Modelli e Algoritmi di Ottimizzazione

2024-2025

Introduzione e problema del simplesso

Definiamo *algoritmo* la struttura matematica della soluzione di un determinato problema. Affrontiamo in questo corso gli algoritmi legati all'ottimizzazione. Un *problema di ottimizzazione* consiste nella ricerca del valore ideale di una *variabile obiettivo* ignota, in base a vincoli e relazioni tra *variabili decisionali* date, allo scopo di raggiungere un determinato fine. Per poter risolvere il problema, è necessario conoscere l'obiettivo dell'ottimizzazione. Può capitare in alcuni casi che possano coesistere diversi criteri secondo i quali cercare di ottimizzare. Ad esempio, in un'azienda, gli interessi dei dirigenti e gli interessi dei lavoratori possono entrare in contrasto. Si parla in questo caso di *programmazione multiobiettivo*. Nello scenario della programmazione multiobiettivo rientrano anche i modelli che affrontano l'incertezza futura. In questi problemi, infatti, si può decidere di massimizzare l'obiettivo principale, oppure di minimizzare il rischio dovuto all'incertezza futura. Sono inoltre distinguibili fin da subito due categorie di problemi di ottimizzazione, in base alla complessità matematica degli algoritmi risolutivi. Il primo gruppo, di più semplice risoluzione, coinvolge tutte e sole variabili continue, ed è detto *programmazione lineare* (LP). La seconda categoria, detta MIP (*mixed-integer programming*) contiene invece una o più variabili discrete, o addirittura binarie. La risoluzione è resa notevolmente più complessa dalla necessità di garantire un risultato intero per la variabile obiettivo.

Affrontiamo innanzitutto l'*algoritmo del simplesso* per la risoluzione di problemi di programamzione lineare. Esso fu sviluppato al termine della Seconda Guerra Mondiale da George Dantzig, addetto di logistica militare presso l'esercito americano, ai fini di generalizzare la soluzione ai problemi da lui affrontati nella propria attività lavorativa. Il primo passaggio consiste nella determinazione della variabile obiettivo. Si definiscono, in secondo luogo, le relazioni tra le variabili decisionali. Sono infine imposti vincoli sulle singole variabili decisionali, lasciando libera la variabile obiettivo. L'algoritmo non consente l'uso diretto delle disequazioni nonostante esse siano una parte fondamentale dei problemi di ottimizzazione. Per questa ragione, le disequazioni si riconducono ad equazioni mediante l'utilizzo di variabili scarto positive o nulle che bilanciano i due membri. Convenzionalmente esse sono positive o nulle e si aggiungono o sottraggono al termine sinistro. Le equazioni ottenute sono messe a sistema, ordinate per variabile. I coefficienti si possono raccogliere in forme matriciali o vettoriali. Le equazioni ottenute sono messe a sistema, ordinate per variabile. I coefficienti si possono raccogliere in forme matriciali o vettoriali. Questo tipo di rappresentazione permette di avere modelli compatti e facilmente riutilizzabili al variare dei valori delle variabili.

$$\begin{cases} 3x + 2y - 4z < 3 \ \rightarrow \ 3x + 2y + v_1 = 3 \\ 2x - y + z > 4 \ \rightarrow \ 2x - y + z - v_2 = 4 \\ x - 4y + 9z < 11 \ \rightarrow \ x - 4y + 9z + v_3 = 11 \end{cases}$$

$$\begin{bmatrix} 3 & 2 & -4 \\ 2 & -1 & 1 \\ 1 & -4 & 9 \end{bmatrix} \quad \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}$$

Figura 1: Trasformazione di una disequazione in equazione usando una variabile scarto

Definiamo ora il concetto di *problemi sparsi*. Essi sono contraddistinti da una matrice dei coefficienti *sparsa*, contenente cioè molti elementi nulli. Il sistema che ne deriva è detto *sottodeterminato*. Si definisce un *coefficiente di sparsità* che rappresenta la quota di coefficienti nulli rispetto al numero totale di elementi nella matrice. Tanto più è alto il coefficiente di sparsità, tanto più inefficiente risulta il calcolo della soluzione. Molto tempo di elaborazione, infatti, risulterà sprecato in operazioni dal risultato nullo. Al fine di mitigare questo problema sono definite strutture dati apposite, che permettono di compattare le matrici sparse sfruttando l'algebra dei puntatori. Così facendo, sono risparmiati spazio in memoria e tempo di calcolo.

$$\begin{pmatrix} 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 6 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 2: Esempio di matrice sparsa

Questo corso impiega il *software* GAMS per la risoluzione dei problemi di ottimizzazione. Nel caso del problema del simplesso, è in grado di trasformare automaticamente le disequazioni in equazioni, mostrando anche le variabili scarto generate a fine elaborazione. L'input ha una sintassi molto leggibile, che permette all'utente di definire variabili ed equazioni, e dare direttive, in linguaggio quasi naturale. Il programma è in grado di gestire input vettoriale e matriciale fino a dimensione 10. La sezione dell'*output* è divisa in cinque sottosezioni. Esse sono *compilation*, *equation listing*, *column listing*, *model statistics* e *solution report*. *Compilation* esprime le informazioni relative alla compilazione dello script fornito dall'utente. *Equation Listing* mostra le disequazioni generate a partire da variabili e relazioni specificate dall'utente nello script. Queste ultime sono ancora mostrate come disequazioni, prima della trasformazione in equazioni con la tecnica precedentemente descritta. In *Column Listing* sono mostrati, sotto forma di vettori colonna, i coefficienti del sistema di equazioni generato a partire dalle equazioni. *Model Statistics* esprime le statistiche relative al modello generato. *Solution Report* contiene infine la soluzione vera e propria al problema di ottimizzazione, divisa in due sezioni. La prima sezione, EQU, indica come si colloca la soluzione rispetto ai vincoli, ed elenca le variabili scarto generate nella trasformazione delle disequazioni in equazioni. La seconda sezione, VAR, mostra nel medesimo formato le variabili decisionali. Il software, nell'enumerare le variabili e le equazioni, distingue tra *blocks* (numero di vettori) e *single* (cardinalità totale).

Le fasi della produzione si possono schematizzare utilizzando un diagramma a nodi. Ogni nodo rappresenta un prodotto o un semilavorato. Le lavorazioni sono indicate inserendo direttamente le loro tabelle d'impiego. Successivamente si tracciano le linee di produzione che, partendo dalla fase iniziale e terminando nel prodotto finito, congiungono nodi e tabelle. Esse sono disegnate facendo riferimento alle informazioni di compatibilità tra prodotti e impianti. Una singola linea di processo può ramificarsi in più archi quando la produzione da essa rappresentata può passare indifferentemente attraverso diversi impianti. Tutti i rami devono ricongiungersi in un'unica linea entro l'arrivo al prodotto finito. Al termine della schematizzazione si impone la garanzia dei bilanci, tramite opportuni vincoli, tenendo conto di ramificazioni e ricongiungimenti delle linee di produzione. Si impone inoltre che l'impiego di tempo sia minore o uguale del tempo totale disponibile. Si costruisce poi una funzione di profitto tenendo conto dei vincoli funzionali. Il profitto si definisce come differenza tra i ricavi e la somma dei costi (sia delle materie prime, che della produzione). Il passaggio finale dell'analisi consiste nella massimizzazione della funzione di profitto.

GAMS permette di definire non solo insiemi di vincoli, ma anche sottoinsiemi di tali insiemi. È richiesto invece di definire variabili nulle per rappresentare le decisioni non attuabili. Si tratta delle combinazioni incompatibili di prodotti e impianti, che vanno a generare vincoli nulli. Questo perché è conveniente creare un vettore dei prodotti e un vettore degli impianti, e moltiplicarli per ottenere un'unica matrice decisionale. All'interno di essa le coppie non attuabili vengono semplicemente rappresentate da un valore nullo, andando a generare un certo grado di sparsità nella matrice. La forma matriciale permette di esprimere vincoli generici usando le sommatorie. Procedendo in questo modo, non risulta necessario fare considerazioni particolari per le coppie incompatibili, oltre all'annullamento delle variabili ad esse associate.

$$\begin{bmatrix} A & B & C \end{bmatrix} \begin{bmatrix} D \\ E \\ F \end{bmatrix} \begin{bmatrix} & A & B & C \\ D & A,D & B,D & C,D \\ E & A,E & B,E & C,E \\ F & 0 & B,F & C,F \end{bmatrix}$$

Figura 3: A, B, C: prodotti; D, E, F: linee; A e F sono incompatibili

Possiamo osservare i vincoli di bilancio come primissimo elemento della sezione EQU nei risultati di GAMS. Le risorse non completamente impiegate sono riconoscibili dal valore nullo del loro marginal. Dove esso non è nullo, si può invece verificare con un semplice procedimento se la struttura della soluzione cambierebbe in seguito all'aumento della risorsa scarsa. Si aggiunge un'unità e si riavvia l'ottimizzazione. Se l'aumento di produttività è pari al *marginal* del caso precedente, allora la struttura della soluzione non cambia. Se invece il valore è diverso, la struttura della soluzione è cambiata. Il marginal è detto anche prezzo ombra. Si può interpretare come la valorizzazione interna della risorsa, ovvero il suo valore ai fini della specifica produzione analizzata. Il concetto è diverso dalla valorizzazione esterna, corrispondente al valore di mercato della risorsa analizzata.

Un'altra distinzione derivabile dai risultati dell'analisi riguarda la divisione dei vincoli in attivi e inattivi. Sono definiti inattivi i vincoli che, se rimossi, non cambierebbero la soluzione. In altre parole, si tratta dei vincoli legati alle risorse

non completamente sfruttate. Il punto ottimale non si trova sul confine definito da tali vincoli. I vincoli attivi, al contrario, sono legati alle risorse limitanti. Il punto ottimale si trova esattamente sul confine da essi definito. Segue dunque che i vincoli di uguaglianza siano sempre attivi. L'attività dei vincoli di disuguaglianza, invece, dipende da che si tratti di disuguaglianza stretta o non stretta. Il concetto di vincoli attivi è fondamentale nella casistica, non affrontata dal corso, dei modelli non lineari. Esiste una classe di algoritmi di ottimizzazione non lineare chiamata active set methods, basata sull'identificazione iterativa di un sottoinsieme di vincoli che si ritiene saranno attivi nel punto ottimale.

Esiste una categoria di problemi di ottimizzazione, chiamata dei *problemi di livello*, molto studiata in tempi recenti. In questo tipo di problemi esistono più decisiori indipendenti. Questo si contrappone all'ottimizzazione classica, in cui un singolo decisore ha a disposizione informazioni sull'intero problema. Decisori differenti gestiscono aree diverse, ma le decisioni di ognuno influenzano indirettamente anche le aree di competenza altrui. I decisori possono essere allo stesso livello, distribuirsi gerarchicamente, oppure assumere una configurazione ibrida. Un esempio di problema di livello è la gestione del mercato libero dell'energia. Il gestore di rete sceglie giorno per giorno le offerte di produzione di varie aziende energetiche in base a una classifica di merito basata sul seguente criterio. I produttori dichiarano ogni giorno quanto saranno disposti a produrre il giorno successivo, e a che prezzo. Nonostante la grande quantità di variabili coinvolte, si tratta di un problema di programmazione lineare. Ogni giorno si osservano scostamenti, e risulta necessario introdurre piccole variazioni nei piani di produzione, sempre ottimizzandole matematicamente. Esistono *mercati di aggiustamento* dedicati al sopperimento di tali bisogni dinamici di energia. I modelli energetici devono essere evolutivi per poter raggiungere obiettivi futuri imposti a livello politico, come ad esempio il piano Energia e Clima dell'Unione Europea.

Ammissibilità

I vincoli lineari dei problemi LP sono rappresentabili come rette in un piano. La loro convergenza, quando esiste, va a generare una *regione di ammissibilità*. I vincoli di disuguaglianza limitano la soluzione ad un semipiano, mentre quelli di uguaglianza limitano la soluzione ai soli punti della retta stessa. L'aspetto della regione di ammissibilità è determinato dai vincoli di non negatività delle variabili. Può prendere la forma di un semipiano, di una retta, di un poligono, di un segmento o di un punto. Le intersezioni tra le frontiere dei vincoli sono chiamate *vertici*. I vertici della regione di ammissibilità sono *soluzioni di base*. Per determinare l'appartenenza di punti del piano alla regione di ammissibilità si valuta il segno della variabile scarto. Essa si annulla in corrispondenza delle soluzioni di base. È fondamentale individuare tali soluzioni perché esistono algoritmi in grado di determinare sequenze di soluzioni ammissibili a partire da una singola soluzione base nota.

$$x_1 + 2x_2 \leq 0$$

$$s = 10 - (x_1 + 2x_2)$$

$$x_1 + 2x_2 + s = 10 \quad \text{con} \quad s_1 \geq 0$$

$$\begin{cases} s > 0\text{: dentro la regione di ammissibilità} \\ s < 0\text{: fuori dalla regione di ammissibilità} \\ s = 0\text{: sulla frontiera} \end{cases}$$

Figura 4: Esempio di determinazione dei punti base

Possiamo dare una rappresentazione matriciale al sistema:

$$\begin{cases} Fw = b \\ w \geq 0 \end{cases}$$

dove:

- F è la matrice dei coefficienti
- w è il vettore delle incognite
- b è il vettore dei termini noti

Sappiamo che $F \in \mathbb{R}^{m,n+m}$ dove n è il numero di variabili decisionali (x) e m è il numero di vincoli. Il rango di F è $\text{rk}(F) = m$. Possiamo separare il il vettore w in due blocchi, uno di variabili decisionali e uno di variabili scarto:

$$w = \begin{pmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ s \end{pmatrix}$$

Anche la matrice dei coefficienti si può spezzare in una matrice di coefficienti delle variabili decisionali, e una matrice identità di coefficienti delle variabili scarto:

$$F = [AI]\,.$$

Riesprimendo in questo modo la precedente formula, abbiamo:

$$[AI] \begin{bmatrix} x \\ s \end{bmatrix} = Ax + s = b.$$

Supponiamo, continuando il corrente esempio, di voler calcolare uno dei punti base. Questo punto è dato dall'intersezione tra le due rette base dei vincoli, con l'annullamento delle rispettive variabili scarto.

$$\begin{array}{c} \begin{matrix} x_1 & x_2 \end{matrix} \\ \left[\begin{array}{cc|cc} 1 & 2 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{array}\right] \\ \begin{matrix} B & N \end{matrix} \end{array} \left(\begin{array}{c} x_1 \\ x_2 \\ \hline s_1 \\ s_2 \end{array}\right)$$

Annullando le due variabili scarto s_1 e s_2, la parte N della matrice diventa in pratica irrilevante e il calcolo si restringe a

$$\begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 10 \\ 4 \end{pmatrix}$$

calcolando solo il prodotto tra la base B e le variabili di base x_i da cui dobbiamo determinare w_B:

$$w_B = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}.$$

Invece w_N è

$$\begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Da questo possiamo dare un'ulteriore riscrittura a $Fw = B$:

$$[BN] \begin{bmatrix} w_B \\ w_N \end{bmatrix} = Bw_B + Nw_N = b$$

$$B^{-1}Bw_B + B^{-1}Nw_N = B^{-1}b \;\rightarrow\; \text{moltiplico tutto per l'inversa di } B$$

e quindi

$$\begin{cases} w_B = B^{-1}b - B^{-1}Nw_n \\ w_N = 0 \end{cases}$$

Quest'ultimo è l'insieme di tutte le soluzioni, anche quelle non ammissibili, del sistema sottodeterminato $Fw = b$ espresso rispetto alla base B.

Possiamo trovare altri punti base annullando altre variabili. Questo può essere ottenuto anche cambiando l'ordine delle colonne di F. Ad esempio, annullando x_2 e s_2, il sistema

$$\begin{bmatrix}1 & 2 & 1 & 0\\ 1 & 0 & 0 & 1\end{bmatrix}\begin{pmatrix}x_1\\ x_2\\ s_1\\ s_2\end{pmatrix} = \begin{pmatrix}10\\ 4\end{pmatrix}$$

diventa (per scambio, riportando a sinistra le due colonne di base):

$$\begin{bmatrix}1 & 1 & 2 & 0\\ 1 & 0 & 0 & 1\end{bmatrix}\begin{pmatrix}x_1\\ s_1\\ x_2\\ s_2\end{pmatrix} = \begin{pmatrix}10\\ 4\end{pmatrix} \quad \Rightarrow \quad \begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}\begin{pmatrix}x_1\\ s_1\end{pmatrix} = \begin{pmatrix}10\\ 4\end{pmatrix}$$

Trovo l'inversa di B:

$$\begin{bmatrix}a & b\\ c & d\end{bmatrix}^{-1} = \frac{1}{ad-bc}\begin{bmatrix}d & -b\\ -c & a\end{bmatrix} \Rightarrow B^{-1} = \begin{bmatrix}0 & 1\\ 1 & -1\end{bmatrix}$$

Per ottenere infine:

$$\begin{matrix}B^{-1}\\ \begin{bmatrix}0 & 1\\ 1 & -1\end{bmatrix}\end{matrix}\begin{matrix}B & & N & & b\\ \begin{bmatrix}1 & 1 & 2 & 0 & | & 10\\ 1 & 0 & 0 & 1 & | & 4\end{bmatrix}\end{matrix} = \begin{matrix}\begin{bmatrix}1 & 0 & | & 0 & 1 & | & 4\\ 0 & 1 & | & 2 & -1 & | & 5\end{bmatrix}\\ B^{-1}B = I \quad B^{-1}N \quad B^{-1}B\end{matrix}$$

$$\text{quindi} \quad \begin{pmatrix}x_1\\ s_1\end{pmatrix} = \begin{pmatrix}6\\ 4\end{pmatrix} - \begin{bmatrix}0 & 1\\ 2 & -1\end{bmatrix}\begin{pmatrix}x_2\\ s_2\end{pmatrix}$$

$$w_B = B^{-1}b - B^{-1}N\ w_N$$

Terminiamo dunque la risoluzione numerica:

$$\begin{cases}x_1 = 4 - s_2\\ s_1 = 6 - 2x_2 + s_2\end{cases} \qquad \text{con } x_2 \text{ e } s_2 \text{ arbitrari in } \mathbb{R}^2.$$

La soluzione di base è

$$x_1 = 4 \quad x_2 = 0$$

$$s_1 = 6 \quad s_2 = 0.$$

Se trovo degli s negativi, non sono in zona ammissibile. Ricordiamo inoltre che la base BB per definizione deve essere singolare, cioè le sue colonne devono essere linearmente indipendenti.

La risoluzione dei problemi LP si divide dunque in due fasi, una chiamata *indagine di ammissibilità*, e l'altra coincidente con l'ottimizzazione vera e propria. L'indagine di ammissibilità consiste nel determinare se la regione di ammissibilità esista, trovando almeno un punto che ne faccia parte. I problemi i cui vincoli siano esclusivamente di minore o uguale relativamente a termini noti non necessitano di una vera indagine di ammissibilità, perché si può verificare immediatamente che l'origine del piano sia una soluzione ammissibile. Questo non vale invece per problemi nei quali coesistono vincoli di uguaglianza e disuguaglianza. In tal caso è necessario costruire un problema ausiliario, immediatamente ammissibile, la cui soluzione ottimale corrisponda ad un punto base della regione ammissibile relativa al problema originario. Il problema è irrisolvibile se la regione di ammissibilità è delimitata da un vincolo limitato da una variabile che necessiti di essere massimizzata nella funzione obiettivo. Esiste infine un caso limite di ammissibilità, in cui esiste una sola soluzione ammissibile puntuale. Tale soluzione è chiamata punto degenere e causa problematiche, seppur risolvibili, nella soluzione algebrica del problema.

Teorema fondamentale della Programmazione Lineare

Il teorema fondamentale della Programmazione Lineare si divide in due sottoteoremi, che chiamiamo FTLP-1 e FTLP-2.

> **FTLP-1**: se la regione ammissibile del problema di PL non è vuota, almeno una delle soluzioni ammissibili è una soluzione di base.

In altre parole: - se esiste una soluzione ammissibile, allora esiste una soluzione ammissibile di base:

$$\exists\ f.s. \Rightarrow \exists\ b.f.s.$$

- se non esiste nessuna soluzione ammissibile di base, allora non esiste nessuna soluzione ammissibile:

$$\nexists\ b.f.s. \Rightarrow \nexists\ f.s.$$

Questo teorema si dimostra per costruzione. Per ipotesi la regione non è vuota, quindi esiste una soluzione ammissibile. Partendo da essa, è possibile generare una sequenza di altre soluzioni ammissibili per ottenere, in un numero finito di passaggi, una soluzione di base. Basandosi su FLTP-1 è possibile sviluppare un algoritmo per studiare l'ammissibilità dei problemi di programmazione lineare. Questo algoritmo esamina solo le soluzioni di base ammissibili di un problema ausiliario opportunamente definito. L'algoritmo trova, alternativamente, una soluzione ammissibile di base per il problema originale, oppure determina che la regione di ammissibilità sia vuota.

> FLTP-2: se la funzione obiettivo è limitata sulla regione ammissibile (inferiormente per problemi di minimo, superiormente per i problemi di massimo) allora almeno una delle soluzioni ammissibili ottime è una soluzione ammissibile di base.

Anche questo sottoteorema si dimostra per costruzione. Esiste una soluzione ammissibile ottima perché la funzione obiettivo è limitata per ipotesi. Partendo da questa soluzione ammissibile, possiamo generare una sequenza di altre soluzioni ammissibili ottime per ottenere, in un numero finito di passaggi, una soluzione di base ottima.

Ripasso

Siano dati una matrice $F = [AI]$ di dimensioni $m \times (n+m)$ e rango $\text{rk}(F) = m$, e un vettore $w = \begin{pmatrix} w \\ s \end{pmatrix}$ tali che $Fw = b$, $\exists\ B\ n \times m$ non singolari estratte da F. Scelta B in F, riordino le colonne di F e le componenti di w in modo che le colonne di B e le componenti di w associate a B siano le prime m.

$$F = [BN] \quad w = \begin{pmatrix} w_B \\ w_n \end{pmatrix}$$

$$B^{-1}[BN] \begin{pmatrix} w_B \\ w_N \end{pmatrix} = B^{-1}b$$

$$w_B + B^{-1}Nw_N = B^{-1}b$$

Quest'ultimo risultato è una rappresentazione di tutte e sole le soluzioni di $Fw = b$ (insieme soluzione). Esso è detto *forma canonica del sistema rispetto alla base* B. Il metodo del simplesso è basato su questa forma. Per ottenere l'intero insieme si sceglie arbitrariamente $w_N \in \mathbb{R}^n$. Esprimendo in un modo alternativo l'equazione notiamo che ciascuna variabile di base viene espressa in termini delle variabili non di base:

$$w_B = B^{-1}(b - Nw_N)$$

Per ottenere la soluzione associata a B si impone $w_N = 0$ (origine di $\mathbb{R}^n$). La soluzione di base è dunque, di fatto,

$$\begin{cases} w_B = B^{-1}B \\ w_N = 0 \end{cases}$$

Annullare le w_N significa, geometricamente, porsi su n frontiere di vincoli.

L'algoritmo del simplesso

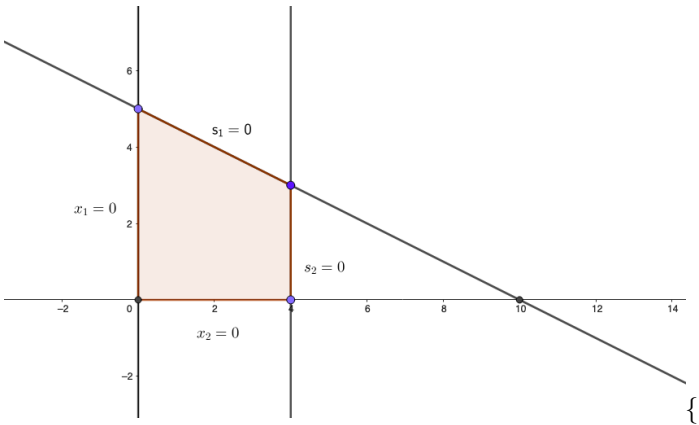
L'algoritmo procede iterativamente ripetendo questi quattro passi: 1. test di ottimalità 2. determinazione della direzione di ricerca 3. determinazione della lunghezza del passo 4. determinazione della forma canonica rispetto alla nuova base adiacente alla precedente

Il nome del primo passaggio è autoesplicativo: si verifica se la soluzione a disposizione, ovvero la prima soluzione di base da cui l'algoritmo inizia, oppure una soluzione base trovata iterativamente, è ottimale. Si procede al secondo

passo soltanto se la soluzione attuale non è ottimale. Nel secondo passo si cerca la direzione lungo la quale il valore della variabile obiettivo tende a diminuire (o aumentare, nel caso di un problema di massimizzazione). Il terzo passo consiste nel determinare di quanto sia necessario muoversi in tale direzione per raggiungere la successiva soluzione di base. Si trova infine la nuova forma canonica legata al punto base trovato. Le basi canoniche trovate sono sempre adiacenti, perché da un passo all'altro si cambia un solo vincolo tra quelli considerati. L'algoritmo si muove lungo la frontiera. Ogni qual volta si determina un nuovo punto base, si ripete il test di ottimalità. Se viene passato, tale punto è la soluzione ottima. Nei problemi privi di soluzione ottimale, in un numero finito di iterazioni si arriva a una lunghezza di passo infinita durante il terzo passaggio. Esistono metodi alternativi al simplesso chiamati metodi a punti interni. Per definizione, la soluzione ottima si trova sulla frontiera. Questi metodi, muovendosi all'interno della regione ammissibile, possono avvicinarsi più rapidamente all'ottimo prendendo scorciatoie. Quando però il punto di ottimo è ritenuto abbastanza vicino, la regola di movimento deve cambiare per potersi avvicinare alla frontiera, e la determinazione del momento di cambio strategia è non banale. Il simplesso risulta tutt'oggi l'algoritmo più efficiente nella maggioranza dei casi.

Esercizio:

$$\begin{aligned} &\max \omega = 2x_1 + 3x_2 \\ \text{s.t.}\quad &x_1 + 2x_2 \leq 10 \\ &x_1 \leq 4 \\ &x_1 \geq 0, \quad x_2 \geq 0 \end{aligned} \qquad \left| \qquad \begin{aligned} &\min \varphi = -2x_1 - 3x_2 \\ &x_1 + 2x_2 + s_1 = 10 \\ &x_1 + s_2 = 4 \\ &x_1 \geq 0, \quad x_2 \geq 0, \quad s_1 \geq 0, \quad s_2 \geq 0 \end{aligned} \right.$$



{ width=30% }

Il sistema è sottodeterminato. Poniamo l'obiettivo in forma di minimo. La scelta è a discrezione di chi scrive l'algoritmo. È infatti possibile trasformare un problema di massimo in minimo rendendo negativi entrambi i membri dell'equazione. In questo caso applichiamo il cambio di variabile $\varphi = -\omega$. Notiamo che il sistema è già in forma canonica rispetto alla base $B = [e_1\ e_2] = I$, perché s_1, s_2 e φ sono già espresse solo in funzione di variabili x_i. Possiamo dunque determinare immediatamente B, N, e le variabili di base e non di base w_B e w_N.

In questo problema la base è formata dalle colonne di s_1 e s_2 (ovvero è una matrice identità):

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \quad w_B = \begin{pmatrix} s_1 \\ s_2 \end{pmatrix}$$

$$N = \begin{bmatrix} 1 & 2 \\ 1 & 0 \end{bmatrix} \quad w_N = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Ciascuna delle variabili (s_1, s_2, φ) è espressa in funzione delle sole variabili x_1 e x_2.

Ecco il tableau iniziale:

$$\begin{array}{ccccccc} x_1 & x_2 & s_1 & s_2 & \varphi & & \\ \left[\begin{array}{ccccc|c} 1 & 2 & 1 & 0 & 0 & 10 \\ 1 & 0 & 0 & 1 & 0 & 4 \\ \hline 2 & 3 & 0 & 0 & 1 & 0 \end{array}\right] \end{array}$$

Per ottenere il secondo punto, ovvero il punto A, annulliamo x_2 e s_2 lasciando x_1 e s_1 come variabili di base. Ricordiamo che l'algoritmo cambia una sola variabile di base per volta per passare da un punto all'altro.

$$w_B = \begin{pmatrix} x_1 \\ s_1 \end{pmatrix} \qquad w_N = \begin{pmatrix} x_2 \\ s_2 \end{pmatrix}$$

Proviamo ora a muoverci dall'origine verso l'alto.

$$\begin{array}{c} x_1 \quad x_2 \quad s_1 \quad s_2 \quad \varphi \\ \left[\begin{array}{ccccc|c} 1 & 2 & 1 & 0 & 0 & 10 \\ 1 & 0 & 0 & 1 & 0 & 4 \\ \hline 2 & 3 & 0 & 0 & 1 & 0 \end{array}\right] \end{array}$$

Test di ottimalità: $2x_1 + 3x_2 + \varphi = 0$: è possibile? - $x_1 \geq 0$ - $x_2 \geq 0$

Di principio sì. Dato che le due variabili possono assumere valori positivi allora phi può diventare negativo.

- $x_1 > 0$, $x_2 = 0$: $\varphi = -2x_1$
- $x_2 > 0$, $x_1 = 0$: $\varphi = -3x_2$

Estraiamo s_i dal tableau:

$$s_1 = 10 - 2x_2 \geq 0, \quad 0 \leq x_2 \leq 5$$

$$s_2 = 4 \geq 0$$

Scegliamo di attribuire a x_2 il massimo valore possibile, ovvero 5. Questo valore corrisponde alla lunghezza del passo. Muovendoci di 5 verso l'alto dall'origine possiamo trovare il punto base situato sopra di essa nel disegno. Le nostre variabili di base per questo step sono x_2 e s_2.

Volendo procedere in modo più rigoroso, l'algoritmo calcola il passo così: - dato che abbiamo stabilito di voler portare in base x_2, si divide il termine noto della riga (10) per il valore del coefficiente di x_2 (2): $10/2 = 5$. È necessario calcolare questo rapporto per tutte le righe, e scegliere il minimo tra quelli ottenuti. Questo calcolo è effettuato solo sui coefficienti strettamente positivi - dobbiamo arrivare ad una nuova forma canonica per il nuovo punto in cui ci spostiamo. Ogni variabile di base deve continuare ad essere espressa in funzione delle variabili non di base.

La nuova B deve essere formata dalle colonne di x_2 e s_2, quindi:

$$B_2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \qquad w_B \begin{pmatrix} x_2 \\ s_2 \end{pmatrix}$$

per arrivare a

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x_2 \\ s_2 \end{pmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ s_1 \end{pmatrix} = \begin{pmatrix} 10 \\ 4 \end{pmatrix}$$

$$B \qquad w_B \qquad + \qquad Nw_N \qquad = \qquad b$$

La forma canonica con le colonne riordinate prende la seguente forma.

$$w_B + \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 1 & 0 \end{bmatrix} w_N = \begin{pmatrix} 5 \\ 4 \end{pmatrix}$$

Non volendo modificare l'allineamento delle variabili, per procedere in maniera puramente algoritmica, il tableau assume questa disposizione:

$$\begin{array}{c} x_1 \quad x_2 \quad s_1 \quad s_2 \quad \varphi \\ \left[\begin{array}{ccccc|c} \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 5 \\ 1 & 0 & 0 & 1 & 0 & 4 \\ \hline \square & 0 & \square & 0 & 1 & \square \end{array}\right] \end{array}$$

L'1 del versore da costruire si trova all'intersezione tra la riga e la colonna dai quali coefficienti è stato generato il rapporto per determinare la lunghezza del passo.

Per determinare i nuovi valori delle righe, procediamo per riduzione gaussiana, in modo da ottenere i valori adeguati alla formazione di una matrice di base che sia identità:

$$\bar{R}_1 = \frac{1}{2} R_1 = \frac{1}{2} \; 1 \; \frac{1}{2} \; 0 \; 0 \; 5$$

$$\bar{R}_2 = R_2$$

$$\bar{R}_0 = R_0 - 3\bar{R}_1 = \begin{array}{ccccccc} 2 & 5 & 0 & 0 & 1 & 0 & - \\ \frac{3}{2} & 3 & \frac{3}{2} & 0 & 0 & 15 & = \\ \hline \frac{1}{2} & 0 & -\frac{3}{2} & 0 & 1 & -15 & \end{array}$$

La nuova struttura è:

$$\begin{array}{ccccc|c} x_1 & x_2 & s_1 & s_2 & \varphi & \\ \left[\begin{array}{ccccc|c} \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 5 \\ 1 & 0 & 0 & 1 & 0 & 4 \\ \hline \frac{1}{2} & 0 & -\frac{3}{2} & 0 & 1 & -15 \end{array}\right] \end{array}$$

Ripartiamo dal primo passo: determinazione dell'ottimalità.

$$\varphi = -15 - \frac{1}{2}x_2 + \frac{3}{\cancel{2}}\cancel{s_1}$$

Non si deve attribuire a s_1 un valore positivo. Non si può infatti ottimizzare scegliendo variabili il cui coefficiente in ultima riga sia negativo. Da tale considerazione è escluso l'1, che è coefficiente della variabile obiettivo. In questo caso la scelta obbligata è x_1. A questo punto x_1 sostituirà la s_2.

$$\begin{array}{cc} x_2 = 5 - \frac{1}{2}x_1 \geq 0 & x_1 \leq 10 \\ s_2 = 4 - x_1 \geq 0 & x_1 \leq 4 \end{array}$$

$$\Rightarrow x_1 = 4$$

Dobbiamo ora trasformare le righe per ottenere che la colonna nuova di x_1 diventi come la vecchia s_2 (0 1 0):

$$\bar{R}_2 = R_2$$

$$\bar{R}_1 = R_1 - \tfrac{1}{2}R_2 = \begin{array}{ccccccc} \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 5 & + \\ -\frac{1}{2} & 0 & 0 & -\frac{1}{2} & 0 & -2 & \\ \hline 0 & 1 & \frac{1}{2} & -\frac{1}{2} & 0 & 3 & \end{array}$$

$$\bar{R}_0 = R_0 - \tfrac{1}{2}R_2 = \begin{array}{cccccc} \frac{1}{2} & 0 & -\frac{3}{2} & 0 & 1 & -15 \\ -\frac{1}{2} & 0 & 0 & -\frac{1}{2} & 0 & -2 \\ \hline 0 & 0 & -\frac{2}{3} & -\frac{1}{2} & 1 & -17 \end{array}$$

La riga finale è chiamata dei coefficienti di costo relativo. Qualora essi siano tutti negativi o nulli, allora la soluzione associata è ottima. Se tutti gli elementi sono negativi, l'ottimo è unico; se sono mescolati termini nulli e negativi, esistono soluzioni ottimali alternative. Sono deducibili altre informazioni dalle righe rimanenti del tableau. In corrispondenza della soluzione ottima, dalla loro analisi è possibile determinare in che modo essa possa modificarsi al variare dei dati in ingresso

Esistenza di soluzioni ottime alternative

Il *tableau* è ricco di informazioni di vario genere. È possibile dedurre, tra le altre cose, la presenza di soluzioni ottime alternative a partire dal tableau corrispondente ad una data soluzione ottima. Osserviamo il seguente esempio:

x_1	x_2	s_1	s_2	φ		
$\frac{1}{2}$	1	$\frac{1}{2}$	0	0	|	5
1	0	0	1	0	|	4
0	0	−2	0	1	|	−20

Le variabili di base sono x_2 e s_1, come possiamo dedurre dal fatto che le loro rispettive colonne siano versori. Osservando l'ultima riga notiamo che tutti i coefficienti sono negativi o nulli. Questo è segno di ottimalità. Da questa considerazione è escluso, come sempre, il coefficiente di φ che è fisso a 1. L'ultima riga, però, rappresenta la funzione obiettivo espressa in funzione delle variabili non di base, x_1 e s_1:

$$\varphi = -20 + 0x_1 + 2s_1$$

Il coefficiente di x_1 è nullo. Questo significa che il valore di x_1 può variare liberamente senza alterare la funzione obiettivo, ovvero senza inficiare l'ottimalità della soluzione. Questo vale, ovviamente, condizionatamente al rispetto dei vincoli: ad esempio x_1, pur variando, deve rimanere positivo, ed è necessario considerare anche i vincoli sulle variabili di base. Le variazioni di x_1 rappresentano, geometricamente, lo scorrimento lungo la retta rappresentata dal vincolo s_1.

Ne scaturiscono dunque infinite soluzioni ottime. Le possiamo esprimere come combinazione lineare convessa delle due soluzioni di base poste lungo s_1, ovvero i punti B e C. Troviamo l'altra soluzione mandando in base x_1:

x_1	x_2	s_1	s_2	φ		
0	1	$\frac{1}{2}$	$-\frac{1}{2}$	0	|	3
1	0	0	1	0	|	4
0	0	-2	0	1	|	-20

Le soluzioni hanno forma:

$$\begin{bmatrix} x_1 \\ x_2 \\ s_1 \\ s_2 \end{bmatrix} = \alpha \begin{bmatrix} 0 \\ 5 \\ 0 \\ 4 \end{bmatrix} + (1-\alpha) \begin{bmatrix} 4 \\ 3 \\ 0 \\ 0 \end{bmatrix}, \quad 0 \leq \alpha \leq 1$$

Inammissibilità dell'origine

L'algoritmo del simplesso, per funzionare, necessita di partire da una soluzione di base ammissibile. Nei problemi contenenti solo vincoli di minoranza o di minore-uguaglianza, è immediato individuare una soluzione banale nell'origine dello spazio vettoriale. In presenza di vincoli di maggioranza, maggiore-uguaglianza o uguaglianza stretta, questa soluzione smette di essere ammissibile. Consideriamo ad esempio il seguente sistema:

$$\min \varphi = 0.4x_1 + 0.5x_2$$

$$\begin{array}{lrcrcl} s.t. & 0.3x_1 & + & 0.1x_2 & \leq & 2.7 \\ & 0.6x_1 & + & 0.4x_2 & \geq & 6 \\ & 0.5x_1 & + & 0.5x_2 & = & 6 \end{array}$$

La seconda e la terza equazione non ammettono $\underline{0}$ come soluzione.

Esiste un metodo per trovare una soluzione di base ammissibile quando l'origine non lo è. Consiste nella risoluzione di un problema ausiliario, che introduca delle variabili scarto aggiuntive. Nello specifico:

1. nelle disequazioni di maggiore o uguale e di maggiore, aggiungiamo una seconda variabile scarto, additiva e ausiliaria:
$$-s_i, \quad s_i \geq 0 \rightarrow -s_i + t_i, \quad s_i, t_i \geq 0$$
2. nelle equazioni aggiungiamo una variabile scarto ausiliaria aggiuntiva:
$$t_i, \quad t_i \geq 0$$
3. costruiamo una funzione obiettivo ausiliaria, da minimizzare, corrispondente alla somma di tutte le variabili scarto ausiliarie:
$$\min \Psi = \sum_i t_i$$
4. risolviamo il problema con il metodo del simplesso.

Costruiamo il sistema di equazioni secondo le regole esposte, e mettiamolo a tableau:

$$\min \Psi = t_2 + t_3$$

$$\begin{array}{lrcrcrcl} s.t. & 0.3x_1 & + & 0.1x_2 & + & s_1 & = & 2.7 \\ & 0.6x_1 & + & 0.4x_2 & - & s_2 + t_2 & = & 6 \\ & 0.5x_1 & + & 0.5x_2 & + & t_3 & = & 6 \end{array}$$

x_1	x_2	s_1	s_2	t_2	t_3	Ψ		
0.3	0.1	1	0	0	0	0	|	2.7
0.6	0.4	0	-1	1	0	0	|	6
0.5	0.5	0	0	0	1	0	|	6
0	0	0	0	1	1	1	|	0

Se la funzione obiettivo della soluzione ottima del problema ausiliario ha valore positivo, signfica che non esistono valori in grado di annullare le variabili scarto ausiliarie, e dunque non esiste nessun punto nella regione di ammissibilità del problema originario. Se invece la funzione obiettivo ha valore nullo, allora la soluzione ottima ausiliaria corrisponde ad una soluzione ammissibile di base del problema originario. In tal caso possiamo continuare dallo stesso *tableau*, nella sua iterazione finale, eliminando le colonne relative alle variabili scarto ausiliarie. Sostituiamo la funzione obiettivo

ausiliaria con la funzione obiettivo originale. Esprimiamo infine la funzione obiettivo in funzione delle variabili di base contenute nella base ottima del problema ausiliario. Se l'ultima riga del *tableau* così ottenuto indica non ottimalità, possiamo procedere applicando il metodo del simplesso secondo le modalità usuali.

Tornando all'esempio, il tableau ottenuto a fine risoluzione del simplesso ausiliario ha la seguente forma:

x_1	x_2	s_1	s_2	t_2	t_3	Ψ	
1	0	5	0	0	-1	0	7.5
0	1	-5	0	0	3	0	4.5
0	0	1	1	-1	0.6	0	0.3
0	0	0	0	-1	-1	1	0

A questo punto possiamo eliminare le colonne t_2 e t_3, sostituire Ψ con φ, e riscrivere l'equazione obiettivo originale. Dobbiamo però esprimerla in funzione delle variabili non di base, applicando le consuete operazioni di riga all'equazione obiettivo:

$$\max \omega : \ 0.4x_1 + 0.5x_2 + 0 \rightarrow \min \varphi : \ -0.4x_1 + -0.5x_2 + 0$$

$$\rightarrow \min \varphi : \ -0.5s_1 + 5.25 = 0$$

Il tableau iniziale per il simplesso del problema originale diventa dunque:

x_1	x_2	s_1	s_2	φ	
1	0	5	0	0	7.50
0	1	-5	0	0	4.5
0	0	1	1	0	0.3
0	0	-0.5	0	1	5.25

che è già una soluzione ottima.

Problemi complementari

Confrontiamo i problemi del compratore e del produttore, tra loro complementari.

Compratore	Produttore
- $m = 2$ variabili- $n = 2$ vincoli	- $n = 2$ variabili- $m = 2$ vincoli

Grafico del compratore:

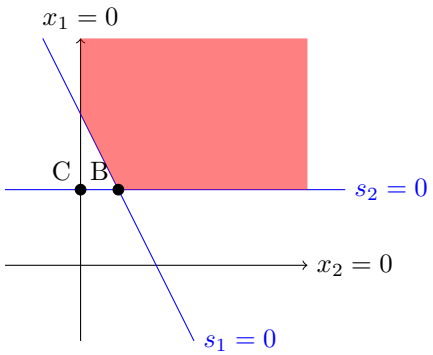
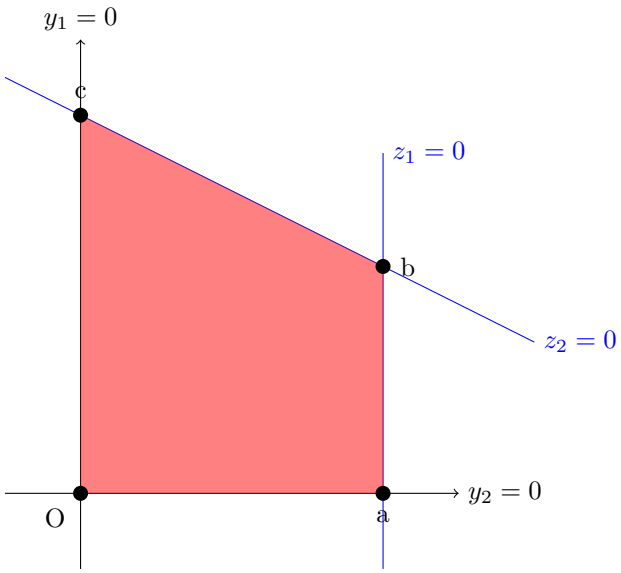


Grafico del venditore:



$$\begin{array}{ll} \min \varphi = 4x_1 + 5x_2 & \max \omega = y_1 + y_2 \\ \\ x_1 + \frac{1}{2}x_2 \geq 1 & y_1 \leq 4 \\ x_2 \geq 1 & \frac{1}{2}y_1 + y_2 \leq 5 \\ x_1, x_2 \geq 0 & y_1, y_2 \geq 0 \end{array}$$

Portiamole in forma di minimo e trasformiamo le disequazioni in equazioni, introducendo variabili scarto s_i per il problema del compratore e z_i per il problema del produttore:

$$\begin{array}{ll} \min \varphi = 4x_1 + 5x_2 & \min \varphi_D = -y_1 - y_2 \\ \\ x_1 + \frac{1}{2}x_2 - s_1 = 1 & y_1 + z_1 = 4 \\ x_2 - s_2 = 1 & \frac{1}{2}y_1 + y_2 + z_2 = 5 \\ x_1, x_2, s_1, s_2 \geq 0 & y_1, y_2, z_1, z_2, \geq 0 \end{array}$$

Il problema del compratore richiederebbe di impostare e risolvere un problema ausiliario. Questo perché la soluzione banale $\underline{x} = \underline{0}$ non è ammissibile. Possiamo agire diversamente facendo una piccola considerazione iniziale. Osserviamo innanzitutto il tableau:

$$\left[\begin{array}{ccccc|c} x_1 & x_2 & s_1 & s_2 & \varphi & \\ \hline 1 & \frac{1}{2} & -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ \hline -4 & -5 & 0 & 0 & 0 & 0 \end{array}\right]$$

Possiamo portarlo in forma canonica prendendo s_1 e s_2 come variabili di base. Le loro colonne sarebbero già versori se venissero invertite di segno. Effettuiamo quindi operazioni di inversione di segno sulle prime due righe del *tableau*:

$$\left[\begin{array}{ccccc|c} x_1 & x_2 & s_1 & s_2 & \varphi & \\ \hline -1 & -\frac{1}{2} & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 & 0 & -1 \\ \hline -4 & -5 & 0 & 0 & 1 & 0 \end{array}\right]$$

La soluzione di base corrispondente è ottima ma non ammissibile. Non possiamo dunque utilizzarla come punto di partenza per iterare con il metodo del simplesso. Consideriamo invece il *tableau* del problema del produttore:

$$\left[\begin{array}{ccccc|c} y_1 & y_2 & z_1 & z_2 & \varphi_D & \\ \hline 1 & 0 & 1 & 0 & 0 & 4 \\ \frac{1}{2} & 1 & 0 & 1 & 0 & 5 \\ \hline 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right]$$

La soluzione è ammissibile ma non ottima. Le sue variabili di base sono z_1 e z_2.

Cerchiamo di trovare una corrispondenza tra i due problemi. Notiamo innanzitutto che i valori delle variabili non di base nel problema del compratore sono l'opposto dei valori delle variabili di base nel problema del produttore, e che il valori delle variabili di base del problema del compratore sono l'opposto dei valori delle variabili non di base del problema del produttore:

- $x_1 = -4 \ \leftrightarrow z_1 = 4$

- $x_2 = -5 \leftrightarrow z_2 = 5$
- $s_1 = -1 \leftrightarrow y_1 = 1$
- $s_2 = -1 \leftrightarrow y_2 = 1$

Per mettere maggiormente in evidenza la corrispondenza, costruiamo dei *tableau ridotti*, che contengano soltanto la forma canonica delle variabili non di base dei rispettivi problemi.

Per il compratore:

	x_1	x_2	
s_1	-1	$-\frac{1}{2}$	-1
s_2	0	-1	-1
φ	-4	-5	

Per il produttore:

	y_1	y_2	
z_1	1	0	4
z_2	$\frac{1}{2}$	1	5
φ_D	1	1	

Notiamo che i due tableau ridotti sono uno l'antitrasposto dell'altro:

$$B_P = -B_D^T$$

In questo modo è possibile passare da una rappresentazione all'altra del problema senza perdita di informazioni. Per teorema, questa regola vale per tutte le coppie di variabili duali, non solo nell'origine del problema come mostrato ora. Si parla di *basi complementari*.

Per mantenere la corrispondenza spostiamo le colonne del tableau del produttore in modo che si mantenga la mappatura posizionale con il problema del compratore:

$$\left[\begin{array}{ccccc|c} z_1 & z_2 & y_1 & y_2 & \varphi_D & \\ \hline 1 & 0 & 1 & 0 & 0 & 4 \\ 0 & 1 & \frac{1}{2} & 1 & 0 & 5 \\ \hline 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right]$$

$$\left[\begin{array}{ccccc|c} x_1 & x_2 & s_1 & s_2 & \varphi & \\ \hline -1 & -\frac{1}{2} & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 1 & 0 & -1 \\ \hline -4 & -5 & 0 & 0 & 1 & 0 \end{array}\right]$$

In questo caso entrambi i problemi hanno due variabili di base e due variabili non di base. Questo permette di avere una corrispondenza dimensionale tra i due *tableau* ridotti. La proprietà non è sempre verificata. Se il numero di variabili di base è diverso dal numero di variabili non di base, allora i due *tableau* avranno dimensioni diverse.

Ri-estendendo il *tableau* ridotto del problema duale, si può notare il passaggio dalla soluzione ottima ma non ammissibile del problema primale alla soluzione ammissibile ma non ottima del problema duale. Da quest'ultima potremmo procedere mediante il metodo del simplesso ordinario, rimanendo sul *tableau* duale. Cerchiamo invece di costruire un metodo del *simplesso duale*, applicabile direttamente al *tableau* primale, antitrasponendo le operazioni che il simplesso ordinario effettuerebbe sul *tableau* duale. Ad esempio, come prima mossa, nel problema duale porteremmo in base y_1 al posto di z_1. Sul primale, l'entrata in base di y_1 corrisponde all'uscita di base di s_1, e dunque, al posto di quest'ultima, faremmo entrare in base x_1 che è complementare di z_1 uscita di base nel duale. Procediamo dunque trasformando la colonna di x_1 come un vettore:

$$\bar{R}_1 = -R_1$$
$$\bar{R}_2 = R_2$$
$$\bar{R}_0 = R_0 + r\bar{R}_1$$

$$\left[\begin{array}{ccccc|c} x_1 & x_2 & s_1 & s_2 & \varphi & \\ \hline 1 & \frac{1}{2} & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 & 0 & -1 \\ \hline 0 & -3 & -4 & 0 & 1 & 4 \end{array}\right]$$

Anche questa nuova soluzione di base è ammissibile ma non ottima, come la precedente. Usiamo i *tableau* ridotti per ricostruire il passo appena eseguito anche per il problema duale.

Per il compratore:

	x_2	s_1	
s_1	$\frac{1}{2}$	-1	1
s_2	-1	0	-1
φ	-3	-4	4

Antitrasponiamo per ottenere il produttore:

	z_1	y_2	
z_2	$-\frac{1}{2}$	1	3
y_1	1	0	4
φ_D	-1	1	4

Ricostruiamo il *tableau* duale completo:

z_1	z_2	y_1	y_2	φ_D		
$-\frac{1}{2}$	1	0	1	0	|	3
1	0	1	0	0	|	4
-1	0	0	1	1	|	4

La soluzione è, come al passo precedente, ammissibile ma non ottima. Nel duale porteremmo in base y_2 facendo uscire z_2, quindi la mossa equivalente sul primale è portare in base x_2 facendo uscire di base s_2. Arriviamo finalmente ad una soluzione ammissibile e ottima, corrispondente al punto B nel grafico iniziale:

x_1	x_2	s_1	s_2	φ		
1	0	-1	$\frac{1}{2}$	0	|	?
0	1	0	-1	0	|	?
0	0	-4	-3	1	|	7

Notiamo che sia il punto B del grafico del compratore che il punto b del grafico del produttore sono punti ammissibili e ottimi per entrambi i problemi.

Dividiamo le soluzioni di base in quattro categorie, con le conseguenti azioni necessarie:

Ammissibile	Ottima	
sì	no	Applico simplesso
sì	sì	Risultato desiderato
no	no	Inutilizzabile
no	sì	Applico simplesso duale

Ricordiamo che il metodo del simplesso duale si muove al di fuori della regione ammissibile, ma pur sempre sulle linee create dai vincoli.

Controlliamo la complementarietà dei punti di entrambi i problemi. Ad esempio, il punto C sul problema del compratore ha come variabili di base x_2 e s_1, mentre il punto c sul problema del produttore ha come variabili non di base le complementari di x_2 e s_1, ovvero z_2 e y_1. Allo stesso modo, D è una soluzione ammissibile ma non ottima per il compratore, mentre d è una soluzione non ammissibile ma ottima per il produttore.

Quando si usa il simplesso duale? Quando, a partire da una soluzione di base ottima, si ha una variazione che causa una perdita di ammissibilità. Lo scostamento può essere causata da una variazione nei dati, come ad esempio nell'analisi di sensitività, oppure per l'inserimento di nuovi vincoli. La variazione dei dati può spostare uno dei vincoli di cui il punto di ottimo è vertice. L'aggiunta di vincoli può tagliare la porzione di regione ammissibile in cui si trova l'ottimo. Il simplesso duale è in grado di muoversi, in pochi passi, verso la nuova soluzione ottima. Non richiede di riavviare il problema dall'inizio, come invece sarebbe necessario volendo procedere solamente per simplesso ordinario.

Definiamo una variazione massima δ per la quale la base ottima attuale resta ammissibile:

$$\tilde{\underline{b}} = \underline{b} + \underline{e}_1 \delta.$$

Calcoliamo di conseguenza il nuovo valore di $\underline{w}_B$:

$$\underline{\tilde{w}}_b = B^{-1}\underline{\tilde{b}} = B^{-1}b + B^{-1}\underline{e}_1\delta$$

Possiamo determinare l'intervallo di valori di δ per i quali la base rimane ammissibile, costruendo un sistema in cui garantiamo che tutte le componenti di $\underline{\tilde{w}}_b$ restino positive o nulle. Possiamo anche ricavare la dipendenza del valore della funzione obiettivo da δ. Con un'analisi più attenta, ci accorgiamo che i valori di δ fuori dall'intervallo trovato rendono negativi uno o più valori sulla riga pivot del *tableau* duale, rendendo inapplicabile la regola dei rapporti e dunque rendendo irrisolvibile il problema.

Teoremi sul simplesso duale

Teorema della dualità forte

> Se un problema (primale) ha soluzione ottima finita, allora anche il suo duale ha soluzione ottima finita, e i valori ottimi delle rispettive variabili obiettivo coincidono.

Teorema degli scarti complementari

> Siano (x, s) una soluzione ammissibile per il problema primale e (z, y) una soluzione ammissibile per il problema duale. Le soluzioni (x, s) e (z, y) sono ottime per i rispettivi problemi se e solo se
>
> $$x_j \cdot z_j = 0 \quad \forall j \qquad \text{e} \qquad y_i \cdot s_i = 0 \quad \forall i$$
>
> Queste condizioni sono dette *condizioni di complementarietà*. Esse si possono anche esprimere nella forma
>
> $$x_j \cdot \left(\sum_{i=1}^{m} a_{i,j}y_i - p_j\right) = 0 \quad \forall j$$
>
> $$\left(b_i - \sum_{j=1}^{n} a_{i,j}x_j\right) = 0 \quad \forall i$$

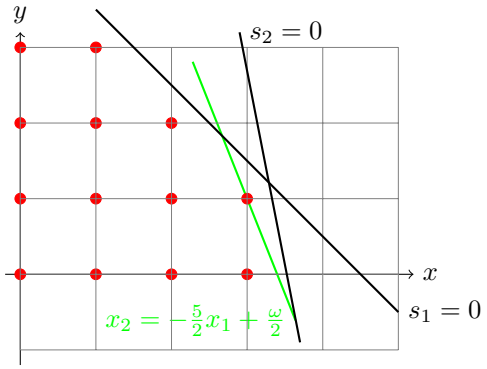
Teorema della dualità debole

Se il duale è illimitato dato che il primale è minore o uguale al duale per i problemi di massimo, allora il primale è irrisolvibile. Vale anche il contrario, ovvero che se il primale è limitato e il duale è da minimizzare, allora quest'ultimo non ha soluzione.

Problemi PLI

L'acronimo PLI sta per *Programmazione Lineare Intera* e indica una classe di problemi di programmazione lineare in cui tutte le variabili decisionali sono intere. Esistono due principali algoritmi risolutivi, il *metodo a piani di taglio* (R. Gomory, IBM, 1963) e il *metodo branch-and-bound*. Entrambi sono basati sul metodo del simplesso duale. Si definisce *mixed-integer linear programming* (MILP) o semplicemente *mixed-integer programming* (MIP) la categoria di problemi di programmazione lineare in cui si osserva una mescolanza tra variabili decisionali intere e variabili decisionali continue. Nel caso in cui tutte le variabili decisionali siano soggette a vincolo di integralità, si parla invece di *programmazione lineare intera* (PLI) o *integer linear programming* (ILP). Il caso MIP può essere facilmente ricavato dopo aver definito un metodo risolutivo per la PLI.

Si inizi da una considerazione sulla regione ammissibile. Per un problema PL a n vincoli, lo spazio delle soluzioni è $\mathbb{R}^n_+$. Per un problema PLI lo spazio perde cardinalità riducendosi a $\mathbb{N}^n$. Volendone dare una rappresentazione cartesiana, la regione ammissibile smette di essere una regione continua di (iper)piano. Essa si riduce infatti a un insieme finito di punti, gli incroci delle linee della porzione di griglia intera racchiusa tra i vincoli del problema. La soluzione ottima si determina graficamente costruendo innanzitutto una retta che abbia come equazione la funzione obiettivo. Dopodiché si costruiscono parallele alla retta della funzione obiettivo passanti per i punti della regione ammissibile. Si cerca infine la parallela con l'intercetta maggiore. Il punto (o i punti) attraverso cui passa tale parallela rappresenta(no) la soluzione ottima al problema.

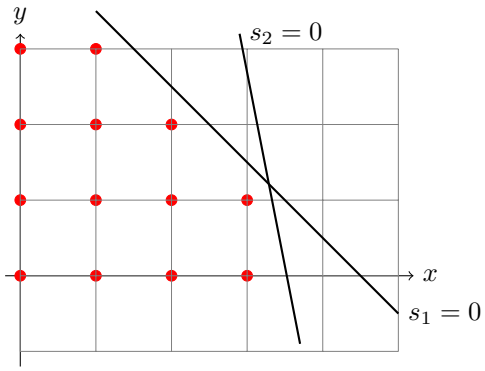


Metodo a piani di taglio

È facile notare che il punto di ottimo si trovi all'interno della regione ammissibile, e non sul suo contorno. Questa significativa differenza impedisce di utilizzare direttamente il metodo del simplesso come precedentemente definito per i problemi PL a variabili continue. Possiamo riadattare il problema, aggiungendo vincoli aggiuntivi che si intersechino presso il punto di ottimo. Per evitare che alterino la soluzione del problema, i nuovi vincoli devono essere ridondanti. Ogni nuovo vincolo deve essere dunque soddisfatto a priori da tutti i punti appartenenti alla regione ammissibile originale. Nello specifico, li ricaviamo dalla forma canonica della soluzione ottima. Questa è l'intuizione di base su cui si fonda il *metodo a piani di taglio*.

Lo si analizzi partendo da un esempio di riferimento:

$$\begin{aligned} \max \omega = \quad & 5x_1 + 2x_2 \\ & 2x_1 + 2x_2 \leq 9 \\ & 3x_1 + x_2 \leq 11 \\ & x_1, x_2 \geq 0 \text{ e intere} \end{aligned}$$



È necessario innanzitutto rilassare i vincoli di integralità e trovare la soluzione ottima continua con il metodo del simplesso. Ci si aspetta che quest'ultima sia vicina alla soluzione integrale che è ancora ignota.

$$\left[\begin{array}{ccccc|c} x_1 & x_2 & s_1 & s_2 & \varphi & \\ 0 & 1 & \frac{3}{4} & -\frac{1}{2} & 0 & \frac{5}{4} \\ 1 & 0 & -\frac{1}{4} & \frac{1}{2} & 0 & \frac{13}{4} \\ \hline 0 & 0 & -\frac{1}{4} & -\frac{3}{2} & 1 & -\frac{75}{4} \end{array}\right]$$

Ogni coefficiente deve essere diviso in una parte intera e una parte frazionaria, entrambe positive:

$$\begin{array}{ll} x_1 + \frac{3}{4}s_1 - \frac{1}{2}s_2 = \frac{5}{4} & x_2 + (0 + \frac{3}{4})s_1 + (-1 + \frac{1}{2})s_2 = 1 + \frac{1}{4} \\ x_1 - \frac{1}{4}s_1 + \frac{1}{2}s_2 = \frac{13}{4} & x_1 + (-1 + \frac{3}{4})s_1 + (0 + \frac{1}{2})s_2 = 3 + \frac{1}{4} \\ \omega + \frac{1}{4}s_1 + \frac{3}{2}s_2 = \frac{75}{4} & \omega + (0 + \frac{1}{4})s_1 + (1 + \frac{1}{2})s_2 = 18 + \frac{3}{4} \end{array}$$

La parte intera coincide con il massimo intero che precede il numero, mentre la parte frazionaria è l'avanzo rimanente.

Successivamente, si sceglie arbitrariamente una delle righe del *tableau* finale. In questo caso scegliamo di ottenere il taglio di Gomory dalla riga 0, relativa alla variabile obiettivo.

$$\begin{gathered} \omega + (0 + \tfrac{1}{4})s_1 + (1 + \tfrac{1}{2})s_2 = 18 + \tfrac{3}{4} \\ \omega + \tfrac{1}{4}s_1 + s_2 + \\ \textit{terza equazione} \end{gathered}$$

Il termine $\frac{1}{4}s_1 + \frac{1}{2}s_2$ è sempre positivo o nullo perché combinazione lineare di variabili scarto (sempre positive o nulle per definizione) con coefficienti positivi.

Sostituiamo s_2 e ω che ricaviamo dalle combinazioni lineari. Otteniamo il nuovo vincolo

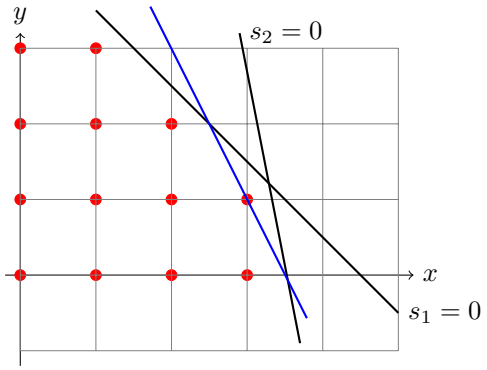
$$x_2 \leq -2x_1 + 7 + \frac{3}{4}$$

che è una retta passante per il punto di ottimo $(\frac{13}{4}, \frac{5}{4})$, e che è ridondante sia per il problema intero che per quello continuo.

Utilizzando invece

$$\omega + s_2 \leq 18,$$

ovvero considerando solo la parte intera del termine noto ed escludendo la parte frazionaria, otteniamo una restrizione del problema a rilassamento continuo. Stiamo infatti escludendo, tra l'altro, l'ottimo continuo. Non stiamo invece eliminando nessuna delle soluzioni ammissibili intere. Nella prima iterazione, corrispondente al puro rilassamento continuo, avevamo la massima regione ammissibile ausiliaria possibile. Ora ne viene esclusa una porzione che non porta alcuna soluzione ammissibile intera. L'idea è arrivare iterativamente al contenitore minimale, noto come *guscio convesso*, il più stretto possibile che continui a contenere tutti i punti interi ammissibili.



Notiamo nella figura che il nuovo vincolo passa per il punto di ottimo, ma questi non è ancora raggiungibile con il metodo del simplesso perché è su un lato ma non su di un vertice. Servirà un nuovo vincolo integrale che passi per il punto.

$$\underbrace{\omega + s_2}_{A} + \underbrace{\frac{1}{4}s_1 + \frac{1}{2}s_2}_{B} = \underbrace{18}_{C} + \underbrace{\frac{3}{4}}_{C}$$

che diventa

$$\begin{matrix} A + B = C + D \\ A \leq C \end{matrix} \Rightarrow B \geq D : \frac{1}{4}s_2$$

$$\left[\begin{array}{cccccc|c} x_1 & x_2 & s_1 & s_2 & s_3 & \varphi & \\ \hline 0 & 1 & \frac{3}{4} & -\frac{1}{2} & 0 & 0 & \frac{5}{4} \\ 1 & 0 & -\frac{1}{4} & \frac{1}{2} & 0 & 0 & \frac{13}{4} \\ 0 & 0 & -\frac{1}{4} & -\frac{1}{2} & 1 & 0 & -\frac{3}{4} \\ \hline 0 & 0 & -\frac{1}{4} & -\frac{3}{2} & 0 & 1 & -\frac{75}{4} \end{array}\right]$$

Abbiamo aggiunto una nuova riga, e questo ci richiede di aggiungere una nuova variabile di base (relativa al vincolo nuovo): s_3. Possiamo procedere per simplesso duale. Abbiamo una riga il cui coefficiente della variabile di base assume coefficiente negativo (s_3). La facciamo dunque uscire di base, capendo invece che debba entrare in base una tra s_1 e s_2, che hanno coefficienti negativi nella riga dell'1 di s_3. Calcoliamo i rapporti con la regola dei rapporti trasposta (tra i coefficienti delle colonne nella riga 0 e i coefficienti delle colonne nella riga relativa a s_3):

$$\begin{matrix} -\frac{1}{4} / -\frac{1}{4} = 1 \\ -\frac{3}{2} / -\frac{1}{2} = 3 \end{matrix}$$

Scelgo il più piccolo (1) che corrisponde a s_1: s_1 entra in base al posto di s_3. In pratica, per annullare s_3 portando in base s_1, scorro verso il basso lungo la retta di s_1 fino al punto in cui si incrocia con s_3. Notiamo sul grafico che questo punto è fuori dalla regione ammissibile, quindi dobbiamo applicare di nuovo il simplesso duale.

$$\left[\begin{array}{cccccc|c} x_1 & x_2 & s_1 & s_2 & s_3 & \varphi & \\ \hline 0 & 1 & 0 & -2 & 3 & 0 & -1 \\ 1 & 0 & 0 & 1 & -1 & 0 & 4 \\ 0 & 0 & 1 & 2 & 4 & 0 & 3 \\ \hline 0 & 0 & 0 & -1 & -1 & 1 & -18 \end{array}\right]$$

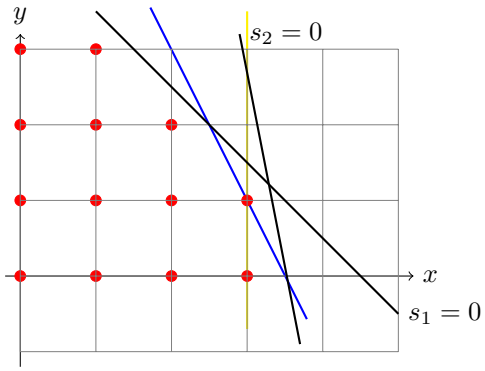
Procediamo nuovamente. Il *pivot* è sulla prima riga e quarta colonna.

$$\left[\begin{array}{cccccc|c} x_1 & x_2 & s_1 & s_2 & s_3 & \varphi & \\ \hline 0 & -\frac{1}{2} & 0 & 1 & -\frac{3}{2} & 0 & \frac{1}{2} \\ 1 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & \frac{7}{2} \\ 0 & 1 & 1 & 0 & -1 & 0 & 2 \\ \hline 0 & -\frac{1}{2} & 0 & 0 & -\frac{5}{2} & 1 & -\frac{35}{2} \end{array}\right]$$

Procediamo di nuovo rilassando l'integralità. La soluzione è ancora frazionaria, quindi serve un altro passo di taglio:

$$\omega + \frac{1}{2}x_2 + 2s_3 + \frac{1}{2}s_3 = 17 + \frac{1}{2}$$

Il taglio in forma intera è $\omega + 2s_3 \leq 17$, mentre la forma frazionaria è $\frac{1}{2}x_2 + \frac{1}{2}s_3 \geq \frac{1}{2}$. Il nuovo vincolo è verticale e si incrocia con il precedente proprio nel punto di ottimo. Il prossimo passo di simplesso conduce proprio a tale punto.



Notiamo che ad ogni passo l'algoritmo aumenta le dimensioni del problema di uno, per aggiungere il nuovo vincolo, e nessuno dei vincoli dei passaggi precedenti può essere scartato nei passaggi nuovi.

Metodi branch-and-bound

Il metodo di soluzione alternativo ai piani di taglio è detto *branch-and-bound*. L'algoritmo procede ricorsivamente, eliminando ad ogni passo una striscia della regione ammissibile che non contenga soluzioni intere. Dopodiché, si riapplica alle due porzioni rimanenti, situate ai lati della striscia eliminata.

Si analizzi ora l'algoritmo in maggiore dettaglio. Per rilassamento continuo si trova la prima soluzione non intera. Nell'esempio si risolveranno graficamente i sottoproblemi, ignorando deliberatamente i tableau, anche se solitamente utilizzati nella risoluzione vera e propria.

> [!Problema d'esempio] Un rivenditore al dettaglio acquista all'ingrosso ogni giorno 2 prodotti che rivende nell'arco della giornata:
>
> - 1 cassa del prodotto 1 costa 10 euro e la vendita al dettaglio genera un profitto di 10 euro
> - 1 cassa del prodotto 2 costa 19 euro e la vendita al dettaglio genera un profitto di 4 euro. Il rivenditore ha un budget giornaliero di 190 euro.
>
> Il rivenditore trasferisce le casse acquistate al negozio mediante il proprio furgone che può trasportare:
>
> - fino a 7 casse, se viene trasportato solo il prodotto 1 (l'ingombro di una cassa di prodotto 1 è 1/7 della capacità del furgone)
> - fino a 20 casse, se trasporta solo il prodotto 2 (l'ingombro di una cassa di prodotto 2 è 1/20 della capacità del furgone)
> - qualsiasi combinazione di casse dei 2 prodotti che non eccede la capacità del furgone. Determinare quante casse di ciascun prodotto il rivenditore deve acquistare per massimizzare il profitto.

La struttura dati utilizzata per immagazzinare i risultati intermedi è nota come *albero di branch-and-bound*. Nella sua iterazione iniziale contiene soltanto un nodo, relativo al problema completo:

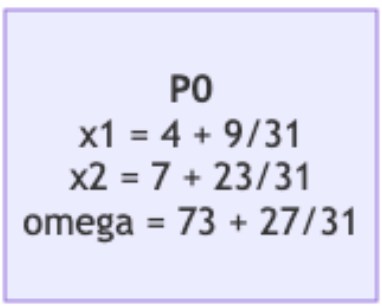


Figura 5: diagram

Scegliamo ad esempio di lavorare su x_1. Dato che il valore di x_1 è intero e compreso tra 4 e 5, tripartiamo il vincolo: - $x_1 \leq 4$ - $4 \leq x_1 \leq 5$ - $x_1 \geq 5$ Possiamo escludere la fascia centrale dalla ricerca, perché non può contenere soluzioni intere

se non agli estremi che sono già compresi nelle altre due fasce. Bipartiamo il problema rispetto a x_1: - $P_1 : P_0 \wedge x_1 \leq 4$ - $P_2 : P_0 \wedge x_1 \geq 5$

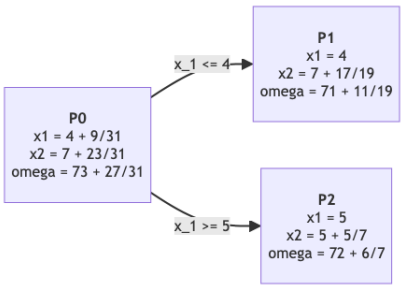
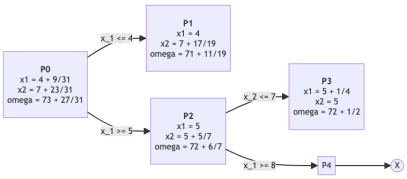
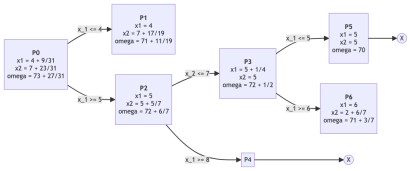


Figura 6: diagram

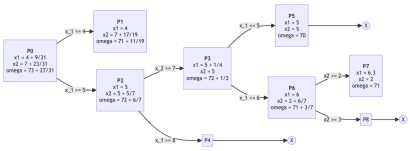


Costruiamo altri due problemi a partire da P_2, sul vincolo x_2:

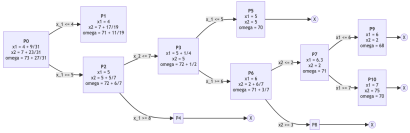
Notiamo che P_4 è inammissibile, e dunque terminiamo il suo ramo. Continuiamo dividendo P_3 in base a x_1:



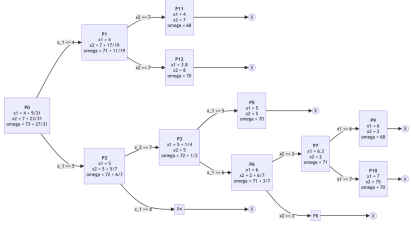
Terminiamo il ramo di P_5 perché abbiamo trovato una soluzione intera ammissibile. Continuiamo da P_6:



Chiudiamo P_8 perché è inammissibile. Procediamo di nuovo da P_7:



Chiudiamo i rami di P_9 e P_{10} perché abbiamo soluzioni ammissibili. Terminiamo ora considerando le ramificazioni di P_1:



Chiudiamo P_{11} per ammissibilità e P_{12} per subottimalità (abbiamo già 2 soluzioni ammissibili con valore obiettivo minore). Possiamo ora scegliere, tra le soluzioni ammissibili foglia, quella con il valore dell'obiettivo più basso come soluzione ottimale vera.

È possibile stimare un *gap di ottimalità* a partire dalle soluzioni trovate fino a una data iterazione, e terminare anticipatamente prima quando il *gap* scende sotto ad un valore desiderato.

Si può evitare il *branch-and-bound* arrotondando direttamente la soluzione del rilassamento continuo, ma in questo modo non si garantiscono né l'ottimalità né l'ammissibilità della soluzione. Si può reintegrare l'ammissibilità ma questo potrebbe aumentare ulteriormente la distanza dalla vera soluzione intera. Si ricordi comunque che l'errore di arrotondamento è tanto più piccolo quanto più sono grandi i numeri considerati.

Ottimizzazione su grafi

I problemi di ottimizzazione sui grafi sono a soluzione integrale. Un esempio di problema appartenente a tale categoria è la selezione (binaria) degli archi da inserire in un cammino a costo minimo. L'integralità della soluzione non è da raggiungere algoritmicamente, ma è una proprietà insita nella struttura del problema. Questo deriva da una proprietà della regola di Cramer, enunciata di seguito:

> *data $A : m \times m$ matrice dei coefficienti di un sistema di equazioni $A\underline{x} = \underline{b}$, le soluzioni sono del tipo*
>
> $$x_j = \frac{\det(A_j)}{\det(A)}$$
>
> *dove A_j è ottenuta sostituendo la j-esima colonna di A con il vettore colonna $\underline{b}$ dei termini noti.*

Una proprietà significativa è che l'integralità del determinante di A_j dipende dall'integralità delle componenti di $\underline{b}$. Dato che la matrice delle variabili di base nel tableau del simplesso ha sempre determinante ± 1, essendo composta da versori, allora se $\underline{b}$ è integrale, le soluzioni x_j saranno a loro volta integrali.

Ripasso grafi

Un grafo $G = (V, E)$ è una coppia di insiemi di *vertici* (*nodi*) e *lati* (*archi*). I collegamenti sono coppie non ordinate nel caso di grafi non orientati e coppie ordinate nel caso di grafi orientati. Si dice *stella* l'insieme dei lati incidenti. Nel caso di grafi orientati si distingue tra *stella entrante* (*backward star*), definita come insieme degli archi entranti, e *stella uscente* (*forward star*) composta di archi uscenti.

In un grafo non orientato chiamiamo ??? (finire, slide 4)

Problema della raggiungibilità

Dato un nodo $s \in N$ del grafo $G = (N, A)$ determinare i nodi raggiungibili da s, ossia il sottoinsieme $m \subseteq N$

Algoritmi di visita

Dato un grafo orientato e un nodo origine r, determinare i nodi raggiungibili da r

Ordinamento degli archi: considerare l'ordine in cui sono in N e poi, per ogni nodo, ordinare gli archi in modo crescente in base al nodo di testa.

Q è la lista dei nodi di testa degli archi che appartengono alle stelle uscenti dei nodi già raggiunti. È una lista dalla quale estrarre ad ogni iterazione il primo elemento. Abbiamo anche un vettore $\underline{p}$ di cui l'elemento i-esimo indica il nodo che precede immediatamente i nel cammino da r a i determinato nelle iterazioni precedenti alla corrente.

Possiamo lavorare con una pila o con una coda.

```
Procedure Cammini orientati(G,r,p):
    begin
        for i := 1 to n do p[i] := 0,
        Q := {r}; p[r] := nul;

        repeat
            i := Select(Q);    Q := Q\{i};
            for each (i,j) in FS(i) do
                if p[j] := 0 then begin p[j] := i; Q := Q U {j} end
        until Q = [];
    end
```

All'inizio della procedura il nodo radice r è l'unico elemento raggiungibile (unico elemento di Q) e non ha predecessori. Nel loop principale si cancella da Q il nodo i selezionato. Nel `for-each` annidato, per ogni arco della stella uscente si controlla il valore del predecessore. Se è nullo, significa che il nodo raggiunto dall'arco non è mai stato raggiunto prima e va messo nella lista dei predecessori. In caso contrario, significa che il nodo è già stato raggiunto e non va considerato. Il loop continua finché ci sono elementi in Q.

I cammini orientati possono essere ricostruiti utilizzando i predecessori dei nodi. Di fatto, utilizzando solo i cammini trovati, costruiamo l'albero dei cammini, che è una versione aciclica del grafo di partenza. L'albero dei cammini è diverso a seconda che si utilizzi la versione dell'algoritmo implementata con la pila o con la coda. In particolare, l'approccio basato sulla pila tende a produrre cammini di lunghezza maggiore o uguale di quelli determinati con la pila.

Il costo della visita è $O(m)$ dove m è il numero di archi del grafo. Questo perché ogni arco è considerato al massimo una volta.

Determinare l'insieme dei nodi raggiungibili con cammini orientati a partire da uno qualunque dei nodi di un dato insieme $R \subset N$ ...

Possiamo trovare la soluzione modificando l'insieme dei nodi definendo un nodo fittizio s chiamato *superradice*, e creando nuovi archi che colleghino tutti i nodi alla superradice.

Determinare l'insieme die nodi dai quali è possibile raggiungere, mediante cammini orientati, un nodo destinazione d. Dobbiamo definire un grafo modificato in cui gli archi siano ottenuti invertendo il verso degli archi del grafo originale.

Alberi

Un grafo $G = (N, A)$ con n nodi è detto *albero* se è connesso (ovvero esiste un cammino tra ogni coppia di nodi) ed è aciclico. Alternativamente, si possono utilizzare le seguenti definizioni: - G è connesso e ha $n-1$ archi - G è aciclico e ha $n-1$ archi - esiste un unico cammino che congiunge ogni coppia di nodi

Un *albero di supporto* (o *di copertura*) è un albero definito come sottografo di un grafo non orientato $G = (V, E)$ contenente tutti gli elementi di V.

Un albero con n nodi ha $n-1$ collegamenti, esiste un unico cammino tra qualunque coppia di nodi, aggiungendo un collegamento qualsiasi si ottiene un unico ciclo. Proprietà di scambio: sia dato un albero di supporto. Gli si aggiunga un lato a caso tra due elementi. Rimuovendo un lato qualsiasi dal ciclo che si forma, la nuova struttura è a sua volta un albero di supporto.

Determinazione dell'MST

Un problema importante è la determinazione dell'albero di supporto a costo minimo (MST), contenente gli archi che minimizzano la sommatoria di tutti i costi.

Una prima tecnica è il metodo di Kruskal. Si inizializza ordinando collegamenti in ordine di costo non decrescente. Si itera tra i collegamenti non ancora considerati per scegliere quello a costo minimo. Se il suo inserimento non genera cicli, lo si inserisce nell'albero di supporto, altrimenti lo si elimina dalla lista dei candidati. L'approccio è *greedy*, ma per questo tipo di problemi produce una soluzione ottima.

```
Algoritmo Kruskal(V,E,T*)
begin
    ordinare i collegamenti di E secondo valori non crescenti
    porre T* = []
    while |T*| < n-1 do
    begin
        scegliere un collegamento e in E di costo minimo
        porre E = E\{e}
            if T* U {e} non ha cicli then T* = T* U {e}
        end
    end
    return T*
end
```

Un'alternativa al metodo di Kruskal è l'algoritmo di Prim. Esso garantisce, al contrario di Kruskal, che l'albero sia sempre connesso anche durante le iterazioni della costruzione. Si sceglie una radice. Si effettua un *taglio*, ovvero una partizione tra i nodi appartenenti all'albero e quelli non ancora appartenenti. Si definiscono *archi di taglio* gli archi attraversati dalla linea di partizione. Essi saranno i candidati tra cui scegliere. Tramite una scelta *greedy*, si seleziona quello di costo minore e lo si inserisce nell'albero ottimale insieme al nodo a cui porta. Si procede iterativamente fino a completare l'albero. In caso di parità di costo sugli archi candidati, la scelta è arbitraria.

```
Algoritmo Prim (V, E, T*)
begin
    porre T = []
    porre S = 1
    while |T*| < n-1 do
    begin
        scegliere un collegamento {v, h} in delta(S) di costo minimo
        porre T* = T* U v, h
        porre S = S U h
    end
    return T*
end
```

Si definisce *lato di diminuzione* un lato che, se aggiunto ad un albero secondo le modalità della proprietà di scambio, genera un ciclo contenente lati di costo maggiore ad esso. Un albero di supporto è detto ottimale se e solo se non ammette collegamenti di diminuzione. La ricerca richiede al peggio $m-(n-1)$ iterazioni con n nodi e m collegamenti. Come regola empirica, conviene esaminare i collegamenti in ordine crescente di costo.

Problema del messaggio segreto

Vogliamo minimizzare la possibilità che un messaggio segreto venga intercettato, facendo comunque in modo che raggiunga tutti i destinatari. Si tratta di trovare un albero di supporto che minimizzi il prodotto delle probabilità di intercettazione. Possiamo trasformare il prodotto in una sommatoria applicandone il logaritmo (che essendo monotono crescente non sposta l'ottimo), in modo da trasformare il problema in una determinazione di MST (Maximum in questo caso).

```
Algoritmo Kruskal_max (V, E, T*)
begin
    ordinare i collegamenti di E secondo valori non crescenti
    porre T* = []
    while |T*| < n-1 do
    begin
        scegliere un collegamento e in E di valore massimo
        porre E = E\{e}
        if T* U {e} non ha cicli then
            T* = T* U {e}
        end
    end
    return T*
end
```

Problema dei cammini a costo minimo

La società Endpower gestisce una rete di distrubuzione del carburante. Il deposito centrale è rappresentato dal nodo 1. Si vuole determinare il percorso migliore per consegnare il carburante fino alla stazione rappresentata dal nodo 8. Determinare il cammino di costo minimo da sorgente a destinazione.

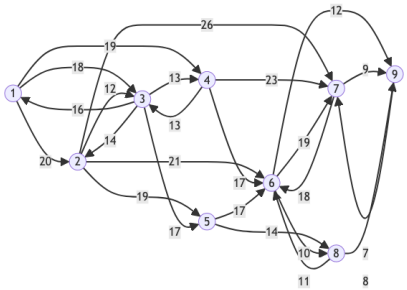


Figura 7: diagram

Per ogni nodo, dobbiamo imporre il bilancio tra gli archi entranti e gli archi uscenti. Questa proprietà vale anche sui singoli percorsi. Alcuni nodi avranno anche ingressi e uscite esterne al grafo. Il deposito rappresentato dal nodo 1 riceve carburante dall'esterno. Gli altri nodi vendono carburante. Anche questo bilancio deve essere mantenuto. Si portano le sommatorie a sinistra dell'uguale e i termini noti a destra:

$$\sum \text{entranti} - \sum \text{uscenti} = \text{termine noto}.$$

I nodi intermedi hanno termine noto nullo, i nodi origine hanno -1 e i nodi destinazione hanno +1. Tale costante è detta *peso del nodo*. Si parla, in questo caso di *vincoli di conservazione del flusso*. Possiamo costruire una matrice di incidenza con i nomi dei nodi sulle righe e i nomi degli archi sulle colonne. La colonna associata ad un arco ha 1 sul nodo in cui entra, -1 su quello da cui esce, 0 sugli altri. Per righe riconosciamo le stelle entranti e uscenti di ogni nodo. Detta E la matrice di incidenza e x il vettore dei flussi sugli archi, i vincoli saranno $Ex = b$ dove b sono i pesi dei nodi. L'obiettivo è ottimizzare $\min c^T x$.

Per la risoluzione del problema dei cammini a costo minimo esistono tre algoritmi, in base alla struttura del grafo. Il primo, detto *algoritmo di Dijkstra*, si applica a grafi con presenza di cicli e $c_{i,j} \geq 0$, $\forall (i,j) \in A$. Il secondo, basato sull'*ordinamento topologico* dei nodi, è per soli grafi aciclici. Il terzo, più generico ma più pesante, è l'agoritmo di *Floyd-Warshall* per grafi con presenza di cicli e con $c_{i,j} < 0$ per qualche $(i,j) \in A$

Analizziamo ora il primo algoritmo.

Caso 1 (grafi a costo non nullo): algoritmo di Dijkstra

Una componente fondamentale di tutte le soluzioni è l'insieme S, che contiene i nodi per i quali è già stato determinato il costo minimo a partire dal nodo radice s. Il nodo i-esimo viene messo in S quando viene trovato il costo minimo da s a i. Durante l'inserimento, ad ogni nodo $i \in S$ viene data un'etichetta che contiene due informazioni: il suo predecessore pred$[i]$ nel cammino a costo minimo e il costo $L[i]$ del cammino.

All'inizio dell'esecuzione dell'algoritmo viene messo in S il nodo radice s, lasciando nullo il campo del predecessore nell'etichetta e segnando come zero il costo del cammino. All'inizio di ogni interazione successiva si effettua un *taglio*, per selezionare gli archi che si trovano sulla partizione $(S, N \setminus S)$. Si individuano i nodi raggiungibili tramite gli archi tagliati. Nel caso dell'esempio in figura, già alla prima iterazione si individua il cammino minimo tra i nodi 1 e 3. Se tutti i costi sono non-negativi, sarà impossibile trovare un cammino più breve passando dal nodo 2, perché già soltanto arrivare a 2 richiede un costo maggiore rispetto al raggiungimento diretto di 3. Lo stesso non si può dire del nodo 2.

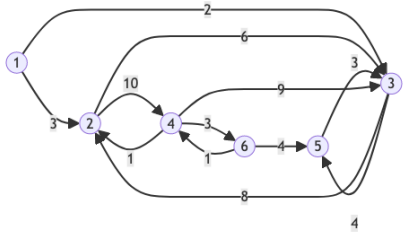


Figura 8: diagram

```
Algoritmo Dijkstra(N,A,c,s,Pred,L)
begin
    porre S={s}, Pred[s] = nil, L[s] = 0;
    while |s| < n do
        begin
            forall(i,j) in deltaplus(s) = {(i,j):(i,j) in A, i in S, j not in S}
                calcolare L[i] + c[i][j];
            determinare (v,h) in deltaplus(s): L[v]+c[v][h] =
                min{L[i]+c[v][h] : (i,j) in deltaplus(S)};
            porre Pred[h] = v; L[h] = L[v] + c[v][h]; S = S U {h};
        end
end
```

Caso 2 (grafi aciclici): ordinamento topologico dei nodi

L'*ordinamento topologico* dei nodi è un'operazione che sostituisce la denominazione dei nodi con una denominazione numerica con la proprietà di dare una corrispondenza d'ordine ai nodi in base agli archi. Il nodo da cui un arco esce deve sempre avere un numero inferiore rispetto al nodo in cui l'arco entra:

$$\forall (i,j) \in A: \quad i < j.$$

Riuscire a completare la procedura è la prova che il grafo non contiene cicli. In presenza di cicli, infatti, si giunge ad un'iterazione non completabile nella quale nessun nodo ha archi entranti. Riuscendo a dare una numerazione topologica ai nodi, e poi considerando i nodi in ordine numerico, si riesce ad avere una piena informazione sui cammini che arrivano fino al punto considerato. Il cammino di costo minimo dal nodo iniziale ad un nodo qualsiasi si ricostruisce percorrendo a ritroso la lista dei predecessori.

Caso 3 (grafi qualsiasi): algoritmo di Floyd-Warshall

In presenza di cicli e cammini a costi negativi, diventano possibili anche *cicli a costo negativo*. Questo causa problemi perché, nella determinazione del cammino a costo minimo, la scelta *greedy* porta a girare all'infinito nel ciclo a costo negativo invece che uscirne per muoversi verso il nodo destinazione. L'algoritmo di Floyd-Warshall è in grado di individuare i cicli a costo negativo e fermarsi anticipatamente in tal caso.

L'algoritmo utilizza una matrice D $n \times n$ i cui elementi $d_{i,j}$ contengono il costo del cammino più breve finora trovato tra i nodi i e j, e una matrice P $n \times n$ i cui elementi $p_{i,j}$ contengono i predecessore del nodo j nel cammino più breve finora trovato tra i e j.

I costi associati agli archi formano la matrice

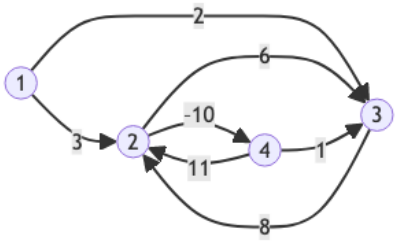


Figura 9: diagram

$$C = \begin{bmatrix} 0 & 3 & 2 & \infty \\ \infty & 0 & 6 & -10 \\ \infty & 8 & 0 & \infty \\ \infty & 11 & 1 & 0 \end{bmatrix}$$

dove si è posto $c_{i,j} = 0$ per ogni $i \in N$ e $c_{i,j} = \infty$ per ogni $(i,j) \notin A$.

Il cammino più breve tra i nodi i e j è inizialmente posto uguale all'arco che li collega, quindi il predecessore del nodo j nel cammino più brvee tra i e j è il nodo i: $p_{i,j} = i$

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

mentre il costo del cammino più breve tra i e k è il costo dell'arco (i,j):

$$C = D = \begin{bmatrix} 0 & 3 & 2 & \infty \\ \infty & 0 & 6 & -10 \\ \infty & 8 & 0 & \infty \\ \infty & 11 & 1 & 0 \end{bmatrix}$$

All'iterazione h viene eseguita la seguente operazione triangolare relativa al nodo h; per ogni coppia di nodi i, j viene considerato il nuovo cammino ottenuto passando per h, ovvero l'unione dei cammini da i a h e da h a j:

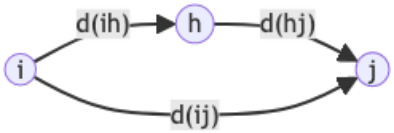


Figura 10: diagram

Se il costo del nuovo cammino (passante per h) da i a j è minore del costo del cammino più breve finora trovato, ossia $d_{i,h} + d_{h,j} < d_{i,j}$, si aggiornano sia il predecessore sia il costo di j per identificare il nuovo cammino più breve trovato. Il predecessore di j nel cammino da i viene aggiornato assegnando ad esso il predecessore di j nel cammino da h: $p_{i,j} \leftarrow p_{h,j}$. Il costo del cammino più breve finora trovato da i a j viene sostituito dalla somma dei costi dei cammini più brevi finora trovati da i ad h e da h a j: $d_{i,h} + d_{h,j} < d_{i,j}$. La presenza di elementi negativi sulla diagonale della matrice D indica la presenza di cicli a costo negativo. Il problema è dunque inferiormente limitato e l'algoritmo si interrompe.

Segue lo pseudocodice dell'algoritmo:

```
Algoritmo Floyd-Warshall (N, A, c)
    begin
        for i = 1 to n do
            for j = 1 to n do
                begin
                    porre d[i,j] = c[i,j] = p[i,j] = i
                end
            for h = 1 to n do \\operazione triangolare su h
                begin
                    for i = 1 to n do
```

```
                    for j = 1 to n do
                    if d[i,h] + d[h,j] < d[i,j] then
                        begin
                            porre d[i,j] = d[i,h] + d[h,j], p[i,j] = p[h,j];
                        end
                for i = 1 to n do
        if d[i,j] < 0 then STOP; \\ ciclo negativo
            end
end
```

Esempio: scheduling delle ispezioni su una linea di produzione

Una linea di produzione è costituita da n celle di lavorazione. La produzione è attualmente organizzata nel seguente modo:

- nella linea di produzione viene immesso un lotto costituito da B pezzi da lavorare:
  - i pezzi del lotto sono tutti "senza difetti", essendo stato effettuato un controllo preventivo nel quale gli eventuali pezzi difettosi sono stati scartati
- in ognuna delle n celle il lotto subisce una lavorazione:
  - costo di lavorazione nella cella j: p_j per pezzo lavorato
  - costo della lavorazione del lotto nella cella j è $B \cdot p_j$

Al termine della lavorazione n viene effettuata un'ispezione per scartare i pezzi divenuti difettosi nel corso della lavorazione e non consegnarli ai clienti.

Costo totale di produzione di un lotto (lavorazione nelle n celle):

$$B \cdot \sum_{j=1}^{n} p_j$$

Attualmente la lavorazione dei pezzi divenuti difettosi continua nelle celle successive anche se saranno scartati nel corso dell'ispezione al termine della lavorazione n:

> Un pezzo divenuto difettoso nel corso della lavorazione $j < n$ viene sottoposto alle lavorazioni da $j+1$ a n, anche se dovrà essere scartato alla fine della lavorazione n.

In alternativa, si possono ispezionare i pezzi lavorati nella cella j, $1 \leq j \leq n$, scartando quelli divenuti difettosi e continuando la lavorazione solo sui pezzi non difettosi, il cui numero è denotato con B_j.

Per l'ispezione alla fine della lavorazione j si devono sostenere un costo fisso, indipendente dal numero egli elementi selezionati, e un costo variabile, proporzionale al numero di elementi selezionati. In generale, entrambi i costi dell'ispezione alla fine della lavorazione j dipendono da quando è stata effettuata l'ispezione i immediatamente precedente:

- $i = 0$: ispezione eseguita al momento della formazione del lotto
- $1 \leq i \leq j-1$: ispezione eseguita al termine della lavorazione i (che precede j)

Definiamo anche

- $f_{i,j}, 0 \leq i \leq j-1$: costo fisso
- $h_{i,j}, 0 \leq i \leq j-1$: costo per ispezione

Il numero delle alternative tra cui scegliere aumenta esponenzialmente con il numero delle lavorazioni intermedie. Le ispezioni prima dell'inserimento del lotto e alla fine della lavorazione n sono sempre effettuate:

- il costo dell'ispezione prima dell'inserimento del lotto dipende solo da B, quindi, fissato B, è una costante nella funzione dei costi complessivi
- il costo dell'ispezione alla fine della lavorazione n dipende da quando è stata effettuata la precedente ispezione

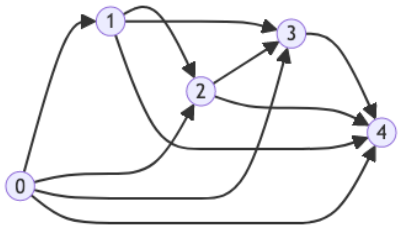


Figura 11: diagram

Il grafo è aciclico e ha già i nodi ordinati. Possiamo procedere con l'algoritmo topologico.

Problema del flusso massimo

Dato un grafo orientato $G = (N, A)$, ogni arco $(i, j) \in A$ ha capacità $u_{i,j}$, un nodo $s \in N$ è detto *nodo sorgente* o di *origine*, e un nodo $t \in N$ è detto *pozzo* o *destinazione*. È detta *problema di massimo flusso* la determinazione della massima quantità di flusso che può essere inviata da s a t in modo che in ogni arco il flusso non superi la capacità dell'arco e che in ogni nodo il flusso entrante sia uguale a quello uscente.

Il problema può essere strutturato come un modello di programmazione lineare, scegliendo come prima variabile decisionale la quantità immessa nella sorgente e prelevata al pozzo, e come altre variabili decisionali i flussi sui singoli archi. La conservazione del flusso al nodo sorgente richiede che il flusso immesso (da massimizzare) sia uguale al flusso uscente tramite gli archi. Nei nodi di trasferimento il flusso proveniente dagli archi entranti deve essere uguale al flusso per gli archi uscenti. Nel nodo pozzo il flusso entrante per gli archi entranti deve essere uguale al flusso prelevato (da massimizzare). Le variabili decisionali relative al flusso sugli archi sono limitate inferiormente a zero e superiormente alla capacità dell'arco considerato. Vogliamo massimizzare il flusso entrante / uscente agli estremi, affinché raggiunga il valore massimo compatibile con i vincoli interni al grafo. Per convenienza, si collegano direttamente il pozzo e la sorgente per trattare il grafo come un sistema isolato.

Massimo flusso, minimo taglio

I flussi sugli archi devono rispettare sia i vincoli di nonnegatività che i vincoli di bilancio dei nodi.

Si consideri un generico taglio $(S, N \setminus S)$, definito come una partizione dei nodi in due insiemi disgiunti S e $N \setminus S$. Per questo taglio, possiamo identificare due insiemi di archi:

- gli archi diretti del taglio, $A^+(S, N \setminus S)$, sono gli archi che partono da un nodo in S e arrivano a un nodo in $N \setminus S$. Questi archi sono definiti come:

$$A^+(S, N \setminus S) = \{(i, j) \in A : i \in S, j \in N \setminus S\}$$

- gli archi diretti inversi del taglio, $A^-(S, N \setminus S)$, sono gli archi che partono da un nodo in $N \setminus S$ e arrivano a un nodo in S. Questi archi sono definiti come:

$$A^-(S, N \setminus S) = \{(i, j) \in A : i \in N \setminus S, j \in S\}$$

Il flusso sul taglio è la somma algebrica del flusso sugli archi di taglio. È importante notare che il flusso sul taglio è sempre uguale al valore di v (flusso esterno) per qualsiasi taglio, e quindi è sempre positivo. Sommando le equazioni di bilancio sia su S che su $N \setminus S$, otteniamo il flusso sul taglio v. Questo risultato è fondamentale per comprendere il flusso totale attraverso il taglio. La capacità di taglio è la somma delle capacità degli archi diretti nel taglio.

Il *teorema del flusso massimo e del taglio minimo* (*max flow, min cut*) afferma che il massimo flusso attraverso la rete coincide con la minima capacità di taglio. Questo significa che per qualsiasi taglio, il valore del flusso è minore o uguale alla capacità di taglio. Utilizzando l'elencazione combinatoria, è possibile creare tutti i possibili tagli e determinare i limiti superiori per il flusso. La soluzione del problema coincide con il minimo di questi limiti superiori. Il teorema si dimostra considerando il modello duale rispetto a quello costruito. L'obiettivo del problema duale è determinare quali sono gli archi di taglio.

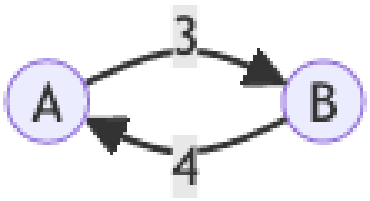
Il metodo risolutivo, noto come *algoritmo di Ford-Fulkerson*, è un procedimento iterativo. All'inizio di ogni iterazione, viene fornito un flusso ammissibile. A partire da questo flusso, si costruisce la *rete incrementale*, che rappresenta tutte le modifiche applicabili ai flussi correnti della rete originaria senza violare i vincoli di capacità e di nonnegatività. Queste modifiche sono quindi ammissibili e, applicandole, si ottiene un'altra soluzione alternativa ammissibile. Sulla rete incrementale si determina l'esistenza di un cammino dalla sorgente al pozzo lungo il quale inviare ulteriore flusso, noto come cammino aumentante. Se tale cammino non esiste, il flusso ammissibile corrente è ottimo. Se invece il cammino esiste, si determina l'incremento di flusso nel rispetto dei vincoli e lo si somma al flusso iniziale per generare il flusso ammissibile iniziale per l'iterazione successiva. L'algoritmo di Ford-Fulkerson non richiede inizializzazioni particolari e può partire da un flusso iniziale nullo.

La rete incrementale è una rappresentazione che consente di identificare le modifiche ammissibili ai flussi correnti della rete originaria senza violare i vincoli di capacità e di nonnegatività. Data una rete di flusso e il suo flusso ammissibile, la rete incrementale aggiunge un arco diretto di capacità residua tra due nodi se l'arco presente non è saturo, ovvero ha un peso inferiore al proprio vincolo di flusso. Inoltre, aggiunge un arco inverso di capacità residua pari al flusso corrente se l'arco non è scarico, ovvero se il flusso non è nullo. Se l'arco non è né saturo né scarico, la rete incrementale includerà entrambi gli archi. È importante notare che gli archi residui rappresentano soltanto i massimi e i minimi per la possibile variazione, e non i valori della variazione stessa.

Ad esempio, si consideri un arco con flusso corrente di 4 unità e flusso massimo di 7 unità.

Figura 12: diagram

In questo caso è possibile aumentare il flusso di al massimo 3 unità per raggiungere il massimo, o rimuoverne al massimo 4 per azzerare il flusso. La rete incrementale rappresenterà questa situazione con due archi: uno diretto con capacità residua di 3 unità, e uno inverso con capacità residua di 4 unità. La rete incrementale può essere visualizzata

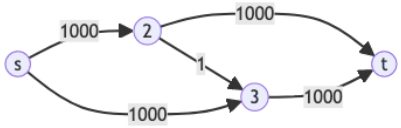


visualizzata con il seguente grafo residuale: In sintesi, la rete incrementale fornisce una struttura chiara per identificare le possibili modifiche ammissibili ai flussi correnti, facilitando la ricerca di cammini aumentanti e l'ottimizzazione del flusso nella rete.

Una volta costruita la rete incrementale, è possibile cercare un cammino composto esclusivamente da archi diretti che collega la sorgente al pozzo, noto come *cammino aumentante*. Il minimo tra gli incrementi presenti lungo questo cammino determinerà l'incremento complessivo rispetto al flusso ammissibile precedente. Successivamente, si aggiorna la rete originaria applicando le variazioni lungo il cammino aumentante. La ricerca del cammino aumentante nella rete incrementale rappresenta un test di ottimalità. Se viene trovato un cammino aumentante, ciò indica che la soluzione corrente non è ottima, poiché è possibile aumentare il flusso.

Un corollario del teorema di *max flow, min cut* permette di identificare il collo di bottiglia della rete. In assenza di un cammino aumentante, è possibile partizionare la rete incrementale in due insiemi: l'insieme dei nodi direttamente raggiungibili dall'origine e l'insieme dei nodi non raggiungibili. Tutti gli archi entranti nel taglio generato da questa partizione sono saturi, mentre tutti quelli uscenti sono scarichi. Questo partizionamento rivela il collo di bottiglia della rete, ovvero il punto in cui il flusso è limitato dalla capacità degli archi.

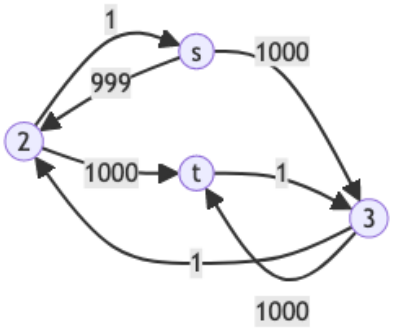
Si osservi un caso limite che evidenzia l'effetto della scelta del cammino aumentante sul numero di iterazioni dell'algoritmo.



Il problema di flusso massimo rappresentato dal seguente grafo si risolve in due iterazioni scegliendo di inviare:

- 1000 unità di flusso lungo il cammino $s, 2, t$
- 1000 unità di flusso lungo il cammino $s, 3, t$

Se invece si scegliesse come cammino aumentante $s, 2, 3, t$, che permette di inviare una sola unità di flusso, alla seconda



iterazione il grafo residuale sarebbe e anche all'iterazione successiva si potrebbe scegliere un cammino aumentante che aumenti solo di un'unità il flusso. Procedendo in questo modo, l'algoritmo richiederebbe 2000 iterazioni. Il problema può essere risolto decidendo di scegliere il cammino con il minor numero di archi (aggiunta di Edmonds e Karp all'algoritmo di Ford-Fulkerson). Questo dipende dalla differenza tra le strategie di visita in ampiezza e in profondità spiegata in precedenza: la scelta è possibile solo se l'algoritmo di visita gestisce l'insieme dei nodi raggiunti come una coda.

Tra i problemi riconducibili al flusso massimo vi è il trasferimento di merce o di valore dall'offerta alla domanda. In situazioni in cui l'offerta supera la domanda, è possibile introdurre un *nodo fittizio di domanda* che bilancia l'offerta in eccesso. Questo nodo fittizio assorbe la quantità di merce o valore che non trova corrispondenza nella domanda reale,

garantendo così l'equilibrio del sistema. Viceversa, in caso di domanda superiore all'offerta, è possibile introdurre un nodo fittizio di offerta che compensi il deficit, assicurando nuovamente l'equilibrio tra offerta e domanda.

Problema del commesso viaggiatore

Un *ciclo hamiltoniano* passa una e una sola volta per tutti i nodi di un grafo tornando infine al nodo iniziale. Il *problema del commesso viaggiatore* (*TSP, Travelling Salesman Problem*) consiste nel trovare il ciclo hamiltoniano di costo minimo tra tutti i possibili in un grafo. Il costo è definito come somma dei pesi degli archi lungo il ciclo.

Si formuli il problema di *TSP simmetrico*. La funzione obiettivo consiste nella somma dei pesi degli archi lungo il ciclo hamiltoniano. Vigono *vincoli di assegnamento*: ogni nodo deve avere esattamente un arco entrante e un arco uscente. Devono inoltre essere imposti vincoli per evitare la formazione di sottocicli: scelto un sottoinsieme qualsiasi di nodi, deve esistere almeno un arco uscente da uno di essi che porti ad un nodo non appartenente a tale sottoinsieme. In una formulazione alternativa, il numero di vincoli di connessione deve essere pari al numero di tagli tra il sottogruppo e i nodi restanti.

Il TSP può essere trattato come un *problema di ottimizzazione combinatoria*. L'insieme delle soluzioni ammissibili è quello di tutti i cicli hamiltoniani possibili sul grafo considerato. Se il grafo è orientato e completo, il numero di cicli hamiltoniani possibili è pari al numero di nodi, meno uno. Se il grafo non è orientato, il numero di cicli hamiltoniani è la metà rispetto al caso precedente. Risolvere il problema procedendo per enumerazione completa è possibile soltanto se il grafo è di piccole dimensioni: il numero di iterazioni richieste è infatti pari al fattoriale del numero di nodi. Fortunatamente esistono metodi alternativi basati su algoritmi *greedy*.

Algoritmo *greedy* per TSP

L'algoritmo si inizializza scegliendo il nodo iniziale come nodo corrente, e lasciando vuoto l'insieme dei nodi nel ciclo hamiltoniano da trovare. Ad ogni iterazione si individuano tutti gli archi tra il nodo corrente e i nodi non ancora inseriti nell'insieme dei nodi del ciclo. Si seleziona tra tali archi il più promettente secondo un'euristica e lo si aggiunge all'insieme. Si sceglie infine il nodo destinazione dell'arco scelto come nodo da considerare per l'iterazione successiva.

Il cirterio più comune è noto come *nearest neighbor* e consiste nello scegliere sempre l'arco di costo minore. Se il grafo è completo l'algoritmo riuscirà sicuramente a trovare un ciclo hamiltoniano. Se non lo è, può capitare che, a una data iterazione, tutti gli archi uscenti dal nodo considerato portino a nodi già inseriti nell'insieme. L'algoritmo si blocca e non viene trovato un cammino hamiltoniano anche se esso esiste. L'algoritmo è veloce ma può prendere strade non ottimali che non è in grado di correggere, perché le scelte passate non convenienti non vengono abbandonate.

Per migliorare una soluzione già trovata, esiste una procedura chiamata *local search* che applica il *2-scambio* al ciclo hamiltoniano trovato. Il 2-scambio consiste nello scambiare le destinazioni tra due coppie di nodi adiacenti. Ad esempio, avendo la connessione `1-2-3-4`, composta dalle coppie `1-2`e `3-4`, il 2-scambio la trasforma in `1-4-3-2`. Le possibilità di miglioramento del costo del cammino aumentano al crescere delle dimensioni del grafo, ma aumenta anche il costo elaborativo della procedura.

TSP metrico

La tecnica del *TSP metrico* è basata sulla disuguaglianza triangolare. L'algorimto sceglie un arco soltanto se non esiste un percorso alternativo di costo inferiore passante per un terzo nodo intermedio. L'euristica *twice around* ha un errore relativo massimo pari a 1. Si costruisce un albero di copertura a costo minimo a partire dal nodo iniziale. Si duplicano i suoi archi in modo da avere coppie di archi direzionati, entrambi di costo pari all'arco non orientato originale, che vadano verso e tornino da ogni nodo. Seguendo gli archi si trova un cammino che copre tutti i nodi in modo ordinato. Da questo grafo è possibile determinare un cammino hamiltoniano. Dove è conveniente, si seguono direttamente gli archi. In altri casi, è meno costoso passare direttamente da un nodo al successivo tramite un collegamento diretto che faccia da scorciatoia. Il cammino hamiltoniano risultante ha un costo compreso tra quello del cammino hamiltoniano a costo minimo e quello del cammino trovato lungo il grafo con gli archi a due vie considerato senza le scorciatoie. Il costo del secondo è pari al doppio del costo dell'albero di copertura determinato all'inizio, dato che è stato costruito raddoppiandone gli archi. L'errore relativo.

Branch-and-bound per TSP simmetrico

Il TSP simmetrico può essere risolto usando un metodo *branch-and-bound*. Per fare questo, è necessario interpretare il ciclo hamiltoniano come un *cammino hamiltoniano* unito ad un lato di chiusura. Il cammino hamiltoniano non è altro che un albero di supporto a costo minimo con nodi di grado minore o uguale a due. Unendo un albero di supporto a costo minimo al lato di costo minimo tra quelli non appartenenti all'albero, otteniamo un 1-albero il cui costo è limite inferiore per il valore ottimo del TSP. Si tratta di un limite inferiore e non di una soluzione perché il lato di costo minimo non corrisponde necessariamente al lato che serve per chiudere l'albero trasformandolo in un ciclo hamiltoniano.

La soluzione di questo problema è dunque un rilassamento del TSP che può produrre o non produrre una soluzione ammissibile per il problema originale.

Se l'albero ottenuto non genera un ciclo hamiltoniano, si effettua il *branching*: si sceglie il nodo di grado massimo e si cerca tra i suoi lati incidenti quello di costo minimo. Se il risultato è un ciclo hamiltoniano, si chiude il ramo e si annota la soluzione ammissibile trovata. Se non lo è, si chiude il ramo. Il limite inferiore per il costo è determinato scegliendo, tra le soluzioni non ammissibili, quella a costo minimo. Su ogni ramo, non è possibile utilizzare come lato di chiusura un arco che sia già stato utilizzato in una iterazione precedente su quello stesso ramo; in tal caso si sceglie il secondo migliore candidato, o il terzo, e così via. I lati già utilizzati sono detti *lati dominati*. Una volta conclusa la procedura di *branching*, si seleziona la soluzione ammissibile di costo minimo tra quelle trovate.

Minimal Cardinality Machine Scheduling

Un certo numero di lavori, con uno specifico momento d'inizio e una durata definita, deve essere eseguito su un certo numero di macchine, tutte uguali tra loro. L'obiettivo è utilizzare il minor numero di macchine evitando allo stesso tempo sovrapposizioni tra i lavori.

Il problema è rappresentabile come un PLI. Serve un numero di variabili binarie pari al prodotto del numero di macchine per il numero di lavori. Queste variabili rappresentano gli assegnamenti dei lavori alle macchine. Per assicurarsi che ogni lavoro sia assegnato esattamente ad una e una sola macchina, si stabiliscono dei *vincoli di semiassegnamento*. Due lavori possono essere eseguiti sulla stessa macchina solo se i loro periodi di esecuzione non si sovrappongono. I *vincoli di non sovrapposizione* si ricavano da una matrice di compatibilità tra lavori. Il numero di tali vincoli dipende dal numero di conflitti e ha un limite superiore dipendente dal numero di lavori e di macchine. La funzione obiettivo, infine, è il numero di macchine utilizzate.

Il problema può essere risolto con algoritmi *greedy*. Al momento dell'inizializzazione nessun lavoro è assegnato e tutte le macchine sono scariche. Ad ogni iterazione si seleziona un lavoro secondo un'euristica. Secondo un'altra euristica, si decide a quale macchina assegnare il lavoro considerato, dando priorità alle macchine già in uso. Solitamente l'euristica per la scelta dei lavori consiste nel procedere in ordine di istante di inizio, mentre le macchine vengono considerate secondo un ordine arbitrario ma fissato all'inizio della risoluzione. Questa versione dell'algoritmo porta alla soluzione ottima.

Minimal Makespan Machine Scheduling

Il problema è simile al precedente. In questo caso non è dato un istante di inizio prefissato per i singoli lavori. L'obiettivo è minimizzare il tempo complessivo di esecuzione. Questo tempo corrisponde al tempo di lavoro della macchina più carica. È possibile interpretarlo come un problema di partizione. Le variabili decisionali sono binarie e il loro numero è pari al prodotto del numero di lavori e di macchine. Anche in questo problema vigono vincoli di semiassegnamento.

Per poter rendere il problema in una formulazione riconducibile alla PLI, è necessario definire un maggiorante dei tempi di esecuzione,e prenderlo come funzione obiettivo da minimizzare. Sarebbe altrimenti necessario cambiare funzione obiettivo ogni volta che, nel corso della risoluzione, cambia la macchina più carica in seguito a nuovi assegnamenti. Nella soluzione ottima, il maggiorante diventa esattamente pari al tempo di esecuzione della macchina più carica.

Anche per questo problema esiste una soluzione mediante algoritmo *greedy*. All'inizializzazione tutte le macchine sono scariche e nessun lavoro è assegnato. I lavori vengono ordinati per durata, in ordine non decrescente (SPT) o non crescente (LPT). Ad ogni iterazione il primo lavoro nella lista viene assegnato alla macchina più scarica, e dopodiché si aumenta il tempo di lavoro della macchina sommandovi il tempo di esecuzione del lavoro assegnato. Generalmente l'ordinamento LPT produce soluzioni di qualità migliore.

È possibile valutare l'errore massimo relativo. Esso corrisponde alla differenza tra il tempo della soluzione ideale e il tempo della soluzione reale, divisa per quest'ultima. Le soluzioni con tempo minore di quella ideale sono ovviamente non ammissibili. La soluzione ideale si ottiene rilassando l'integralità dei lavori e dividendo uniformemente il tempo totale di lavoro sul numero di macchine. Si tratta di un limite inferiore e non di una vera soluzione. La stima del caso peggiore, invece, si ottiene distribuendo uniformemente il tempo di tutti i lavori tranne l'ultimo sulle macchine, e poi assegnando l'ultimo lavoro ad una macchina qualsiasi. La stima dell'errore massimo relativo è pari alla differenza tra il tempo massimo e il tempo minimo, diviso il tempo minimo. L'errore massimo relativo è anche riconducibile al numero di macchine meno una, diviso il numero di macchine. Nel caso SPT si tratta di una maggiorazione stretta che vale il doppio del numero di macchine, meno uno.

Problemi di selezione di sottoinsiemi

Sono dati un insieme finito e una famiglia di suoi sottoinsiemi, a ciascuno dei quali è assegnato un costo. Si desidera determinare una sottofamiglia di sottoinsiemi rispettando dei vincoli, minimizzando il costo complessivo. La famiglia è

rappresentata da una matrice che ha per righe gli elementi e per colonne i sottoinsiemi. I valori della matrice sono pari a 1 per indicare l'appartenenza di un elemento ad un sottoinsieme, e zero altrove.

Si rappresenta l'appartenenza del sottoinsieme alla sottofamiglia dei sottoinsiemi tramite variabili decisionali binarie. Il costo della sottofamiglia, che è la funzione obiettivo, è la sommatoria dei costi dei sottoinsiemi scelti.

Esistono tre problemi principali di questo tipo: copertura, partizione e riempimento.

Il *problema di copertura* consiste nell'effettuare una scelta di sottoinsiemi tale da includere tutti gli elementi dell'insieme almeno una volta. Matematicamente, usando la matrice precedentemente definita, il vincolo di copertura impone che esista almeno un sottoinsieme che contenga ogni elemento, ovvero per ogni riga la sommatoria dei valori deve essere maggiore o uguale a uno. Il problema è formulabile come PLI. L'ottimalità della soluzione contenente un certo numero di sottoinsiemi si ottiene verificando che nessuna delle soluzioni possibili usando un sottoinsieme in meno sia ammissibile.

Il *problema di partizione* consiste nel scegliere la sottofamiglia affinché ogni elemento dell'insieme appaia in uno e un solo sottoinsieme. Questo significa che nella matrice ogni riga debba contenere esattamente un elemento di valore unitario e che tutti gli altri siano nulli. La sommatoria di ogni riga deve dunque essere esattamente uguale a uno.

Il *problema di riempimento* consiste nel determinare la famiglia di sottoinsiemi che massimizzi il numero di elementi scelti dell'insieme, evitando ripetizioni. Alcuni elementi potrebbero non essere presenti in nessuno dei sottoinsiemi. La funzione obiettivo è il numero di elementi scelti, e va massimizzata. Il *vincolo di riempimento* impone che per ogni riga della matrice debba esserci al più un uno, dunque la sommatoria di ciascuna riga deve essere minore o uguale a uno.

Per i problemi di copertura esiste una soluzuone *greedy*. La famiglia dei sottoinsiemi è inizializzata vuota. Ad ogni iterazione si seleziona in modo euristico un sottoinsieme non ancora considerato. Se esso copre almeno un elemento ancora scoperto, allora lo si aggiunge alla sottofamiglia. L'algoritmo termina dopo aver esaminato tutti i sottoinsiemi o dopo aver coperto tutti gli elementi. Le possibili euristiche possono essere basate sul costo non decrescente dei sottoinsiemi, sul costo medio non decrescente di copertura di un elemento mediante un sottoinsieme, oppure sui costi unitari attualizzati non decrescenti. Questi ultimi corrispondono al costo medio di copertura di un elemento non incluso, da parte del sottoinsieme considerato. I primi due criteri effettuano un solo ordinamento iniziale, che può essere scorso come una lista ordinata ad ogni iterazione in tempo $O(n \log n)$. Il terzo criterio deve ricostruire l'ordinamento ogni volta che un sottoinsieme viene inserito nella sottofamiglia. Questo criterio è solitamente il migliore perché è in grado di considerare la soluzione corrente come informazione.

La scelta di produzione mutuamente esclusiva su diverse linee o stabilimenti, con vincoli differenti a seconda del caso, non richiede la risoluzione di due problemi separati. È sufficiente utilizzare una variabile binaria che rappresenti la mutua esclusione tra i vincoli.