Ricerca Operativa

Branch-and-Bound per problemi di Programmazione Lineare Intera

L. De Giovanni

AVVERTENZA: le note presentate di seguito non hanno alcuna pretesa di completezza, né hanno lo scopo di sostituirsi alle spiegazioni del docente. Il loro scopo è quello di fissare alcuni concetti presentati in classe. Le note contengono un numero limitato di esempi ed esercizi svolti. Questi rappresentano una parte fondamentale nella comprensione della materia e sono presentati in classe.

Contents

1 Programmazione lineare intera e ottimizzazione combinatoria

Consideriamo un problema di Programmazione Lineare Intera (PLI) nella forma:

$$\begin{array}{ll} \min / \max & c^T x \\ \text{s.t.} & Ax = b \\ & x \in \mathbb{Z}^n_+ \end{array}$$

e chiamiamo

$$P = \{x \in R^n_+ : Ax = b\}$$

il poliedro delle soluzioni ammissibili ignorando il vincolo di interezza delle variabili e

$$X = P \cap \mathbb{Z}^n$$

l'insieme delle soluzioni ammissibili, cioè intere (vedi Figura 1).

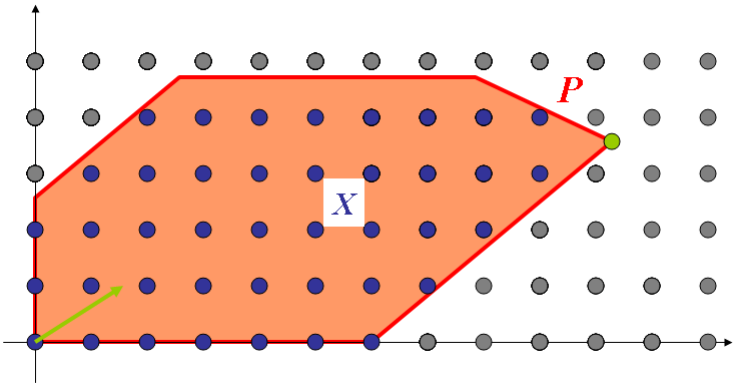


Figure 1: Soluzioni ammissibili per un PLI.

Il nostro obiettivo è quello di trovare una soluzione del PLI, ossia $x^* \in X : c^T x^* \leq c^T x, \forall\, x \in X$.

Osserviamo, come è evidente dalla stessa Figura 1, che il problema non può essere risolto con metodi per programmazione lineare a variabili reali (ad esempio, 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!

Uno dei metodi risolutivi per problemi di PLI è il metodo del Branch-and-Bound, che descriveremo nel prossimo paragrafo. Tale metodo è un metodo generale, applicabile ai

problemi di ottimizzazione combinatoria. In effetti, i problemi di PLI possono essere visti come casi particolari di problemi di ottimizzazione combinatoria.

Un **problema di ottimizzazione combinatoria** è definito come:

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

dove X è un insieme *FINITO* di punti e $f(x)$ è una generica funzione obiettivo. Esempi di problemi di ottimizzazione combinatoria, oltre ai PLI, sono:

- problema del cammino minimo:
  $X = \{$tutti i possibili cammini da s a $t\}$, $f(x)$: costo del cammino $x \in X$.
- colorazione di un grafo:
  $X = \{$tutte le combinazioni ammissibili di colori assegnati ai vertici$\}$, $f(x)$: numero di colori utilizzati dalla combinazione $x \in X$.
- programmazione lineare:
  $X = \{$tutte le soluzioni ammissibili di base$\}$, $f(x) = c^T x$.
- **etc. etc. etc.**

Per alcuni problemi di ottimizzazione combinatoria esistono algoritmi efficienti (problemi nella classe $\mathcal{P}$, come ad esempio il problema del cammino minimo), mentre per gli altri si deve considerare un algoritmo risolutivo esatto di complessità esponenziale, oppure, se è sufficiente una "buona" soluzione e non forzatamente quella ottima, si può considerare un algoritmo efficiente ma approssimato (algoritmi euristici e meta-euristici). In questo corso ci limiteremo ad accennare a un algoritmo esatto.

2 Metodi enumerativi e Branch-and-Bound

Il metodo esatto per problemi di ottimizzazione combinatoria che consideriamo si basa su uno schema enumerativo che sfrutta la *finitezza* dello spazio delle soluzioni ammissibili. Lo schema è detto *Algoritmo universale per ottimizzazione combinatoria*:

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 $f(x)$.

Ovviamente, lo schema è molto semplice ma soffre di due evidenti problemi. Il primo, è che la valutazione di $f(x)$ potrebbe non essere banale (ad es. potrebbe essere necessaria una simulazione per valutare la "bontà" di una soluzione). Il secondo, più generale, è che *la cardinalità di X potrebbe essere* ***molto*** *elevata*. In particolare, la seconda osservazione pone due questioni:

1. come *genero* lo spazio delle soluzioni (ammissibili)?
2. come *esploro* efficientemente lo spazio delle soluzioni?

L'algoritmo del branch-and-bound implementa lo schema dell'enumerazione sopra visto cercando di dare una risposta a queste esigenze.

2.1 Generazione delle soluzioni: operazione di branch

Per capire come generare le soluzioni ammissibili, osserviamo che:

> *Dato un problema di ottimizzazione combinatoria* $z = \max / \min\{f(x) : x \in X\}$ *e data una suddivisione della regione ammissibile* X *in insiemi* $X_1, \ldots, X_n$: $\bigcup_{i=1}^{n} X_i = X$, *sia* $z^{(k)} = \max / \min\{f(x) : x \in X_k\}$. *Allora la soluzione del problema* $z = \max / \min_{k=1,\ldots,n} z^{(k)}$.

Possiamo quindi applicare il principio *divide et impera*: si suddivide X in sottoinsiemi più piccoli, e si risolve il problema su ogni sotto-insieme. Questo viene fatto ricorsivamente, dividendo a loro volta le regioni ammissibili dei sotto-problemi in sottoinsiemi. Se tale ricorsione venisse svolta completamente, alla fine enumereremmo tutte le possibili soluzioni ammissibili del problema. La divisione ricorsiva dell'insieme delle soluzioni del problema di partenza (insieme X) dà luogo ad un *albero delle soluzioni ammissibili*, come rappresentato in Figura 2.

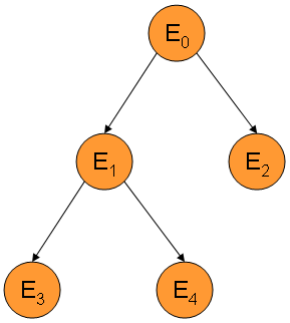


Figure 2: Generazione delle soluzioni: operazione di *Branch*.

Sia P_0 il problema di ottimizzazione combinatoria in esame cui corrisponde l'insieme delle soluzioni $E_0 = X$. E_0 è la radice dell'albero e, in generale, E_i è l'insieme delle soluzioni

associate al nodo i. L'operazione di suddivisione di un nodo E_i dà luogo a dei nodi *figli*. Tale suddivisione deve garantire che

$$E_i = \bigcup_{j \text{ figlio di } i} E_j$$

cioè, le soluzioni presenti nel nodo padre devono essere presenti in (almeno) uno dei suoi figli. In altre parole, nello sviluppo dell'albero devo garantire di non perdere soluzioni (altrimenti l'osservazione sopra riportata non sarebbe valida). Una caratteristica auspicabile, ma non necessaria, della suddivisione di un nodo E_i è la disgiunzione dei sottoinsiemi figli (la suddivisione sarebbe in questo caso una partizione in senso insiemistico):

$$E_j \cap E_k = \emptyset, \ \forall \ j,k \text{ figlio di } i$$

L'operazione di suddivisione si fermerebbe nel momento in cui ogni nodo contiene una sola soluzione. Pertanto, se E_f è un nodo foglia, si ha $|E_f| = 1$.

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

Esempio 1 *Si consideri un problema di ottimizzazione combinatoria con n variabili binarie $x_i \in \{0,1\}, i = 1..n$.*

In questo caso, dato un problema P_i e il corrispondente insieme di soluzioni ammissibili E_i, possiamo facilmente ottenere due sotto-problemi e due sottoinsiemi di E_i fissando una delle variabili binarie a 0 per un sotto-problema e a 1 per l'altro sotto-problema. Utilizzando questa *regola di branch* binario (ogni nodo è suddiviso in *due* nodi figli), otterremmo l'albero delle soluzioni in Figura 3.
Ad ogni livello viene fissato a 0 o a 1 il valore di una delle variabili. Un nodo di livello h, quindi, contiene tutte le soluzioni ammissibili con le variabili $x_1 \dots x_h$ fissate ad un preciso valore e, pertanto, tutti i nodi sono disgiunti. Le foglie si trovano pertanto al livello n, dove tutte le variabili sono state fissate: ogni foglia rappresenta una delle possibili stringhe binarie di n bit e, quindi, una (ed una sola) possibile soluzione del problema in questione. Si noti come il numero di foglie è 2^n e il numero di livelli è $n+1$ (incluso il livello 0 della radice).

2.2 Esplorazione efficiente: operazione di bound

In generale, il numero di foglie ottenute con un'operazione di branch è esponenziale. L'esplosione completa di un albero di soluzioni, corrispondente all'enumerazione di tutte le soluzioni in X è pertanto non praticabile come metodo risolutivo. Cerchiamo quindi un metodo che ci permetta di esplorare solo aree "buone" della regione ammissibile, cercando di escludere a priori che la soluzione ottima del problema si possa trovare in altre aree. Per fare questo, consideriamo, per ogni nodo dell'albero, un *BOUND*, ossia una valutazione *ottimistica* (non peggiore) del valore che la funzione obiettivo può assumere

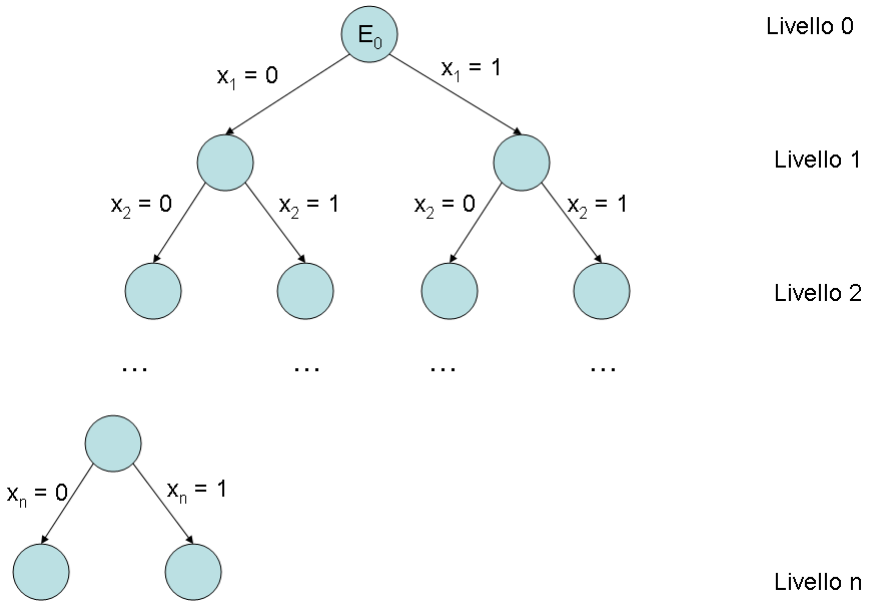


Figure 3: Branch binario.

in ciascuna delle soluzioni rappresentate dal nodo stesso. Ad esempio, consideriamo un problema di minimizzazione con variabili binarie e supponiamo di disporre di una soluzione ammissibile di valore 10. Supponiamo di associare, all'insieme delle soluzioni ammissibili per il problema di partenza, il nodo E_0 e di sviluppare il primo livello dell'albero di branching, fissando la variabile x_1 a 0 e a 1 e ottenendo due nodi figli E_1 ed E_2 (vedi Figura 4).

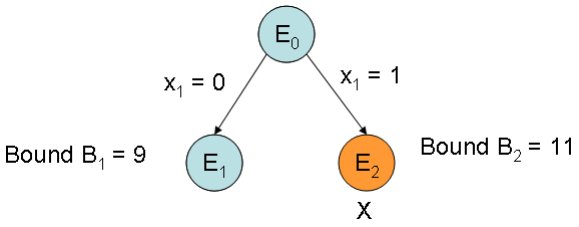


Figure 4: Uso dei Bound.

Chiamiamo $z^{(1)}$ il valore ottimo della funzione obiettivo nel sottoinsieme E_1 e $z^{(2)}$ il valore ottimo della funzione obiettivo nel sottoinsieme E_2. Ricordiamo che, come precedentemente osservato, il valore ottimo della funzione obiettivo del problema in esame è

$$z = \min\{z^{(1)}, z^{(2)}\}$$

Supponiamo ora di riuscire a stabilire, con qualche ragionamento e senza considerare ad una ad una le soluzioni, che il valore delle soluzioni ammissibili per il problema, una volta

fissato a 0 il valore di x_1 non possa essere minore di 9. In altre parole, il valore della funzione obiettivo in corrispondenza di ciascuna delle soluzioni rappresentare dal nodo E_1 è sicuramente *non* inferiore a 9. 9 rappresenta quindi una valutazione ottimistica della funzione obiettivo per il sottoinsieme E_1, un limite inferiore (*lower bound*) sotto il quale il valore della funzione obiettivo non può scendere, se consideriamo solo soluzioni in E_1: cioè $z^{(1)} \geq 9$. Analogamente, supponiamo di disporre di un *lower bound* per E_2 e sia tale bound pari a 11: nessuna della soluzioni con $x_1 = 1$ (soluzioni in E_1) ha un valore della funzione obiettivo più basso di 11: cioè $z^{(1)} \geq 11$. Ora, secondo la nostra prima osservazione, $z = \min\{z^{(1)}, z^{(2)}\}$. Inoltre, utilizzando l'informazione sulla soluzione ammissibile a disposizione, $z \leq 10$ e, pertanto $z^{(2)} \geq 11 \geq 10$, cioè, non è possibile trovare una soluzione con valore migliore di 10 tra le soluzioni nel nodo E_2. Pertanto,

> *E_2 non contiene sicuramente la soluzione ottima ed è inutile sviluppare ed esplorare il sotto-albero con radice E_2.*

Lo stesso ragionamento non vale per il nodo E_1: una delle soluzioni in questo nodo *potrebbe* avere valore inferiore a 10 e, pertanto, tale nodo potrebbe contenere la soluzione ottima.

In generale, se è nota una soluzione ammissibile $\bar{x}$ di valore $f(\bar{x}) = \bar{f}$, la disponibilità di un *bound* associato ai nodi dell'albero di branch ci permette di "potare" (non sviluppare) sotto-alberi che sicuramente non contengono la soluzione ottima, cioè i sotto-alberi radicati in un nodo con bound non migliore di $\bar{f}$.

Osserviamo che, anche se non abbiamo esplicitato tutti i nodi foglia del sotto-albero di E_2, siamo comunque in grado di stabilire che il valore di ciascuno di essi non è migliore di 11 e, grazie alla soluzione ammissibile, tale informazione è sufficiente per escluderli come soluzioni ottime: è come se avessimo esplorato tutto il sotto-albero in maniera *implicita*. Attraverso l'operazione di bound è quindi possibile effettuare una *enumerazione implicita* di tutte le soluzioni di un problema di ottimizzazione combinatoria.

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

Dalle osservazioni precedenti, è possible definire il metodo del Branch and Bound (B&B) per la soluzione di problemi di ottimizzazione combinatoria. Si tratta di un metodo che enumera in modo esplicito o implicito tutte le soluzioni del problema, basandosi sui seguenti elementi:

- *operazione di branch*: costruzione dell'albero delle soluzioni ammissibili;
- disponibilità di una soluzione ammissibile di valore $\bar{f}$;
- *operazione di bound*: valutazione ottimistica della funzione obiettivo per le soluzioni rappresentate da ciascun nodo (*bound*), per evitare lo sviluppo completo di sotto-alberi (enumerazione implicita delle soluzioni rappresentate dai nodi con bound non migliore di $\bar{f}$).

3 Il metodo di Branch-and-Bound

Il metodo del Branch-and-Bound può essere schematizzato come segue. Dato un problema di ottimizzazione combinatoria $z = \min / \max\{f(x) : x \in X\}$, sia:

- P_0: problema di ottimizzazione iniziale;
- L: lista dei nodi aperti. Ogni nodo è una coppia (P_i, B_i), dove P_i è il sotto-problema e B_i è il relativo bound;
- $\bar{z}$: valore della migliore soluzione ammissibile.
- $\bar{x}$: migliore soluzione ammissibile corrente;

Metodo di Branch-and-Bound

0.	*Inizializzazione*:	Esegui una stima ottimistica B_0 della funzione obiettivo e poni $L = \{(P_0, B_0)\}$, $\bar{x} = \emptyset$, $\bar{z} = +\infty(\min)[-\infty(\max)]$
1.	*Criterio di Stop*:	Se $L = \emptyset$, allora STOP: $\bar{x}$ è la soluzione ottima. Se superati limiti di tempo, nodi esplorati, nodi aperti $\|L\|$ etc. STOP: $\bar{x}$ è una soluzione (non necessariamente ottima).
2.	*Selezione nodo*:	Seleziona $(P_i, B_i) \in L$ per effettuare il branch
3.	*Branching*:	Dividi P_i in t sotto-problemi $P_{ij}, j = 1..t$ $(\cup_j P_j = P_i)$
4.	*Bounding*:	Valuta una stima ottimistica B_{ij} (in corrispondenza di una soluzione non necessariamente ammissibile x^R_{ij}) per ciascun sotto-problema P_{ij}
5.	*Fathoming*:	Se P_{ij} non è ammissibile, vai a 1. Se B_{ij} non è migliore di $\bar{z}$ ammissibile, vai a 1. Se x^R_{ij} è ammissibile ed è migliore di $\bar{z}$, poni $\bar{z} \leftarrow B_{ij}$, $\bar{x} \leftarrow x^R_{ij}$; elimina da L tutti i nodi k con L_k non migliore di $\bar{z}$; vai a 1. Altrimenti, aggiungi (P_{ij}, B_{ij}) a L e vai a 1.

Quello sopra esposto è uno schema di principio per la soluzione di problemi di ottimizzazione combinatoria. Per implementare un algoritmo B&B per uno specifico problema, 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 da confrontare con i bound per chiudere nodi).

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

3.1 Regole di Branching

Le regole di branching permettono di costruire un albero di soluzioni ammissibili e devono rispecchiare il principio del *divide et impera*: si passa da un problema P con soluzioni E a una partizione in sotto-problemi P_i con soluzioni $E_i : \cup_i E_i = E$ (non si perdono soluzioni ammissibili). Inoltre, per motivi di efficienza computazionale, conviene avere $E_i \cap E_j = \emptyset$, in modo da evitare duplicazioni di soluzioni in porzioni diverse dell'albero delle soluzioni. Osserviamo che:

- 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 restrittive 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.

Ovviamente, esistono vari modi per effettuare il branch di un nodo e bisogna, di volta in volta, stabilire la *strategia di branching* più adatta al problema. Bisogna, ad esempio, stabilire:

- quanti nodi figli generare: ad esempio, si può effettuare un branching binario (2 figli per nodo) o t-ario (t figli per nodo);
- come effettuare il branching: in generale, ciascuno dei sotto-insiemi da associare ai nodi figli si ottiene ponendo dei limiti alla variazione di una o più variabili decisionali e, per ciascun nodo, bisogna decidere quali variabili considerare e che limiti imporre;
- etc. etc. etc.

3.2 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. Osserviamo che, per problemi di min, la valutazione ottimistica corrisponde ad un valore sotto il quale siamo sicuri di non scendere, ossia si tratta di un limite inferiore, un *lower bound* (LB) per il nodo E_i ($LB_i \leq$ soluzione ottima in E_i). Per problemi di max, serve invece stabilire un

limite superiore che siamo sicuri di non poter superare, un *upper bound* (UB) per ciascun nodo E_i ($UB_i \geq$ soluzione ottima in E_i).

Ribadiamo che, per ottenere il bound di un nodo, non ha senso considerare tutte le soluzioni rappresentate dal nodo stesso (a meno che, trovandosi a livelli molto profondi dell'albero di ricerca, le soluzioni non siano veramente poche). In generale, quindi, è necessario determinare un metodo per calcolare il bound che utilizzi solo una descrizione implicita delle caratteristiche delle soluzioni rappresentate da un nodo. Nello stabilire il metodo bisogna valutare il compromesso tra la facilità di calcolo (efficienza computazionale) e la qualità (efficacia) del bound che si ottiene.

Per quanto riguarda l'efficienza computazionale, osserviamo che il bound viene calcolato ad ogni nodo e, come è facile immaginare, il numero di nodi da valutare potrebbe essere molto elevato (esponenziale nei casi peggiori). Pertanto il metodo di valutazione del bound deve garantire dei tempi di calcolo molto rapidi, altrimenti il metodo del B&B nel suo complesso diventa inefficiente.

Per quanto riguarda invece la qualità del bound, il limite ottenuto deve essere il più *stringente* possibile, sempre garantendo che il bound sia una valutazione ottimistica del nodo. In questo modo, una volta disponibile una soluzione ammissibile per il problema, si permettere di potare un maggior numero di nodi. Si fa notare che, per problemi di min, il bound deve essere il più alto possibile (ma sempre minore o uguale al valore ottimo per quel nodo), mentre, per problemi di max, il più basso possibile (ma sempre al di sopra del valore ottimo per quel nodo).

Esempio 2 *Si consideri il seguente problema di ottimizzazione combinatoria: una grossa azienda di costruzioni edili deve decidere la combinazione ottimale degli appalti da accettare per la costruzione degli edifici A, B e C. I profitti attesi per i tre edifici sono di* 3, 5 *e* 7 *milioni di euro rispettivamente. L'azienda dispone di* 4 *ruspe speciali e gli edifici richiedono rispettivamente* 3, 2 *e* 3 *ruspe. È possibile inoltre affittare fino a due altre ruspe speciali per la durata dei lavori, al costo di un milione di euro a ruspa. Al momento, il management suggerisce di eseguire gli appalti B e C e di affittare, di conseguenza, una ruspa aggiuntiva, con profitto netto di 11 milioni di euro.*

Associamo a ciascun appalto la variabile decisionale binaria x_i che vale 1 se si decide di accettare l'appalto, 0 altrimenti. Consideriamo inoltre la variabile intera y associata al numero di ruspe affittate. La funzione obiettivo del problema è quindi: $z = \max 3x_1 + 5x_2 + 7x_3 - 1y$. Cerchiamo di determinare un albero delle soluzioni ammissibili e un (upper) bound (UB) per i diversi nodi. Una possibile semplice regola per determinare il bound di ciascun nodo è la seguente: mettere a 1 le variabili con coefficiente in funzione obiettivo positivo e a 0 quelle con coefficiente negativo. Notiamo che il valore così ottenuto è comunque maggiore o uguale del valore della funzione obiettivo in una qualsiasi soluzione ammissibile e rappresenta, pertanto, un upper bound. Al nodo radice E_0 posso stabilire subito che $UB_0 = 15$ (si noti che non sono state valutate tutte le possibili soluzioni

ammissibili, ma è stato sufficiente seguire la semplice regola). Ovviamente stiamo solo dicendo che nessuna soluzione ammissibile potrà valere più di 15, ma non sappiamo se esiste una soluzione ammissibile che permetta di raggiungere tale valore. Consideriamo il nodo E_1, che rappresenta tutte le soluzioni con $x_1 = 0$ e il nodo E_2, che rappresenta tutte le soluzioni con $x_1 = 1$. Attraverso la stessa semplice regola, è possibile stabilire che $UB_1 = 12$ e $UB_2 = 15$.
Ovviamente, la regola utilizzata è molto semplice e, per ottenere dei bound migliori, sarebbe necessario complicare la regola (elevando il costo computazionale per valutarla), tenendo ad esempio in considerazione non solo la funzione obiettivo del problema, ma anche i vincoli. Ad esempio, possiamo osservare che, potendo affittare fino a due ruspe aggiuntive, avremmo a disposizione un massimo di 6 ruspe, che permetterebbero di eseguire al massimo 2 appalti. Possiamo quindi adottare la seguente regola per ottenere un upper bound: mettere a 1 le due variabili con coefficiente in funzione obiettivo positivo e di valore più elevato e a 0 quelle con coefficiente negativo. Anche in questo caso si ottiene una limitazione superiore al valore della funzione obiettivo per le soluzioni ammissibili (per le quali non più di due appalti sono accettati). Procediamo alla valutazione dei nodi con questa regola:

- nodo E_0: tutte gli appalti possono essere scelti e quindi $UB_0 = 7 + 5 = 12$;
- nodo E_1: avendo fissato $x_A = 0$, $UB_1 = 5 + 7 = 12$;
- nodo E_2: avendo fissato $x_A = 1$, posso scegliere soltanto un altro appalto, e scelgo l'appalto C, ottenendo $UB_2 = 3 + 7 = 10$.

Si noti come, al costo di una maggiore complessità nella valutazione, abbiamo ottenuto dei bound migliori, cioè degli upper-bound che, restando limitazioni superiori ai possibili valori della funzione obiettivo, sono però più bassi. Ad esempio, siamo stati in grado di stabilire che il valore delle soluzioni in E_2 è sicuramente non superiore a 10 (mentre i primi semplici ragionamenti ci avevano indicato un limite superiore a 15). Quindi, sia 15 che 10 sono degli upper bound validi per il nodo E_2, ma è da preferire il più basso tra i due (il più stringente). Infatti, con questa seconda valutazione, possiamo subito stabilire che il nodo E_2 non contiene la soluzione ottima, visto che esiste una soluzione ammissibile, quella suggerita dal management, che è migliore. Possiamo quindi evitare l'esplorazione esplicita di E_2.

3.3 Regole di potatura o *fathoming*

La disponibilità di un bound (da confrontare con il valore di una soluzione ammissibile) rende possibile dichiarare *esplorati* in modo implicito (*fathomed*) i nodi che sicuramente non contengono una soluzione ottima. Questi nodi non sono ulteriormente sviluppati (vengono cioè *chiusi*) e il relativo sotto-albero viene *potato*. 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, e quindi non ha senso continuare l'esplorazione del sotto-albero.

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.

Nel caso **S.A.**, il valore del bound B_i coincide con una soluzione ammissibile ed è pertanto il valore di una soluzione ammissibile. Si procede quindi a confrontare il bound ottenuto, con il valore della migliore soluzione ammissibile a disposizione e, se B_i è migliore, si aggiorna la soluzione ammissibile corrente e si procede a verificare la condizione **N.M.** sugli altri nodi ancora da esplorare. È possibile infatti che il valore migliorato della soluzione ammissibile sia non peggiore dei bound calcolati in precedenza, permettendo così la chiusura di alcuni nodi.

3.4 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 del metodo. Possono essere adottate diverse *strategie di esplorazione* che permettono di stabilire quale sia il prossimo nodo per il branch:

- *Depth First*: il nodo di livello maggiore (più profondo). Il metodo è semplice da implementare, permette di ottenere presto delle soluzioni ammissibili (ci si avvicina più rapidamente alle foglie) e limita la memoria necessaria per memorizzare l'albero delle soluzioni, visto che si tendono a chiudere molti nodi per ammissibilità e rimangono pochi nodi *contemporaneamente* aperti. Per contro, presenta il rischio di esplorare completamente sotto-alberi con soluzioni scadenti;

- *Best Bound First* o *Best node*: si sceglie il nodo più promettente, ossia il nodo con il bound migliore (lower bound più basso, per problemi di minimo, o upper bound più alto, per problemi di massimo). Tipicamente, tale strategia permette di limitare il numero di nodi visitati esplicitamente e tende pertanto a essere più efficiente. Per contro, l'esplorazione tende a rimanere a livelli poco profondi, dove i problemi sono meno vincolati e, di conseguenza, i bound sono più promettenti. Di conseguenza, difficilmente si ottengono presto soluzioni ammissibili che migliorino quella corrente per applicare efficacemente le regole di fathoming, e, pertanto è maggiore la richiesta di memoria per i nodi aperti *contemporaneamente*.

- *Regole miste*: i nodi vengono scelti alternando i diversi criteri, per evitarne gli svantaggi. Ad esempio, all'inizio si applica una strategia Depth First e, quando si ha una "buona" soluzione ammissibile, si passa alla strategia Best Bound First.

3.5 Valutazione di soluzioni ammissibili

Per applicare efficacemente le regole di fathoming, è necessario disporre di soluzioni ammissibili di buona qualità. Nella predisposizione di un algoritmo di Branch-and-Bound, bisogna quindi stabilire come e quando calcolare soluzioni ammissibili. Tra le varie possibilità, citiamo:

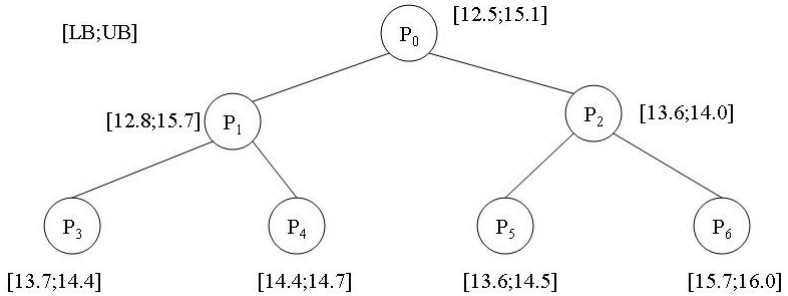
- aspettare semplicemente che l'enumerazione generi un nodo foglia ammissibile;
- implementare un algoritmo euristico che valuti una buona soluzione all'inizio, prima dell'esplorazione;
- sfruttare, con frequenza da valutare, l'informazione raccolta durante l'esplorazione dell'albero per costruire soluzioni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

In ogni caso, bisogna sempre valutare il compromesso tra la qualità della soluzione ammissibile corrente e lo sforzo computazionale per ottenerla.

3.6 Criteri di arresto

Il metodo del Branch-and-Bound si arresta quando tutti i nodi sono dichiarati *fathomed*. In questo caso, la soluzione ammissibile corrente corrisponde ad una soluzione ottima. Possono anche essere adottati criteri relativi a limiti computazionali, come ad esempio raggiunti limiti di tempo di calcolo o di memoria, ma non è garantito che l'eventuale soluzione ammissibile corrente sia ottima.

Esercizio 1 *Si consideri il seguente albero di sviluppo del Branch and Bound per un problema di ottimizzazione combinatoria con funzione obiettivo di minimo:*



1. *Come si può capire che si tratta di un problema di minimo?*
2. *È possibile chiudere dei nodi? Se sì, quali?*

3. *In quale intervallo è sicuramente compreso il valore della funzione obiettivo?*

4. *Quale nodo sarà sviluppato per primo da una strategia best first?*

5. *Si supponga che lo sviluppo di cui al punto precedente porti a due nodi figli, di cui uno è relativo ad un insieme di soluzioni vuoto. Si dia un esempio di valori di lower e upper bound relativi al secondo nodo che consentano di riconoscere subito la soluzione ottima del problema.*

1. Si capisce che è un problema di minimo perché i valori contrassegnati come LB sono crescenti di padre in figlio nell'albero e pertanto possono essere associati a valutazioni ottimistiche di problemi di minimo via via più vincolati. I valori contrassegnati come UB non sono decrescenti di padre in figlio e non possono essere associati a valutazioni ottimistiche di problemi di massimo via via più vincolati. I valori UB sono quindi le valutazioni della funzione obiettivo di minimo in corrispondenza di soluzioni ammissibili.

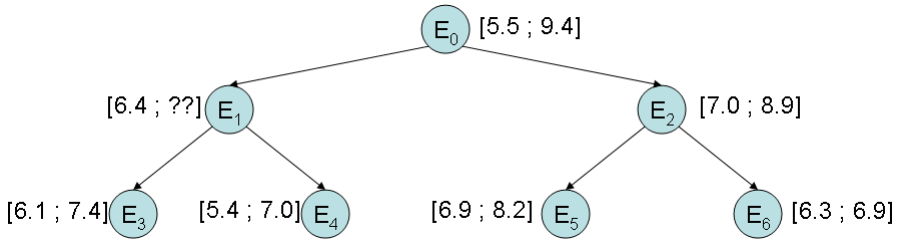
2. La migliore soluzione ammissibile vale 14.0 (vedi nodo P_2). Quindi è possibile chiudere i nodi P_4 e P_6 perché non miglioranti.

3. L'ottimo della funzione obiettivo è compreso tra 13.6 (il miglior lower bound - nodo P_5) e 14.0 (migliore soluzione disponibile).

4. Il nodo che sarà sviluppato per primo in una strategia *best bound first* è quello che ha la valutazione più promettente (LB più basso) tra quelli che rimangono aperti, cioè il nodo P_5.

5. Nel caso ipotizzato, rimangono aperti il nodo P_3 con $(LB, UB) = (13.7, 14.4)$ e un nodo P_7 con $(LB, UB) = (lb, ub)$. Basta quindi che sia $lb = ub$ (lb corrisponde a una soluzione ammissibile) per poter chiudere il nodo in esame P_7 e che $lb \leq 13.7$, per poter chiudere P_3. Deve inoltre essere $lb \geq 13.6$, per compatibilità con il lower bound del nodo padre P_5. Ad esempio $(LB, UB) = (13.65, 13.65)$, o $(LB, UB) = (13.7, 13.7)$, o $(LB, UB) = (13.6, 13.6)$ etc.

Esercizio 2 *Si consideri il seguente albero di B&B relativo ad un problema di massimo. Ad ogni nodo sono stati valutati sia un upper bound (UB) sia una soluzione ammissibile (SA), come riportato accanto ad ogni nodo, nel formato [SA;UB].*



- *Quale è un possibile valore per l'upper bound al nodo E_1?* $\{UB_1 \in [7.4\ 9.4]\}$[1]
- *Quale nodo sarà selezionato per il branching se si adotta una strategia di esplorazione* Best Bound First*?* $\{E_5\}$
- *Entro quale intervallo di valori è sicuramente compreso il valore ottimo della funzione obiettivo?* $\{[7.0\ 8.2]\}$
- *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 SA e UB per il nodo E_7 che permettano di riconoscere subito una soluzione ottima, senza ulteriori operazioni di branching.* $\{SA_7 = LB_7 \in [7.4\ 8.2]\}$

4 Branch-and-Bound per problemi di programmazione lineare intera

Introduciamo alcune definizioni. Consideriamo *problema di programmazione lineare intera* nella forma

$$\begin{array}{c} z_I = \max c^T x \\ Ax \leq b \\ x \geq 0 \\ x_i \in \mathbb{Z}, \qquad i \in I. \end{array} \tag{1}$$

dove $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ e $I \subseteq \{1, \ldots, n\}$ è l'insieme degli indici delle *variabili intere*. Le variabili x_i, $i \notin I$ sono invece dette *variabili continue*. Se il problema ha sia variabili intere che variabili continue, allora è detto un *problema di programmazione lineare intera mista*, mentre se tutte le variabili sono intere, il problema è detto di *programmazione lineare intera pura*.

L'insieme

$$X = \{x \in \mathbb{R}^n : Ax \leq b,\ x \geq 0,\ x_i \in \mathbb{Z} \text{ per ogni } i \in I\}$$

è la *regione ammissibile* del problema.

$$\begin{array}{c} z_L = \max c^T x \\ Ax \leq b \\ x \geq 0 \end{array} \tag{2}$$

[1] Il bound deve peggiorare (un upper bound deve essere non crescente, un lower bound non decrescente) man mano che si scende nei livelli dell'albero delle soluzioni, visto che i nodi corrispondono a problemi via via più vincolati. Non è possibile quindi che la valutazione ottimistica di un nodo sia peggiore della valutazione ottimistica di un nodo figlio, visto che il nodo padre include tutte le soluzioni del nodo figlio. Quindi, se si mettesse $UB_1 = 9.5 > 9.4$, darei al figlio una valutazione migliore del nodo padre, anche se il nodo padre contiene tutte le soluzioni del nodo figlio, incluso 9.4; se si mettesse $UB_1 = 7.1 < 7.4$, allora il nodo E_1 avrebbe una valutazione peggiore del figlio E_3.

corrisponde al problema (1) senza i vincoli di interezza delle variabili ed è detto il *rilassamento lineare* di (1).

Si noti il seguente facile fatto:

$$z_I \leq z_L. \tag{3}$$

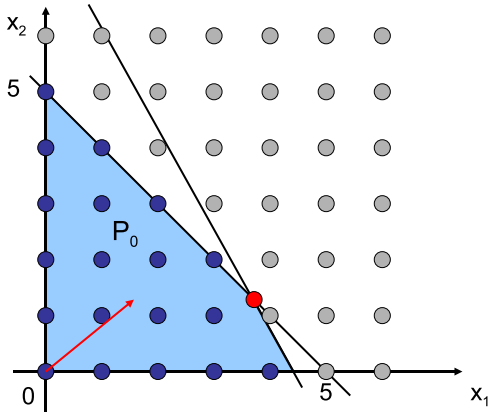
Infatti, se x^I é la soluzione ottima di (1) e x^L é la soluzione ottima di (2), allora x^I soddisfa i vincoli di (2), e dunque $z_I = c^T x^I \leq c^T x^L = z_L$. z_L rappresenta quindi un upper bound per z_I.

Vediamo ora come è possibile definire le varie componenti di un metodo di Branch-and-Bound per problemi di programmazione lineare intera. Illustriamo il metodo con un esempio, come applicazione dello schema presentato nella sezione precedente.

Si consideri il problema (P_0):

$$\begin{array}{lrcll} z_I^0 = \max & 5x_1 + \frac{17}{4}x_2 & & & \\ & x_1 + x_2 & \leq & 5 & \\ & 10x_1 + 6x_2 & \leq & 45 & (P_0) \\ & x_1, x_2 & \geq & 0 & \\ & x_1, x_2 & \in & \mathbb{Z} & \end{array}$$

La regione ammissibile di (P_0) e del suo rilassamento lineare è rappresentata nella figura successiva.

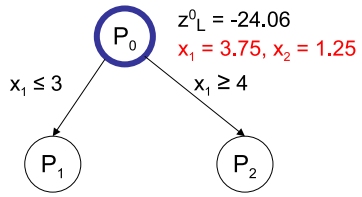


Risolvendo il rilassamento lineare di (P_0), otteniamo la soluzione ottima $x_1 = 3.75$, $x_2 = 1.75$, con valore $z_L^0 = 24.06$.

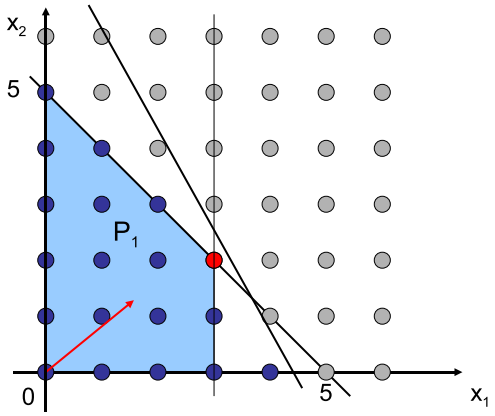
Dunque abbiamo ottenuto un upper bound per il valore ottimo z_I^0 di (P_0), ovvero $z_I^0 \leq 24.06$. Ora, poiché in una soluzione ottima di (P_0) x_1 avrà valore intero, allora la soluzione ottima soddisferà $x_1 \leq 3$ oppure $x_1 \geq 4$. Dunque la soluzione ottima di (P_0) sarà la soluzione migliore tra le due soluzioni dei problemi (P_1) e (P_2) così definiti:

$$z_I^1 = \max \quad 5x_1 + \tfrac{17}{4}x_2 \qquad (P_1) \quad , \qquad z_I^2 = \max \quad 5x_1 + \tfrac{17}{4}x_2 \qquad (P_2)$$
$$\begin{array}{rcl} x_1 + x_2 & \leq & 5 \\ 10x_1 + 6x_2 & \leq & 45 \\ x_1, & \leq & 3 \\ x_1, x_2 & \geq & 0 \\ x_1, x_2 & \in & \mathbb{Z} \end{array} \qquad \begin{array}{rcl} x_1 + x_2 & \leq & 5 \\ 10x_1 + 6x_2 & \leq & 45 \\ x_1, & \geq & 4 \\ x_1, x_2 & \geq & 0 \\ x_1, x_2 & \in & \mathbb{Z} \end{array}$$

Diremo che abbiamo fatto *branching sulla variabile* x_1. Si noti che in questo modo la soluzione $(3.75, 1.25)$ non appartiene al rilassamento lineare di (P_1) o (P_2). Possiamo rappresentare graficamente i sotto-problemi e i rispettivi bounds mediante un albero, detto albero di Branch-and-Bound.

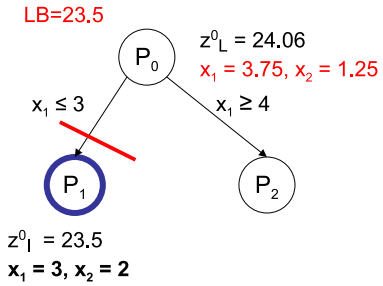


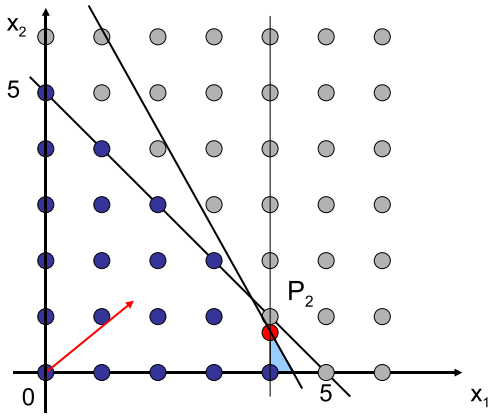
I *problemi attivi* sono le foglie dell'albero, nella fattispecie (P_1) e (P_2).
Consideriamo il problema (P_1), rappresentato in figura.



La soluzione ottima del rilassamento lineare di (P_1) è $x_1 = 3$, $x_2 = 2$, con valore $z_L^1 = 23.5$. Si noti che tale soluzione è intera, e dunque, poiché $z_I^1 \leq z_L^1$, in tal caso $(3, 2)$ è anche la soluzione ottima intera. Dunque non occorre fare ulteriore branching per il nodo (P_1), che può dunque essere potato. Diciamo che (P_1) è *potato per ottimalità (S.A.)*. Si noti inoltre che la soluzione ottima di (P_0) avrà valore $z_I^0 \geq z_I^1 = 23.5$. $(3, 2)$ è la *soluzione intera corrente*, ovvero la miglior soluzione intera trovata fino a questo punto, il cui valore è $\bar{z} = 23.5$.

L'albero di Branch-and-Bound è ora il seguente. L'unica foglia non potata è $(P2)$ che

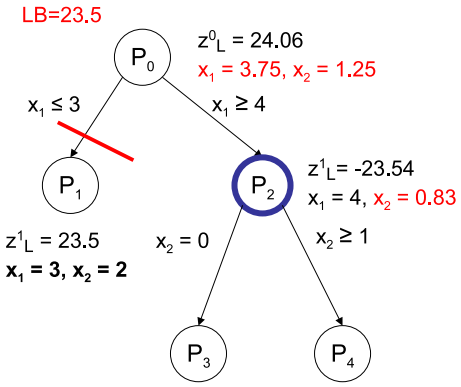




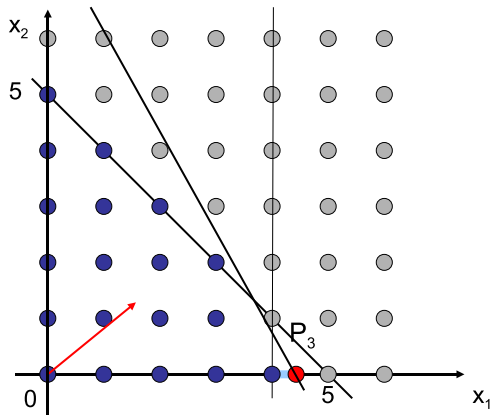
è l'unico problema attivo, ed è rappresentato nella figura successiva.
La soluzione ottima del rilassamento lineare di (P_2) è $x_1 = 4$, $x_2 = 0.83$, con valore $z_L^2 = 23.54$. Dunque $z_I^2 \leq 23.54$, e 23.54 è un upper-bound al valore ottimo di (P_2). Si noti che $\bar{z} = 23.5 < 23.54$, dunque (P_1) potrebbe avere soluzione migliore della soluzione intera corrente. Poiché la componente x_2 della soluzione ottima di (P_2) ha valore 0.83, facciamo branching su x_2, ottenendo i seguenti due sotto-problemi (P_3) e (P_4) di (P_2).

$$
\begin{array}{lrcll}
z_I^2 = \max & 5x_1 + \frac{17}{4}x_2 & & & \\
& x_1 + x_2 & \leq & 5 & \\
& 10x_1 + 6x_2 & \leq & 45 & \\
& x_1, & \geq & 4 & (P_3) \\
& x_2, & \leq & 0 & \\
& x_1, x_2 & \geq & 0 & \\
& x_1, x_2 & \in & \mathbb{Z} &
\end{array}
\quad , \quad
\begin{array}{lrcll}
z_I^2 = \max & 5x_1 + \frac{17}{4}x_2 & & & \\
& x_1 + x_2 & \leq & 5 & \\
& 10x_1 + 6x_2 & \leq & 45 & \\
& x_1, & \geq & 4 & (P_4) \\
& x_2, & \geq & 1 & \\
& x_1, x_2 & \geq & 0 & \\
& x_1, x_2 & \in & \mathbb{Z} &
\end{array}
$$

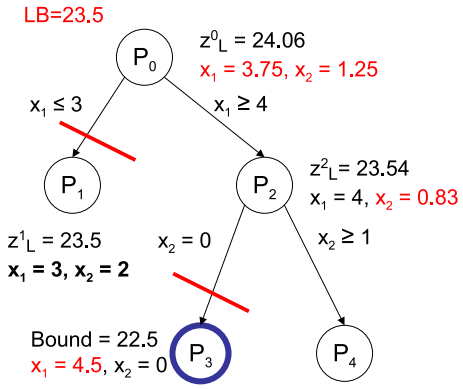
L'albero di Branch-and-Bound è ora il seguente.



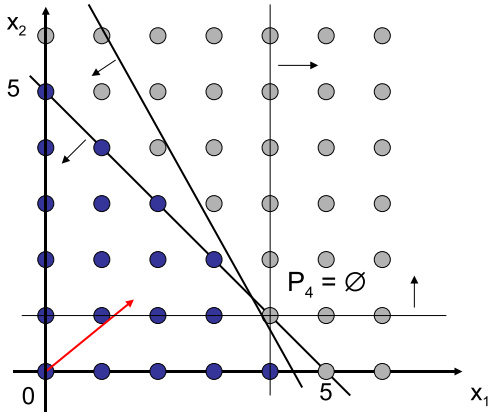
I nodi attivi sono $(P3)$ e (P_4). Risolviamo il rilassamento lineare di (P_3) (rappresentato in figura),



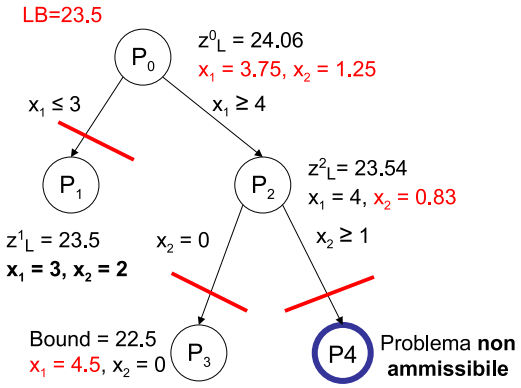
ottenendo soluzione ottima $x_1 = 4.5$, $x_2 = 0$, con valore $z_L^3 = 22.5$. Dunque la soluzione ottima intera di (P_3) avrà valore $z_I^3 \leq 22.5$, ma poiché abbiamo già determinato una soluzione ottima intera con valore 23.5, è inutile esplorare ulteriormente la regione ammissibile di (P_3) poiché sappiamo che non vi sarà nessuna soluzione intera di valore maggiore di $22.5 < 23.5$. Possiamo dunque *potare il nodo* (P_3) *perché non migliorante (N.M).* L'albero di Branch-and-Bound corrente, rappresentato nella figura successiva, contiene un unico problema attivo, ovvero (P_4).



Risolvendo il rilassamento lineare di (P_4), si determina che tale rilassamento non ha alcuna soluzione ammissibile, e dunque (P_4) non ha neppure soluzioni intere.



Possiamo dunque *potare il nodo* (P_4) *per inammissibilità (N.A.)*. L'albero di Branch-and-Bound corrente, rappresentato nella figura successiva, non ha alcun problema attivo, e dunque la soluzione intera corrente è la migliore possibile. Dunque $(3, 2)$ è la soluzione ottima di (P_0)



In quanto segue, esponiamo il metodo di Branch-and-Bound per programmazione lineare (mista) intera in maniera generale. Vogliamo risolvere il problema (P_0)

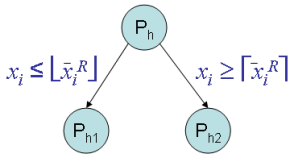
$$\begin{aligned} z_I = \max\, & c^T x \\ & Ax \leq b \\ & x \geq 0 \\ & x_i \in \mathbb{Z}, \qquad i \in I. \end{aligned}$$

dove I come al solito è l'insieme di indici delle variabili intere.

Il metodo sopra esemplificato corrisponde al metodo generale di Branch-and-Bound, dove sono state esplicitate la tecnica per la valutazione del bound ad ogni nodo e la tecnica per il branch.

Il bound è stato ottenuto con il rilassamento continuo del problema di programmazione lineare mista intera corrispondente ad ogni nodo: il rilassamento continuo è risolvibile in modo efficiente, ad esempio con il metodo del simplesso.

Il branch è stato effettuato in accordo con la seguente regola: sia $\bar{x}^R$ la soluzione del rilassamento continuo in un nodo P_h e $x_i = \bar{x}_i^R, i \in I$ una variabile con componente frazionaria $\varphi(\bar{x}_i^R) \neq 0$; allora si effettua un *branching dicotomico* secondo la seguente figura:



In caso di più variabili frazionarie, si può dare priorità alla x_i frazionaria con $\varphi(\bar{x}_i^R)$ più prossimo a 0.5.