Metodi e Modelli per il Supporto alle Decisioni

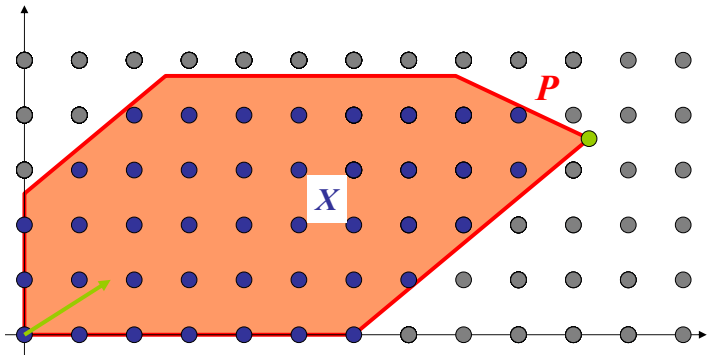
14. Metodi esatti per problemi di Programmazione Lineare Intera e Ottimizzazione Combinatoria: Branch and Bound.

Problemi di Programmazione Lineare Intera - PLI

- Consideriamo un problema di Programmazione Lineare Intera nella forma:

$$\begin{array}{ll} \min / \max & c^T x \\ \text{s.t.} & Ax = b \\ & x \in \mathbb{Z}^n_+ \end{array} \qquad \begin{array}{l} P = \{x \in R^n_+ : Ax = b\} \\ X = P \cap \mathbb{Z}^n \end{array}$$

- Il problema non può essere risolto con il metodo del simplesso: in generale, il vertice ottimale non è a coordinate intere (dipende dal poliedro ammissibile e dalla funzione obiettivo) e non sempre l'approssimazione intera ottenuta per arrotondamento è ottima e/o ammissibile!



Problemi di Ottimizzazione Combinatoria

- PLI è un caso particolare di problemi di ottimizzazione combinatoria - OC

$$\begin{array}{ll} \min / \max & f(x) \\ \text{s.t.} & x \in X \end{array}$$

X insieme FINITO di punti
$f(x)$ generica, anche non in forma chiusa

- Esempi di problemi di ottimizzazione combinatoria: oltre ai PLI,
  - problema del cammino minimo:
    $X = \{\text{tutti i possibili cammini da } s \text{ a } t\}$, $f(x)$: costo del cammino $x \in X$.
  - colorazione di un grafo:
    $X = \{\text{tutte le combinazioni ammissibili di colori assegnati ai vertici}\}$, $f(x)$: numero di colori utilizzati dalla combinazione $x \in X$.
  - programmazione lineare:
    $X = \{\text{tutte le soluzioni ammissibili di base}\}$, $f(x) = c^T x$.
  - **etc. etc. etc.**
- Per alcuni problemi di OC esistono algoritmi efficienti (classe $\mathcal{P}$), per gli altri:
  - considero un algoritmo di complessità esponenziale;
  - oppure considero un algoritmo efficiente ma approssimato (algoritmi euristici e metaeuristici).

Algoritmo universale per OC

- Sfruttando la finitezza dello spazio delle soluzioni ammissibili, posso risolvere un qualsiasi problema di OC come segue:
  1. genero tutte le possibili soluzioni x;
  2. verifico l'ammissibilità della soluzione $x \in X$;
  3. valuto $f(x)$
  4. scelgo la x ammissibile cui corrisponde la migliore $f(x)$.
- Alcuni potenziali problemi:
  - la valutazione di $f(x)$ potrebbe non essere banale (ad es. per simulazione);
  - la cardinalità di X potrebbe essere **molto** elevata.
- Come genero lo spazio delle soluzioni (ammissibili)?
- Come esploro efficientemente lo spazio delle soluzioni?
- L'algoritmo **Branch and Bound** risponde a queste esigenze.

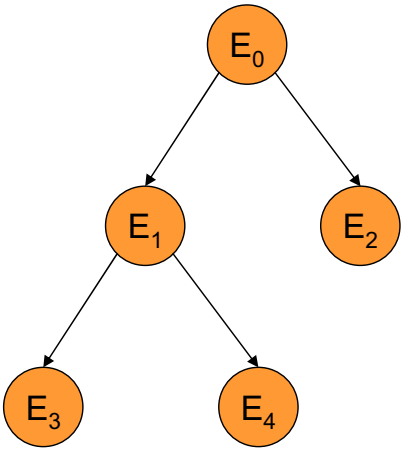
Generazione delle soluzioni: operazione di branch

- Per generare le soluzioni ammissibili posso utilizzare un **albero delle soluzioni ammissibili**
- Sia P_0 l'insieme delle soluzioni iniziali cui corrisponde $E_0 = \mathrm{X}$.

* E_0 è la radice dell'albero.
* In generale, E_i è l'insieme delle soluzioni associate al nodo i
* I nodi fratelli sono una **partizione** dell'insieme delle soluzioni del padre

  $E_i = \bigcup_{j \text{ figlio di } i} E_j$: non perdo soluzioni;

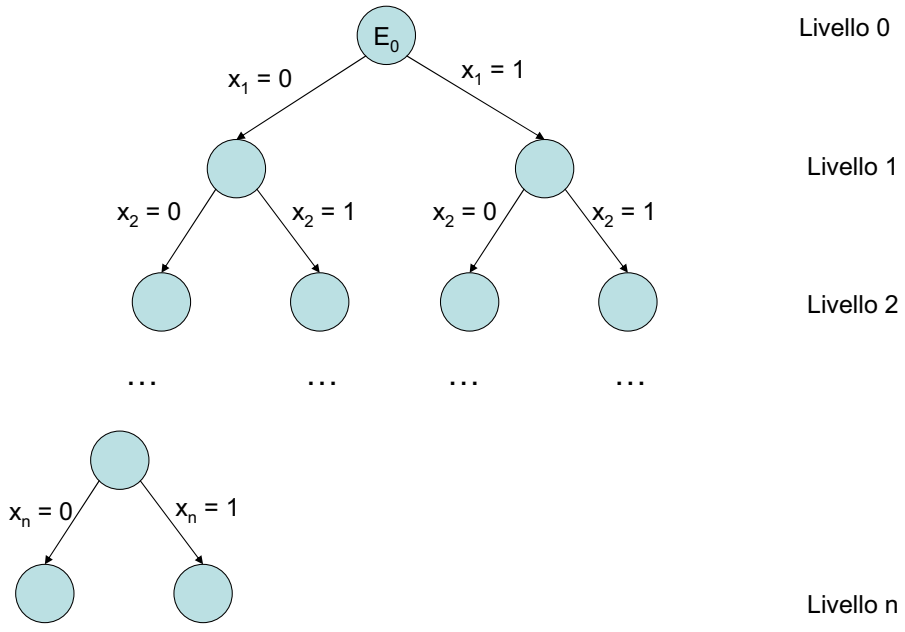
  ($E_j \cap E_k = \emptyset$, $\forall\, j, k$ figlio di i, auspicabile ma non necessario)
* nodi foglia: nodi i per cui $|E_i| = 1$.



- La fase di costruzione di nodi fratelli per partizione di un nodo padre è detta **BRANCH**: un insieme di livello h viene suddiviso in t insiemi di livello $h+1$.

Esempio di branch binario

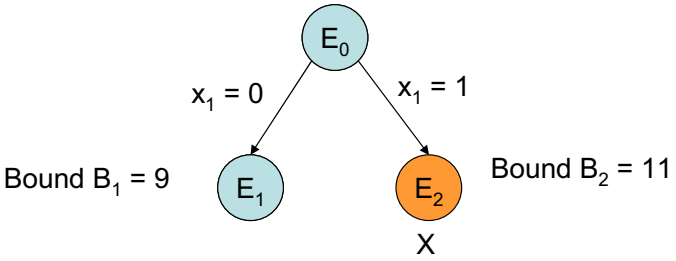
Con variabili binarie $x_i \in \{0, 1\}, i = 1..n$:



$n+1$ livelli, 2^{n-1} nodi intermedi, 2^n **foglie!**

Esplorazione efficiente: operazione di bound

- In generale, il numero di foglie è esponenziale
- Per ogni nodo si considera un **BOUND**, una valutazione *ottimistica* (non peggiore) del valore della funzione obiettivo per tutte le soluzioni nel nodo.
- Esempio:
  supponiamo di disporre si una soluzione ammissibile di valore 10 per un problema di min.



  È inutile sviluppare ed esplorare il sotto-albero con radice E_2!
- L'operazione di *bounding* ci permette di "potare" (non sviluppare) sotto-alberi che sicuramente non contengono la soluzione ottima!
- **Enumerazione implicita**: esplorazione implicita dell'albero delle soluzioni.

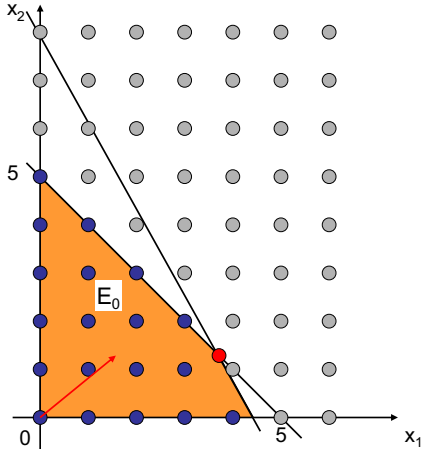
Metodo del Branch and Bound (B&B): idea di base

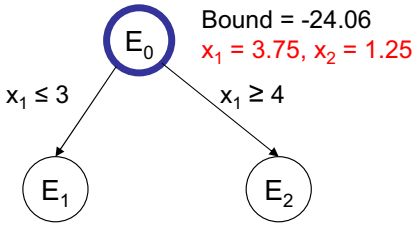
Il metodo del Branch and Bound (B&B) per la soluzione di **problemi di ottimizzazione combinatoria** si basa sui seguenti elementi:

- **operazione di branching**:
  costruzione dell'albero delle soluzioni ammissibili;
- **operazione di bounding**:
  valutazione ottimistica dei nodi, per evitare lo sviluppo completo di sotto-alberi (enumerazione implicita)

Esempio: un problema di PLI

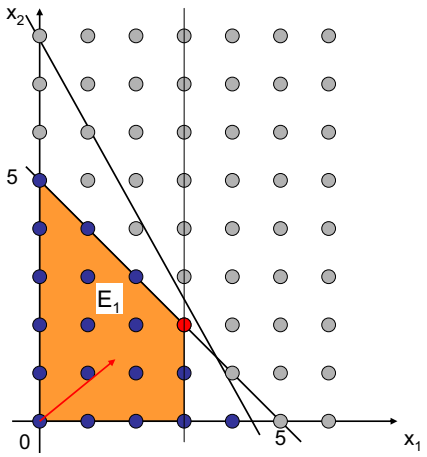
$$(P_0) \quad \min \; -5x_1 - \frac{17}{4}x_2$$
$$\text{s.t.} \; x_1 + x_2 \leq 5$$
$$10x_1 + 6x_2 \leq 45$$
$$x_1, x_2 \in \mathbb{Z}_+$$

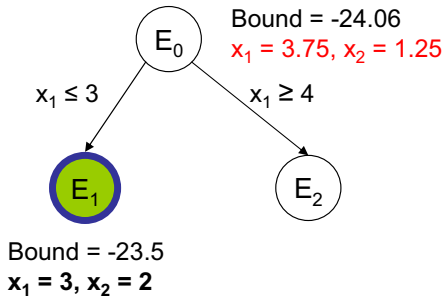




Esempio: un problema di PLI

$$(P_1) \quad \min \; -5x_1 - \frac{17}{4}x_2$$
$$\text{s.t.} \; x_1 + x_2 \leq 5$$
$$10x_1 + 6x_2 \leq 45$$
$$x_1 \leq 3$$
$$x_1, x_2 \in \mathbb{Z}_+$$

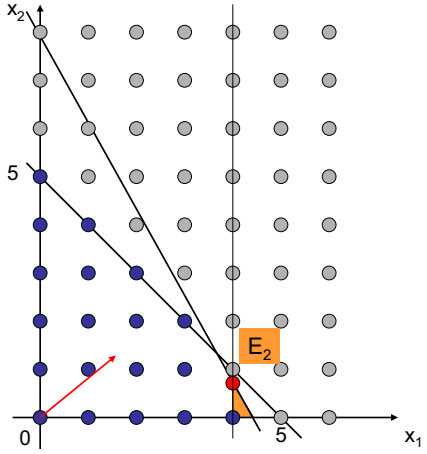


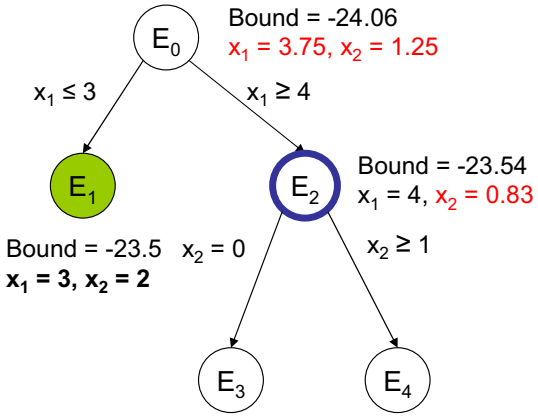


Esempio: un problema di PLI

$$(P_2) \quad \min \; -5x_1 - \frac{17}{4}x_2$$

$$\text{s.t.} \quad x_1 + x_2 \leq 5$$
$$10x_1 + 6x_2 \leq 45$$
$$x_1 \geq 4$$
$$x_1, x_2 \in \mathbb{Z}_+$$

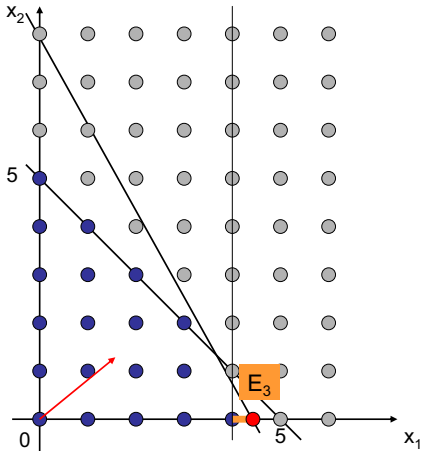


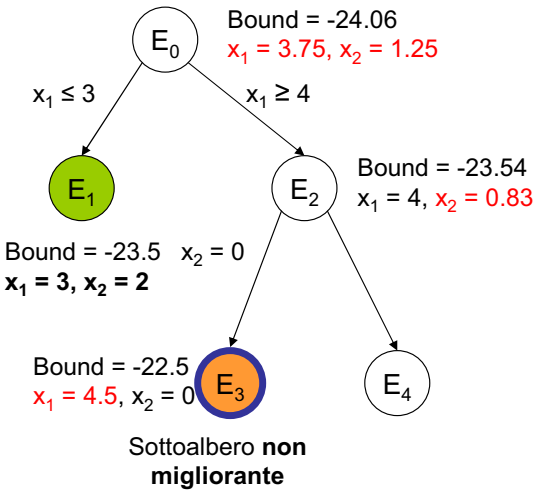


Esempio: un problema di PLI

$$(P_3) \quad \min \; -5x_1 - \frac{17}{4}x_2$$

$$\text{s.t.} \quad x_1 + x_2 \leq 5$$
$$10x_1 + 6x_2 \leq 45$$
$$x_1 \geq 4$$
$$x_2 = 0$$
$$x_1, x_2 \in \mathbb{Z}_+$$

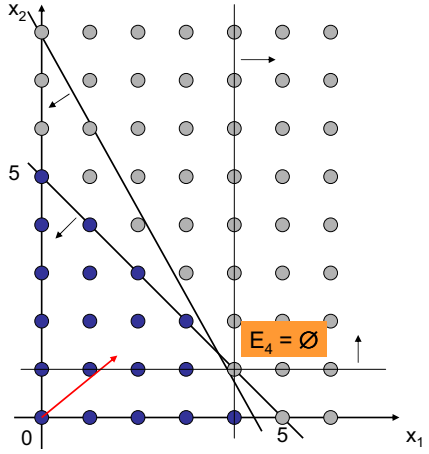


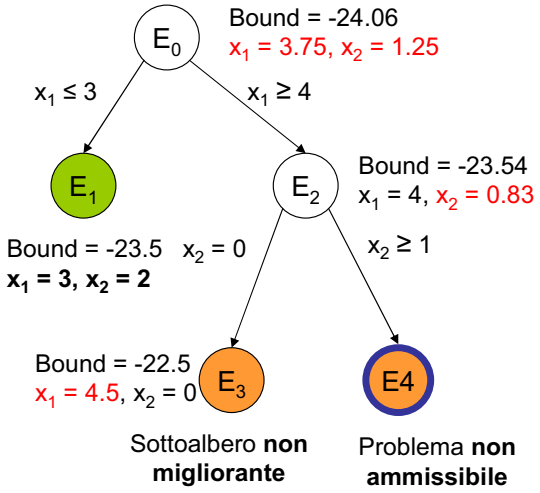


Esempio: un problema di PLI

$$(P_4) \quad \min \; -5x_1 - \frac{17}{4}x_2$$

$$\text{s.t.} \quad x_1 + x_2 \leq 5$$
$$10x_1 + 6x_2 \leq 45$$
$$x_1 \geq 4$$
$$x_2 \geq 1$$
$$x_1, x_2 \in \mathbb{Z}_+$$





Componenti del metodo B&B

Per implementare un algoritmo B&B per un dato problema di OC, bisogna determinare i seguenti elementi essenziali:

(1) Regole di Branching: come costruire l'albero delle soluzioni.

(2) Calcolo del Bound: come valutare i nodi.

(3) Regole di Fathoming: come chiudere e dichiarare sondati (*fathomed*) i nodi.

(4) Regole di esplorazione dell'albero: definire le priorità di visita dei nodi aperti.

(5) Come valutare una o più soluzioni ammissibili: soluzioni che permettano di utilizzare i bound ottimistici per chiudere nodi.

(6) Criteri di stop: condizioni di terminazione dell'algoritmo.

Componenti del metodo B&B

(1) Regole di Branching.

Principio del *divite et impera*: si passa da un problema P con soluzioni E a una partizione in sottoproblemi P_i con soluzioni $E_i : \cup_i E_i = E$, $(E_i \cap E_j = \emptyset)$.

La condizione $\cup_i E_i = E$ assicura che la soluzione ottima si trovi in almeno uno dei nodi figli!

Gli E_i sono sempre più piccoli: ciascun nodo E_i eredita ricorsivamente le condizioni dei genitori (i corrispondenti problemi P_i sono sempre più vincolati).

I P_i sono sempre più semplici: al limite, in un nodo foglia f, E_f contiene una sola soluzione e la soluzione di P_f è immediata.

Bisogna decidere la strategia di branching!

- Quanti nodi figli generare? branching binario (2 figli) o t-ario (t figli)
- Su quale variabile effettuare il branching?
- etc. etc. etc.

Componenti del metodo B&B

(2) Calcolo del Bound.

Dato un problema P_i ed il relativo E_i, è necessario, ad ogni nodo, calcolare una stima ottimistica della migliore soluzione in E_i. Tale stima rappresenta un limite al miglior valore ottenibile se si sviluppasse il sotto-albero con radice E_i.

min: serve un limite inferiore: **lower bound** LB (LB $\leq$ sol. ottima)

max: serve un limite superiore: **upper bound** UB (UB $\geq$ sol. ottima)

Bisogna decidere il metodo per calcolare il bound!

Compromesso tra la qualità (efficacia) del bound e la facilità di calcolo (efficienza computazionale).

- Il limite deve essere il più **stringente** possibile, per permettere di esplorare implicitamente (chiudere) un maggior numero di nodi:
  - min: il più alto possibile
  - max: il più basso possibile
- Il limite deve essere **calcolato rapidamente** (viene valutato ad ogni nodo!).

Componenti del metodo B&B

(3) Regole di fathoming.

Un nodo E_i con bound B_i viene chiuso (e quindi esplorato implicitamente) se si verifica (almeno) una delle seguenti condizioni

N.M. **Assenza di soluzione migliorante**: la valutazione ottimistica B_i è NON MIGLIORE di una soluzione ammissibile nota.

S.A. **Soluzione ammissibile**: la valutazione ottimistica B_i è in relazione ad una soluzione ammissibile. Nel nodo E_i non si possono trovare soluzioni ammissibili migliori di quella che ho calcolato! Se B_i è migliore, aggiorno la soluzione ammissibile corrente.

N.A. **Problema non ammissibile**: il problema P_i corrispondente al nodo in esame non ammette soluzioni ($E_i = \emptyset$). L'insieme delle condizioni che determinano il problema P_i si ottiene considerando tutte le condizioni del problema originario più le condizioni che hanno generato i progenitori (problemi sul percorso dal nodo radice al nodi in esame). Tali condizioni potrebbero entrare in conflitto tra loro.

Componenti del metodo B&B

(4) Regole di esplorazione dell'albero.

Tra i diversi nodi ancora aperti, bisogna decidere su quale nodo effettuare il branching. La scelta influenza il numero di nodi complessivamente aperti (e quindi l'efficienza).

Bisogna decidere la strategia di esplorazione!

* *Depth First*: nodo di livello maggiore
  + semplice da implementare;
  + permette di ottenere presto soluzioni ammissibili (raggiungere le foglie);
  + limita la memoria necessaria (pochi nodi *contemporaneamente* aperti);
  - rischio di esplorare completamente sotto-alberi con soluzioni scadenti.
* *Best Bound First*: nodo più promettente
  + tipicamente permette di limitare il numero di nodi aperti *nel complesso* (più efficiente);
  - l'esplorazione tende a rimanere a livelli poco profondi, dove i problemi sono meno vincolati e, di conseguenza, i bound sono migliori: difficilmente si ottengono soluzioni ammissibili che migliorino quella corrente per applicare efficacemente le regole di fathoming;
  - maggiore richiesta di memoria per i nodi aperti *contemporaneamente*.
* Regole miste: ad es. all'inizio Depth First e, quando si ha una "buona" soluzione ammissibile, Best Bound First.

Componenti del metodo B&B

(5) Valutazione di soluzioni ammissibili.

Avere soluzioni ammissibili di buona qualità rende più efficace il fathoming.

Si deve decidere come e quando calcolare soluzioni ammissibili

- Aspettare semplicemente che l'enumerazione generi un nodo foglia ammissibile.
- Implementare un algoritmo euristico che valuti una buona soluzione all'inizio
- Si può sfruttare, con frequenza da valutare, l'informazione raccolta durante l'esplorazione dell'albero per costruire soluzioni ammissibili sempre migliori.

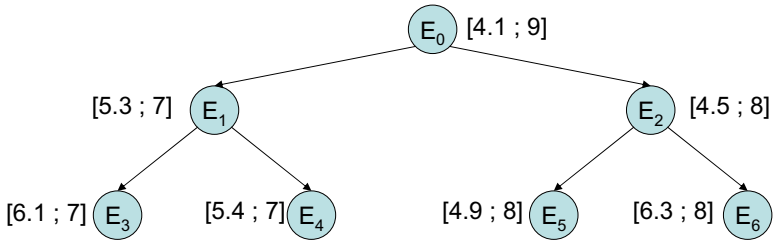
valutare sempre la qualità della soluzione ammissibile corrente con lo sforzo computazionale per ottenerla

(6) Criteri di stop.

- classico: tutti i nodi sono chiusi. La soluzione ammissibile corrente è ottima.
- limiti computazionali: raggiunti limiti di memoria o di tempo di calcolo. L'eventuale soluzione ammissibile corrente non è ottima.

Esercizio

Si consideri il seguente albero di B&B relativo ad un problema di minimo. Ad ogni nodo sono stati valutati sia un lower bound (LB) sia una soluzione ammissibile (SA), come riportato accanto ad ogni nodo, nel formato [LB;SA].



1. Quale nodo sarà selezionato per il branching se si adotta una strategia di esplorazione *Best Bound First*? $\{E_5\}$
2. Entro quale intervallo di valori è sicuramente compreso il valore ottimo della funzione obiettivo? $\{[4.9\ 7]\}$
3. Si supponga di fare branching sul nodo E_5, ottenendo due nodi figli E_7 ed E_8. Sia inoltre il problema P_8 inammissibile ($E_8 = \emptyset$). Dare un esempio di possibili valori LB e SA per il nodo E_7 che permettano di riconoscere subito una soluzione ottima, senza ulteriori operazioni di branching. $\{SA_7 = LB_7 \in [4.9\ 5.4]\}$