal value: the largest profit attained within the capacity
36      int qstar = 0;
37      for (int q = 0; q <= P; ++q) if (u[n][q] <= c) qstar = q;
38
39      // reconstruct an optimal solution backwards
40      int q = qstar;
41      for (int i = n; i >= 1; --i)
42      {
43          if (u[i][q] == u[i - 1][q]) x[i - 1] = 0;
44          else { x[i - 1] = 1; q -= p[i - 1]; }
45      }
46
47      for (int i = 0; i <= n; ++i) delete[] u[i];
48      delete[] u;
49      return qstar;
50  }
```

- Segue un'implementazione in `C++` dell'**algoritmo Basic-FPTAS**. L'accuratezza ε viene passata come frazione $\frac{\texttt{eps_num}}{\texttt{eps_den}}$, così che i profitti scalati $\lfloor \frac{p_j}{K} \rfloor = \left\lfloor \frac{p_j \, n \, \texttt{eps_den}}{\texttt{eps_num} \, GR^e(I)} \right\rfloor$ si calcolano in aritmetica intera esatta, senza errori di arrotondamento, e la tabella di `DPProfits` è troncata a $\lfloor 2n/\varepsilon \rfloor$.

```
1   // items are assumed to be sorted by non-increasing efficiency
2   // the accuracy is eps = eps_num / eps_den, with 0 < eps < 1
3   // x[i] stores the selection decision for item i
4   // the function returns the profit, in the original instance, of the solution x
5
6   int BasicFPTAS(int n, int c, int* w, int* p, int eps_num, int eps_den, int* x)
7   {
8       int lb = ExtendedGreedyKP(n, c, w, p, x);   // value of the Ext-Greedy
        solution
9
10      // scaling factor K = eps * lb / n and scaled profits p'[j] = floor(p[j] /
        K);
11      // with eps = eps_num / eps_den, p[j] / K = p[j] * n * eps_den / (eps_num *
         lb),
12      // which is computed in exact integer arithmetic
13      int* ps = new int[n];                       // scaled profits p'
14      for (int j = 0; j < n; ++j)
15      {
16          ps[j] = (int)(((long long)p[j] * n * eps_den) / ((long long)eps_num *
        lb));
17      }
18
19      // the optimal value of the scaled instance is at most floor(2n / eps): the
         table is truncated there
20      int qmax = (int)((2LL * n * eps_den) / eps_num);
21
22      DPProfits(n, c, w, ps, qmax, x);            // scaled instance solved to
        optimality
23
24      // profit of the solution x in the original instance
25      int z = 0;
26      for (int j = 0; j < n; ++j)
```

```
27      {
28          if (x[j] == 1) z = z + p[j];
29      }
30
31      delete[] ps;
32      return z;
33  }
```

- Tutte le funzioni di questa sezione sono nella cartella `KP_HEUR_CODE/src`. Il comando `make test` le compila ed esegue ogni esempio di questo capitolo e ogni istanza degli esercizi, confrontando il valore trovato con quello atteso.

Problemi di packing

Capitolo 3

Algoritmi di approssimazione per il bin packing

Prerequisiti

- Da questo modulo, il capitolo «Principi degli algoritmi di approssimazione»: rapporto di prestazione, garanzia di prestazione relativa, garanzia tight e rapporto di prestazione nel caso peggiore $r(\mathscr{A})$.
- Dal modulo *Problemi di ottimizzazione combinatoria*, il capitolo «Problemi di packing»: il BPP, le sue istanze e il lower bound di conteggio $\left\lceil \frac{\sum_{i \in \mathscr{I}} w_i}{c} \right\rceil$ sul valore ottimo.
- Dal modulo *Algoritmi euristici*, il capitolo «Euristiche costruttive»: algoritmi euristici che costruiscono una soluzione un oggetto alla volta.

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- eseguire a mano gli algoritmi Next Fit, First Fit, Best Fit e Worst Fit, e scrivere l'assegnamento oggetto-contenitore e il numero di contenitori;
- decidere se un algoritmo appartiene alla classe Any Fit, e se una soluzione soddisfa la proprietà di riempimento a metà;
- dimostrare che Next Fit e ogni algoritmo Any Fit hanno una garanzia di prestazione relativa pari a 2;
- descrivere le famiglie di istanze che rendono tight la garanzia 2 per Next Fit e Worst Fit, ed enunciare il rapporto nel caso peggiore $\frac{17}{10}$ di First Fit e Best Fit;
- leggere ed eseguire le implementazioni in `C++` dei quattro algoritmi.

Il problema di questo capitolo è il problema del bin packing (BPP): la sua definizione e le sue ipotesi sono nella sua pagina.

3.1 Notazione

- Come nella pagina del problema, c è la capacità dei contenitori e $\mathscr{I} = \{1, 2, \ldots, n\}$ l'insieme degli oggetti; in questo capitolo l'oggetto $i \in \mathscr{I}$ ha peso w_i.

- Una **soluzione ammissibile del BPP** è una partizione di $\mathscr{I}$ in **schemi di carico** (di impacchettamento) non vuoti, cioè sottoinsiemi di oggetti assegnati ciascuno a un contenitore, tali che il peso totale di ogni schema di carico non superi la capacità c dei contenitori.

- Indichiamo con $z(I)$ il **valore ottimo** del BPP sull'istanza I, cioè il numero minimo di contenitori, e con $\mathscr{A}(I)$ il numero di contenitori usati dalla soluzione ammissibile restituita da un algoritmo euristico $\mathscr{A}$. Poiché ogni algoritmo euristico di questo capitolo restituisce una soluzione ammissibile,

$$\mathscr{A}(I) \geq z(I) \qquad \text{e} \qquad \frac{\mathscr{A}(I)}{z(I)} \geq 1$$

  per ogni istanza I del BPP.

3.2 Next Fit

Il *Next Fit Algorithm* (indicato con NF) è un'euristica costruttiva per il BPP. Esamina gli oggetti nell'ordine in cui sono dati nell'istanza, mantenendo in ogni momento esattamente un contenitore aperto. Per ogni oggetto, NF lo pone nel contenitore aperto se vi entra; altrimenti chiude il contenitore, ne apre uno nuovo, e vi pone l'oggetto.

- **Struttura dati per la soluzione.** Usiamo un vettore $bin = (bin_1, bin_2, \ldots, bin_n)$ di lunghezza n per memorizzare la soluzione. In posizione i, il vettore contiene il numero del contenitore a cui è assegnato l'oggetto i. Più formalmente:

$$bin_i = b \quad \Longleftrightarrow \quad \text{l'oggetto } i \text{ è posto nel contenitore } b.$$

  Per esempio, se $bin_3 = 2$, l'oggetto 3 è assegnato al contenitore 2. Inoltre, manteniamo una variabile nb che memorizza il numero totale di contenitori usati, cioè alla fine dell'algoritmo $NF(I) = nb$.

- Lo pseudocodice di NF è il seguente:

Algoritmo

Algorithm: Next Fit

Input: istanza del BPP I con n oggetti, pesi $w[1..n]$, capacità c

Output: assegnamento oggetto-contenitore $bin[1..n]$ e numero di contenitori usati nb

```
1  Function NextFit(n, w, c)
2      bin := nuovo vettore[1..n]          // assegnamento oggetto-contenitore
3      nb := 1                             // contenitore corrente
4      rc := c                             // capacità residua
5      for i := 1 to n do
6          if w[i] > rc then
7              nb ← nb + 1                 // aprire un nuovo contenitore
8              rc ← c
9          end
10         bin[i] ← nb                     // assegnare l'oggetto al contenitore
11         rc ← rc − w[i]
12     end
13     return bin, nb
```

Esempio 3.1: Esecuzione di NF

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

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

Inizialmente $nb := 1$, $rc := c$. L'algoritmo NF produce la seguente soluzione ammissibile:

1. Oggetto 1 (peso $w[1] = 6$): $w[1] = 6 \leq rc = 10$, quindi non si apre un nuovo contenitore. L'oggetto 1 è posto nel contenitore $nb = 1$, e $rc \leftarrow 10 - 6 = 4$.
2. Oggetto 2 (peso $w[2] = 5$): $w[2] = 5 > rc = 4$, quindi $nb \leftarrow 2$ e $rc \leftarrow c = 10$. L'oggetto 2 è posto nel contenitore $nb = 2$, e $rc \leftarrow 10 - 5 = 5$.
3. Oggetto 3 (peso $w[3] = 4$): $w[3] = 4 \leq rc = 5$, quindi non si apre un nuovo contenitore. L'oggetto 3 è posto nel contenitore $nb = 2$, e $rc \leftarrow 5 - 4 = 1$.
4. Oggetto 4 (peso $w[4] = 7$): $w[4] = 7 > rc = 1$, quindi $nb \leftarrow 3$ e $rc \leftarrow c = 10$. L'oggetto 4 è posto nel contenitore $nb = 3$, e $rc \leftarrow 10 - 7 = 3$.
5. Oggetto 5 (peso $w[5] = 3$): $w[5] = 3 \leq rc = 3$, quindi non si apre un nuovo contenitore. L'oggetto 5 è posto nel contenitore $nb = 3$, e $rc \leftarrow 3 - 3 = 0$.

Quindi NF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 2 & 3 & 3 \end{pmatrix}, \qquad nb = 3.$$

e quindi

$$NF(I) = nb = 3.$$

- Dato l'output di un algoritmo, cioè il vettore $bin = (bin_1, bin_2, \ldots, bin_n)$ e il numero di contenitori nb, possiamo ottenere lo **schema di carico** P_b di ogni contenitore b, cioè l'insieme degli oggetti assegnati a esso, come segue:

$$P_b = \{\ i \in \{1, 2, \ldots, n\} : bin_i = b\ \}, \qquad \text{per } b \in \{1, 2, \ldots, nb\}.$$

  In altre parole, P_b è l'insieme di tutti gli oggetti i tali che $bin_i = b$; P_b è lo schema di carico del contenitore b, non il contenitore stesso.

Esempio 3.2: Pattern dal vettore bin

Consideriamo l'istanza dell'esempio precedente, su cui NF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 2 & 3 & 3 \end{pmatrix}, \qquad nb = 3.$$

Gli schemi di carico sono:

$$P_1 = \{i \in \{1, 2, 3, 4, 5\} : bin_i = 1\} = \{1\},$$
$$P_2 = \{i \in \{1, 2, 3, 4, 5\} : bin_i = 2\} = \{2, 3\},$$
$$P_3 = \{i \in \{1, 2, 3, 4, 5\} : bin_i = 3\} = \{4, 5\}.$$

Sono esattamente gli insiemi di oggetti posti nei contenitori costruiti dall'algoritmo:

- Il contenitore 1 contiene l'oggetto 1 con peso $w_1 = 6$;
- Il contenitore 2 contiene gli oggetti 2 e 3 con peso totale $w_2 + w_3 = 5 + 4 = 9$;
- Il contenitore 3 contiene gli oggetti 4 e 5 con peso totale $w_4 + w_5 = 7 + 3 = 10$.

3.3 La classe Any Fit

Un'euristica costruttiva per il BPP che mantiene tutti i contenitori aperti finora, e apre un nuovo contenitore *solo quando* l'oggetto corrente non entra in nessuno di essi, si chiama *Any Fit Algorithm* (indicato con AF). Any Fit non è quindi un singolo algoritmo ma una *famiglia* di algoritmi: ciò che la condizione lascia libero è la regola con cui scegliere un contenitore fra quelli in cui l'oggetto entra, e ogni regola dà un membro diverso della famiglia.

Definizione 3.1: Algoritmo Any Fit

Un'euristica costruttiva $\mathscr{A}$ per il BPP è un **algoritmo Any Fit** se, indicati con $P_1, P_2, \ldots, P_{nb}$ gli schemi di carico dei contenitori aperti finora, apre un nuovo contenitore per l'oggetto i solo quando

$$\sum_{h \in P_b} w_h + w_i > c \qquad \text{per ogni } b \in \{1, 2, \ldots, nb\},$$

mentre la scelta del contenitore, quando più di uno può accogliere l'oggetto, non è specificata.

- **Gli oggetti di due contenitori non entrano mai insieme.** Siano b e b' due contenitori aperti da un algoritmo Any Fit, con $b < b'$, sia f il primo oggetto posto nel contenitore b', e sia W_b il peso totale degli oggetti contenuti nel contenitore b nel momento in cui f è stato esaminato. Il contenitore b' è stato aperto perché f non entrava in nessuno dei contenitori aperti fino ad allora, e in particolare non nel contenitore b, quindi $W_b + w_f > c$. Il peso totale di un contenitore non diminuisce mai durante l'esecuzione, quindi $\sum_{i \in P_b} w_i \geq W_b$, e $f \in P_{b'}$ dà $\sum_{i \in P_{b'}} w_i \geq w_f$. Di conseguenza

  $$\sum_{i \in P_b} w_i + \sum_{i \in P_{b'}} w_i \;\geq\; W_b + w_f \;>\; c$$

  per ogni coppia di contenitori distinti b e b': di conseguenza, due contenitori di una soluzione Any Fit non si possono fondere. Qui si usa solo la condizione Any Fit, la regola di scelta non ha alcun ruolo.

- **NF non è un algoritmo Any Fit.** NF mantiene un solo contenitore aperto e lo chiude appena un oggetto non vi entra, anche quando quell'oggetto entrerebbe in un contenitore chiuso in precedenza.

Esempio 3.3: NF non è un algoritmo Any Fit

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

	$i = 1$	$i = 2$	$i = 3$	$i = 4$
w_i	3	8	3	8

Inizialmente $nb := 1$, $rc := c$. L'algoritmo NF produce la seguente soluzione ammissibile:

1. Oggetto 1 (peso $w[1] = 3$): $w[1] = 3 \leq rc = 10$, quindi non si apre un nuovo contenitore. L'oggetto 1 è posto nel contenitore $nb = 1$, e $rc \leftarrow 10 - 3 = 7$.
2. Oggetto 2 (peso $w[2] = 8$): $w[2] = 8 > rc = 7$, quindi $nb \leftarrow 2$ e $rc \leftarrow c = 10$. L'oggetto 2 è posto nel contenitore $nb = 2$, e $rc \leftarrow 10 - 8 = 2$.
3. Oggetto 3 (peso $w[3] = 3$): $w[3] = 3 > rc = 2$, quindi $nb \leftarrow 3$ e $rc \leftarrow c = 10$. L'oggetto 3 è posto nel contenitore $nb = 3$, e $rc \leftarrow 10 - 3 = 7$.
4. Oggetto 4 (peso $w[4] = 8$): $w[4] = 8 > rc = 7$, quindi $nb \leftarrow 4$ e $rc \leftarrow c = 10$. L'oggetto 4 è posto nel contenitore $nb = 4$, e $rc \leftarrow 10 - 8 = 2$.

Quindi NF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}, \qquad nb = 4.$$

Alla terza iterazione l'oggetto 3 entrerebbe nel contenitore 1, la cui capacità residua è 7, e NF apre comunque un nuovo contenitore: NF viola la condizione Any Fit.

Un algoritmo Any Fit, al contrario, non può aprire un terzo contenitore per l'oggetto 3, perché l'oggetto entra nel contenitore 1: vi pone l'oggetto 3 e poi apre un nuovo contenitore per l'oggetto 4, che non entra in nessuno dei due, usando 3 contenitori invece di 4. Questo vale per ogni algoritmo della classe, qualunque sia la regola di scelta.

3.3.1 First Fit

Il *First Fit Algorithm* (indicato con FF) è un'euristica costruttiva per il BPP. Esamina gli oggetti nell'ordine in cui sono dati nell'istanza, mantenendo tutti i contenitori aperti finora. Per ogni oggetto, FF scorre i contenitori nell'ordine di apertura e pone l'oggetto nel *primo* contenitore in cui entra; se nessun contenitore corrente può accogliere l'oggetto, FF apre un nuovo contenitore e vi pone l'oggetto.

- FF è un algoritmo Any Fit: apre un nuovo contenitore solo quando l'oggetto non entra in nessuno dei contenitori aperti finora, e la regola lasciata libera dalla Definizione 3.1 è qui fissata prendendo il primo contenitore in cui l'oggetto entra.

- Lo pseudocodice di FF è il seguente:

Algoritmo

Algorithm: First Fit

Input: istanza del BPP I con n oggetti, pesi $w[1..n]$, capacità c

Output: assegnamento oggetto-contenitore $bin[1..n]$ e numero di contenitori usati nb

```
1  Function FirstFit(n, w, c)
2      bin := nuovo vettore[1..n]          // assegnamento oggetto-contenitore
3      rc := nuovo vettore[1..n]           // capacità residue
4      nb := 1                             // numero di contenitori
5      rc[1] ← c                           // inizializzare il primo contenitore
6      for i := 1 to n do
7          s := −1                         // contenitore scelto
8          for b := 1 to nb do
9              if w[i] ≤ rc[b] then
10                 s ← b
11                 break
12             end
13         end
14         if s = −1 then
15             nb ← nb + 1                 // aprire un nuovo contenitore
16             rc[nb] ← c
17             s ← nb
18         end
19         bin[i] ← s                      // assegnare l'oggetto
20         rc[s] ← rc[s] − w[i]
21     end
22     return bin, nb
```

Esempio 3.4: Esecuzione di FF

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

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

L'algoritmo FF produce la seguente soluzione ammissibile:

1. Oggetto 1 (peso 6): il contenitore 1 è aperto all'inizio ($nb = 1$, $rc[1] = c = 10$) e $w[1] \leq rc[1]$, quindi $s = 1$ e FF pone l'oggetto 1 nel contenitore 1, e $rc[1] \leftarrow 10-6 = 4$.
2. Oggetto 2 (peso 5): $b = 1$: $w[2] = 5 \leq rc[1] = 4$? No. Nessun contenitore può accoglierlo ($s = -1$), quindi FF apre il contenitore 2 ($rc[2] \leftarrow 10$). L'oggetto 2 è posto nel contenitore 2, e $rc[2] \leftarrow 10 - 5 = 5$.

3. Oggetto 3 (peso 4): $b = 1$: $w[3] = 4 \leq rc[1] = 4$? Sì, quindi $s \leftarrow 1$ e la scansione si ferma. L'oggetto 3 è posto nel contenitore 1, e $rc[1] \leftarrow 4 - 4 = 0$.
4. Oggetto 4 (peso 7): $b = 1$: $w[4] = 7 \leq rc[1] = 0$? No. $b = 2$: $w[4] = 7 \leq rc[2] = 5$? No. Nessun contenitore può accoglierlo ($s = -1$), quindi FF apre il contenitore 3 ($rc[3] \leftarrow 10$). L'oggetto 4 è posto nel contenitore 3, e $rc[3] \leftarrow 10 - 7 = 3$.
5. Oggetto 5 (peso 3): $b = 1$: $w[5] = 3 \leq rc[1] = 0$? No. $b = 2$: $w[5] = 3 \leq rc[2] = 5$? Sì, quindi $s \leftarrow 2$ e la scansione si ferma. L'oggetto 5 è posto nel contenitore 2, e $rc[2] \leftarrow 5 - 3 = 2$.

Quindi FF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 & 2 \end{pmatrix}, \qquad nb = 3.$$

e quindi

$$FF(I) = nb = 3.$$

Esempio 3.5: Pattern dal vettore bin

Consideriamo l'istanza dell'esempio precedente, su cui FF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 & 2 \end{pmatrix}, \qquad nb = 3.$$

Gli schemi di carico sono:

$$P_1 = \{i \in \{1,2,3,4,5\} : bin_i = 1\} = \{1,3\},$$
$$P_2 = \{i \in \{1,2,3,4,5\} : bin_i = 2\} = \{2,5\},$$
$$P_3 = \{i \in \{1,2,3,4,5\} : bin_i = 3\} = \{4\}.$$

Sono esattamente gli insiemi di oggetti posti nei contenitori costruiti dall'algoritmo:

- Il contenitore 1 contiene gli oggetti 1 e 3 con peso totale $w_1 + w_3 = 6 + 4 = 10$;
- Il contenitore 2 contiene gli oggetti 2 e 5 con peso totale $w_2 + w_5 = 5 + 3 = 8$;
- Il contenitore 3 contiene l'oggetto 4 con peso $w_4 = 7$.

3.3.2 Best Fit

Il *Best Fit Algorithm* (indicato con BF) è un'euristica costruttiva per il BPP. Esamina gli oggetti nell'ordine in cui sono dati nell'istanza, mantenendo tutti i contenitori aperti finora. Per ogni oggetto, BF considera tutti i contenitori in cui l'oggetto entra e lo pone in quello con la *capacità residua minima* (in caso di pareggio, il contenitore aperto per primo); se nessun contenitore può accogliere l'oggetto, BF apre un nuovo contenitore e vi pone l'oggetto.

- L'idea è riempire i contenitori il più possibile: usando il contenitore in cui l'oggetto lascia meno spazio inutilizzato, BF tiene disponibili i contenitori con capacità residua maggiore per gli oggetti più grandi che possono ancora arrivare.

- BF è un algoritmo Any Fit: apre un nuovo contenitore solo quando l'oggetto non entra in nessuno dei contenitori aperti finora, e la regola lasciata libera dalla Definizione 3.1 è qui fissata prendendo il contenitore con la capacità residua minima.

- Lo pseudocodice di BF è il seguente:

Algoritmo

Algorithm: Best Fit

Input: istanza del BPP I con n oggetti, pesi $w[1..n]$, capacità c

Output: assegnamento oggetto-contenitore $bin[1..n]$ e numero di contenitori usati nb

```
1  Function BestFit(n, w, c)
2      bin := nuovo vettore[1..n]            // assegnamento oggetto-contenitore
3      rc := nuovo vettore[1..n]             // capacità residue
4      nb := 1                               // numero di contenitori
5      rc[1] ← c                             // inizializzare il primo contenitore
6      for i := 1 to n do
7          minrc := c + 1                    // capacità residua minima trovata finora
8          s := −1                           // contenitore scelto
9          for b := 1 to nb do
10             if w[i] ≤ rc[b] e rc[b] < minrc then
11                 s ← b
12                 minrc ← rc[b]
13         end
14     end
15     if s = −1 then
16         nb ← nb + 1                       // aprire un nuovo contenitore
17         rc[nb] ← c
18         s ← nb
19     end
20     bin[i] ← s                            // assegnare l'oggetto
21     rc[s] ← rc[s] − w[i]
22 end
23 return bin, nb
```

Esempio 3.6: Esecuzione di BF

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

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

L'algoritmo BF produce la seguente soluzione ammissibile:

1. Oggetto 1 (peso 6): il contenitore 1 è aperto all'inizio ($nb = 1$, $rc[1] = c = 10$) e $w[1] \leq rc[1]$, quindi $s = 1$ e BF pone l'oggetto 1 nel contenitore 1, e $rc[1] \leftarrow 10-6 = 4$.

2. Oggetto 2 (peso 5): $minrc := 11$, $s := -1$. $b = 1$: $w[2] = 5 \leq rc[1] = 4$? No. Nessun contenitore può accoglierlo ($s = -1$), quindi BF apre il contenitore 2 ($rc[2] \leftarrow 10$). L'oggetto 2 è posto nel contenitore 2, e $rc[2] \leftarrow 10 - 5 = 5$.

3. Oggetto 3 (peso 4): $minrc := 11$, $s := -1$. $b = 1$: $w[3] = 4 \leq rc[1] = 4$ e $rc[1] = 4 < 11$, quindi $s \leftarrow 1$, $minrc \leftarrow 4$. $b = 2$: $w[3] = 4 \leq rc[2] = 5$ e $rc[2] = 5 < minrc = 4$? No. Quindi $s = 1$: l'oggetto 3 è posto nel contenitore 1 (quello che lascia la capacità residua minima, 0, contro 1 del contenitore 2), e $rc[1] \leftarrow 4 - 4 = 0$.

4. Oggetto 4 (peso 7): $b = 1$: $w[4] = 7 \leq rc[1] = 0$? No. $b = 2$: $w[4] = 7 \leq rc[2] = 5$? No. Nessun contenitore può accoglierlo ($s = -1$), quindi BF apre il contenitore 3 ($rc[3] \leftarrow 10$). L'oggetto 4 è posto nel contenitore 3, e $rc[3] \leftarrow 10 - 7 = 3$.

5. Oggetto 5 (peso 3): $minrc := 11$, $s := -1$. $b = 1$: $w[5] = 3 \leq rc[1] = 0$? No. $b = 2$: $w[5] = 3 \leq rc[2] = 5$ e $rc[2] = 5 < 11$, quindi $s \leftarrow 2$, $minrc \leftarrow 5$. $b = 3$: $w[5] = 3 \leq rc[3] = 3$ e $rc[3] = 3 < minrc = 5$, quindi $s \leftarrow 3$, $minrc \leftarrow 3$. Quindi $s = 3$: l'oggetto 5 è posto nel contenitore 3 (quello che lascia la capacità residua minima, 0, contro 2 del contenitore 2), e $rc[3] \leftarrow 3 - 3 = 0$.

Quindi BF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 & 3 \end{pmatrix}, \qquad nb = 3.$$

e quindi

$$BF(I) = nb = 3.$$

Esempio 3.7: Pattern dal vettore bin

Consideriamo l'istanza dell'esempio precedente, su cui BF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 & 3 \end{pmatrix}, \qquad nb = 3.$$

Gli schemi di carico sono:

$$P_1 = \{i \in \{1,2,3,4,5\} : bin_i = 1\} = \{1,3\},$$
$$P_2 = \{i \in \{1,2,3,4,5\} : bin_i = 2\} = \{2\},$$
$$P_3 = \{i \in \{1,2,3,4,5\} : bin_i = 3\} = \{4,5\}.$$

Sono esattamente gli insiemi di oggetti posti nei contenitori costruiti dall'algoritmo:

- Il contenitore 1 contiene gli oggetti 1 e 3 con peso totale $w_1 + w_3 = 6 + 4 = 10$;
- Il contenitore 2 contiene l'oggetto 2 con peso $w_2 = 5$;
- Il contenitore 3 contiene gli oggetti 4 e 5 con peso totale $w_4 + w_5 = 7 + 3 = 10$.

3.3.3 Worst Fit

Il *Worst Fit Algorithm* (indicato con WF) è un'euristica costruttiva per il BPP. Esamina gli oggetti nell'ordine in cui sono dati nell'istanza, mantenendo tutti i contenitori aperti finora. Per ogni oggetto, WF considera tutti i contenitori in cui l'oggetto entra e lo pone in quello con la *capacità residua massima* (in caso di pareggio, il contenitore aperto per primo); se nessun contenitore può accogliere l'oggetto, WF apre un nuovo contenitore e vi pone l'oggetto.

- L'idea è opposta a quella di BF: ponendo ogni oggetto nel contenitore più vuoto, WF evita di creare contenitori con una capacità residua così piccola che nessun oggetto futuro possa usarla, e mantiene lo spazio libero distribuito fra i contenitori aperti.

- WF è un algoritmo Any Fit: apre un nuovo contenitore solo quando l'oggetto non entra in nessuno dei contenitori aperti finora, e la regola lasciata libera dalla Definizione 3.1 è qui fissata prendendo il contenitore con la capacità residua massima.

- Lo pseudocodice di WF è il seguente:

Algoritmo

Algorithm: Worst Fit

Input: istanza del BPP I con n oggetti, pesi $w[1..n]$, capacità c

Output: assegnamento oggetto-contenitore $bin[1..n]$ e numero di contenitori usati nb

```
1  Function WorstFit(n, w, c)
2      bin := nuovo vettore[1..n]          // assegnamento oggetto-contenitore
3      rc := nuovo vettore[1..n]           // capacità residue
4      nb := 1                             // numero di contenitori
5      rc[1] ← c                           // inizializzare il primo contenitore
6      for i := 1 to n do
7          maxrc := −1                     // capacità residua massima trovata finora
8          s := −1                         // contenitore scelto
9          for b := 1 to nb do
10             if w[i] ≤ rc[b] e rc[b] > maxrc then
11                 s ← b
12                 maxrc ← rc[b]
13             end
14         end
15         if s = −1 then
16             nb ← nb + 1                 // aprire un nuovo contenitore
17             rc[nb] ← c
18             s ← nb
19         end
20         bin[i] ← s                      // assegnare l'oggetto
21         rc[s] ← rc[s] − w[i]
22     end
23     return bin, nb
```

Esempio 3.8: Esecuzione di WF

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

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

L'algoritmo WF produce la seguente soluzione ammissibile:

1. Oggetto 1 (peso 6): il contenitore 1 è aperto all'inizio ($nb = 1$, $rc[1] = c = 10$) e $w[1] \leq rc[1]$, quindi $s = 1$ e WF pone l'oggetto 1 nel contenitore 1, e $rc[1] \leftarrow 10-6 = 4$.

2. Oggetto 2 (peso 5): $maxrc := -1$, $s := -1$. $b = 1$: $w[2] = 5 \leq rc[1] = 4$? No. Nessun contenitore può accoglierlo ($s = -1$), quindi WF apre il contenitore 2 ($rc[2] \leftarrow 10$). L'oggetto 2 è posto nel contenitore 2, e $rc[2] \leftarrow 10 - 5 = 5$.

3. Oggetto 3 (peso 4): $maxrc := -1$, $s := -1$. $b = 1$: $w[3] = 4 \leq rc[1] = 4$ e $rc[1] = 4 > -1$, quindi $s \leftarrow 1$, $maxrc \leftarrow 4$. $b = 2$: $w[3] = 4 \leq rc[2] = 5$ e $rc[2] = 5 > maxrc = 4$, quindi $s \leftarrow 2$, $maxrc \leftarrow 5$. Quindi $s = 2$: l'oggetto 3 è posto nel contenitore 2 (quello con la capacità residua massima), e $rc[2] \leftarrow 5 - 4 = 1$.

4. Oggetto 4 (peso 7): $b = 1$: $w[4] = 7 \leq rc[1] = 4$? No. $b = 2$: $w[4] = 7 \leq rc[2] = 1$? No. Nessun contenitore può accoglierlo ($s = -1$), quindi WF apre il contenitore 3 ($rc[3] \leftarrow 10$). L'oggetto 4 è posto nel contenitore 3, e $rc[3] \leftarrow 10 - 7 = 3$.

5. Oggetto 5 (peso 3): $maxrc := -1$, $s := -1$. $b = 1$: $w[5] = 3 \leq rc[1] = 4$ e $rc[1] = 4 > -1$, quindi $s \leftarrow 1$, $maxrc \leftarrow 4$. $b = 2$: $w[5] = 3 \leq rc[2] = 1$? No. $b = 3$: $w[5] = 3 \leq rc[3] = 3$ e $rc[3] = 3 > maxrc = 4$? No. Quindi $s = 1$: l'oggetto 5 è posto nel contenitore 1 (quello con la capacità residua massima), e $rc[1] \leftarrow 4 - 3 = 1$.

Quindi WF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 2 & 3 & 1 \end{pmatrix}, \qquad nb = 3.$$

e quindi

$$WF(I) = nb = 3.$$

Esempio 3.9: Pattern dal vettore bin

Consideriamo l'istanza dell'esempio precedente, su cui WF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 2 & 3 & 1 \end{pmatrix}, \qquad nb = 3.$$

Gli schemi di carico sono:

$$P_1 = \{i \in \{1,2,3,4,5\} : bin_i = 1\} = \{1,5\},$$
$$P_2 = \{i \in \{1,2,3,4,5\} : bin_i = 2\} = \{2,3\},$$
$$P_3 = \{i \in \{1,2,3,4,5\} : bin_i = 3\} = \{4\}.$$

Sono esattamente gli insiemi di oggetti posti nei contenitori costruiti dall'algoritmo:

- Il contenitore 1 contiene gli oggetti 1 e 5 con peso totale $w_1 + w_5 = 6 + 3 = 9$;
- Il contenitore 2 contiene gli oggetti 2 e 3 con peso totale $w_2 + w_3 = 5 + 4 = 9$;
- Il contenitore 3 contiene l'oggetto 4 con peso $w_4 = 7$.

3.4 Proprietà di riempimento a metà

- I risultati di questa sezione non dipendono dalla regola usata per scegliere il contenitore: si dimostrano usando *solo* la condizione Any Fit, e valgono quindi per ogni algoritmo della famiglia, compresi FF, BF e WF.
- La **proprietà di riempimento a metà** di una soluzione ammissibile del BPP afferma che, con l'eccezione di al più un contenitore, tutti i contenitori sono riempiti per più della metà della loro capacità o, equivalentemente, che per ogni coppia di contenitori distinti almeno uno dei due è riempito per più della metà della capacità.
- La proprietà di riempimento a metà da sola *non* garantisce che due contenitori non si possano fondere: per esempio, con $c = 10$ due contenitori di peso totale 6 e 1 soddisfano la proprietà ma si possono fondere in un solo contenitore. Il suo ruolo è diverso: è esattamente ciò che serve per dimostrare una garanzia di prestazione relativa pari a 2.

Definizione 3.2: Proprietà di riempimento a metà

Data una soluzione ammissibile del BPP con k contenitori e schemi di carico $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ (dove P_b è l'insieme degli oggetti posti nel contenitore b, per $b \in \{1, 2, \ldots, k\}$), diciamo che la soluzione soddisfa la **proprietà di riempimento a metà** se vale una delle seguenti condizioni equivalenti:

- esiste al più un contenitore $b \in \{1, 2, \ldots, k\}$ tale che
$$\sum_{i \in P_b} w_i \leq \frac{c}{2}.$$
- per ogni coppia di contenitori distinti $b, b' \in \{1, 2, \ldots, k\}$, almeno uno dei due è riempito per più della metà della sua capacità, cioè
$$\sum_{i \in P_b} w_i > \frac{c}{2} \quad \text{o} \quad \sum_{i \in P_{b'}} w_i > \frac{c}{2}.$$

Esempio 3.10: Proprietà di riempimento a metà

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

	$i = 1$	$i = 2$	$i = 3$	$i = 4$
w_i	5	5	5	5

Inizialmente $nb := 1$, $rc := c$. L'algoritmo NF produce la seguente soluzione ammissibile:

1. Oggetto 1 (peso $w[1] = 5$): $w[1] = 5 \leq rc = 10$, quindi non si apre un nuovo contenitore. L'oggetto 1 è posto nel contenitore $nb = 1$, e $rc \leftarrow 10 - 5 = 5$.

2. Oggetto 2 (peso $w[2] = 5$): $w[2] = 5 \leq rc = 5$, quindi non si apre un nuovo contenitore. L'oggetto 2 è posto nel contenitore $nb = 1$, e $rc \leftarrow 5 - 5 = 0$.

3. Oggetto 3 (peso $w[3] = 5$): $w[3] = 5 > rc = 0$, quindi $nb \leftarrow 2$ e $rc \leftarrow c = 10$. L'oggetto 3 è posto nel contenitore $nb = 2$, e $rc \leftarrow 10 - 5 = 5$.

4. Oggetto 4 (peso $w[4] = 5$): $w[4] = 5 \leq rc = 5$, quindi non si apre un nuovo contenitore. L'oggetto 4 è posto nel contenitore $nb = 2$, e $rc \leftarrow 5 - 5 = 0$.

Quindi NF restituisce:

$$bin = \begin{pmatrix} 1 & 1 & 2 & 2 \end{pmatrix}, \qquad nb = 2.$$

Gli schemi di carico sono $P_1 = \{1, 2\}$, $P_2 = \{3, 4\}$, con pesi totali:

$$\sum_{i \in P_1} w_i = 5 + 5 = 10 > \frac{c}{2} = 5, \qquad \sum_{i \in P_2} w_i = 5 + 5 = 10 > \frac{c}{2} = 5.$$

Poiché *nessun* contenitore ha peso totale al più $\frac{c}{2}$, la soluzione **soddisfa** la proprietà di riempimento a metà.

Proposizione 3.3

Per ogni istanza del BPP, le soluzioni euristiche prodotte da *qualsiasi* algoritmo Any Fit soddisfano la proprietà di riempimento a metà. In particolare questo vale per FF, BF e WF.

Dimostrazione. Sia $\mathscr{A}$ un algoritmo Any Fit, sia $k := \mathscr{A}(I)$, e siano $P_1, P_2, \ldots, P_k$ gli schemi di carico prodotti da $\mathscr{A}$, indicizzati nell'ordine di apertura dei contenitori. Se $k = 1$ non c'è niente da dimostrare, quindi supponiamo $k \geq 2$.

Supponiamo, per assurdo, che esistano due contenitori distinti b e b' tali che

$$\sum_{i \in P_b} w_i \leq \frac{c}{2} \qquad \text{e} \qquad \sum_{i \in P_{b'}} w_i \leq \frac{c}{2}.$$

Sommando le due disuguaglianze si ottiene

$$\sum_{i \in P_b} w_i + \sum_{i \in P_{b'}} w_i \leq \frac{c}{2} + \frac{c}{2} = c,$$

che contraddice la proprietà della classe Any Fit dimostrata nella Sezione 3.3, cioè che gli oggetti di due contenitori distinti non entrano mai insieme in un solo contenitore. Di conseguenza può esserci al più un contenitore con peso totale al più $\frac{c}{2}$.

Si usa solo la condizione Any Fit, quindi l'enunciato vale per ogni algoritmo della classe. □

Esempio 3.11: Proprietà di riempimento a metà

Come illustrazione della Proposizione 3.3, consideriamo contenitori di capacità $c = 10$ e un insieme $\mathscr{I} = \{1, 2, 3, 4\}$ di 4 oggetti ($n = 4$), con i seguenti pesi degli oggetti:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$
w_i	7	7	3	4

Inizialmente $nb := 1$, $rc[1] := c = 10$. L'algoritmo FF produce la seguente soluzione ammissibile:

1. Oggetto 1 (peso 7): $b = 1$: $w[1] = 7 \leq rc[1] = 10$? Sì, quindi $s \leftarrow 1$. L'oggetto 1 è posto nel contenitore 1, e $rc[1] \leftarrow 10 - 7 = 3$.
2. Oggetto 2 (peso 7): $b = 1$: $w[2] = 7 \leq rc[1] = 3$? No. Nessun contenitore può accoglierlo, quindi FF apre il contenitore 2 ($rc[2] \leftarrow 10$). L'oggetto 2 è posto nel contenitore 2, e $rc[2] \leftarrow 10 - 7 = 3$.
3. Oggetto 3 (peso 3): $b = 1$: $w[3] = 3 \leq rc[1] = 3$? Sì, quindi $s \leftarrow 1$. L'oggetto 3 è posto nel contenitore 1, e $rc[1] \leftarrow 3 - 3 = 0$.
4. Oggetto 4 (peso 4): $b = 1$: $w[4] = 4 \leq rc[1] = 0$? No. $b = 2$: $w[4] = 4 \leq rc[2] = 3$? No. Nessun contenitore può accoglierlo, quindi FF apre il contenitore 3 ($rc[3] \leftarrow 10$). L'oggetto 4 è posto nel contenitore 3, e $rc[3] \leftarrow 10 - 4 = 6$.

Quindi FF restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 1 & 3 \end{pmatrix}, \qquad nb = 3.$$

Gli schemi di carico sono $P_1 = \{1, 3\}$, $P_2 = \{2\}$, $P_3 = \{4\}$, con pesi totali:

$$\sum_{i \in P_1} w_i = 7 + 3 = 10 > \frac{c}{2} = 5, \qquad \sum_{i \in P_2} w_i = 7 > \frac{c}{2} = 5, \qquad \sum_{i \in P_3} w_i = 4 \leq \frac{c}{2} = 5.$$

Poiché c'è *esattamente un* contenitore (il contenitore 3) con peso totale al più $\frac{c}{2}$, la soluzione **soddisfa** la proprietà di riempimento a metà, come garantito dalla Proposizione 3.3, poiché FF è un algoritmo Any Fit.

Il Next Fit può produrre soluzioni con molti contenitori riempiti per meno della metà della capacità, e quindi non soddisfa la proprietà di riempimento a metà.

Esempio 3.12: Proprietà di riempimento a metà

Consideriamo di nuovo l'istanza dell'Esempio 3.3, con capacità dei contenitori $c = 10$ e pesi degli oggetti $w = (3, 8, 3, 8)$, su cui NF apre un nuovo contenitore per ogni oggetto e restituisce:

$$bin = \begin{pmatrix} 1 & 2 & 3 & 4 \end{pmatrix}, \qquad nb = 4.$$

Gli schemi di carico sono:

$$P_1 = \{i \in \{1,2,3,4\} : bin_i = 1\} = \{1\},$$
$$P_2 = \{i \in \{1,2,3,4\} : bin_i = 2\} = \{2\},$$
$$P_3 = \{i \in \{1,2,3,4\} : bin_i = 3\} = \{3\},$$
$$P_4 = \{i \in \{1,2,3,4\} : bin_i = 4\} = \{4\}.$$

I pesi totali nei contenitori sono:

$$\sum_{i \in P_1} w_i = 3 \leq \frac{c}{2} = 5, \quad \sum_{i \in P_2} w_i = 8 > \frac{c}{2} = 5, \quad \sum_{i \in P_3} w_i = 3 \leq \frac{c}{2} = 5, \quad \sum_{i \in P_4} w_i = 8 > \frac{c}{2} = 5.$$

Poiché ci sono *due* contenitori (i contenitori 1 e 3) con peso totale al più $\frac{c}{2}$, la soluzione **non** soddisfa la proprietà di riempimento a metà.

3.5 Garanzie di prestazione

- Determiniamo ora il rapporto di prestazione nel caso peggiore dei quattro algoritmi. Next Fit non è un algoritmo Any Fit, quindi richiede un argomento a sé. Per i tre algoritmi Any Fit il bound $\mathscr{A}(I) \leq 2\, z(I)$ segue subito dalla proprietà di riempimento a metà, senza alcuna ipotesi sulla regola usata per scegliere il contenitore, e la domanda è quanto sia tight questo bound: per WF il fattore 2 non si può migliorare, mentre FF e BF rispettano un bound strettamente migliore.

Proposizione 3.4

Sia $\mathscr{A}$ un algoritmo euristico per il BPP che produce soluzioni che soddisfano la proprietà di riempimento a metà. Allora, per ogni istanza I del BPP, abbiamo

$$\frac{\mathscr{A}(I)}{z(I)} \leq 2.$$

Dimostrazione. Sia $k := \mathscr{A}(I)$, e sia $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ l'insieme degli schemi di carico prodotti da $\mathscr{A}$, indicizzati nell'ordine di apertura dei contenitori.

Poiché la soluzione soddisfa la proprietà di riempimento a metà, esiste al più un contenitore $b \in \{1, 2, \ldots, k\}$ tale che

$$\sum_{i \in P_b} w_i \leq \frac{c}{2}.$$

Di conseguenza, almeno $k-1$ contenitori hanno peso totale strettamente maggiore di $\frac{c}{2}$. Sommando su questi contenitori, otteniamo

$$\sum_{i \in \mathscr{I}} w_i \;>\; (k-1)\,\frac{c}{2}.$$

D'altra parte, ogni impacchettamento ammissibile che usa $z(I)$ contenitori soddisfa

$$\sum_{i \in \mathscr{I}} w_i \;\leq\; z(I)\, c.$$

Combinando le due disuguaglianze si ottiene

$$(k-1)\,\frac{c}{2} \;<\; \sum_{i \in \mathscr{I}} w_i \;\leq\; z(I)\, c \qquad \Longrightarrow \qquad k-1 \;<\; 2\, z(I) \qquad \Longrightarrow \qquad k \;<\; 2\, z(I)+1.$$

Poiché sia k sia $z(I)$ sono interi positivi, abbiamo

$$k \;\leq\; 2\, z(I).$$

Dividendo per $z(I) \geq 1$ si ottiene

$$\frac{k}{z(I)} = \frac{\mathscr{A}(I)}{z(I)} \;\leq\; 2.$$

□

- Poiché ogni algoritmo Any Fit produce soluzioni che soddisfano la proprietà di riempimento a metà, segue immediatamente che

  $$\frac{\mathscr{A}(I)}{z(I)} \leq 2 \qquad \text{per ogni algoritmo Any Fit } \mathscr{A}$$

  e per ogni istanza I del BPP. In particolare

  $$\frac{FF(I)}{z(I)} \leq 2, \qquad \frac{BF(I)}{z(I)} \leq 2 \qquad \text{e} \qquad \frac{WF(I)}{z(I)} \leq 2,$$

  e quindi FF, BF e WF sono algoritmi di 2-approssimazione.

3.5.1 Il rapporto di prestazione nel caso peggiore di NF

Proposizione 3.5

Per ogni istanza I del BPP, abbiamo:

$$\frac{NF(I)}{z(I)} \leq 2. \tag{3.1}$$

Dimostrazione. Sia $k := NF(I)$, e siano $P_1, P_2, \ldots, P_k$ gli schemi di carico prodotti da NF, indicizzati nell'ordine di apertura dei contenitori.

Fissiamo un qualsiasi $b \in \{1, 2, \ldots, k-1\}$. Lo schema di carico P_b corrisponde a un contenitore che viene chiuso quando l'oggetto successivo (chiamiamolo f) non vi entra,

quindi

$$\sum_{i \in P_b} w_i + w_f > c.$$

Lo stesso oggetto f è il primo oggetto posto nel contenitore successivo, il cui schema di carico è P_{b+1}, quindi

$$w_f \leq \sum_{i \in P_{b+1}} w_i.$$

Di conseguenza,

$$\sum_{i \in P_b} w_i + \sum_{i \in P_{b+1}} w_i \; > \; c.$$

Consideriamo ora le coppie consecutive disgiunte $(P_1, P_2), (P_3, P_4), \ldots$. Sommando la disuguaglianza precedente sulle $\lfloor k/2 \rfloor$ coppie si ottiene

$$\underbrace{\sum_{b=1}^{k} \sum_{i \in P_b} w_i}_{= \sum_{i \in \mathscr{I}} w_i \; \leq \; z(I)\, c} \; > \left\lfloor \frac{k}{2} \right\rfloor c \quad \Longrightarrow \quad z(I) \; > \; \left\lfloor \frac{k}{2} \right\rfloor.$$

Poiché $z(I)$ è intero, segue che

$$z(I) \geq \left\lfloor \frac{k}{2} \right\rfloor + 1.$$

Allora

$$\begin{cases} \text{se } k \text{ è pari, allora } \left\lfloor \frac{k}{2} \right\rfloor = \frac{k}{2} \implies z(I) \geq \frac{k}{2} + 1 \implies k \leq 2z(I) - 2 < 2z(I), \\ \text{se } k \text{ è dispari, allora } \left\lfloor \frac{k}{2} \right\rfloor = \frac{k-1}{2} \implies z(I) \geq \frac{k+1}{2} \implies k \leq 2z(I) - 1 < 2z(I). \end{cases}$$

Di conseguenza $k < 2z(I)$, e dividendo per $z(I) \geq 1$ si ottiene

$$\frac{k}{z(I)} = \frac{NF(I)}{z(I)} \leq 2.$$

□

Proposizione 3.6

Esistono istanze I_n del BPP tali che:

$$\frac{NF(I_n)}{z(I_n)} \; \to \; 2 \qquad \text{per} \qquad n \to +\infty. \tag{3.2}$$

Dimostrazione. Fissiamo un intero $n \geq 8$ divisibile per 4 e consideriamo l'istanza I_n del BPP con capacità dei contenitori $c = n$ e insieme di oggetti $\mathscr{I} = \{1, 2, \ldots, n\}$, dove i

pesi degli oggetti sono

$$w_i = \begin{cases} \frac{n}{2}, & \text{se } i \text{ è dispari,} \\ 2, & \text{se } i \text{ è pari.} \end{cases}$$

Gli oggetti sono esaminati da NF nell'ordine naturale $1, 2, \ldots, n$. Per ogni i dispari,

$$w_i + w_{i+1} = \frac{n}{2} + 2 < n,$$

quindi NF pone gli oggetti i e $i+1$ nello stesso contenitore. Lo spazio rimanente è

$$n - \left(\frac{n}{2} + 2\right) = \frac{n}{2} - 2 < \frac{n}{2},$$

quindi l'oggetto successivo, di peso $\frac{n}{2}$, non entra e NF apre un nuovo contenitore. Di conseguenza,

$$NF(I_n) = \frac{n}{2}.$$

Ponendo gli $\frac{n}{2}$ oggetti di peso $\frac{n}{2}$ due per contenitore si ottengono $\frac{n}{4}$ contenitori pieni, e ponendo tutti gli $\frac{n}{2}$ oggetti piccoli insieme (poiché $\frac{n}{2} \cdot 2 = n$) in un contenitore aggiuntivo si ha $z(I_n) \leq \frac{n}{4} + 1$. D'altra parte,

$$z(I_n) \geq \left\lceil \frac{\sum_{i \in \mathscr{I}} w_i}{c} \right\rceil = \left\lceil \frac{\frac{n}{2}\,\frac{n}{2} + \frac{n}{2}\,2}{n} \right\rceil = \left\lceil \frac{n}{4} + 1 \right\rceil = \frac{n}{4} + 1,$$

Quindi

$$z(I_n) = \frac{n}{4} + 1.$$

Di conseguenza,

$$\lim_{n\to\infty} \frac{NF(I_n)}{z(I_n)} = \lim_{n\to\infty} \frac{\frac{n}{2}}{\frac{n}{4}+1} = \lim_{n\to\infty} \frac{\frac{n}{2}}{\frac{n+4}{4}} = \lim_{n\to\infty} \frac{2n}{n+4} = \lim_{n\to\infty} \frac{2}{1+\frac{4}{n}} = 2.$$

□

Dalla Proposizione 3.5 e dalla Proposizione 3.6 segue che il rapporto di prestazione nel caso peggiore di NF è due, cioè abbiamo

$$r(NF) = 2$$

3.5.2 Il rapporto di prestazione nel caso peggiore di WF

Proposizione 3.7

Esistono istanze I_n del BPP tali che:

$$\frac{WF(I_n)}{z(I_n)} \to 2 \qquad \text{per} \qquad n \to +\infty. \tag{3.3}$$

Dimostrazione. Fissiamo un intero $n \geq 8$ divisibile per 4 e consideriamo l'istanza I_n del BPP con capacità dei contenitori $c = n$ e insieme di oggetti $\mathscr{I} = \{1, 2, \ldots, n\}$, dove i pesi degli oggetti sono

$$w_i = \begin{cases} \dfrac{n}{2}, & \text{se } i \text{ è dispari,} \\ 2, & \text{se } i \text{ è pari,} \end{cases}$$

cioè la stessa istanza usata sopra per NF. Gli oggetti sono esaminati da WF nell'ordine naturale $1, 2, \ldots, n$.

Mostriamo per induzione che, dopo che sono stati esaminati i primi $2k$ oggetti, WF ha aperto esattamente k contenitori, ciascuno con capacità residua $\frac{n}{2} - 2$.

Per $k = 1$, l'oggetto 1 (di peso $\frac{n}{2}$) è posto nel contenitore 1, la cui capacità residua diventa $\frac{n}{2}$; poi l'oggetto 2 (di peso 2) è posto nel contenitore 1, l'unico contenitore aperto, la cui capacità residua diventa $\frac{n}{2} - 2$.

Supponiamo che l'affermazione valga per un certo k, e consideriamo gli oggetti $2k+1$ e $2k+2$. L'oggetto $2k+1$ ha peso $\frac{n}{2}$, mentre ogni contenitore aperto ha capacità residua $\frac{n}{2} - 2 < \frac{n}{2}$: l'oggetto non entra da nessuna parte, quindi WF apre il contenitore $k+1$, la cui capacità residua diventa $\frac{n}{2}$. L'oggetto $2k+2$ ha peso 2 ed entra in ogni contenitore aperto; poiché il contenitore $k+1$ è quello con la capacità residua *massima*, WF vi pone l'oggetto, e la capacità residua del contenitore $k+1$ diventa $\frac{n}{2} - 2$. Questo dimostra l'affermazione per $k+1$.

È proprio la regola Worst Fit a mantenere ogni contenitore a capacità residua $\frac{n}{2} - 2$: gli oggetti piccoli sono sempre posti nel contenitore più vuoto, quindi nessun contenitore riceve mai due oggetti di peso $\frac{n}{2}$. Prendendo $k = \frac{n}{2}$ otteniamo

$$WF(I_n) = \frac{n}{2}.$$

Ponendo gli $\frac{n}{2}$ oggetti di peso $\frac{n}{2}$ due per contenitore si ottengono $\frac{n}{4}$ contenitori pieni, e ponendo tutti gli $\frac{n}{2}$ oggetti piccoli insieme (poiché $\frac{n}{2} \cdot 2 = n$) in un contenitore aggiuntivo si ha $z(I_n) \leq \frac{n}{4} + 1$. D'altra parte,

$$z(I_n) \geq \left\lceil \frac{\sum_{i \in \mathscr{I}} w_i}{c} \right\rceil = \left\lceil \frac{\frac{n}{2}\,\frac{n}{2} + \frac{n}{2}\,2}{n} \right\rceil = \left\lceil \frac{n}{4} + 1 \right\rceil = \frac{n}{4} + 1,$$

quindi

$$z(I_n) = \frac{n}{4} + 1.$$

Di conseguenza,

$$\lim_{n\to\infty} \frac{WF(I_n)}{z(I_n)} = \lim_{n\to\infty} \frac{\frac{n}{2}}{\frac{n}{4}+1} = \lim_{n\to\infty} \frac{2n}{n+4} = \lim_{n\to\infty} \frac{2}{1+\frac{4}{n}} = 2.$$

□

Dalla Proposizione 3.4 e dalla Proposizione 3.7 segue che il rapporto di prestazione nel caso peggiore di WF è due, cioè abbiamo

$$r(WF) = 2$$

Per WF il fattore 2 non si può quindi migliorare.

- Abbiamo $r(WF) = r(NF) = 2$: nel caso peggiore WF non è migliore di NF. I due algoritmi sono però lontani dall'essere equivalenti. WF non apre mai un nuovo contenitore finché l'oggetto entra ancora in uno dei contenitori aperti finora, mentre NF chiude un contenitore appena un oggetto non vi entra e non lo riconsidera mai. Di conseguenza NF può lasciare un numero arbitrario di contenitori riempiti per meno della metà della loro capacità, mentre WF, come mostrato nella sezione sulla proprietà di riempimento a metà, può lasciarne al più uno.

- Il valore $r(WF) = 2$ è dovuto a

  D.S. Johnson, *Fast algorithms for bin packing*, Journal of Computer and System Sciences, 8(3):272–314, 1974.

3.5.3 I rapporti di prestazione nel caso peggiore di FF e BF

Proposizione 3.8

Per ogni istanza I del BPP, abbiamo:

$$FF(I) \leq \frac{17}{10} z(I) \quad \text{e} \quad BF(I) \leq \frac{17}{10} z(I). \tag{3.4}$$

Dimostrazione. Dimostrazione omessa. I due bound sono stati dimostrati per la prima volta nella forma più debole

$$FF(I) \leq \frac{17}{10} z(I) + 2 \qquad \text{e} \qquad BF(I) \leq \frac{17}{10} z(I) + 2$$

in

D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey e R.L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM Journal on

Computing, 3(4):299–325, 1974.

La costante additiva è stata poi eliminata, nella forma enunciata sopra, rispettivamente in

G. Dósa e J. Sgall, *First Fit bin packing: a tight analysis*, STACS 2013, pp. 538–549,

G. Dósa e J. Sgall, *Optimal analysis of Best Fit bin packing*, ICALP 2014, pp. 429–441. □

Proposizione 3.9

Esistono istanze I_n del BPP tali che:

$$\frac{FF(I_n)}{z(I_n)} \to \frac{17}{10} \quad \text{e} \quad \frac{BF(I_n)}{z(I_n)} \to \frac{17}{10} \quad \text{per} \quad n \to +\infty. \tag{3.5}$$

Dimostrazione. Dimostrazione omessa. Famiglie di istanze su cui il rapporto di FF e BF tende a $\frac{17}{10}$ sono date nell'articolo di Johnson, Demers, Ullman, Garey e Graham citato sopra, e in

D.S. Johnson, *Near-optimal bin packing algorithms*, tesi di dottorato, Massachusetts Institute of Technology, 1973. □

Dalla Proposizione 3.8 e dalla Proposizione 3.9 segue che il rapporto di prestazione nel caso peggiore di FF e BF è diciassette decimi, cioè abbiamo

$$r(FF) = r(BF) = \frac{17}{10}$$

A differenza di WF, FF e BF restano strettamente sotto il fattore 2 che la sola condizione Any Fit garantisce.

3.5.4 Che cosa dà e che cosa non dà la condizione Any Fit

- **Che cosa dà la condizione Any Fit.** La sola condizione Any Fit garantisce la proprietà di riempimento a metà, e quindi il bound

$$\mathscr{A}(I) \leq 2\, z(I)$$

per ogni algoritmo $\mathscr{A}$ della classe. Questo bound non si può migliorare per la classe nel suo complesso: WF è un algoritmo Any Fit e, come mostrato sopra, $r(WF) = 2$.

- **Che cosa lascia libero la condizione Any Fit.** La regola usata per scegliere uno dei contenitori in cui l'oggetto entra è ciò che distingue i membri della famiglia. Sulle istanze della Proposizione 3.9 il rapporto di FF e BF tende a $\frac{17}{10}$, mentre su quelle della Proposizione 3.7 il rapporto di WF tende a 2: la sola regola di scelta spiega tutto il divario fra $\frac{17}{10}$ e 2.

- È noto che nessun algoritmo della classe può scendere sotto $\frac{17}{10}$, quindi $\frac{17}{10}$ e 2 sono esattamente il comportamento migliore e quello peggiore che la condizione Any Fit consente, ed entrambi sono raggiunti, il primo da FF e BF, il secondo da WF.

- La classe Any Fit e i bound precedenti si devono all'articolo di Johnson già citato nella Sezione 3.5.2: in particolare, il lower bound $\frac{17}{10}$ per ogni algoritmo Any Fit è dovuto a quell'articolo (D.S. Johnson, 1974).

3.6 Errori tipici

Attenzione

- **Credere che Next Fit sia un algoritmo Any Fit.** NF non riconsidera mai un contenitore chiuso, quindi può aprire un nuovo contenitore per un oggetto che entrerebbe in uno più vecchio (Esempio 3.3).
- **Scegliere il contenitore sbagliato in BF e WF.** BF pone l'oggetto nel contenitore con la capacità residua *minima* fra i contenitori in cui entra, WF in quello con la capacità residua *massima*; in caso di pareggio, entrambi prendono il contenitore di indice minimo.
- **Leggere la proprietà di riempimento a metà come «nessuna coppia di contenitori si può fondere».** Due contenitori di peso totale 6 e 1 con $c = 10$ soddisfano la proprietà e si possono fondere; la proprietà è ciò che dimostra la garanzia 2.
- **Trattare $r = 2$ come un valore raggiunto.** I rapporti nel caso peggiore di NF e WF sono estremi superiori: le famiglie di istanze si avvicinano a 2 al crescere di n, e nessuna istanza lo raggiunge.
- **Cambiare l'ordine degli oggetti.** I quattro algoritmi esaminano gli oggetti nell'ordine in cui sono dati; ordinarli prima è un algoritmo diverso.

3.7 In sintesi

In sintesi

- NF, FF, BF e WF collocano ogni oggetto appena arriva; NF mantiene un solo contenitore aperto, gli altri tre mantengono tutti i contenitori aperti finora e aprono un nuovo contenitore solo quando l'oggetto non entra da nessuna parte (classe Any Fit).
- In una soluzione Any Fit gli oggetti di due contenitori non entrano mai insieme, quindi tutti i contenitori tranne al più uno sono pieni per più della metà (proprietà di riempimento a metà), e $\mathscr{A}(I) \leq 2\,z(I)$.
- $r(NF) = r(WF) = 2$ e $r(FF) = r(BF) = \frac{17}{10}$: la sola regola di scelta separa il comportamento migliore e quello peggiore della classe Any Fit.

Quadro riassuntivo dell'istanza del BPP $c = 10$, $w = (6, 5, 4, 7, 3)$. I bound su $z(I)$ trovati finora:

Fonte	Lower bound	Upper bound	Da
Bound di conteggio	3	–	$\lceil \frac{25}{10} \rceil$
Questo capitolo, NF	–	3	$bin = (1, 2, 2, 3, 3)$
Questo capitolo, FF, BF, WF	–	3	3 contenitori ciascuno, con tre assegnamenti diversi
Valore ottimo	3	3	lower e upper bound coincidono

3.8 Implementazione in C++

- In questa sezione sull'implementazione in C++ usiamo indici a partire da 0 (oggetti e contenitori sono numerati a partire da 0). La logica algoritmica non cambia.
- Le esecuzioni passo per passo riportate in questa sezione si riferiscono tutte alla stessa istanza del BPP, cioè contenitori di capacità $c = 10$ e un insieme $\mathscr{I} = \{1, 2, 3, 4, 5\}$ di 5 oggetti ($n = 5$), con i seguenti pesi degli oggetti:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$
w_i	6	5	4	7	3

  Con gli indici a partire da 0 di questa sezione gli oggetti sono numerati $0, 1, 2, 3, 4$ e i loro pesi sono $w[0] = 6$, $w[1] = 5$, $w[2] = 4$, $w[3] = 7$ e $w[4] = 3$. Questa è l'istanza usata negli esempi delle sezioni precedenti, quindi le soluzioni trovate qui sono quelle già calcolate lì.
- Questa è un'implementazione in C++ di **Next Fit**:

```cpp
// items are processed in the input order
// the function returns the number of bins used
// bin[i] stores the bin assigned to item i
// it is assumed that n >= 1

int NextFit(int n, int c, int *w, int *bin)
{
    int nb = 1;   // current bin (number of open bins)
    int rc  = c;   // residual capacity of the current bin

    for (int i = 0; i < n; i++)
    {
        // if item i does not fit into the current bin, open a new bin
        if (w[i] > rc)
        {
            nb++;  // move to the next bin
            rc = c; // reset the residual capacity
        }

        bin[i] = nb - 1; // pack item i into the current bin
        rc -= w[i];        // update the residual capacity
    }

    return nb;
}
```

Questa è l'esecuzione passo per passo dell'implementazione C++ di NF su questa istanza.

Inizializzazione

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	-	-	-	-	-

$nb = 1$ $\quad rc = 10$

1. Iterazione 1 – l'oggetto 0 è posto nel contenitore 0

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	-	-	-	-

$nb = 1$ $\quad rc = 4$

2. Iterazione 2 – l'oggetto 1 è posto nel contenitore 1

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	-	-	-

$nb = 2$ $\quad rc = 5$

3. Iterazione 3 – l'oggetto 2 è posto nel contenitore 1

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	-	-

$nb = 2$ $\quad rc = 1$

4. Iterazione 4 – l'oggetto 3 è posto nel contenitore 2

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	2	-

$nb = 3$ $rc = 3$

5. Iterazione 5 – l'oggetto 4 è posto nel contenitore 2

$bin[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	2	2

$nb = 3$ $rc = 0$

- Questa è un'implementazione in C++ di **First Fit**:

```cpp
// items are processed in the input order
// the function returns the number of bins used
// bin[i] stores the bin assigned to item i
// it is assumed that n >= 1

int FirstFit(int n, int c, int *w, int *bin)
{
    int nb = 1;           // number of currently open bins
    int *rc = new int[n]; // residual capacities of bins (at most n bins)
    rc[0] = c;            // residual capacity of the first bin

    for (int i = 0; i < n; i++)
    {
        int s = -1;       // selected bin for item i (-1 means: not found yet)

        // scan bins in the order they were opened and pick the first that fits
        for (int b = 0; b < nb; b++)
        {
            if (w[i] <= rc[b])
            {
                s = b;  // selected bin
                break;
            }
        }

        // if no open bin can accommodate item i, open a new bin
        if (s == -1)
        {
            rc[nb] = c;   // initialize residual capacity of the new bin
            s = nb;      // select the new bin
            nb++;         // increase the number of open bins
        }

        bin[i] = s;      // pack item i into the selected bin
        rc[s] -= w[i];   // update the residual capacity
    }

    delete[] rc;
    return nb;
}
```

Questa è l'esecuzione passo per passo dell'implementazione C++ di FF su questa istanza.

Inizializzazione

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	-	-	-	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	10	-	-	-	-

$nb = 1$

1. Iterazione 1 – l'oggetto 0 è posto nel contenitore 0

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	-	-	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	-	-	-	-

$nb = 1$

2. Iterazione 2 – l'oggetto 1 è posto nel contenitore 1

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	-	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	5	-	-	-

$nb = 2$

3. Iterazione 3 – l'oggetto 2 è posto nel contenitore 0

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	5	-	-	-

$nb = 2$

4. Iterazione 4 – l'oggetto 3 è posto nel contenitore 2

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	2	-

rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	5	3	-	-

$$nb = 3$$

5. Iterazione 5 – l'oggetto 4 è posto nel contenitore 1

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	2	1

rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	2	3	-	-

$$nb = 3$$

- Questa è un'implementazione in `C++` di **Best Fit**:

```cpp
// items are processed in the input order
// the function returns the number of bins used
// bin[i] stores the bin assigned to item i
// it is assumed that n >= 1

int BestFit(int n, int c, int *w, int *bin)
{
    int nb = 1;           // number of currently open bins
    int *rc = new int[n]; // residual capacities of bins (at most n bins)
    rc[0] = c;            // residual capacity of the first bin

    for (int i = 0; i < n; i++)
    {
        int s = -1;       // selected bin for item i (-1 means: not found yet)
        int minrc = c + 1; // smallest residual capacity found so far

        // among the bins where item i fits, pick the one with the smallest residual capacity
        for (int b = 0; b < nb; b++)
        {
            if (w[i] <= rc[b] && rc[b] < minrc)
            {
                minrc = rc[b];
                s = b;  // selected bin
            }
        }

        // if no open bin can accommodate item i, open a new bin
        if (s == -1)
        {
            rc[nb] = c;   // initialize residual capacity of the new bin
            s = nb;      // select the new bin
            nb++;         // increase the number of open bins
```

```
33          }
34
35          bin[i] = s;      // pack item i into the selected bin
36          rc[s] -= w[i];    // update the residual capacity
37      }
38
39      delete[] rc;
40      return nb;
41  }
```

Questa è l'esecuzione passo per passo dell'implementazione C++ di BF su questa istanza.

Inizializzazione

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	-	-	-	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	10	-	-	-	-

$nb = 1$

1. Iterazione 1 – l'oggetto 0 è posto nel contenitore 0

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	-	-	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	-	-	-	-

$nb = 1$

2. Iterazione 2 – l'oggetto 1 è posto nel contenitore 1

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	-	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	5	-	-	-

$nb = 2$

3. Iterazione 3 – l'oggetto 2 è posto nel contenitore 0

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	5	-	-	-

$nb = 2$

4. Iterazione 4 – l'oggetto 3 è posto nel contenitore 2

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	2	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	5	3	-	-

$$nb = 3$$

5. Iterazione 5 – l'oggetto 4 è posto nel contenitore 2

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	0	2	2
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	0	5	0	-	-

$$nb = 3$$

- Questa è un'implementazione in C++ di **Worst Fit**:

```cpp
// items are processed in the input order
// the function returns the number of bins used
// bin[i] stores the bin assigned to item i
// it is assumed that n >= 1

int WorstFit(int n, int c, int *w, int *bin)
{
    int nb = 1;           // number of currently open bins
    int *rc = new int[n]; // residual capacities of bins (at most n bins)
    rc[0] = c;            // residual capacity of the first bin

    for (int i = 0; i < n; i++)
    {
        int s = -1;       // selected bin for item i (-1 means: not found yet)
        int maxrc = -1;   // largest residual capacity found so far

        // among the bins where item i fits, pick the one with the largest residual capacity
        for (int b = 0; b < nb; b++)
        {
            if (w[i] <= rc[b] && rc[b] > maxrc)
            {
                maxrc = rc[b];
                s = b;  // selected bin
            }
        }

        // if no open bin can accommodate item i, open a new bin
```

```
28          if (s == -1)
29          {
30              rc[nb] = c;   // initialize residual capacity of the new bin
31              s = nb;      // select the new bin
32              nb++;         // increase the number of open bins
33          }
34
35          bin[i] = s;      // pack item i into the selected bin
36          rc[s] -= w[i];    // update the residual capacity
37      }
38
39      delete[] rc;
40      return nb;
41  }
```

Questa è l'esecuzione passo per passo dell'implementazione C++ di WF su questa istanza.

Inizializzazione

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	-	-	-	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	10	-	-	-	-

$nb = 1$

1. Iterazione 1 – l'oggetto 0 è posto nel contenitore 0

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	-	-	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	-	-	-	-

$nb = 1$

2. Iterazione 2 – l'oggetto 1 è posto nel contenitore 1

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	-	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	5	-	-	-

$nb = 2$

3. Iterazione 3 – l'oggetto 2 è posto nel contenitore 1

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	-	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	1	-	-	-

$nb = 2$

4. Iterazione 4 – l'oggetto 3 è posto nel contenitore 2

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	2	-
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	4	1	3	-	-

$nb = 3$

5. Iterazione 5 – l'oggetto 4 è posto nel contenitore 0

bin[i]	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$
	0	1	1	2	0
rc[b]	$b=0$	$b=1$	$b=2$	$b=3$	$b=4$
	1	1	3	-	-

$nb = 3$

- Tutte le funzioni di questa sezione sono nella cartella `BPP_HEUR_CODE/src`. Il comando `make test` le compila ed esegue ogni esempio di questo capitolo e ogni istanza degli esercizi, confrontando il numero di contenitori e l'assegnamento oggetto-contenitore con quelli attesi.

Problemi di colorazione

Capitolo 4

Colorazione dei vertici: limiti dell'approssimazione

Prerequisiti

- Da questo modulo, il capitolo «Principi degli algoritmi di approssimazione»: rapporto di prestazione, algoritmo di α-approssimazione, garanzia di prestazione additiva.
- Dal modulo *Problemi di ottimizzazione combinatoria*, il capitolo «Problemi di colorazione»: la definizione del VCP, il numero cromatico $\chi(G)$, il bound di clique $\omega(G) \leq \chi(G)$ e il grafo di Petersen.
- Dal modulo *Ottimizzazione su reti*, il capitolo «Grafi»: la visita in ampiezza.
- Dal modulo *Problemi di ottimizzazione combinatoria*, il capitolo «Ottimizzazione combinatoria: definizioni»: tempo polinomiale, le classi P e NP, NP-completezza.

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- dimostrare che nessun algoritmo polinomiale ha un rapporto di prestazione inferiore a $\frac{4}{3}$ per il VCP, a meno che $P = NP$;
- enunciare il risultato di Zuckerman e spiegare perché il VCP non ha una garanzia costante;
- colorare all'ottimo i grafi bipartiti e i grafi di intervalli, e limitare i colori della colorazione greedy con $\Delta(G) + 1$;
- dimostrare che i grafi planari si possono colorare con 5 colori in tempo polinomiale, e ricavarne una garanzia additiva.

Il problema di questo capitolo è il problema della colorazione dei vertici (VCP): la sua definizione e le sue ipotesi sono nella sua pagina.

4.1 Notazione

- Come nella pagina del problema, $G = (V, E)$ è un grafo semplice con n vertici e m spigoli, $N(v)$ è l'insieme dei vicini di v, $\deg(v) = |N(v)|$ il suo grado, e $\Delta(G) = \max_{v \in V} \deg(v)$ il **grado massimo**. Il valore ottimo del VCP è il numero cromatico, $z(I) = \chi(G)$.
- **Richiamo.** Come nel capitolo «Principi degli algoritmi di approssimazione», $\mathscr{A}(I)$ è il numero di colori della colorazione restituita dall'algoritmo $\mathscr{A}$. Il VCP è un problema di minimo, quindi
$$\mathscr{A}(I) \geq \chi(G) \qquad \text{e} \qquad \frac{\mathscr{A}(I)}{\chi(G)} \geq 1$$
per ogni istanza I.
- Dare a ogni vertice un colore diverso è sempre ammissibile, quindi $\chi(G) \leq n$, e il rapporto n è banale. Questo capitolo mostra che un algoritmo polinomiale non può fare molto meglio su tutti i grafi, e poi le classi di grafi su cui invece può.

4.2 Il VCP non è approssimabile

4.2.1 Nessun rapporto inferiore a quattro terzi

- La dimostrazione segue lo schema di tutti i risultati negativi dell'approssimazione: un algoritmo con una garanzia troppo buona deciderebbe in tempo polinomiale un problema NP-completo.
- Il problema NP-completo è la **3-colorabilità**: dato un grafo G, decidere se $\chi(G) \leq 3$. Resta NP-completo sui grafi planari di grado massimo 4 (Garey, Johnson e Stockmeyer, 1976).

Teorema 4.1

Sia $\mathscr{A}$ un algoritmo polinomiale per il VCP. Se $P \neq NP$, esiste un grafo G tale che
$$\frac{\mathscr{A}(I)}{\chi(G)} \geq \frac{4}{3}.$$

Dimostrazione. Supponiamo che $\frac{\mathscr{A}(I)}{\chi(G)} < \frac{4}{3}$ per ogni grafo G, ed eseguiamo $\mathscr{A}$ su un grafo G.

1. Se $\chi(G) \leq 3$, allora $\mathscr{A}(I) < \frac{4}{3}\chi(G) \leq 4$. Il numero di colori è un intero, quindi $\mathscr{A}(I) \leq 3$.
2. Se $\chi(G) \geq 4$, allora $\mathscr{A}(I) \geq \chi(G) \geq 4$, perché $\mathscr{A}$ restituisce una colorazione ammissibile.

Quindi $\chi(G) \leq 3$ se e solo se $\mathscr{A}(I) \leq 3$. Eseguire $\mathscr{A}$ e contarne i colori richiede tempo polinomiale, quindi la 3-colorabilità sarebbe in P, e $P = NP$. □

- La dimostrazione usa solo l'interezza di $\chi(G)$: un rapporto inferiore a $\frac{4}{3}$ non può confondere 3 colori con 4. Con 2 colori lo stesso argomento non dice nulla, perché decidere se $\chi(G) \leq 2$ è facile (Sezione 4.3.1).

- I grafi planari di grado massimo 4 sono un caso particolare del VCP: il teorema vale anche su di essi.

4.2.2 Risultati molto più forti

- Il teorema esclude solo i rapporti inferiori a $\frac{4}{3}$. Si sa molto di più, e lo enunciamo senza dimostrazione.

Teorema 4.2: Zuckerman, 2007

Per ogni $\varepsilon > 0$, se $P \neq NP$, nessun algoritmo polinomiale $\mathscr{A}$ per il VCP soddisfa, per ogni grafo G con n vertici,

$$\frac{\mathscr{A}(I)}{\chi(G)} \leq n^{1-\varepsilon}.$$

- Il rapporto n è banale, quindi il numero cromatico è sostanzialmente difficile da approssimare quanto più è possibile. La dimostrazione usa le dimostrazioni verificabili in modo probabilistico (PCP) e gli estrattori.

- In precedenza, Garey e Johnson (1976) avevano alzato il bound $\frac{4}{3}$ a 2, e Feige e Kilian (1998) avevano dimostrato il bound $n^{1-\varepsilon}$ sotto un'ipotesi più forte di $P \neq NP$.

Corollario 4.3

Se $P \neq NP$, il VCP non ha alcun algoritmo di α-approssimazione, per nessuna costante $\alpha \geq 1$.

Dimostrazione. Sia $\mathscr{A}$ un algoritmo polinomiale di α-approssimazione, e prendiamo $\varepsilon = \frac{1}{2}$. I grafi con $n > \alpha^2$ vertici soddisfano $\frac{\mathscr{A}(I)}{\chi(G)} \leq \alpha < n^{1/2}$. I grafi con $n \leq \alpha^2$ vertici si risolvono esattamente per enumerazione, in un tempo limitato da una costante che dipende solo da α, con rapporto $1 \leq n^{1/2}$. L'algoritmo che si ottiene è polinomiale e ha rapporto al più $n^{1/2}$ su ogni grafo, contro il Teorema 4.2. □

Il VCP non è approssimabile: a meno che $P = NP$, nessun algoritmo polinomiale ha una garanzia di prestazione costante, e nemmeno una garanzia della forma $n^{1-\varepsilon}$. I risultati positivi esistono solo su classi particolari di grafi.

4.3 Casi particolari

- Sulle quattro classi di questa sezione il VCP si risolve esattamente, o con una garanzia, in tempo polinomiale. Le classi si riconoscono dal grafo, o sono date con esso (gli intervalli di un grafo di intervalli).

4.3.1 Grafi bipartiti

Definizione 4.4: grafo bipartito

Un grafo $G = (V, E)$ è **bipartito** se $\chi(G) \leq 2$, cioè se V si può dividere in due insiemi stabili.

Proposizione 4.5

Decidere se $\chi(G) \leq 2$, e trovare una colorazione con 2 colori quando esiste, richiede tempo $O(n + m)$.

Dimostrazione. Eseguiamo una visita in ampiezza da un vertice di ogni componente connessa, e sia $\ell(v)$ il livello di v, la sua distanza dalla radice della sua componente. Coloriamo v con il colore 1 se $\ell(v)$ è pari e con il colore 2 altrimenti.

1. Se nessuno spigolo unisce due vertici di livelli con la stessa parità, la colorazione è ammissibile.
2. Altrimenti, sia $\{u, v\}$ uno spigolo di questo tipo. I cammini dell'albero di visita dalla radice a u e a v, con lo spigolo $\{u, v\}$, formano un cammino chiuso di lunghezza dispari $\ell(u) + \ell(v) + 1$. Un cammino chiuso di lunghezza dispari contiene un ciclo di lunghezza dispari, che richiede 3 colori: $\chi(G) \geq 3$.

La visita e il controllo degli spigoli richiedono tempo $O(n + m)$. □

- La dimostrazione mostra la caratterizzazione classica: un grafo è bipartito se e solo se non ha cicli di lunghezza dispari (Kőnig, 1936).
- Su un grafo con almeno uno spigolo, la visita dà $\chi(G) = 2$, oppure il certificato $\chi(G) \geq 3$ fornito da un ciclo dispari.

4.3.2 Grafi di grado massimo limitato

- La **colorazione greedy** GC esamina i vertici in un ordine dato e assegna a ogni vertice il colore minimo, fra $1, 2, 3, \ldots$, non usato dai suoi vicini già colorati. Richiede tempo $O(n+m)$.

Proposizione 4.6

Per ogni grafo G e ogni ordine dei vertici,

$$GC(I) \;\leq\; \Delta(G)+1.$$

Dimostrazione. Quando il vertice v viene colorato, i suoi vicini già colorati usano al più $\deg(v) \leq \Delta(G)$ colori. Uno dei colori $1, 2, \ldots, \Delta(G)+1$ è quindi libero, e il colore libero minimo è al più $\Delta(G)+1$. □

- Il bound confronta $GC(I)$ con $\Delta(G)+1$, non con $\chi(G)$: su tutti i grafi non è una garanzia di prestazione, perché $\Delta(G)$ può valere $n-1$ mentre $\chi(G)=2$ (una stella).
- Sui grafi di grado massimo al più una costante *fissata* Δ, lo è. Ogni grafo con uno spigolo ha $\chi(G) \geq 2$, quindi

$$\frac{GC(I)}{\chi(G)} \;\leq\; \frac{\Delta+1}{2}.$$

- Il bound $\Delta(G)+1$ è raggiunto dai grafi completi e dai cicli di lunghezza dispari, qualunque sia l'ordine.
- **Osservazione (Brooks, 1941).** Un grafo connesso che non è né completo né un ciclo di lunghezza dispari soddisfa $\chi(G) \leq \Delta(G)$, e una colorazione di questo tipo si trova in tempo lineare.

4.3.3 Grafi planari

- Un grafo è **planare** se si può disegnare nel piano senza spigoli che si incrociano. Stabilire se un grafo è planare, e trovarne un disegno quando lo è, si fa in tempo lineare (Hopcroft e Tarjan, 1974).

Lemma 4.7

Ogni grafo planare con $n \geq 3$ vertici ha $m \leq 3n-6$ spigoli, e un vertice di grado al più 5.

Dimostrazione. Aggiungere spigoli a un disegno senza creare incroci non diminuisce m, quindi possiamo supporre che non si possa aggiungere alcuno spigolo di questo tipo: il disegno è connesso e ogni faccia è un triangolo. La formula di Eulero per un disegno planare connesso con f facce dà $n-m+f=2$. Ogni faccia ha 3 spigoli e ogni spigolo delimita 2 facce, quindi $3f=2m$, e

$$2 = n-m+f \;=\; n-m+\frac{2m}{3} \qquad\Longrightarrow\qquad m \leq 3n-6.$$

I gradi sommano a $2m \leq 6n - 12 < 6n$, quindi qualche vertice ha grado al più 5. □

Teorema 4.8: Heawood, 1890

Ogni grafo planare si può colorare con 5 colori, e una colorazione di questo tipo si trova in tempo polinomiale.

Dimostrazione. Per induzione su n; i grafi con $n \leq 5$ sono banali. Sia v un vertice di grado al più 5 (Lemma 4.7), e coloriamo $G - v$, che è planare, con 5 colori.

1. Se i vicini di v usano al più 4 colori, v prende il quinto.
2. Altrimenti v ha 5 vicini $v_1, \ldots, v_5$, in quest'ordine intorno a v nel disegno, con colori $1, \ldots, 5$. Sia H_{13} il grafo indotto dai vertici di colore 1 e 3.
3. Se v_1 e v_3 stanno in componenti diverse di H_{13}, scambiamo i colori 1 e 3 nella componente di v_1: la colorazione resta ammissibile, e il colore 1 diventa libero per v.
4. Altrimenti un cammino di colori 1 e 3 unisce v_1 a v_3; con v, chiude un ciclo che separa v_2 da v_4 nel disegno. Un cammino da v_2 a v_4 deve attraversare quel ciclo, quindi v_2 e v_4 stanno in componenti diverse di H_{24}: scambiando 2 e 4 nella componente di v_2 si libera il colore 2 per v.

Ogni passo usa una visita in tempo $O(n + m) = O(n)$, e i passi sono n: in totale tempo $O(n^2)$. □

- **Osservazione (il teorema dei quattro colori).** Ogni grafo planare si può colorare con 4 colori (Appel e Haken, 1977). La dimostrazione usa un calcolatore; Robertson, Sanders, Seymour e Thomas (1997) l'hanno semplificata e hanno dato un algoritmo che trova una colorazione di questo tipo in tempo $O(n^2)$.

- L'algoritmo PL per i grafi planari combina i risultati precedenti: se G non ha spigoli, usa 1 colore; se G è bipartito, usa 2 colori (Proposizione 4.5); altrimenti usa i 5 colori del Teorema 4.8.

Corollario 4.9

PL è un algoritmo polinomiale di approssimazione 2-additivo per il VCP sui grafi planari:

$$PL(I) - \chi(G) \leq 2.$$

Con l'algoritmo dei quattro colori al posto di quello dei cinque colori, l'errore additivo è al più 1.

Dimostrazione. Nei primi due casi PL è esatto. Nel terzo caso G ha un ciclo di lunghezza dispari, quindi $\chi(G) \geq 3$, mentre $PL(I) \leq 5$: l'errore additivo è al più $5 - 3 = 2$. Con 4 colori è al più $4 - 3 = 1$. □

- L'errore additivo 1 non si può ridurre a 0: decidere se un grafo planare ha $\chi(G) \leq 3$ è NP-completo. Per lo stesso motivo, il rapporto $\frac{4}{3}$ della versione con quattro colori è il migliore possibile sui grafi planari (Teorema 4.1).

4.3.4 Grafi di intervalli

Definizione 4.10: grafo di intervalli

Un grafo $G = (V, E)$ è un **grafo di intervalli** se a ogni vertice v è associato un intervallo chiuso $[a_v, b_v]$ della retta reale, e $\{u, v\} \in E$ se e solo se i due intervalli si intersecano.

- I grafi di intervalli descrivono richieste nel tempo che sono in conflitto quando si sovrappongono: lezioni che richiedono un'aula, prenotazioni che richiedono una risorsa. Un colore è un'aula.

Proposizione 4.11

Su un grafo di intervalli, la colorazione greedy con i vertici in ordine di estremo sinistro a_v non decrescente usa $\omega(G)$ colori. È quindi ottima: $GC(I) = \omega(G) = \chi(G)$.

Dimostrazione. Sia v un vertice che riceve il colore k. I colori $1, \ldots, k-1$ sono usati da vicini u di v colorati prima di v, quindi con $a_u \leq a_v$.

1. Poiché $[a_u, b_u]$ interseca $[a_v, b_v]$ e $a_u \leq a_v$, abbiamo $b_u \geq a_v$: il punto a_v sta nell'intervallo di ciascuno di questi vicini, e nell'intervallo di v.
2. I k intervalli che contengono a_v si intersecano a due a due, quindi i loro vertici formano una clique e $k \leq \omega(G)$.

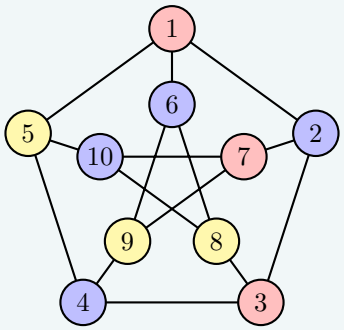
Quindi $GC(I) \leq \omega(G) \leq \chi(G) \leq GC(I)$, per il bound di clique. □

- Ordinare gli intervalli richiede tempo $O(n \log n)$, e la colorazione $O(n + m)$.
- L'ordine conta: con un altro ordine la colorazione greedy può usare più di $\omega(G)$ colori su un grafo di intervalli (Esempio 4.2).

4.4 Esempi

Esempio 4.1: I casi particolari sul grafo di Petersen

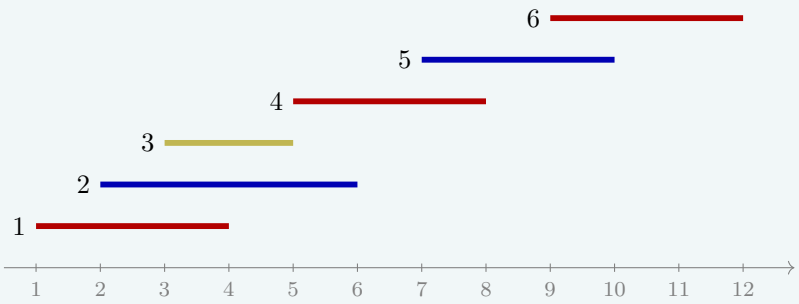
L'istanza di riferimento del VCP è il grafo di Petersen, con $n = 10$, $m = 15$ e $\Delta(G) = 3$.



- **È bipartito?** La visita dal vertice 1 dà il livello 0 al vertice 1, 1 ai vertici $2, 5, 6$, e 2 agli altri sei. Lo spigolo $\{3, 4\}$ unisce due vertici di livello 2: con i cammini $1, 2, 3$ e $1, 5, 4$ chiude il ciclo dispari $1, 2, 3, 4, 5$, quindi $\chi(G) \geq 3$.
- **Colorazione greedy.** Nell'ordine $1, 2, \ldots, 10$, GC assegna i colori $1, 2, 1, 2, 3, 2, 1, 3, 3, 2$ (rosso, blu, giallo nella figura): $GC(I) = 3 \leq \Delta(G) + 1 = 4$.
- Insieme, $3 \leq \chi(G) \leq GC(I) = 3$: la colorazione greedy è ottima su questa istanza.

Esempio 4.2: La colorazione greedy su un grafo di intervalli

Sei intervalli, con gli spigoli fra intervalli che si intersecano:



$$[a_v, b_v]: \ [1,4],\ [2,6],\ [3,5],\ [5,8],\ [7,10],\ [9,12],$$

$$E = \big\{\{1,2\},\{1,3\},\{2,3\},\{2,4\},\{3,4\},\{4,5\},\{5,6\}\big\}.$$

- Gli estremi sinistri sono già in ordine crescente. GC assegna i colori $1, 2, 3, 1, 2, 1$ (rosso, blu, giallo nella figura): $GC(I) = 3$.
- Gli intervalli $1, 2, 3$ contengono il punto 3, quindi $\{1, 2, 3\}$ è una clique e $\omega(G) = 3$: la colorazione è ottima, come garantisce la Proposizione 4.11.
- Con l'ordine $1, 2, 3, 5, 4, 6$ i colori sono $1, 2, 3$, poi 1 al vertice 5 e 4 al vertice 4, i cui vicini $2, 3, 5$ usano i colori $1, 2, 3$: 4 colori, uno in più di $\chi(G)$.

4.5 Fonti

- M.R. Garey, D.S. Johnson, L. Stockmeyer, Some simplified NP-complete graph problems, *Theoretical Computer Science* 1(3):237–267, 1976.
- M.R. Garey, D.S. Johnson, The complexity of near-optimal graph coloring, *Journal of the ACM* 23(1):43–49, 1976.
- U. Feige, J. Kilian, Zero knowledge and the chromatic number, *Journal of Computer and System Sciences* 57(2):187–199, 1998.
- D. Zuckerman, Linear degree extractors and the inapproximability of max clique and chromatic number, *Theory of Computing* 3:103–128, 2007.
- R.L. Brooks, On colouring the nodes of a network, *Proceedings of the Cambridge Philosophical Society* 37(2):194–197, 1941.
- P.J. Heawood, Map-colour theorem, *Quarterly Journal of Pure and Applied Mathematics* 24, 1890.
- J. Hopcroft, R. Tarjan, Efficient planarity testing, *Journal of the ACM* 21(4):549–568, 1974.
- K. Appel, W. Haken, Every planar map is four colorable, *Illinois Journal of Mathematics* 21, 1977.
- N. Robertson, D. Sanders, P. Seymour, R. Thomas, The four-colour theorem, *Journal of Combinatorial Theory, Series B* 70(1):2–44, 1997.

4.6 Errori tipici

Attenzione

- **Chiamare il bound $\Delta(G)+1$ una garanzia di prestazione.** Confronta la colorazione greedy con $\Delta(G)+1$, non con $\chi(G)$; diventa un rapporto costante solo sui grafi di grado massimo limitato.
- **Leggere il teorema del $\frac{4}{3}$ come un upper bound.** Afferma che i rapporti inferiori a $\frac{4}{3}$ sono impossibili, non che un rapporto di $\frac{4}{3}$ sia raggiungibile: per il risultato di Zuckerman, nessun rapporto costante lo è.
- **Usare il teorema dei quattro colori come algoritmo esatto.** Colora ogni grafo planare con al più 4 colori, ma decidere se bastano 3 colori resta NP-completo sui grafi planari.
- **Ordinare gli intervalli per estremo destro, o per lunghezza.** La Proposizione 4.11 usa l'ordine degli estremi sinistri: gli intervalli dei colori usati prima di v contengono tutti il punto a_v.
- **Dimenticare il ciclo dispari.** Una visita in ampiezza che trova un conflitto dimostra $\chi(G) \geq 3$ solo attraverso il ciclo dispari che chiude; il ciclo è il certificato.

4.7 In sintesi

In sintesi

- A meno che $P = NP$, nessun algoritmo polinomiale colora ogni grafo con meno di $\frac{4}{3}\chi(G)$ colori: deciderebbe la 3-colorabilità. Per il risultato di Zuckerman, non è possibile nemmeno il rapporto $n^{1-\varepsilon}$: il VCP non è approssimabile.
- I grafi bipartiti e i grafi di intervalli si colorano all'ottimo in tempo polinomiale; sui grafi di grado limitato la colorazione greedy ha rapporto $\frac{\Delta+1}{2}$; sui grafi planari l'errore additivo è al più 2, o 1 con il teorema dei quattro colori.

grafi	algoritmo	risultato
tutti i grafi	polinomiale	nessun rapporto costante, salvo $P = NP$
bipartiti	visita in ampiezza	esatto, $O(n+m)$
grado massimo $\leq \Delta$	colorazione greedy	al più $\Delta + 1$ colori
planari	PL	errore additivo ≤ 2 (o ≤ 1)
di intervalli	greedy per estremo sinistro	esatto, $\omega(G)$ colori

Quadro riassuntivo dell'istanza di riferimento.

Istanza	Lower bound	Valore euristico	Valore ottimo
grafo di Petersen	3, ciclo dispari	GC 3	3

Problemi di clique

Capitolo 5

Clique massima: limiti dell'approssimazione

Prerequisiti

- Da questo modulo, il capitolo «Principi degli algoritmi di approssimazione»: rapporto di prestazione per un problema di massimo, algoritmo di α-approssimazione, PTAS.
- Dal modulo *Problemi di ottimizzazione combinatoria*, il capitolo «Problemi di clique»: le definizioni del MCP e del MSSP, il numero di clique $\omega(G)$, il grafo complementare e $\omega(G) = \alpha(\overline{G})$.
- Da questo modulo, il capitolo «Colorazione dei vertici: limiti dell'approssimazione»: i grafi planari e il bound $m \leq 3n - 6$.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- costruire il prodotto forte di due grafi e dimostrare $\omega(G \boxtimes H) = \omega(G)\,\omega(H)$;
- dimostrare che un algoritmo a rapporto costante per il MCP ne darebbe uno per ogni rapporto costante;
- enunciare i risultati di Håstad e Zuckerman, e concludere che il MCP e il MSSP non sono approssimabili;
- risolvere il MCP all'ottimo sui grafi planari e sui grafi di grado limitato, e approssimare il MSSP sui grafi di grado limitato.

Il problema di questo capitolo è il problema della clique massima (MCP): la sua definizione e le sue ipotesi sono nella sua pagina.

5.1 Notazione

- Come nella pagina del problema, $G = (V, E)$ è un grafo semplice con n vertici e m spigoli, $N(v)$ è l'insieme dei vicini di v e $\Delta(G)$ il grado massimo. Il valore ottimo del MCP è il numero di clique, $z(I) = \omega(G)$, e quello del MSSP è il numero di stabilità $\alpha(G)$.
- **Richiamo.** Come nel capitolo «Principi degli algoritmi di approssimazione», $\mathscr{A}(I)$ è il numero di vertici della clique restituita dall'algoritmo $\mathscr{A}$. Il MCP è un problema di massimo, quindi
$$\mathscr{A}(I) \leq \omega(G) \qquad \text{e} \qquad \frac{\omega(G)}{\mathscr{A}(I)} \geq 1$$
per ogni istanza I.
- Un singolo vertice è una clique, quindi il rapporto n è banale. Questo capitolo mostra che un algoritmo polinomiale non può fare molto meglio su tutti i grafi, e poi le classi di grafi su cui può.

5.2 Il MCP non è approssimabile

5.2.1 Il prodotto forte

Definizione 5.1: prodotto forte

Il **prodotto forte** di due grafi $G = (V, E)$ e $H = (W, F)$ è il grafo $G \boxtimes H$ con insieme dei vertici $V \times W$, in cui due vertici distinti (u, x) e (v, y) sono adiacenti se e solo se
$$\big(u = v \ \text{ oppure } \ \{u, v\} \in E\big) \quad \text{e} \quad \big(x = y \ \text{ oppure } \ \{x, y\} \in F\big).$$

- $G \boxtimes H$ ha $|V|\,|W|$ vertici e si costruisce in tempo polinomiale nella dimensione di G e H. In particolare $G \boxtimes G$ ha n^2 vertici.
- Per un insieme $K \subseteq V \times W$, la **proiezione** è $p(K) = \{u \in V : (u, x) \in K \text{ per qualche } x\}$, e la **fibra** di $u \in p(K)$ è $K_u = \{x \in W : (u, x) \in K\}$.

Proposizione 5.2

Per ogni coppia di grafi G e H:

1. se $K \subseteq V \times W$ è una clique di $G \boxtimes H$, allora $p(K)$ è una clique di G, ogni fibra K_u è una clique di H, e $|K| \leq |p(K)| \cdot \max_u |K_u|$;
2. $\omega(G \boxtimes H) = \omega(G)\,\omega(H)$.

Dimostrazione. Parte 1. Siano $u \neq v$ in $p(K)$, con $(u, x), (v, y) \in K$. Questi due vertici sono adiacenti e $u \neq v$, quindi $\{u, v\} \in E$: $p(K)$ è una clique. Siano $x \neq y$ in K_u: (u, x) e (u, y) sono adiacenti, quindi $\{x, y\} \in F$, e K_u è una clique. Le fibre partizionano K, quindi
$$|K| \;=\; \sum_{u \in p(K)} |K_u| \;\leq\; |p(K)| \cdot \max_u |K_u|.$$

Parte 2. Se K_G e K_H sono clique massime di G e H, ogni coppia di vertici distinti di $K_G \times K_H$ soddisfa la definizione, quindi $\omega(G \boxtimes H) \geq \omega(G)\,\omega(H)$. Viceversa, per la Parte 1 ogni clique K di $G \boxtimes H$ ha $|K| \leq |p(K)| \cdot \max_u |K_u| \leq \omega(G)\,\omega(H)$. □

- Anche il prodotto *lessicografico* soddisfa $\omega(G[H]) = \omega(G)\,\omega(H)$; usiamo il prodotto forte, la cui definizione è simmetrica nei due grafi.

5.2.2 Un rapporto costante darebbe ogni rapporto costante

- La Parte 1 della Proposizione 5.2 è costruttiva: da una clique K di $G \boxtimes G$, $p(K)$ oppure la fibra più grande è una clique di G con almeno $\sqrt{|K|}$ vertici, perché il prodotto delle loro dimensioni è almeno $|K|$. Entrambe si calcolano in tempo polinomiale.

Teorema 5.3

Se il MCP ha un algoritmo polinomiale di α-approssimazione per qualche costante $\alpha \geq 1$, allora ha un algoritmo polinomiale di β-approssimazione per ogni costante $\beta > 1$.

Dimostrazione. Sia $\mathscr{A}$ un algoritmo polinomiale di α-approssimazione, e definiamo l'algoritmo $\mathscr{B}$:

1. costruire $G \boxtimes G$ ed eseguire $\mathscr{A}$ su di esso, ottenendo una clique K;
2. restituire la più grande fra $p(K)$ e la fibra più grande di K, una clique di G con almeno $\sqrt{|K|}$ vertici.

Per la Proposizione 5.2, $\omega(G \boxtimes G) = \omega(G)^2$, quindi $|K| \geq \frac{\omega(G)^2}{\alpha}$ e

$$\mathscr{B}(I) \;\geq\; \sqrt{|K|} \;\geq\; \frac{\omega(G)}{\sqrt{\alpha}}\,:$$

$\mathscr{B}$ è un algoritmo polinomiale di $\sqrt{\alpha}$-approssimazione. Applicando la costruzione t volte si ottiene il rapporto $\alpha^{1/2^t}$, con $\mathscr{A}$ eseguito su un grafo con n^{2^t} vertici. Scegliamo la costante t con $2^t \geq \frac{\ln \alpha}{\ln \beta}$: allora $\alpha^{1/2^t} \leq \beta$, e n^{2^t} è un polinomio. □

- In altre parole, il MCP ha un algoritmo a rapporto costante se e solo se ha un PTAS: il prodotto forte *amplifica* qualunque rapporto costante. Il tempo di esecuzione cresce come una potenza n^{2^t}, che è polinomiale solo perché t è una costante.

5.2.3 I risultati di Håstad e Zuckerman

- Il lato negativo è noto, e lo enunciamo senza dimostrazione: usa le dimostrazioni verificabili probabilisticamente.

Teorema 5.4: Håstad, 1999; Zuckerman, 2007

Per ogni $\varepsilon > 0$, se $P \neq NP$, nessun algoritmo polinomiale $\mathscr{A}$ per il MCP soddisfa, per ogni grafo G con n vertici,

$$\frac{\omega(G)}{\mathscr{A}(I)} \leq n^{1-\varepsilon}.$$

- Håstad (1999) ha dimostrato il teorema nell'ipotesi che i problemi NP non si possano risolvere in tempo polinomiale randomizzato ($NP \neq ZPP$); Zuckerman (2007) lo ha dimostrato nell'ipotesi $P \neq NP$.

- Come per il VCP, il teorema esclude ogni rapporto costante: un algoritmo di α-approssimazione ha rapporto al più $\alpha < n^{1/2}$ sui grafi con $n > \alpha^2$ vertici, e i grafi più piccoli si risolvono per enumerazione in tempo costante. Insieme al Teorema 5.3: a meno che $P = NP$, il MCP non ha né un rapporto costante né un PTAS.

Il MCP non è approssimabile: a meno che $P = NP$, nessun algoritmo polinomiale ha una garanzia di prestazione costante, e nemmeno una garanzia della forma $n^{1-\varepsilon}$. Esistono risultati positivi solo su classi particolari di grafi.

5.3 Casi particolari

5.3.1 Grafi planari

Proposizione 5.5

Ogni grafo planare ha $\omega(G) \leq 4$, e sui grafi planari il MCP si risolve all'ottimo in tempo $O(n^4)$.

Dimostrazione. Un sottografo di un grafo planare è planare. Il grafo completo con 5 vertici ha 10 spigoli, più di $3 \cdot 5 - 6 = 9$, quindi non è planare, e un grafo planare non ha clique con 5 vertici. I sottoinsiemi di al più 4 vertici sono $O(n^4)$, e ciascuno si verifica in tempo costante. □

5.3.2 Grafi di grado massimo limitato

Proposizione 5.6

Ogni grafo ha $\omega(G) \le \Delta(G)+1$, e il MCP si risolve all'ottimo in tempo $O(n\,2^{\Delta}\,\Delta^2)$, con $\Delta = \Delta(G)$.

Dimostrazione. Una clique che contiene il vertice v è contenuta in $\{v\} \cup N(v)$, che ha al più $\Delta+1$ vertici. Per ogni vertice v, enumeriamo i $2^{\deg(v)} \le 2^{\Delta}$ sottoinsiemi $S \subseteq N(v)$ e teniamo il più grande $\{v\} \cup S$ che sia una clique; verificare le coppie di S richiede tempo $O(\Delta^2)$. Ogni clique massima si trova a partire da uno qualsiasi dei suoi vertici. □

- Il tempo è polinomiale per ogni Δ fissato, e anche quando $\Delta \le \log_2 n$, perché allora $2^{\Delta} \le n$.

5.3.3 Il problema dell'insieme stabile massimo

- Per $\omega(G) = \alpha(\overline{G})$ (capitolo «Problemi di clique» del modulo *Problemi di ottimizzazione combinatoria*), un insieme stabile di G è una clique di $\overline{G}$, che si costruisce in tempo $O(n^2)$ e ha lo stesso n.
- Un algoritmo con rapporto $\rho(n)$ per uno dei due problemi, applicato al grafo complementare, ha lo stesso rapporto $\rho(n)$ per l'altro.

Corollario 5.7

Per ogni $\varepsilon > 0$, se $P \neq NP$, nessun algoritmo polinomiale per il MSSP ha rapporto $\frac{\alpha(G)}{\mathscr{A}(I)} \le n^{1-\varepsilon}$ su ogni grafo. In particolare, il MSSP non ha un rapporto costante.

- I casi particolari *non* si trasferiscono: il complementare di un grafo planare, o di un grafo di grado limitato, è denso. Su quelle classi il MSSP resta NP-difficile, ma è approssimabile.

Proposizione 5.8

Su ogni grafo, un insieme stabile **massimale** S (a cui non si può aggiungere nessun vertice) ha $|S| \ge \frac{n}{\Delta(G)+1}$. Quindi l'algoritmo greedy che aggiunge vertici finché è possibile soddisfa

$$\frac{\alpha(G)}{|S|} \;\le\; \Delta(G)+1.$$

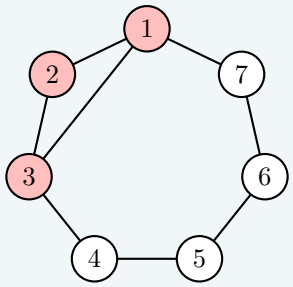
Dimostrazione. Ogni vertice fuori da S ha un vicino in S, altrimenti si potrebbe aggiungere. Quindi $V = \bigcup_{v \in S} \big(\{v\} \cup N(v)\big)$, e $n \le |S|\,(\Delta(G)+1)$. Poiché $\alpha(G) \le n$, il rapporto è al più $\frac{n}{|S|} \le \Delta(G)+1$. □

- Sui grafi di grado massimo al più un Δ fissato questo è un rapporto costante. Sui grafi planari il MSSP ha addirittura un PTAS (Baker, 1994).

5.4 Esempio

Esempio 5.1: Prodotto forte ed enumerazione sull'istanza di riferimento del MCP

L'istanza di riferimento del MCP è il ciclo $1, 2, \ldots, 7, 1$ con la corda $\{1, 3\}$: $n = 7$, $m = 8$, $\Delta(G) = 3$ e $\omega(G) = 3$, con l'unica clique massima $\{1, 2, 3\}$ (in rosso).



- **Prodotto forte.** $G \boxtimes G$ ha $7^2 = 49$ vertici e 240 spigoli, e $\omega(G \boxtimes G) = 3^2 = 9$, raggiunto da $\{1, 2, 3\} \times \{1, 2, 3\}$.
- Supponiamo che un algoritmo restituisca su $G \boxtimes G$ la clique $K = \{1, 2, 3\} \times \{4, 5\}$, con $|K| = 6$. La sua proiezione $p(K) = \{1, 2, 3\}$ e le sue fibre $K_u = \{4, 5\}$ sono clique di G; la più grande ha $3 \geq \sqrt{6}$ vertici.
- **Grado limitato.** L'enumerazione della Proposizione 5.6 esamina $\sum_v 2^{\deg(v)} = 2^3 + 2^2 + 2^3 + 4 \cdot 2^2 = 36$ sottoinsiemi, e trova $\{1, 2, 3\}$ dai vertici 1, 2 e 3.
- **Planarità.** Il grafo è planare, come mostra la figura, e $\omega(G) = 3 \leq 4$.

5.5 Fonti

- M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979 (l'auto-miglioramento del MCP con i prodotti di grafi).
- J. Håstad, Clique is hard to approximate within $n^{1-\varepsilon}$, *Acta Mathematica* 182(1):105–142, 1999.
- D. Zuckerman, Linear degree extractors and the inapproximability of max clique and chromatic number, *Theory of Computing* 3:103–128, 2007.
- B.S. Baker, Approximation algorithms for NP-complete problems on planar graphs, *Journal of the ACM* 41(1):153–180, 1994.

5.6 Errori tipici

Attenzione

- **Invertire il rapporto.** Il MCP è un problema di massimo: il rapporto è $\frac{\omega(G)}{\mathscr{A}(I)}$, almeno 1.
- **Iterare il prodotto un numero di volte che cresce con n.** Il grafo dopo t prodotti ha n^{2^t} vertici: è polinomiale solo per t costante, quindi l'amplificazione dà ogni rapporto costante, non il rapporto 1.
- **Trasferire i casi particolari al MSSP.** Il complementare di un grafo planare non è planare: il MCP è facile sui grafi planari, il MSSP lì è NP-difficile.
- **Confondere il bound $\Delta(G)+1$ del MCP con quello del MSSP.** Per il MCP, $\omega(G) \leq \Delta(G)+1$ limita l'ottimo e rende veloce l'enumerazione; per il MSSP, $\Delta(G)+1$ limita il rapporto di un insieme stabile massimale.

5.7 In sintesi

In sintesi

- Il prodotto forte eleva al quadrato il numero di clique, e una clique del prodotto restituisce una clique di G di dimensione almeno pari alla radice quadrata della sua: un rapporto costante per il MCP darebbe ogni rapporto costante.
- Per i risultati di Håstad e Zuckerman, a meno che $P = NP$ non è possibile nemmeno il rapporto $n^{1-\varepsilon}$: il MCP non è approssimabile, e nemmeno il MSSP, attraverso il grafo complementare.
- Sui grafi planari e sui grafi di grado limitato il MCP si risolve all'ottimo per enumerazione; sui grafi di grado limitato un insieme stabile massimale ha rapporto al più $\Delta(G)+1$ per il MSSP.

problema e grafi	algoritmo	risultato
MCP, tutti i grafi	polinomiale	nessun rapporto costante, salvo $P = NP$
MCP, planari	enumerazione	esatto, $\omega(G) \leq 4$
MCP, grado $\leq \Delta$	enumerazione	esatto, $O(n\,2^{\Delta}\Delta^2)$
MSSP, tutti i grafi	polinomiale	nessun rapporto costante, salvo $P = NP$
MSSP, grado $\leq \Delta$	stabile massimale	rapporto $\leq \Delta + 1$

Cruscotto dell'istanza di riferimento.

Istanza	Upper bound	Valore trovato	Valore ottimo
ciclo con una corda	$\Delta(G)+1 = 4$	3, enumerazione	3

Problemi di routing

Capitolo 6

TSP: il caso generale e il caso metrico

Prerequisiti

- Da questo modulo, il capitolo «Principi degli algoritmi di approssimazione»: rapporto di prestazione, garanzia di prestazione relativa, garanzia tight e rapporto di prestazione nel caso peggiore.
- Dal modulo *Problemi di ottimizzazione combinatoria*, il capitolo «Problemi di routing»: la definizione e l'istanza di riferimento del TSP.
- Dal modulo *Ottimizzazione su reti*, il capitolo «Alberi ricoprenti di costo minimo»: l'algoritmo di Kruskal.
- Dal modulo *Problemi di ottimizzazione combinatoria*, il capitolo «Ottimizzazione combinatoria: definizioni»: tempo polinomiale, le classi P e NP, NP-completezza.

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- dimostrare che per il TSP generale non esiste una garanzia costante, a meno che $P = NP$;
- spiegare perché è la disuguaglianza triangolare a rendere approssimabile il TSP;
- eseguire a mano l'algoritmo del doppio albero e l'algoritmo di Christofides su un'istanza metrica;
- dimostrare le garanzie 2 del doppio albero e $\frac{3}{2}$ dell'algoritmo di Christofides;
- costruire le famiglie di istanze che rendono tight queste garanzie.

Il problema di questo capitolo è il problema del commesso viaggiatore (TSP): la sua definizione e le sue ipotesi sono nella sua pagina.

6.1 Il TSP generale non è approssimabile

- Come nella pagina del problema, un'istanza ha un insieme $\mathscr{V} = \{1, 2, \ldots, n\}$ di città, $n \geq 3$, e distanze simmetriche $d_{ij} = d_{ji} \in \mathbb{Z}_{\geq 1}$; un giro σ è una permutazione di $\mathscr{V}$, di lunghezza $L(\sigma) = \sum_{k=1}^{n-1} d_{\sigma_k \sigma_{k+1}} + d_{\sigma_n \sigma_1}$, e la lunghezza va minimizzata.
- **Richiamo.** Come nel capitolo «Principi degli algoritmi di approssimazione», $z(I)$ è il valore ottimo dell'istanza I e $\mathscr{A}(I)$ il valore della soluzione restituita dall'algoritmo $\mathscr{A}$. Il TSP è un problema di minimo, quindi

$$\mathscr{A}(I) \geq z(I) \qquad \text{e} \qquad \frac{\mathscr{A}(I)}{z(I)} \geq 1$$

per ogni istanza I, e il rapporto di prestazione nel caso peggiore è $r(\mathscr{A}) = \sup_I \mathscr{A}(I)/z(I)$.
- Per il TSP generale nessun algoritmo polinomiale ha una garanzia costante, a meno che $P = NP$. La dimostrazione segue lo schema di tutti i risultati di questo tipo: un algoritmo di approssimazione con una garanzia troppo buona deciderebbe in tempo polinomiale un problema NP-completo.

Proposizione 6.1

Sia $\alpha \geq 1$ una costante. Se $P \neq NP$, non esiste un algoritmo polinomiale $\mathscr{A}$ per il TSP tale che, per ogni istanza I del TSP,

$$\frac{\mathscr{A}(I)}{z(I)} \leq \alpha.$$

Dimostrazione. Il **problema del ciclo hamiltoniano** chiede se un grafo $G = (V, E)$ ha un ciclo che passa per tutti i suoi vertici; è NP-completo. Supponiamo che $\mathscr{A}$ esista e, dato un grafo G con $n \geq 3$ vertici, costruiamo l'istanza I_G del TSP sulle città $\mathscr{V} = V$ con

$$d_{ij} = \begin{cases} 1 & \text{se } \{i, j\} \in E, \\ \lceil \alpha n \rceil + 1 & \text{altrimenti.} \end{cases}$$

Se G ha un ciclo hamiltoniano, quel ciclo è un giro di lunghezza n, quindi $z(I_G) = n$ e $\mathscr{A}(I_G) \leq \alpha n$. Se G non ha un ciclo hamiltoniano, ogni giro usa almeno una coppia $\{i, j\} \notin E$, quindi

$$\mathscr{A}(I_G) \;\geq\; z(I_G) \;\geq\; (n-1) + \lceil \alpha n \rceil + 1 \;>\; \alpha n.$$

Quindi G è hamiltoniano se e solo se $\mathscr{A}(I_G) \leq \alpha n$. Costruire I_G ed eseguire $\mathscr{A}$ richiede tempo polinomiale, quindi il problema del ciclo hamiltoniano sarebbe in P, e $P = NP$. □

- L'istanza I_G viola la **disuguaglianza triangolare** $d_{ij} \leq d_{ik} + d_{kj}$ non appena G ha un cammino i, k, j con $\{i, j\} \notin E$: $d_{ij} = \lceil \alpha n \rceil + 1 > 2 = d_{ik} + d_{kj}$. La dimostrazione ha bisogno di distanze così grandi, e non dice nulla sulle istanze che soddisfano la disuguaglianza.
- Il risultato e la sua dimostrazione sono dovuti a Sahni e Gonzalez (1976), e la NP-completezza del problema del ciclo hamiltoniano a Karp (1972).

Il TSP generale non è approssimabile: a meno che $P = NP$, nessun algoritmo polinomiale ha una garanzia di prestazione costante. Il TSP *metrico*, il caso particolare con la disuguaglianza triangolare, è approssimabile, e il resto del capitolo è dedicato a esso.

6.2 Un caso particolare: il TSP metrico

Definizione 6.2: TSP metrico

Un'istanza del TSP, con città $\mathscr{V} = \{1, 2, \ldots, n\}$ e distanze simmetriche $d_{ij} = d_{ji} \in \mathbb{Z}_{\geq 1}$, è **metrica** se le distanze soddisfano la disuguaglianza triangolare

$$d_{ij} \leq d_{ik} + d_{kj} \qquad \text{per ogni terna di città distinte } i, j, k \in \mathscr{V}. \tag{6.1}$$

- Il **TSP metrico** è il TSP ristretto alle istanze metriche. Le istanze I_G della Proposizione 6.1 non sono metriche: è proprio la disuguaglianza triangolare a rendere approssimabile il TSP, e ogni garanzia delle prossime sezioni la usa.
- L'istanza di riferimento ha 5 città, è metrica, e il suo unico giro ottimo è $(1, 3, 5, 2, 4)$, con $z(I) = 18$. Le sue distanze sono:

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	—

6.3 L'algoritmo del doppio albero e l'algoritmo di Christofides

- I due algoritmi di questa sezione costruiscono un giro a partire da un **albero ricoprente di costo minimo** (MST) del grafo completo su $\mathscr{V}$, con pesi degli spigoli d_{ij}. Lo calcoliamo con l'algoritmo di Kruskal: gli spigoli si esaminano per distanza non decrescente (pareggi per coppia di indici) e uno spigolo si prende se non chiude un ciclo con gli spigoli già presi. Il peso di un albero T è $d(T) = \sum_{\{i,j\} \in T} d_{ij}$.
- Entrambi gli algoritmi costruiscono prima un cammino chiuso che passa per ogni città, eventualmente più di una volta, e poi lo trasformano in un giro con le **scorciatoie** (short-cutting): si segue il cammino e si salta ogni città già visitata, cioè una città si mantiene solo alla sua prima occorrenza.
- Un cammino chiuso che usa ogni spigolo di un (multi)grafo esattamente una volta è un **circuito euleriano**. Negli esempi lo costruiamo con la **regola di Fleury**: si parte dalla città 1 e, a ogni passo, si lascia la città corrente lungo uno spigolo non usato verso il vicino di indice minimo, a meno che quello spigolo sia un *ponte* degli spigoli non usati (la sua rimozione li sconnette) e sia disponibile un altro spigolo.

6.3.1 L'algoritmo del doppio albero

Il *Double Tree Algorithm* (algoritmo del doppio albero, indicato con DT) è un'euristica costruttiva per il TSP metrico. Calcola un albero ricoprente di costo minimo T, raddoppia ciascuno dei suoi spigoli, segue un circuito euleriano dell'albero raddoppiato, e lo trasforma in un giro con le scorciatoie.

Algoritmo

Algorithm: Double Tree

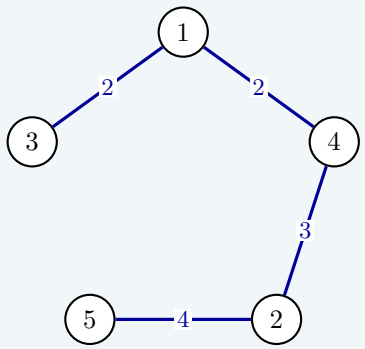
Input: istanza del TSP metrico con n città e distanze $d[1..n][1..n]$
Output: un giro σ e la sua lunghezza $L(\sigma)$

```
1 Function DoubleTree(n, d)
2     T := un albero ricoprente di costo minimo del grafo completo (Kruskal)
3     D := il multigrafo con due copie di ogni spigolo di T
4     W := un circuito euleriano di D dalla città 1 (Fleury)
5     σ := le città di W in ordine di prima occorrenza          // scorciatoie
6     return σ, L(σ)
```

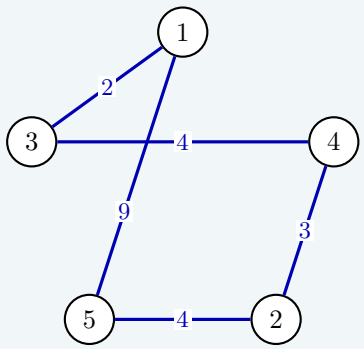
- Nell'albero raddoppiato, la regola di Fleury non torna mai al padre finché resta da visitare un figlio, perché l'ultima copia dello spigolo verso il padre è un ponte. Il circuito euleriano è quindi una visita in profondità di T dalla città 1, con i figli in ordine crescente, e il giro σ è l'ordine in cui la visita incontra le città per la prima volta (il *preordine*).
- Con l'algoritmo di Kruskal e una visita in profondità, DT richiede tempo $O(n^2 \log n)$.

Esempio 6.1: DT sull'istanza di riferimento del TSP

L'algoritmo di Kruskal esamina $\{1,3\}$ (2, preso), $\{1,4\}$ (2, preso), $\{2,4\}$ (3, preso), $\{2,3\}$ (4, chiude il ciclo $2,3,1,4$: scartato), $\{2,5\}$ (4, preso). L'albero è $T = \{\{1,3\},\{1,4\},\{2,4\},\{2,5\}\}$, con $d(T) = 2+2+3+4 = 11$, ed è l'unico MST dell'istanza.



MST T, $d(T) = 11$



giro di DT, $L(\sigma) = 22$

Il circuito euleriano dell'albero raddoppiato dalla città 1 è

$$W = (1,3,1,4,2,5,2,4,1), \qquad 2+2+2+3+4+4+3+2 = 22 = 2\,d(T).$$

Mantenendo ogni città alla sua prima occorrenza otteniamo $\sigma = (1, 3, 4, 2, 5)$, di lunghezza

$$L(\sigma) = d_{13} + d_{34} + d_{42} + d_{25} + d_{51} = 2 + 4 + 3 + 4 + 9 = 22.$$

Le scorciatoie sostituiscono $3 \to 1 \to 4$ $(2 + 2)$ con $3 \to 4$ (4) e $5 \to 2 \to 4 \to 1$ $(4 + 3 + 2)$ con $5 \to 1$ (9): su questa istanza entrambe le disuguaglianze triangolari usate valgono all'uguaglianza, e le scorciatoie non fanno risparmiare nulla. Quindi $DT(I) = 22$ e $\frac{DT(I)}{z(I)} = \frac{22}{18} = \frac{11}{9}$.

6.3.2 L'algoritmo di Christofides

Il *Christofides Algorithm* (algoritmo di Christofides, indicato con CH) è un'euristica costruttiva per il TSP metrico. Calcola un albero ricoprente di costo minimo T e, invece di raddoppiarlo, aggiunge a T un abbinamento perfetto di peso minimo M sulle città di grado dispari in T; poi segue un circuito euleriano di $T \cup M$ e lo trasforma in un giro con le scorciatoie.

Algoritmo

Algorithm: Christofides

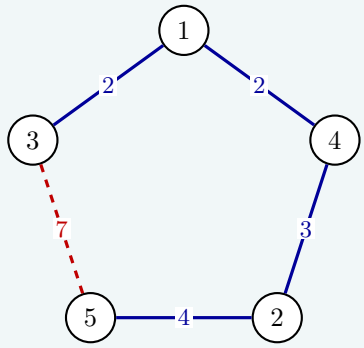
Input: istanza del TSP metrico con n città e distanze $d[1..n][1..n]$
Output: un giro σ e la sua lunghezza $L(\sigma)$

Function *Christofides*(n, d)
  $T :=$ un albero ricoprente di costo minimo del grafo completo (Kruskal)
  $O := \{ v \in \mathcal{V} : v$ ha grado dispari in $T \}$ `// |O| è pari`
  $M :=$ un abbinamento perfetto di peso minimo delle città in O
  $D := T \cup M$ `// multigrafo, tutti i gradi pari`
  $W :=$ un circuito euleriano di D dalla città 1 (Fleury)
  $\sigma :=$ le città di W in ordine di prima occorrenza `// scorciatoie`
  return σ, $L(\sigma)$

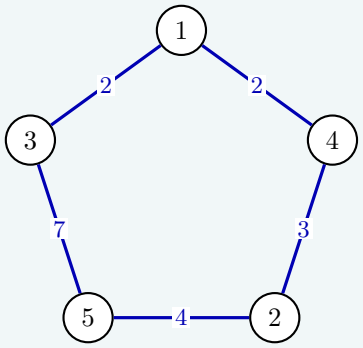
- **Perché $|O|$ è pari.** I gradi di un grafo sommano al doppio del numero di spigoli, un numero pari, quindi il numero di vertici di grado dispari è pari, e un abbinamento perfetto di O esiste (il grafo è completo).
- **Perché $T \cup M$ ha gradi pari.** Ogni città di O riceve da M esattamente uno spigolo in più, e le altre città nessuno. Se uno spigolo appartiene sia a T sia a M, D ne contiene due copie.
- Un abbinamento perfetto di peso minimo si può calcolare in tempo polinomiale, $O(n^3)$, con l'algoritmo blossom di Edmonds; negli esempi e negli esercizi $|O| \leq 4$ e gli abbinamenti si enumerano ($|O| = 4$ dà 3 abbinamenti perfetti).

Esempio 6.2: CH sull'istanza di riferimento del TSP

L'MST è quello dell'Esempio 6.1, con gradi $\deg(1) = 2$, $\deg(2) = 2$, $\deg(3) = 1$, $\deg(4) = 2$, $\deg(5) = 1$. Quindi $O = \{3, 5\}$ e l'unico abbinamento perfetto è $M = \{\{3, 5\}\}$, con $d(M) = 7$.



T (continuo) e M (tratteggiato)



giro di CH, $L(\sigma) = 18$

$T \cup M$ è un ciclo che passa per tutte le città. La regola di Fleury dalla città 1 (vicini 3 e 4) va in 3 e dà $W = (1, 3, 5, 2, 4, 1)$, di lunghezza $d(T) + d(M) = 11 + 7 = 18$, in cui nessuna città si ripete: non serve alcuna scorciatoia e $\sigma = (1, 3, 5, 2, 4)$, con $CH(I) = 18 = z(I)$. Sull'istanza di riferimento CH trova il giro ottimo, mentre DT lo supera di 4 unità.

6.4 Proprietà strutturali

- Le garanzie della prossima sezione confrontano la lunghezza di un giro euristico con un lower bound su $z(I)$. Servono tre proprietà: le scorciatoie non aumentano la lunghezza, un MST è più corto di ogni giro, e lo stesso vale per (il doppio di) un abbinamento perfetto minimo di un insieme pari di città.

Osservazione 6.3

In un'istanza del TSP metrico, sia W un cammino chiuso che visita ogni città almeno una volta, e sia σ il giro ottenuto da W con le scorciatoie. Allora $L(\sigma) \leq$ lunghezza di W.

Dimostrazione. Ogni spigolo $\{\sigma_k, \sigma_{k+1}\}$ del giro (e lo spigolo di chiusura $\{\sigma_n, \sigma_1\}$) sostituisce un tratto $\sigma_k = u_0, u_1, \ldots, u_q = \sigma_{k+1}$ di W le cui città interne vengono saltate. Applicando la disuguaglianza triangolare (6.1) $q - 1$ volte,

$$d_{u_0 u_q} \;\leq\; d_{u_0 u_1} + d_{u_1 u_q} \;\leq\; \ldots \;\leq\; \sum_{p=0}^{q-1} d_{u_p u_{p+1}}.$$

I tratti sono parti disgiunte di W, quindi sommando su tutti gli spigoli di σ si ottiene $L(\sigma) \leq$ lunghezza di W. □

Osservazione 6.4

Per ogni istanza del TSP (metrica o no) e ogni albero ricoprente di costo minimo T,

$$d(T) \leq z(I).$$

Dimostrazione. Togliendo uno spigolo qualsiasi da un giro ottimo resta un cammino che passa per tutte le città, che è un albero ricoprente. Il suo peso è $z(I)$ meno la distanza tolta, al più $z(I)$ poiché le distanze sono positive, e l'MST non pesa più di alcun albero ricoprente. □

Osservazione 6.5

In un'istanza del TSP metrico, sia $O \subseteq \mathcal{V}$ con $|O|$ pari e $|O| \geq 2$, e sia M un abbinamento perfetto di peso minimo di O. Allora

$$d(M) \leq \frac{z(I)}{2}.$$

Dimostrazione. Sia $\tilde{\sigma}$ un giro ottimo e applichiamo le scorciatoie restringendolo alle città di O: otteniamo un ciclo $o_1, o_2, \ldots, o_{|O|}, o_1$ sulle città di O che, per l'Osservazione 6.3, ha lunghezza al più $z(I)$. Poiché $|O|$ è pari, i suoi spigoli si dividono in due abbinamenti perfetti di O, prendendo alternativamente uno spigolo ciascuno:

$$M_1 = \{\{o_1, o_2\}, \{o_3, o_4\}, \ldots\}, \qquad M_2 = \{\{o_2, o_3\}, \{o_4, o_5\}, \ldots, \{o_{|O|}, o_1\}\}.$$

Quindi $d(M_1) + d(M_2) \leq z(I)$, uno dei due pesa al più $\frac{z(I)}{2}$, e $d(M) \leq \min\{d(M_1), d(M_2)\} \leq \frac{z(I)}{2}$. (Per $|O| = 2$ il ciclo è lo spigolo $\{o_1, o_2\}$ percorso due volte e $M_1 = M_2$.) □

- L'ultima proprietà che serve è classica: un multigrafo connesso in cui ogni vertice ha grado pari ha un circuito euleriano (un risultato classico di Eulero), e la regola di Fleury ne trova uno. L'albero raddoppiato di DT e il multigrafo $T \cup M$ di CH sono connessi (contengono T) e hanno gradi pari.
- Sull'istanza di riferimento: $d(T) = 11 \leq 18$, e con $O = \{3, 5\}$ il giro ottimo ristretto a O è $3, 5, 3$, di lunghezza $14 \leq 18$, quindi $d(M) = 7 \leq 9$.

6.5 Garanzie di prestazione

6.5.1 Il rapporto di prestazione nel caso peggiore di DT

Proposizione 6.6

Per ogni istanza I del TSP metrico,

$$\frac{DT(I)}{z(I)} \leq 2.$$

Dimostrazione. Il circuito euleriano W dell'albero raddoppiato usa ogni spigolo di T due volte, quindi la sua lunghezza è $2\,d(T)$, e visita ogni città perché T è ricoprente. Per l'Osservazione 6.3 e l'Osservazione 6.4,

$$DT(I) = L(\sigma) \leq 2\,d(T) \leq 2\,z(I).$$

□

Proposizione 6.7

Esistono istanze I_n del TSP metrico, con $n \geq 5$ dispari, tali che

$$\frac{DT(I_n)}{z(I_n)} = \frac{2n}{n+2} \to 2 \qquad \text{per} \qquad n \to +\infty.$$

Dimostrazione. Siano $n = 2k+1$ con $k \geq 2$, $a := n$ e $b := n+1$. La città 1 è un centro e le città $2, \ldots, n$ sono foglie; le foglie sono unite dal cammino

$$P = (3, 5, \ldots, 2k+1,\ 2, 4, \ldots, 2k),$$

in cui città consecutive non hanno mai indici consecutivi (le differenze valgono 2, e $2k-1 \geq 3$ per lo spigolo $\{2k+1, 2\}$). Le distanze sono

$$d_{ij} = \begin{cases} a & \text{se } i = 1 \text{ o } j = 1, \\ b & \text{se } \{i,j\} \text{ è uno spigolo di } P, \\ 2a & \text{altrimenti.} \end{cases}$$

Tutte le distanze sono comprese fra a e $2a$, quindi $d_{ij} \leq 2a \leq d_{ik} + d_{kj}$: l'istanza è metrica.

DT. Le sole distanze pari ad a sono quelle dal centro, e un albero ricoprente ha $n-1$ spigoli, ciascuno di peso almeno a: la stella di centro 1 è l'unico MST, con $d(T) = (n-1)a$. La visita in profondità dalla città 1 incontra le foglie in ordine crescente, $\sigma = (1, 2, 3, \ldots, n)$, e nessuna coppia di foglie consecutive è uno spigolo di P:

$$DT(I_n) = a + (n-2)\,2a + a = 2(n-1)a = 2n(n-1).$$

Ottimo. Un giro ha esattamente due spigoli nella città 1 (di lunghezza a ciascuno) e $n-2$ spigoli fra foglie (di lunghezza almeno b ciascuno), quindi $z(I_n) \geq 2a + (n-2)b$. Il giro

$(1, 3, 5, \ldots, 2k+1, 2, 4, \ldots, 2k)$ segue P e raggiunge questo valore, quindi

$$z(I_n) = 2a + (n-2)b = 2n + (n-2)(n+1) = (n+2)(n-1).$$

Pertanto

$$\frac{DT(I_n)}{z(I_n)} = \frac{2n(n-1)}{(n+2)(n-1)} = \frac{2n}{n+2} = \frac{2}{1+\frac{2}{n}} \to 2.$$

□

Dalla Proposizione 6.6 e dalla Proposizione 6.7 segue che DT è un algoritmo di 2-approssimazione per il TSP metrico, e che il suo rapporto di prestazione nel caso peggiore è

$$r(DT) = 2$$

- Per $n = 5$: $a = 5$, $b = 6$, $P = (3, 5, 2, 4)$, $DT(I_5) = 40$, $z(I_5) = 28$ e il rapporto è $\frac{10}{7}$; per $n = 7$ e $n = 9$ i rapporti sono $\frac{14}{9}$ e $\frac{18}{11}$.
- Sull'istanza di riferimento $DT(I) = 22 = 2\,d(T)$: l'upper bound della dimostrazione è raggiunto, ma $z(I) = 18 > d(T) = 11$, e il rapporto è solo $\frac{11}{9}$.
- L'analisi di DT è dovuta a Rosenkrantz, Stearns e Lewis (1977).

6.5.2 Il rapporto di prestazione nel caso peggiore di CH

Proposizione 6.8

Per ogni istanza I del TSP metrico,

$$\frac{CH(I)}{z(I)} \leq \frac{3}{2}.$$

Dimostrazione. Il circuito euleriano W di $T \cup M$ usa ogni spigolo una volta, quindi la sua lunghezza è $d(T) + d(M)$, e visita ogni città. Per l'Osservazione 6.3, l'Osservazione 6.4 e l'Osservazione 6.5 (con l'insieme O delle città di grado dispari, $|O| \geq 2$ perché un albero con almeno due vertici ha almeno due foglie),

$$CH(I) = L(\sigma) \;\leq\; d(T) + d(M) \;\leq\; z(I) + \frac{z(I)}{2} \;=\; \frac{3}{2}\, z(I).$$

□

Proposizione 6.9

Esistono istanze I_n del TSP metrico, con $n \geq 5$ dispari, tali che

$$\frac{CH(I_n)}{z(I_n)} \to \frac{3}{2} \qquad \text{per} \qquad n \to +\infty.$$

Dimostrazione. Siano $n \geq 5$ dispari, $a := n$ e $b := n + 1$, così che $a < b < 2a$. Consideriamo il grafo H sulle città $1, \ldots, n$ con gli spigoli $\{i, i+1\}$ di lunghezza a e $\{i, i+2\}$ di lunghezza b (una striscia di triangoli), e sia d_{ij} la lunghezza di un cammino minimo da i a j in H; le distanze di cammino minimo soddisfano sempre la disuguaglianza triangolare. Un cammino da i a $j > i$ avanza di 1 (costo a) o di 2 (costo b) a ogni spigolo, e poiché $b < 2a$ conviene avanzare di 2 il più possibile:

$$d_{ij} = \left\lfloor \frac{|i-j|}{2} \right\rfloor b + \big(|i-j| \bmod 2\big)\, a.$$

CH. Le distanze pari ad a sono esattamente quelle fra città consecutive, quindi, come nella dimostrazione precedente, il cammino $T = \{\{1,2\},\{2,3\},\ldots,\{n-1,n\}\}$ è l'unico MST, con $d(T) = (n-1)a$.

Le sue città di grado dispari sono $O = \{1, n\}$, quindi $M = \{\{1, n\}\}$, con $d(M) = \frac{n-1}{2} b$. $T \cup M$ è il ciclo $1, 2, \ldots, n, 1$, la regola di Fleury lo segue dalla città 1 verso la città 2, e non serve alcuna scorciatoia:

$$CH(I_n) = (n-1)a + \frac{n-1}{2}\, b = (n-1)\,\frac{3n+1}{2}.$$

Un upper bound sull'ottimo. Il giro $(1, 3, 5, \ldots, n, n-1, n-3, \ldots, 2)$ usa $n-2$ spigoli di lunghezza b e i due spigoli $\{n, n-1\}$ e $\{2, 1\}$ di lunghezza a, quindi

$$z(I_n) \leq (n-2)b + 2a = (n-2)(n+1) + 2n = (n+2)(n-1).$$

Insieme alla Proposizione 6.8,

$$\frac{3}{2} \geq \frac{CH(I_n)}{z(I_n)} \geq \frac{(n-1)(3n+1)}{2(n+2)(n-1)} = \frac{3n+1}{2n+4} = \frac{3+\frac{1}{n}}{2+\frac{4}{n}} \to \frac{3}{2}.$$

□

Dalla Proposizione 6.8 e dalla Proposizione 6.9 segue che CH è un algoritmo di $\frac{3}{2}$-approssimazione per il TSP metrico, e che il suo rapporto di prestazione nel caso peggiore è

$$r(CH) = \frac{3}{2}$$

- Per $n = 5, 7, 9$ l'enumerazione di tutti i giri mostra che l'upper bound su $z(I_n)$ è l'ottimo (28, 54, 88), e i rapporti sono $\frac{32}{28} = \frac{8}{7}$, $\frac{66}{54} = \frac{11}{9}$, $\frac{112}{88} = \frac{14}{11}$.
- L'algoritmo è stato trovato indipendentemente da Christofides (1976) e da Serdyukov (1978). Per più di quarant'anni non si è conosciuto per il TSP metrico alcun algoritmo polinomiale con una costante migliore; il primo miglioramento, di una costante piccolissima, è un algoritmo randomizzato di Karlin, Klein e Oveis Gharan (2021).

6.6 Fonti

- D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis II, An analysis of several heuristics for the traveling salesman problem, *SIAM Journal on Computing* 6(3):563–581, 1977.
- N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388, Carnegie-Mellon University, 1976.
- A.I. Serdyukov, On some extremal walks in graphs, *Upravlyaemye Sistemy* 17:76–79, 1978.
- S. Sahni, T. Gonzalez, P-complete approximation problems, *Journal of the ACM* 23(3):555–565, 1976.
- R.M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations*, Plenum Press, 85–103, 1972.

6.7 Errori tipici

Attenzione

- **Applicare DT o CH a un'istanza non metrica.** Le garanzie 2 e $\frac{3}{2}$ usano la disuguaglianza triangolare, nel passo delle scorciatoie; senza di essa non esiste alcuna garanzia costante.
- **Saltare una città visitata per la prima volta.** Seguendo il circuito euleriano, una città si salta solo se è già stata visitata; la prima occorrenza si mantiene.
- **Abbinare le città sbagliate in CH.** L'abbinamento perfetto si calcola sulle città di grado *dispari* nell'albero ricoprente, il cui numero è pari, e ha peso minimo.

6.8 In sintesi

In sintesi

- Il TSP generale non è approssimabile: un algoritmo con una garanzia costante deciderebbe il problema del ciclo hamiltoniano in tempo polinomiale.
- Il TSP metrico è il caso particolare approssimabile. Le garanzie di DT e CH si dimostrano con lower bound su $z(I)$: l'albero ricoprente di costo minimo e metà dell'abbinamento perfetto minimo; le scorciatoie, che usano la disuguaglianza triangolare, trasformano il circuito euleriano in un giro non più lungo.
- Entrambi i rapporti sono tight, con famiglie di istanze costruite in questo capitolo.

problema	algoritmo	rapporto nel caso peggiore
TSP generale	qualsiasi polinomiale	nessuna costante, a meno che $P = NP$
TSP metrico	DT	2 (tight)
TSP metrico	CH	$\frac{3}{2}$ (tight)

Quadro riassuntivo dell'istanza di riferimento. I bound su $z(I)$ trovati finora:

Istanza	Lower bound	Valori euristici	Valore ottimo
TSP metrico, 5 città	11, albero ricoprente di costo minimo	DT 22, CH 18	18

Problemi di scheduling

Capitolo 7

Approssimazione per $P||C_{\max}$

Prerequisiti

- Da questo modulo, il capitolo «Principi degli algoritmi di approssimazione»: rapporto di prestazione, garanzia di prestazione relativa, garanzia tight e rapporto di prestazione nel caso peggiore.
- Dal modulo *Problemi di ottimizzazione combinatoria*, il capitolo «Problemi di scheduling»: la definizione e l'istanza di riferimento di $P\,||\,C_{\max}$, e il suo lower bound elementare.
- Dal modulo *Algoritmi euristici*, il capitolo «Euristiche costruttive»: gli algoritmi $LIST$ e LPT come regole costruttive.

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- eseguire a mano $LIST$ e LPT e disegnare lo schedule che ne risulta;
- dimostrare le garanzie $2 - \frac{1}{m}$ di $LIST$ e $\frac{4}{3} - \frac{1}{3m}$ di LPT;
- costruire le famiglie di istanze che rendono tight queste garanzie;
- spiegare perché ordinare i lavori per tempo di esecuzione non crescente migliora la garanzia.

Il problema di questo capitolo è il problema di scheduling su macchine parallele identiche ($P\,||\,C_{\max}$): la sua definizione e le sue ipotesi sono nella sua pagina.

7.1 Notazione

- Come nella pagina del problema, ci sono m macchine identiche e un insieme $\mathscr{J} = \{1, 2, \ldots, n\}$ di lavori con tempi di esecuzione t_j. Una soluzione assegna ogni lavoro $j \in \mathscr{J}$ a una macchina $mach_j = h$; l'insieme dei lavori della macchina h è J_h, il suo carico è $L_h = \sum_{j \in J_h} t_j$, e il makespan è $C_{\max} = \max_h L_h$.

- **Richiamo.** Come nel capitolo «Principi degli algoritmi di approssimazione», $z(I)$ è il valore ottimo dell'istanza I e $\mathscr{A}(I)$ il valore della soluzione restituita dall'algoritmo $\mathscr{A}$. $P\,||\,C_{\max}$ è un problema di minimo, quindi

$$\mathscr{A}(I) \geq z(I) \qquad \text{e} \qquad \frac{\mathscr{A}(I)}{z(I)} \geq 1$$

  per ogni istanza I, e il rapporto di prestazione nel caso peggiore è $r(\mathscr{A}) = \sup_I \mathscr{A}(I)/z(I)$.

- Usiamo il lower bound elementare del capitolo «Problemi di scheduling» del modulo *Problemi di ottimizzazione combinatoria*:

$$z(I) \;\geq\; \max\left\{ \max_{j \in \mathscr{J}} t_j,\;\; \frac{\sum_{j \in \mathscr{J}} t_j}{m} \right\}. \tag{7.1}$$

- L'istanza di riferimento ha $m = 3$ e $t = (5, 5, 4, 4, 3, 3, 3)$; il bound dà $\max\{5, 27/3\} = 9$, e $z(I) = 9$ con $J_1 = \{1, 3\}$, $J_2 = \{2, 4\}$, $J_3 = \{5, 6, 7\}$.

7.2 Gli algoritmi *LIST* e *LPT*

7.2.1 List scheduling

Il *List Scheduling Algorithm* (indicato con $LIST$) è un'euristica costruttiva per $P\,||\,C_{\max}$. Esamina i lavori nell'ordine di una lista λ data e assegna ogni lavoro a una macchina di carico corrente minimo (in caso di pareggio, alla macchina di indice minimo).

Algoritmo

Algorithm: List Scheduling

Input: istanza di $P\,||\,C_{\max}$ con n lavori, tempi $t[1..n]$, m macchine; lista $\lambda[1..n]$
Output: assegnamento lavori-macchine $mach[1..n]$ e makespan $C_{\max}$

```
1  Function ListScheduling(n, m, t, λ)
2      mach := nuovo vettore[1..n]                    // assegnamento lavori-macchine
3      for h := 1 to m do
4          L[h] ← 0                                   // carichi delle macchine
5      end
6      for k := 1 to n do
7          j := λ[k]                                  // k-esimo lavoro della lista
8          s := una macchina con L[s] minimo (indice minimo)   // macchina meno
               carica
9          mach[j] ← s                                // assegnare il lavoro j alla macchina s
10         L[s] ← L[s] + t[j]
11     end
12     return mach, C_max = max_h L[h]
```

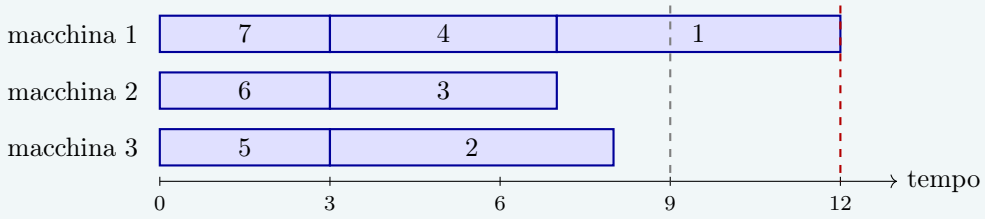
- *LIST* è lo stesso algoritmo del capitolo «Euristiche costruttive» del modulo *Algoritmi euristici*, dove è introdotto come regola greedy. Qui ci interessa il suo caso peggiore: quanto può essere cattiva una lista?
- Ogni macchina esegue i suoi lavori senza tempi morti, quindi il lavoro j assegnato all'iterazione k **inizia** all'istante S_j, pari al carico della sua macchina subito prima dell'assegnamento, e termina in $S_j + t_j$.

Esempio 7.1: *LIST* sull'istanza di riferimento, prima i lavori più brevi

Consideriamo la lista $\lambda = (7, 6, 5, 4, 3, 2, 1)$. I carichi prima di ogni assegnamento e la macchina scelta sono:

lavoro j	t_j	carichi (L_1, L_2, L_3) prima	macchina
7	3	$(0, 0, 0)$	1
6	3	$(3, 0, 0)$	2
5	3	$(3, 3, 0)$	3
4	4	$(3, 3, 3)$	1
3	4	$(7, 3, 3)$	2
2	5	$(7, 7, 3)$	3
1	5	$(7, 7, 8)$	1

I carichi finali sono $(12, 7, 8)$, con $J_1 = \{7, 4, 1\}$, $J_2 = \{6, 3\}$, $J_3 = \{5, 2\}$, e $LIST(I) = 12$:



L'ultimo lavoro, il lavoro 1, inizia in $S_1 = 7$, il carico minimo in quel momento: tutte le macchine sono occupate almeno fino all'istante 7. È l'osservazione su cui si costruisce la garanzia di $LIST$.

7.2.2 Longest processing time first

Il *Longest Processing Time first Algorithm* (indicato con LPT) è un'euristica costruttiva per $P\,||\,C_{\max}$. Ordina i lavori per tempo di esecuzione non crescente e applica $LIST$ a questa lista: ogni lavoro, dal più lungo al più breve, viene assegnato a una macchina meno carica.

Algoritmo

Algorithm: Longest Processing Time first

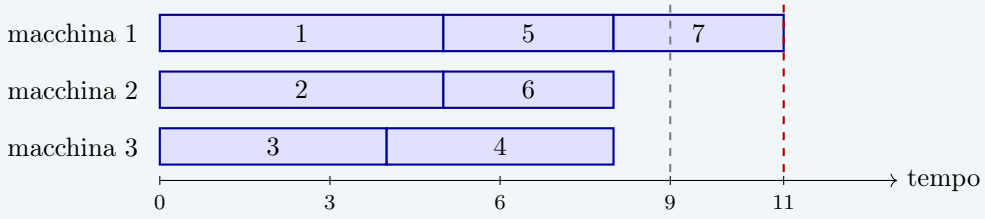
Input: istanza di $P\,||\,C_{\max}$ con n lavori, tempi $t[1..n]$, m macchine
Output: assegnamento lavori-macchine $mach[1..n]$ e makespan $C_{\max}$

```
1 Function LPT(n, m, t)
2 |  λ := i lavori ordinati per t[j] non crescente (pareggi per indice)   // prima i
  |   lavori più lunghi
3 |  return ListScheduling(n, m, t, λ)
```

- Nelle dimostrazioni conviene rinumerare i lavori nell'ordine di λ, così che $t_1 \geq t_2 \geq \cdots \geq t_n$ e LPT assegna i lavori nell'ordine $1, 2, \ldots, n$.

Esempio 7.2: *LPT* sull'istanza di riferimento

I lavori sono già ordinati, $\lambda = (1, 2, 3, 4, 5, 6, 7)$. I lavori 1, 2, 3 vanno sulle tre macchine vuote, con carichi $(5, 5, 4)$; il lavoro 4 va sulla macchina 3, la meno carica: $(5, 5, 8)$; il lavoro 5 va sulla macchina 1: $(8, 5, 8)$; il lavoro 6 sulla macchina 2: $(8, 8, 8)$; il lavoro 7 trova tre macchine di carico 8 e va sulla macchina 1: $(11, 8, 8)$. Quindi $J_1 = \{1, 5, 7\}$, $J_2 = \{2, 6\}$, $J_3 = \{3, 4\}$ e $LPT(I) = 11$:



Con $z(I) = 9$, il rapporto è $\frac{11}{9}$: l'ordinamento migliora il makespan da 12 a 11, ma l'ultimo

lavoro, per quanto breve, sbilancia tre macchine perfettamente bilanciate.

7.3 Una proprietà del list scheduling

- Le garanzie della prossima sezione confrontano il valore di una soluzione euristica con il lower bound (7.1) su $z(I)$. Oltre al bound, usano una proprietà degli schedule costruiti da $LIST$: il lavoro che termina per ultimo non può iniziare troppo tardi.

Osservazione 7.1

Sia ℓ un lavoro che termina per ultimo in uno schedule costruito da $LIST$ (con una lista qualsiasi). Allora il suo istante di inizio soddisfa

$$S_\ell \le \frac{1}{m} \sum_{j \in \mathscr{J} \setminus \{\ell\}} t_j.$$

Dimostrazione. Quando ℓ viene assegnato, S_ℓ è il minimo degli m carichi, quindi non supera la loro media. I carichi in quel momento contengono solo lavori assegnati prima di ℓ, quindi la loro somma è al più $\sum_{j \neq \ell} t_j$, e $S_\ell \le \frac{1}{m} \sum_{j \neq \ell} t_j$. □

7.4 Garanzie di prestazione

7.4.1 Il rapporto di prestazione nel caso peggiore di LIST

Proposizione 7.2

Per $P \,||\, C_{\max}$, per ogni istanza I e ogni lista λ,

$$\frac{LIST(I)}{z(I)} \le 2 - \frac{1}{m}.$$

Dimostrazione. Sia ℓ un lavoro che termina per ultimo, così che $LIST(I) = S_\ell + t_\ell$. Per l'Osservazione 7.1,

$$LIST(I) \le \frac{1}{m}\sum_{j \neq \ell} t_j + t_\ell = \underbrace{\frac{1}{m}\sum_{j \in \mathscr{J}} t_j}_{\le z(I)} + \Big(1 - \frac{1}{m}\Big) \underbrace{t_\ell}_{\le z(I)} \le \Big(2 - \frac{1}{m}\Big) z(I),$$

dove entrambi i bound vengono dalla (7.1). Dividendo per $z(I)$ si ottiene la tesi. □

Proposizione 7.3

Per ogni $m \ge 2$ esistono, per $P \,||\, C_{\max}$, un'istanza I_m con m macchine e una lista λ tali

che

$$\frac{LIST(I_m)}{z(I_m)} = 2 - \frac{1}{m}.$$

Dimostrazione. I_m ha $n = m(m-1)+1$ lavori: i lavori $1, \ldots, m(m-1)$ con $t_j = 1$ e il lavoro n con $t_n = m$; la lista è $\lambda = (1, 2, \ldots, n)$. Abbiamo $n > m$ per $m \geq 2$.

I lavori unitari vengono assegnati a turni di m: in ogni turno tutte le macchine hanno lo stesso carico, i lavori vanno sulle macchine $1, 2, \ldots, m$ in quest'ordine e ogni carico cresce di 1. Dopo gli $m-1$ turni tutti i carichi valgono $m-1$, e il lavoro n inizia in $m-1$ sulla macchina 1:

$$LIST(I_m) = (m-1) + m = 2m - 1.$$

Mettendo il lavoro n da solo sulla macchina 1 e gli $m(m-1)$ lavori unitari, m per macchina, sulle macchine $2, \ldots, m$ si ottiene makespan m, e la (7.1) dà $z(I_m) \geq t_n = m$. Quindi $z(I_m) = m$ e

$$\frac{LIST(I_m)}{z(I_m)} = \frac{2m-1}{m} = 2 - \frac{1}{m}.$$

□

Dalla Proposizione 7.2 e dalla Proposizione 7.3 segue che $LIST$ è un algoritmo di $\left(2 - \frac{1}{m}\right)$-approssimazione, qualunque sia la lista, e che il rapporto di prestazione nel caso peggiore di $LIST$ su m macchine è

$$r(LIST) = 2 - \frac{1}{m}$$

- Per $m = 3$ la famiglia ha $t = (1, 1, 1, 1, 1, 1, 3)$, $LIST(I_3) = 5$ e $z(I_3) = 3$. Il lavoro critico è quello lungo, messo per ultimo: LPT lo metterebbe per primo e troverebbe l'ottimo.
- Nell'Esempio 7.1, $LIST(I) = 12 \leq \frac{5}{3} \cdot 9 = 15$.
- Il bound e la famiglia sono dovuti a Graham (1966).

7.4.2 Il rapporto di prestazione nel caso peggiore di LPT

Proposizione 7.4

Per $P \,||\, C_{\max}$, per ogni istanza I,

$$\frac{LPT(I)}{z(I)} \leq \frac{4}{3} - \frac{1}{3m}.$$

Dimostrazione. Numeriamo i lavori in modo che $t_1 \geq t_2 \geq \cdots \geq t_n$, l'ordine in cui LPT li assegna, e sia ℓ un lavoro che termina per ultimo, così che $LPT(I) = S_\ell + t_\ell$. I lavori dopo ℓ non hanno alcun ruolo: sia I' l'istanza con i soli lavori $1, \ldots, \ell$. LPT li assegna su I' esattamente come su I, quindi $LPT(I') = LPT(I)$, e $z(I') \leq z(I)$ poiché uno schedule di I ristretto ai lavori di I' è uno schedule di I'. Basta quindi dimostrare $LPT(I') \leq \left(\frac{4}{3} - \frac{1}{3m}\right) z(I')$. In I' il lavoro ℓ è il più breve. Distinguiamo due casi.

Caso 1: $t_\ell \le \frac{z(I')}{3}$**.** Come nella dimostrazione della Proposizione 7.2, con l'Osservazione 7.1 applicata a I',

$$LPT(I') \;\le\; \frac{1}{m}\sum_{j=1}^{\ell} t_j + \Big(1-\frac{1}{m}\Big)t_\ell \;\le\; z(I') + \Big(1-\frac{1}{m}\Big)\frac{z(I')}{3} \;=\; \Big(\frac{4}{3}-\frac{1}{3m}\Big)\,z(I').$$

Caso 2: $t_\ell > \frac{z(I')}{3}$**.** Ogni lavoro di I' è più lungo di $\frac{z(I')}{3}$, quindi in uno schedule ottimo di I' nessuna macchina ha tre lavori, e $\ell \le 2m$. Se $\ell \le m$, il lavoro ℓ inizia su una macchina vuota e $LPT(I') = t_\ell \le z(I')$. Altrimenti $\ell = m + r$ con $1 \le r \le m$; poniamo $k := m - r + 1$, così che $1 \le k \le m$. Mostriamo che $z(I') \ge t_k + t_\ell \ge LPT(I')$.

Lower bound. Consideriamo uno schedule ottimo di I'. Se due dei lavori $1,\dots,k$ condividono una macchina, il suo carico è almeno $2t_k \ge t_k + t_\ell$. Altrimenti stanno su k macchine diverse; le altre $m - k = r - 1$ macchine ricevono al più $2(r-1)$ lavori, mentre i lavori $k+1,\dots,\ell$ sono $\ell - k = 2r - 1$. Quindi uno di essi condivide una macchina con uno dei lavori $1,\dots,k$, e quel carico è almeno $t_k + t_\ell$. In entrambi i casi $z(I') \ge t_k + t_\ell$.

Upper bound. I primi m lavori vanno su m macchine diverse, poiché una macchina vuota ha carico 0 e ogni lavoro ha durata positiva. Prima del lavoro ℓ, le m macchine contengono $\ell - 1 = m + r - 1$ lavori, almeno uno per macchina, quindi al più $r - 1$ macchine contengono due o più lavori e almeno $m - r + 1 = k$ macchine contengono esattamente un lavoro, uno dei lavori $1,\dots,m$. Sono k lavori diversi fra $1,\dots,m$, quindi il massimo indice fra di essi è almeno k, e la sua durata è al più t_k. Quindi il carico minimo prima del lavoro ℓ è $S_\ell \le t_k$, e $LPT(I') = S_\ell + t_\ell \le t_k + t_\ell$.

Pertanto nel Caso 2 vale $LPT(I') \le z(I')$, risultato ancora migliore della tesi. □

Proposizione 7.5

Per ogni $m \ge 2$ esiste, per $P\,||\,C_{\max}$, un'istanza I_m con m macchine tale che

$$\frac{LPT(I_m)}{z(I_m)} \;=\; \frac{4}{3} - \frac{1}{3m}.$$

Dimostrazione. I_m ha $n = 2m + 1$ lavori, già ordinati: due lavori per ciascuna durata $2m-1, 2m-2, \dots, m+1$, e tre lavori di durata m,

$$t = (2m-1,\ 2m-1,\ 2m-2,\ 2m-2,\ \dots,\ m+1,\ m+1,\ m,\ m,\ m).$$

Per $j \le 2m$ la durata è $t_j = 2m - \lceil j/2 \rceil$, e $t_{2m+1} = m$.

Ottimo. La durata totale è $2\sum_{v=m+1}^{2m-1} v + 3m = 3m^2$, quindi la (7.1) dà $z(I_m) \ge 3m$. Lo schedule che accoppia un lavoro di durata v con un lavoro di durata $3m - v$, per $v = m+1,\dots,2m-1$ (entrambe le durate stanno in $\{m+1,\dots,2m-1\}$, e ogni durata si usa due volte), riempie $m-1$ macchine con carico $3m$, e mette i tre lavori di durata m sull'ultima macchina, di nuovo con carico $3m$. Quindi $z(I_m) = 3m$.

LPT. I lavori $1,\dots,m$ vanno sulle macchine $1,\dots,m$. Poi il lavoro $m+1$ va su una macchina con il lavoro singolo più breve, t_m, e il suo carico diventa $t_m + t_{m+1}$. In generale,

affermiamo che il lavoro $m+q$, per $q = 1, \ldots, m$, va su una macchina il cui unico lavoro ha durata t_{m+1-q}, e il carico diventa

$$t_{m+1-q} + t_{m+q} = 3m - 1.$$

L'identità. Con $j = m+1-q$, così che $2m+1-j = m+q$: se $j = 2i-1$ è dispari allora $\lceil j/2 \rceil = i$ e $\lceil (2m+1-j)/2 \rceil = m-i+1$, e se $j = 2i$ è pari le due parti intere superiori sono ancora i e $m-i+1$. In entrambi i casi $t_j + t_{2m+1-j} = (2m-i)+(m+i-1) = 3m-1$.

L'affermazione. Segue per induzione. Una macchina con due lavori ha carico $3m-1 > 2m-1 \geq$ ogni lavoro singolo, quindi non è mai la meno carica finché restano macchine con un solo lavoro. Fra le macchine con un solo lavoro, la meno carica contiene il lavoro rimanente più breve, di durata t_{m+1-q} (a parità di durata i carichi sono uguali).

Dopo $2m$ lavori tutti i carichi valgono $3m-1$, e l'ultimo lavoro, di durata m, dà

$$LPT(I_m) = 3m - 1 + m = 4m - 1, \qquad \frac{LPT(I_m)}{z(I_m)} = \frac{4m-1}{3m} = \frac{4}{3} - \frac{1}{3m}.$$

□

Dalla Proposizione 7.4 e dalla Proposizione 7.5 segue che LPT è un algoritmo di $\left(\frac{4}{3} - \frac{1}{3m}\right)$-approssimazione, e che il suo rapporto di prestazione nel caso peggiore su m macchine è

$$r(LPT) = \frac{4}{3} - \frac{1}{3m}$$

Ordinare la lista abbassa la garanzia di $LIST$ da $2 - \frac{1}{m}$ a $\frac{4}{3} - \frac{1}{3m}$.

- Per $m = 3$ la famiglia è $t = (5, 5, 4, 4, 3, 3, 3)$, l'istanza di riferimento: i carichi dopo sei lavori sono $(8, 8, 8) = 3m-1$, il settimo lavoro dà $LPT(I) = 11 = 4m-1$, e $z(I) = 9 = 3m$ (Esempio 7.2). L'istanza di riferimento è un caso peggiore per LPT: $\frac{11}{9} = \frac{4}{3} - \frac{1}{9}$.
- Sulle istanze tight l'ultimo lavoro ha $t_\ell = m = \frac{z(I_m)}{3}$: stanno esattamente sul confine fra i due casi della dimostrazione, e ogni disuguaglianza del Caso 1 vale all'uguaglianza.
- Il bound e la famiglia sono dovuti a Graham (1969).

7.5 Fonti

- R.L. Graham, Bounds for certain multiprocessing anomalies, *Bell System Technical Journal* 45(9):1563–1581, 1966.
- R.L. Graham, Bounds on multiprocessing timing anomalies, *SIAM Journal on Applied Mathematics* 17(2):416–429, 1969.

7.6 Errori tipici

Attenzione

- **Usare la garanzia di LPT per una lista non ordinata.** $LIST$ ha la garanzia $2 - \frac{1}{m}$ per ogni lista; la garanzia migliore $\frac{4}{3} - \frac{1}{3m}$ richiede i lavori ordinati per tempo di esecuzione non crescente.
- **Calcolare il rapporto con il lower bound.** Il bound (7.1) può essere minore di $z(I)$: il rapporto $\mathscr{A}(I)/\mathrm{LB}$ è solo un upper bound sul rapporto di prestazione $\mathscr{A}(I)/z(I)$, e i due coincidono solo quando il bound è tight.
- **Risolvere i pareggi con un'altra regola.** $LIST$ assegna ogni lavoro a una macchina meno carica di indice minimo; un'altra regola per i pareggi può dare uno schedule diverso, e le soluzioni degli esercizi seguono la regola dell'algoritmo List Scheduling.

7.7 In sintesi

In sintesi

- Le garanzie di $LIST$ e LPT si dimostrano con il bound sui carichi (7.1) su $z(I)$ e con l'Osservazione 7.1: il lavoro che termina per ultimo inizia non più tardi del carico medio degli altri lavori.
- Entrambi i rapporti sono tight, con le famiglie di istanze di Graham; per $m = 3$ l'istanza di riferimento è l'istanza tight di LPT.
- Ordinare la lista abbassa la garanzia da $2 - \frac{1}{m}$ a $\frac{4}{3} - \frac{1}{3m}$.

problema	algoritmo	rapporto nel caso peggiore
$P \,\|\|\, C_{\max}$	$LIST$	$2 - \frac{1}{m}$ (tight)
$P \,\|\|\, C_{\max}$	LPT	$\frac{4}{3} - \frac{1}{3m}$ (tight)

Quadro riassuntivo dell'istanza di riferimento. I bound su $z(I)$ trovati finora:

Istanza	Lower bound	Valori euristici	Valore ottimo
$P \,\|\|\, C_{\max}$, $m = 3$, $t = (5, 5, 4, 4, 3, 3, 3)$	9, bound sui carichi	$LIST$ 12 (lista $7, 6, \ldots, 1$), LPT 11	9