

Modelli e Software per l'Ottimizzazione Discreta

0. Introduzione

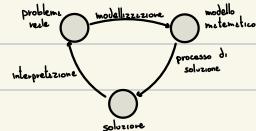
Possiamo effettuare l'ottimizzazione matematica se ci sono

- Un insieme di decisioni (che si influenzano e vicende)

- Possiamo misurare il risultato delle decisioni.

- Vogliamo cercare la decisione migliore.

Essendo un modello matematico dobbiamo prima modellare il problema reale, poi risolverlo (gestito da librerie apposite) e infine bisogna interpretare la soluzione nel caso reale.



I modelli matematici sono del tipo:

$$\begin{cases} \min f(x_1, \dots, x_n) \\ g_1(x_1, \dots, x_n) \geq 0 \\ \vdots \\ g_m(x_1, \dots, x_n) \geq 0 \end{cases}$$

vincoli

Purtroppo un problema di ottimizzazione arbitrario è indecidibile, dobbiamo quindi restringere i vincoli che si possono specificare.

Questo può però ridurre l'applicabilità dei modelli.

Un buon compromesso lo si trova nella programmazione lineare intera.

- funzione obiettivo e vincoli: interi

- variabili e valori: interi

Vedremo che non sono vincoli stringenti: ed è molto una tecnologia molto mettuta.

Purtroppo il problema rimane esponenziale, ma in 50 anni di ricerca le sue prestazioni sono migliorate 80000 volte.

FUN-FACT Un problema di ottimizzazione convessa ha come funzione obiettivo una funzione convessa (anziché una funzione lineare) e l'insieme delle soluzioni è convesso ($\bigcup V, \bigcap X$) (invece che un poliedro)

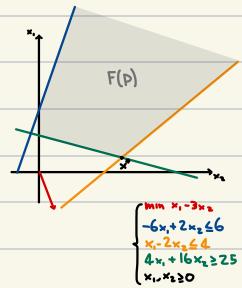
0.1 Problemi di Ottimizzazione

Sia $D = D_1 \times \dots \times D_n$ e $x = (x_1, \dots, x_n)$ dove $x_i \in D_i$. Sia $f: D \rightarrow \mathbb{R}$ e S un insieme finito di vincoli.

Allora un problema di ottimizzazione P si può formulare come:

$$\begin{cases} \min \text{ o } \max f(x) \\ S \\ x \in D \end{cases}$$

Un vincolo $c \in S$ è una funzione $c: x \rightarrow \{\text{Vero Falso}\}$ dove $x_i \leq x$ (alcune variabili del problema)



Def Ogni $x \in D$ si dica soluzione del problema P

Def Una soluzione si dice soluzione ammessa se soddisfa le condizioni in S .

$F(P)$ è l'insieme delle soluzioni ammesse di P .

Def Una soluzione ammessa $x^* \in F(P)$ si dice ottima se: $f(x^*) \leq f(x) \forall x \in F(P)$

Def Se il dominio D è discreto si parla di ottimizzazione discreta.

Se D è finito si parla di ottimizzazione combinatoria.

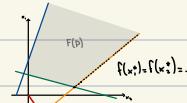
Def Se $F(P) = \emptyset$ P si dice impossibile

Def Se non esiste alcun limite inferiore a $f(x)$, P si dice illimitato.

Def Se P ammette una soluzione ottima si dice finito.

Nel corso assumeremo che P sia sempre impossibile, illimitato o con ottimo finito.

Quindi diremo che un problema è risolto se ne troviamo una soluzione ottima oppure se dimostriamo che il problema è impossibile o illimitato.



0.2 Ricerca

La ricerca in un problema di ottimizzazione significa risolvere una serie di restrizioni P_1, \dots, P_m .

Def Una restrizione è un problema P' ottenuto aggiungendo dei vincoli a P

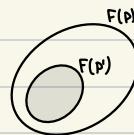
$$F(P') \subseteq F(P)$$

PROP

- Se $x \in F(P') \Rightarrow x \in F(P)$

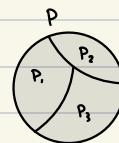
- Se x^* è ottimo di $P' \Rightarrow x^* \in F(P)$

OSS Trovare la soluzione ottima di una restrizione P' significa trovare un upper-bound del valore della soluzione ottima di P



Def Una ricerca si dice esauriente se l'insieme delle restrizioni copre tutto $F(P)$

$$\bigcup_{i=1}^m F(P_i) = F(P)$$



Metodi di risoluzione di un problema o di una sua restrizione:

1) generate-and-test

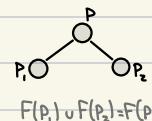
Generare tutte le soluzioni $x \in F(P)$ e scegliere la migliore.

Equivale a considerare le restrizioni di P dove x è fissato.

2) tree search

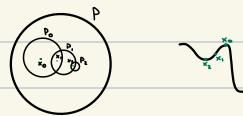
Costruisco un albero di restrizioni (due o più) la cui unione è uguale al genitore.

Le restrizioni sono generate in modo dinamico finché il problema è abbastanza semplice da poterlo risolvere.



3) local search (algoritmi euristici)

Possono trovare delle buone soluzioni, ma non sono né uniche né ottime. Questo perché la ricerca non è esauritiva e si possono trovare minimi locali.

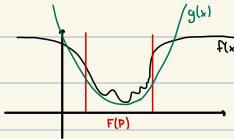


0.3 Rilassamento

Ovvero un problema R ottenuto da P in uno dei seguenti modi:

1. Eliminando dei vincoli $F(P) \subseteq F(R)$

2. Sostituendo la funzione obiettivo $f(x)$ con una sua approssimazione inferiore $g(x)$ $g(x) \leq f(x) \quad \forall x \in F(P)$



PROP

$$- F(R) = \emptyset \Rightarrow F(P) = \emptyset$$

- Se x^* ottimo di R , $x^* \in F(P)$, $g(x^*) = f(x^*) \Rightarrow x^*$ ottimo di P

OSS Trovare la soluzione ottima di un rilassamento R significa trovare un lower-bound del valore della soluzione ottima di P .

1. Programmazione Lineare

Un problema di programmazione lineare ha, in generale, la forma:

$$\begin{cases} \min / \max c^T x \\ \delta_i \leq x \leq b_i & \forall i=1, \dots, m \\ l_j \leq x_j \leq u_j & \forall j=1, \dots, n \end{cases}$$

dove c è un vettore di numeri, x uno di variabili
 $\sim \epsilon, z, =$
 $l_j \in \mathbb{R} \cup \{-\infty\}, u_j \in \mathbb{R} \cup \{+\infty\}$

1.1 Formulazioni Equivalenti

Possiamo tradurre problemi di programmazione lineari in formulazioni equivalenti.

In particolare possiamo convertire il problema in forma normale o canonica senza perdita di generalità.

$$\begin{cases} \min c^T x \\ Ax = b \\ x \geq 0 \end{cases}$$

forma standard

$$\begin{cases} \min c^T x \\ Ax \geq b \\ x \geq 0 \end{cases}$$

forma canonica

Per mostrare che le forme sono generali devo poter trasformare ogni problema da una forma all'altra.

1. max \Leftrightarrow min

$$\max C^T x = -\min -C^T x$$

2. $\geq \Leftrightarrow =$

$$a_i^T x \geq b_i \Leftrightarrow \begin{cases} a_i^T x - s_i = b_i \\ s_i \geq 0 \end{cases} \quad s_i: \text{variabile di surplus}$$

3. $\leq \Leftrightarrow =$

$$a_i^T x \leq b_i \Leftrightarrow \begin{cases} a_i^T x + s_i = b_i \\ s_i \geq 0 \end{cases} \quad s_i: \text{variabile di slack}$$

4. $= \Leftrightarrow \geq$

$$a_i^T x = b_i \Leftrightarrow \begin{cases} a_i^T x \geq b_i \\ a_i^T x \leq b_i \end{cases}$$

5. variabile libera $\rightarrow \geq 0$

$$x_i: \text{libera} \Leftrightarrow \begin{cases} x_i = x_i^+ - x_i^- \\ x_i^+, x_i^- \geq 0 \end{cases}$$

6. lower bound $\rightarrow \geq 0$

$$x_i \geq l_i \Leftrightarrow \begin{cases} x_i = x_i^+ + l_i \\ x_i^+ \geq 0 \end{cases}$$

7. upper bound $\rightarrow \geq 0$

$$x_i \leq u_i \Leftrightarrow \begin{cases} x_i = u_i - x_i' \\ x_i' \geq 0 \end{cases}$$

8. bounded $\rightarrow \geq 0$

$$l_i \leq x_i \leq u_i \Leftrightarrow \begin{cases} x_i = x_i^+ + l_i \\ x_i^+ + s_i = u_i - l_i \\ x_i^+, s_i \geq 0 \end{cases}$$

oss Queste trasformazioni peggiorano le prestazioni in modo limitato (si può reduplicare variabili e vincoli)

1.2 Intuizione Geometrica agli Algoritmi Risolutivi

Def $\{x \in \mathbb{R}^n \mid a_i^T x = c_i\}$ è un iperpiano (ovvero un sottospazio di dimensione $n-1$)

$\{x \in \mathbb{R}^n \mid a_i^T x \leq c_i\}$ è uno semispazio affine (ovvero una delle parti ottenute dividendo uno spazio con un iperpiano)

Def Si dice poliedro l'intersezione di un numero finito di semispazi affini ed iperpiani.

Se un poliedro è limitato si dice politopo.

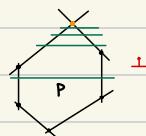


oss L'insieme delle soluzioni ammissibili $F(P)$ di un problema di programmazione lineare è un poliedro

oss Ogni politopo ha un numero finito di vertici.

TEO Se il problema di programmazione lineare $\min\{cx: x \in P\}$ ammette ottimo finito,

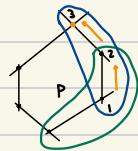
allora esiste un vertice ottimo.



Possiamo restringere la ricerca delle soluzioni ottima ad un numero finito di possibilità.

1.2.1 Algoritmo del Simplex

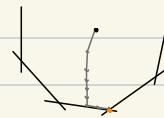
L'algoritmo del simplex usa questo teorema per muoversi tra i vertici fino a trovare una soluzione localmente ottima. Si può poi mostrare che l'ottimo locale è ottimo in generale.



L'复杂度 è esponenziale al caso peggiore, ma è poco più che lineare in quasi tutte le applicazioni reali.

1.2.2 Algoritmo a Barriera

Un algoritmo migliore dal punto di vista matematico (复杂度 polinomiale) è l'algoritmo a barriera, risolve una serie di sistemi che convergono ad un punto (non necessariamente un vertice)



OSS

- È difficile prevedere quale algoritmo avrà prestazioni migliori, spesso si eseguono in parallelo
- Se è richiesto il warm start (risoluzione di un problema simile a quello già risolto)
il simplex supera quello a barriera

1.3 Dualità

Un problema di programmazione lineare P può essere interpretato come un problema di ricerca ovvero si trovano soluzioni via via migliori cercando tra le soluzioni possibili.
L'ricerca genera quindi upper-bound via via miglior.



Un modo alternativo è quello di interpretarlo come un problema di inferenza D . (aggiunta di vincoli) ovvero vengono aggiunti vincoli via via più stretti sul problema, generando lower bound via via miglior.
Le variabili del duale sono quindi i vincoli del problema (una variabile per ogni vincolo)

Alllo stesso tempo per assicurarsi che i bound del duale siano soluzioni del problema, i vincoli del duale sono le variabili del problema (un vincolo per ogni variabile)

Eseguendo in parallelo la risoluzione del problema e del suo duale riusciamo a dimostrare la soluzione ottima.

In modo matematicamente rigoroso:

cerchiamo diseguaglianze del tipo $c^T x \geq c_0$ (ovvero lower-bound). In particolare cerchiamo c_0 massimo.

Dato il problema di ottimizzazione $P: \begin{cases} \min c^T x \\ Ax \leq b \end{cases}$

Vogliamo usare il vincolo $Ax \leq b$ per ottenere $c^T x \geq c_0$.

moltiplico quindi $Ax \leq b$ per $u \in \mathbb{R}_+$: $\frac{(u^T A)x}{c} \geq \frac{u^T b}{c_0}$, ottenendo così $c^T x \geq c_0$

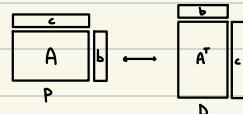
TEO - Lemma di Farkas

Una diseguaglianza $c^T x \geq c_0$ è valida per P se e solo se $\exists u \in \mathbb{R}_+^n | c \geq u^T A$ e $c_0 \leq u^T b$

Voglio quindi cercare in u per ottenere $c_0 = u^T b$ massima. (è un problema di ottimizzazione!)

$$D: \max_{c \in \mathbb{R}} \left| c_0 \mid c^T x \geq c_0, \forall x \in P \right| = \max_{c \in \mathbb{R}} \left| c_0 \mid c_0 \leq u^T b, c \geq u^T A \right| = \max_{u \in \mathbb{R}_+^n} \left| u^T b \mid c \geq u^T A \right| = \begin{cases} \max u^T b \\ c \geq u^T A \end{cases}$$

Lemme di Farkas



Dal lato pratico gli esercizi si possono risolvere in modo generale come segue:

u è libero, il suo segno non modifica l'ugualanza

$$\begin{cases} \min c^T x \\ Ax = b \\ u \geq 0 \end{cases} \rightarrow \begin{cases} \min u^T b + \text{f.o.} \\ c = u^T A + u^T I \\ u \text{ libero, } u \geq 0 \end{cases} \Rightarrow \begin{cases} \max u^T b \\ c \geq u^T A \\ u \text{ libero} \end{cases}$$

uso per mantenere l'ugualanza $c^T x \geq c_0$

posso togliere u trasformando l'espressione in diseguazione

OSS Il duale di un problema di programmazione lineare è a sua volta un problema di PL.

OSS Il duale di D è P . In altri termini il duale del duale di P è P

OSS Il duale dimostra la correttezza della soluzione $u^T b = c^T x^*$ ($\text{lower-bound} = \text{upper-bound}$)

TEO - d) dualità

1. $u \in F(D) \Leftrightarrow x^* \in F(P)$

2. se P ammette ottimo allora D ammette ottimo. ($u^T b = c^T x^*$)

P/D	Imp.	Sol finita	Illim.
Imp.	✓	✗	✓
Sol finita	✗	✓	✗
Illim.	✓	✗	✗

Es

$$\begin{array}{l}
 \begin{cases} \min 3x_1 + 2x_2 + 7x_3 + 5x_4 \\ x_2 - x_4 \geq 6 \\ x_1 + x_2 + x_3 = 4 \\ 3x_1 - x_3 + 5x_4 \leq 3 \\ x_1 \geq 0 \quad x_2 \leq 0 \quad x_3, x_4 \text{ liberi} \\ 0, x_2 \geq 0 \quad x_2 \leq 0 \end{cases} \\
 \xrightarrow{\quad} \begin{cases} \max 6\mu_1 + 4\mu_2 + 3\mu_3 \\ \mu_2 + 3\mu_3 + \mu_4 \leq 3 \\ \mu_1 + \mu_2 + \mu_4 = 2 \\ \mu_2 - \mu_3 = 7 \\ -\mu_1 + 5\mu_3 = 5 \\ \mu_1 \geq 0 \quad \mu_3 \geq 0 \quad \mu_2 \text{ libero} \quad 0 \leq 0 \quad 0 \leq 0 \end{cases} \\
 C^T x \geq c_0
 \end{array}$$

$$\begin{array}{l}
 \begin{cases} \max 3x_1 - 2x_2 + x_3 \\ x_1 + x_2 - x_3 \geq 2 \\ -x_2 + 6x_4 \leq -1 \\ x_1 \geq 0 \quad x_2 \leq 0 \quad x_3, x_4 \text{ liberi} \\ 0, x_2 \geq 0 \quad x_2 \leq 0 \end{cases} \\
 \xrightarrow{\quad} \begin{cases} \min 2\mu_1 - \mu_2 \\ \mu_1 \geq 3 \\ \mu_1 - \mu_2 \leq -2 \\ -\mu_1 + 6\mu_2 = 1 \\ \mu_1 \geq 0 \quad \mu_2 \geq 0 \end{cases} \\
 C^T x \leq c_0
 \end{array}$$

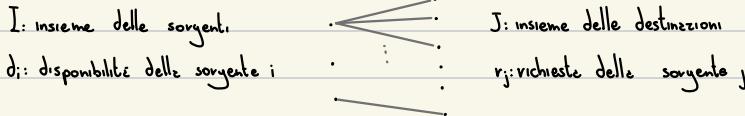
1.4 Modellazione di un Problema

Step per modellare problemi di PL.

- 1) Individuare gli insiemi e concetti del problema.
 - 2) Individuare i dati del problema (non variabili) (si dovrebbe trovare dati per ogni insieme)
 - 3) Individuare le variabili (cosa posso decidere? cosa è necessario per descrivere ad altri perché riescano a risolvere il problema)
 - 4) Individuare la funzione obiettivo
 - 5) Individuare i vincoli
- $\left. \begin{array}{l} \\ \end{array} \right\}$ tutto lineare

1.5 Esempi

1.5.1 Trasporto



c_{ij}: costo unitario del trasporto da i a j

x_{ij}: quantità spedita da i a j

$$\begin{cases} \min \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \\ \sum_{i \in I} x_{ij} \geq r_j \quad \forall j \in J \\ \sum_{j \in J} x_{ij} \leq d_i \quad \forall i \in I \\ x_{ij} \geq 0 \end{cases}$$

1.5.2 Dieta

I: sostanze nutritive

J: cibi

b_i : quantità minima della sostanza i

a_{ij} : quantità di sostanza i nell'elemento j

c_j : costo del cibo j

x_j : quantità di cibo j

$$\begin{cases} \min \sum_{j \in J} c_j x_j \\ \sum_{j \in J} x_j \cdot a_{ij} \geq b_i \quad \forall i \in I \\ x_j \geq 0 \quad \forall j \in J \end{cases}$$

oss || suo duale è:

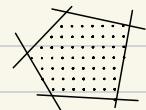
$$\begin{cases} \max \sum_{i \in I} u_i b_i \\ \sum_{i \in I} u_i \cdot a_{ij} \leq c_j \quad \forall j \in J \\ u_i \geq 0 \quad \forall i \in I \end{cases}$$

Questo problema ha una buona interpretazione del suo duale, se un'azienda vuole sintetizzare dei cibi con specifici valori nutritivi. Voglio creare cibi che costino meno dei rispettivi naturali, così da poter creare un guadagno.

2. Programmazione Lineare Intera

Rispetto alla programmazione lineare viene aggiunta la possibilità che alcune variabili siano limitate a valori interi.

$$\begin{cases} \min c^T x \\ a_i^T x = b_i \quad i=1, \dots, m \\ l_j \leq x_j \leq u_j \quad j=1, \dots, n \\ x_j \in \mathbb{Z} \quad \forall j \in J \subseteq \{1, \dots, n\} \end{cases}$$



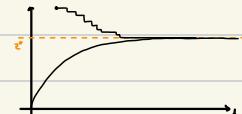
La programmazione lineare intera è meno restrittiva della programmazione lineare (posso aggiungere vincoli, ma non sono obbligato)

Def Se ogni variabile è intera parliamo di programmazione lineare intera pura, altrimenti di PLI mista.

I vertici, in generale, non contengono più le soluzioni ottime. Gli algoritmi visti per la programmazione lineare non sono più utili.

2.1 Algoritmo Branch&Bound

L'algoritmo Branch&Bound usa una strategia divide-and-conquer per trovare la soluzione ottimale. In particolare costruisce un albero di restrizioni sulle quali risolve dei rilassamenti per poter eliminare alcuni rami.



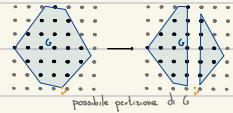
Sia $F: \mathcal{S} \rightarrow \mathbb{R}$ la funzione obiettivo da minimizzare. Sia $\bar{x} \in \mathcal{S}$ una soluzione trovata con un algoritmo euristico.

E sia $z = c(\bar{x})$ un upper-bound del problema, detto incumbent.

Se un euristico non esiste per il problema $z = +\infty$.

Sia A una coda che inizialmente contiene il problema; finché A non è vuota:

1. Scelgo un problema da A e risolvo un rilassamento (usualmente è il problema di PL, toglie i vincoli di intereza)
2. Se non ammette soluzione si può eliminare il ramo.
3. Se la soluzione trovata appartiene al dominio del problema iniziale ed è minore di z ho trovato un nuovo incumbent, sostituisco z (la soluzione è intera)
4. Se la soluzione del rilassamento è maggiore di z , si può eliminare il ramo.
5. Se il problema non è scartato lo divido ulteriormente $F(P) = \bigcup_{i \in I} F_i$ (PROP $\min_{x \in F(P)} c(x) \geq \min_{i \in I} \min_{x \in F_i} c(x)$)
6. Rimuovo il problema



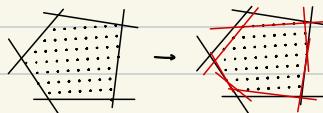
oss La scelta del rilassamento è particolarmente importante, deve generare problemi facili da risolvere e dei lower bound ragionevoli.

oss In ogni istante viene mantenuto un upper e lower bound, anche se termino l'esecuzione prim.

2.2 Algoritmo Cutting Planes

L'algoritmo Cutting Planes è un algoritmo più elegante in termini matematici.

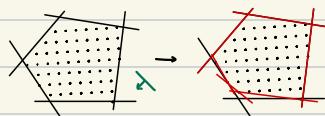
Consiste nello stringere i vincoli fino a far coincidere i vertici con delle soluzioni del problema.



A questo è possibile applicare gli algoritmi visti per la programmazione lineare.

Non è però semplice calcolare i vincoli in \mathbb{R}^n , il calcolo è esponenziale.

È però possibile calcolarli localmente alla soluzione (so la direzione della funzione obiettivo)



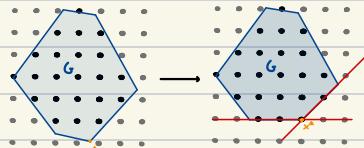
2.3 Algoritmo Branch&Cut

L'algoritmo Branch&Cut è un ibrido dei due algoritmi visti in precedenza.

Esso migliora la soluzione ottenuta dal rilassamento usando i passi di taglio:

- Una soluzione intera del rilassamento
- Un lower bound migliore, più efficace nel pruning

OSS: Una soluzione a cui si può pensare è l'arrotondamento della soluzione del rilassamento del PLI, ma esso è arbitrariamente lontano dall'ottimo del PLI.



2.4 Modellazione PLI/MIP

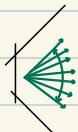
2.4.0 Rappresentabilità MIP

Un insieme $S \subseteq \mathbb{R}^n$ nelle variabili x è MIP-rappresentabile se esiste un sistema del tipo:

$$\begin{cases} Ax + By + Dz \geq b \\ y \in \{0,1\} \end{cases} \quad \begin{array}{l} \text{dove } x \text{ sono variabili continue} \\ \text{e } y \text{ sono variabili binarie} \end{array}$$

tale che la proiezione delle sue regione ammissibile su x è S .

Def: Un cono di recessione è una struttura intorno ad un poliedro formato da tutte le direzioni verso cui posso andare all'infinito senza uscire dal poliedro.



TEO: di Jeroslow
Un insieme S è MIP-rappresentabile \Leftrightarrow è l'unione di un numero finito di poliedri con lo stesso cono di recessione.

Vediamo alcune tecniche utili a modellare problemi con la programmazione lineare.

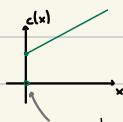
2.4.1 Variabili Discrete

$$x \in \{S_1, \dots, S_n\} \rightarrow \begin{cases} x = \sum_{i=1}^K S_i y_i \\ \sum_{i=1}^K y_i = 1 \\ y_i \in \{0,1\}, i=1, \dots, K \end{cases} \quad \text{dove } y_i = \begin{cases} 1 & \text{se } x = S_i \\ 0 & \text{se } x \neq S_i \end{cases}$$

2.4.2 Costi Fissi

$$c(x) = \begin{cases} cx + b & x > 0 \\ 0 & x = 0 \end{cases}$$

$$\begin{cases} c(xy) = cx + by \\ 0 \leq x \leq y, y \geq 0, l \end{cases}$$



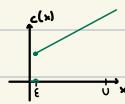
in generale è diverso dal reale, ve dimostrato

Se $y=0$ allora $x=0$ per il vincolo su x , ma non c'è vincolo sull'opposto.

Vz quindi mostrato che se $x=0$ il modello impone sempre $y=0$.

Oppure, per includere sempre $(0,0)$ tra le soluzioni:

$$\begin{cases} c(x,y) = cx + by \\ \forall y \in x \in U_y \\ y \in [0,l] \rightarrow y = \begin{cases} 1 & \text{se } x \geq l \\ 0 & \text{altrimenti} \end{cases} \end{cases}$$



2.4.3 AND/OR/NOT

\vee : OR, \wedge : AND, \neg : NOT

$$\begin{cases} x_1 \vee \neg x_2 \\ x_2 \wedge \neg x_3 \end{cases} \xrightarrow{\textcircled{1}} \begin{cases} x_1 \vee \neg x_2 \\ \neg(\neg x_2 \vee x_3) \end{cases} \xrightarrow{\textcircled{2}} \begin{cases} x_1 \vee (1-x_2) \geq 1 \\ 1 - (\neg x_2 \vee x_3) \geq 1 \\ x_2 \cdot c(x_2) \end{cases}$$

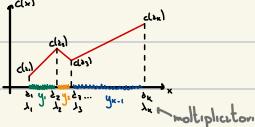
① Vuso $x = \neg(\neg x)$ e leggi di De Morgan $\frac{x_1 \wedge \bar{x}_2}{x_1 \vee \bar{x}_2} = \bar{x}_1 \vee \bar{x}_2$

$$\frac{x_1 \vee \bar{x}_2}{x_1 \vee x_2} = \bar{x}_1 \wedge \bar{x}_2$$

② $\neg x \rightarrow (1-x) \rightarrow x_1 \vee x_2 \rightarrow x_1 + x_2 \geq 1$

2.4.4 Funzioni Lineari a Tratti

Ovvero una funzione continua composta da un numero finito di segmenti



Il modello decide il segmento sul quale operare linearmente utilizzando le variabili y_i .

Le variabili λ possono essere $\neq 0 \Leftrightarrow$ sono adiacenti a $y_i = 1$

Il calcolo di x e $c(x)$ è gestito da somme lineari dei due moltiplicatori λ_i, λ_{i+1} . Variandone i valori posso ottenere tutti i detti compresi tra x_{i+1} e x_i oppure $c(x_{i+1})$ e $c(x_i)$.

Geogebra

$$x = \sum_{i=1}^k \lambda_i x_i$$

$$c(x) = \sum_{i=1}^k \lambda_i c(x_i)$$

$$\sum_{i=1}^k y_i = 1; \sum_{i=1}^k \lambda_i = 1$$

$$\lambda_i \leq y_i \leq y_{i+1} + y_i; \lambda_i \leq y_{i+1}$$

$$y_i \in \{0,1\}; \lambda_i \geq 0 \quad \forall i=1, \dots, k$$

2.4.5 Disgiunzioni I

Una disgiunzione è una lista di vincoli di cui almeno uno deve essere soddisfatto. ("o")

Assumo che tutte le variabili siano bounded.

Utilizzo delle variabili binarie $y_i = \begin{cases} 1 & \text{se l'esimo termine della disgiunzione deve essere soddisfatto} \\ 0 & \text{altrimenti} \end{cases}$

$$\begin{cases} z^T x \leq b_1 \vee z^T x \leq b_2 \\ l \leq x \leq u \end{cases} \longrightarrow \begin{cases} z^T x \leq b_1 + M(1-y_1) \\ z^T x \leq b_2 + M(1-y_2) \\ y_1 + y_2 = 1 \quad (\text{xor, se } y_1 + y_2 \geq 1 \text{ all}) \\ l \leq x \leq u \\ y_i \in \{0,1\} \quad \forall i=1,2 \end{cases}$$

dove M è un coefficiente sufficientemente grande da disabilitare il vincolo. (x è bounded per avere M finito)

2.4.6 Disgiunzioni II

Una disgiunzione creata del valore assoluto.

$$|z^T x| \leq b \rightarrow \begin{cases} z^T x \leq b \\ z^T x \geq -b \end{cases}$$

$$|z^T x| \geq b \rightarrow z^T x \geq b \vee z^T x \leq -b$$

$$\begin{cases} |x-2| \geq 1 \\ 0 \leq x \leq 4 \end{cases} \longrightarrow \begin{cases} x \geq 3y \\ x \leq 1+3y \\ y \in \{0,1\} \\ 0 \leq x \leq 4 \end{cases} \longrightarrow$$

2.5 Esempi di Modellazione MIP

2.5.1 Knapsack Binario

I: insieme di oggetti.

c_i : peso oggetto i

p_i : profitto oggetto i

C: capacità totale

x_i : $\begin{cases} 1 & \text{prendo l'oggetto } i \\ 0 & \text{altrimenti} \end{cases}$



$$\begin{cases} \max \sum_{i=1}^n x_i p_i \\ \sum c_i x_i \leq C \\ x_i \in \{0,1\} \end{cases}$$

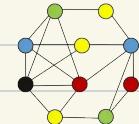
2.5.2 Vertex Coloring

Sia $G = (V, E)$ il grafo non orientato (V vertici, E rim) voglio assegnare ad ogni vertice un colore diverso

C: lista dei colori

$$x_{ic} = \begin{cases} 1 & \text{il nodo } i \text{ usa il colore } c \\ 0 & \text{altrimenti} \end{cases}$$

$$w_c = \begin{cases} 1 & \text{uso il colore } c \\ 0 & \text{altrimenti} \end{cases}$$

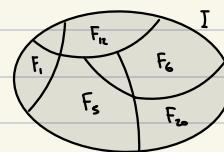


$$\begin{cases} \min \sum_{c \in C} w_c \\ \sum_{c \in C} x_{ic} = 1 \quad \forall i \in V \\ x_{ic} + x_{jc} \leq w_c \quad \forall (i,j) \in E \quad \forall c \in C \quad \text{"per ogni coppia le somme sono \leq 1 per ogni colore scelto"} \\ x_{ic}, w_c \in \{0,1\} \quad \forall i \in V, \forall c \in C \end{cases}$$

2.5.3 Set Covering

I = ground set

$J = \{F_1, \dots, F_n\}, \{F_i \subseteq I | i=1, \dots, n\}$ famiglia di sottoinsiemi di I



Vogliamo cercare un sottoinsieme di \mathcal{F} tale che se messo assieme I venga totalmente coperto

$$x_j = \begin{cases} 1 & \text{se prendo } F_j \\ 0 & \text{altrimenti} \end{cases}$$

$$\begin{cases} \min \sum_{j=1}^n x_j \\ \sum_{j | i \in F_j} x_j \geq 1 \quad (=1 \text{ se set partitioning}) \quad \forall i \in I \\ x_j \in \{0,1\} \quad \forall j = 1, \dots, n \end{cases}$$

2.5.4 Plant-Facility

J: insieme utenti

I: insieme locazioni possibili

d_{ij} : costo di attivazione della location i

c_{ij} : costo di servizio per utente j da location i

$$y_i = \begin{cases} 1 & \text{attivo la location } i \\ 0 & \text{altrimenti} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{se il cliente } j \text{ si serve dalla location } i \\ 0 & \text{altrimenti} \end{cases}$$

$$\begin{cases} \min \sum_{i \in I} d_{ij} y_i + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \\ \sum_{i \in I} x_{ij} = 1 \quad \forall j \in J \\ x_{ij} \leq y_i \quad \forall i \in I, j \in J \quad (\text{non posso collegarmi ad una facility non aperta}) \\ x_{ij} \in \{0,1\} \quad \forall i \in I, j \in J \quad y_i \in \{0,1\} \quad \forall i \in I \end{cases}$$

2.5 Modellazione in Python

```
from pycipopt import Model,quicksum

# Sets
products = ["gas", "chloride"]
components = ["nitrogen", "hydrogen", "chlorine"]

# Data
demand = {"gas": {"nitrogen": 1, "hydrogen": 3, "chlorine": 0},
           "chloride": {"nitrogen": 1, "hydrogen": 4, "chlorine": 1}}
profit = {"gas": 40, "chloride": 50}
stock = {"nitrogen": 50, "hydrogen": 180, "chlorine": 40}

def buildmodel():
    model = Model()

    x = {} #variabili
    for p in products:
        x[p] = model.addVar(name="x[%s]" % p)

    #objective
    model.setObjective(quicksum(profit[p] * x[p] for p in products), sense="maximize")

    #constraints
    for c in components:
        model.addCons(quicksum(demand[p][c] * x[p] for p in products) <= stock[c])
    model.data = x
    return model

if __name__ == '__main__':
    model = buildmodel()
    model.hideOutput() # silent mode
    model.optimize()
    print("Optimal value:", model.getObjVal())
    x = model.data
    for p in products:
        print("{} = {}".format(p, model.getVal(x[p])))
```

3. Programmazione con Vincoli

La programmazione con vincoli (o constraint programming) è un modo alternativo per la risoluzione dei problemi di ottimizzazione.

Def Un constraint satisfaction problem, CSP è definito da una tripletta $P=(X, D, C)$, dove:

X è l'insieme delle variabili $X=(x_1, \dots, x_n)$

D è l'insieme dei domini delle variabili $D=(D_1, \dots, D_n)$ dove $x_i \in D_i$; e ciascun dominio è finito. (oss I domini possono non essere numerici)

C è l'insieme dei vincoli $C=(C_1, \dots, C_m)$

Def Un vincolo è una coppia $C_i = (R_i, S_i)$ dove

S_i è il suo scope, indica quali variabili sono coinvolte nel vincolo.

R_i è una relazione nelle variabili S_i , ovvero un sottoinsieme del prodotto cartesiano $R_i \subseteq S_{i,1} \times \dots \times S_{i,n}$

oss È quindi un'enumerazione di tutti i possibili valori. Questo non è pratico, ma viene utilizzato a livello concettuale per generalità.

Def Una soluzione di un CSP P è una n-upla $A=(a_1, \dots, a_n)$ tale che $a_i \in D_i$.

Def Una soluzione A si dice ammessa se ogni vincolo è soddisfatto.

Ovvero, $\forall C_i \in C = (R_i, S_i)$ la proiezione di A sullo scope S_i appartiene a R_i .

oss È possibile aggiungere una funzione obiettivo e trovare così una soluzione ottima. In questo caso si parla di Constraint Optimization Problem.

oss La programmazione con vincoli non presenta problemi di modellazione, al contrario della programmazione lineare.

3.1 Metodi di Risoluzione

3.1.1 Ricerca

Per risolvere un problema CSP è possibile utilizzare l'algoritmo generale and test, tuttavia non è accettabile, è esponenziale.

Un algoritmo migliore è quello del tree search: consiste nel separare i domini scegliendo il valore delle variabili.

Quando in un nodo zero si fissare tutte le variabili nello scope di un vincolo allora posso verificarlo se il vincolo non è rispettato posso eliminare il nodo.

Questo presenta delle inefficienze es. $C_i = (R_i, \{x_1=5, x_2=3\})$, non riesco a riconoscere che $x_1=5, x_2=3$ viola già il vincolo.

Posso migliorare l'algoritmo con del lavoro preventivo, l'inferenza.

3.1.1 Inferenza

Derivare nuovi vincoli che sono logicamente implicati dai vincoli del problema (primi di Bayle)

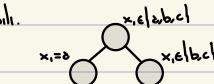
Def La propagazione dei vincoli è la principale tecnica di inferenza.

Consiste nel rimuovere dai domini delle variabili alcuni valori che non sono ammessibili perché violerebbero alcuni vincoli.

es Si dà $x_1, x_2 \in \{1, \dots, 10\}$, $C_1: x_1 - x_2 \geq 5$ allora posso rimuovere 5 e 6 dai domini, non soddisfano mai il vincolo.

L'operazione può aggiungere $C_2: x_1 \neq 5, 6$ $C_3: x_2 \neq 5, 6$

oss Le due tecniche vengono utilizzate assieme, ed ogni nodo del tree search si effettua la propagazione.



3.1.2 Vincoli Globali

Tradizionalmente lo scope dei vincoli è sempre di 2 variabili, o poco più.

Definiamo un vincolo globale un vincolo che può avere un numero qualsiasi di variabili nello scope.

Questi vincoli sono più compatti nella scrittura, inoltre danno al risolutore più informazioni per la propagazione.

ES $x_1, x_2, x_3 \in \{1, 2\}$

1. $x_1 \neq x_2$
 $x_2 \neq x_3$
 $x_3 \neq x_1$ \rightarrow il risolutore non può togliere nessun valore dal dominio.
 viene propagato un vincolo per volta.

2. $\text{all-different}(x_1, x_2, x_3) \rightarrow$ qui il risolutore si accorgere che il problema è impossibile.

VINCOLI GLOBALI COMUNI

1. $\text{element}(y, z, x_1, \dots, x_n) = \{ (e_i, f_i, d_1, \dots, d_n) \mid e \in D_y, f \in D_z, d_i \in D_{x_i}, \forall i, f = d_i \} = \{ (y, z, x_1, \dots, x_n) \mid z = x_y \} =$ l'elemento y deve essere uguale a z .

2. $\text{all-different}(x_1, \dots, x_n) = \{ (d_1, \dots, d_n) \mid d_i \in D_{x_i}; \forall i, d_i \neq d_j, \forall i \neq j \}$

3. $\text{gcl}(x_1, \dots, x_n, v_1, \dots, v_m, l_1, \dots, l_m, m_1, \dots, m_m) = \{ d = (d_1, \dots, d_n) \mid l_i \leq \text{occ}(v_i, d) \leq m_i; \forall i, v_i$ deve essere presente almeno l_i e al massimo m_i :

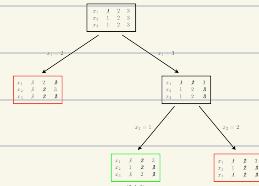
ogni valore v_i deve essere assegnato ad almeno l_i variabili ed al più a m_i variabili.

$\text{occ}(v_i, d)$ conta quante volte v compare in x_1, \dots, x_n .

4. $\text{linear}(z, b, x) = z \leq b$

ES-risoluzione di un problema di constraint programming

$\{\text{all-different}(x_1, x_2, x_3),$
 $2x_1 + x_2 \geq 6,$
 $x_2 \leq x_3,$
 $x_1, x_2, x_3 \in \{1, 2, 3\}$



4. SAT

Def Una formula logica è definita ricorsivamente come:

- una formula vuota, \emptyset (falsa per definizione)
- delle proposizioni atomiche x_i , possono essere vere o falsa.
- espressioni quali $\neg(F), (F) \vee (G), (F) \wedge (G)$ dove F e G sono formule logiche.

oss In modo meno formale: una formula è una funzione booleana $f(x_1, \dots, x_n): \{T, F\}^n \rightarrow \{T, F\}$

prop $F \vee G \Rightarrow "0"$

$$F \wedge G \Rightarrow "0"$$

$$F \neg G \Rightarrow \neg(F) \vee (G)$$

$$F \equiv G \Rightarrow (F \neg G) \wedge (G \neg F)$$

Def Una clausola booleana è una disgiunzione (\vee) di literali (proposizioni atomiche e negazioni)

es. $x_1 \vee x_3 \vee \bar{x}_4$

PROP

siano F e G due clausole

- $F \rightarrow G \Leftrightarrow F$ assorbe G ovvero ogni literal di F è literal di G
- $F \equiv G \Leftrightarrow F = G$

Def f è in forma congiuntiva normale (CNF) se è una congiunzione (\wedge) di clausole.

es. $(x_1 \vee \bar{x}_2) \wedge (x_3 \vee x_4) \wedge \bar{x}_5$

PROP posso portare ogni f in CNF se posso aggiungere variabili.

Un problema di soddisficiabilità booleana è nel trovare valori delle variabili che rendono vera una certa f

|| SAT è un esempio di constraint programming.

es. $(x_1 \vee \bar{x}_2) \wedge (x_3 \vee x_4) \wedge \bar{x}_5 \longrightarrow \begin{cases} x_1 \vee \bar{x}_2 \\ x_3 \vee x_4 \\ \bar{x}_5 \end{cases}$

Def Una clausola si dice di Horn se ha al più un literal non negato. (es. $\bar{x}_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4$)

$$x_k \vee \left(\bigvee_j \neg x_j \right) \Leftrightarrow \left(\bigwedge_j x_j \right) \rightarrow x_k$$

4.1 Risoluzione

4.1.1 Inferenza

Sia una funzione booleana contenente 2 clausole che contengono una un literal e l'altro la sua negazione.

$$\begin{cases} x_1 \vee x_2 \vee x_3 \\ \bar{x}_1 \vee x_2 \vee \bar{x}_4 \end{cases} \longrightarrow x_2 \vee x_3 \vee \bar{x}_4$$

Se $x_1 = T \Rightarrow \begin{cases} \text{①} = T, \text{ ②} = T \Leftrightarrow x_2 \vee \bar{x}_4 = T \\ \text{se } x_1 = F \Rightarrow \begin{cases} \text{②} = T, \text{ ①} = T \Leftrightarrow x_2 \vee x_3 \vee \bar{x}_4 \text{ deve essere vero} \end{cases} \end{cases}$

Si applica ricorsivamente queste regole, se infine si ottiene una clausola vuota allora la formula iniziale è soddisfacibile.

OSS Una implementazione è l'unit resolution che richiede che una delle clausole sia formata da un singolo literal.

Funziona bene con le clausole di Horn.

$$\left| \begin{array}{l} x_i \\ \bar{x}_i \end{array} \right|_{\text{vl...}} \circ \left| \begin{array}{l} \bar{x}_i \\ x_i \end{array} \end{array} \right|_{\text{vl...}} \text{resolution. l...l}$$

4.1.2 Algoritmo DPLL

È un algoritmo di ricerca ed albero, il branching viene effettuato fissando il valore di una variabile x_i o \bar{x}_i

Su ogni nodo è così possibile applicare la unit resolution (x_i e \bar{x}_i sono clausole formate da un solo literal)



$h = 5 + 1$

con algoritmo DFS, utilizzo della memoria costante, h .

5. Tecniche di Decomposizione

La più semplice decomposizione è nel caso in cui $A \in \mathbb{Z}^m \times \mathbb{Z}^n$

$$\begin{cases} \min c^T x + d^T y \\ Ax = b \\ By = f \\ x, y \in \mathbb{Z}^m \\ (x^*, y^*) \end{cases}$$

$$\boxed{\begin{array}{|c|c|} \hline A & |x| \\ \hline \hline B & |y| \\ \hline \end{array}} = \boxed{\begin{array}{|c|c|} \hline b & |f| \\ \hline \end{array}}$$

—————

$$\begin{array}{l} \textcircled{1} \quad \begin{cases} \min c^T x \\ Ax = b \\ x \in \mathbb{Z}^m \\ x^* \end{cases} \\ \textcircled{2} \quad \begin{cases} \min d^T y \\ By = f \\ y \in \mathbb{Z}^n \\ y^* \end{cases} \end{array}$$

Questo è molto più veloce con l'algoritmo branch&bound, n_1+n_2 al posto di $n_1 \cdot n_2$, infatti B&B crea un albero del tipo:

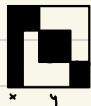


Con cutting planes non ci sarebbe il problema, ma è poco usato.

Vorremmo però applicare la decomposizione nel caso $A \in \mathbb{Z}^m \times \mathbb{R}^n$

$$\begin{array}{l} \text{Dantzig-Wolfe} \\ \text{Benders} \end{array}$$

5.1 Decomposizione di Benders



Vogliamo fissare le variabili x così da rendere il resto del problema più semplice (anche se non è bloccato)

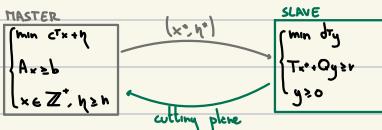
$$\begin{cases} \min c^T x + d^T y \\ Ax = b \\ T^T Q y \geq r \\ y \geq 0, x \in \mathbb{Z}^m \end{cases}$$

oss. y devono essere variabili continue

Scriviamo un rilettamento del problema che provi ad indovinare $d^T y$ con una variabile η , chiameremo questo problema master.

Inoltre scriviamo un secondo problema che verifichi la soluzione (il master ritorna solo un lower-bound).

Se la soluzione non è ottima lo slave ritorna un vincolo correttivo al master.



Le soluzioni dello slave ha 3 possibili casi:

1. slave è impossibile $\Rightarrow x^*$ è impossibile
2. l'ottimo dello slave è $y^* > \eta^*$ $\Rightarrow x^*$ è ammissibile, ma η^* è troppo basso.
3. l'ottimo dello slave è $y^* = \eta^*$ $\Rightarrow (x^*, \eta^*)$ ottimo

Definiamo ora i piani di taglio

$$1. \text{ (primo)} \begin{cases} \min_0 \\ Qy \geq r - Tx^* \\ y \geq 0 \end{cases} \xrightarrow{\text{duale}} \begin{cases} \max_{\pi^*} \pi^*(r - Tx^*) \\ \pi^* Q \leq 0 \\ \pi^* \geq 0 \end{cases}$$

$\exists \pi^* \geq 0$ t.c. $\pi^* T x \geq \pi^* r$ è valido ed è violato da x^*

$\pi^* = 0$ è una soluzione

So che il primale è impossibile, quindi il duale è o impossibile o unbounded

Il duale è unbounded quando $\exists \pi^* \geq 0$ t.c. $\pi^* T x \geq \pi^* r > 0 \Rightarrow \pi^* T x^* < \pi^* r$ qui c., il vincolo è

$\pi^* T x + \pi^* Q y \geq \pi^* r \Rightarrow (\pi^* T) x + (\pi^* Q) y \geq \pi^* r$ è un taglio valido per il problema che viene violato da x^*

$$2. \begin{cases} \min_0 d^T y \\ Qy \geq r - Tx^* \\ y \geq 0 \end{cases} \xrightarrow{\text{duale}} \begin{cases} \max_{\pi^*} \pi^*(r - Tx^*) \\ \pi^* Q \leq d^T \\ \pi^* \geq 0 \end{cases}$$

Entrambi ammettono ottimo finito, sia π^* ottimo del duale $\pi^* Q \leq d^T$
 $\pi^* (r - Tx^*) = y^* \geq \eta$

$$\eta \geq d^T y \Rightarrow \eta - d^T y \geq 0 \Rightarrow \eta + \pi^* T x + (\pi^* Q - d^T) y \geq \pi^* r \Rightarrow \eta \geq \pi^* (r - Tx)$$

ES

Nel caso di A=

$$\begin{cases} \min c^T x + d^T y_1 + d^T y_2 \\ Ax = b \\ T_1 x + Q_1 y_1 \geq v_1 \\ T_2 x + Q_2 y_2 \geq v_2 \\ y_1, y_2 \geq 0, x \in \mathbb{Z}^+ \end{cases}$$

Benders

$$\begin{cases} \min c^T x + \eta_1 + \eta_2 \\ Ax = b \\ x \in \mathbb{Z}^+ \end{cases}$$

$$\begin{cases} \min d^T y_1 \\ Q_1 y_1 \geq v_1 - T_1 x^* \\ y_1 \geq 0 \end{cases}$$

$$\begin{cases} \min d^T y_2 \\ Q_2 y_2 \geq v_2 - T_2 x^* \\ y_2 \geq 0 \end{cases}$$

5.2 Decomposizione di Dantzig-Wolfe



Vogliamo rimuovere i vincoli complicati così da rendere il resto del problema

più semplice (anche se non è blocco)

$$\begin{cases} \min c^T x \\ Ax = b \\ Dx = d \\ y \geq 0, x \in \mathbb{Z}^+ \end{cases}$$

La regione ammissibile è $X = \left| x \in \mathbb{R}^n \mid Ax = b \right| \cap \left| x \in \mathbb{R}^n \mid Dx = d \right|$

diciamo $P_x = \left| x \in \mathbb{R}^n \mid x = \sum_{g \in G} \lambda_g x_g, \sum_{g \in G} \lambda_g = 1, \lambda_g \geq 0 \right|$ dove $|x_g|_{g \in G}$ è l'insieme di vertici di P_x .

È ora possibile sostituire l'espressione per x nel problema iniziale.

$$\begin{cases} \min \sum_{g \in G} (c^T x_g) \lambda_g \\ \sum_{g \in G} (D x_g) \lambda_g = d \\ \sum_{g \in G} \lambda_g = 1 \\ \lambda_g \geq 0 \quad \forall g \in G \end{cases}$$

Così facendo ho ridotto il numero di vincoli del problema, ma ho introdotto un numero esponenziale di variabili. L'algoritmo introduce le variabili un po' alla volta, so che non dovrà aggiungere tutte.

TEO Se ho m vincoli e n variabili esiste una soluzione con $n-m$ variabili = 0 e m variabili $\neq 0$

Scrivo ora il duale del problema:

$$(P) \begin{cases} \min \sum_{g \in G} (c^T x_g) \lambda_g \\ \sum_{g \in G} (D x_g) \lambda_g = d \\ \sum_{g \in G} \lambda_g = 1 \\ \lambda_g \geq 0 \quad \forall g \in G \end{cases}$$

$$(D) \xrightarrow{\text{duale}} \begin{cases} \max \pi^T d + z \\ \pi^T (D x_g + z) \leq c^T x_g \quad \forall g \in G \\ \pi \text{ liberi} \end{cases}$$

MASTER

1. Sia $\bar{G} \subseteq G$ (inizialmente $\bar{G} = \emptyset$), risolvo il problema duale in \bar{G} e ottengo (π^*, z^*)

2. Risolvo il problema di pricing

$$\begin{cases} \min (c - D^T \pi^*)^T x \\ x \in P_x \end{cases}$$

La soluzione è \bar{x} , $z^* = (c - D^T \pi^*)^T \bar{x}$

Se $z^* \geq \bar{z}^*$ \Rightarrow nessun vincolo del problema iniziale è violato, l'algoritmo termina

Se $z^* < \bar{z}^*$ \Rightarrow ho trovato un nuovo vertice $\bar{G} = \bar{G} \cup \{\bar{x}\}$

3. Torno al passo 1

OSS Se Benders o Dantzig-Wolfe vengono terminati prima ritornano rispettivamente un lower e upper bound.



OSS Il master deve essere un problema di PL, non PLI

5.3 Generazione di Colonne

Estendiamo ora la decomposizione di Dantzig-Wolfe ai problemi di PLI.

Rivediamo la definizione di P_x , in particolare sia $Z = \{x | x \in P_x, x \in \mathbb{Z}^n\}$

Per definire Z abbiamo 2 metodi: convessificazione e discretizzazione

5.3.1 Convessificazione

Le soluzioni intere di Z sono ottenute come combinazione convessa dei vertici P_x .

$$Z = \{x | x = \sum_{g \in G} \lambda_g x_g, \sum_{g \in G} \lambda_g = 1, \lambda_g \geq 0, x \in \mathbb{Z}^n\} \quad (\text{come in Dantzig-Wolfe + vincolo di interezza})$$

Facendo così devo però riformulare il problema master includendo le variabili x .

$$\begin{cases} \min \sum_{g \in G} (c_g x_g) \lambda_g \\ \sum_{g \in G} (d_g x_g) \lambda_g = d \\ \sum_{g \in G} \lambda_g = 1 \\ x = \sum_{g \in G} \lambda_g x_g \\ \lambda_g \geq 0 \quad \forall g \in G \\ x \in \mathbb{Z}^n \end{cases}$$

5.3.2 Discretizzazione

Le soluzioni intere di P_x sono ottenute come combinazione convessa e coefficienti interi delle soluzioni di Z .

$$Z = \{x | x = \sum_{g \in G} \lambda_g x_g, \sum_{g \in G} \lambda_g = 1, \lambda_g \in [0, 1]\}$$

La riformulazione del problema diventa:

$$\begin{cases} \min \sum_{g \in G} (c_g x_g) \lambda_g \\ \sum_{g \in G} (d_g x_g) \lambda_g = d \\ \sum_{g \in G} \lambda_g = 1 \\ \lambda_g \in \{0, 1\} \quad \forall g \in G \end{cases}$$

