Ricerca Operativa M

Kevin Michael Frick Corinna Marchili Sofia Montebugnoli

25 agosto 2021

2. Programmazione matematica

(1) Problema di programmazione non lineare

> **R:** Un problema di programmazione non lineare minimizza una funzione obiettivo f(x) che rispetti i vincoli $h_j(x) = 0$ e $g_i(x) \geq 0$ per $j \in \{1, ..., p\}, i \in \{1, ..., q\}$, con f, h, g su cui non si effettua alcuna assunzione. Non è risolvibile efficientemente.

(2) Intorno

> **R:** Si definisce intorno una funzione che associa ad un insieme F un suo qualsiasi sottoinsieme $N : F \mapsto 2^F$ con N funzione, 2^F insieme di tutti i possibili sottoinsiemi di F.

(3) Intorno euclideo

> **R:** Si definisce intorno euclideo la funzione che associa ad ogni punto x della regione ammissibile un sottoinsieme di tutti i punti y che distano meno di ϵ da x.

(4) Intorno esatto

> **R:** Dato un problema (F, d) ed un intorno N, l'intorno N è detto esatto se un punto $f \in F$ è localmente ottimo rispetto ad N, allora f è globalmente ottimo.

(5) Ottimo locale

> **R:** Dato un problema (F, d) ed un intorno N, si dice che $f \in F$ è localmente ottimo in N se vale $d(f) \leq d(y)$ per ogni y nell'intorno di f, ovvero se f è migliore di ogni punto del sottoinsieme di F associato ad f.

(6) Combinazione convessa di due punti

> **R:** Dati due punti $x, y \in R^n$, si dice combinazione convessa di x e y qualunque punto $w \in R^n$ definito da $w = \lambda x + (1 - \lambda)y$. Al variare di λ la combinazione convessa w descrive tutti i punti del segmento.

(7) Insieme convesso

R: Un insieme $S \subseteq R^n$ è un insieme convesso se il segmento che congiunge due punti qualsiasi di S appartiene interamente ad S stesso.

Proprietà: Dati n insiemi S_i convessi, la loro intersezione è un insieme convesso.

(8) Funzione convessa

R: Dati un insieme $S \subseteq R^n$ convesso e una funzione $c : S \mapsto R^1$, c è una funzione convessa in S se la corda che congiunge due punti qualsiasi della funzione sta al di sopra della funzione stessa.

Proprietà: Comunque si scelga un valore soglia t per produrre una restrizione di S, questa è a sua volta un insieme convesso, ovvero l'insieme $S_t = \{w \in R^n : c(x) \leq t\}$.

(9) Funzione concava

R: Una funzione $c(x)$ definita su un insieme S convesso è detta concava se $-c(x)$ è convessa in S (prosaicamente, una funzione convessa si protende verso il basso mentre una funzione concava si protende verso l'alto). Una funzione lineare è sia concava che convessa.

(10) Problema di programmazione convessa

R: Se la funzione obiettivo è convessa, le funzioni h_j sono lineari per ogni j e le funzioni g_i sono concave per ogni i, allora il problema di programmazione è detto di programmazione convessa.

La programmazione convessa gode della proprietà secondo la quale dato un problema (F,c) con F convessa e c convessa su F, l'intorno euclideo $N_\epsilon t = \{y \in F : \|x - y\| \leq \epsilon\}$ è esatto per qualunque $\epsilon > 0$.

3. Programmazione lineare

(1) Forma generale

R: Sia A una matrice intera di m righe ed n colonne; b un vettore intero di m elementi, c un vettore intero di n elementi. Sia $(M, \bar{M})$ una partizione delle righe $\{1, \ldots, m\}$ di A e $(N, \bar{N})$ una partizione delle colonne $\{1, \ldots, n\}$ di A. La forma generale prevede un vincolo di uguaglianza per i vincoli in M e di $\geq$ per i vincoli in $\bar{M}$; le variabili in N sono non negative mentre quelle in $\bar{N}$ sono libere.

(2) Forma canonica

R: Nella forma canonica tutti i vincoli sono espressi da disequazioni e tutte le variabili sono non negative.

(3) Forma standard

R: Nella forma standard tutti i vincoli sono espressi da equazioni e tutte le variabili sono non negative.

(4) Equivalenza delle forme

R: Le tre forme sono equivalenti, poiché

1. una variabile libera può essere sostituita da $x_j^+ - x_j^-$, entrambe non negative;
2. un vincolo di uguaglianza può essere sostituito da due vincoli di disuguaglianza con versi opposti;
3. un vincolo di $\geq$ è equivalente ad un vincolo di uguaglianza se si sottrae una *variabile surplus* positiva al primo termine della disuguaglianza;
4. un vincolo di $\leq$ è equivalente ad un vincolo di uguaglianza se si aggiunge una *variabile slack* positiva al primo termine della disuguaglianza.

Applicare 1, 3, 4 aumenta il numero di colonne n, applicare 2 aumenta il numero di righe m.

(5) Assunzioni dell'algoritmo del simplesso

R:

1. Si considera solo la forma standard con $m < n$ (n.b. gli altri casi non hanno senso: $m = n$ ha una sola soluzione, $m > n$ non ammette soluzioni) non fa perdere di generalità all'algoritmo.
2. La matrice A contiene m colonne A_j linearmente indipendenti, ossia la matrice A è di rango m. L'algoritmo deve verificare se ciò non è soddisfatto e trovare una soluzione ottima.
3. La regione ammissibile non è vuota, ovvero esiste sempre una soluzione ammissibile, ed è limitata nella direzione di decrescita della funzione obiettivo, ovvero il valore della soluzione non tende a $-\infty$. L'algoritmo deve verificare se ciò non è soddisfatto e trovare una soluzione ottima.

(6) Base

R: Si dice base della matrice A un insieme m di colonne linearmente indipendenti. Indicando con $\mathcal{B}(i), (1, \ldots, m)$ gli indici delle colonne, indichiamo una base con $\mathcal{B} = \{A_{\mathcal{B}_1}, \ldots, A_{\mathcal{B}_m}\}$. Una base individua una sottomatrice quadrata m x m $B = [A_{\mathcal{B}_i}]$

(7) Soluzione base

R: Si dice soluzione base corrispondente a $\mathcal{B}$ la soluzione ottenuta sfruttando gli n-m gradi di libertà del sistema Ax=b per (1) fissare a 0 il valore delle variabili fuori base, (2) determinare *l'unica soluzione* del sistema quadrato $Bx_{\mathcal{B}} = b$ risultante, con $x_{\mathcal{B}}$ sottovettore di x corrispondente alle variabili base. Una SBA soddisfa sicuramente $Ax = b$.

(8) Soluzione base ammissibile (SBA)

R: Una soluzione è detta SBA se, oltre a soddisfare $Ax = b$, soddisfa anche il vincolo $x \geq 0$.

(9) Iperpiano

R: Dati uno spazio R^d di dimensione d, un vettore $h^T \neq 0$ di d elementi e uno scalare g, l'iperpiano è l'insieme di punti di R^d che verifica l'uguaglianza $h^T x = g$. Un iperpiano suddivide lo spazio in due semispazi, rispettivamente definiti da $h^T x \geq g$ e $h^T x \leq g$

(10) Proprietà dei semispazi

R: Dato che uno spazio è un insieme convesso e un semispazio è un sottoinsieme di uno spazio, il semispazio è un insieme convesso. Di conseguenza, un'intersezione di semispazi è un insieme convesso a sua volta.

(11) Politopo convesso

R: L'intersezione di un numero finito di semispazi è detta politopo convesso. Talvolta questa intersezione deve essere limitata e non vuota. La dimensione massima di un politopo è la minima dimensione di uno spazio che lo può contenere, pertanto la dimensione di un politopo può essere inferiore a quella dello spazio in cui è rappresentato. L'insieme dei vincoli di un LP definisce un politopo.

(12) Regione ammissibile di un LP

R: La regione ammissibile di un LP è un politopo (perché ogni LP può essere sempre posto in forma canonica ed avere una regione ammissibile definita dall'intersezione di semispazi.)

(13) Faccia, faccetta, spigolo, vertice

R: Dati un politopo P, un iperpiano H ed un semispazio HS definito da H, sia f l'intersezione tra P e HS. Se l'intersezione f è non vuota e rappresenta un sottoinsieme dell'iperpiano H, allora f è detta *faccia* di P. Se il politopo è di dimensione d:

1. una faccia di dimensione d-1 è detta *faccetta*;
2. una faccia di dimensione 1 è detta *spigolo*;
3. una faccia di dimensione 0 è detta *vertice*.

(14) Combinazione convessa di p punti

R: La combinazione convessa di p punti $x^{(1)}, \ldots, x^{(p)} \in R^n$ è il punto $w = \sum_{i=1}^{p} \alpha_i x^{(i)}$, con $\sum_{i=1}^{p} \alpha_i = 1$ e $\alpha_i \geq 0$.

(15) Vertici

R: Ogni punto di un politopo è combinazione convessa dei vertici, ogni combinazione convessa dei vertici è un punto del politopo. Un vertice **non** è combinazione stretta $(0 < \lambda < 1)$ di due punti distinti del politopo.

(16) Vertici e SBA

R: Dato un politopo P definito dai vincoli di un LP, condizione necessaria e sufficiente perché un punto sia un vertice è che il corrispondente vettore x sia una SBA.

(17) Vertice ottimo

R: Dato un qualunque problema LP, esiste sempre un vertice ottimo (e, quindi, una SBA ottima). Ogni combinazione convessa di vertici ottimi è ottima.

(18) SBA degenere

R: Dal momento che una base $\mathcal{B}$ determina una soluzione base univoca, se due SBA sono diverse sono prodotte da basi diverse. Due basi diverse possono produrre la stessa SBA (SBA degenere) se, oltre agli n-m zeri delle variabili fuori base, contiene anche altre variabili a zero.

Proprietà: se due basi $\mathcal{B}$ e $\bar{\mathcal{B}}$ producono la stessa SBA x, allora x è degenere.

4. **Algoritmo del simplesso** Si vuole minimizzare la funzione obiettivo $c^T x$, con x appartenente ad un politopo F espresso come LP in forma standard, sotto le assunzioni (da verificare) che A sia di rango m, F sia non vuoto e limitato nella direzione di decrescita della funzione obiettivo. Un insieme di m colonne linearmente indipendenti è detto *base*, e vi corrisponde una soluzione del sistema Ax=b che si ottiene imponendo il valore 0 alle variabili corrispondenti a colonne non in base e risolvendo il sistema quadrato risultante. Se la soluzione soddisfa anche $x \geq 0$ è detta SBA, che individua un vertice del politopo. Una delle SBA è sempre una soluzione ottima.

(1) SBA adiacenti

R: Due SBA distinte, corrispondenti a due basi $\mathcal{B}$ e $\bar{\mathcal{B}}$, si dicono adiacenti se le due basi differiscono tra loro di una sola colonna, ovvero esistono $A_j \in \mathcal{B}$ e $A_k \in \bar{\mathcal{B}}$ tali che

$$\bar{\mathcal{B}} = (B \backslash \{A_j\}) \cup \{A_k\}$$

(2) Principio dell'algoritmo del simplesso

R: Partendo da una SBA qualunque, iterativamente si sostituisce la SBA corrente con una SBA ad essa adiacente e di costo non maggiore, fino a trovare la SBA ottima.

(3) Spostamento da SBA a SBA (caso non degenere)

R: Sia x_0 una SBA corrispondente alla base $\mathcal{B}$. La soluzione, comprensiva delle variabili fuori base, è $x_0^T = (y_{10}, y_{20}, \ldots, y_{m0}, 0, \ldots, 0) \geq 0$. Poiché $Ax_0 = b$, ne deriva $\sum_{i=1}^{m} y_{i0} A_{\mathcal{B}(i)} = b$. Essendo le colonne in base linearmente indipendenti, le colonne fuori base si possono esprimere come combinazione lineare con dei coefficienti y_{ij}: $\sum_{i=1}^{m} y_{ij} A_{\mathcal{B}(i)} = A_j$. Moltiplicando per $\vartheta > 0$ la seconda equazione e sottraendola dalla prima, geometricamente all'aumentare del valore di ϑ si percorrono tutte le soluzioni corrispondenti ai punti del segmento che unisce due vertici adiacenti relativi a due SBA. Di conseguenza:

1. $\vartheta = 0$: esprime la soluzione base iniziale;
2. aumento di ϑ: si trova una famiglia di nuove soluzioni ammissibili con m+1 componenti positive. Il valore di ϑ può aumentare fino a ϑ_{max}, oltre il quale la differenza tra i coefficienti che si moltiplica per $A_{\mathcal{B}(i)}$ diventa negativa (e quindi non ammissibile per LP): $\vartheta_{max} = min_{i:y_{ij}>0} \frac{y_{i0}}{y_{ij}}$
3. $\vartheta = \vartheta_{max}$: la relazione esprime una nuova SBA con m componenti positive. Nell'insieme di colonne corrispondenti è inclusa A_j ma non la colonna il cui coefficiente è azzerato.

(4) Spostamento da SBA a SBA (caso degenere)

R: Se x_0 è degenere, contiene almeno uno zero tra le variabili in base ($y_{i'0} = 0$). Se il coefficiente corrispondente $y_{i'j} > 0$, allora $\vartheta_{max} = 0$ e la nuova soluzione (pur comprendendo colonne in base diverse) è identica alla precedente.

(5) Spostamento da SBA a SBA (caso illimitato)

R: Nel caso in cui nessun coefficiente y_{ij} fosse positivo, il valore di ϑ può crescere illimitatamente senza violare l'ammissibilità delle soluzioni. Ciò significa che il politopo è illimitato nella direzione di decrescita della funzione obiettivo, violando la terza assunzione sul simplesso, quindi il problema è risolto ed il valore della soluzione è $-\infty$.

(6) Pivoting

R: Sia $\vartheta_{max} = min_{i:y_{ij}>0} \frac{y_{i0}}{y_{ij}} = \frac{y_{l0}}{y_{lj}}$. y_{lj} è detto *pivot*, e tramite l'operazione si definisce una nuova SBA $\tilde{y_{i0}}$ che assume valore $y_{i0} - \vartheta_{max} y_{ij}$ nelle righe diverse da quella del pivot, ϑ_{max} nella riga del pivot.

(7) Operazione elementare di riga

R: Le operazioni elementari di riga comprendono moltiplicare una riga per una costante non nulla e sommare un multiplo di una riga ad un'altra. La soluzione del sistema lineare non varia.

(8) Costo relativo

R: Rappresenta la variazione del costo per ciascuna unità di x_j che entra in base. Poiché il simplesso prevede la minimizzazione del costo, è conveniente far entrare in base una colonna A_j solo se $\bar{c_j} < 0$. Il costo relativo di una colonna in base deve essere nullo.

(9) Tableau: informazioni generali

R: A qualsiasi passo dell'algoritmo del simplesso, il tableau riporta una matrice identità in corrispondenza delle colonne in base e contiene una rappresentazione compatta dei coefficienti di un sistema lineare equivalente a Ax=b.

(10) Tableau: colonna 0

R: La colonna 0 riporta i valori delle variabili base nella SBA corrispondente. Per ogni colonna fuori base, i coefficienti y_{ij} corrispondenti sono i valori nella colonna j-esima del tableau.

(11) Tableau: riga 0

R: La riga 0 fornisce il costo relativo per le colonne fuori base, mentre nella colonna 0 fornisce l'opposto del valore z_0 della soluzione base attuale.

(12) Criterio di ottimalità

R: Se tutti i costi relativi sono non negativi, allora la soluzione attuale x_0 è ottima. Questo perché considerando la riga 0 del tableau attuale si ha che il valore z di qualunque soluzione è dato dal valore z_0 più una combinazione lineare degli attuali costi relativi. Dato che le variabili devono essere non negative, se i costi relativi delle variabili fuori base sono non negativi allora qualsiasi altra SBA può avere un valore maggiore o uguale all'attuale.

(13) Fase 2

R: Nella fase 2 dell'algoritmo del simplesso manca un metodo per costruire un tableau da una base iniziale, ovvero la verifica che la matrice A sia di rango m e che la regione ammissibile sia non vuota.

(14) Regola di Dantzig

R: Per favorire la convergenza dell'algoritmo in un tempo ridotto, **entra in base** la colonna A_j con il costo relativo più negativo - nei casi di parità, la si risolve in maniera casuale. La regola di Dantzig non garantisce la convergenza in caso di degenerazione ciclante.

(15) Regola di Bland

R: Metodo deterministico che evita la degenerazione ciclante (ma è meno efficiente di Dantzig in termini di velocità di convergenza), prevede che **entra in base** la colonna A_j di indice minimo tra quelle con costo relativo negativo; in caso di parità **esce dalla base** la colonna A_j di indice minimo.

(16) Fase 1

R: Consente di ottenere una SBA iniziale per la fase 2. La fase 1 prevede:

1. cambiare il segno ai termini noti negativi;
2. aggiungere m variabili artificiali non negative, sommandone una a ciascuna equazione.

Si ottiene una SBA corrispondente alla base con le variabili artificiali. Poiché la soluzione non è ammissibile per il problema originale, si minimizza ζ, ovvero il valore della funzione obiettivo nelle variabili artificiali. Se il sistema originale ammette una SBA $\zeta = 0$, altrimenti $\zeta > 0$:

1. se $\zeta = 0$ e le variabili artificiali sono tutte fuori base il tableau ha una base nelle variabili vere, altrimenti se una variabile artificiale rimane in base e non si può portare fuori significa che la matrice non è di rango m e si può eliminare una riga;
2. se $\zeta > 0$ il problema originale non ha nessuna soluzione ammissibile, e si viola l'assunzione sulla regione ammissibile non vuota; Il caso $\zeta < 0$ non può mai verificarsi, poiché la funzione obiettivo artificiale è uguale alla somma di variabili non negative.

5. **Dualità**

(1) Dualità forte

R: Se un problema primale di LP ha soluzione ottima finita, allora anche il suo duale la ha e le due soluzioni sono uguali.

(2) Dualità debole

R: Il costo di ogni soluzione ammissibile del primale è maggiore o uguale al costo di ogni soluzione ammissibile del duale.

Tutte le soluzioni ammissibili e non ottime del primale sono inammissibili per il duale, tutte le soluzioni ammissibili del duale con costo inferiore all'ottimo sono inammissibili per il primale. L'unica soluzione ammissibile per entrambi i problemi è quella ottima.

(3) Duale del duale

R: Il duale del duale è il primale.

(4) Coppie possibili di soluzioni primali-duali

R:

duale/primale	ottimo finito	illimitato	impossibile
ottimo finito	Sì	No	No
illimitato	No	No	Sì
impossibile	No	Sì	Sì

(5) Lemma di Farkas

R: Solo uno tra i due sistemi $Ax = b, x \geq 0$ e $y^T A \leq 0, b^T y > 0$ è possibile. Il primo è possibile se e solo se il secondo è impossibile.

(6) Complementary slackness

R: Siano x, π due soluzioni ammissibili per il primale e il duale. Esse sono ottime se e solo se per ogni i, o l'i-esima variabile duale è nulla o l'i-esimo vincolo primale è risolto all'uguaglianza; per ogni j, o il j-esimo vincolo duale è risolto all'uguaglianza o la j-esima variabile primale è nulla.

(7) Algoritmo del simplesso duale

R: L'algoritmo del simplesso duale mantiene una soluzione ammissibile per il duale e, tramite pivoting, sposta la soluzione verso l'ammissibilità del primale. Poiché si cerca di eliminare l'inammissibilità dal tableau, si sceglie una riga con $y_{i0} < 0$ e il pivot viene scelto come il valore massimo di y_{0j}/y_{ij} per cui il valore di $y_{ij} < 0$. La scelta del pivot garantisce il minimo aumento del valore della soluzione. Se non dovesse esistere alcun valore negativo di questo tipo il duale può crescere illimitatamente, pertanto il primale sarebbe impossibile.

(8) Affinità-divergenze tra il simplesso primale e duale nel risolvere LP

R: Nel simplesso primale ci si sposta da SBA a SBA, in quello duale da SB inammissibile a inammissibile. Nel primo caso z parte da un valore superiore all'ottimo e decresce, nel duale da uno inferiore e cresce. Nel primale si sceglie prima la variabile che entra in base, nel duale quella che la lascia; poi nel primale si sceglie la variabile che lascia la base, nel duale quella che entra. Nel primale il pivot è dato dal minore tra i valori positivi, nel duale dal maggiore tra quelli negativi.

(9) Affinità-divergenze tra il problema primale e duale

R: Il duale di un problema di minimo è un problema di massimo nel quale:

- i vincoli di uguaglianza ($a_i^T x = b_i$) corrispondono a variabili π_i libere in segno, i cui costi sono i termini noti del primale
- i vincoli di disuguaglianza ($a_i^T x \geq b_i$) corrispondono a variabili $\pi_i \geq 0$, i cui costi sono i termini noti del primale
- le variabili non negative primali ($x_j \geq 0$) corrispondono a vincoli di disuguaglianza $\pi^T A_j \leq c_j$, i cui termini noti sono i costi del primale
- le variabili libere in segno primali corrispondono a vincoli di uguaglianza $\pi^T A_j = c_j$ i cui termini noti sono i costi del primale

(10) Duale e fase 1

R: Non si effettua una fase 1 nel simplesso duale perché in genere lo si adotta in situazioni in cui è dato il tableau ottimo di un LP primale, si aggiungono uno o più vincoli che rendono inammissibile la soluzione e si vuole la nuova soluzione ottima.

(11) Analisi di sensitività

R: Esamina l'effetto di una variazione di uno dei dati di input, in particolare del termine noto di un vincolo o di un coefficiente della funzione obiettivo.

(12) Intervallo di stabilità

R: Se un dato viene modificato ma rimane all'interno dell'intervallo di stabilità, la base corrente rimane ottima. Diventa necessario ricalcolare esclusivamente il valore delle variabili alla luce dei nuovi dati.

(13) Modifica del termine noto di un vincolo

R: Se cambia solamente un termine noto di un vincolo, tenendo conto del fatto che i termini noti si usano solo per calcolare i valori della colonna 0 mentre i costi relativi non cambiano si ha che la base ottima attuale rimane tale per tutti i valori del termine noto per cui la soluzione rimane ammissibile (ovvero, il prodotto tra l'inversa della base e il nuovo vettore dei termini noti rimane non negativo).

(14) Modifica di un coefficiente della funzione obiettivo

R: I coefficienti della funzione obiettivo si adoperano esclusivamente per il calcolo della riga 0 del tableau, pertanto il valore della soluzione cambia solo se la variabile cui è associato il coefficiente è in base. Perché la SBA resti ottima si verifica che i costi relativi restino non negativi.

(15) Prezzi ombra

R: Il prezzo ombra di una risorsa è l'incremento della funzione obiettivo che si ottiene incrementando di una unità la quantità disponibile della risorsa stessa, purchè il valore incrementato rimanga all'interno dell'intervallo di sensitività.

Proprietà: il valore ottimo della variabile duale π_i fornisce il prezzo ombra della risorsa associata al vincolo i.

6. Programmazione Lineare Intera

(1) Programmazione lineare intera

R: Un problema di programmazione lineare intera aggiunge come ulteriore vincolo che le variabili decisionali assumano valori interi. Geometricamente, la regione ammissibile di un ILP è costituita dai punti con coordinate intere contenute nel politopo definito dai vincoli dell'LP.

(2) Rilassamento continuo dell'ILP

R: Problema LP corrispondente all'ILP cui è rimosso il vincolo d'interezza sulle variabili.

Proprietà Se z(ILP) e z(LP) sono i valori delle soluzioni rispettivamente di un ILP di minimizzazione e del suo rilassamento continuo, si ha $z(LP) \leq z(ILP)$ perché la regione ammissibile dell'ILP è contenuta in quella dell'LP.

(3) Unimodularità

R: Una matrice **quadrata** intera B è unimodulare se $det(B) = \pm 1$.

(4) Totale unimodularità

R: Una matrice **rettangolare** intera A è totalmente unimodulare (TUM) se ogni sua sottomatrice quadrata non singolare è UM, ovvero se $\det B \in \{0, +1, -1\} \forall B$ sottomatrice di A. Si noti che l'unico caso in cui una sottomatrice B può non essere unimodulare senza pregiudicare la totale unimodularità di A è quello in cui B è singolare ($\det B = 0$).

Proprietà Se una matrice A è TUM, i vertici della regione ammissibile $\{x : Ax = b, x \geq 0\}$ (LP in forma standard) o di $\{x : Ax \leq b, x \geq 0\}$ (LP in forma generale/canonica) sono interi per ogni b intero.

(5) Algoritmi per ILP

R: Per matrici non TUM si adottano due tipologie di algoritmi, talvolta combinati tra loro:

1. *cutting plane*: algoritmi basati su piani di taglio;
2. *branch-and-bound*.

(6) **Cutting Plane**

R: Algoritmo che aggiunge all'ILP una condizione lineare (taglio valido) che :

- non elimini alcuna soluzione ammissibile intera
- elimini una parte della regione ammissibile contentente l'ottimo non intero del rilassamento continuo

(7) Parte intera

R: La parte intera di $y \in R^1$ è $\lfloor y \rfloor = \max q \in Z : q \leq y$.

(8) Tagli di Gomory

R: Definita la parte frazionaria di y_{ij} come $f_{ij} = y_{ij} - \lfloor y_{ij} \rfloor$, con $0 \leq f_{ij} \leq 1$, il taglio di Gomory corrispondente alla riga i (*riga generatrice*) è $\sum_{A_j \notin \mathcal{B}} f_{ij} x_j \geq f_{i0}$ (ovvero la sommatoria delle variabili fuori base, ciascuna moltiplicata per la corrispondente parte frazionaria, ha un valore $\geq$ della corrispondente parte frazionaria del termine noto sulla stessa riga).

Teorema Se si aggiunge al tableau finale di un LP un taglio di Gomory non si elimina alcun punto ammissibile e il nuovo tableau contiene una base non ammissibile per il primale, se y_{i0} non è intero, ed ammissibile per il duale.

(9) Convergenza dell'algoritmo di Gomory

R: In assenza di degenerazioni l'algoritmo converge perché ad ogni iterazione si considera una base diversa dalle precedenti e di costo maggiore; in caso di degenerazioni il metodo converge se (1) si sceglie sempre la prima riga frazionaria e (2) si sceglie il pivot del simplesso duale secondo un metodo lessicografico simile alla regola di Bland.

(10) Gomory e rilassamento continuo illimitato inferiormente

R: Se la regione ammissibile è illimitata nella direzione di decrescita della funzione obiettivo generalmente conterrà anche punti interi all'infinito ed il problema intero avrà soluzione illimitata; esiste una tipologia geometrica per cui l'LP è illimitato ma l'ILP è impossibile.

(11) **Branch and bound**

R: Gli algoritmi branch and bound per ILP dividono un problema iniziale P^0 in più sottoproblemi P^k mediante due o più condizioni esaustive. La mutua esclusività tra le condizioni solitamente favorisce la convergenza dell'algoritmo ma non è richiesta. In seguito, si continua a creare più sottoproblemi da ogni problema a meno che $x^{(k)}$ non sia intero o il rilassamento di P^k sia impossibile. In questi casi, P^k è risolto.

Se si continua a ramificare fino a che ogni nodo non può più generare figli, la soluzione ottima è quindi data dalla foglia dell'albero dei sottoproblemi che ha costo minimo, nel caso di un problema di minimizzazione. Gli algoritmi B&B applicano il principio del *bounding* che rende più efficiente la computazione, dal momento che se si completasse l'albero ramificando finché nessun nodo può più generare figli la soluzione ottima sarebbe data dalla soluzione della foglia di costo minimo. Poiché tutti i coefficienti sono interi, dato il valore della soluzione del problema P_1 nessuna soluzione prodotta da P_1 o dai suoi discendenti (se scendo nell'albero, restringo la regione ammissibile) può avere un valore inferiore a $\lceil L_k \rceil$ (con $\lceil y \rceil$ più piccolo intero q tale che $q \geq y$).

(12) Albero decisionale binario

R: Strumento adottato per rappresentare i problemi e le relative soluzioni. Accanto ad ogni nodo si indica il valore della soluzione L_k del rilassamento continuo, accanto ad ogni ramo la condizione che produce il nodo figlio.

(13) Uccisione di un nodo

R: Se per un problema P_k si ha $\lceil L_k \rceil$ maggiore o uguale al valore della miglior soluzione intera trovata fino ad allora, non è conveniente risolvere il problema P_k e di conseguenza il nodo corrispondente viene ucciso.

Nel caso di problemi di massimizzazione si adopera l'upper bound, dato da $\lfloor U_k \rfloor$ (parte intera più grande $\leq U_k$), e si uccide un nodo k se $\lfloor U_k \rfloor \leq z$.

(14) Strategie di esplorazione

R: Due decisioni influenzano un algoritmo branch-and-bound:

1. la sequenza di generazione ed esplorazione dei nodi dell'albero decisionale - che influisce sulla velocità di convergenza;
2. come e quando calcolare i bound relativi ai nodi.

(15) Depth-first

R: *Forward Step*: si genera un figlio dell'ultimo nodo P_k generato finché si ottiene una soluzione intera/$L_k \geq z/P_k$ non ha figli ammissibili non ancora esplorati; *Backtracking*: si risale al padre di P_k e, se $L_{padre} < z$ si genera il suo figlio successivo, altrimenti si risale al padre del padre. Si termina quando si cerca di risalire al di sopra di P_0.

Vantaggi: mantiene un piccolo numero di nodi aperti e produce una struttura semplice padre-figlio, produce rapidamente soluzioni ammissibili.

(16) Lowest-first

R: Definisce l'insieme dei nodi attivi e genera tutti i figli di quel nodo che ha lower bound più basso, eventualmente aggiornando z dopo la computazione dei nodi più facilmente risolvibili. Vantaggi: esplora sempre il nodo più "promettente", quindi ottiene una soluzione esplorando un ridotto numero di nodi; svantaggi: mantiene un grande numero di nodi aperti e produce una struttura complessa, non produce rapidamente soluzioni ammissibili.

(17) Depth-first rivisitata

R: Struttura depth-first, ma *forward Step*: si generano tutti i figli del nodo attuale, si calcolano i loro lower bound e si esplora il figlio con il minimo lower bound; *backtracking*: si riprende l'esplorazione dal nodo non esplorato con lower bound minimo fra i figli del nodo cui si è risaliti.

(18) Breadth-first

R: Genera tutti i figli del problema P_0 e tutti i figli di ogni figlio di P_0 non immediatamente risolubile. Strategia usata se interessano tutte le migliori soluzioni ammissibili del problema/tutte quelle con un certo valore massimo.

(19) Programazione lineare binaria LP01

R: Le variabili dell'LP01 possono assumere solo valori binari. Se la matrice A ha una sola riga si parla di problema Knapsack 0-1 (KP01).

(20) Knapsack 0-1

R:

$$max \sum_{j=1}^{n} p_j x_j$$

$$\sum_{j=1}^{n} w_j x_j \leq c$$

$$x_j \in \{0,1\} (j = 1, \ldots, n)$$

n elementi, ciascun elemento j ha un profitto p_j ed un peso w_j, la capacità massima dello *knapsack* è c. Si ordinano gli elementi per valori decrescenti di profitto per unità di peso, si adotta un albero decisionale binario che al livello k decide se porre o meno nella soluzione l'oggetto k ed una strategia depth-first, affinché l'insieme di nodi attivi contenga al più un nodo generato da $x_k = 1$ e uno o più generati da $x_k = 0$. L'upper bound si calcola mediante il rilassamento, che consente di inserire uno dopo l'altro in soluzione gli elementi migliori, frazionando il primo che non si può inserire per intero (*elemento critico*).

Proprietà l'upper bound generato da $x_j = 1$ è uguale all'upper bound del nodo padre.

(21) Branch-and-cut

R: Ad ogni nodo dell'albero decisionale con soluzione non intera si generano tagli, cercando di trovare una soluzione intera, o quantomeno di ottenere un bound migliore. Non si usano i tagli di Gomory perché dipendono dalle condizioni imposte ai progenitori, si adottano dei *global cuts* validi per tutto l'albero decisionale da memorizzare in una struttura dati apposita (*pool* dei vincoli).

8. Complessità

(1) Teoria della complessità

R: Individua la complessità di un problema, intesa come complessità del miglior algoritmo (= misura del tempo che esso richiede nel caso peggiore) in grado di risolvere il problema in questione.

(2) Istanza di un problema

R: Si definisce istanza di un problema P uno specifico caso numerico del problema stesso.

(3) Problema

R: Un problema è definito come l'insieme illimitato di tutte le sue possibili istanze.

(4) Complessità di un problema

R: La complessità di un problema P è la misura del tempo necessario per risolvere, nel caso peggiore, un'istanza di P. Il tempo deve essere indivuato in relazione alla dimensione dell'istanza.

(5) Dimensione di un'istanza

R: La dimensione di un'istanza è il numero di bit necessari per codificarne l'input/il numero di valori che costituiscono l'input.

(6) Complessità KP01 (caso branch and bound)

R: Il branch-and-bound risolve il problema KP01 in tempo $O(2^n)$ nel caso peggiore (andamento *esponenziale* nella dimensione dell'input, è un problema difficile per istanze di grandi dimensioni).

(7) Versione riconoscimento e versione ottimizzazione di un problema

R: La versione riconoscimento di un problema (RV) non propone come soluzione un valore, bensì un "sì" o un "no" relativo all'esistenza di una soluzione. La versione ottimizzazione (OV) cerca la soluzione numerica del problema. La teoria della complessità tratta i problemi in RV.

Proprietà: dal punto di vista del trovare o meno la soluzione in tempo polinomiale, le due versioni hanno la stessa diffcoltà. Questo perché un algoritmo che risolve OV risolve implicitamente anche RV; un algoritmo che risolve RV risolve anche OV mediante una serie di esecuzioni in ricerca binaria.

(8) Classe $\mathcal{P}$

R: La classe $\mathcal{P}$ contiene i problemi in RV per i quali si conosce un algoritmo polinomiale, ovvero sono risolubili in tempo polinomiale da una macchina di Turing deterministica.

(9) Classe $\mathcal{NP}$

R: La classe $\mathcal{NP}$ contiene i problemi in RV per i quali non si può escludere l'esistenza di un algoritmo polinomiale, ovvero sono risolubili in tempo polinomiale da una macchina di Turing non deterministica. Quest'ultima corrisponde ad un algoritmo ideale che prevede la possibilità di effettuare delle scelte, ed ogni volta effettua la scelta giusta. Un algoritmo simile risolverebbe in tempo polinomiale qualsiasi problema con albero dezisionale di altezza polinomiale. La classe $\mathcal{NP}$ comprende la classe $\mathcal{P}$.

Se un problema non appartiene alla classe $\mathcal{NP}$ non c'è speranza di trovare un algoritmo polinomiale che lo risolva.

(10) Trasformazione polinomiale

R: Un problema A di $\mathcal{NP}$ è trasformabile polinomialmente in un altro problema B di $\mathcal{NP}$ ($A \propto B$) se esiste un algoritmo polinomiale che, per ogni istanza di A, definisce un'istanza di B che ha soluzione "sì" se e solo se l'istanza di A ha soluzione "sì".

Proprietà Se $A \propto B$ e si conosce un algoritmo polinomiale per B, allora si ha anche un algoritmo polinomiale per A. Se $A \propto B$ e $B \propto C$, allora $A \propto C$.

(11) Problemi $\mathcal{NP}$-completi

R: Un problema A di $\mathcal{NP}$ si dice $\mathcal{NP}$-completo se, per ogni problema B di $\mathcal{NP}$, si ha $A \propto B$. Questo significa che se un problema è $\mathcal{NP}$-completo un eventuale algoritmo polinomiale per risolverlo potrebbe risolvere tutti i problemi di $\mathcal{NP}$.

Per stabilire che un problema è $\mathcal{NP}$-completo occorre:

1. dimostrare che A è in $\mathcal{NP}$ (facile applicando la definizione);

2. si può dimostrare che (1) per ogni problema $B \in \mathcal{NP}$ vale $B \propto A$ (molto difficile) o (2) esiste un problema $\mathcal{NP}$-completo C tale che $C \propto A$, che è più facile ammesso che si conoscono altri problemi $\mathcal{NP}$-completi.

Se si trovasse un algoritmo polinomiale per un solo problema $\mathcal{NP}$-completo risulterebbe $\mathcal{P} = \mathcal{NP}$ e si avrebbe un algoritmo polinomiale per tutti i problemi di $\mathcal{NP}$ (e quindi per tutti i problemi di ottimizzazione).

(12) Problemi $\mathcal{NP}$-difficili

R: Un problema per cui non si può dimostrare l'appartenenza a $\mathcal{NP}$ ma , per ogni problema di B di $\mathcal{NP}$, si ha $A \propto B$, è detto $\mathcal{NP}$-difficile o $\mathcal{NP}$-hard.

(13) Satisfiability problem SAT

R: SAT è un problema $\mathcal{NP}$-completo che afferma che date n variabili booleane $x_1, \ldots, x_n$ (con $x_i \in Vero, Falso$) ed una formula booleana esiste un assegnamento di valori x_1 per cui il risultato è Vero. SAT è risolubile mediante un albero decisionale binario di altezza n.

Qualunque problema $\mathcal{NP}$ può essere trasformato in tempo polinomiale in SAT.

(14) Lista dei problemi $\mathcal{NP}$-completi

R: Le RV di KP01, ILP e MILP (queste ultime due generalizzazioni di KP01) sono $\mathcal{NP}$-complete.

(15) Complessità della programmazione lineare

R: Il simplesso può richiedere un tempo esponenziale nel caso peggiore, ma "raramente" richiede più di $m \log n$ iterazioni. Sono state pensate delle varianti per LP quali l'algoritmo Khachiyan, basato sul metodo degli ellissoidi, o Karmarkar. Alcuni algoritmi hanno complessità polinomiale, ma nella pratica il simplesso è comunque più veloce nella maggior parte dei casi: la complessità logaritmica nel caso medio è preferibile a quella polinomiale nel caso peggiore.

(16) Programmazione dinamica

R: La programmazione dinamica riconduce la soluzione di un problema complesso alla soluzione di un numero elevato di suoi sottoproblemi, utilizzando per la soluzione di un sottoproblema le soluzioni ottenute per i sottoproblemi di dimensione inferiore.

(17) Problema subset-sum SSP

R: Dato un insieme di n valori interi, il Subset Sum Problem (SSP) è definito come la ricerca del sottoinsieme la cui somma sia più alta possibile ma pari o inferiore a c.

Per risolverlo con la programmazione dinamica, si definisce ricorsivamente $M_0 = \emptyset$, $M_j =$ *insieme dei valori di tutte le soluzioni ammissibili ottenibili prendendo o no ogni elemento*

tra i primi j. Ciascun insieme M_j contiene tutti i valori nell'insieme M_{j-1}, M_j contiene poi anche tutti i valori che si ottengono sommando il valore w_j (il j-esimo elemento) ai valori di M_{j-1}, ad eccezione di quelli che superano il valore massimo assegnato c.

(18) Problema knapsack 0-1 DP

R: Sia S un sottoinsieme di $\{1, \ldots, j\}$. Si definisce gli insiemi $M_j = \{coppie(p_k, w_k)\}$. Se $\sum_{k \in S'} w_k \leq \sum_{k \in S''} w_k$ e $\sum_{k \in S'} p_k \geq \sum_{k \in S''} p_k$ si dice che S' *domina* S'', nel senso che qualunque soluzione ottenibile aggiungendo elementi ad S'' è non migliore di quella ottenibile aggiungendo gli stessi elementi ad S' e per questo è più conveniente memorizzare solo la coppia prodotta da S'.

In totale la complessità di Knapsack DP è di O(nc), ovvero pseudo-polinomiale perché il numero di bit per codificare l'input di un'istanza di KP01 è $(2n+2)\lceil \log(c+1) \rceil$.

(19) Algoritmo pseudo-polinomiale

R: Dato un problema combinatorio A su valori interi, per ogni istanza I di A sia NUM(I) il più grande intero che compare in I e sia DIM(I) il numero di bit per codificare I. Un algoritmo per A è pseudo-polinomiale se risolve qualunque istanza I di A in un tempo limitato da una funzione polinomiale di DIM(I) e NUM(I). Questo significa che la complessità di questi algoritmi dipende non solo dal numero di valori dell'istanza, ma anche dalla loro magnitudo.

Esempio: KP01 DP ha NUM(I)=c e $DIM(I) = (2n+2)\lceil \log(c+1) \rceil$, quindi la complessità O(nc) è pseudo-polinomiale.

(20) Problema fortemente NP-completo

R: Non tutti i problemi NP-completi ammettono soluzioni polinomiali. Dato un problema combinatorio A, su valori interi, e una funzione polinomiale $p : N \mapsto N$, sia A_p la restrizione di A alle sole istanze I per le quali $NUM(I) \leq p(DIM(I))$. Il problema A si dice fortemente NP-completo se esiste un polinomio p per il quale A_p è NP-completo.

Per nessun problema fortemente NP-completo può esistere un algoritmo pseudo-polinomiale, a meno che P=NP.

Disclaimer: Questo documento può contenere errori e imprecisioni che potrebbero danneggiare sistemi informatici, terminare relazioni e rapporti di lavoro, liberare le vesciche dei gatti sulla moquette e causare un conflitto termonucleare globale. Procedere con cautela.