



SAPIENZA
UNIVERSITÀ DI ROMA

Programmazione lineare intera

Volume 4 — Euristiche MIP

Arrotondamento, diving, feasibility pump, RINS, local branching, relax-and-fix e fix-and-optimize. Esempi ed esercizi con la soluzione completa.

Fabio Furini

fabiofurini.github.io

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

Versione online e materiale scaricabile:

fabiofurini.github.io/programmazione-lineare-intera/
github.com/fabiofurini/programmazione-lineare-intera

Anno accademico 2026–2027

Indice

V	Euristiche MIP	2
10	Euristiche MIP	3
10.1	Euristiche basate sulla PLI	4
10.2	Gli esempi del capitolo	5
10.3	Arrotondamento	6
10.4	Diving	8
10.5	La feasibility pump	11
10.6	Migliorare l'incumbent con un sotto-MIP	15
10.7	Relax-and-fix e fix-and-optimize	22
10.8	Le euristiche dentro il solver	25
10.9	Che cosa danno le euristiche basate sulla PLI	26
10.10	Implementazione in Python	27
10.11	Errori tipici	35
10.12	In sintesi	35



Parte V

Euristiche MIP



Euristiche MIP

Prerequisiti

- Da questo modulo, i capitoli «Bound primali e bound duali» e «Algoritmi branch-and-bound basati su LP»: il rilassamento lineare, i lower bound e gli upper bound, il gap di ottimalità e le operazioni di branching.
- Da questo modulo, il capitolo «Risolutori MIP: che cosa fanno»: la soluzione incumbent, i sotto-problemi risolti da un solver e le euristiche al suo interno.
- Dal modulo *Programmazione lineare*: l'[algoritmo del simplesso duale \(Capitolo 10\)](#), che riottimizza un LP dopo la modifica di un limite su una variabile.

Obiettivi

Alla fine di questo capitolo sarai in grado di:

- spiegare come un algoritmo euristico usa il rilassamento lineare e un solver MIP;
- eseguire a mano l'arrotondamento, il diving, la feasibility pump, RINS, il local branching e la proximity search su un piccolo MIP;
- eseguire relax-and-fix e fix-and-optimize su un MIP le cui variabili sono suddivise in blocchi;
- decidere quale metodo usare quando manca una soluzione ammissibile, o quando una soluzione incumbent va migliorata;
- calcolare il gap di una soluzione euristica quando è disponibile un bound duale.

10.1 Euristiche basate sulla PLI

Definizione 10.1: MIP in forma generale

Un **MIP** con N variabili $\mathbf{x} = (x_1, x_2, \dots, x_N)$ si scrive

$$\min f(\mathbf{x}) \quad (10.1a)$$

$$\text{soggetto a } A\mathbf{x} \leq b, \quad (10.1b)$$

$$l_j \leq x_j \leq u_j, \quad \forall j \in \{1, 2, \dots, N\}, \quad (10.1c)$$

$$x_j \in \mathbb{Z}, \quad \forall j \in \mathcal{D}, \quad (10.1d)$$

dove f è una funzione lineare e $\mathcal{D} \subseteq \{1, 2, \dots, N\}$ è l'insieme degli indici delle **variabili intere**. La regione ammissibile del **rilassamento lineare** è

$$\mathcal{L} = \{\mathbf{x} \in \mathbb{R}^N : A\mathbf{x} \leq b, l \leq \mathbf{x} \leq u\},$$

e $z(\text{LP}) = \min\{f(\mathbf{x}) : \mathbf{x} \in \mathcal{L}\}$ è il valore del rilassamento lineare.

- Un problema di massimizzazione, come l'esempio dell'auto, si tratta nello stesso modo, scambiando min e max, $\leq$ e $\geq$ nei confronti fra valori della funzione obiettivo.

Definizione 10.2: euristica basata sulla PLI

Un **algoritmo euristico** restituisce una soluzione ammissibile di un problema MIP (10.1) senza alcuna garanzia di ottimalità. È **basato sulla PLI** (una **matheuristica**) se usa il rilassamento lineare e un solver MIP per cercare la soluzione.

- Un'euristica fornisce un bound primale (un upper bound per un problema di minimizzazione, un lower bound per un problema di massimizzazione), mai un bound duale. L'algoritmo branch-and-bound di questo modulo trova la soluzione incumbent solo per caso, come foglia dell'albero.
- La soluzione ottima del **rilassamento lineare** indica *dove cercare*: è la soluzione di un problema più semplice, spesso vicina alle buone soluzioni del problema originale.
- Il **solver MIP** esplora *in modo esatto* un insieme di soluzioni troppo grande per essere enumerato, purché l'insieme sia descritto da vincoli lineari.

Definizione 10.3: incumbent, arrotondamento e completamento

- Una soluzione ammissibile $\bar{\mathbf{x}}$ del MIP (10.1), la migliore nota, è la soluzione **incumbent**, e $f(\bar{\mathbf{x}})$ è il suo valore.
- L'**arrotondamento** di un numero reale v all'intero più vicino è $[v] = \lfloor v + \frac{1}{2} \rfloor$, così che un valore $\frac{1}{2}$ viene arrotondato per eccesso.
- Un punto $\bar{\mathbf{x}}$ con $\bar{x}_j \in \mathbb{Z}$ per ogni $j \in \mathcal{D}$ viene **completato** fissando le variabili intere ai valori $\bar{x}_j$ e risolvendo l'LP sulle variabili continue.

- In tutto il capitolo, $\tilde{\mathbf{x}}$ indica una soluzione ottima di un rilassamento lineare, e $\bar{\mathbf{x}}$ un punto le cui componenti $\bar{x}_j$, $j \in \mathcal{D}$, sono intere.
- Se l'LP del completamento ha una soluzione ottima, $\bar{\mathbf{x}}$ diventa una soluzione ammissibile del MIP; se è inammissibile, lo è anche $\tilde{\mathbf{x}}$. Quando tutte le variabili sono intere, il completamento si limita a verificare i vincoli.
- I metodi si basano su cinque idee, una o due per sezione.

idea	metodi
arrotondare il rilassamento	arrotondamento; diving (fissare una variabile, risolvere di nuovo, ripetere)
inseguire l'ammissibilità	la feasibility pump
fissare dove rilassamento e incumbent concordano	RINS
limitare la distanza dall'incumbent	local branching; proximity search
scomporre in blocchi di variabili	relax-and-fix; fix-and-optimize

- Gli esempi sono risolti con il solver MIP Gurobi, usato da Python tramite `gurobipy`; il modulo *Modellazione MIP* ([fabiofurini.github.io/modellazione-mip](https://github.com/fabiofurini/modellazione-mip), capitolo 3 «Dal modello a Python/Gurobi») spiega come usarlo. Quando un LP ha più soluzioni ottime, il testo riporta quella restituita dal solver; tutti i codici sono nell'ultima sezione.

10.2 Gli esempi del capitolo

- I metodi sono illustrati sull'**esempio dell'auto** di questo modulo e su quattro piccoli problemi del **catalogo dei problemi MIP**. Sono tutti problemi PLI in forma canonica di massimizzazione, con dati interi e $x_j \in \mathbb{Z}_{\geq 0}$ per ogni j , così che $\mathcal{D} = \{1, 2, \dots, N\}$.
- L'**esempio dell'auto** si scrive

$$\max 3x_1 + 2x_2 \quad \text{s.a.} \quad 10x_1 + 4x_2 \leq 45, \quad 4x_1 + 14x_2 \leq 49, \quad x_1, x_2 \in \mathbb{Z}_{\geq 0}$$

problema	obiettivo max	vincoli
P7	$3x_1 + x_2$	$x_1 + 4x_2 \leq 12, \quad 2x_1 + x_2 \leq 5$
P42	$3x_1 + 2x_2 + 4x_3$	$x_1 + x_2 + 2x_3 \leq 4, \quad 2x_1 + 3x_3 \leq 5, \quad 2x_1 + x_2 + 3x_3 \leq 7$
P46	$5x_1 + 6x_2 + 10x_3$	$9x_1 + 2x_2 + 2x_3 \leq 11, \quad 2x_1 + 6x_2 + 6x_3 \leq 16, \quad 4x_1 + 2x_2 + 7x_3 \leq 10$
P72	$7x_1 + 6x_2 + 5x_3 - 2x_4 + 3x_5$	$x_1 + 3x_2 + 5x_3 - 2x_4 + 2x_5 \leq 4, \quad 4x_1 + 2x_2 - 2x_3 + x_4 + x_5 \leq 3, \quad 2x_1 + 4x_2 + 4x_3 - 2x_4 + 5x_5 \leq 5, \quad 3x_1 + x_2 + 2x_3 - x_4 - 2x_5 \leq 1$

- La tabella seguente riporta il valore $z(\text{LP})$ del rilassamento lineare, l'upper bound $\lfloor z(\text{LP}) \rfloor$ (la funzione obiettivo assume valori interi) e il valore ottimo $z(\text{ILP})$, tutti calcolati con l'algoritmo del simplesso in aritmetica esatta e verificati con Gurobi.

problema	variabili	vincoli	$z(\text{LP})$	$\lfloor z(\text{LP}) \rfloor$	$z(\text{ILP})$
auto	2	2	$\frac{31}{2}$	15	14
P7	2	2	$\frac{15}{2}$	7	7
P42	3	3	$\frac{21}{2}$	10	10
P46	3	3	$\frac{499}{25}$	19	16
P72	5	4	$\frac{507}{59}$	8	7

Esempio 10.1: Le soluzioni LP degli esempi

problema	soluzione ottima $\tilde{x}$ del rilassamento lineare
auto	$(\frac{7}{2}, \frac{5}{2})$
P7	$(\frac{5}{2}, 0)$
P42	$(\frac{5}{2}, \frac{3}{2}, 0)$
P46	$(\frac{17}{25}, \frac{49}{25}, \frac{12}{25})$
P72	$(\frac{39}{59}, 0, \frac{91}{59}, \frac{166}{59}, \frac{37}{59})$

Le prime tre soluzioni hanno al più due componenti frazionarie, con parte frazionaria $\frac{1}{2}$; le ultime due sono gli esempi in cui i metodi hanno più occasioni di fallire.

10.3 Arrotondamento

Definizione 10.4: arrotondamento

L'*arrotondamento* (rounding) è la più semplice euristica basata sulla PLI. Risolve il rilassamento lineare, arrotonda ogni variabile intera della soluzione LP all'intero più vicino e completa le variabili continue. Restituisce una soluzione ammissibile, oppure fallisce se il punto arrotondato viola un vincolo.

- Lo pseudocodice dell'arrotondamento è il seguente:

Algoritmo

Algorithm: Arrotondamento

Input: MIP (10.1) con variabili intere $\mathcal{D}$

Output: Soluzione ammissibile $\bar{x}$, oppure FALLIMENTO

```

1 Function Rounding(MIP)
2    $\tilde{x} :=$  soluzione ottima del rilassamento lineare           //  $\min\{f(x) : x \in \mathcal{L}\}$ 
3   for  $j \in \mathcal{D}$  do
4      $\bar{x}_j \leftarrow \lfloor \tilde{x}_j \rfloor$                                // arrotonda all'intero più vicino
5   end
6   fissa  $x_j := \bar{x}_j$  per ogni  $j \in \mathcal{D}$  e risolvi l'LP sulle variabili continue
7   if l'LP è ammissibile then
8     return  $\bar{x}$  completato con la parte continua della soluzione LP
9   else
10    return FALLIMENTO // il punto arrotondato viola un vincolo
11  end

```

- **Perché può funzionare.** Quando la formulazione è forte, la soluzione LP è vicina a una soluzione intera ottima, e molte delle sue variabili intere sono già intere: l'arrotondamento ne cambia solo poche, ciascuna di meno di $\frac{1}{2}$.
- **Perché può fallire.** L'arrotondamento tratta le variabili una alla volta, mentre i vincoli le legano fra loro. Un vincolo con tutti i coefficienti non negativi e termine noto $b_i \geq 0$ resta soddisfatto se le variabili vengono arrotondate *per difetto*, e uno della forma $\geq$ se vengono arrotondate *per eccesso*.
- Un vincolo di **uguaglianza** è quasi sempre violato da arrotondamenti indipendenti: due valori frazionari con somma 1 diventano, per esempio, 1 e 1.

Osservazione 10.5

Siano tutti i coefficienti di A non negativi, $b \geq 0$ e $l = 0$. Se $\tilde{x} \in \mathcal{L}$, allora il punto $\lfloor \tilde{x} \rfloor$, ottenuto arrotondando per difetto ogni variabile intera, appartiene a $\mathcal{L}$.

Dimostrazione. Per ogni j , $0 \leq \lfloor \tilde{x}_j \rfloor \leq \tilde{x}_j \leq u_j$, quindi i limiti sono rispettati. Per ogni riga i , poiché $a_{ij} \geq 0$, $\sum_j a_{ij} \lfloor \tilde{x}_j \rfloor \leq \sum_j a_{ij} \tilde{x}_j \leq b_i$. $\square$

- L'esempio dell'auto soddisfa le ipotesi dell'osservazione: l'arrotondamento per difetto è sempre ammissibile e, in generale, non ottimo.

Esempio 10.2: Arrotondamento sull'esempio dell'auto e su $P42$

Esempio dell'auto. La soluzione LP $\tilde{x} = (\frac{7}{2}, \frac{5}{2})$ viene arrotondata a $\bar{x} = (4, 3)$, che viola il primo vincolo:

$$10 \cdot 4 + 4 \cdot 3 = 52 > 45.$$

L'arrotondamento fallisce. L'arrotondamento per difetto dà invece $(3, 2)$: $38 \leq 45$, $40 \leq 49$, di valore 13, mentre $z(\text{ILP}) = 14$.

$P42$. La soluzione LP $(\frac{5}{2}, \frac{3}{2}, 0)$ viene arrotondata a $(3, 2, 0)$, che viola il secondo vincolo: $2 \cdot 3 = 6 > 5$. **L'arrotondamento fallisce.** L'arrotondamento per difetto dà $(2, 1, 0)$, di valore 8, mentre $z(\text{ILP}) = 10$.

$P72$. La soluzione LP viene arrotondata a $(1, 0, 2, 3, 1)$, che viola il primo vincolo: $1 + 10 - 6 + 2 = 7 > 4$. **L'arrotondamento fallisce.**

- Gli esempi riassumono il metodo: l'arrotondamento all'intero più vicino fallisce quando più variabili sono frazionarie contemporaneamente e i vincoli sono stringenti; l'arrotondamento per difetto è sicuro, ma lascia per strada una parte del valore dell'LP.

10.4 Diving

- L'arrotondamento prende tutte le sue decisioni in una volta, sulla base di un'unica soluzione LP. Il **diving** le prende una alla volta, e dopo ogni decisione interroga di nuovo l'LP: la soluzione LP viene ricalcolata con le decisioni prese fino a quel momento, e si adatta a esse.

Definizione 10.6: diving

Il *diving* risolve ripetutamente il rilassamento lineare, sceglie una variabile intera con valore frazionario, la fissa a un valore intero e risolve di nuovo l'LP. Si ferma con una soluzione ammissibile quando la soluzione LP è intera su tutte le variabili intere, e fallisce quando l'LP diventa inammissibile. Una *regola* decide quale variabile fissare e a quale valore.

- Lo pseudocodice del diving è il seguente:

Algoritmo

Algorithm: Diving

Input: MIP (10.1); una regola; per il diving guidato, una soluzione incumbent $\bar{x}$

Output: Soluzione ammissibile, oppure FALLIMENTO

```

1 Function Diving(MIP, regola)
2   while vero do
3     risolvi il rilassamento lineare con i limiti correnti  $l, u$ 
4     if l'LP è inammissibile then
5       return FALLIMENTO // il diving finisce in un LP
6       inammissibile
7     end
8      $\tilde{x} :=$  soluzione ottima dell'LP
9      $F := \{j \in \mathcal{D} : \tilde{x}_j \notin \mathbb{Z}\}$  // variabili intere frazionarie
10    if  $F = \emptyset$  then
11      return  $\tilde{x}$  // intera su  $\mathcal{D}$ : ammissibile
12    end
13    scegli  $j \in F$  e un valore  $v \in \mathbb{Z}$  secondo la regola
14     $l_j \leftarrow v; \quad u_j \leftarrow v$  // fissa  $x_j := v$ 
15  end

```

- **Diving frazionario.** Si sceglie la variabile *meno* frazionaria, cioè la $j \in F$ che minimizza $|\tilde{x}_j - \lfloor \tilde{x}_j \rfloor|$, e la si fissa a $v = \lfloor \tilde{x}_j \rfloor$. L'LP è quasi sicuro del valore di questa variabile, quindi fissarla cambia la soluzione LP il meno possibile. Nei codici di questo capitolo, i pareggi si risolvono a favore della variabile di indice più piccolo.
- **Diving guidato** [3]. Data una soluzione incumbent $\bar{x}$, si sceglie la $j \in F$ che minimizza $|\tilde{x}_j - \bar{x}_j|$ e la si fissa a $v = \bar{x}_j$. L'incumbent guida il diving verso una regione di buone soluzioni, e l'LP decide in quale ordine adottarne i valori.
- Esistono molte altre regole: il *coefficient diving* fissa la variabile che compare nel minor numero di vincoli che il suo arrotondamento può violare; il *pseudo-cost diving* usa la storia dell'albero di branch-and-bound; il *vector-length diving* è pensato per i vincoli di set covering. I solver moderni ne eseguono diversi.
- **Diving e branch-and-bound.** Fissare $x_j := \lfloor \tilde{x}_j \rfloor$ è una decisione di branching. Un diving è un unico cammino dalla radice dell'albero di branch-and-bound a una foglia, che prende sempre uno dei due figli, senza mai tornare indietro: una ricerca in profondità senza backtracking.
- **Costo.** Ogni passo risolve un LP che differisce dal precedente per un solo limite su una variabile, quindi l'algoritmo del simpleso duale lo riottimizza partendo dalla base precedente in poche iterazioni. Un diving con $|\mathcal{D}|$ variabili intere richiede al più $|\mathcal{D}|$ LP, di solito molti meno, perché una soluzione LP ha già molte componenti intere.

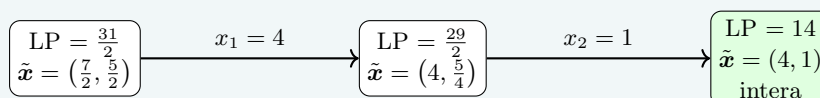
- **Fallimento.** L'LP diventa inammissibile quando le fissazioni fatte fino a quel momento sono incompatibili. Le implementazioni allora annullano l'ultima fissazione e provano l'altro valore (un solo backtracking), oppure rinunciano.

Esempio 10.3: Diving frazionario sull'esempio dell'auto

Passo 1. La soluzione LP è $\tilde{x} = (\frac{7}{2}, \frac{5}{2})$, di valore $\frac{31}{2}$. Entrambe le variabili sono a distanza $\frac{1}{2}$ dall'intero più vicino; il pareggio si risolve con l'indice: $x_1 := \lceil \frac{7}{2} \rceil = 4$.

Passo 2. Con $x_1 = 4$ i vincoli diventano $4x_2 \leq 5$ e $14x_2 \leq 33$, quindi la soluzione LP è $\tilde{x} = (4, \frac{5}{4})$, di valore $12 + \frac{5}{2} = \frac{29}{2}$. Si fissa $x_2 := \lceil \frac{5}{4} \rceil = 1$.

Passo 3. La soluzione LP $(4, 1)$, di valore 14, è intera: il diving restituisce $(4, 1)$, che è la soluzione **ottima** del problema PLI.



Esempio 10.4: Diving guidato sull'esempio dell'auto e su P7

Esempio dell'auto. La soluzione incumbent è $\bar{x} = (3, 2)$, di valore 13 (arrotondamento per difetto). Nella soluzione LP $(\frac{7}{2}, \frac{5}{2})$ entrambe le variabili sono a distanza $\frac{1}{2}$ da $\bar{x}$; si fissa $x_1 := 3$. La nuova soluzione LP è $(3, \frac{37}{14})$, di valore $\frac{100}{7}$; si fissa $x_2 := \bar{x}_2 = 2$. La soluzione LP $(3, 2)$ è intera, di valore 13: il diving **restituisce l'incumbent**, senza migliorarlo.

P7. La soluzione incumbent è $\bar{x} = (2, 0)$, di valore 6 (arrotondamento per difetto di $(\frac{5}{2}, 0)$). L'unica variabile frazionaria è x_1 : si fissa $x_1 := 2$. La soluzione LP $(2, 1)$, di valore 7, è intera: il diving guidato trova la soluzione **ottima** dopo una sola fissazione, anche se l'incumbent ha $\bar{x}_2 = 0$, perché l'LP ricostruisce le variabili che non sono fissate.

- L'incumbent decide solo i *valori* delle variabili fissate: sull'esempio dell'auto entrambi i valori fissati coincidono con quelli dell'incumbent scadente, su P7 il valore fissato è quello giusto.

Esempio 10.5: Un diving che fallisce su P42 e su P72

P42. La soluzione LP è $(\frac{5}{2}, \frac{3}{2}, 0)$; x_1 e x_2 sono a distanza $\frac{1}{2}$, e il pareggio si risolve con l'indice: $x_1 := \lceil \frac{5}{2} \rceil = 3$. Il secondo vincolo dà $2 \cdot 3 + 3x_3 \leq 5$, violato per ogni $x_3 \geq 0$: l'LP è inammissibile, **il diving fallisce al primo passo**.

P72. Le componenti frazionarie della soluzione LP sono $\frac{39}{59}, \frac{91}{59}, \frac{166}{59}, \frac{37}{59}$ (per x_1, x_3, x_4, x_5), e la meno frazionaria è x_4 , a distanza $\frac{11}{59}$ da 3: $x_4 := 3$. Il nuovo LP ha valore $\frac{728}{85}$ e soluzione $(\frac{56}{85}, 0, \frac{138}{85}, 3, \frac{52}{85})$; la variabile meno frazionaria è x_1 , arrotondata a 1. Fissare $x_1 := 1$ e $x_4 := 3$ rende l'LP **inammissibile**.

- In entrambi i casi le prime fissazioni seguono l'LP, che non vede che i vincoli, tutti insieme, non lasciano spazio. Un diving che fallisce non dà nessuna soluzione: è questa la ragione della feasibility pump della prossima sezione.

10.5 La feasibility pump

- L'arrotondamento e il diving falliscono su $P42$ e su $P72$, e su molti MIP falliscono la maggior parte delle volte: per un MIP generale, anche solo *decidere se esiste una soluzione ammissibile* è NP-completo. La **feasibility pump** di Fischetti, Glover e Lodi [4] rinuncia alla funzione obiettivo e si concentra su un unico scopo: trovare una soluzione ammissibile.
- L'idea è lavorare con due punti contemporaneamente. Un punto di $\mathcal{L}$ soddisfa tutti i vincoli lineari, ma può essere frazionario; un punto intero è intero, ma può violare i vincoli. Una soluzione ammissibile è un punto che ha entrambe le proprietà. La pompa fa inseguire i due punti finché non coincidono, e per farlo deve misurare quanto sono lontani l'uno dall'altro.

Definizione 10.7: Distanza da un punto intero

Sia $\bar{x}$ un punto con $\bar{x}_j \in \mathbb{Z}$ per ogni $j \in \mathcal{D}$. La **distanza** di un punto $x \in \mathbb{R}^N$ da $\bar{x}$ è

$$\Delta(x, \bar{x}) = \sum_{j \in \mathcal{D}} |x_j - \bar{x}_j|,$$

la distanza L_1 calcolata solo sulle variabili intere.

- Quando tutte le variabili intere sono **binarie**, $|x_j - \bar{x}_j|$ vale x_j se $\bar{x}_j = 0$ e $1 - x_j$ se $\bar{x}_j = 1$, per ogni $x_j \in [0, 1]$. La distanza è allora una funzione *lineare* di x :

$$\Delta(x, \bar{x}) = \sum_{j \in \mathcal{D}: \bar{x}_j=0} x_j + \sum_{j \in \mathcal{D}: \bar{x}_j=1} (1 - x_j) \quad \text{per variabili binarie}$$

- Per una variabile intera generale vale lo stesso se $\bar{x}_j = l_j$ (il termine è $x_j - l_j$) o $\bar{x}_j = u_j$ (il termine è $u_j - x_j$). Altrimenti $|x_j - \bar{x}_j|$ si sostituisce con una variabile ausiliaria d_j con i due vincoli $d_j \geq x_j - \bar{x}_j$ e $d_j \geq \bar{x}_j - x_j$; all'ottimo, $d_j = |x_j - \bar{x}_j|$. In tutti i casi, il punto di $\mathcal{L}$ più vicino a $\bar{x}$ si calcola risolvendo un **LP**.

Definizione 10.8: feasibility pump

La *feasibility pump* costruisce due successioni di punti: punti $\tilde{x}^0, \tilde{x}^1, \tilde{x}^2, \dots$ di $\mathcal{L}$, che soddisfano i vincoli lineari, e punti interi $\bar{x}^0, \bar{x}^1, \bar{x}^2, \dots$, che soddisfano i vincoli di integralità. Parte dalla soluzione LP $\tilde{x}^0$; ogni punto intero è l'*arrotondamento* dell'ultimo punto di $\mathcal{L}$, e ogni punto di $\mathcal{L}$ è la *proiezione* dell'ultimo punto intero, cioè il punto di $\mathcal{L}$ a distanza minima da esso. Si ferma quando la distanza è 0: i due punti coincidono e sono una soluzione ammissibile. Quando l'arrotondamento restituisce un punto intero già visitato, la pompa lo *perturba*.

- Lo pseudocodice della feasibility pump è il seguente:

Algoritmo

Algorithm: Feasibility pump

 Input: MIP (10.1); numero massimo di iterazioni $maxIter$; seme dei numeri casuali

Output: Soluzione ammissibile, oppure FALLIMENTO

```

1 Function FeasibilityPump(MIP, maxIter, seed)
2    $\tilde{x} :=$  soluzione ottima del rilassamento lineare
3   if  $\tilde{x}_j \in \mathbb{Z}$  per ogni  $j \in \mathcal{D}$  then
4     | return  $\tilde{x}$ 
5   end
6    $\bar{x} \leftarrow \lfloor \tilde{x} \rfloor$  // arrotonda le componenti intere
7   for  $it := 1$  to  $maxIter$  do
8      $\tilde{x} \leftarrow \arg \min \{ \Delta(x, \bar{x}) : x \in \mathcal{L} \}$  // proiezione: un LP
9     if  $\Delta(\tilde{x}, \bar{x}) = 0$  then
10      | return  $\tilde{x}$  //  $\tilde{x}$  è intero su  $\mathcal{D}$ : ammissibile
11    end
12     $\hat{x} \leftarrow \lfloor \tilde{x} \rfloor$ 
13    if  $\hat{x}_j = \bar{x}_j$  per ogni  $j \in \mathcal{D}$  then // ciclo corto
14      |  $j \leftarrow \arg \max_{k \in \mathcal{D}} |\tilde{x}_k - \bar{x}_k|$ 
15      | sposta  $\hat{x}_j$  di un'unità verso  $\tilde{x}_j$  // flip
16    end
17    if  $\hat{x}$  coincide con uno degli ultimi 3 punti  $\bar{x}$  then // ciclo lungo
18      | for  $j \in \mathcal{D}$  do
19        |  $\rho_j \leftarrow$  numero casuale uniforme in  $[-0.3, 0.7]$ 
20        | if  $|\tilde{x}_j - \hat{x}_j| + \max\{\rho_j, 0\} > \frac{1}{2}$  then
21          |  $\hat{x}_j \leftarrow 1 - \hat{x}_j$  // restart
22        | end
23      | end
24    end
25     $\bar{x} \leftarrow \hat{x}$ 
26  end
27  return FALLIMENTO
    
```

- **Perché la pompa funziona.** Ogni proiezione avvicina il punto di $\mathcal{L}$ all'ultimo punto intero, e ogni arrotondamento porta il punto intero su quello più vicino. La distanza fra le due successioni tende a diminuire, e i punti vengono *pompato* verso una regione in cui $\mathcal{L}$ contiene punti interi. Ogni iterazione costa un LP e un arrotondamento.
- **Cicli corti.** Se l'arrotondamento della proiezione restituisce lo stesso punto intero, la pompa ripeterebbe all'infinito gli stessi due punti. Cambia allora le componenti di $\bar{x}$ su cui $\tilde{x}$ è più incerto, quelle con il $|\tilde{x}_j - \bar{x}_j|$ più grande.
- La pompa originale fa il flip di un numero casuale di componenti compreso fra 10 e 30. Sulle piccole istanze di queste note basta un flip: il codice cambia la sola componente con il $|\tilde{x}_j - \bar{x}_j|$ più grande (la prima, in caso di pareggio).

- **Cicli lunghi.** La pompa può anche ciclare fra più punti interi. Quando il nuovo punto intero coincide con uno degli ultimi, la pompa originale esegue un **restart**: una componente binaria viene invertita quando $|\tilde{x}_j - \hat{x}_j| + \max\{\rho_j, 0\} > \frac{1}{2}$, con ρ_j casuale uniforme in $[-0.3, 0.7]$, così che le componenti in cui $\tilde{x}_j$ è lontano da $\hat{x}_j$ vengano invertite con alta probabilità.
- **Numeri casuali.** Nei codici di questo capitolo, i numeri casuali sono prodotti dal generatore *minimal standard* di Park e Miller, con aritmetica intera a 64 bit:

$$seed \leftarrow (16807 \cdot seed) \bmod (2^{31} - 1), \quad u = \frac{seed}{2^{31} - 1} \in (0, 1),$$

e nella pompa $\rho_j = u - 0.3$. Fissare il seme iniziale rende ogni esecuzione riproducibile.

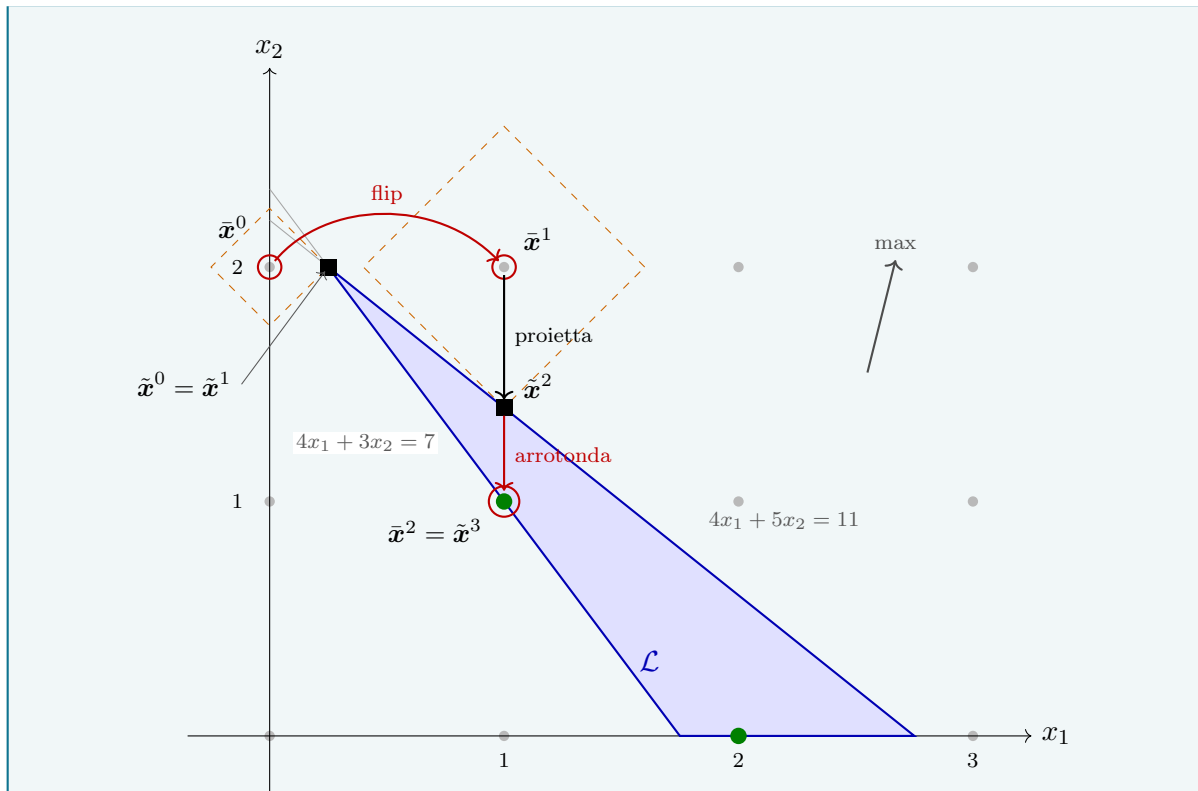
- **La funzione obiettivo.** Dopo il primo LP, la funzione obiettivo originale f viene ignorata: la pompa trova una soluzione ammissibile, ma non necessariamente buona. La *objective feasibility pump* di Achterberg e Berthold [1] minimizza una combinazione $(1 - \alpha) \Delta(\mathbf{x}, \bar{\mathbf{x}}) + \alpha f(\mathbf{x})$ con un peso α che diminuisce a ogni iterazione: le soluzioni trovate sono molto migliori, al prezzo di qualche iterazione in più.
- **Variabili intere generali.** La pompa è nata per variabili binarie, dove il flip cambia $\bar{x}_j$ in $1 - \bar{x}_j$; per variabili intere generali sposta $\bar{x}_j$ di un'unità verso $\tilde{x}_j$, e la distanza usa le variabili ausiliarie d_j [2].

Esempio 10.6: La feasibility pump in due variabili

Consideriamo il programma intero

$$\begin{aligned} \max \quad & x_1 + 4x_2 \\ \text{soggetto a} \quad & 4x_1 + 3x_2 \geq 7 \\ & 4x_1 + 5x_2 \leq 11 \\ & x_1 \in \{0, 1, \dots, 5\} \\ & x_2 \in \{0, 1, \dots, 5\} \end{aligned}$$

La regione $\mathcal{L}$ è il triangolo di vertici $(\frac{1}{4}, 2)$, $(\frac{7}{4}, 0)$, $(\frac{11}{4}, 0)$, che contiene solo due punti interi, $(1, 1)$ e $(2, 0)$. La figura mostra le due successioni di punti: quadrati per i punti di $\mathcal{L}$, cerchi per i punti interi, e i rombi $\{\mathbf{x} : \Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq r\}$ delle due proiezioni.



Le iterazioni sono le seguenti.

k	punto intero	proiezione $\tilde{x}^k$	$\Delta(\tilde{x}^k, \bar{x}^{k-1})$	punto intero successivo
0	—	ottimo LP $(\frac{1}{4}, 2)$	—	arrotondamento: $\bar{x}^0 = (0, 2)$
1	$\bar{x}^0 = (0, 2)$	$(\frac{1}{4}, 2)$	$\frac{1}{4}$	$[\tilde{x}^1] = \bar{x}^0$: flip di x_1 , $\bar{x}^1 = (1, 2)$
2	$\bar{x}^1 = (1, 2)$	$(1, \frac{7}{5})$	$\frac{3}{5}$	arrotondamento: $\bar{x}^2 = (1, 1)$
3	$\bar{x}^2 = (1, 1)$	$(1, 1)$	0	stop: ammissibile

Iterazione 0. L'ottimo LP è il vertice $(\frac{1}{4}, 2)$, di valore $\frac{33}{4}$. Il suo arrotondamento $(0, 2)$ viola il primo vincolo: $4 \cdot 0 + 3 \cdot 2 = 6 < 7$.

Iterazione 1. Con $\bar{x}^0 = (0, 2)$, x_1 è al suo limite inferiore e $\Delta(\bar{x}, \bar{x}^0) = x_1 + |x_2 - 2|$. La distanza L_1 di un punto da un segmento che non è né orizzontale, né verticale, né di pendenza ± 1 è minima in un estremo o dove il segmento incontra la retta orizzontale o quella verticale passante per il punto; i candidati sono quindi i tre vertici e queste intersezioni. La retta orizzontale $x_2 = 2$ tocca $\mathcal{L}$ solo nel vertice $(\frac{1}{4}, 2)$, e la retta verticale $x_1 = 0$ non lo incontra; il punto più vicino è $\tilde{x}^1 = (\frac{1}{4}, 2)$, a distanza $\frac{1}{4}$. Il suo arrotondamento è di nuovo $(0, 2) = \bar{x}^0$: un ciclo corto. La componente con il $|\tilde{x}_j^1 - \bar{x}_j^0|$ più grande è x_1 ($\frac{1}{4}$ contro 0), che si sposta di un'unità verso $\frac{1}{4}$: $\bar{x}^1 = (1, 2)$.

Iterazione 2. Il punto $(1, 2)$ viola il secondo vincolo ($14 > 11$). La retta verticale $x_1 = 1$ attraversa $\mathcal{L}$ fra $(1, 1)$ e $(1, \frac{7}{5})$, a distanze 1 e $\frac{3}{5}$ da $(1, 2)$; la retta orizzontale $x_2 = 2$ dà $(\frac{1}{4}, 2)$, a distanza $\frac{3}{4}$; gli altri due vertici sono più lontani. Quindi $\tilde{x}^2 = (1, \frac{7}{5})$, e il suo arrotondamento $(1, 1)$ è un punto nuovo.

Iterazione 3. Il punto $(1, 1)$ appartiene a $\mathcal{L}$ ($4 + 3 = 7 \geq 7$ e $4 + 5 = 9 \leq 11$): la sua distanza da $\mathcal{L}$ è 0, e la pompa restituisce $(1, 1)$, di valore 5. È ottima, perché l'altro punto intero $(2, 0)$ ha valore 2.

Esempio 10.7: La feasibility pump sull'esempio dell'auto e su P72

Esempio dell'auto. L'arrotondamento della soluzione LP dà $\bar{x}^0 = (4, 3)$, che è inammissibile.

k	proiezione $\tilde{x}^k$	$\Delta(\tilde{x}^k, \bar{x}^{k-1})$	punto intero successivo
1	$(\frac{7}{2}, \frac{5}{2})$	1	$[\tilde{x}^1] = \bar{x}^0$: flip di x_1 , $\bar{x}^1 = (3, 3)$
2	$(3, \frac{37}{14})$	$\frac{5}{14}$	$[\tilde{x}^2] = \bar{x}^1$: flip di x_2 , $\bar{x}^2 = (3, 2)$
3	$(3, 2)$	0	stop: ammissibile

All'iterazione 1 le due componenti di $\tilde{x}^1$ sono alla stessa distanza $\frac{1}{2}$ da $\bar{x}^0$, e il flip prende la prima. La pompa ha bisogno di **due flip** e restituisce $(3, 2)$, di valore 13: è ammissibile, ma non ottima ($z(\text{ILP}) = 14$), perché dopo il primo LP la funzione obiettivo viene ignorata.

P72. L'arrotondamento e il diving falliscono (Esempi 10.2, 10.5). La pompa arrotonda la soluzione LP a $\bar{x}^0 = (1, 0, 2, 3, 1)$, che viola il primo vincolo, e ha bisogno di tre proiezioni e di un flip (di x_3 , da 2 a 1) per arrivare alla soluzione ammissibile

$$\bar{x} = (0, 0, 1, 3, 1), \quad f(\bar{x}) = 2,$$

molto lontana da $z(\text{ILP}) = 7$ e dall'upper bound 8.

- La pompa è un metodo per trovare una *prima* soluzione ammissibile, e lo fa bene. La qualità della soluzione è compito dei metodi di miglioramento delle prossime sezioni, che partono da una soluzione incumbent.

10.6 Migliorare l'incumbent con un sotto-MIP

- I metodi delle sezioni precedenti costruiscono una soluzione a partire dal rilassamento lineare. I tre metodi di questa sezione partono da una soluzione **incumbent** $\bar{x}$, calcolata con una qualunque euristica — arrotondamento, diving, feasibility pump — e cercano di migliorarla.
- Seguono tutti lo stesso schema: aggiungono al MIP alcuni vincoli che lo restringono a un insieme di soluzioni *legate a* $\bar{x}$, e risolvono il problema ristretto, detto **sotto-MIP**, con il solver MIP.
- Il sotto-MIP è un intorno di $\bar{x}$, ma molto ampio, esplorato in modo esatto dal solver e non per enumerazione. È l'idea della **large neighbourhood search** (ricerca in un intorno ampio), con il solver come metodo di riparazione.
- Interessano solo le soluzioni *migliori* dell'incumbent. Con un passo di miglioramento $\theta > 0$, il sotto-MIP contiene il **vincolo di cutoff**

$$f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - \theta \quad (\text{minimizzazione}), \quad f(\mathbf{x}) \geq f(\bar{\mathbf{x}}) + \theta \quad (\text{massimizzazione}).$$

- Quando f assume valori interi su tutte le soluzioni ammissibili, come in tutti gli esempi di questo capitolo, $\theta = 1$ esclude esattamente le soluzioni che non sono migliori di $\bar{\mathbf{x}}$. Se il sotto-MIP con il cutoff è inammissibile, l'intorno non contiene soluzioni migliori: il solver ha *dimostrato* che $\bar{\mathbf{x}}$ è un ottimo locale per quell'intorno.
- I metodi differiscono per *quali* vincoli definiscono l'intorno: RINS fissa variabili (**fissaggio rigido**, hard fixing), il local branching e la proximity search limitano il numero di variabili che cambiano (**fissaggio morbido**, soft fixing).

10.6.1 RINS

Definizione 10.9: RINS

RINS (Relaxation Induced Neighbourhood Search, ricerca nell'intorno indotto dal rilassamento) [3] confronta l'incumbent $\bar{\mathbf{x}}$ con la soluzione $\tilde{\mathbf{x}}$ del rilassamento lineare. Le variabili intere su cui le due soluzioni *concordano* vengono fissate al loro valore comune; il sotto-MIP sulle variabili rimanenti, con il vincolo di cutoff, viene risolto dal solver MIP.

- Lo pseudocodice di RINS è il seguente:

Algoritmo

Algorithm: RINS

Input: MIP (10.1); incumbent $\bar{\mathbf{x}}$; passo di miglioramento θ

Output: Una soluzione buona almeno quanto $\bar{\mathbf{x}}$

```

1 Function RINS(MIP,  $\bar{\mathbf{x}}$ ,  $\theta$ )
2    $\tilde{\mathbf{x}} :=$  soluzione ottima del rilassamento lineare
3   for  $j \in \mathcal{D}$  do
4     if  $\tilde{x}_j = \bar{x}_j$  then
5       | fissa  $x_j := \bar{x}_j$                                 // LP e incumbent concordano
6     end
7   end
8   aggiungi il vincolo di cutoff  $f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - \theta$ 
9   risolvi il sotto-MIP con il solver MIP (con un limite sui nodi)
10  if il sotto-MIP ha una soluzione ammissibile  $\mathbf{x}'$  then
11    | return  $\mathbf{x}'$                                            // migliore di  $\bar{\mathbf{x}}$ 
12  else
13    | return  $\bar{\mathbf{x}}$                                            // nessuna soluzione migliore nell'intorno
14  end
```

- **Perché l'accordo.** La soluzione LP è ottimista e forse inammissibile, l'incumbent è ammissibile e forse scadente: due consiglieri con difetti opposti. Dove concordano, il valore

è probabilmente giusto anche nelle buone soluzioni; dove non concordano, c'è qualcosa da decidere, e la decisione è lasciata al solver.

- **L'ampiezza dell'intorno non si sceglie.** È il numero di variabili su cui $\tilde{x}$ e $\bar{x}$ non concordano: piccolo se l'incumbent è già vicino alla soluzione LP, grande altrimenti. Per questo il sotto-MIP si risolve con un limite sul numero di nodi.
- **RINS dentro il branch-and-bound.** Danna, Rothberg e Le Pape applicano RINS nei nodi dell'albero di branch-and-bound, con la soluzione LP del nodo: nodi più profondi hanno soluzioni LP diverse, quindi intorni diversi dello stesso incumbent. Ogni solver MIP moderno esegue RINS in questo modo.
- **Fissaggio rigido.** Una variabile fissata non può cambiare nel sotto-MIP. Se una soluzione migliore richiede di cambiare una variabile su cui $\tilde{x}$ e $\bar{x}$ concordano, RINS non può trovarla.

Esempio 10.8: RINS su $P7$ e su $P42$

$P7$. L'incumbent è $\bar{x} = (2, 0)$, di valore 6 (arrotondamento per difetto), e la soluzione LP è $\tilde{x} = (\frac{5}{2}, 0)$. Concordano su $x_2 = 0$, che viene fissata. Il sotto-MIP

$$\max 3x_1 \quad \text{soggetto a} \quad x_1 \leq 12, \quad 2x_1 \leq 5, \quad 3x_1 \geq 7, \quad x_1 \in \mathbb{Z}_{\geq 0}$$

è inammissibile: il secondo vincolo dà $x_1 \leq 2$, il cutoff $x_1 \geq 3$. RINS restituisce l'incumbent. La soluzione ottima $(2, 1)$ ha $x_2 = 1$, ma x_2 è fissata a 0 perché l'LP e l'incumbent concordano su di essa.

$P42$. L'incumbent è $\bar{x} = (2, 1, 0)$, di valore 8 (arrotondamento per difetto), e $\tilde{x} = (\frac{5}{2}, \frac{3}{2}, 0)$. Concordano solo su $x_3 = 0$. Il sotto-MIP su x_1, x_2 con il cutoff $3x_1 + 2x_2 \geq 9$ viene risolto da Gurobi, che restituisce $(2, 2, 0)$, di valore 10: **RINS trova una soluzione ottima.**

- Su $P7$ la variabile che deve cambiare è proprio quella su cui l'LP e l'incumbent concordano; su $P42$ l'accordo riguarda una variabile che non deve cambiare, e l'intorno contiene l'ottimo.

10.6.2 Local branching

- RINS decide *quali* variabili possono cambiare. Il local branching [5] decide solo *quante* possono cambiare, e lascia al solver la scelta di quali. Per un MIP le cui variabili intere sono binarie (oppure, con le variabili ausiliarie d_j della distanza, intere generali), il **vincolo di local branching** di raggio $k \in \mathbb{Z}_{\geq 1}$ è

$$\Delta(\mathbf{x}, \bar{\mathbf{x}}) = \sum_{j \in \mathcal{D}: \bar{x}_j=1} (1 - x_j) + \sum_{j \in \mathcal{D}: \bar{x}_j=0} x_j \leq k$$

- $\Delta(\mathbf{x}, \bar{\mathbf{x}})$ è la distanza della Definizione 10.7 per variabili binarie: sui punti binari conta le variabili il cui valore differisce da quello dell'incumbent. Il vincolo è lineare, quindi si può aggiungere al MIP come qualunque altro vincolo.

Definizione 10.10: local branching

Il *local branching* risolve, con il solver MIP, il sotto-MIP formato dal MIP, dal vincolo di cutoff e dal vincolo di local branching $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$. Se trova una soluzione migliore, la nuova soluzione diventa l'incumbent e l'intorno viene centrato su di essa. Se dimostra che l'intorno non contiene soluzioni migliori, il raggio k viene aumentato. In entrambi i casi l'intorno esplorato viene escluso dai sotto-MIP successivi con il vincolo *inverso* $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \geq k + 1$.

Algoritmo

Algorithm: Local branching

Input: MIP (10.1); incumbent $\bar{\mathbf{x}}$; raggio k ; raggio massimo $k_{\max}$; passo k_{step} ; passo di miglioramento θ

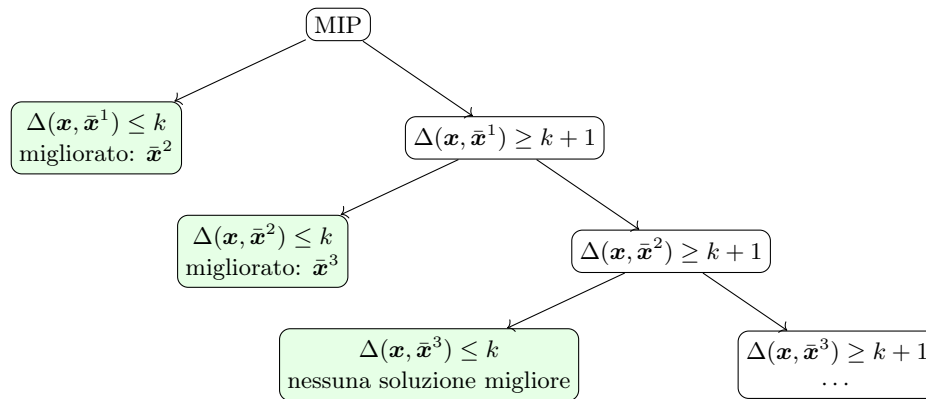
Output: Una soluzione buona almeno quanto $\bar{\mathbf{x}}$

```

1 Function LocalBranching(MIP,  $\bar{\mathbf{x}}$ ,  $k$ ,  $k_{\max}$ ,  $k_{\text{step}}$ ,  $\theta$ )
2    $\mathcal{R} := \emptyset$  // vincoli inversi aggiunti finora
3   while  $k \leq k_{\max}$  do
4     risolvi il MIP con i vincoli  $\mathcal{R}$ ,  $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$  e  $f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - \theta$ 
5      $\mathcal{R} \leftarrow \mathcal{R} \cup \{\Delta(\mathbf{x}, \bar{\mathbf{x}}) \geq k + 1\}$  // non esplorarlo mai più
6     if si trova una soluzione ammissibile  $\mathbf{x}'$  then
7        $\bar{\mathbf{x}} \leftarrow \mathbf{x}'$  // ricentra l'intorno
8     else
9        $k \leftarrow k + k_{\text{step}}$  // nessuna soluzione migliore entro distanza  $k$ 
10    end
11  end
12  return  $\bar{\mathbf{x}}$ 
    
```

- **Fissaggio morbido.** Con $|\mathcal{D}|$ variabili binarie, l'intorno $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$ contiene $\sum_{i=0}^k \binom{|\mathcal{D}|}{i}$ punti binari: per k piccolo è molto più piccolo dell'intero spazio, ma non si vincola a nessuna variabile in particolare. Possono cambiare k variabili qualsiasi.

- **Uno schema di branching.** Il nome viene dall'uso originale del vincolo come *regola di branching*: si risolve prima il ramo sinistro $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$, e il ramo destro $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \geq k + 1$ contiene il resto del problema. Se il ramo sinistro migliora l'incumbent, il ramo destro viene suddiviso di nuovo attorno al nuovo incumbent; l'albero è una lunga catena di rami destri:



- **Il raggio.** k deve essere abbastanza piccolo perché il sotto-MIP si risolva in fretta, e abbastanza grande perché l'intorno contenga soluzioni migliori. Il metodo originale usa valori piccoli rispetto a $|\mathcal{D}|$, un limite di tempo su ogni sotto-MIP, e una *diversificazione* — un k più grande, o un salto verso una soluzione diversa — quando non trova miglioramenti.
- **Local branching e ricerca locale.** Con $k = 1$ l'intorno contiene i punti ottenuti da $\bar{\mathbf{x}}$ cambiando di un'unità una sola variabile intera (una variabile binaria viene invertita); con k più grande contiene i punti che differiscono da $\bar{\mathbf{x}}$ in più variabili, o di più unità. È una ricerca locale il cui intorno viene esplorato in modo esatto dal solver.

Esempio 10.9: Local branching su $P7$ e su $P72$

$P7$. Si parte da $\bar{\mathbf{x}}^1 = (2, 0)$, di valore 6, con $k = 1$, $k_{\max} = 3$, $k_{\text{step}} = 1$ e $\theta = 1$.

$k = 1$. Il sotto-MIP con $\Delta(\mathbf{x}, \bar{\mathbf{x}}^1) \leq 1$ e il cutoff $3x_1 + x_2 \geq 7$ restituisce $(2, 1)$, di valore 7: il nuovo incumbent $\bar{\mathbf{x}}^2$, trovato a distanza 1 (la variabile x_2 cambia di un'unità). Si aggiunge $\Delta(\mathbf{x}, \bar{\mathbf{x}}^1) \geq 2$.

$k = 1, 2, 3$, **attorno a $\bar{\mathbf{x}}^2$** . Nessuna soluzione di valore almeno 8: k diventa $4 > k_{\max}$ e il metodo si ferma con $(2, 1)$, l'ottimo. Dove RINS falliva, il local branching lascia al solver la scelta di cambiare x_2 .

$P72$. Si parte dalla soluzione della feasibility pump, $\bar{\mathbf{x}}^1 = (0, 0, 1, 3, 1)$, di valore 2, con gli stessi parametri.

k	attorno a	soluzione restituita	valore
1	$\bar{x}^1$	$\bar{x}^2 = (0, 0, 1, 2, 1)$, a distanza 1	4
1	$\bar{x}^2$	nessuna soluzione di valore ≥ 5	—
2	$\bar{x}^2$	$\bar{x}^3 = (0, 1, 1, 2, 0)$, a distanza 2	7
2, 3	$\bar{x}^3$	nessuna soluzione di valore ≥ 8	—

Con $k_{\max} = 3$ il metodo si ferma con valore 7, l'ottimo, dopo due miglioramenti. Il raggio passa da 1 a 2 perché il miglioramento richiede di cambiare insieme x_2 e x_5 , cioè due unità di distanza.

- Il local branching esegue una ricerca locale con un intorno la cui ampiezza è controllata da k , esplorato in modo esatto dal solver, e allarga l'intorno quando resta bloccato: è proprio l'idea della variable neighbourhood search, con il solver che esegue l'esplorazione.

10.6.3 Proximity search

- Il local branching mantiene la funzione obiettivo e vincola la distanza. La **proximity search** di Fischetti e Monaci [6] fa il contrario: mantiene il cutoff come vincolo e *minimizza* la distanza.

Definizione 10.11: proximity search

Il sotto-MIP di prossimità attorno all'incumbent $\bar{x}$ si scrive

$$\begin{aligned}
 & \min \quad \Delta(\mathbf{x}, \bar{\mathbf{x}}) \\
 & \text{soggetto a} \quad A\mathbf{x} \leq b, \\
 & \quad l_j \leq x_j \leq u_j, \quad \forall j \in \{1, 2, \dots, N\}, \\
 & \quad f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - \theta, \\
 & \quad x_j \in \mathbb{Z}, \quad \forall j \in \mathcal{D}.
 \end{aligned}$$

La *proximity search* lo risolve, sostituisce l'incumbent con la sua soluzione e ripete, finché il sotto-MIP di prossimità non diventa inammissibile.

Algoritmo

Algorithm: Proximity search

Input: MIP (10.1); incumbent $\bar{x}$; passo di miglioramento θ

Output: Una soluzione buona almeno quanto $\bar{x}$

```

1 Function ProximitySearch(MIP,  $\bar{x}$ ,  $\theta$ )
2   while vero do
3     risolvi  $\min\{\Delta(x, \bar{x}) : x \text{ ammissibile per il MIP, } f(x) \leq f(\bar{x}) - \theta\}$ 
4     if il sotto-MIP di prossimità è inammissibile then
5       return  $\bar{x}$  // nessuna soluzione migliore di  $f(\bar{x}) - \theta$ 
6     end
7      $x' :=$  la sua soluzione ottima
8     fissa  $x_j := x'_j$  per  $j \in \mathcal{D}$  e riottimizza  $f$  sulle variabili continue
9      $\bar{x} \leftarrow x'$  // la soluzione migliorante più vicina
10  end

```

- **Perché minimizzare la distanza.** Spesso è proprio la funzione obiettivo originale a rendere difficile un MIP: il suo rilassamento lineare spinge il solver verso punti frazionari lontani dalle soluzioni ammissibili. Nel sotto-MIP di prossimità la funzione obiettivo è la distanza, e la soluzione LP resta vicina a $\bar{x}$, che è quasi ammissibile: le euristiche del solver funzionano bene, e di solito si trova in fretta una soluzione ammissibile.
- In pratica il sotto-MIP di prossimità non si risolve all'ottimo: qualunque soluzione ammissibile è un miglioramento.
- **Nessun raggio da scegliere.** L'ampiezza della mossa la decide il sotto-MIP stesso: il più piccolo cambiamento che migliora la funzione obiettivo di almeno θ .
- **Riottimizzare la parte continua.** La soluzione del sotto-MIP ha la parte intera giusta, ma le sue variabili continue sono state scelte per minimizzare la distanza, non f . Fissando le variabili intere e risolvendo l'LP sulle variabili continue si ottiene il vero valore del nuovo incumbent.
- **Il passo θ .** Un θ grande chiede miglioramenti grandi, quindi mosse ampie e sotto-MIP difficili; un θ piccolo dà sotto-MIP facili e molte iterazioni. Con $\theta = 1$ e valori interi della funzione obiettivo, il metodo dimostra anche l'ottimalità.

Proposizione 10.12

Se f assume valori interi su tutte le soluzioni ammissibili del MIP e il sotto-MIP di prossimità attorno a $\bar{x}$ con $\theta = 1$ è inammissibile, allora $f(\bar{x}) = z(I)$.

Dimostrazione. La regione ammissibile del sotto-MIP di prossimità è l'insieme delle soluzioni ammissibili $\mathbf{x}$ del MIP con $f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - 1$; non dipende dalla funzione obiettivo Δ . Se è vuota, ogni soluzione ammissibile ha $f(\mathbf{x}) > f(\bar{\mathbf{x}}) - 1$, quindi $f(\mathbf{x}) \geq f(\bar{\mathbf{x}})$ perché entrambi i valori sono interi. Dunque $\bar{\mathbf{x}}$ è ottima. $\square$

Esempio 10.10: Proximity search su $P7$ e su $P72$

$P7$. Da $\bar{\mathbf{x}} = (2, 0)$, di valore 6: la soluzione più vicina di valore almeno 7 è $(2, 1)$, a distanza 1. Attorno a essa, nessuna soluzione ha valore almeno 8 (il bound LP è $\frac{15}{2}$): per la Proposizione 10.12, $(2, 1)$ è **ottima**.

$P72$. Dalla soluzione della feasibility pump, di valore 2:

iterazione	cutoff	min Δ	soluzione restituita	valore
1	$f(\mathbf{x}) \geq 3$	1	$(0, 0, 1, 2, 1)$	4
2	$f(\mathbf{x}) \geq 5$	2	$(0, 1, 1, 2, 0)$	7
3	$f(\mathbf{x}) \geq 8$	—	inammissibile	—

Alla terza iterazione il sotto-MIP è inammissibile, e la Proposizione 10.12 dimostra che 7 è ottimo, mentre l'upper bound $\lfloor z(\text{LP}) \rfloor = 8$ della radice non basta a dimostrarlo.

- La dimostrazione di ottimalità viene dal solver, che dimostra l'inammissibilità dell'ultimo sotto-MIP con i propri bound duali: la proximity search, usata con $\theta = 1$, è un metodo esatto che migliora l'incumbent strada facendo.

10.7 Relax-and-fix e fix-and-optimize

- Le variabili di molti MIP si presentano in **blocchi** con un ordine naturale, per esempio perché sono indicizzate da una sequenza, e le decisioni di un blocco interagiscono soprattutto con quelle dei blocchi vicini. I due metodi di questa sezione sfruttano una partizione delle variabili intere in blocchi: risolvono una sequenza di MIP piccoli invece di uno grande.
- Una volta fissate le variabili intere, il resto è un LP: le decisioni difficili sono le variabili intere, e il rilassamento lineare è lo strumento che stima l'effetto delle decisioni non ancora prese.

10.7.1 Relax-and-fix

Definizione 10.13: relax-and-fix

Il *relax-and-fix* suddivide le variabili intere in blocchi $B_1, B_2, \dots, B_p$, in un ordine dato. Al passo r risolve il MIP in cui le variabili di B_r sono intere, quelle di $B_1, \dots, B_{r-1}$ sono fissate ai valori decisi nei passi precedenti, e quelle di $B_{r+1}, \dots, B_p$ sono *rilassate* (continue). Poi fissa le variabili di B_r ai loro valori e passa al blocco successivo. La soluzione dell'ultimo passo è ammissibile.

Algoritmo

Algorithm: Relax-and-fix

Input: MIP (10.1); partizione $B_1, B_2, \dots, B_p$ di $\mathcal{D}$, in un ordine dato

Output: Soluzione ammissibile, oppure FALLIMENTO

```

1 Function RelaxAndFix(MIP,  $B_1, \dots, B_p$ )
2   for  $r := 1$  to  $p$  do
3     risolvi il MIP con:  $x_j$  fissata per  $j \in B_1 \cup \dots \cup B_{r-1}$ ,  $x_j$  intera per  $j \in B_r$ ,  $x_j$ 
       continua per  $j \in B_{r+1} \cup \dots \cup B_p$ 
4     if è inammissibile then
5       return FALLIMENTO
6     end
7      $x' :=$  la sua soluzione ottima
8     fissa  $x_j := x'_j$  per ogni  $j \in B_r$ 
9   end
10  return  $x'$                                 // tutte le variabili intere sono intere

```

- **Perché.** Le decisioni dei primi blocchi si prendono con cura (interi), mentre le altre sono solo stimate (rilassate). Ogni MIP ha le variabili intere di un solo blocco, quindi è piccolo.
- **Miopia.** I blocchi rilassati sono ottimisti: il rilassamento può comprare una frazione di decisione per una frazione del suo costo. Una decisione che sembra buona perché il resto costa poco può rivelarsi cattiva quando il resto diventa intero. Le decisioni fissate possono perfino rendere inammissibile un passo successivo.
- **Varianti.** Blocchi più grandi danno soluzioni migliori e MIP più difficili; con blocchi *sovrapposti* si risolve $B_r \cup B_{r+1}$ come intero ma si fissa solo B_r , per vedere un po' più lontano.

Esempio 10.11: Relax-and-fix su P_{46}

Le variabili intere sono x_1, x_2, x_3 . Il rilassamento lineare ha valore $\frac{499}{25}$, l'upper bound è 19 e $z(\text{ILP}) = 16$, raggiunto da $(0, 1, 1)$.

Una variabile per blocco, $B_1 = \{x_1\}$, $B_2 = \{x_2\}$, $B_3 = \{x_3\}$.

passo r	valore del MIP del passo r	soluzione del MIP	fissaggio
1	$\frac{296}{15} \approx 19.73$	$(0, \frac{26}{15}, \frac{14}{15})$	$x_1 := 0$
2	$\frac{56}{3} \approx 18.67$	$(0, 2, \frac{2}{3})$	$x_2 := 2$
3	12	$(0, 2, 0)$	$x_3 := 0$

Il risultato è $(0, 2, 0)$, di valore 12, con un gap di $\frac{19-12}{12} \approx 58.3\%$. La decisione sbagliata si prende al passo 2: con x_3 rilassata, $x_2 = 2$ dà $\frac{56}{3} \approx 18.67$ e $x_2 = 1$ dà $\frac{122}{7} \approx 17.43$, quindi $x_2 := 2$; ma allora il secondo vincolo lascia spazio solo per $x_3 \leq \frac{2}{3}$, e x_3 diventa 0 quando è intera.

Blocchi più grandi, $B_1 = \{x_1\}$, $B_2 = \{x_2, x_3\}$. Il passo 1 è lo stesso. Al passo 2 il MIP su x_2, x_3 si risolve con entrambe le variabili intere: restituisce $(0, 1, 1)$, di valore 16, l'ottimo.

- L'ampiezza dei blocchi bilancia la qualità della soluzione e la difficoltà dei MIP: il blocco più grande vede l'interazione fra x_2 e x_3 che i blocchi di una sola variabile non colgono.

10.7.2 Fix-and-optimize

Definizione 10.14: fix-and-optimize

La *fix-and-optimize* migliora un incumbent $\bar{x}$. Considera una sequenza di *finestre* $W_1, W_2, \dots, W_q \subseteq \mathcal{D}$, tipicamente periodi consecutivi. Per ogni finestra fissa tutte le variabili intere *fuori* dalla finestra ai loro valori in $\bar{x}$, e risolve il sotto-MIP sulle variabili della finestra, con il vincolo di cutoff. Una soluzione migliore diventa l'incumbent. Le passate sulle finestre si ripetono finché un'intera passata non porta miglioramenti.

Algoritmo

Algorithm: Fix-and-optimize

Input: MIP (10.1); incumbent $\bar{x}$; finestre $W_1, \dots, W_q$; passo di miglioramento θ

Output: Una soluzione buona almeno quanto $\bar{x}$

```

1 Function FixAndOptimize(MIP,  $\bar{x}$ ,  $W_1, \dots, W_q$ ,  $\theta$ )
2    $improved := \text{vero}$ 
3   while  $improved$  do
4      $improved \leftarrow \text{falso}$ 
5     for  $r := 1$  to  $q$  do
6       risolvi il MIP con  $x_j := \bar{x}_j$  per  $j \in \mathcal{D} \setminus W_r$  e  $f(x) \leq f(\bar{x}) - \theta$ 
7       if si trova una soluzione ammissibile  $x'$  then
8          $\bar{x} \leftarrow x'$ ;  $improved \leftarrow \text{vero}$ 
9       end
10    end
11  end
12  return  $\bar{x}$ 

```

Esempio 10.12: Fix-and-optimize su $P46$ e su $P72$

$P46$. Si parte dalla soluzione del relax-and-fix, $\bar{x} = (0, 2, 0)$, di valore 12, con le finestre $W_1 = \{x_1\}$ e $W_2 = \{x_2, x_3\}$ e $\theta = 1$.

finestra	incumbent dopo la finestra	
$\{x_1\}$	12	nessun miglioramento ($x_1 = 1$ viola il primo vincolo)
$\{x_2, x_3\}$	16	$\bar{x} = (0, 1, 1)$
seconda passata, entrambe le finestre	16	nessun miglioramento: stop

Con le tre finestre di una variabile ciascuna, $\{x_1\}$, $\{x_2\}$, $\{x_3\}$, nessuna finestra migliora $(0, 2, 0)$: è un ottimo locale per quelle finestre. La finestra $\{x_2, x_3\}$ rivede la decisione sbagliata del relax-and-fix e raggiunge l'ottimo 16.

$P72$. Si parte dalla soluzione della feasibility pump, $(0, 0, 1, 3, 1)$, di valore 2, con le finestre $\{x_1, x_2\}$, $\{x_3, x_4\}$, $\{x_5\}$. La prima finestra non porta miglioramenti; la seconda dà $(0, 0, 2, 4, 1)$, di valore 5; la terza e la seconda passata non ne portano. Il metodo si ferma con $5 < 7 = z(\text{ILP})$: un ottimo locale rispetto a queste finestre, perché la soluzione ottima $(0, 1, 1, 2, 0)$ ne differisce in x_2 , x_3 , x_4 e x_5 , più variabili di quante ne contenga una qualunque finestra.

- Il fix-and-optimize è una **ricerca locale** le cui mosse cambiano tutte insieme le variabili intere di una finestra, esplorate in modo esatto dal solver: l'equivalente, nella programmazione intera, degli intorni della ricerca locale. Si ferma in un ottimo locale rispetto a tutte le finestre.
- Relax-and-fix e fix-and-optimize sono complementari: il primo costruisce una soluzione blocco per blocco, il secondo ripara le decisioni miopi guardando più blocchi insieme.

10.8 Le euristiche dentro il solver

- Un solver MIP moderno non è solo un algoritmo branch-and-bound: esegue, in automatico, gran parte delle euristiche di questo capitolo (il capitolo «Risolutori MIP: che cosa fanno» mostra log reali). In Gurobi:
 - *prima dell'LP della radice*, euristiche che non hanno bisogno del rilassamento cercano una prima soluzione;
 - *alla radice*, dopo l'LP, vengono eseguiti l'arrotondamento, diverse euristiche di diving e la feasibility pump;
 - *durante il branch-and-bound*, RINS e altre euristiche basate su sotto-MIP vengono eseguite in alcuni nodi, con la soluzione LP del nodo, e i nuovi incumbent sono segnalati nel log da una riga che inizia con H (euristica) invece che con * (soluzione LP intera di un nodo).
- Lo sforzo dedicato alle euristiche è controllato da parametri. I più importanti sono:

parametro	default	significato
Heuristics	0.05	frazione del lavoro dedicata alle euristiche (0 minimo e massimo 1)
MIPFocus	0	1: puntare a trovare buone soluzioni ammissibili più che a migliorare il bound
RINS	-1	frequenza di RINS (ogni n nodi; -1 automatica, 0 disattivata)
PumpPasses	-1	passate della feasibility pump alla radice (-1 automatico)
SubMIPNodes	500	limite sui nodi dei sotto-MIP di RINS e di euristiche simili
NoRelHeurWork	0	lavoro dedicato all'euristica senza rilassamento prima della radice
ImproveStartGap	0	quando il gap raggiunge questo valore, si passa a migliorare solo l'incumbent

- Una soluzione euristica calcolata fuori dal solver — con uno qualunque dei metodi di questo capitolo — gli si può passare come **MIP start** (l'attributo **Start** delle variabili): il solver parte da essa come incumbent, la usa per la potatura e per RINS, e non restituisce mai niente di peggio.
- I numeri dipendono dall'istanza, dalla versione del solver e dalla macchina; una singola istanza non dimostra nulla. La lezione è qualitativa: quando il bound è forte, il tempo di risoluzione è il tempo necessario a trovare una buona soluzione, e a deciderlo sono le euristiche del solver — o un buon MIP start.

10.9 Che cosa danno le euristiche basate sulla PLI

- Un'euristica fornisce solo un lato del gap, e senza un bound duale non si può dire nulla sulla qualità della sua soluzione. Un'euristica basata sulla PLI lavora con un modello MIP, e il modello porta con sé il suo rilassamento lineare; eseguita dentro un solver MIP, ha a disposizione anche il bound duale del branch-and-bound. L'euristica **sa quanto è lontana dall'ottimo** mentre è in esecuzione: il gap di ottimalità

$$\text{gap di ottimalità} = \frac{\text{UB} - \text{LB}}{|\text{LB}|}$$

si calcola in ogni momento, e quando vale 0 la soluzione è *dimostrata* ottima.

istanza	soluzione	LB	UB = $\lfloor z(\text{LP}) \rfloor$	gap	certificata da
auto, arrotondamento per difetto	(3, 2)	13	15	15.4%	—
auto, diving	(4, 1)	14	15	7.1%	proximity search da (3, 2)
P7, arrotondamento per difetto	(2, 0)	6	7	16.7%	—
P7, local branching	(2, 1)	7	7	0%	il bound
P42, RINS	(2, 2, 0)	10	10	0%	il bound
P46, relax-and-fix	(0, 2, 0)	12	19	58.3%	—
P46, fix-and-optimize	(0, 1, 1)	16	19	18.8%	proximity search
P72, feasibility pump	(0, 0, 1, 3, 1)	2	8	300%	—
P72, local branching	(0, 1, 1, 2, 0)	7	8	14.3%	proximity search

- Su $P7$ e $P42$ l'arrotondamento per difetto del valore LP è già il valore ottimo: la difficoltà è trovare una soluzione che raggiunga il bound, non dimostrarlo. Sulle altre istanze il bound $\lfloor z(LP) \rfloor$ non è stretto, e il certificato viene dalla proximity search (Proposizione 10.12) o dal branch-and-bound del solver.

Un'euristica basata sulla PLI non si limita a trovare soluzioni: dentro un solver MIP lavora con un bound duale, quindi il gap della sua soluzione è sempre noto, e un gap uguale a 0 certifica l'ottimalità.

- I metodi di questo capitolo, in sintesi:

metodo	richiede l'LP	richiede l'incumbent	fissaggio	idea
arrotondamento	sì	no	—	arrotondare la soluzione LP
diving	sì	guidato: sì	una alla volta	fissare una variabile, risolvere di nuovo l'LP, ripetere
feasibility pump	sì	no	—	alternare arrotondamento e proiezione finché si incontrano
RINS	sì	sì	rigido	fissare dove LP e incumbent concordano, risolvere il sotto-MIP
local branching	no	sì	morbido	sotto-MIP con $\Delta(\mathbf{x}, \bar{\mathbf{x}}) \leq k$
proximity search	no	sì	morbido	minimizzare $\Delta(\mathbf{x}, \bar{\mathbf{x}})$ soggetto al cutoff
relax-and-fix	sì	no	a blocchi	intere ora, rilassate dopo, poi fissate
fix-and-optimize	no	sì	a finestre	riottimizzare una finestra, il resto fissato

10.10 Implementazione in Python

- I codici usano Python 3 e `gurobipy`. Alle variabili di un modello si accede tramite la loro posizione j in `m.getVars()`, e la lista I delle posizioni delle variabili intere è l'insieme $\mathcal{I}$; gli indici partono da 0.
- Il modulo `common.py` contiene l'ambiente del solver e le operazioni usate da tutti i metodi: il rilassamento lineare, l'arrotondamento $\lfloor \cdot \rfloor$, il completamento di un punto, la distanza Δ e il vincolo di cutoff.

```

1 import math
2 import gurobipy as gp
3 from gurobipy import GRB
4
5 EPS = 1e-6                                # integrality and
    comparison tolerance
6
7 ENV = gp.Env(empty=True)                  # one silent, single-
    thread (deterministic) environment
8 ENV.setParam("OutputFlag", 0)
9 ENV.setParam("Threads", 1)

```

```

10 ENV.start()
11
12
13 def int_index(m):
14     """positions j (in m.getVars()) of the integer and binary variables
        : the set I"""
15     return [j for j, v in enumerate(m.getVars()) if v.VType != GRB.
        CONTINUOUS]
16
17
18 def nearest(v):
19     """[v] = floor(v + 1/2): rounding to the nearest integer, 1/2
        rounded up"""
20     return math.floor(v + 0.5 + EPS)
21
22
23 def is_integral(v):
24     return abs(v - round(v)) <= EPS
25
26
27 def solve_lp(m):
28     """optimal solution of the LP relaxation of m (with its current
        bounds), None if infeasible"""
29     m.update()
30     r = m.relax()
31     r.optimize()
32     if r.Status != GRB.OPTIMAL:
33         return None
34     return [v.X for v in r.getVars()]
35
36
37 def obj_of(m, x):
38     """value of the objective function of m at the point x"""
39     e = m.getObjective()
40     return e.getConstant() + sum(e.getCoeff(k) * x[e.getVar(k).index]
        for k in range(e.size()))
41
42
43 def round_point(x_star, I):
44     """x_tilde = [x*]: round the integer components, copy the
        continuous ones"""
45     x_tilde = list(x_star)
46     for j in I:
47         x_tilde[j] = nearest(x_star[j])
48     return x_tilde
49
50
51 def evaluate(m, x_int):
52     """fix the integer variables to x_int and optimize the continuous
        ones:
        returns (value, x), or None if the point cannot be completed to
        a feasible solution"""
53     f = m.copy()

```

```

55     V = f.getVars()
56     for j in int_index(f):
57         V[j].LB = V[j].UB = x_int[j]
58     f.optimize()
59     if f.Status != GRB.OPTIMAL:
60         return None
61     return f.ObjVal, [v.X for v in V]
62
63
64 def solve_mip(m):
65     """optimal value and solution of the MIP m, None if infeasible"""
66     m.optimize()
67     if m.SolCount == 0:
68         return None
69     return m.ObjVal, [v.X for v in m.getVars()]
70
71
72 def distance(m, x_tilde, I):
73     """the linear expression Delta(x, x_tilde) = sum_{j in I} |x_j -
74     x_tilde_j| in the variables of m:
75     x_j - l_j if x_tilde_j is at its lower bound, u_j - x_j if at
76     its upper bound,
77     otherwise an auxiliary variable d_j >= |x_j - x_tilde_j|"""
78     V = m.getVars()
79     expr = gp.LinExpr()
80     for j in I:
81         if x_tilde[j] == V[j].LB:
82             expr += V[j] - x_tilde[j]
83         elif x_tilde[j] == V[j].UB:
84             expr += x_tilde[j] - V[j]
85         else:
86             d = m.addVar(name=f"d[{j}]")
87             m.addConstr(d >= V[j] - x_tilde[j])
88             m.addConstr(d >= x_tilde[j] - V[j])
89             expr += d
90     return expr
91
92 def delta(x, x_tilde, I):
93     """value of Delta(x, x_tilde)"""
94     return sum(abs(x[j] - x_tilde[j]) for j in I)
95
96 def add_cutoff(m, z, theta):
97     """require a solution better than z by at least theta"""
98     obj = m.getObjective()
99     if m.ModelSense == GRB.MINIMIZE:
100         m.addConstr(obj <= z - theta, name="cutoff")
101     else:
102         m.addConstr(obj >= z + theta, name="cutoff")
103
104
105 def note(trace, *items):

```

```

106     """record a step of the algorithm (used to print the examples of
107         the notes)"""
108     if trace is not None:
109         trace.append(items)

```

- Il modello degli esempi (models.py): $\max c^T x$ soggetto a $Ax \leq b$, con variabili intere in $\mathbb{Z}_{\geq 0}$ e limiti superiori facoltativi.

```

1 from common import *
2
3
4 def build_mip(c, A, b, ub=None, name="MIP"):
5     """max c x subject to Ax ≤ b, 0 ≤ x ≤ ub, x integer"""
6     m = gp.Model(name, env=ENV)
7     n = len(c)
8     x = m.addVars(n, lb=0, ub=GRB.INFINITY if ub is None else ub, vtype
9                 =GRB.INTEGER, name="x")
10    m.setObjective(gp.quicksum(c[j] * x[j] for j in range(n)), GRB.
11                MAXIMIZE)
12    for i in range(len(b)):
13        m.addConstr(gp.quicksum(A[i][j] * x[j] for j in range(n)) ≤ b[
14                        i], name=f"c{i}")
15    m.update()
16    return m

```

- Arrotondamento e diving:

```

1 def rounding(m, trace=None):
2     I = int_index(m)
3     x_star = solve_lp(m)                                # optimal solution of
4     note(trace, "LP", x_star)                            the LP relaxation
5     x_tilde = round_point(x_star, I)                    # x_tilde_j = [x*_j] for
6     note(trace, "rounded", x_tilde)                    every integer variable
7     return evaluate(m, x_tilde)                        # (value, x) if feasible
8                                                         , None otherwise

```

```

1 def diving(m, rule="fractional", x_inc=None, trace=None):
2     d = m.copy()                                         # the bounds of the copy
3     are changed, not those of m
4     V = d.getVars()
5     I = int_index(d)
6     while True:
7         x_star = solve_lp(d)                            # LP relaxation with the
8         if x_star is None:                                current bounds
9             note(trace, "LP infeasible")
10            return None                                    # the dive fails
11            F = [j for j in I if not is_integral(x_star[j])]

```

```

11     note(trace, "LP", obj_of(d, x_star), x_star)
12     if not F:
13         return evaluate(m, round_point(x_star, I))
14     if rule == "fractional": # the least fractional
15         variable, to its nearest integer
16         j = min(F, key=lambda k: round(abs(x_star[k] - nearest(
17             x_star[k])), 9))
18         v = nearest(x_star[j])
19     else: # guided: the variable
20         closest to the incumbent, to its value
21         j = min(F, key=lambda k: round(abs(x_star[k] - x_inc[k]),
22             9))
23         v = x_inc[j]
24     note(trace, "fix", V[j].VarName, v)
25     V[j].LB = V[j].UB = v # fix x_j := v

```

- La feasibility pump:

```

1 def projection(m, x_tilde, I):
2     """x* = argmin { Delta(x, x_tilde) : x in P }, P the LP relaxation
3     of m"""
4     p = m.relax()
5     p.setObjective(distance(p, x_tilde, I), GRB.MINIMIZE)
6     p.optimize()
7     return [v.X for v in p.getVars()][:m.NumVars]
8
9 def feasibility_pump(m, max_iter=100, seed=1, trace=None):
10     I = int_index(m)
11     x_star = solve_lp(m) # x* = optimal solution
12     of the LP relaxation
13     note(trace, "LP", x_star)
14     if all(is_integral(x_star[j]) for j in I):
15         return evaluate(m, x_star)
16     x_tilde = round_point(x_star, I) # x_tilde = [x*]
17     note(trace, "round", x_tilde)
18     history = [x_tilde] # the points x_tilde
19     visited so far
20     for it in range(1, max_iter + 1):
21         x_star = projection(m, x_tilde, I) # the point of P closest
22         to x_tilde
23         dist = delta(x_star, x_tilde, I)
24         note(trace, "project", x_star, dist)
25         if dist <= EPS: # x* = x_tilde is
26             integer and feasible
27             return evaluate(m, x_tilde)
28         x_new = round_point(x_star, I)
29         if all(x_new[j] == x_tilde[j] for j in I): # short
30             cycle: [x*] = x_tilde
31             j = max(I, key=lambda k: (round(abs(x_star[k] - x_tilde[k]),
32                 9), -k))
33             x_new[j] += 1 if x_star[j] > x_tilde[j] else -1 # flip the
34             most uncertain component

```

```

27     note(trace, "flip", x_new, j)
28     else:
29         note(trace, "round", x_new)
30         if any(all(x_new[j] == y[j] for j in I) for y in history[-3:]):
31             # long cycle
32             for j in I:
33                 # restart: random flips
34                 (binary variables)
35                 seed = (16807 * seed) % 2147483647
36                 rho = seed / 2147483647 - 0.3 # rho_j uniform in
37                 [-0.3, 0.7]
38                 if abs(x_star[j] - x_new[j]) + max(rho, 0) > 0.5:
39                     x_new[j] = 1 - x_new[j]
40             note(trace, "restart", x_new)
41         x_tilde = x_new
42         history.append(x_tilde)
43     return None # no feasible solution
44         found

```

- RINS, local branching e proximity search:

```

1 def rins(m, x_tilde, theta=1, trace=None):
2     I = int_index(m)
3     z_tilde = obj_of(m, x_tilde) # value of the incumbent
4     x_star = solve_lp(m) # optimal solution of
5     # the LP relaxation
6     s = m.copy()
7     V = s.getVars()
8     fixed = []
9     for j in I:
10         if abs(x_star[j] - x_tilde[j]) <= EPS: # LP and incumbent
11             # agree on x_j
12             V[j].LB = V[j].UB = x_tilde[j]
13             fixed.append(j)
14     note(trace, "LP", x_star, "fixed", [V[j].VarName for j in fixed])
15     add_cutoff(s, z_tilde, theta) # look only for better
16     # solutions
17     res = solve_mip(s) # the sub-MIP on the
18     # free variables
19     if res is None:
20         note(trace, "sub-MIP infeasible")
21         return z_tilde, x_tilde
22     note(trace, "sub-MIP", res[0])
23     return res

```

```

1 def local_branching(m, x_tilde, k, k_max, k_step, theta=1, trace=None):
2     I = int_index(m)
3     w = m.copy() # the master MIP, which
4     # receives the reverse constraints
5     z_tilde = obj_of(m, x_tilde)
6     while k <= k_max:
7         s = w.copy()

```

```

7      s.addConstr(distance(s, x_tilde, I) <= k, name="local_branch")
          # Delta(x, x_tilde) <= k
8      add_cutoff(s, z_tilde, theta)
9      res = solve_mip(s)
10     w.addConstr(distance(w, x_tilde, I) >= k + 1)
          # never explore it again
11     if res is not None:
          # better solution: re-
            center the neighbourhood
12         note(trace, "improved", k, res[0], res[1])
13         z_tilde, x_tilde = res
14     else:
          # no better solution
            within distance k: enlarge it
15         note(trace, "no improvement", k)
16         k += k_step
17     return z_tilde, x_tilde

```

```

1 def proximity_search(m, x_tilde, theta=1, trace=None):
2     I = int_index(m)
3     z_tilde = obj_of(m, x_tilde)
4     while True:
5         p = m.copy()
6         add_cutoff(p, z_tilde, theta)
          # better than the
          incumbent by at least theta
7         p.setObjective(distance(p, x_tilde, I), GRB.MINIMIZE)
          # as
          close as possible to it
8         res = solve_mip(p)
9         if res is None:
          # no solution better
            than z_tilde - theta exists
10            note(trace, "infeasible")
11            return z_tilde, x_tilde
12        z_new, x_new = evaluate(m, res[1])
          # value of the new
          solution, continuous part re-optimized
13        note(trace, "found", res[0], z_new, x_new)
14        z_tilde, x_tilde = z_new, x_new

```

- Relax-and-fix e fix-and-optimize:

```

1 def relax_and_fix(m, blocks, trace=None):
2     r = m.copy()
3     V = {v.VarName: v for v in r.getVars()}
4     vtype = {name: V[name].VType for B in blocks for name in B}
5     for B in blocks:
          # at first, all the
            blocks are relaxed
6         for name in B:
7             V[name].VType = GRB.CONTINUOUS
8     for B in blocks:
          # the current block is
            integer again
9         for name in B:
10            V[name].VType = vtype[name]
11        res = solve_mip(r)
12        if res is None:

```

```

13         note(trace, "infeasible")
14         return None
15     note(trace, "block", B, res[0], {n: V[n].X for BB in blocks for
16         n in BB})
17     for name in B:                                     # fix the current block
18         to its value
19         V[name].LB = V[name].UB = round(V[name].X)
20     return res

```

```

1 def fix_and_optimize(m, x_tilde, windows, theta=1, trace=None):
2     I = int_index(m)
3     names = [v.VarName for v in m.getVars()]
4     z_tilde = obj_of(m, x_tilde)
5     improved = True
6     while improved:                                     # one pass over all the
7         windows
8         improved = False
9         for W in windows:
10             s = m.copy()
11             V = s.getVars()
12             for j in I:
13                 if names[j] not in W:                   # outside the window:
14                     fixed to the incumbent
15                     V[j].LB = V[j].UB = x_tilde[j]
16             add_cutoff(s, z_tilde, theta)
17             res = solve_mip(s)
18             if res is not None:
19                 z_tilde, x_tilde = res
20                 improved = True
21             note(trace, "window", W, z_tilde)
22     return z_tilde, x_tilde

```

- Tutte le funzioni di questa sezione sono nella cartella MIP_HEUR_CODE, insieme a `instances.py`, che contiene le istanze del capitolo e degli esercizi. Il comando `python3 run_tests.py` esegue tutti gli esempi del capitolo e tutte le istanze degli esercizi, e confronta ogni valore con il valore atteso, scritto nel file.

10.11 Errori tipici

Attenzione

- **Arrotondare un problema con vincoli di uguaglianza.** Arrotondamenti indipendenti delle variabili di un vincolo di uguaglianza non lo soddisfano quasi mai: l'arrotondamento fallisce, e serve un diving.
- **Credere che un diving possa annullare una decisione.** Un diving è un unico cammino dell'albero, senza backtracking: una fissazione sbagliata non viene mai rivista, e una fissazione che rende l'LP inammissibile chiude il diving.
- **Usare la feasibility pump per ottenere una buona soluzione.** Dopo il primo LP la pompa ignora la funzione obiettivo: su *P72* restituisce una soluzione di valore 2, contro il valore ottimo 7.
- **Fidarsi del fissaggio rigido.** RINS fissa le variabili su cui l'LP e l'incumbent concordano, quindi non può trovare una soluzione migliore che ne cambi una (*P7*).
- **Dimenticare il vincolo di cutoff.** Senza $f(\mathbf{x}) \leq f(\bar{\mathbf{x}}) - \theta$ il sotto-MIP restituisce di nuovo la soluzione incumbent; con il vincolo, un sotto-MIP inammissibile dimostra un ottimo locale.
- **Dichiarare l'ottimalità senza le ipotesi.** La Proposizione 10.12 richiede $\theta = 1$ e valori interi della funzione obiettivo su tutte le soluzioni ammissibili.
- **Ignorare la miopia del relax-and-fix.** I blocchi rilassati sono ottimisti: una decisione che sembra economica può costringere più avanti a una decisione cattiva (*P46*).
- **Generalizzare da una sola istanza o da una sola esecuzione del solver.** Cammini e tempi dipendono dalla versione del solver e dalle regole di spargio, quindi le lezioni sono qualitative.

10.12 In sintesi

In sintesi

- Le euristiche basate sulla PLI usano il rilassamento lineare per decidere dove cercare e un solver MIP per esplorare in modo esatto un intorno ampio.
- L'arrotondamento e il diving costruiscono una soluzione a partire dall'LP; la feasibility pump cerca solo l'ammissibilità; RINS, il local branching e la proximity search migliorano una soluzione incumbent con un sotto-MIP.
- Il relax-and-fix costruisce una soluzione blocco per blocco, il fix-and-optimize la migliora finestra per finestra.
- Dentro un solver MIP è disponibile il bound duale: il gap della soluzione euristica è noto, e un gap uguale a 0 certifica l'ottimalità.

Cruscotto dell'esempio dell'auto. I lower bound e gli upper bound su $z(\text{ILP})$ trovati finora in questo modulo:

Capitolo	Lower bound	Upper bound	Origine
Programmazione lineare intera	12, 14	–	soluzioni euristiche (4, 0) e (4, 1)
Rilassamento lineare e bound duali	–	15	$\lfloor z(\text{LP}) \rfloor$, con $z(\text{LP}) = \frac{31}{2}$
Algoritmo dei piani di taglio	14	14	5 tagli frazionari di Gomory
Branch-and-bound basato su LP	14	14	6 problemi LP risolti
Questo capitolo, arrotondamento per difetto	13	15	(3, 2)
Questo capitolo, diving	14	15	(4, 1), certificata ottima dalla proximity search

Bibliografia

- [1] T. Achterberg, T. Berthold, Improving the feasibility pump, *Discrete Optimization* 4 (2007) 77–86.
- [2] L. Bertacco, M. Fischetti, A. Lodi, A feasibility pump heuristic for general mixed-integer problems, *Discrete Optimization* 4 (2007) 63–76.
- [3] E. Danna, E. Rothberg, C. Le Pape, Exploring relaxation induced neighborhoods to improve MIP solutions, *Mathematical Programming* 102 (2005) 71–90.
- [4] M. Fischetti, F. Glover, A. Lodi, The feasibility pump, *Mathematical Programming* 104 (2005) 91–104.
- [5] M. Fischetti, A. Lodi, Local branching, *Mathematical Programming* 98 (2003) 23–47.
- [6] M. Fischetti, M. Monaci, Proximity search for 0-1 mixed-integer convex programming, *Journal of Heuristics* 20 (2014) 709–731.