



SAPIENZA
UNIVERSITÀ DI ROMA

Ottimizzazione combinatoria

Volume 6 — Algoritmi ad hoc

Algoritmi ad hoc: il branch-and-bound sull'elemento critico per il problema dello zaino 0/1 e il branch-and-bound DSATUR per la colorazione dei vertici.

Fabio Furini

fabiofurini.github.io

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

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

Anno accademico 2026–2027

Indice

Problemi di zaino	2
41 Zaino: branch-and-bound sull'elemento critico	3
41.1 L'algoritmo per il KP	4
41.2 Un esempio svolto	6
41.3 Errori tipici	12
41.4 In sintesi	12
 Problemi di colorazione	 13
42 Colorazione dei vertici: il branch-and-bound DSATUR	14
42.1 L'algoritmo per il VCP	15
42.2 Un esempio svolto	19
42.3 Osservazioni	22
42.4 Errori tipici	22
42.5 In sintesi	23

Problemi di zaino



Zaino: branch-and-bound sull'elemento critico

Prerequisiti

- Dal modulo *Programmazione lineare intera*, [Capitolo 7 «Branch-and-bound»](#): operazioni di branching e di bounding, lista dei problemi, criteri di potatura, lower bound LB e upper bound UB. Qui l'algoritmo generale si applica al problema dello zaino 0/1.
- Da questo modulo, [Capitolo 9 «Formulazioni dei problemi di zaino»](#): i modelli ILP_{KP} e LP_{KP} , l'efficienza di un oggetto, l'oggetto critico s , la capacità residua rc e il bound di Dantzig.

Obiettivi

Alla fine di questo capitolo saremo in grado di:

- scrivere il problema LP_{KP}^k di un nodo e risolverlo con le formule di Dantzig, senza l'algoritmo del simplesso;
- fare branching sulla variabile dell'oggetto critico, escludendo o selezionando quell'oggetto;
- eseguire a mano l'algoritmo di branch-and-bound per il KP, con la regola FIFO o LIFO;
- spiegare perché l'algoritmo è più semplice di quello generale, e che cosa resta uguale.

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

41.1 L'algoritmo per il KP

Definizione 41.1: Nodo di branching e problema LP_{KP}^k

Nell'algoritmo di branch-and-bound per il modello ILP_{KP} , i vincoli di branching su una variabile x_j , $j \in \mathcal{J}$, sono $x_j \leq 0$ e $x_j \geq 1$. In un nodo di branching k dell'albero di branch-and-bound abbiamo:

1. un sottoinsieme di **oggetti esclusi** $\rightsquigarrow$ indicato con $\mathcal{J}_{out}^k$
2. un sottoinsieme di **oggetti selezionati** $\rightsquigarrow$ indicato con $\mathcal{J}_{in}^k$
3. un sottoinsieme di **oggetti rimanenti** $\rightsquigarrow$ indicato con $\mathcal{J}^k := \mathcal{J} \setminus \{\mathcal{J}_{out}^k \cup \mathcal{J}_{in}^k\}$

Il problema LP_{KP} associato al nodo k , detto problema LP_{KP}^k , è il seguente:

$$\begin{aligned} \max \quad & \sum_{j \in \mathcal{J}} p_j x_j \\ \text{soggetto a} \quad & \sum_{j \in \mathcal{J}} w_j x_j \leq c, \\ & x_j \geq 1, \quad \forall j \in \mathcal{J}_{in}^k, \\ & x_j \leq 0, \quad \forall j \in \mathcal{J}_{out}^k, \\ & x_j \leq 1, \quad \forall j \in \mathcal{J}^k, \\ & x_j \geq 0, \quad \forall j \in \mathcal{J}. \end{aligned}$$

- I vincoli di branching $x_j \leq 0$ e $x_j \geq 1$ hanno l'effetto di **escludere** o **selezionare** un oggetto.

Il problema LP_{KP}^k di un nodo di branching k si può riscrivere in modo equivalente come:

$$\underbrace{\sum_{j \in \mathcal{J}_{in}^k} p_j}_{=p(k)} + \max \sum_{j \in \mathcal{J}^k} p_j x_j \quad (41.1a)$$

$$\text{soggetto a} \quad \sum_{j \in \mathcal{J}^k} w_j x_j \leq c - \underbrace{\sum_{j \in \mathcal{J}_{in}^k} w_j}_{=c(k)}, \quad (41.1b)$$

$$x_j \leq 1, \quad \forall j \in \mathcal{J}^k, \quad (41.1c)$$

$$x_j \geq 0, \quad \forall j \in \mathcal{J}^k. \quad (41.1d)$$

- Il problema LP_{KP}^k di un nodo di branching k è **strutturalmente equivalente** al problema LP_{KP} , ma:

1. l'insieme degli oggetti è $\mathcal{J}^k$ invece di $\mathcal{J}$
2. la capacità si riduce a $c(k) := c - \sum_{j \in \mathcal{J}_{\text{in}}^k} w_j$
3. c'è una costante nella funzione obiettivo $p(k) := \sum_{j \in \mathcal{J}_{\text{in}}^k} p_j$

41.1.1 Risoluzione del problema LP_{KP}^k di un nodo

- Il problema LP_{KP}^k è un problema LP di zaino sugli oggetti di $\mathcal{J}^k$, con capacità $c(k)$: si risolve con le formule di Dantzig, senza alcuna iterazione del semplice. Gli oggetti di $\mathcal{J}^k$ restano ordinati per efficienza non crescente.
- Se $c(k) < 0$ il problema è **inammissibile**. Se $\sum_{j \in \mathcal{J}^k} w_j \leq c(k)$ tutti gli oggetti di $\mathcal{J}^k$ entrano, e la soluzione ottima è $\tilde{x}_j = 1$ per ogni $j \in \mathcal{J}^k$: è intera.
- Altrimenti l'**oggetto critico** s del nodo (in inglese **split item** o **critical item**) è il primo oggetto di $\mathcal{J}^k$ che non entra per intero, e la **capacità residua** è la capacità lasciata dagli oggetti che lo precedono:

$$\sum_{j \in \mathcal{J}^k, j < s} w_j \leq c(k) < \sum_{j \in \mathcal{J}^k, j \leq s} w_j, \quad rc = c(k) - \sum_{j \in \mathcal{J}^k, j < s} w_j.$$

- La soluzione ottima è $\tilde{x}_j = 1$ per gli oggetti di $\mathcal{J}^k$ prima di s , $\tilde{x}_s = \frac{rc}{w_s}$ e $\tilde{x}_j = 0$ per gli oggetti dopo s , e il valore ottimo è

$$z(\text{LP}_{\text{KP}}^k) = p(k) + \sum_{j \in \mathcal{J}^k, j < s} p_j + p_s \frac{rc}{w_s}.$$

In ogni nodo al più una variabile è frazionaria, quella dell'oggetto critico, e il valore del nodo è il bound di Dantzig di uno zaino più piccolo: il nodo si risolve con una scansione degli oggetti.

41.1.2 Strategia di scelta della variabile di branching

- La soluzione ottima di un problema LP_{KP}^k data dalle formule della sottosezione precedente ha al più una variabile frazionaria, quella associata all'oggetto critico. **Il branching si può fare sulla variabile associata all'oggetto critico**: questa strategia di scelta della variabile di branching si chiama strategia dell'oggetto critico.
- La soluzione ottima di un problema LP_{KP} non è unica quando alcuni oggetti hanno la stessa efficienza: le altre soluzioni ottime possono avere più variabili frazionarie, per esempio $x = (\frac{1}{2}, \frac{1}{2})$ per $p = (2, 2)$, $w = (2, 2)$, $c = 2$. Qui, come nell'algoritmo generale, usiamo sempre la soluzione data dalle formule.

41.1.3 Che cosa cambia rispetto all'algoritmo generale

- L'algoritmo è il branch-and-bound basato sulla PL del modulo *Programmazione lineare intera* (Capitolo 7 «Branch-and-bound»), con la sua lista dei problemi, i suoi criteri di potatura e il suo gap di ottimalità.
- **Bounding.** Il problema di un nodo si risolve con le formule precedenti. Tutti i profitti sono interi, quindi il bound di un nodo è $\lfloor z(\text{LP}_{\text{KP}}^k) \rfloor$: il nodo è potato per bound se $\lfloor z(\text{LP}_{\text{KP}}^k) \rfloor \leq \text{LB}$, per inammissibilità se $c(k) < 0$, per interezza se la sua soluzione è intera (e allora LB si aggiorna quando il valore è maggiore). Un nodo intero è potato per interezza anche quando il suo valore non supera LB.
- **Branching.** La variabile di branching è quella dell'oggetto critico s : i figli sono $x_s \leq 0$ (down, oggetto escluso) e $x_s \geq 1$ (up, oggetto selezionato), e il nodo non si risolve con un tableau del simplesso ma si descrive con gli insiemi $\mathcal{J}_{\text{in}}^k$, $\mathcal{J}_{\text{out}}^k$ e $\mathcal{J}^k$.
- **Scelta del nodo.** Il nodo della lista dei problemi si estrae con la regola FIFO o LIFO, e il nodo di down-branching si risolve prima del nodo di up-branching; i nodi sono numerati nell'ordine di risoluzione. Il lower bound scritto in un nodo è il valore al momento della sua risoluzione.

41.2 Un esempio svolto

Esempio 41.1: Il modello ILP_{KP}

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

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

Chiamiamo questa istanza del KP [Esempio 41.1](#). Il modello ILP_{KP} per questa istanza del KP è:

$$\begin{aligned}
 \max \quad & 6x_1 + 5x_2 + 8x_3 + 9x_4 + 6x_5 + 7x_6 \\
 \text{soggetto a} \quad & 2x_1 + 3x_2 + 6x_3 + 7x_4 + 5x_5 + 9x_6 \leq 9 \\
 & x_1, x_2, x_3, x_4, x_5, x_6 \in \{0, 1\}
 \end{aligned}$$

Una soluzione ottima $\tilde{x}(\text{ILP}_{\text{KP}})$ è:

$$\tilde{x}_1 = 1, \tilde{x}_2 = 0, \tilde{x}_3 = 0, \tilde{x}_4 = 1, \tilde{x}_5 = 0, \tilde{x}_6 = 0$$

Il valore ottimo $z(\text{ILP}_{\text{KP}})$ è 15. La soluzione ottima associata del KP è $\tilde{S} = \{1, 4\}$.

Gli oggetti sono ordinati per efficienza non crescente, poiché:

$$\frac{6}{2} \geq \frac{5}{3} \geq \frac{8}{6} \geq \frac{9}{7} \geq \frac{6}{5} \geq \frac{7}{9}$$

Esempio 41.2: Algoritmo di branch-and-bound per il modello ILP_{KP}

Un algoritmo di branch-and-bound per il modello ILP_{KP} e l'istanza del KP dell'Esempio 41.1 che usa:

1. la strategia dell'oggetto critico come strategia di scelta della variabile di branching;
2. la regola FIFO come strategia di scelta del nodo di branching;
3. risolvere prima il nodo di down-branching e poi il nodo di up-branching;
4. porre la soluzione incumbent $\bar{x}(ILP_{KP})$ uguale alla seguente soluzione euristica:

$$\bar{x}_1 = 1, \quad \bar{x}_2 = 0, \quad \bar{x}_3 = 0, \quad \bar{x}_4 = 1, \quad \bar{x}_5 = 0, \quad \bar{x}_6 = 0$$

e porre di conseguenza il lower bound LB.

• **Inizializzazione**

Per il problema LP_{KP}^1 , l'oggetto critico è $s = 3$ e la capacità residua è $rc = 4$. Una soluzione ottima $\tilde{x}(LP_{KP}^1)$ è:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = 1, \quad \tilde{x}_3 = \frac{2}{3}, \quad \tilde{x}_4 = 0, \quad \tilde{x}_5 = 0, \quad \tilde{x}_6 = 0$$

il valore ottimo è:

$$z(LP_{KP}^1) = 6 \cdot 1 + 5 \cdot 1 + 8 \cdot \frac{2}{3} = \frac{49}{3}$$

La soluzione ottima $\tilde{x}(LP_{KP}^1)$ è **frazionaria** e il problema LP_{KP}^1 è **ammissibile**, quindi il problema LP_{KP}^1 viene inserito nella lista dei problemi L e il lower bound LB viene posto a 15 ($LB(\bar{x}) = 15$).

Dati il lower bound LB e l'upper bound UB correnti, abbiamo:

$$15 \leq z(ILP_{KP}) \leq \underbrace{\left\lfloor \frac{49}{3} \right\rfloor}_{=16}$$

• **Iterazione 1**

Il problema LP_{KP}^1 viene estratto e cancellato dalla lista dei problemi L .

La variabile di branching è x_3 . I problemi LP_{KP}^2 e LP_{KP}^3 vengono creati escludendo ($x_3 \leq 0$) e selezionando ($x_3 \geq 1$) l'oggetto 3.

1. Soluzione del problema LP_{KP}^2

Abbiamo

$$\mathcal{J}^2 = \{1, 2, 4, 5, 6\}, \quad \mathcal{J}_{out}^2 = \{3\}, \quad \mathcal{J}_{in}^2 = \emptyset$$

La capacità è $c(2) = 9$ e la costante nella funzione obiettivo è $p(2) = 0$. L'oggetto critico è $s = 4$ e la capacità residua è $rc = 4$. Una soluzione ottima $\tilde{x}(LP_{KP}^2)$ è:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = 1, \quad \tilde{x}_4 = \frac{4}{7}, \quad \tilde{x}_5 = 0, \quad \tilde{x}_6 = 0$$

e

$$z(LP_{KP}^2) = 6 \cdot 1 + 5 \cdot 1 + 9 \cdot \frac{4}{7} = \frac{113}{7}$$

Il problema LP_{KP}^2 è **ammissibile** e abbiamo:

$$\lfloor z(LP_{KP}^2) \rfloor > LB \quad \text{poiché} \quad \underbrace{\left\lfloor \frac{113}{7} \right\rfloor}_{=16} > 15$$

La soluzione ottima $\tilde{x}(LP_{KP}^2)$ è **frazionaria**, quindi il problema LP_{KP}^2 viene **inserito nella lista dei problemi L** .

 2. Soluzione del problema LP_{KP}^3

Abbiamo

$$\mathcal{J}^3 = \{1, 2, 4, 5, 6\}, \quad \mathcal{J}_{out}^3 = \emptyset, \quad \mathcal{J}_{in}^3 = \{3\}$$

La capacità è $c(3) = 3$ e la costante nella funzione obiettivo è $p(3) = 8$. L'oggetto critico è $s = 2$ e la capacità residua è $rc = 1$. Una soluzione ottima $\tilde{x}(LP_{KP}^3)$ è:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = \frac{1}{3}, \quad \tilde{x}_4 = 0, \quad \tilde{x}_5 = 0, \quad \tilde{x}_6 = 0 \quad \text{e} \quad z(LP_{KP}^3) = 8 + 6 \cdot 1 + 5 \cdot \frac{1}{3} = \frac{47}{3}$$

Il problema LP_{KP}^3 è **ammissibile** e abbiamo:

$$\lfloor z(LP_{KP}^3) \rfloor \leq LB \quad \text{poiché} \quad \underbrace{\left\lfloor \frac{47}{3} \right\rfloor}_{=15} \leq 15$$

quindi il nodo di branching è potato per upper bound.

Alla fine della prima iterazione, il Nodo 2 è inserito nella lista dei problemi L , mentre il Nodo 3 è potato per upper bound. L'albero di branch-and-bound corrispondente è:

Nodo 1

$$\begin{array}{l}
 \max \quad 6x_1 + 5x_2 + 8x_3 + 9x_4 + 6x_5 + 7x_6 \\
 2x_1 + 3x_2 + 6x_3 + 7x_4 + 5x_5 + 9x_6 \leq 9 \\
 x_1, x_2, x_3, x_4, x_5, x_6 \leq 1 \\
 x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \\
 \tilde{x}_1 = 1, \tilde{x}_2 = 1, \tilde{x}_3 = \frac{2}{3}, \tilde{x}_4 = 0, \tilde{x}_5 = 0, \tilde{x}_6 = 0 \\
 \lfloor z(\text{LP}_{\text{KP}}^1) \rfloor = \underbrace{\left\lfloor \frac{49}{3} \right\rfloor}_{=16} \\
 \text{LB} = 15
 \end{array}$$

Nodo 2

$$x_3 \leq 0$$

$$\begin{array}{l}
 \max \quad 6x_1 + 5x_2 + 9x_4 + 6x_5 + 7x_6 \\
 2x_1 + 3x_2 + 7x_4 + 5x_5 + 9x_6 \leq 9 \\
 x_1, x_2, x_4, x_5, x_6 \leq 1 \\
 x_1, x_2, x_4, x_5, x_6 \geq 0 \\
 \tilde{x}_1 = 1, \tilde{x}_2 = 1, \tilde{x}_4 = \frac{4}{7}, \tilde{x}_5 = 0, \tilde{x}_6 = 0 \\
 \lfloor z(\text{LP}_{\text{KP}}^2) \rfloor = \underbrace{\left\lfloor \frac{113}{7} \right\rfloor}_{=16} \\
 \text{LB} = 15
 \end{array}$$

Nodo 3

$$x_3 \geq 1$$

$$\begin{array}{l}
 8 + \max \quad 6x_1 + 5x_2 + 9x_4 + 6x_5 + 7x_6 \\
 2x_1 + 3x_2 + 7x_4 + 5x_5 + 9x_6 \leq 3 \\
 x_1, x_2, x_4, x_5, x_6 \leq 1 \\
 x_1, x_2, x_4, x_5, x_6 \geq 0 \\
 \tilde{x}_1 = 1, \tilde{x}_2 = \frac{1}{3}, \tilde{x}_4 = 0, \tilde{x}_5 = 0, \tilde{x}_6 = 0 \\
 \lfloor z(\text{LP}_{\text{KP}}^3) \rfloor = \underbrace{\left\lfloor \frac{47}{3} \right\rfloor}_{=15} \\
 \text{LB} = 15 \\
 \text{potato per upper bound}
 \end{array}$$

Dati il lower bound LB e l'upper bound UB correnti, abbiamo:

$$15 \leq z(\text{ILP}_{\text{KP}}) \leq \underbrace{\left\lfloor \frac{113}{7} \right\rfloor}_{=16}$$

• Iterazione 2

Il problema LP_{KP}^2 viene estratto e cancellato dalla lista dei problemi L .

La variabile di branching è x_4 . I problemi LP_{KP}^4 e LP_{KP}^5 vengono creati escludendo ($x_4 \leq 0$) e selezionando ($x_4 \geq 1$) l'oggetto 4.

 1. Soluzione del problema LP_{KP}^4

Abbiamo

$$\mathcal{J}^4 = \{1, 2, 5, 6\}, \quad \mathcal{J}_{\text{out}}^4 = \{3, 4\}, \quad \mathcal{J}_{\text{in}}^4 = \emptyset$$

La capacità è $c(4) = 9$ e la costante nella funzione obiettivo è $p(4) = 0$. L'oggetto critico è $s = 5$ e la capacità residua è $rc = 4$. Una soluzione ottima $\tilde{x}(\text{LP}_{\text{KP}}^4)$ è:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = 1, \quad \tilde{x}_5 = \frac{4}{5}, \quad \tilde{x}_6 = 0 \quad \text{e} \quad z(\text{LP}_{\text{KP}}^4) = 6 \cdot 1 + 5 \cdot 1 + 6 \cdot \frac{4}{5} = \frac{79}{5}$$

Il problema LP_{KP}^4 è **ammissibile** e abbiamo:

$$\lfloor z(LP_{KP}^4) \rfloor \leq LB \quad \text{poiché} \quad \underbrace{\left\lfloor \frac{79}{5} \right\rfloor}_{=15} \leq 15$$

quindi il nodo di branching è potato per upper bound.

2. Soluzione del problema LP_{KP}^5

Abbiamo

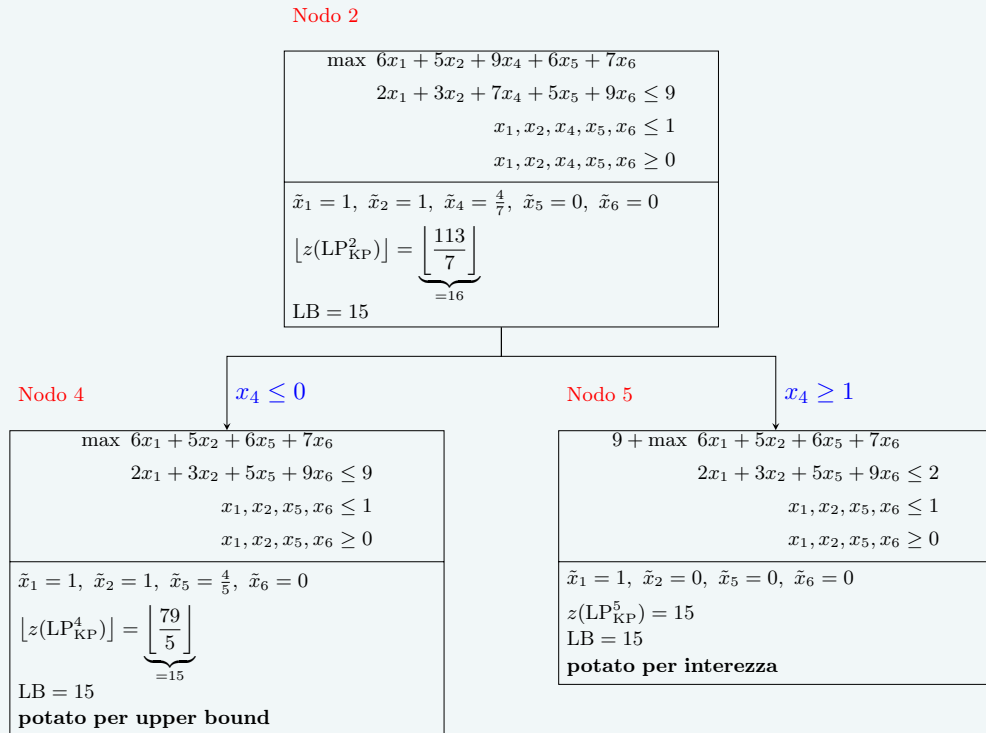
$$\mathcal{J}^5 = \{1, 2, 5, 6\}, \quad \mathcal{J}_{out}^5 = \{3\}, \quad \mathcal{J}_{in}^5 = \{4\}$$

La capacità è $c(5) = 2$ e la costante nella funzione obiettivo è $p(5) = 9$. L'oggetto critico è $s = 2$ e la capacità residua è $rc = 0$. Una soluzione ottima $\tilde{x}(LP_{KP}^5)$ è:

$$\tilde{x}_1 = 1, \quad \tilde{x}_2 = 0, \quad \tilde{x}_5 = 0, \quad \tilde{x}_6 = 0 \quad \text{e} \quad z(LP_{KP}^5) = 9 + 6 \cdot 1 = 15$$

Il problema LP_{KP}^5 è **ammissibile** e la sua soluzione ottima $\tilde{x}(LP_{KP}^5)$ è **intera**. Il suo valore non migliora la soluzione incumbent, perché $z(LP_{KP}^5) = 15 \leq 15 = LB$, quindi il lower bound LB resta 15. Di conseguenza, il nodo di branching è potato per interezza: quando un nodo si potrebbe potare sia per interezza sia per bound, lo si pota per interezza.

Alla fine della seconda iterazione, il Nodo 4 è potato per upper bound e il Nodo 5 è potato per interezza. L'albero di branch-and-bound corrispondente è:

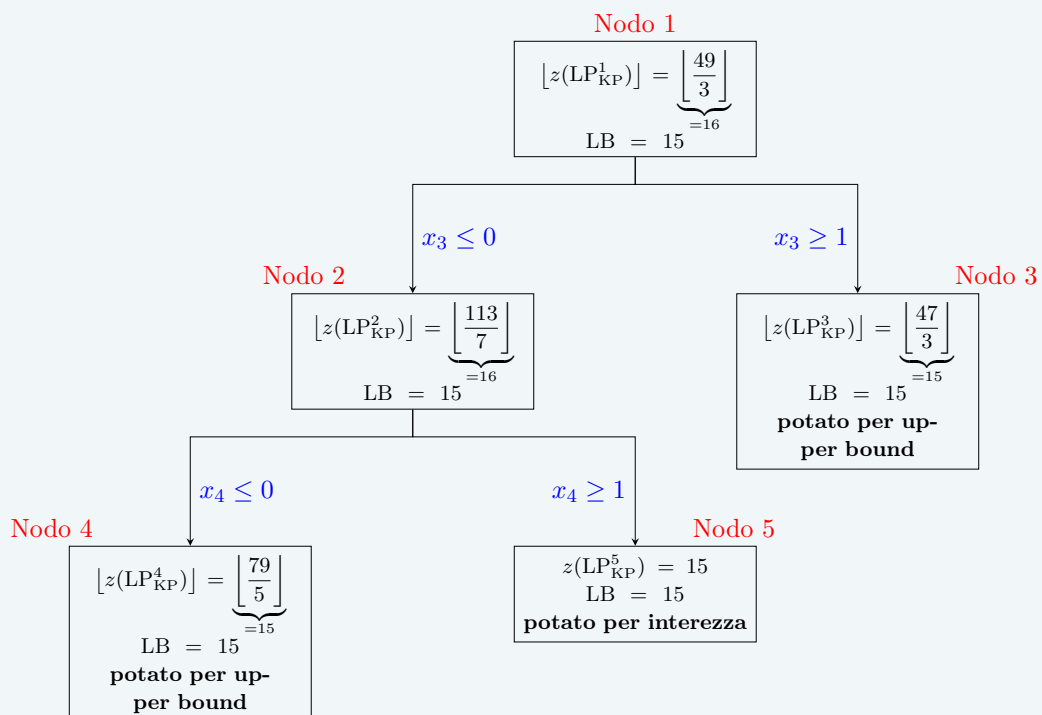


- **Terminazione**

Poiché la lista dei problemi L è vuota, il problema ILP_{KP} è risolto all'ottimo.

Il valore ottimo è $z(\text{ILP}_{\text{KP}}) = 15$ (il valore del lower bound LB) e una soluzione ottima $\tilde{\mathbf{x}}(\text{ILP}_{\text{KP}})$ è $\tilde{x}_1 = 1, \tilde{x}_2 = 0, \tilde{x}_3 = 0, \tilde{x}_4 = 1, \tilde{x}_5 = 0, \tilde{x}_6 = 0$ (la soluzione incumbent $\bar{\mathbf{x}}(\text{ILP}_{\text{KP}})$).

Questo è l'albero di branch-and-bound della precedente esecuzione dell'algoritmo di branch-and-bound, che riporta per ogni nodo di branching il valore ottimo del problema LP_{KP} associato, il lower bound LB e i criteri di potatura.



41.3 Errori tipici

Attenzione

- **Dimenticare i vincoli $x_j \leq 1$.** Senza di essi il problema LP di un nodo metterebbe tutta la capacità sull'oggetto di efficienza migliore: i box dei nodi devono contenere $x_j \leq 1$.
- **Calcolare l'oggetto critico sull'istanza originale.** In un nodo, l'oggetto critico e la capacità residua si calcolano sugli oggetti di $\mathcal{J}^k$ con la capacità $c(k)$, dopo aver fissato gli oggetti di $\mathcal{J}_{in}^k$ e $\mathcal{J}_{out}^k$.
- **Dimenticare la costante $p(k)$.** Il valore del nodo è $p(k)$ più il valore degli oggetti di $\mathcal{J}^k$: il profitto degli oggetti selezionati conta.
- **Usare il lower bound vecchio.** Il lower bound si aggiorna appena un nodo ha una soluzione intera di valore maggiore; i nodi della lista che diventano dominati si scartano.
- **Numerare i figli saltati.** Un figlio che non viene risolto non riceve un indice di nodo, esattamente come nell'algoritmo generale.

41.4 In sintesi

In sintesi

- L'algoritmo di branch-and-bound per il KP è l'algoritmo basato sulla PL con due semplificazioni: il problema di ogni nodo si risolve con le formule di Dantzig, e la variabile di branching è quella dell'oggetto critico.
- Un nodo è descritto dagli insiemi $\mathcal{J}_{in}^k$, $\mathcal{J}_{out}^k$ e $\mathcal{J}^k$, dalla capacità $c(k)$ e dalla costante $p(k)$.
- Il bound di un nodo è $\lfloor z(\text{LP}_{KP}^k) \rfloor$; quando la lista dei problemi è vuota il lower bound e l'upper bound coincidono e la soluzione incumbent è ottima.

Quadro riassuntivo dell'istanza del KP dell'Esempio 41.1 ($c = 9, p = (6, 5, 8, 9, 6, 7), w = (2, 3, 6, 7, 5, 9)$). I bound su $z(\text{ILP}_{KP})$ trovati finora:

Fonte	Lower bound	Upper bound	Da
Bound di Dantzig	–	16	$\lfloor \frac{49}{3} \rfloor$
Questo capitolo, inizializzazione	15	16	soluzione euristica $\{1, 4\}$
Questo capitolo, iterazione 1	15	16	nodo 2 nella lista, $\lfloor \frac{113}{7} \rfloor$
Questo capitolo, terminazione	15	15	lista vuota, 5 nodi

Problemi di colorazione

Colorazione dei vertici: il branch-and-bound DSATUR

Prerequisiti

- Dal modulo *Programmazione lineare intera*, [Capitolo 7 «Branch-and-bound»](#): operazioni di branching e di bounding, lista dei problemi, criteri di potatura, bound LB e UB.
- Da questo modulo, [Capitolo 4 «Problemi di colorazione»](#): il problema della colorazione dei vertici, clique, insiemi stabili e il bound della clique $\omega(G) \leq \chi(G)$.
- Da questo modulo, [Capitolo 30 «Euristiche costruttive»](#): il grado di saturazione e l euristica *DSATUR*.

Obiettivi

Al termine di questo capitolo saremo in grado di:

- descrivere un nodo dell'albero come colorazione parziale, con i suoi colori usati e i suoi gradi di saturazione;
- fare branching sul vertice di grado di saturazione massimo, con un figlio per ogni colore ammissibile e uno per un colore nuovo;
- calcolare l'upper bound con *DSATUR* e il lower bound con una clique greedy, e potare un nodo;
- eseguire a mano l'algoritmo di branch-and-bound DSATUR e dimostrare che è esatto.

Il problema di questo capitolo è il [problema della colorazione dei vertici \(VCP\)](#): la sua definizione e le sue ipotesi sono nella sua pagina. Su un grafo $G = (V, E)$, il suo valore ottimo è il numero cromatico $\chi(G)$.

42.1 L'algoritmo per il VCP

Definizione 42.1: Nodo di branching e colorazione parziale

Nell'algoritmo di branch-and-bound DSATUR i colori sono gli interi $1, 2, 3, \dots$, e un nodo di branching k dell'albero è una **colorazione parziale**:

1. un sottoinsieme di **vertici colorati** $\rightsquigarrow$ indicato con $V^k \subseteq V$;
2. un colore $col^k(v)$ per ogni $v \in V^k$, con $col^k(u) \neq col^k(v)$ per ogni $\{u, v\} \in E$, $u, v \in V^k$;
3. un sottoinsieme di **vertici non colorati** $\rightsquigarrow$ indicato con $U^k := V \setminus V^k$.

I colori usati nel nodo k sono $1, 2, \dots, q(k)$, dove $q(k)$ è il **numero di colori usati**, e le classi di colore sono $Q_c^k := \{v \in V^k : col^k(v) = c\}$, $c = 1, \dots, q(k)$. Per un vertice non colorato $v \in U^k$:

$$\begin{aligned}
 sat_k(v) &:= |\{col^k(u) : u \in N(v) \cap V^k\}| && \text{(grado di saturazione),} \\
 du_k(v) &:= |N(v) \cap U^k| && \text{(grado fra i vertici non colorati),} \\
 F_k(v) &:= \{c \in \{1, \dots, q(k)\} : Q_c^k \cap N(v) = \emptyset\} && \text{(colori ammissibili).}
 \end{aligned}$$

- Il grado di saturazione conta i colori *diversi* fra i vicini colorati di v , e $|F_k(v)| = q(k) - sat_k(v)$.
- Un nodo con $U^k = \emptyset$ è una **colorazione completa**: una soluzione ammissibile del VCP con $q(k)$ colori.

Un nodo dell'albero è una colorazione parziale; ogni figlio colora un vertice in più. Non si risolve alcun problema LP: i bound di un nodo vengono dal conteggio dei colori.

42.1.1 Branching

- **Vertice di branching.** Nel nodo k facciamo branching sul vertice non colorato v di **grado di saturazione massimo** $sat_k(v)$; i pareggi si risolvono con il $du_k(v)$ massimo, e poi con l'indice minimo. È la regola dell'euristica *DSATUR* (Brélaz, 1979).
- **Figli.** I figli del nodo k danno un colore a v :
 1. un figlio per ogni colore ammissibile $c \in F_k(v)$, in ordine crescente di c ;
 2. un figlio con il **colore nuovo** $q(k) + 1$.
- I colori $q(k) + 2, q(k) + 3, \dots$ non si usano: un figlio con colore $q(k) + 2$ differirebbe dal figlio con colore $q(k) + 1$ solo per il nome di un colore vuoto. I figli **non sono copie simmetriche** l'uno dell'altro.
- Il vertice più saturo ha meno figli: $|F_k(v)| + 1$. Se $F_k(v) = \emptyset$, il nodo ha solo il figlio con il colore nuovo.

Il vertice di branching è il più vincolato; i suoi figli sono i suoi colori ammissibili più un colore nuovo, quindi due figli non descrivono mai la stessa colorazione con i colori rinominati.

42.1.2 Bound e potatura

- Il VCP è un problema di minimo: la soluzione incumbent dà l'**upper bound** UB, e un lower bound viene da una clique.
- **Upper bound.** Prima del branching, l'euristica *DSATUR* del [Capitolo 30 «Euristiche costruttive»](#) dà la prima soluzione incumbent, e UB è il suo numero di colori.
- **Clique greedy.** I vertici si scorrono per grado non crescente (pareggi: indice minimo), e un vertice entra in K se è adiacente a tutti i vertici già in K . L'insieme $K = \{k_1, \dots, k_r\}$ è una clique massimale, e $LB := r \leq \omega(G) \leq \chi(G)$.
- **Pre-colorazione.** I vertici di K richiedono r colori diversi in ogni colorazione, e i nomi dei colori sono arbitrari: il nodo radice colora k_i con il colore i , $i = 1, \dots, r$. Questo rompe la simmetria dei primi r colori.
- **Lower bound di un nodo.** I colori di un nodo non vengono mai tolti nel suo sottoalbero, quindi ogni colorazione completa sotto il nodo k usa almeno $q(k)$ colori:

$$LB(k) := \max\{q(k), |K|\} = q(k),$$

dove l'uguaglianza vale perché la clique è pre-colorata.

- **Potatura.** Il numero di colori è un intero, quindi un nodo può portare a una soluzione migliore solo se usa meno di UB colori. Il nodo k è **potato per bound** se

$$q(k) \geq UB.$$

- **Colorazione completa.** Un nodo con $U^k = \emptyset$ e $q(k) < UB$ è una nuova soluzione incumbent: $UB \leftarrow q(k)$, e i nodi della lista con $q \geq UB$ vengono rimossi. Se $UB = |K|$, la soluzione incumbent è ottima e l'algoritmo si ferma.

DSATUR dà UB, una clique greedy dà LB; un nodo è potato appena usa UB colori, perché il suo sottoalbero non può contenere una colorazione con meno colori.

42.1.3 Scelta del nodo e pseudocodice

- L'albero si esplora **in profondità**: la lista L è una pila, i figli di un nodo si valutano in ordine crescente di colore, e il figlio con il colore minimo si estrae per primo.
- I nodi sono numerati nell'ordine di valutazione. Un figlio si valuta appena viene creato: la valutazione calcola solo q e controlla se la colorazione è completa.

Algoritmo

Input: un'istanza del VCP, cioè un grafo $G = (V, E)$.

Output: una colorazione ottima e il numero cromatico $\chi(G)$.

Passo 0: Inizializzazione

- Eseguire *DSATUR*: porre la soluzione incumbent uguale alla sua colorazione e UB uguale al suo numero di colori.
- Trovare la clique greedy $K = \{k_1, \dots, k_r\}$ e porre $LB \leftarrow r$. Se $LB = UB$, la soluzione incumbent è ottima. **STOP**.
- Creare il nodo 1 con $col^1(k_i) = i$, $i = 1, \dots, r$, e $q(1) = r$. Inserire il nodo 1 nella lista L e porre il contatore dei nodi $c \leftarrow 1$.

Passo 1: Test di ottimalità

Se L è vuota, UB è il valore ottimo e la soluzione incumbent è ottima. **STOP**.

Passo 2: Scelta del nodo di branching

Togliere da L il nodo k inserito per ultimo.

Passo 3: Branching

Scegliere $v \in U^k$ con $sat_k(v)$ massimo, poi $du_k(v)$ massimo, poi indice minimo. Per ogni colore $a \in F_k(v) \cup \{q(k) + 1\}$, in ordine crescente, creare il nodo $c + 1$ come il nodo k con $col(v) = a$, valutarlo con la **sotto-procedura del nodo** e porre $c \leftarrow c + 1$. Inserire i figli sopravvissuti in L in ordine decrescente di colore e andare al Passo 1.

- Nel Passo 3, il contatore c dei nodi dà il prossimo indice libero, e a è il colore dato a v nel figlio.
- Questo è lo pseudocodice della **sotto-procedura del nodo**.

Algoritmo

Input: un nodo h , cioè una colorazione parziale con $q(h)$ colori.

Passo 1: Operazioni di bounding

- Se $q(h) \geq UB$, il nodo è potato per bound. **STOP**.
- Se $U^h = \emptyset$, aggiornare l'upper bound e la soluzione incumbent: $UB \leftarrow q(h)$. Togliere da L tutti i nodi con $q \geq UB$. Se $UB = LB$, la soluzione incumbent è

ottima: STOP dell'algoritmo. Altrimenti STOP.

Passo 2: Sopravvivenza

Il nodo sopravvive: viene inserito in L nel Passo 3. STOP.

42.1.4 Correttezza

Proposizione 42.2: Branching senza simmetrie

Consideriamo l'albero dell'algoritmo di branch-and-bound DSATUR senza potatura. Le sue foglie sono colorazioni complete, e ogni partizione di V in insiemi stabili, cioè ogni soluzione ammissibile del VCP, è l'insieme delle classi di colore di **esattamente una** foglia.

Dimostrazione. Sia $\mathcal{Q}$ una partizione di V in insiemi stabili. I vertici della clique K sono adiacenti a due a due, quindi stanno in r classi diverse; diamo il nome i alla classe di k_i .

1. **Esistenza.** Seguiamo un cammino dalla radice mantenendo questo invariante: le classi del nodo sono le classi di $\mathcal{Q}$ ristrette a V^k , ciascuna con il suo nome. La radice lo soddisfa. Nel nodo k , siano v il vertice di branching e $Q \in \mathcal{Q}$ la sua classe.

- Se Q ha già un vertice colorato, con nome c , allora nessun vicino di v ha colore c , perché Q è stabile. Quindi $c \in F_k(v)$, e il figlio con colore c mantiene l'invariante.
- Altrimenti Q non ha ancora vertici colorati: il figlio con il colore nuovo $q(k) + 1$ mantiene l'invariante, e Q prende il nome $q(k) + 1$.

Ogni passo colora un vertice, quindi dopo $|U^1|$ passi raggiungiamo una foglia le cui classi sono $\mathcal{Q}$.

2. **Unicità.** Due figli diversi di un nodo mettono v in classi diverse: con i vertici colorati di due colori diversi, oppure con uno di essi e da solo. La partizione ristretta a $V^k \cup \{v\}$ è diversa, quindi le foglie sotto due figli diversi danno partizioni diverse. $\square$

- **L'algoritmo è esatto.** Un nodo è potato solo se $q(k) \geq \text{UB}$, e ogni foglia sotto di esso usa almeno $q(k)$ colori: non si perde alcuna colorazione con meno di UB colori. Quando la lista è vuota, $\text{UB} = \chi(G)$.
- Senza la regola del colore nuovo, ogni partizione con q classi comparirebbe una volta per ogni modo di dare nomi alle sue classi con i colori $1, \dots, \text{UB} - 1$: l'albero conterrebbe molte copie simmetriche della stessa soluzione.

Senza potatura, l'albero enumera ogni soluzione ammissibile del VCP esattamente una volta, a meno di una ridenominazione dei colori. La potatura per bound elimina solo sottoalberi

con almeno UB colori: l'algoritmo è esatto.

42.2 Un esempio svolto

Esempio 42.1: Branch-and-bound DSATUR sul grafo di Petersen

Consideriamo il grafo di Petersen, l'istanza di riferimento del [Capitolo 4 «Problemi di colorazione»](#), con $n = 10$ vertici, $m = 15$ spigoli e tutti i gradi uguali a 3:

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

L'algoritmo di branch-and-bound DSATUR usa le regole della Sezione 42.1: branching sul vertice più saturo, visita in profondità, figli in ordine crescente di colore.

- **Inizializzazione**

1. L'euristica *DSATUR* (Esempio «*DSATUR* sul grafo di Petersen» del [Capitolo 30](#)) colora i vertici nell'ordine 1, 2, 3, 4, 5, 6, 8, 7, 9, 10 e dà la soluzione incumbent

$$\bar{Q} = \{ \{1, 3, 7\}, \{2, 4, 6, 10\}, \{5, 8, 9\} \}, \quad \text{UB} = 3.$$

2. Tutti i gradi sono uguali, quindi la clique greedy scorre i vertici 1, 2, ..., 10. Il vertice 1 entra, il vertice 2 è adiacente a 1 ed entra, e nessun altro vertice è adiacente a entrambi. Quindi $K = \{1, 2\}$ e $\text{LB} = 2$. Il grafo di Petersen non ha triangoli, quindi $\omega(G) = 2$.
3. Il nodo 1 colora $\text{col}^1(1) = 1$ e $\text{col}^1(2) = 2$, con $q(1) = 2$. Poiché $\text{LB} = 2 < 3 = \text{UB}$, il nodo 1 viene inserito nella lista L :

$$2 \leq \chi(G) \leq 3.$$

- **Iterazione 1**

Il nodo 1 viene estratto da L . I vertici 3, 5, 6, 7 hanno un vicino colorato ciascuno, quindi $\text{sat}_1 = 1$; gli altri hanno $\text{sat}_1 = 0$. I quattro vertici in pareggio hanno $\text{du}_1 = 2$, quindi vince l'indice minimo: il vertice di branching è 3. Il suo vicino 2 ha colore 2, quindi $F_1(3) = \{1\}$.

1. Nodo 2: $\text{col}(3) = 1$, con $q(2) = 2 < \text{UB}$. Il nodo sopravvive.
2. Nodo 3: $\text{col}(3) = 3$, il colore nuovo, con $q(3) = 3 \geq \text{UB}$. Il nodo è potato per bound.

- **Iterazione 2**

Il nodo 2 viene estratto da L , con $Q_1 = \{1, 3\}$ e $Q_2 = \{2\}$. I vertici 4, 5, 6, 7, 8 hanno $\text{sat}_2 = 1$ e $\text{du}_2 = 2$, e 9, 10 hanno $\text{sat}_2 = 0$: il vertice di branching è 4. Il suo vicino 3 ha colore 1, quindi $F_2(4) = \{2\}$.

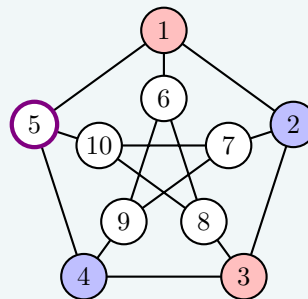
1. Nodo 4: $col(4) = 2$, con $q(4) = 2 < UB$. Il nodo sopravvive.
2. Nodo 5: $col(4) = 3$, con $q(5) = 3 \geq UB$. Il nodo è potato per bound.

- **Iterazione 3**

Il nodo 4 viene estratto da L , con $Q_1 = \{1, 3\}$ e $Q_2 = \{2, 4\}$. Il vertice 5 ha i vicini 1 (colore 1) e 4 (colore 2): $sat_4(5) = 2$, è l'unico vertice con grado di saturazione 2 e quindi è il vertice di branching. I suoi colori ammissibili sono $F_4(5) = \emptyset$.

1. Nodo 6: $col(5) = 3$, l'unico figlio, con $q(6) = 3 \geq UB$. Il nodo è potato per bound.

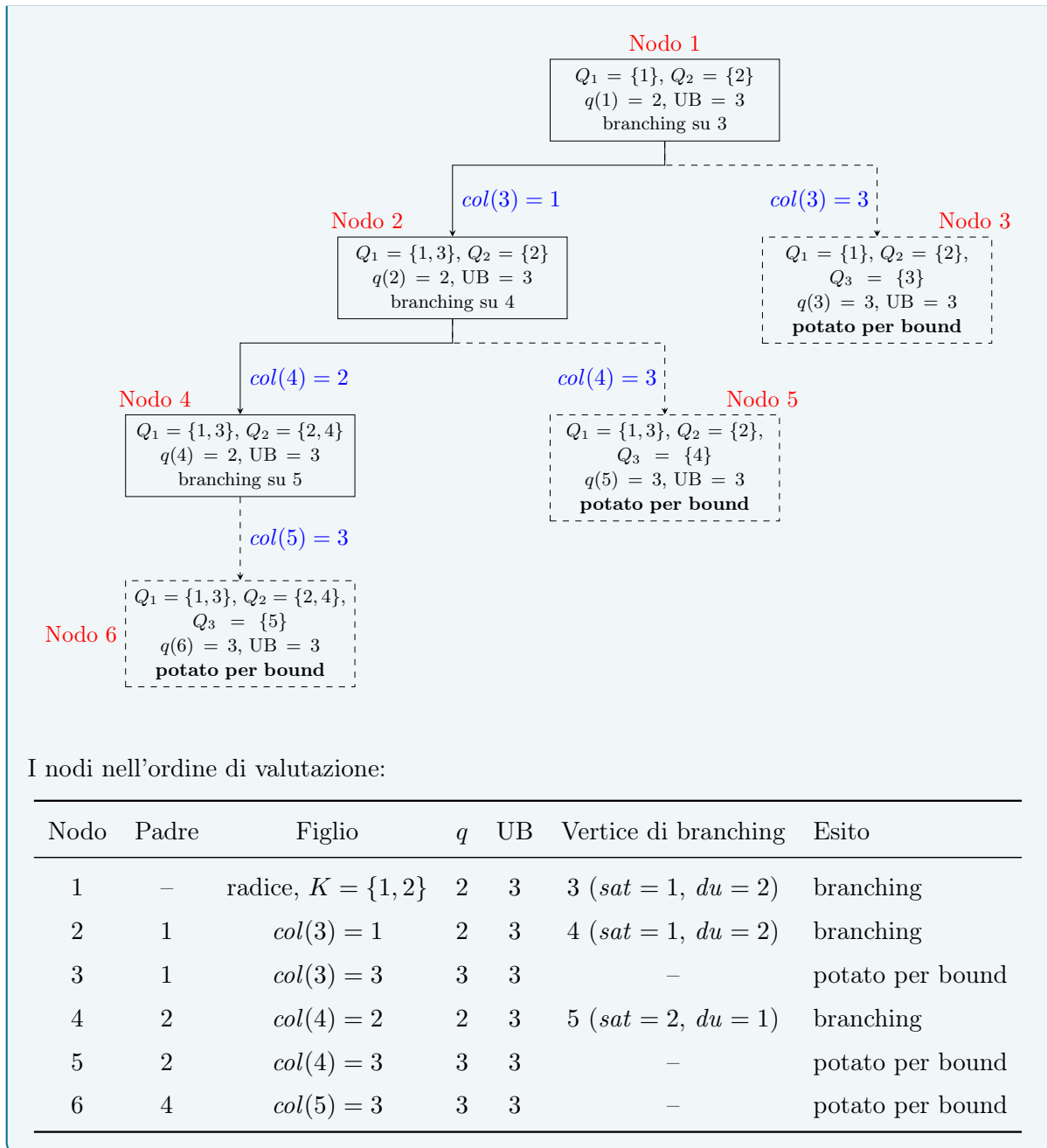
La colorazione parziale del nodo 4 è disegnata qui sotto: il ciclo 1, 2, 3, 4, 5 ha cinque vertici, e due colori non bastano per chiuderlo.



- **Terminazione**

La lista L è vuota: la soluzione incumbent $\bar{Q}$ è ottima e $\chi(G) = UB = 3$. L'albero ha 6 nodi.

Questo è l'albero di branch-and-bound, che riporta per ogni nodo le sue classi di colore, il numero di colori q , l'upper bound UB al momento della valutazione, e il vertice di branching o il criterio di potatura. I nodi potati sono tratteggiati.



- L'algoritmo non migliora la soluzione incumbent: dimostra che $UB = 3$ è ottimo. La dimostrazione è il ciclo dispari 1, 2, 3, 4, 5, trovato dalla ricerca in tre livelli.
- Senza la potatura, l'albero del grafo di Petersen ha 9 776 nodi e 7 430 foglie, una per ogni partizione ammissibile contata nel [Capitolo 4](#), come prevede la [Proposizione 42.2](#).

42.3 Osservazioni

- **Origine.** Il grado di saturazione e l'algoritmo esatto che fa branching su di esso sono dovuti a Brélaz (1979). Pre-colorare una clique è il modo usuale di eliminare la simmetria dei primi colori.
- **Regola di pareggio di Sewell (1996).** Fra i vertici di grado di saturazione massimo, Sewell sceglie il vertice v che massimizza il numero di colori ammissibili in comune con i suoi vicini non colorati, $\sum_{u \in N(v) \cap U^k} |F_k(v) \cap F_k(u)|$. Colorare quel vertice toglie più opzioni al resto del grafo.
- **PASS (San Segundo, 2012).** PASS applica la regola di Sewell solo a vertici selezionati, calcolando il conteggio sui vertici in pareggio per il grado di saturazione massimo, che costa meno. Entrambe le regole cambiano solo la gestione dei pareggi: i figli, i bound e la Proposizione 42.2 restano gli stessi.
- Riferimenti: D. Brélaz, «New methods to color the vertices of a graph», *Communications of the ACM* 22(4), 251–256, 1979; E. C. Sewell, «An improved algorithm for exact graph coloring», in D. S. Johnson, M. A. Trick (a cura di), *Cliques, Coloring, and Satisfiability*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26, 1996; P. San Segundo, «A new DSATUR-based algorithm for exact vertex coloring», *Computers & Operations Research* 39(7), 1724–1733, 2012.

42.4 Errori tipici

Attenzione

- **Contare i vicini colorati invece dei colori.** Il grado di saturazione conta i colori *diversi* attorno a v : due vicini con lo stesso colore contano una volta sola.
- **Un figlio per ogni colore $1, \dots, UB - 1$.** I colori non usati sono tutti equivalenti: solo il primo colore nuovo $q(k) + 1$ dà un figlio, altrimenti l'albero contiene copie simmetriche della stessa colorazione.
- **Dimenticare il colore nuovo.** Il figlio con colore $q(k) + 1$ si crea sempre, anche quando $F_k(v) \neq \emptyset$; senza di esso alcune colorazioni non si raggiungono mai.
- **Potare con $q(k) > UB$.** Un nodo con $q(k) = UB$ non può portare a una colorazione migliore, perché il numero di colori è un intero: la regola è $q(k) \geq UB$.
- **Pre-colorare un insieme che non è una clique.** Solo i vertici di una clique richiedono colori diversi a due a due in ogni colorazione; pre-colorare con colori diversi due vertici non adiacenti può escludere tutte le colorazioni ottime.
- **Continuare dopo $UB = |K|$.** Quando una colorazione completa raggiunge il bound della clique, è ottima: i nodi ancora nella lista non si esplorano.

42.5 In sintesi

In sintesi

- Un nodo del branch-and-bound DSATUR è una colorazione parziale; l'algoritmo fa branching sul vertice più saturo, con un figlio per ogni colore ammissibile e uno per un colore nuovo.
- *DSATUR* dà il primo upper bound, una clique greedy dà il lower bound ed è pre-colorata; un nodo è potato quando $q(k) \geq \text{UB}$.
- Senza potatura l'albero enumera ogni colorazione ammissibile una volta, a meno di una ridenominazione dei colori: l'algoritmo è esatto.

Quadro riassuntivo del grafo di Petersen ($n = 10, m = 15$). I bound su $\chi(G)$ trovati finora:

Fonte	Lower bound	Upper bound	Da
Bound della clique (Capitolo 4)	2	–	$\omega(G) = 2$
<i>DSATUR</i> (Capitolo 30)	–	3	$\{1, 3, 7\}, \{2, 4, 6, 10\}, \{5, 8, 9\}$
Questo capitolo, inizializzazione	2	3	clique greedy $K = \{1, 2\}$
Questo capitolo, terminazione	3	3	lista vuota, 6 nodi