Branch-and-bound

Giovanni Righini

Ricerca Operativa

Università degli Studi
di Milano

Ottimizzazione discreta

I problemi di ottimizzazione discreta in generale sono molto difficili da risolvere perché:

- il numero di soluzioni cresce esponenzialmente con numero di variabili;
- gli strumenti del calcolo differenziale, come le derivate (utili per caratterizzare i punti di ottimo) non sono disponibili.

A causa della esplosione combinatoria del numero di soluzioni, l'enumerazione esplicita non è praticabile.

Tuttavia esistono tecniche di enumerazione implicita:

- branch-and-bound,
- programmazione dinamica.

Branch-and-bound

In un algoritmo branch-and-bound

- un problema difficile $\mathcal{P}$ viene ricorsivamente scomposto in più sotto-problemi $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n$ più facili.
  La scomposizione (branching, cioè ramificazione) deve rispettare la seguente condizione per assicurare la correttezza dell'algoritmo:

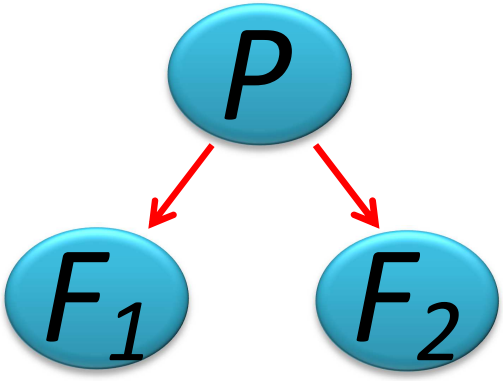
$$\mathcal{X}(\mathcal{P}) = \bigcup_{i=1}^{n} \mathcal{X}(\mathcal{F}_i).$$

- la soluzione ottima di $\mathcal{P}$ è determinata confrontando le soluzioni ottime dei sotto-problemi originati da esso.
  In caso di minimizzazione:

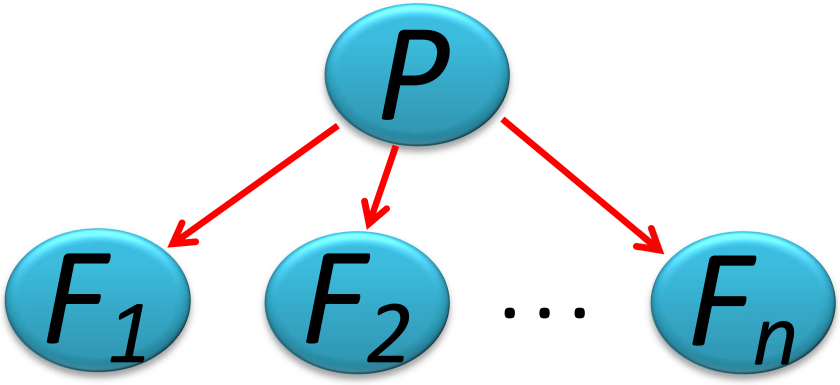
$$z^*(\mathcal{P}) = \min_{i=1,\ldots,n} \{z^*(\mathcal{F}_i)\}.$$

Il branch-and-bound tree

La scomposizione ricorsiva di problemi in sotto-problemi genera un'arborescenza (detta anche *decision tree* o *search tree*), in cui la radice corrisponde al problema originale $\mathcal{P}$ ed ogni altro nodo corrisponde ad un sotto-problema.



Branching binario



Branching *n*-ario

Branching

A scopo di efficienza, la scomposizione solitamente implica una partizione di $\mathcal{X}(\mathcal{P})$ in sottinsiemi disgiunti di modo che nessuna soluzione debba essere (implicitamente) considerata più di una volta:

$$\mathcal{X}(\mathcal{F}_i) \cap \mathcal{X}(\mathcal{F}_j) = \emptyset \qquad \forall i \neq j = 1, \ldots, n.$$

Ci sono due modi principali di fare branching:

- fissaggio di variabili;
- inserzione di vincoli.

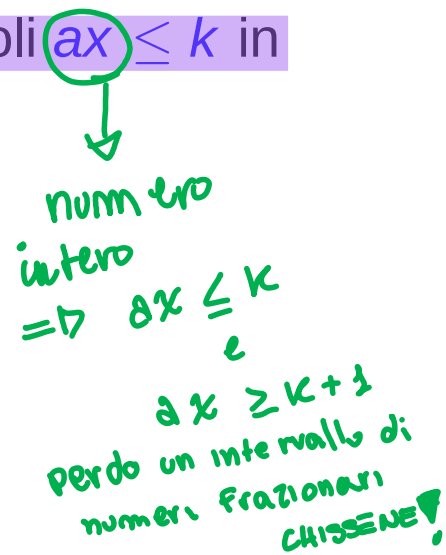
Ogni sotto-problema è una restrizione del suo predecessore ed un rilassamento dei suoi successori.

Branching binario

Regole di branching comuni sono le seguenti.

- **Branching su una variabile binaria.**
  Una variabile binaria x viene selezionata.
  Due sotto-problemi vengono generati fissando $x = 0$ in uno e $x = 1$ nell'altro.

- **Branching su un vincolo intero.**
  Vengono scelti un vettore di variabili intere $(x_1, x_2, \ldots, x_n)$, un opportuno vettore di coefficienti interi $(a_1, a_2, \ldots, a_n)$ e un opportuno termine noto intero k.
  Vengono genrati due sotto-problemi inserendo i vincoli $ax \leq k$ in uno e $ax \geq k + 1$ nell'altro.



Branching n-ario

Regole di branching n-ario sono le seguenti.

- **Branching su una variabile intera.**
  Viene selezionata una variabile intera $x \in [1, \ldots, n]$.
  Vengono generati n sotto-problemi fissando $x = 1$, $x = 2$,..., $x = n$.

- **Branching su n variabili binarie.**
  Viene scelto un vettore di n variabili binarie $(x_1, x_2, \ldots, x_n)$.
  Vengono generati $n + 1$ sotto-problemi fissando alcune variabili come segue (una riga per ogni sotto-problema):

Solve. sotto problemi sono sbilanciate ∇
(1) Fisso 1 variabile
(2) Fisso tutte le var. ∇

(1) $x_1 = 1$

$x_1 = 0, x_2 = 1$

$x_1 = x_2 = 0, x_3 = 1$

Quelle prima fisso a 0, quelle dopo lascio libere

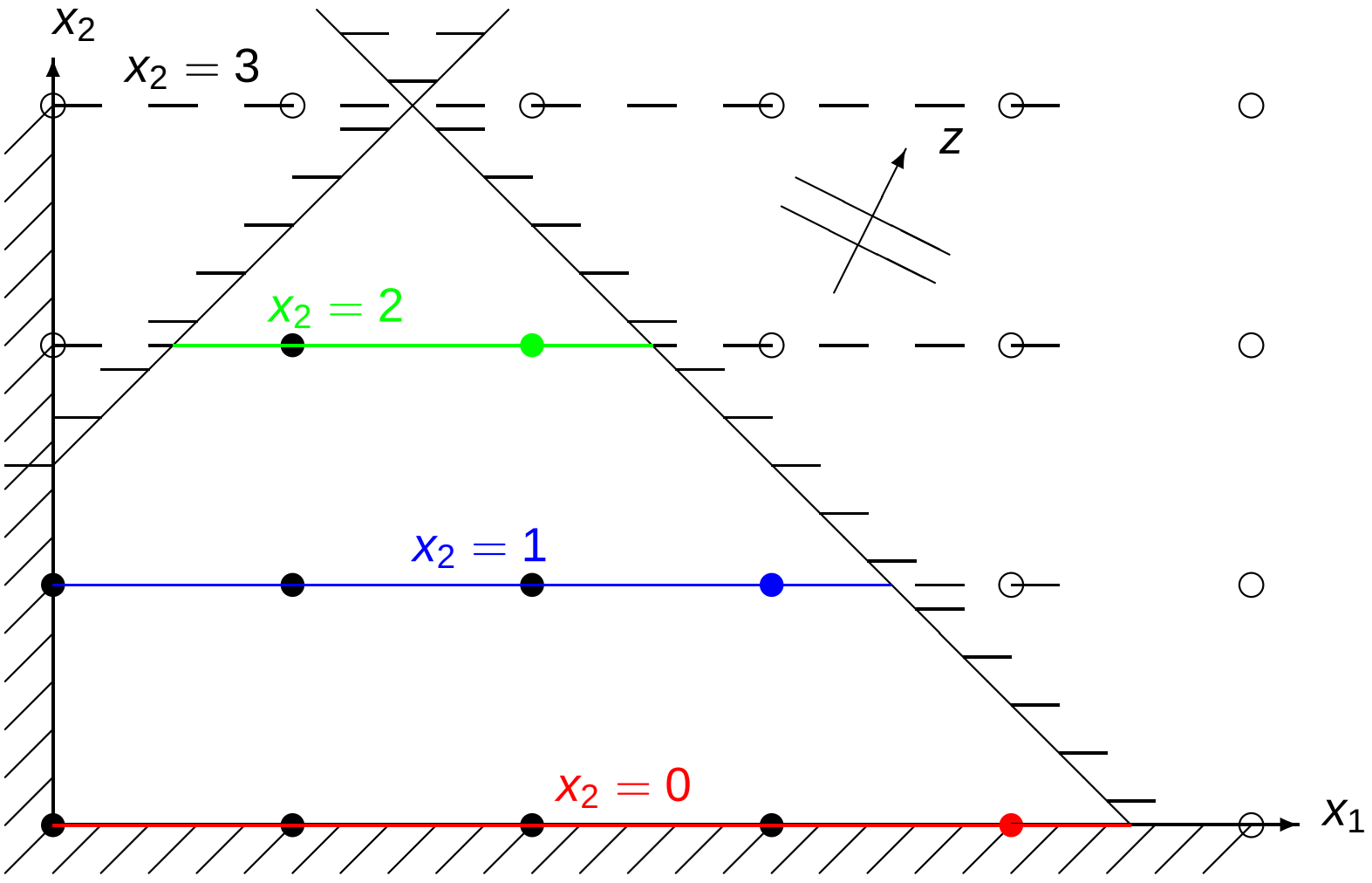
...

(2) $x_1 = x_2 = \ldots = x_{n-1} = 0, x_n = 1$

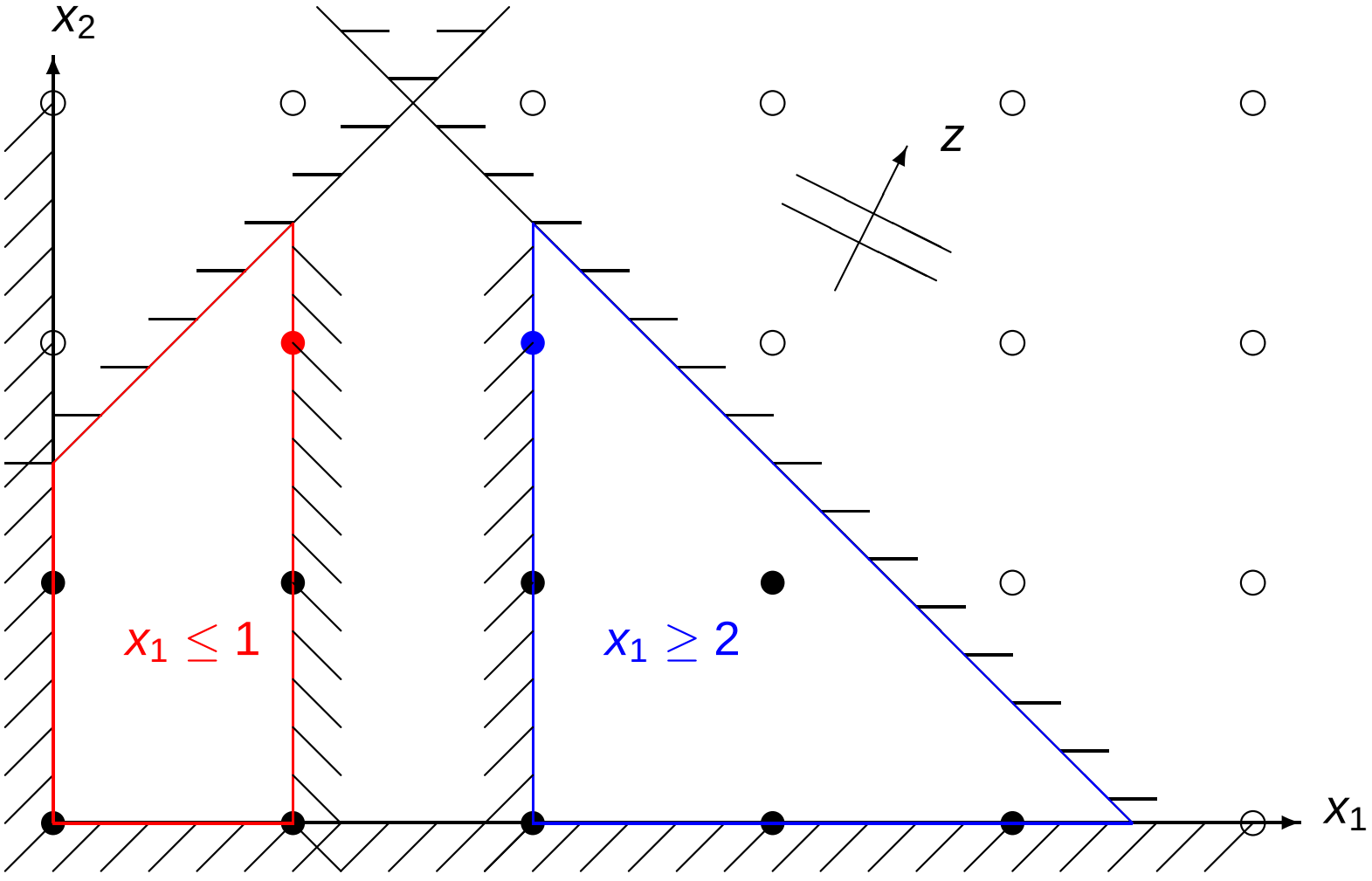
(2) $x_1 = x_2 = \ldots = x_n = 0$

n+1 sotto-problema

Branching tramite fissaggio di variabili



Branching tramite inserzione di vincoli



Foglie dell’albero

Solitamente un sotto-problema è “risolto” dal branching, cioè è sostituito da altri sotto-problemi.

Tuttavia questa procedura ricorsiva termina quando il sotto-problema corrente...

- ...è inammissibile;
- ...è risolto all’ottimo;
- ...può essere rtrascurato.

Tutti e tre i casi possono essere scoperti risolvendo un rilassamento del sotto-problema corrente.

Rilassamenti

Dato un problema $\mathcal{P}$,

$$\begin{aligned} &\text{minimize } z_{\mathcal{P}}(x) \\ &\text{s.t. } x \in \mathcal{X}_{\mathcal{P}} \end{aligned}$$

un problema $\mathcal{R}$

$$\begin{aligned} &\text{minimize } z_{\mathcal{R}}(x) \\ &\text{s.t. } x \in \mathcal{X}_{\mathcal{R}} \end{aligned}$$

è un rilassamento di $\mathcal{P}$ se e solo se valgono le due condizioni:

- $\mathcal{X}_{\mathcal{P}} \subseteq \mathcal{X}_{\mathcal{R}}$
- $z_{\mathcal{R}}(x) \leq z_{\mathcal{P}}(x) \;\; \forall x \in \mathcal{X}_{\mathcal{P}}$.

Il valore ottimo del rilassamento non è mai peggiore del valore ottimo del problema originale:

$$z^*_{\mathcal{R}} \leq z^*_{\mathcal{P}}.$$

F.O. di min

Rilassamenti

Come conseguenza della definizione di rilassamento, valgono questi corollari.

Corollario 1. Se $\mathcal{R}$ è inammissibile, anche $\mathcal{P}$ è inammissibile.

Corollario 2. Se x^* è ottima per $\mathcal{R}$ ed è ammissibile per $\mathcal{P}$ e $z_{\mathcal{R}}(x) = z_{\mathcal{P}}(x)$, allora x^* è ottima anche per $\mathcal{P}$.

Corollario 3. Se $z^*_{\mathcal{R}} \geq \bar{z}$, allora $z^*_{\mathcal{P}} \geq \bar{z}$.

Il Corollario 3 è sfruttato nell'operazione di bounding.

Bounding

Stiamo minimizzando

Il bounding consiste nell'associare un bound duale ad ogni *sotto-problema* $\mathcal{F}$.

Poiché

$$z^*_{\mathcal{R}} \leq z^*_{\mathcal{P}}$$

il valore ottimo di $\mathcal{R}(\mathcal{F})$ (un rilassamento di $\mathcal{F}$) fornisce un bound duale ogni sotto-problema $\mathcal{F}$:

$$z^*_{\mathcal{R}(\mathcal{F})} \leq z^*_{\mathcal{F}}.$$

Il bound duale è confrontato con un bound primale che corrisponde al valore $z_{\mathcal{P}}(\bar{x})$ di una soluzione ammissibile $\bar{x} \in \mathcal{X}(\mathcal{P})$.

Se il bound duale di $\mathcal{F}$ risulta essere non-migliore del bound primale, allora $\mathcal{F}$ può essere scartato.

If $z^*_{\mathcal{R}(F)} \geq z_{\mathcal{P}}(\bar{x})$ then Fathom $\mathcal{F}$.

Bounding

La correttezza del bounding è data dalla concatenazione di due disuguaglianze.

- La prima garantisce che nessuna soluzione può esistere in $\mathcal{X}(\mathcal{F})$ con un valore migliore di $z^*_{\mathcal{R}(\mathcal{F})}$, poiché

$$z^*_{\mathcal{F}} \geq z^*_{\mathcal{R}(\mathcal{F})}.$$

- La seconda è $z^*_{\mathcal{R}(\mathcal{F})} \geq z_{\mathcal{P}}(\bar{x})$.

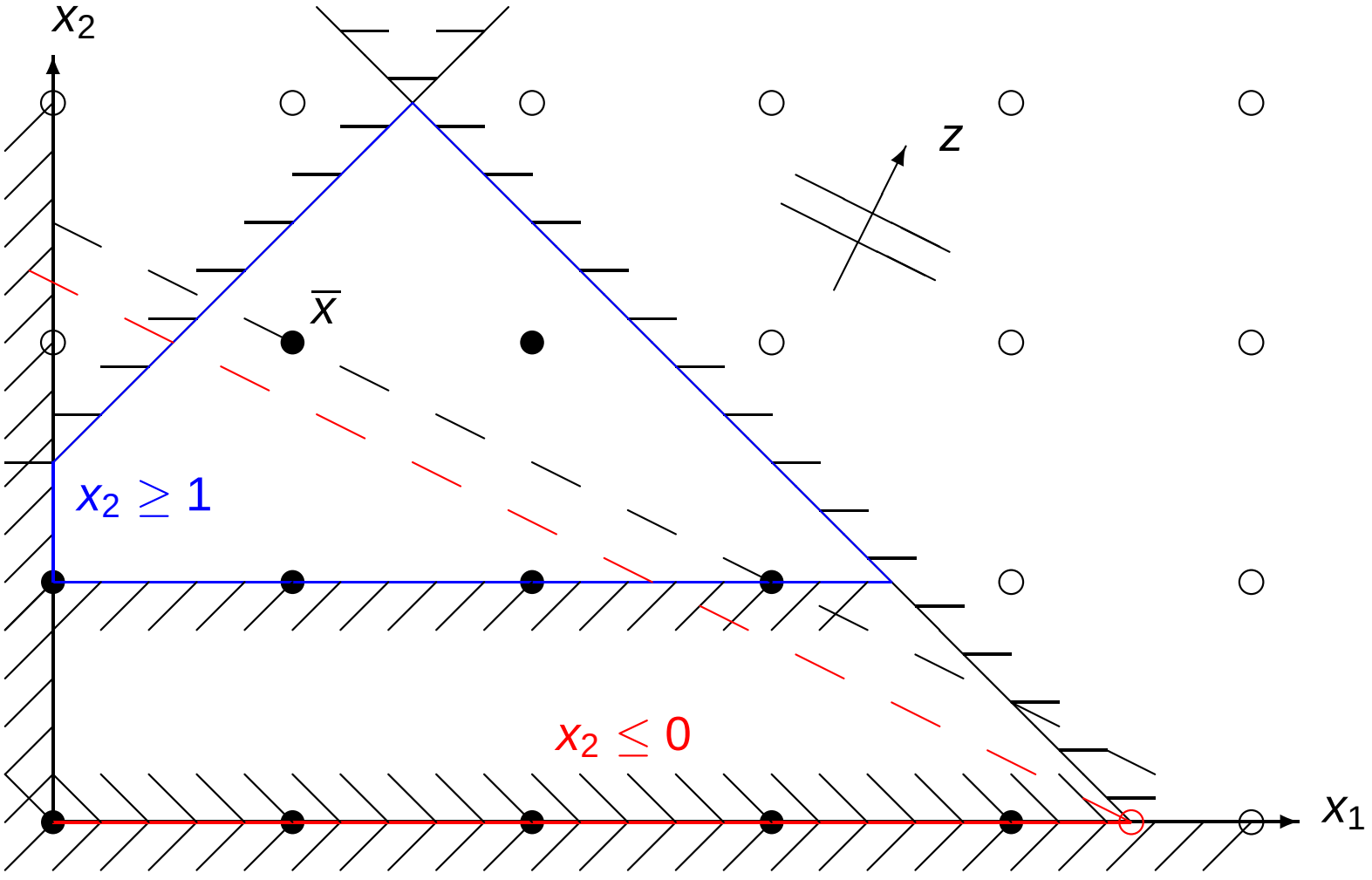
Concatenandole si conclude che

$$z^*_{\mathcal{F}} \geq z^*_{\mathcal{R}(\mathcal{F})} \geq z_{\mathcal{P}}(\bar{x})$$

che significa che risolvere il problema $\mathcal{F}$ all'ottimo è inutile, perché esso non può fornire alcuna soluzione migliore di quella già nota, $\bar{x}$.

Scartare sotto-problemi in un algoritmo branch-and-bound è cruciale per risparmiare tempo e memoria.

Esempio



Strategia di visita dell'albero

Ogni volta che due o più sotto-problemi vengono generati, essi vengono appesi ad una lista di nodi aperti, cioè di sotto-problemi da risolvere.

Questo è necessario perché l'algoritmo viene eseguito su una macchina seriale e i sotto-problemi non possono essere esaminati in parallelo.

La politica segujita per decidere quali nodi visitare per primi è detta search strategy.

Il *sotto-problema corrente* è quello che viene risolto ad un generico istante durante l'esecuzione dell'algoritmo.

Strategia di visita dell'albero

Si possono usare vari criteri per gestire la lista dei nodi aperti:

- FIFO: breadth-first search
- LIFO: depth-first search
- Lista ordinata: best-first search

La Best-first search è solitamente basata sul valore del bound duale: vengono esplorati prima i nodi più promettenti.

Per mantenere la lista ordinata è utile utilizzare uno *heap*.