Matheuristics for Combinatorial Optimization problems

PhD in Computer Science

Roberto Cordone

DI - Università degli Studi di Milano

Phone nr.: 02 503 16235

E-mail: roberto.cordone@unimi.it

Web page: https://homes.di.unimi.it/cordone/courses/2022-mh/2022-mh.html

Lesson 5: Heuristic branch-and-bound Milano, A.A. 2021/22

Programma del secondo modulo

Trattiamo euristiche che usano risolutori di Programmazione Matematica, in particolare di Programmazione Lineare Intera (*PLI*)

- Lezione 5 (Martedì 15 Febbraio, 10.30–13.00):
  - Richiami su branch-and-bound e risolutori per la *PLI*
  - Euristiche basate sulla limitazione dell'albero di branching
- Lezione 6 (Venerdì 18 Febbraio, 10.30–13.00):
  - Richiami sul *branch-and-price*
  - Euristiche basate sulla *column generation*
- Lezione 7 (Martedì 22 Febbraio, 10.30–13.00):
  - Euristiche per raffinare il branch-and-bound
  - *Feasibility pump* e *local branching*
- Lezione 8 (Giovedì 24 Febbraio, 10.30–13.00):
  - Richiami di ricerca locale
  - Euristiche basate su interazioni fra risolutori e ricerca locale

Ottimizzazione Combinatoria (OC)

Un problema di Ottimizzazione Combinatoria (OC) è definito da

1. un insieme di soluzioni ammissibili X finito

oppure

2. un insieme di soluzioni ammissibili $X \subseteq 2^B$ con B insieme base finito, cioè le soluzioni ammissibili sono i sottoinsiemi dell'insieme base che soddisfano opportune condizioni

Le due definizioni sono equivalenti:

$2 \Rightarrow 1$: se l'insieme base B è finito, ogni collezione $X \subseteq 2^B$ è finita

$1 \Rightarrow 2$: se l'insieme delle soluzioni è finito, si può chiamarlo B e definire X come la collezione dei singoletti di B (*"soluzione" è un insieme contenente una sola soluzione*)

La definizione sofisticata permette un'analisi più approfondita perché

- definisce X in modo compatto e significativo
- consente di non limitarsi a enumerare X

La definizione dell'insieme base non è in genere univoca

Esempio: Knapsack Problem (KP)

Si vuole scegliere da un insieme di oggetti voluminosi un sottoinsieme di valore massimo che si possa racchiudere in uno zaino di capacità limitata

- un insieme O di oggetti elementari
- una funzione $v : O \to \mathbb{N}$ che descrive il volume di ogni oggetto
- un numero $V \in \mathbb{N}$ che descrive la capacità di uno zaino
- una funzione $\phi : O \to \mathbb{N}$ che descrive il valore di ogni oggetto

Un insieme base può banalmente essere quello degli oggetti: $B = O$

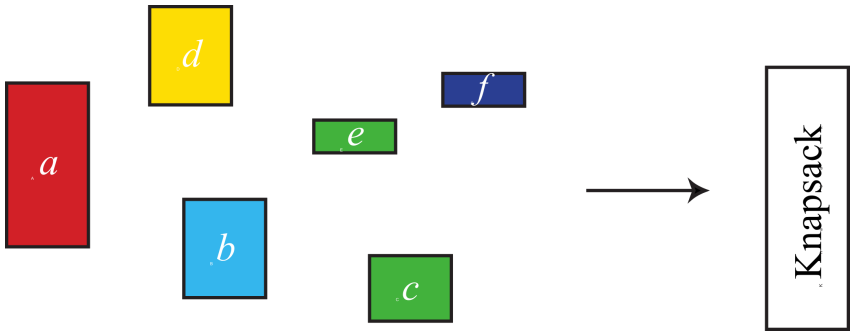
La regione ammissibile contiene i sottoinsiemi di oggetti di volume totale non superiore alla capacità dello zaino

$$X = \left\{ \xi \subseteq B : \sum_{j \in \xi} v_j \leq V \right\}$$

L'obiettivo è massimizzare il valore complessivo degli oggetti scelti

$$\max_{\xi \in X} f(\xi) = \sum_{j \in \xi} \phi_j$$

Esempio

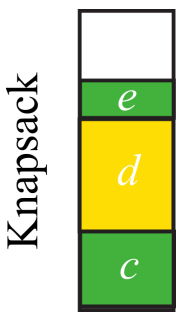


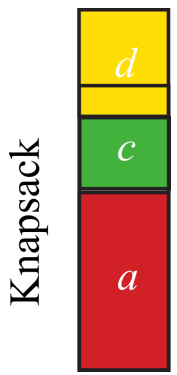
O	a	b	c	d	e	f
ϕ	7	2	4	5	4	1
v	5	3	2	3	1	1

$V = 8$

$$\xi' = \{c, d, e\} \in X \qquad f(\xi') = 13$$

$$\xi'' = \{a, c, d\} \notin X \qquad f(\xi'') = 16$$





Esempio: *Maximum Diversity Problem* (*MDP*)

Si vuole scegliere da un insieme di punti un sottoinsieme di k punti con la massima somma delle distanze reciproche

- un insieme P di punti
- una funzione $d : P \times P \to \mathbb{N}$ che dà la distanza fra coppie di punti
- un numero $k \in \{1, \ldots, |P|\}$ che dà il numero di punti da scegliere

Un insieme base può essere l'insieme dei punti: $B = P$

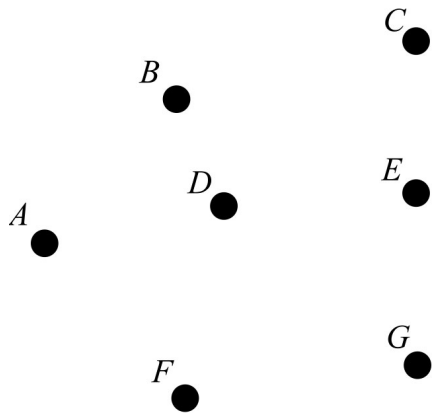
La regione ammissibile contiene i sottoinsiemi di k punti

$$X = \{\xi \subseteq B : |\xi| = k\}$$

L'obiettivo è massimizzare la distanza totale reciproca fra i punti scelti

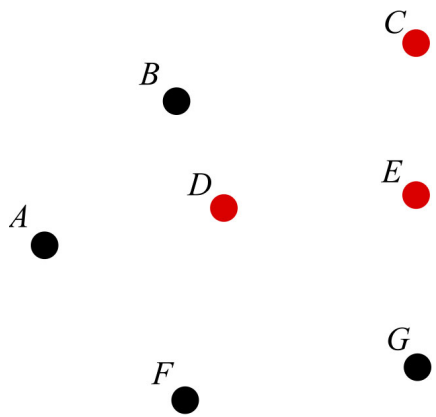
$$\max_{\xi \in X} f(\xi) = \sum_{i,j \in \xi} d_{ij}$$

Esempio

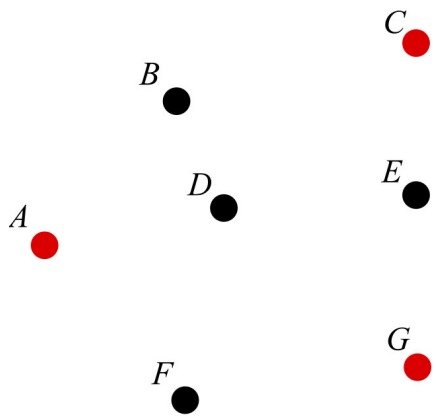


$k = 3$

$\xi' = \{C, D, E\} \in X$
$f(\xi') = 24$



$\xi'' = \{A, C, G\} \in X$
$f(\xi'') = 46$

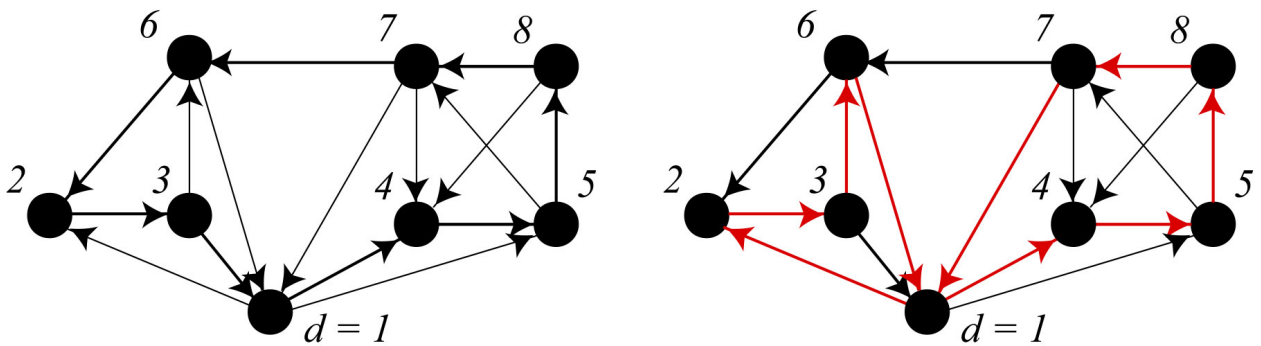


Esempio: Vehicle Routing Problem (*VRP*)

Dati

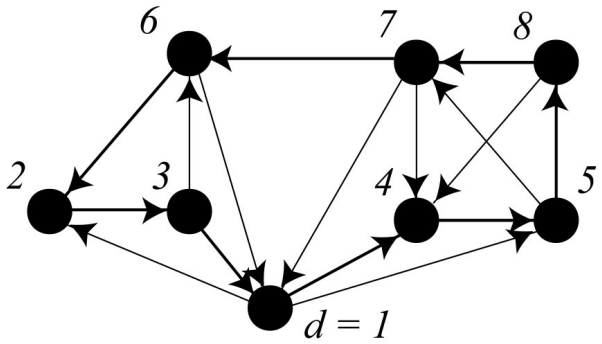
- un grafo orientato $G = (N, A)$ con un nodo deposito $d \in N$
- una funzione $c : A \to \mathbb{N}$ che assegna un costo ad ogni arco
- una funzione $w : N \to \mathbb{N}$ che assegna un peso ad ogni nodo
- un numero $W \in \mathbb{N}$ che descrive una capacità massima

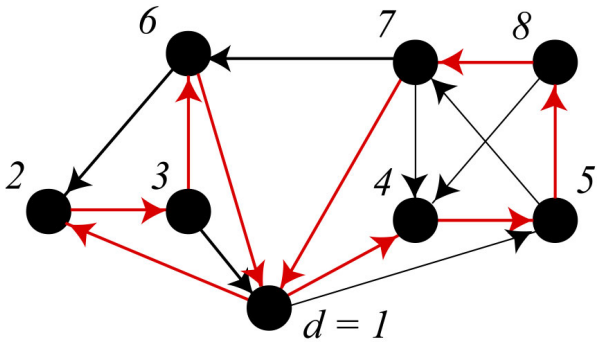
il *VRP* cerca l'insieme di circuiti di costo minimo che toccano tutti i nodi e ciascuno dei quali visita d e ha peso totale $\leq W$



$w = [\,0\,1\,1\,1\,1\,1\,1\,1\,]$ $\qquad$ $W = 4$

Esempio: Vehicle Routing Problem





$\bar{\xi} \in X$

L'insieme base si può definire (per esempio) come

- l'insieme degli archi del grafo: $B = A$

  $$\bar{\xi} = \{(d,2),(2,3),(3,6),(6,d),(d,4),(4,5),(5,8),(8,7),(7,d)\} \subseteq B$$

- l'insieme delle coppie (nodo,veicolo): $B = N \times V$ (Lez. 4 pag. 5)

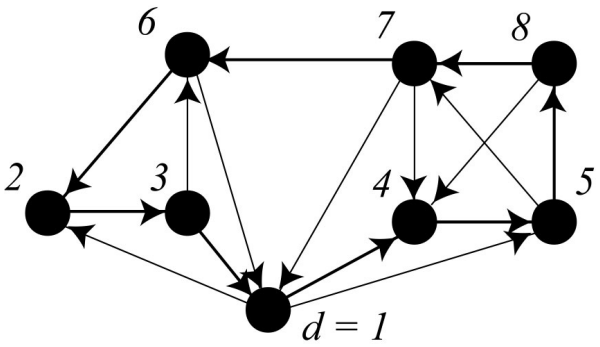
  $$\bar{\xi} = \{(2,V_1),(3,V_1),(6,V_1),(4,V_2),(5,V_2),(7,V_2),(8,V_2)\} \subseteq B$$

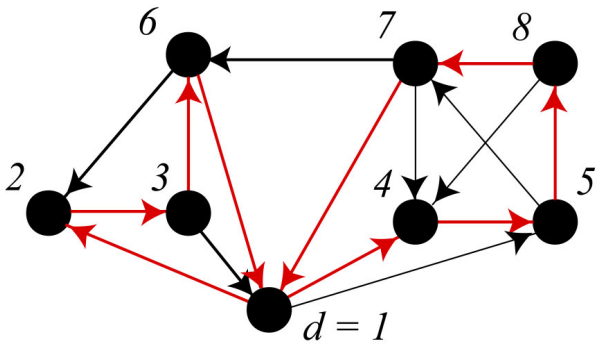
- l'insieme delle coppie (arco,veicolo): $B = A \times V$

  $$\bar{\xi} = \{(d,2,V_1),(2,3,V_1),(3,6,V_1),(6,d,V_1),(d,4,V_2),$$
  $$(4,5,V_2),(5,8,V_2),(8,7,V_2),(7,d,V_2)\} \subseteq B$$

  Sono tutte *definizioni compatte* su diversi insiemi B "piccoli"

Esempio: Vehicle Routing Problem





$\bar{\xi} \in X$

L'insieme base si può definire anche (per esempio) come

- l'insieme dei circuiti che visitano d e hanno peso $\leq W$

$$B = \{(d,2,3,d),(d,2,3,6,d),(d,4,5,8,7,d),(d,5,8,7,d),(d,5,8,7,6,d)$$

$$\bar{\xi} = \{(d,2,3,6,d),(d,4,5,8,7,d)\} \subseteq B$$

(*definizione estesa*)

- l'insieme delle famiglie di circuiti ammissibili

$$B = \{(d,2,3,6,d,4,5,8,7,d),(d,2,3,d,5,8,7,6,d),\ldots\}$$

$$\bar{\xi} = \{(d,2,3,6,d,4,5,8,7,d)\} \subseteq B$$

(*definizione estremamente estesa*)

Il vettore di incidenza

Definito un insieme base B, ogni soluzione $\xi \in X$ è un sottoinsieme di B e si può descrivere associando valori binari x_j agli elementi di B

$$x_j = \begin{cases} 1 \text{ indica che } j \in \xi \\ 0 \text{ indica che } j \in B \setminus \xi \end{cases}$$

Ad ogni soluzione $\xi \in X$ corrisponde un vettore di incidenza $x \in \mathbb{B}^{|B|}$

Per esempio, un vettore binario potrebbe corrispondere

- nel *KP* a un insieme di oggetti
- nel *MDP* a un insieme di punti (o di coppie di punti)
- nel *VRP* a un insieme di archi, o di coppie (nodo, veicolo), o di coppie (arco, veicolo), o di circuiti che visitano d e hanno peso $\leq W$

In genere esitono molte definizioni alternative

Formulazioni di problemi di *OC*

Il vettore di incidenza trasforma problemi su sottoinsiemi
in problemi su vettori numerici (Programmazione Matematica)

$$\begin{array}{l} \min\ \phi(\xi) \\ \xi \in X \end{array} \longrightarrow \begin{array}{rll} \min & f(x) & \\ & g_i(x) \leq 0 & i = 1, \ldots, m \\ & x_j \in \mathbb{B} & j = 1, \ldots, n \end{array}$$

dove

- $x \in \mathbb{B}^n$, cioè una soluzione è un vettore di n variabili binarie
- $\{x \in \mathbb{B}^n : g_i(x) \leq 0, i = 1, \ldots, m\}$, cioè la regione ammissibile è l'insieme dei vettori che soddisfano le disuguaglianze (vincoli)

(*per semplicità continuiamo a indicarla con* X)

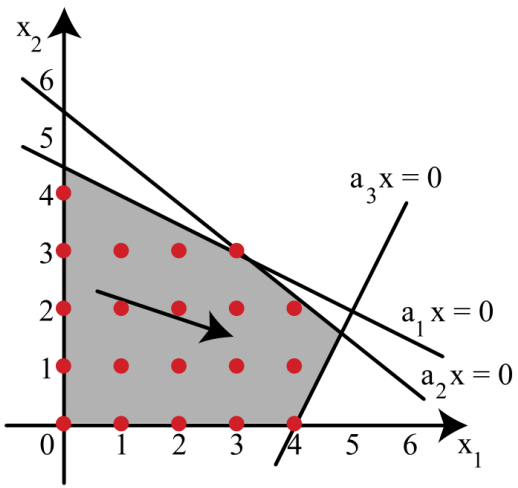
Le matheuristic sfruttano le informazioni derivate dalla formulazione, cioè le proprietà delle funzioni f e g_i $(i = 1, \ldots, m)$

Ogni formulazione genererà euristiche diverse

Programmazione Lineare Intera (*PLI*)

I problemi di Programmazione Lineare Intera (*PLI*) hanno

- funzione obiettivo affine
- vincoli di (dis)uguaglianza affini
- variabili di decisione intere



$$\begin{aligned} \min f(x) &= \sum_{j \in B} c_j x_j + d \quad (\Pi) \\ \sum_{j \in B} a_{ij} x_j &\leq b_i \quad i = 1, \ldots, m \\ x_j &\in \mathbb{Z} \quad i = 1, \ldots, m \end{aligned}$$

che si può riassumere in forma matriciale

$$\begin{aligned} \min f &= c^T x + d \quad (\Pi) \\ Ax &\leq b \\ x &\in \mathbb{Z}^n \end{aligned}$$

Sono a tutti gli effetti problemi non lineari: $x \in \mathbb{Z}^n \Leftrightarrow \sum_{j=1}^{n} \sin^2(\pi x_j) = 0$

Relazione fra *OC* e *PLI*

Tutti i problemi di *OC* possono essere formulati come problemi di *PLI*

- le variabili sono intere (binarie) grazie al vettore di incidenza
- obiettivo e vincoli possono essere resi affini con linearizzazioni

La cosa è tutt'altro che indolore, perché in generale

- i vincoli linearizzati sono più numerosi di quelli originali
- i vincoli linearizzati sono più deboli di quelli originali, cioè ammettono soluzioni frazionarie che violano i vincoli originali e quindi il loro rilassamento continuo è peggiore

ma qui vogliamo solo indicare che è possibile in linea di principio

Al contrario i problemi di *PLI* sono un insieme più esteso di quelli di *OC*
(per esempio, possono avere infinite soluzioni)

Appendice: traduzione da *OC* e *PLI*

Tutti i problemi di *OC* possono essere formulati come problemi di *PLI*

Infatti, per ogni coppia di vettori $x, \bar{x} \in \mathbb{B}^n$

$$\prod_{j \in B} [\bar{x}_j \, x_j + (1 - \bar{x}_j) \, (1 - x_j)] = \begin{cases} 1 & \text{per } x = \bar{x} \\ 0 & \text{per } x \neq \bar{x} \end{cases}$$

Questo permette di vietare uno per uno i vettori $\bar{x} \notin X$ con i vincoli

$$\prod_{j \in B} (\bar{x}_j \, x_j + (1 - \bar{x}_j) \, (1 - x_j)) = 0$$

che si linearizzano facilmente in:

$$\sum_{j \in B} [\bar{x}_j \, x_j + (1 - \bar{x}_j) \, (1 - x_j)] \leq n - 1$$

Per la funzione obiettivo si può porre $f(x) = \sum_{x \in \mathbb{B}^n} c_x y_x$, dove
c_x è il costo di ogni vettore $x \in \mathbb{B}^n$ e y_x una variabile binaria associata ad esso, legata alle variabili originali x_j dai vincoli

$$y_{\bar{x}} = \prod_{j \in B} (\bar{x}_j x_j + (1 - \bar{x}_j)(1 - x_j))$$

che vanno linearizzati uno per uno. Ovviamente, in generale non è pratico.

Risolutori di *PLI*

Esistono risolutori di problemi di *PLI*, sia commerciali

- GUROBI (`https://www.gurobi.com/`)
- CPLEX (`https://www.ibm.com/analytics/cplex-optimizers`)
- MOSEK, MINTO, XPRESS...

sia gratuiti

- GLPK (`https://www.gnu.org/software/glpk/`)
- COIN-OR SYMPHONY (`https://www.coin-or.org/`)
- SCIP (`https://scip.zib.de/`)

ai quali bisogna passare i dati con opportune strutture in memoria

Linguaggi e software di modellazione

Esistono

- linguaggi di modellazione per descrivere ad alto livello i problemi (usando famiglie di variabili e vincoli anziché singoli elementi)
- modellatori per caricare in memoria problemi e istanze e risolverle

fra i quali

- OPL-STUDIO (incluso in CPLEX)
- AMPL (`https://ampl.com/`), parzialmente simulato in GLPK
- MPL, LINDO, MOSEL, GAMS...

Vediamo qualche esempio di strumento all'opera

Complessità della *PLI*

Che cosa fanno concretamente i risolutori?

Dato un problema generico Π di *PLI* non esistono algoritmi polinomiali

1. né per decidere se Π ha soluzione
2. né per decidere se Π ha soluzione di valore $f(x) \leq \bar{f}$
   e quindi risolverlo (sfruttando la ricerca dicotomica)

(*in effetti è lo stesso problema*)

Sono entrambi problemi $\mathcal{NP}$-completi

. . . che fare?

Enumerazione esplicita

Un'idea (disperata) è l'enumerazione esplicita delle soluzioni

Sia P il poliedro definito dai vincoli

$$P = \{x \in \mathbb{R}^n : Ax \leq b\}$$

Se P è limitato, la regione ammissibile $X = P \cap \mathbb{Z}^n$ è finita, e quindi ciascuna variabile è limitata

$$\exists l_j, u_j \in \mathbb{R} : \lceil l_j \rceil \leq x_j \leq \lfloor u_j \rfloor \text{ per ogni } x \in X$$

Nei problemi di OC è banalmente $0 \leq x_j \leq 1$

Si può quindi

- generare uno ad uno i vettori x con $x_j = \lceil l_j \rceil, \ldots, \lfloor u_j \rfloor$
- verificare l'ammissibilità del vettore corrente ($Ax \leq b$?)
- se ammissibile, confrontare il suo valore col miglior valore noto ($c^T x < c^T \tilde{x}$?) e, nel caso, aggiornare la miglior soluzione nota $\tilde{x}$

Considerando un numero di vettori esponenziale, il metodo non è pratico

Divide et impera

Se si partiziona la regione ammissibile in sottoinsiemi disgiunti

$$X = \bigcup_{i=1}^{r} X_i = X_1 \cup \ldots \cup X_r \quad \text{con } X_j \cap X_k = \emptyset \; \forall j \neq k$$

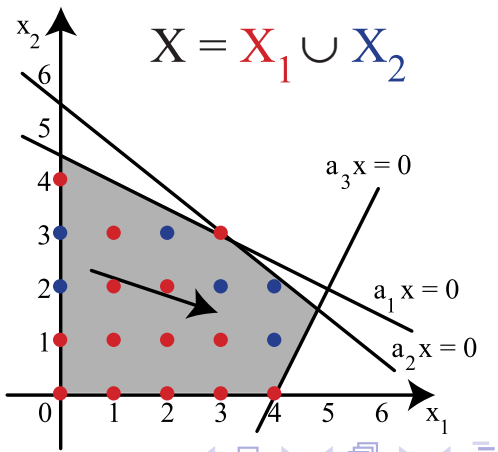
la soluzione ottima è la migliore fra quelle dei singoli sottoinsiemi

$$x^* = \arg\min_{x \in X} f(x) = \arg \min_{i=1,\ldots,r} \left[\min_{x \in X_i} f(x) \right]$$

cioè si può risolvere Π risolvendo invece r problemi più piccoli

$$\begin{aligned} \min f &= cx + d \qquad (\Pi_i) \\ x &\in X_i \end{aligned}$$

$$\begin{aligned} \min f = -3x_1 + x_2 & \qquad (\Pi) \\ x_1 + x_2 &\leq 9 \\ 6x_1 + 7x_2 &\leq 39 \\ 2x_1 - x_2 &\leq 8 \\ x &\in \mathbb{N}^n \end{aligned}$$



Un approccio ricorsivo

Perché è meglio risolvere Π_i per $i = 1, \ldots, r$ piuttosto che Π?

- perché a volte Π_i è facile (casi base)
  - $X_i = \emptyset$
  - $X_i = \{x_i^*\}$, per cui $\arg\min_{x \in X_i} f(x) = x_i^*$
  - X_i ammette un algoritmo polinomiale per risolvere Π_i e trovare x_i^*
- perché si può ripetere il procedimento sui Π_i (caso ricorsivo)

Per procedere ricorsivamente, bisogna che ogni Π_i sia un problemi di PLI

$\Rightarrow$ si divide X aggiungendo vincoli lineari disgiuntivi alla formulazione

$$X_i = X \cap \left\{ x \in \mathbb{R}^n : A^{(i)} x \leq b^{(i)} \right\} = P \cap \mathbb{Z}^n \cap \left\{ x \in \mathbb{R}^n : A^{(i)} x \leq b^{(i)} \right\}$$

Regola di branching è il criterio di costruzione dei vincoli $a_s^{(i)} x \leq b_s^{(i)}$ dove $a_s^{(i)}$ è la riga s della matrice $A^{(i)}$

Regole di branching per la PLI

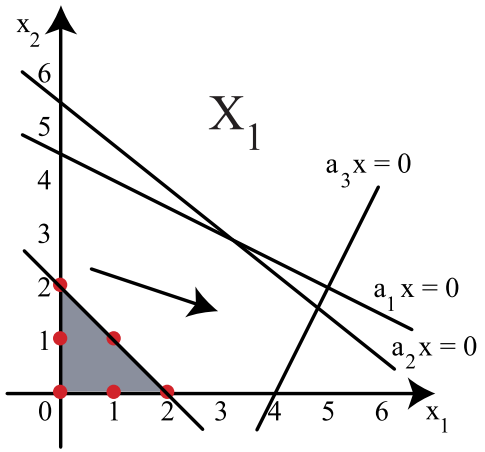
Si possono usare vincoli paralleli fissando $\begin{cases} a \in \mathbb{Z}^n \\ b_s \in \mathbb{Z} \text{ per } s = 1, \ldots, r-1 \end{cases}$

(*per esempio* $a^T = [1\ 1]$, $r = 3$, $b_1 = 2$ *e* $b_2 = 4$)

e poi creando i sottoproblemi con l'aggiunta dei vincoli:

1. $a^T x \leq b_1$ in Π_1 (ad es., $x_1 + x_2 \leq 2$)
2. $b_1 + 1 \leq a^T x \leq b_2$ in Π_2 (ad es., $3 \leq x_1 + x_2 \leq 4$)
3. ...
4. $b_{r-1} + 1 \leq a^T x$ in Π_r (ad es., $x_1 + x_2 \geq 5$)

$$
\begin{array}{rcll}
\min f & = & -3x_1 + x_2 & (\Pi) \\
x_1 + x_2 & \leq & 9 & \\
6x_1 + 7x_2 & \leq & 39 & \\
2x_1 - x_2 & \leq & 8 & \\
x_1 + x_2 & \leq & 2 & \\
x & \in & \mathbb{N}^n &
\end{array}
$$



Regole di branching per la PLI

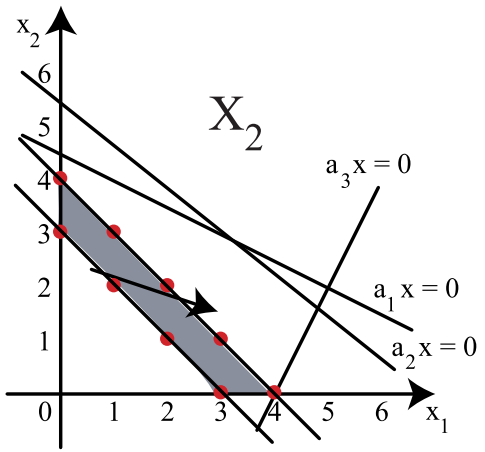
Si possono usare vincoli paralleli fissando $\begin{cases} a \in \mathbb{Z}^n \\ b_i \in \mathbb{Z} \text{ per } i = 1, \ldots, r-1 \end{cases}$

(*per esempio* $a^T = [1\ 1]$, $r = 3$, $b_1 = 2$ *e* $b_2 = 4$)

e poi creando i sottoproblemi con l'aggiunta dei vincoli:

1. $ax \leq b_1$ in Π_1 (ad es., $x_1 + x_2 \leq 2$)
2. $b_1 + 1 \leq ax \leq b_2$ in Π_2 (ad es., $3 \leq x_1 + x_2 \leq 4$)
3. ...
4. $b_{r-1} + 1 \leq ax$ in Π_r (ad es., $x_1 + x_2 \geq 5$)

$$
\begin{array}{rcll}
\min f & = & -3x_1 + x_2 & (\Pi) \\
x_1 + x_2 & \leq & 9 & \\
6x_1 + 7x_2 & \leq & 39 & \\
2x_1 - x_2 & \leq & 8 & \\
3 \leq x_1 + x_2 & \leq & 4 & \\
x & \in & \mathbb{N}^n &
\end{array}
$$



Regole di branching per la PLI

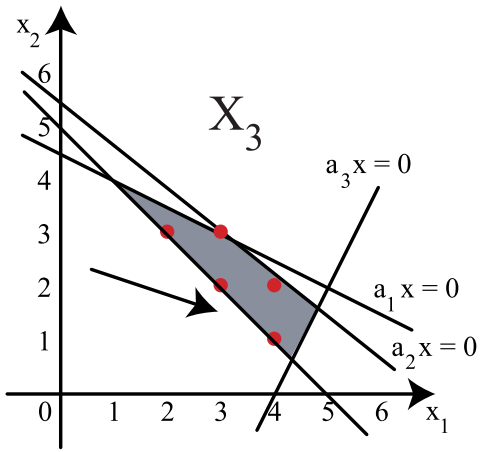
Si possono usare vincoli paralleli fissando $\begin{cases} a \in \mathbb{Z}^n \\ b_i \in \mathbb{Z} \text{ per } i = 1, \ldots, r-1 \end{cases}$

(*per esempio* $a^T = [1\ 1]$, $r = 3$, $b_1 = 2$ *e* $b_2 = 4$)

e poi creando i sottoproblemi con l'aggiunta dei vincoli:

1. $ax \leq b_1$ in Π_1 (ad es., $x_1 + x_2 \leq 2$)
2. $b_1 + 1 \leq ax \leq b_2$ in Π_2 (ad es., $3 \leq x_1 + x_2 \leq 4$)
3. ...
4. $b_{r-1} + 1 \leq ax$ in Π_r (ad es., $x_1 + x_2 \geq 5$)

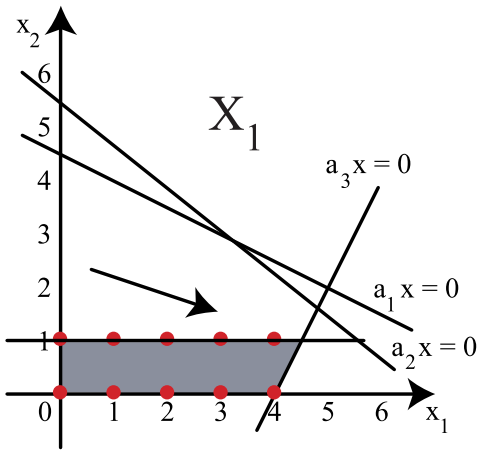
$$
\begin{array}{rcll}
\min f & = & -3x_1 + x_2 & (\Pi) \\
x_1 + x_2 & \leq & 9 & \\
6x_1 + 7x_2 & \leq & 39 & \\
2x_1 - x_2 & \leq & 8 & \\
x_1 + x_2 & \geq & 4 & \\
x & \in & \mathbb{N}^n &
\end{array}
$$

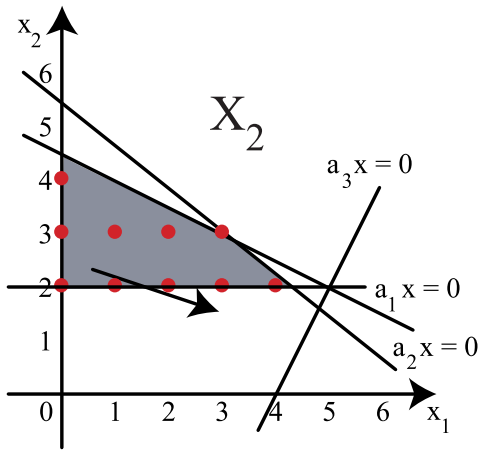


La regola principe

La regola di branching "parallela" più semplice e comune consiste nello scegliere una variabile di branching x_{j^*} e imporre

- $x_{j^*} \leq b$ in Π_1
- $x_{j^*} \geq b + 1$ in Π_2





Nel caso di variabili binarie, questo significa

1. scegliere una variabile di branching x_{j^*}
2. imporre
   - $x_{j^*} = 0$ in Π_1
   - $x_{j^*} = 1$ in Π_2

Regole gerarchiche

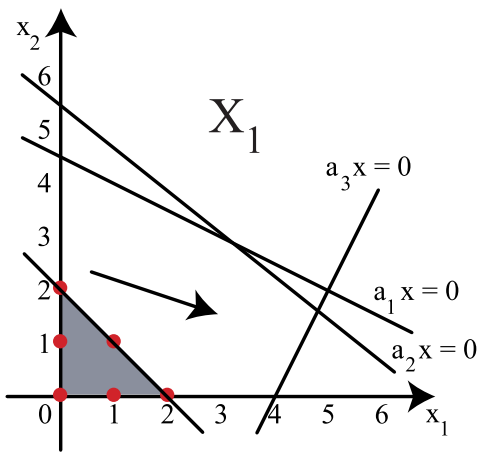
Si possono usare vincoli gerarchici fissando $\begin{cases} a_i \in \mathbb{Z}^n \text{ per } i = 1, \ldots, r-1 \\ b_i \in \mathbb{Z} \text{ per } i = 1, \ldots, r-1 \end{cases}$

$$(\textit{per esempio } r = 3,\ a_1^T = [1\ 1],\ b_1 = 2 \textit{ e } a_1^T = [-1\ 1],\ b_2 = 0)$$

e poi creando i sottoproblemi con l'aggiunta dei vincoli:

1. $a^{(1)}x \leq b^{(1)}$ in Π_1 $\quad (x_1 + x_2 \leq 2)$
2. $a^{(1)}x \geq b^{(1)} + 1$ e $a^{(2)}x \leq b^{(2)}$ in Π_2 $\quad (x_1 + x_2 \geq 3,\ -x_1 + x_2 \leq 0)$
3. $\ldots$
4. $a^{(1)}x \geq b^{(1)} + 1, \ldots,$ e $a^{(r-1)}x \geq b^{(r-1)}$ in Π_r $(x_1 + x_2 \geq 3,\ -x_1 + x_2 \geq 1)$

$$\begin{array}{rcll} \min f & = & -3x_1 + x_2 & (\Pi) \\ x_1 + x_2 & \leq & 9 & \\ 6x_1 + 7x_2 & \leq & 39 & \\ 2x_1 - x_2 & \leq & 8 & \\ x_1 + x_2 & \leq & 2 & \\ x & \in & \mathbb{N}^n & \end{array}$$



Regole gerarchiche

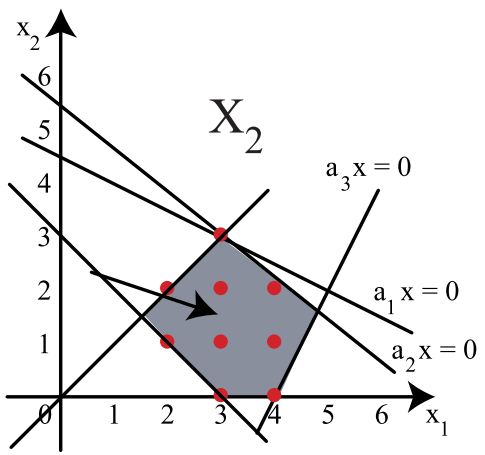
Si possono usare vincoli gerarchici fissando $\begin{cases} a_i \in \mathbb{Z}^n \text{ per } i = 1, \ldots, r-1 \\ b_i \in \mathbb{Z} \text{ per } i = 1, \ldots, r-1 \end{cases}$

$$(\textit{per esempio } r = 3,\ a_1^T = [1\ 1],\ b_1 = 2 \textit{ e } a_1^T = [-1\ 1],\ b_2 = 0)$$

e poi creando i sottoproblemi con l'aggiunta dei vincoli:

1. $a^{(1)}x \leq b^{(1)}$ in Π_1 $\quad (x_1 + x_2 \leq 2)$
2. $a^{(1)}x \geq b^{(1)} + 1$ e $a^{(2)}x \leq b^{(2)}$ in Π_2 $\quad (x_1 + x_2 \geq 3,\ -x_1 + x_2 \leq 0)$
3. ...
4. $a^{(1)}x \geq b^{(1)} + 1, \ldots,$ e $a^{(r-1)}x \geq b^{(r-1)}$ in Π_r $\quad (x_1 + x_2 \geq 3,\ -x_1 + x_2 \geq 1)$

$$\begin{array}{rcll} \min f & = & -3x_1 + x_2 & (\Pi) \\ x_1 + x_2 & \leq & 9 & \\ 6x_1 + 7x_2 & \leq & 39 & \\ 2x_1 - x_2 & \leq & 8 & \\ x_1 + x_2 & \geq & 3 & \\ -x_1 + x_2 & \leq & 0 & \\ x & \in & \mathbb{N}^n & \end{array}$$



Regole gerarchiche

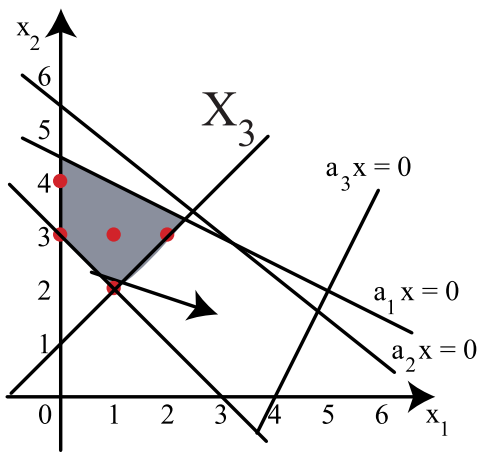
Si possono usare vincoli gerarchici fissando $\begin{cases} a_i \in \mathbb{Z}^n \text{ per } i = 1, \ldots, r-1 \\ b_i \in \mathbb{Z} \text{ per } i = 1, \ldots, r-1 \end{cases}$

$$(\textit{per esempio } r = 3,\ a_1^T = [1\ 1],\ b_1 = 2 \textit{ e } a_1^T = [-1\ 1],\ b_2 = 0)$$

e poi creando i sottoproblemi con l'aggiunta dei vincoli:

1. $a^{(1)}x \leq b^{(1)}$ in Π_1 $\quad (x_1 + x_2 \leq 2)$
2. $a^{(1)}x \geq b^{(1)} + 1$ e $a^{(2)}x \leq b^{(2)}$ in Π_2 $\quad (x_1 + x_2 \geq 3,\ -x_1 + x_2 \leq 0)$
3. $\ldots$
4. $a^{(1)}x \geq b^{(1)} + 1, \ldots,$ e $a^{(r-1)}x \geq b^{(r-1)}$ in Π_r $(x_1 + x_2 \geq 3,\ -x_1 + x_2 \geq 1)$

$$\begin{array}{rcll} \min f & = & -3x_1 + x_2 & (\Pi) \\ x_1 + x_2 & \leq & 9 & \\ 6x_1 + 7x_2 & \leq & 39 & \\ 2x_1 - x_2 & \leq & 8 & \\ x_1 + x_2 & \geq & 3 & \\ -x_1 + x_2 & \geq & 1 & \\ x & \in & \mathbb{N}^n & \end{array}$$



Regole gerarchiche

La regola di branching "gerarchica" più semplice e comune consiste nello scegliere una sequenza di variabili di branching $x_{j_1^*}, \ldots, x_{j_{r-1}^*}$ e imporre

- $x_{j_1^*} \leq b_1$ in Π_1
- $x_{j_1^*} \geq b_1 + 1$ e $x_{j_2^*} \leq b_2$ in Π_2
- $\ldots$
- $x_{j_1^*} \geq b_1 + 1$, $x_{j_2^*} \geq b_2 + 1$, $\ldots$ e $x_{j_r^*} \leq b_{r-1}$ in Π_{r-1}
- $x_{j_1^*} \geq b_1 + 1$, $x_{j_2^*} \geq b_2 + 1$, $\ldots$ e $x_{j_{r-1}^*} \geq b_{r-1} + 1$ in Π_r

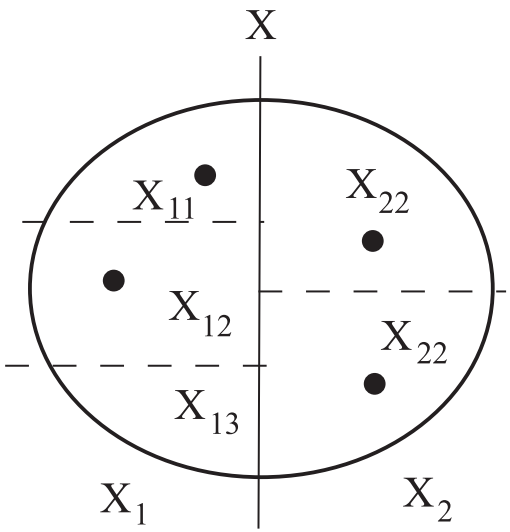
Nel caso di variabili binarie, questo significa

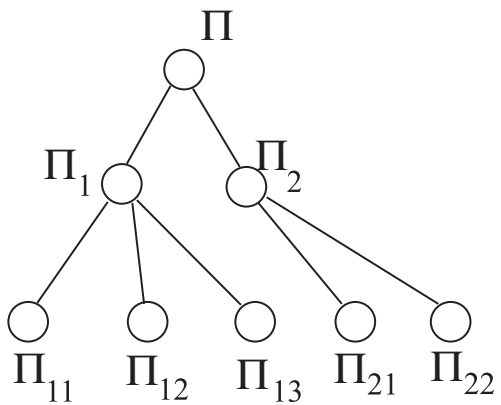
1. scegliere una sequenza di variabili di branching $x_{j_1^*}, \ldots, x_{j_{r-1}^*}$
2. imporre
   - $x_{j_1^*} = 0$ in Π_1
   - $x_{j_1^*} = 1$ e $x_{j_2^*} = 0$ in Π_2
   - $\ldots$
   - $x_{j_1^*} = 1$, $x_{j_2^*} = 1$, $\ldots$ e $x_{j_r^*} = 0$ in Π_{r-1}
   - $x_{j_1^*} = 1$, $x_{j_2^*} = 1$, $\ldots$ e $x_{j_{r-1}^*} = 1$ in Π_r

Albero di branching

Il risultato del procedimento è una struttura gerarchica ad albero con

- nodi associati a problemi, cioè sottoinsiemi di X
  - la radice è associata a Π, cioè X
  - l'insieme associato a un nodo è l'unione di quelli associati ai nodi figli
  - le foglie sono associate a problemi senza soluzioni o a problemi polinomialmente risolubili
- archi associati a operazioni di branching





Le foglie sono ancora più numerose delle soluzioni
Ed è anche $\mathcal{NP}$-completo decidere se un dato nodo è una foglia

Dove sta il vantaggio?

Enumerazione implicita tramite rilassamenti

Enumerazione implicita: esamina in blocco X_i per dimostrare

1. che non ha soluzioni ammissibili
2. che non ha soluzioni ottime per Π

Solo se non è possibile, si procede a scomporre Π_i in sottoproblemi

Si tratta di costruire un rilassamento R_i di Π_i

$$(\Pi_i) \begin{cases} \min f(x) \\ x \in X_i \end{cases} \qquad (R_i) \begin{cases} \min f_{R_i}(x) \\ x \in X_{R_i} \end{cases} \text{ con } \begin{cases} X_{R_i} \supseteq X_i \\ f_{R_i}(x) \leq f(x), \forall x \in X_i \end{cases}$$

da cui deriva necessariamente che

$$X_{R_i} = \emptyset \Rightarrow X_i = \emptyset$$

$$\min_{x \in X_{R_i}} f_{R_i}(x) = f^*_{R_i} \leq f^*_{\Pi_i} = \min_{x \in X_i} f(x)$$

Ovviamente, si sceglie un rilassamento R_i abbastanza facile da risolvere

Informazioni derivate dal rilassamento

Un rilassamento fornisce informazioni utili a limitare l'albero di branching

1. se R_i è inammissibile, anche Π_i lo è

$$X_i^R = \emptyset \Rightarrow X_i = \emptyset$$

2. se la soluzione ottima di R_i è ammissibile per Π_i e l'obiettivo è lo stesso, la soluzione è ottima anche per Π_i

$$x_{R_i}^* \in X_i \Rightarrow x_{R_i}^* = \arg\min_{x \in X_i} f(x)$$

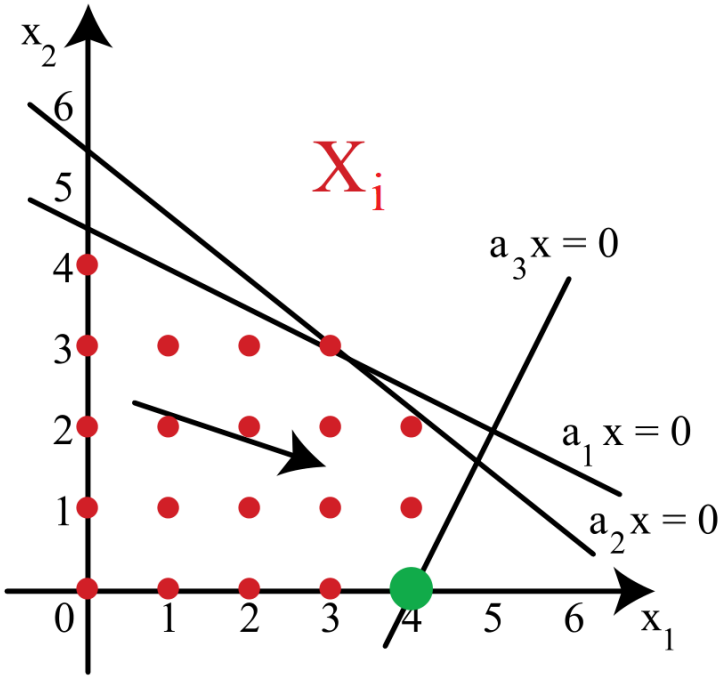
3. se la soluzione ottima di R_i non è migliore di una soluzione ammissibile $\bar{x} \in X$, allora anche la soluzione ottima di Π_i non lo è

$$\exists \bar{x} \in X : f_{R_i}^* \geq f(\bar{x}) \Rightarrow f_{\Pi_i}^* \geq f(\bar{x})$$

In tutti e tre i casi, si può chiudere il nodo Π_i senza ulteriori analisi

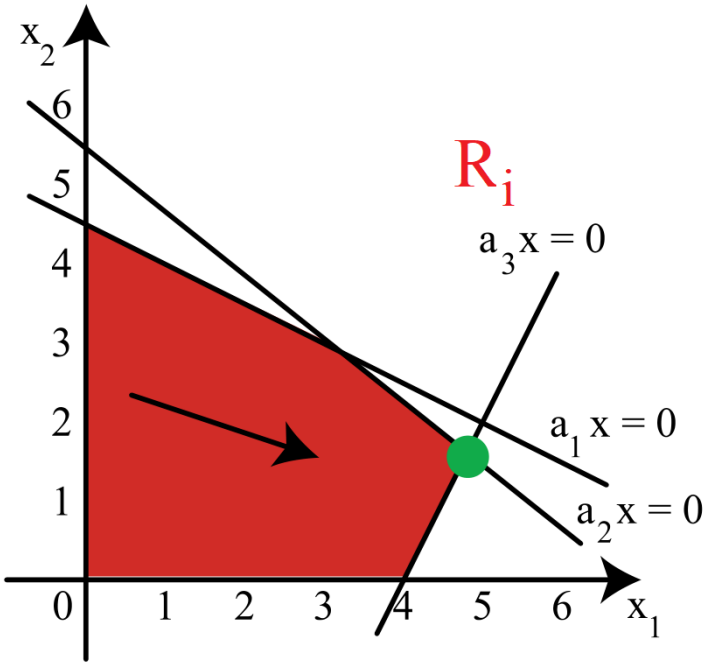
Rilassamento continuo

$$\min f(x) = c^T x + d \qquad (\Pi_i)$$
$$Ax \leq b$$
$$\cancel{x \in \mathbb{N}^n}$$



$$x^*_{\Pi_i} = (4, 0) \Rightarrow f^*_{\Pi_i} = -12$$

$$\min f(x) = c^T x + d \qquad (R_i)$$
$$Ax \leq b$$



$$x^*_{R_i} = (4.75, 1.5) \Rightarrow f^*_{R_i} = -12.75$$

Ma esistono anche rilassamenti lagrangiani, surrogati, duali, . . .

Rafforzare la formulazione

Per limitare l'albero di branching occorrono quindi

1. un rilassamento molto forte: $f^*_{R_i} \approx f^*_{\Pi_i}$
   (*di questo si è parlato in altre lezioni*)
2. una soluzione euristica molto buona: $f(\bar{x}) \approx f^*_{\Pi}$
   a. eseguendo un buon algoritmo euristico prima del branch-and-bound
   b. conservando le soluzioni via via trovate nei sottoproblemi
   c. eseguendo un buon algoritmo euristico ad ogni nodo
      in modo da usare informazioni prodotte dal rilassamento
      (arrotondamento, euristiche lagrangiane, ...)

ma anche

3. una formulazione molto stretta
   - sia al nodo radice
   - sia nei nodi inferiori (i vincoli iniziali e di branching interagiscono)

   dato che essa fornisce bound migliori e soluzioni euristiche migliori

Presolve

Con presolve o preprocessing si intendono le operazioni per rafforzare la formulazione prima di risolvere il rilassamento e di eseguire le euristiche

- dimostrare l'inammissibilità del problema
- eliminare vincoli ridondanti
- restringere l'intervallo di definizione delle variabili, possibilmente sino a fissarne il valore
- aggiungere nuovi vincoli

Inammissibilità e ridondanza

Dato un vincolo affine $a^T x \leq b$, sia J l'insieme degli indici delle variabili e

$$J^+ = \{j \in J : a_j > 0\}, J^- = \{j \in J : a_j < 0\} \text{ e } J^0 = \{j \in J : a_j = 0\}$$

Si può scrivere il vincolo come

$$a^T x = \sum_{j \in J^+} a_j x_j + \sum_{j \in J^-} a_j x_j \leq b$$

Supponiamo che le variabili siano limitate: $l_j \leq x_j \leq u_j$ per ogni $j \in J$

Valgono le seguenti proprietà

- se $\sum_{j \in J^+} a_j l_j + \sum_{j \in J^-} a_j u_j > b$, il problema è inammissibile
- se $\sum_{j \in J^+} a_j u_j + \sum_{j \in J^-} a_j l_j \leq b$, il vincolo è ridondante

Restrizione degli intervalli di definizione

Portiamo da sinistra a destra un termine

$$\sum_{j \in J^+} a_j x_j + \sum_{j \in J^-} a_j x_j - a_k x_k \leq b - a_k x_k$$

e stimiamo il membro di sinistra per difetto in X

$$\underline{a}_k \leq \min_{x \in X} \left(\sum_{j \in J^+} a_j x_j + \sum_{j \in J^-} a_j x_j - a_k x_k \right)$$

da cui $\underline{a}_k \leq b - a_k x_k$ che può stringere l'intervallo di definizione di x_k

$$x_k \leq \left\lfloor \frac{b - \underline{a}_k}{a_k} \right\rfloor \quad \text{per ogni } k \in J^+$$

$$x_k \geq \left\lceil \frac{b - \underline{a}_k}{a_k} \right\rceil \quad \text{per ogni } k \in J^-$$

Una stima molto veloce potrebbe essere

$$\underline{a}_k = \sum_{j \in J^+ \setminus \{k\}} a_j l_j + \sum_{j \in J^- \setminus \{k\}} a_j u_j$$

Aggiunta di vincoli logici

Ad ogni formulazione non esatta è possibile aggiungere vincoli che sono

- rispettati da tutte le soluzioni ammissibili
- violati da alcune soluzioni dei rilassamenti

La formulazione aumentata risulta più forte

Consideriamo vincoli di natura logica (*ce ne sono di molti altri tipi*)

Per ogni variabile binaria x_j

1. si prova a fissare $x_j = \bar{x}_j$ (*come per fare branching*)
2. si esegue il presolve, ma non il rilassamento
   - se il sottoproblema è inammissibile, si fissa $x_j = 1 - \bar{x}_j$
   - se un'altra variabile binaria x_k è forzata al valore $\bar{x}_k$, si aggiunge un vincolo in base alla seguente tabella

$\bar{x}_j$	$\bar{x}_k$	Vincolo
0	0	$x_j - x_k \geq 0$
0	1	$x_j + x_k \geq 1$
1	0	$x_j + x_k \leq 1$
1	1	$x_j - x_k \leq 0$

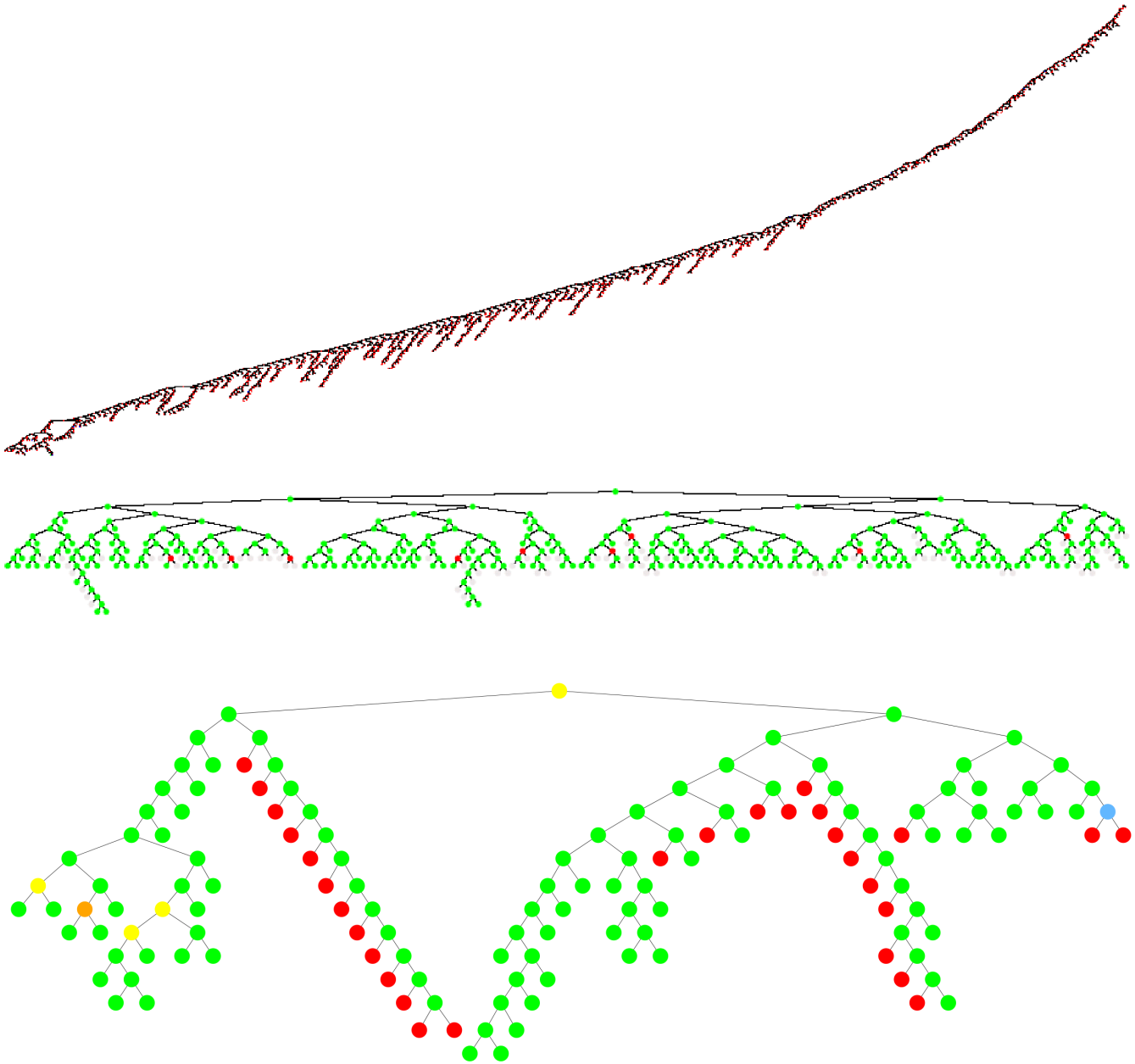
Visita dell'albero

Fra i nodi generati bisogna scegliere quale processare volta per volta

Strategia di visita è il criterio di scelta del primo nodo da processare

- depth-first (in profondità): sceglie l'ultimo nodo generato
  - tiene pochi nodi aperti (risparmia memoria)
  - scende subito ai livelli inferiori
    (fornisce rapidamente soluzioni ammissibili, non sempre buone)
- breadth-first (in ampiezza): sceglie il primo nodo generato
  - rimane ai livelli superiori
- best-first: sceglie il nodo più promettente secondo un criterio, come il valore del bound $f^*_{R_i}$ o il valore della soluzione euristica $f(\bar{x}_{R_i})$
  (spesso indicati come best-bound e best-estimate)
  - tende a fornire rapidamente soluzioni buone
- ibrida: cambia il criterio di scelta nodo per nodo

Esempi



Componenti fondamentali di un algoritmo di B&B

Un algoritmo di branch-and-bound è definito da

1. una regola di branching
2. delle procedure di presolve per ridurre il problema
3. un rilassamento e un algoritmo per risolverlo
4. un'algoritmo per il calcolo di soluzioni euristiche
5. una strategia di visita dell'albero di branching

Poiché la complessità è esponenziale, queste componenti sono cruciali per l'efficienza dell'algoritmo nel

- trovare una buona soluzione ammissibile
- dimostrare che la soluzione trovata è ottima

Questi due scopi sono però parzialmente in conflitto

Usare il B&B in modo euristico

Applicando tutti i meccanismi correttamente, il B&B

1. trova una sequenza di soluzioni ammissibili $\bar{x}$, via via miglioranti
2. a un certo punto trova una soluzione ottima x^*
3. al termine dimostra che x^* è ottima

Terminato anticipatamente, diventa un algoritmo euristico

Ma quando?

Possibili condizioni di terminazione anticipata sono:

1. fissare un tempo limite (indicato con `TimeLimit` *in GUROBI*)
   - vantaggi: controllo totale sul tempo di calcolo
   - svantaggi: nessun controllo sulla qualità della soluzione (potremmo anche non trovarne)
2. un numero limite di nodi (`NodeLimit`)
   - consente di valutare la qualità di rilassamento, presolve e branching
3. un numero limite di soluzioni ammissibili (`SolutionLimit`)
   - garantisce almeno una soluzione, se esiste

Uso delle tolleranze

Anziché chiudere tutti i nodi da un certo punto in poi, sembra più intelligente chiudere i nodi meno promettenti durante l'intero processo

Il meccanismo di chiusura basato sulla dominanza fra bound ed euristica

$$f^*_{R_i} \geq f(\bar{x}) \Rightarrow \text{ chiudere } \Pi_i$$

si può rilassare parametricamente in modo assoluto (`MIPGapAbs`)

$$f^*_{R_i} \geq f(\bar{x}) - \Delta \Rightarrow \text{ chiudere } \Pi_i$$

oppure in modo relativo (`MIPGap`)

$$f^*_{R_i} \geq f(\bar{x})(1-\delta) \Rightarrow \text{ chiudere } \Pi_i$$

Al termine, $f(x_H)$ è approssimata, anche se potrebbe non essere ottima

$$f(x_H) \leq f^* + \Delta \quad \text{oppure} \quad f(x_H) \leq \frac{f^*}{1-\delta}$$

Al crescere di Δ (o di δ)

- la qualità della soluzione peggiora
- il tempo di calcolo diminuisce, non sempre monotonicamente

Perché?

Ricerca di soluzioni o dimostrazione di ottimalità?

Gli algoritmi di branch-and-bound perseguono due obiettivi in gran parte, ma non del tutto concordi

- trovare buone soluzioni ammissibili richiede di
  - investire tempo nelle euristiche
  - ridurre velocemente il problema facendo branching
  - scegliere variabili di branching vicine all'interezza

  con lo scopo fondamentale di abbassare $f(\bar{x})$
- dimostrare che la soluzione corrente è ottima richiede di
  - investire tempo nel presolve e nel rilassamento
  - scegliere variabili di branching lontane dall'interezza

  con lo scopo fondamentale di alzare il bound $\min_{\Pi_i \text{ aperti}} f^*_{R_i}$

I risolutori offrono parametri (piuttosto oscuri) per modulare questi comportamenti favorendo l'uno o l'altro o bilanciandoli (`MIPFocus`)

Questi parametri interagiscono con i limiti e le tolleranze già discussi

Diving/plunging

Molte euristiche sfruttano i meccanismi di branching e di visita

L'euristica di diving consiste nello scendere rapidamente dalla radice a una foglia promettente dell'albero

1. risolve il rilassamento R_i del nodo corrente Π_i
   (ed eventualmente chiude il nodo)
2. se Π_i è ancora aperto, calcola per ogni variabile x_j
   - un valore promettente $\tilde{x}_j$
   - un indice di confidenza ϕ_j
3. assegna alle δ variabili con ϕ_j migliore il valore $\tilde{x}_j$ e torna al punto 1

Ad ogni iterazione l'euristica scende di δ livelli nell'albero o termina

- in un nodo inammissibile o dominato dal miglior valore noto $f(\bar{x})$
- in una soluzione ammissibile x_H con $f(x_H) < f(\bar{x})$

È una versione adattiva delle euristiche di arrotondamento
dato che ad ogni passo corregge e riottimizza le variabili non fissate

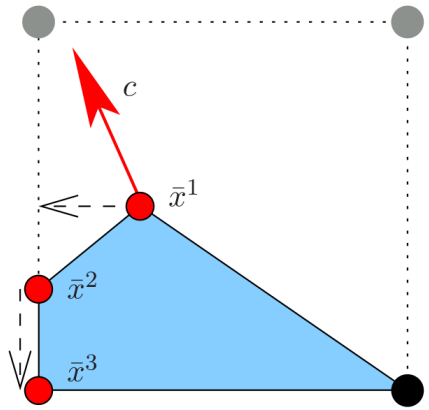
Si può estendere a rilassamenti diversi da quello continuo

Scelta delle variabili da fissare

L'indice di confidenza può essere basato su

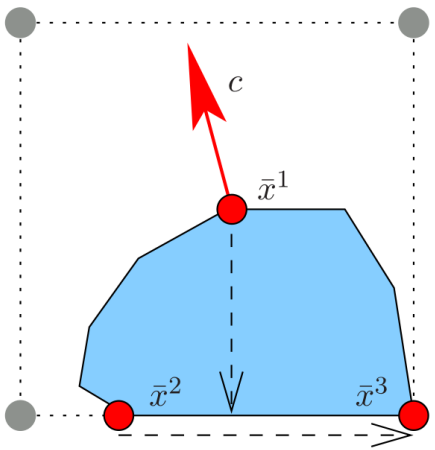
- la frazionalità delle variabili: $\phi_j = \min\left(x^*_{R_ij}, 1 - x^*_{R_ij}\right)$

  Valori bassi indicano variabili vicine a 0 *o a* 1



- il numero di vincoli "conflittuali" con il suo valore promettente

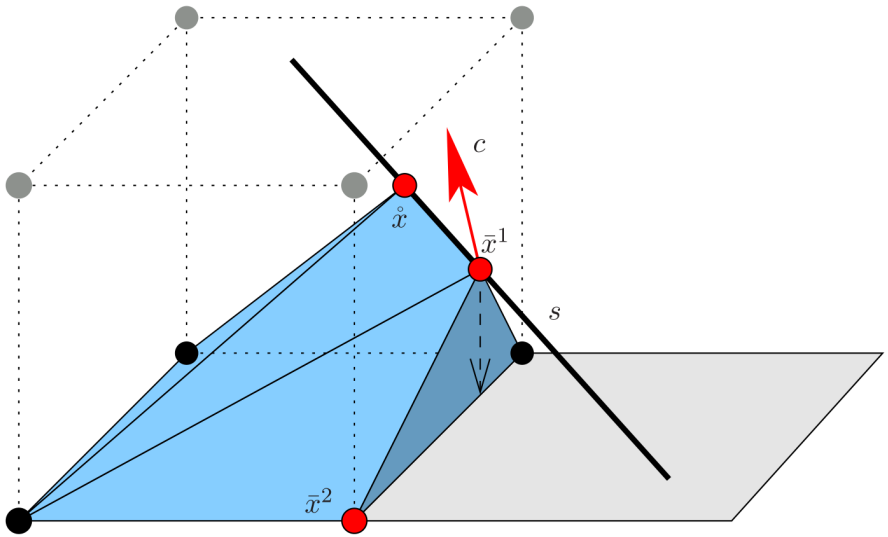
  Valori bassi indicano variabili meno critiche per l'ammissibilità



Scelta delle variabili da fissare

- la distanza dalla soluzione rilassata al nodo radice $x^*_{R_0}$:
  lungo la retta che collega $x^*_{R_i}$ a $x^*_{R_0}$, si cerca la prima intersezione con un piano $x_j = 0$ o $x_j = 1$

  Valori bassi indicano vicinanza a un punto ideale



- la distanza dalla miglior soluzione nota: $\phi_j = |x^*_{R_i j} - \bar{x}_j|$

  Valori bassi indicano vicinanza a un punto di buona qualità

*Diverse figure sono tratte dalla tesi di dottorato di T. Berthold

Scelta delle variabili da fissare

- pseudocosti che stimano l'impatto del fissaggio sull'obiettivo

  Esempio: sia $x^*_{R_i h} = 0.9$ e $x^*_{R_i k} = 0.6$, ma $c_h \gg c_k$

  Può essere meglio arrotondare x_k a 1 piuttosto che x_h

  Si può anche tener conto dell'effetto su classi di vincoli semplici

  Esempio: sia $x^*_{R_i h} = 0.9$ e $x^*_{R_i k} = 0.6$,
  ma vi siano molti vincoli di copertura $\sum_j a_{ij} x_j \geq 1$
  e x_h ne soddisfi molti meno di x_k $(\sum_i a_{ih} \ll \sum_i a_{ik})$

  Può essere meglio arrotondare x_k a 1 piuttosto che x_h

Scelta del numero di variabili da fissare

Il numero δ di variabili fissate ad ogni iterazione deve essere

- abbastanza basso da non chiudere i nodi con soluzioni miglioranti
- abbastanza alto da non richiedere troppo tempo di calcolo

Gli approcci più comuni sono i seguenti

1. fissare la variabile migliore ($\delta = 1$), specialmente nei problemi in cui le variabili interagiscono fortemente a causa dei vincoli
2. fissare una frazione $\alpha \in [0, 1]$ del numero di variabili ancora libere
3. fissare tutte le variabili con $\phi_j \leq \alpha \min_j \phi_j + (1 - \alpha) \max_j \phi_j$, dove $\alpha \in [0, 1]$ modula l'indice massimo per le variabili da fissare

Schema generale

Diversi altri parametri intervengono a

- terminare il metodo se lo sforzo computazionale è eccessivo
- prolungare il metodo se i risultati sembrano promettenti

Input: $\bar{x}$ LP-feasible point

1 $\bar{I} \leftarrow$ index set of all fractional variables of $\bar{x}$;

2 $c_{\max} \leftarrow \begin{cases} \infty & \text{if no feasible solution found so far} \\ c^T\tilde{x} & \text{else, with } \tilde{x} \text{ being the incumbent solution} \end{cases}$;

3 $nlp \leftarrow 0, i \leftarrow 0, nfr \leftarrow |\bar{I}|$;

4 **while** $\bar{I} \neq \emptyset$ **and** $c^T\bar{x} < c_{\max}$ **and** *(* $(nlp < nlp_{\max}$ **and** $i < i_{\max})$ **or** $i < i_{\min}$ **or** $|\bar{I}| \leq nfr - \frac{i}{2}$*)* **do**

5 $\quad i \leftarrow i + 1$;

6 $\quad$ Select some $j \in \bar{I}$;

7 $\quad$ Bound x_j: either $l_j \leftarrow \lceil \bar{x}_j \rceil$ or $u_j \leftarrow \lfloor \bar{x}_j \rfloor$;

8 $\quad$ **if** *modified LP has a feasible solution* **then**

9 $\quad\quad \bar{x} \leftarrow$ optimal solution of modified LP ;

10 $\quad$ **else**

11 $\quad\quad$ **stop!**

12 $\quad \bar{I} \leftarrow$ index set of all fractional variables of $\bar{x}$, $nfr \leftarrow |\bar{I}|$;

13 $\quad$ **if** *all* x_j *with* $j \in \bar{I}$ *are trivially roundable* **then**

14 $\quad\quad$ Round $\bar{x}$; `/* yields feasible solution */`

15 $\quad nlp \leftarrow nlp +$ number of LP-iterations needed to solve LP in Step 9;

Si veda la bibliografia per i dettagli

Un algoritmo di diving lagrangiano per il *SCP*

Nel 1994 Caprara, Fischetti e Toth vinsero la competizione FASTER, che proponeva problemi di crew scheduling delle Ferrovie dello Stato modellati come *SCP* fino a 5 000 righe e 1 000 000 colonne

$$\begin{aligned} \min f(x) &= \sum_{j \in J} c_j x_j \\ \sum_{j \in J} a_{ij} x_j &\geq 1 \qquad i \in I \\ x_j &\in \{0,1\} \qquad j \in J \end{aligned}$$

L'algoritmo fa un rilassamento lagrangiano di tutti i vincoli

$$\begin{aligned} \min f_\lambda(x) &= \sum_{j \in J} \Big(c_j - \sum_{i \in I} \lambda_i a_{ij}\Big) x_j + \sum_{i \in I} \lambda_i \\ x_j &\in \{0,1\} \qquad j \in J \end{aligned}$$

Un algoritmo di diving lagrangiano per il *SCP*

$$\min f_\lambda(x) = \sum_{j \in J} \left(c_j - \sum_{i \in I} \lambda_i a_{ij}\right) x_j + \sum_{i \in I} \lambda_i$$
$$x_j \in \{0,1\} \qquad j \in J$$

Ovviamente il rilassamento R si risolve ponendo

$$x^*_{Rj} = \begin{cases} 0 & \text{se } \tilde{c}_j(\lambda) > 0 \\ 1 & \text{se } \tilde{c}_j(\lambda) \leq 0 \end{cases} \quad \text{dove } \tilde{c}_j(\lambda) = c_j - \sum_{i \in I} \lambda_i a_{ij} \text{ (costo lagrangiano)}$$

e fornisce

1. un bound che si può ottimizzare con il metodo del sottogradiente

$$f^*_R(\lambda) = \sum_{j \in J} \min\left(\tilde{c}_j(\lambda), 0\right) + \sum_{i \in I} \lambda_i$$

2. una soluzione lagrangiana x^*_R, ammissibile o riparabile aggiungendo colonne in modo da ottenere una soluzione euristica

Diving per il *SCP*: test logici

Ci sono operazioni di presolve basate su test logici

1. colonne essenziali: se una riga è coperta da una sola colonna, tale colonna è indispensabile
2. dominanza per colonne: se una colonna copre un insieme di righe più piccolo di un'altra e non costa di meno, si può cancellare
3. dominanza per righe: se una riga è coperta da un insieme di colonne più grande di un'altra, si può cancellare

c	4	6	5	10	10	2

A					
1	0	1	1	0	0
0	1	1	0	0	0
1	1	0	0	0	1
0	0	0	1	1	0
1	0	1	0	0	0

1. la colonna 4 domina la colonna 5
2. la riga 5 domina la riga 1
3. la colonna 4 è sola a coprire la riga 4 (all'inizio non era così)

Diving per il *SCP*: test logici

Ci sono operazioni di presolve basate su test logici

1. colonne essenziali: se una riga è coperta da una sola colonna, tale colonna è indispensabile
2. dominanza per colonne: se una colonna copre un insieme di righe più piccolo di un'altra e non costa di meno, si può cancellare
3. dominanza per righe: se una riga è coperta da un insieme di colonne più grande di un'altra, si può cancellare

c	4	6	5	10	~~10~~	2

A						
	1	0	1	1	~~0~~	0
	0	1	1	0	~~0~~	0
	1	1	0	0	~~0~~	1
	0	0	0	1	~~1~~	0
	1	0	1	0	~~0~~	0

1. la colonna 4 domina la colonna 5
2. la riga 4 domina la riga 1
3. la colonna 4 è sola a coprire la riga 4 (all'inizio non era così)

Diving per il *SCP*: test logici

Ci sono operazioni di presolve basate su test logici

1. colonne essenziali: se una riga è coperta da una sola colonna, tale colonna è indispensabile
2. dominanza per colonne: se una colonna copre un insieme di righe più piccolo di un'altra e non costa di meno, si può cancellare
3. dominanza per righe: se una riga è coperta da un insieme di colonne più grande di un'altra, si può cancellare

c	4	6	5	10	~~10~~	2

A						
	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~	~~0~~
	0	1	1	0	~~0~~	0
	1	1	0	0	~~0~~	1
	0	0	0	1	~~1~~	0
	1	0	1	0	~~0~~	0

1. la colonna 4 domina la colonna 5
2. la riga 4 domina la riga 1
3. la colonna 4 è sola a coprire la riga 4 (all'inizio non era così)

Diving per il *SCP*: test logici

Ci sono operazioni di presolve basate su test logici

1. colonne essenziali: se una riga è coperta da una sola colonna, tale colonna è indispensabile
2. dominanza per colonne: se una colonna copre un insieme di righe più piccolo di un'altra e non costa di meno, si può cancellare
3. dominanza per righe: se una riga è coperta da un insieme di colonne più grande di un'altra, si può cancellare

c	4	6	5	10	~~10~~	2

A						
	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~	~~0~~
	0	1	1	0	~~0~~	0
	1	1	0	0	~~0~~	1
	~~0~~	~~0~~	~~0~~	~~1~~	~~1~~	~~0~~
	1	0	1	0	~~0~~	0

1. la colonna 4 domina la colonna 5
2. la riga 4 domina la riga 1
3. la colonna 4 è sola a coprire la riga 4
   si fissa la colonna e si cancella la riga

Diving per il *SCP*: penalità lagrangiane

Altre operazioni di presolve usano il rilassamento (penalità lagrangiane)

Sappiamo che

$$x^*_{Rj} = \begin{cases} 0 & \text{se } \tilde{c}_j(\lambda) > 0 \\ 1 & \text{se } \tilde{c}_j(\lambda) \leq 0 \end{cases} \qquad f^*_R(\lambda) = \sum_{j \in J} \min(\tilde{c}_j(\lambda), 0) + \sum_{i \in I} \lambda_i$$

Se forziamo a 1 una variabile attualmente a 0 il bound diventa

$$f^*_{R_1}(\lambda) = f^*_R(\lambda) + \tilde{c}_j(\lambda)$$

mentre forzando a 0 una variabile attualmente a 1 diventa

$$f^*_{R_1}(\lambda) = f^*_R(\lambda) - \tilde{c}_j(\lambda)$$

Se $f^*_R(\lambda) + |\tilde{c}_j(\lambda)| \geq f(\bar{x})$, si deve fissare x_j al valore corrente per poter migliorare $\bar{x}$

Introdurre una tolleranza Δ o δ renderebbe il fissaggio euristico

Un algoritmo di diving lagrangiano per il *SCP*

L'euristica di diving

- risolve al meglio il rilassamento lagrangiano
  con i test logici e le penalità lagrangiane
- se Π_i è ancora aperto, per ogni variabile x_j
  - considera valore promettente quello della soluzione lagrangiana $x^*_{R_i}$
  - calcola un (complicato) indice di confidenza ϕ_j per stimare il contributo di ciascuna alla differenza tra $\bar{x}$ e $x^*_{R_i}$
- ordina per indice crescente le variabili con $x^*_{R_ij} = 1$
- fissa a 1 le variabili con indice minimo
  fino a coprire una frazione α delle righe e torna al punto 1

L'euristica seconda classificata (Ceria, Nobili e Sassano) è simile, ma

- calcola anche un rilassamento lagrangiano del problema duale, con l'idea di stimare le variabili x_j con i moltiplicatori lagrangiani duali μ_j
- sceglie le variabili da fissare con opportune combinazioni di $\tilde{c}_j$ e μ_j
- ripete il procedimento compiendo scelte parzialmente casuali
  (*se il diving è un algoritmo greedy, questo è un semigreedy*)

Relax and fix

Una generalizzazione del diving consiste nel rilassare il problema in parte, tenendo alcune variabili intere e allargandone gradualmente l'insieme

Detto J l'insieme delle variabili, ad ogni passo k si definiscono

- il sottoinsieme $J_{\text{int}}^{(k)} \subseteq J$ delle variabili da rendere intere
- il sottoinsieme $J_{\text{fix}}^{(k)} \subseteq J_{\text{int}}$ delle variabili fissate a un valore $\bar{x}_j$

che danno luogo a un sottoproblema Π_k e a un suo rilassamento R_k

$$\begin{aligned} \min f(x) = c^T x & \\ Ax \leq b & \\ x_j = \bar{x}_j \qquad & j \in \bigcup_{\ell=1}^{k-1} J_{\text{fix}}^{(\ell)} \\ x_j \in \{0,1\} \qquad & j \in \bigcup_{\ell=1}^{k} J_{\text{int}}^{(\ell)} \\ 0 \leq x_j \leq 1 \qquad & j \in J \end{aligned}$$

L'insieme $\bigcup_{\ell=1}^{k} J_{\text{int}}^{(\ell)} \setminus \bigcup_{\ell=1}^{k-1} J_{\text{fix}}^{(\ell)}$ è spesso detto finestra

Relax and fix

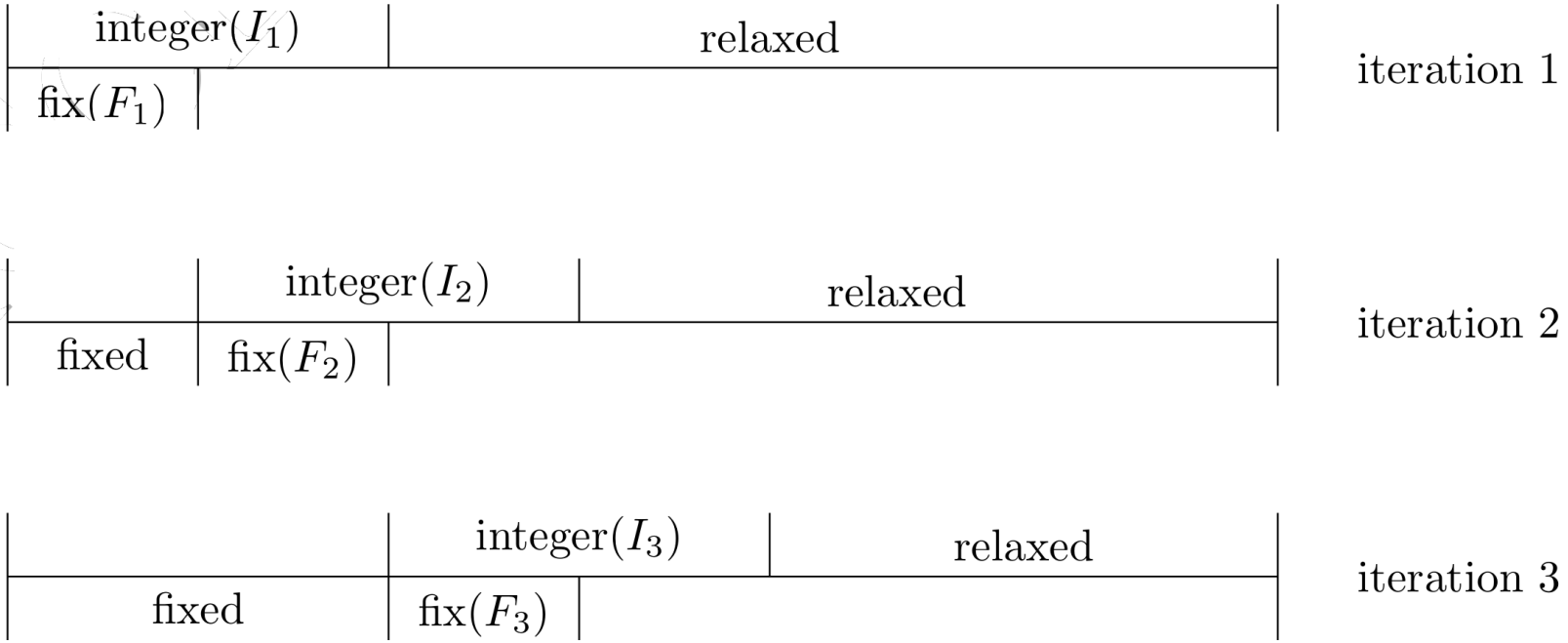
All'inizio $J_{\text{fix}}^{(0)} := \emptyset$ e $k := 1$

Ad ogni iterazione k

1. si eredita un insieme $\bigcup_{\ell=1}^{k-1} J_{\text{fix}}^{(\ell)}$ di variabili fissate e di valori
2. si impone l'interezza su alcune nuove variabili $J_{\text{int}}^{(k)}$
3. si risolve il rilassamento misto R_k
4. si fissano alcune nuove variabili $J_{\text{fix}}^{(k)} \subseteq J_{\text{int}}^{(k)}$ al valore ottenuto
5. se il problema è ammissibile e ci sono variabili con valori frazionari, si incrementa k e si torna al punto 1

```
1:  function RelaxAndFix(sol.w, windowSize, windowType, overlap, timeLimit)
2:      window ← initWindow( windowSize, windowType, sol.w )
3:      w_fix ← ∅
4:      w_MIP ← window
5:      w_LP ← sol.w − window
6:      while fixed solution not reached and elapsedTime < timeLimit do
7:          Solve( w_fix, w_MIP, w_LP )
8:          window ← moveWindow( overlap, windowSize )
9:          w_fix ← w_fix ∪ (w_MIP − window)
10:         w_MIP ← window
11:         w_LP ← w_LP − window
12:     end while
13:     sol.w ← w_fix
14:     return sol.w
15: end function
```

Relax and fix



Il punto cruciale del metodo è come scegliere i due insiemi ad ogni passo

- un insieme $J_{\text{fix}}^{(\ell)}$
  - ampio accelera il metodo, ma peggiora la qualità (e l'ammissibilità)
  - ridotto consente la riottimizzazione
- un insieme $J_{\text{int}}^{(\ell)}$
  - ampio migliora la qualità
  - ridotto riduce il tempo di calcolo
- di solito la scelta è guidata dall'applicazione

Relax and fix per il lot-sizing problem

Un'applicazione tipica sono i problemi di lot-sizing: decidere i livelli di produzione x_t e magazzino I_t in ogni periodo di un dato orizzonte minimizzando costi fissi e variabili di produzione e costi di magazzino

$$\min f(x) = \sum_{t=1}^{T} (s_t y_t + c_t x_t + h_t I_t)$$

$$\begin{aligned} I_t &= I_{t-1} + x_t - d_t && t \in T \\ x_t &\leq M y_t && t \in T \\ y_t &\in \{0,1\} && t \in T \\ x_t, I_t &\geq 0 && t \in T \end{aligned}$$

L'approccio tipico è

- fissare le variabili y_t fino a un dato periodo t_k
- imporre l'interezza sulle variabili y_t dal periodo t_{k+1} al periodo t'_k
- lasciar libere le variabili y_t dal periodo $t'_k + 1$ alla fine

Anche nei problemi di scheduling le variabili sono temporalmente ordinate

Fix and optimize

La scelta di non fissare subito tutte le variabili intere ($J_{\text{fix}}^{(k)} \subset J_{\text{int}}^{(k)}$)

- consente alle iterazioni seguenti di modificare alcuni valori interi per adattarsi all'introduzione dei futuri vincoli di interezza
- rende più flessibile la soluzione e aiuta a conservare l'ammissibilità
- consente di generare soluzioni diverse di passo in passo

Nel Fix and Optimize l'insieme delle variabili J è diviso in

- variabili da rendere intere $J_{\text{int}}^{(k)}$ (rolling window, o finestra scorrevole)
- variabili fissate $J_{\text{fix}}^{(k)} = J \setminus J_{\text{int}}^{(k)}$
- non ci sono variabili libere (*serve una soluzione iniziale*)

Questo consente di rilassare ciclicamente le variabili fissate
(per esempio di ripassare più e più volte sull'orizzonte temporale)

Fix and Optimize

```
1:  function FixAndOptimize(sol.w, windowSize, overlap, timeLimit)
2:      windowType ← 0
3:      repeat
4:          window ← initWindow( windowSize, windowType )
5:          w_MIP ← window
6:          w_fix ← sol.w − window
7:          while window not reached end do
8:              Solve( w_fix, w_MIP )
9:              window ← moveWindow( overlap, windowSize)
10:             w_MIP ← window
11:             w_fix ← sol.w − window
12:             sol.w ← w_fix ∪ w_MIP
13:         end while
14:         windowType ← 1 − windowType
15:     until elapsedTime < timeLimit and windowType ≠ 0
16: end function
```

In entrambi i metodi la finestra può essere

- definita in vari modi
  - in base all'applicazione (*ad es., in ordine temporale*)
  - in base alla soluzione del rilassamento (*frazionarietà delle variabili*)
- fatta scorrere in vari modi
  - in base agli indici delle variabili (*ad es., periodo, macchina, ecc. . .*)
  - in base a diversi ordinamenti delle variabili

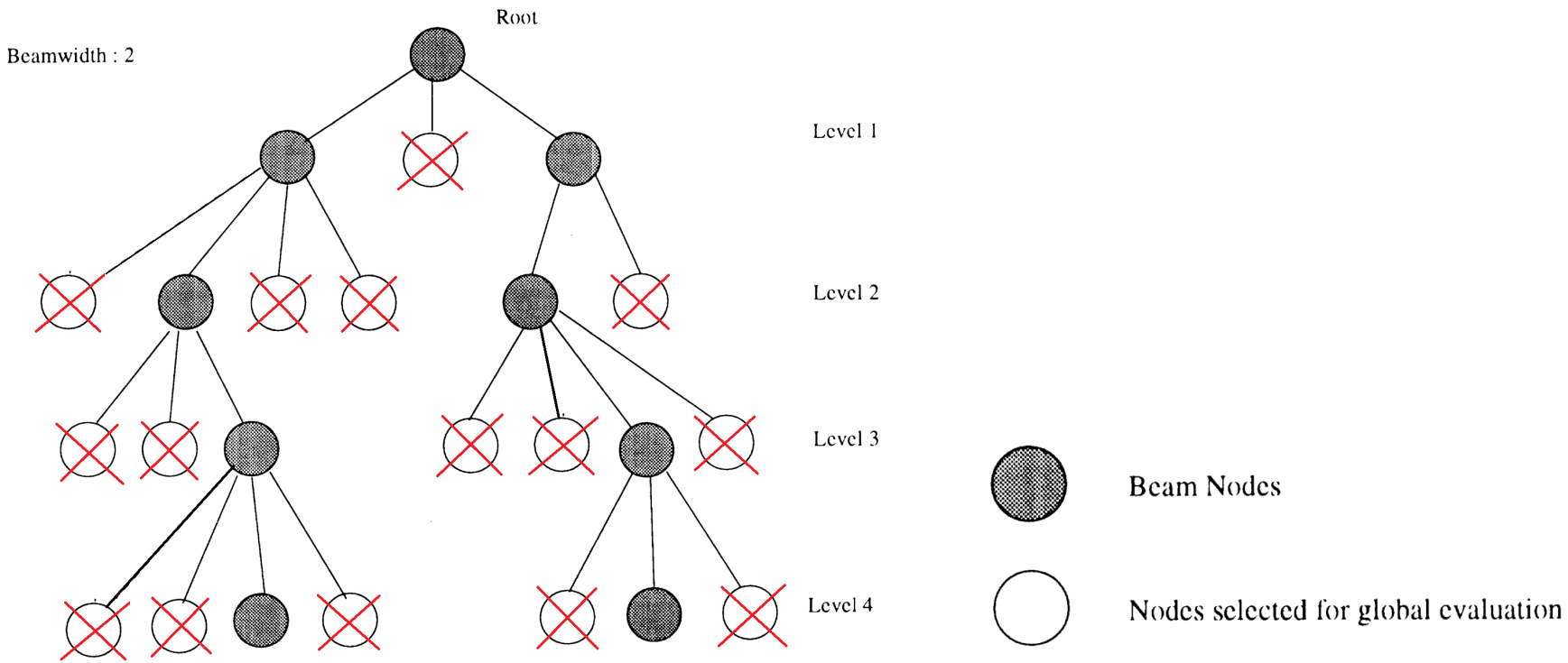
Beam search

Un'altra generalizzazione del diving consiste nello scendere in parallelo dalla radice a più foglie anziché una sola, limitando il numero di foglie

Si segue un "fascio" (beam) di cammini anziché uno solo

L'euristica di beam search

- visita l'albero di branching in ampiezza
- ad ogni livello, quando genera i nodi del livello successivo, tiene aperti solo i w (beam width) nodi più promettenti



Beam search: complessità

La complessità dell'algoritmo è in genere polinomiale perché

- si mantengono al massimo $w\,h_{\max}$ nodi,
  dove $h_{\max}$ è l'altezza dell'albero di branching, spesso $\leq |B|$
- ogni nodo genera al massimo $r\,(n)$ nodi, dove r è spesso costante

Con $w = 1$ si ha l'euristica di diving (un solo cammino da radice a foglia)

- valori alti di w consentono di trovare soluzioni migliori
- valori bassi di w riducono il tempo di calcolo

Il criterio che indica se un nodo Π_i è promettente può essere

- una stima raffinata dell'ottimo $f^*_{\Pi_i}$
  - stima per difetto $f^*_{R_i}$ valutata con un rilassamento
  - stima per eccesso $f\,(\bar{x}_{\Pi_i})$ valutata con un'euristica
- un criterio di qualità grossolano, ma veloce da valutare

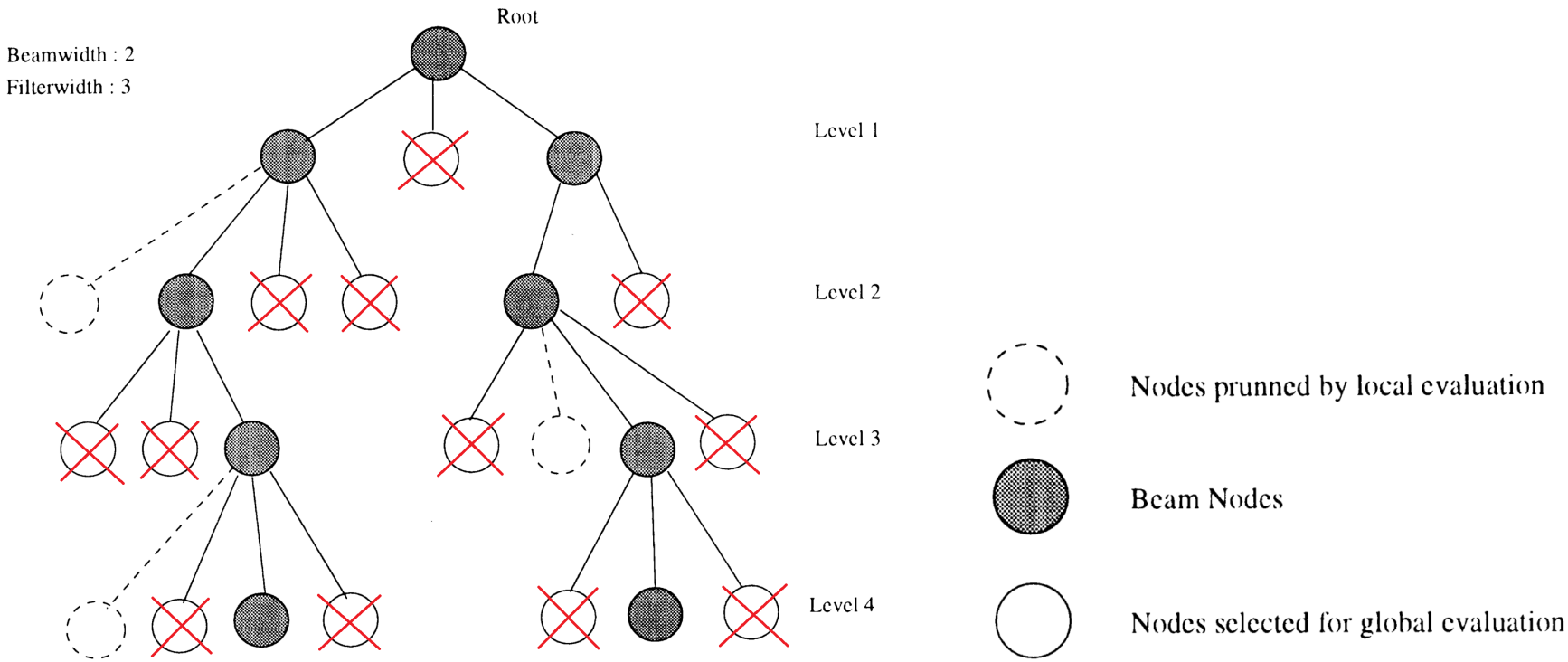
Oppure entrambe le cose

Filtered beam search

Questa variante opera in due fasi

1. valuta i nodi con un criterio "locale", rozzo ma molto veloce, e mantiene aperti solo i w_f nodi migliori
   - fra i figli di ciascun nodo aperto del livello precedente (quindi $w_f \leq r$)
   - fra i nodi dell'intero livello (quindi $w \leq w_f \leq w\,r$)

   secondo che non abbia o abbia senso confrontare figli di nodi diversi
2. valuta i nodi sopravvissuti al filtraggio con un criterio "globale", lento ma preciso, e mantiene aperti solo i w nodi migliori del livello



Filtered beam search per problemi di scheduling

Le applicazioni più convincenti riguardano i problemi di scheduling:

- assegnare operazioni (job) a macchine e sequenziarle nel tempo
- ottimizzando una funzione di prestazione
  (tempo massimo, totale, ritardi, anticipi...)
- rispettando vincoli di produzione (scadenze, precedenze...)

Tipicamente, in questi algoritmi di beam search

- il branching corrisponde alla scelta del job successivo da eseguire
  e della macchina a cui assegnarlo
- il criterio locale è una valutazione della soluzione parziale ottenuta
  (tempo di completamento, earliness, tardiness...)
- il criterio globale è il valore fornito da un'euristica
  (*un po' come succede negli algoritmi di roll-out*)

Recovering beam search per il p-median problem

La Recovering Beam Search ovvia al principale difetto del metodo (che i nodi chiusi non possono più essere ripresi in considerazione) sfruttando la ricerca locale per spostarsi in altre zone dell'albero

Della Croce, Ghirardi e Tadei la applicano al problema delle p-mediane: dati n punti e le loro distanze reciproche, scegliere p punti tali che la distanza totale fra loro e gli altri sia minima (localizzazione di *facilities*)

$$\min \sum_{i \in N} \sum_{j \in N} d_{ij} x_{ij}$$

$$\sum_{i \in N} x_{ij} = 1 \qquad j \in N$$

$$\sum_{i \in N} x_{ii} = p$$

$$\sum_{j \in N} x_{ij} \leq |N| \, x_{ii} \qquad i \in N$$

$$x_{ij} \in \{0, 1\}$$

Recovering beam search per il p-median problem

$$\min \sum_{i \in N} \sum_{j \in N} d_{ij} x_{ij}$$

$$\sum_{i \in N} x_{ij} = 1 \qquad j \in N$$

$$\sum_{i \in N} x_{ii} = p$$

$$\sum_{j \in N} x_{ij} \leq |N| \, x_{ii} \qquad i \in N$$

$$x_{ij} \in \{0, 1\}$$

L'algoritmo fa un rilassamento lagrangiano dei vincoli di assegnamento

$$\min \sum_{i \in N} \sum_{j \in N} (d_{ij} - \lambda_j) \, x_{ij} + \sum_{j \in N} \lambda_j$$

$$\sum_{i \in N} x_{ii} = p$$

$$\sum_{j \in N} x_{ij} \leq |N| \, x_{ii} \qquad i \in N$$

Recovering beam search per il p-median problem

$$\min \sum_{i \in N} \sum_{j \in N} (d_{ij} - \lambda_j)\, x_{ij} + \sum_{j \in N} \lambda_j$$

$$\sum_{i \in N} x_{ii} = p$$

$$\sum_{j \in N} x_{ij} \leq |N|\, x_{ii} \qquad i \in N$$

$$x_{ij} \in \{0, 1\}$$

Definito il costo lagrangiano $\tilde{d}_{ij}(\lambda) = d_{ij} - \lambda_j$, il rilassamento R si risolve

- aprendo le p mediane con $\sum_{j \in J} \min(d_{ij} - \lambda_j, 0)$ minimo
- assegnando ogni nodo j a tutte le mediane con $\tilde{d}_{ij}(\lambda) \leq 0$

e fornisce un bound $f^*_{R_i}$ e una soluzione lagrangiana,
riparabile assegnando ogni nodo alla mediana più vicina per ottenere $\bar{x}_{\Pi_i}$

Recovering beam search per il p-median problem

L'euristica base di Beam Search

- fa branching sull'ultima x_{ii} fissata a 1 in R o sulla prima fissata a 0
- valuta i nodi del livello corrente con il criterio globale

$$\alpha f^*_{R_i} + (1 - \alpha)\, f\,(\bar{x}_{\Pi_i})$$

  che fa una combinazione convessa di bound e valore euristico con un opportuno coefficiente $\alpha \in [0, 1]$
- non usa alcun criterio locale per filtrare velocemente i nodi
- arrivato ad una foglia, cioè fissate p variabili x_{ii} a 1, risolve il problema assegnando ogni nodo j alla mediana più vicina

Recovering beam search per il p-median problem

Al generico livello ℓ, i problemi aperti corrispondono a partizioni di N in

- un insieme N_1 di ℓ_1 nodi le cui variabili x_{ii} sono fissate a 1
- un insieme N_0 di $\ell_0 = \ell - \ell_1$ nodi le cui variabili x_{ii} sono fissate a 0
- un insieme N_f di $n - \ell$ nodi le cui variabili x_{ii} sono libere

Partendo da ciascun problema, se ne esplorano altri al livello ℓ

- le variabili libere sono fissate al loro valore corrente
- una variabile x_{ii} fissata a 1 si scambia con una fissata a 0,
  cioè si scambia una mediana fra N_1 e N_0 (*il loro numero non cambia*)

Se il nuovo assegnamento è meglio di quello corrente

- si fissano le variabili di N_1 e N_0 ai nuovi valori,
  cioè ci si sposta lungo l'albero in orizzontale
- si ricomincia a scambiare variabili

Quando non ci sono più scambi miglioranti

- si liberano ancora le variabili associate a N_f
- si generano i nodi figli a partire dal nuovo nodo corrente

Scandito tutto il livello, si scelgono i w nodi migliori e si chiudono gli altri

Bibliografia

- S. Martello, (1994). Algoritmi Branch-and-Bound: Strategie di Esplorazione e Rilassamenti. In: G. Di Pillo, Metodi di ottimizzazione per le decisioni, Masson, Milano, 1994
- M. Karamanov, G. Cornuéjols, (2011). Branching on General Disjunctions. *Mathematical Programming*, 128 (1-2), 403–436.
- M.W.P. Savelsbergh, (1994). Preprocessing and Probing Techniques for Mixed Integer Programming Problems. *ORSA Journal on Computing*, 6 (4), 445–454.
- T. Berthold, (2006). Primal Heuristics for Mixed Integer Programs. Ph. D. thesis, Konrad-Zuse-Zentrum für Informationstechnik, Technischen Universität Berlin.
- A. Caprara, M. Fischetti, P. Toth, (1999). A heuristic method for the set covering problem, *Operations Research*, 47, 730–743.
- S. Helber, F. Sahling, (2010). A Fix-and-Optimize Approach for the Multi-Level Capacitated Lot Sizing Problem. *International Journal of Production Economics*, 123 (2), 247–256
- F. Della Croce, M. Ghirardi, R. Tadei, (2004). Recovering Beam Search: Enhancing the Beam Search Approach for Combinatorial Optimization Problems. *Journal of Heuristics*, 10, 89–104.