# Ricerca Operativa

Stefano Mercogliano

May 28, 2020

## Contents

# 1 Introduzione alla ricerca operativa

## 1.1 Introduzione

La Ricerca operativa è una branca matematica che nasce con finalità militari. Essa trova infatti il suo sviluppo prima della seconda guerra mondiale, e i motivi per i quali ciò accadde erano fondamentalmente due: uno relativo all'incremento dei fondi destinati all'utilizzo militare e il secondo invece è da ricercare negli anni 30. Infatti è in questo

periodo che nasce la scienza informatica, e naturalmente la ricerca operativa ne è intrinsecamente legata. Studieremo infatti algoritmi che verranno utilizzati per risolvere problemi decisionali da fare eseguire ad un calcolatore.
Una prima applicazione della ricerca operativa si fa risalire alla fine degli anni 30 quando l'inghilterra creò un sistema radar per controllare il mare del nord. Nonostante tali radar funzionassero egregiamente, ciò che era complicato era l'utilizzo delle informazioni fornite dal radar. Per esempio come far arrivare l'aereo più presto possibile al punto target. Mancava dunque un approccio operativo che riuscisse ad utilizzare al meglio le informazioni fornite.
Dunque fu creato un OR team, ovvero un gruppo di scienziati, il cui scopo era sviluppare metodi quantitativi che permettessero di utilizzare al meglio queste informazioni. Da qui, l'interesse in tale disciplina crebbe nei vari paesi del mondo. Lo scopo era quello di sviluppare metodologie quantitative che permettessero di utilizzare al meglio delle risorse limitate. Sarà questo il paradigma che guiderà i nostri problemi. A partire da risorse limitate, dobbiamo imparare a sfruttarle al meglio per raggiungere determinati obiettivi. La ricerca operativa conobbe la sua massima crescita nel 60, quando i computer ebbero una grande diffusione nelle grosse organizzazioni, e successivamente nelle case di tutti.

## 1.2 Definizioni e Modelli

La **ricerca operativa** si occupa di problemi decisionali, definibili come : *uno o più decisori devono effettuare delle scelte tra diverse alternative possibili, tenendo conto di determinati obiettivi.* Ovviamente nella definizione dell'insieme delle alternative dobbiamo tener conto di quelli che vengono chiamati **vincoli del problema** e che tipicamente sono *le risorse utilizzabili in quantità finita.* Nella valutazione della qualità di una scelta/alternativa, vanno valutate secondo gli obiettivi che i decisori devono tener in conto.
Vediamo un semplice esempio, chiamato problema di addetti mansioni. Ipotizziamo di dover gestire un'organizzazione con 70 lavoratori e 70 lavori da far svolgere. L'ipotesi in cui ci poniamo è che ogni lavoratore può eseguire qualsiasi delle 70 mansioni. Tuttavia le esegue con un costo differente. Per esempio un lavoratore svolge meglio alcune mansioni, con costo minore, piuttosto che altre, che le svolgerà con un costo maggiore. Indichiamo con $C_{ij}$ *il costo con il quale il generico*

*lavoratore i esegue la generica mansione j.* Vogliamo assegnare esattamente una mansione a ciascun lavoratore e l'obiettivo che ci poniamo è che il costo totale sia il più basso possibile.
Dal punto di vista strettamente matematico, tale problema è banale poiché un algoritmo in grado di risolverlo posso descriverlo con un **approccio a forza bruta**. Mi accorgo infatti che l'insieme di tutte le decisioni possibili è un insieme finito.Allora mi basta scegliere una permutazione delle 70 persone, ovvero 70! (tutte le possibili soluzioni); Valuto per ogni combinazione la somma dei costi $C_{ij}$, e alla fine scelgo la permutazione a cui corrisponde il costo più basso. in conclusione, l'algoritmo terminerà e mi restituirà la soluzione ottima, con certezza matematica.
Il problema naturalmente è che 70 fattoriale È dell'ordine di $100^{100}$, un numero enorme! Per intenderci, se ho una macchina che è in grado di ispezionare $10^6$ al secondo, per esplorare l'insieme delle soluzioni avrò bisogno di $10^{94}$ secondi, ovvero $10^{87}$, che è più del tempo che è trascorso dall'origine dei tempi. Naturalmente anche una macchina più veloce di questa, anche 1000 volte, ci impiegherei comunque $10^{84}$ anni, quindi non miglioreremmo significativamente i tempi di esecuzione, neanche ponendoci nell'ipotesi di calcolo in parallelo. Questi problemi sono matematicamente semplici, ma per poterli risolvere in maniera pratica abbiamo bisogno di metodologie e algoritmi che non siano come questo approccio di attacco a forza bruta. Non possiamo infatti pretendere di elencare tutte le soluzioni, ma necessitiamo di algoritmi che trovino la decisione ottimale senza confrontare tutte le possibili decisioni, ma che lo facciano in tempi ragionevoli.
Dunque, torniamo ai problemi di decisione. Uno o più decisori devono prendere una decisione sulla base di uno o più obiettivi che questi decisori si sono posti, per cui se pensiamo ad una classificazione di tutti i problemi di decisione, possiamo classificarli tenendo conto di tre gradi di libertà:

- *Numerodi decisori:*
- *Numerodi di obiettivi:*
- *Grado di incertezza:*

Se guardo al numero di decisori, i problemi di decisione li posso classificare in problemi di un solo decisore, e in problemi di più decisori, dove la scelta di un decisore può influenzare la qualità della scelta fatta da un altro decisore. Nel caso di un solo decisore parlo di **problemi di ottimizzazione**, mentre nel caso in cui abbiamo a che fare

con problemi con più decisori, parliamo di teoria dei giochi. Noi ci concentreremo nel caso di problemi di ottimizzazione.
Vediamo tuttavia un semplice esempio di teoria dei giochi, a titolo informativo, il **dilemma dei due prigionieri**: *Abbiamo due persone, due criminali, che vengono arrestati per aver commesso un reato; tuttavia la polizia non ha prove a carico dei due, e dunque non può condannarli. Le due persone vengono quindi messe in due stanze diverse e interrogate separatamente.* A questo punto vanno stabilite delle regole, dette **regole del gioco**. Tali regole prevedono che :

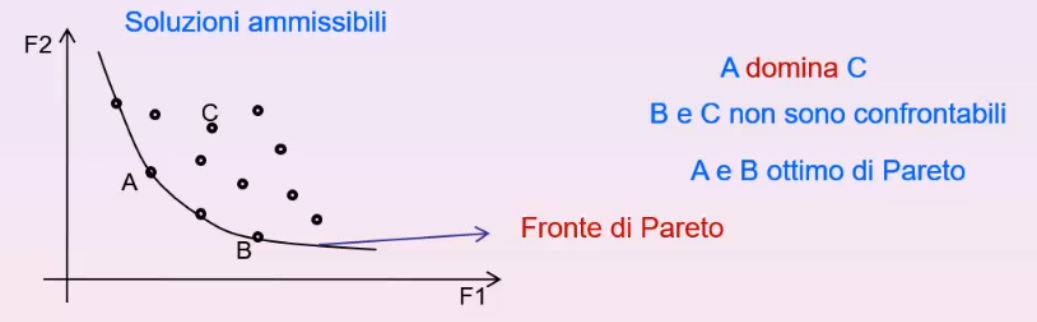
- Se entrambi confessano entrambi sono condannati a 3 anni.
- Se nessuno confessa, non possono essere condannati per un reato commesso, e vengono condannati per un reato minore, e quindi ad 1 anno ciascuno.
- Se uno dei due confessa, e l'altro no, chi ha confessato non viene condannato, mentre l'altro viene condannato a 6 anni di carcere

Queste sono le regole del gioco, che in problemi di ottimizzazione prenderanno nome di vincoli. Questi due decisori, i criminali, devono indipendentemente prendere una decisione, e vogliono prendere quella che gli conviene di più. Potremmo dunque riassumere le regole nella matrice:

$$\begin{pmatrix} (-3,-3) & (0,-6) \\ (-6,0) & (-1,-1) \end{pmatrix}$$

Naturalmente se ci fosse un capobanda, ovvero un unico decisore, ordinerebbe ad entrambi di non confessare, prendendo così entrambi 1 anno a testa, ovvero al quantità minima combinata. Tuttavia la decisione dipende da due decisori, indipendenti. Ovviamente la cosa migliore è fare una scelta tale che nel momento in cui si viene a sapere della decisione dell'altro, la sua scelta rimanga la migliore. La migliore decisione è ovviamente quella di confessare. Dal punto di vista della banda dei due prigionieri, non è l'ottimo, ma è un punto di equilibrio, detto **equilibrio di Nash**. E allora vogliamo decisioni del quale il decisore non debba mai pentirsi, detti appunti, punti di equilibrio.

Anche i problemi di ottimizzazione li possiamo in qualche modo classificare sulla base del secondo grado di libertà che ci eravamo detti, ovvero il numero di obiettivi. E allora parliamo di problemi di ottimizzazione *multiobiettivo e monobiettivo*. Anche qui ci manterremo sulla parte monobiettivo. Vediamo comunque un esempio di risoluzione di




problema di ottimizzazione multiobiettivo.

In uno **scenario multiobiettivo**, ciò che accade è che la qualità di una decisione viene valutata sulla base di due obiettivi, o più di due, diversi tra loro, e spesso in contrasto tra loro. Ipotizziamo dunque di avere una azienda della quale vogliamo programmare la produzione. Nel prendere le decisioni tipicamente il decisore dovrà tener conto di due obiettivi: *un primo è minimizzare il numero di clienti insoddisfatti, ovvero massimizzare la qualità, e al contempo vorrei minimizzare i costi complessivi di produzione.*
Siano questi obiettivi $F1$ ed $F2$, in contrasto tra loro. Immaginiamo di aver rappresentato in questo diagramma cartesiano il rapporto tra $F1$ ed $F2$. Ciascuna scelta è rappresentata da uno di questi punti in grafico.

Se vado a confrontare la soluzione $A$ e quella $C$, chiaramente $A$ è migliore di $C$, o diremo che $A$ domina $C$. questo perché $A$ è caratterizzata da un valore più piccolo, e dunque migliore, della prima funzione $F1$ e anche della seconda $F2$. Il problema però è che se confronto $C$ con $B$, non posso dire con certezza che $B$ sia migliore di $C$, ne il viceversa. Però la soluzione individuata su $B$ ha una caratteristica che $C$ non ha. $B$ e $C$ non sono confrontabili, a meno di dare priorità ad una $F$ piuttosto che ad un'altra. Se guardiamo a $B$ però, esso non è dominato da nessuna soluzione, mentre $C$ invece lo è. Ma allora ci sono *soluzioni non dominate da nessuna*. Queste vengono chiamate **soluzioni ottimali secondo Pareto**. E allora risolvere un problema di ottimizzazione multiobiettivo è trovare il fronte di Pareto, ovvero tutte le soluzioni non dominate.
Ciò è però complicato e si cerca infatti di ricondursi spesso ad un problema monobiettivo, costruendo per esempio una funzione obiettivo $F'$

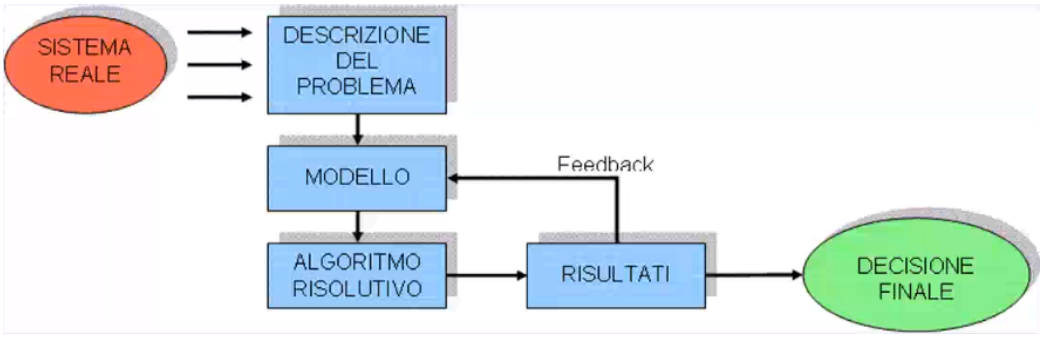
che è combinazione lineare di $F1$ ed $F2$, tramite un coefficiente di priorità. Altra soluzione potrebbe essere scegliere una sola delle due funzioni. Per esempio valuto solo $F1$, e trasformo $F2$ in un vincolo di budget. Ovvero fissato un budget non voglio superare una soglia su $F2$.
In caso di incertezza invece, possiamo avere dei *dati di input stocastici, o matematici*. Nel caso degli addetti mansioni di prima, i dati erano i costi $C_{ij}$. Se di questo costo posso dare una valutazione deterministica, avrò a che fare con quantità deterministica. Naturalmente a volte non posso stabilirlo con certezza, ma posso stabilire solo *"valori medi e varianze"*. E allora parliamo di dati stocastici. Il nostro obiettivo sarà concentrarci su dati a programmazione matematica.

## 1.3 Modelli Matematici

**Un problema di programmazione matematica** è individuato da un vettore $X$, un insieme di variabili di decisione $X1, x2, ....., xn$. Il valore che assegnerò a tale variabili rappresenterà la decisione presa. Avrò poi una funzione $f$, ed un insieme $X$ di soluzioni possibili. Il mio obiettivo è quello di trovare

$$minf(x)x \in X$$

$F$ è detta **funzione obiettivo**, e $X$ è detto **dominio di ammissibilità**. Vogliamo trovare come prima cosa un modo per rappresentare il dominio di ammissibilità. Tipicamente lo rappresentiamo come*insieme di soluzioni di un sistema di disequazioni*, dette **vincoli**. Questi vincoli sono tutti espressi per convenzione con $\leq$ e chiaramente ossiamo sempre ricondurci a questa forma a partire da $\geq$ o $=$ . Se devo massimizzare la funzione obiettivo la posso inoltre portare a minizzare cambiando semplicemente segno.
Dunque mi riporterò sempre ad una forma standard di questo tipo. è allora chiaro che anche questi problemi possono essere classificati sulla base delle caratteristiche di vincoli e funzioni di obiettivi. Il caso più semplice è che sia la funzione obiettivo, sia le funzioni $g(x)$ dei vincoli, siano lineari. Parliamo in questo caso di *problemi di programmazione lineare*. Se invece $f(x)$ e le $g(x)$ non sono tutte lineari parliamo di *problemi di programmazione non lineari*.
Nel nostro caso prenderemo in analisi i problemi di programmazione lineare intera, ovvero nel discreto.Quello che vogliamo adoperare, sarà




un approccio modellistico: A partire dal problema reale, devo effettuarne una descrizione, ovvero quali sono le grandezze importanti dal punto di vista della decisione finale che devo andare a prendere. Ad ognuna di queste grandezze viene associata una variabile di decisione. Successivamente realizzerò un modello, che elaborerò tramite un algoritmo, producendo dei risultati che mi porteranno alla decisione finale.

Naturalmente io devo conoscere il problema da cui parto, ed è chiaro che i possibili problemi da affrontare siano altamente variabili. Inoltre devo sapere cosa ho a disposizione per risolvere il mio modello. Devo conoscere i miei strumenti e le mie risorse. Vediamo come fare costruzione formale di un modello di programmazione matematica. La prima cosa da fare è decidere quali sono le grandezze reali che mi interessano, e a ciascuna devo associare una variabile di decisione, ovvero una *incognita del problema*. Dopodichè devo indivduare i legami logici tra le grandezze in gioco, e ognuno di questi legami devono esser tradotti in vicoli matematici : disequazioni. L'insieme delle soluzioni di questi vincoli mi da le soluzioni ammissibili del problema. Infine devo individuare la funzione vincolo da minimizzare, che mi darà indice di qualità per la decisione rappresentata dai valori delle variabili in gioco.

Facciamone un esempio, ipotizzando di aver già analizzato le grandezze in gioco e i loro rapporti logici. Il problema ha a che fare con la produzione. Devo decidere come produrre dei capi di abbigliamento. Gli oggetti da produrre saranno pantaloni, giacche e camicie.

| | Costi Unitari | Prezzi Unitari |
|---|---|---|
| Pantaloni | 25 euro | 45 euro |
| Giacche | 35 euro | 70 euro |
| Camicie | 20 euro | 35,50 euro |

Per ogni oggetto ho costo unitario e prezzo unitario, ovvero prezzo di vendita. Stiamo tra l'altro ipotizzando che qualunque quantità io creii abbia sempre lo stesso costo e lo stesso prezzo di vendita. Naturalmente all'ingrosso di solito i costi di produzione sono più bassi, il che mi porterebbe ad avere un comportamento non lineare, ma con crescita più lenta all'aumentare del numero di prodotti. Queste semplificazioni sono necessarie per avereunl problema lineare.

Dunque se guardo alla curva dei costi, ovvero all'aumentare dei prodotti, il costo non aumenta più. Tuttavia lo vado ad approssimare con una retta. Posseggo inoltre, come visto in tabella, delle risorse limitate, che voglio usare al meglio. In questo esempio le risorse sono i fattori temporali dei tre reparti : taglio, cucitura e stiratura. Questi 3 reparti hanno una disponibilità massima in ore al giorno, e per ogni unità abbiamo un tempo di produzione per ogni reparto. Le ore giornaliere sono dunque le mie risorse limitate.

| | Taglio | Cucitura | Stiratura |
|---|---|---|---|
| Pantaloni | 10 min | 15 min | 10 min |
| Giacche | 20 min | 20 min | 15 min |
| Camicie | 15 min | 20 min | 20 min |
| Disponibilità Giornaliera | 20 ore | 18 ore | 16 ore |

Quante unità di capi mi conviene produrre ogni giorno senza sforare le mie risorse a disposizione?
Dobbiamo poi definire la funzione obiettivo: vogliamo massimizzare i ricavi? Oppure vogliamo minimizzare i costi di produzione?

Una volta schematizzato il problema, modellarlo diventa estremamente semplicemente. Come prima cosa stabiliamo le variabili del problema, stabilendo quanti numeri mi servono per descrivere una possibile decisione. In questo caso, una mia decisione è rappresentata da 3 numeri, quanti pantaloni, giacche e camicie produrre. Il mio problema presenterà banalmente 3 variabili. Associo le mie tre variabili alle tre grandezze che definiscono le decisioni. $Xp$ è il numero di pantaloni prodotti, $Xg$ le giacche e $Xc$ le camicie. I vincoli del

problema sono disequazioni che dicono che ciascuna variabile non può superare le risorse massime disponibili. (disponibilità del reparto). Nel taglio la consumo $10Xp$ minuti. Allo stesso modo nelle giacche avrò $20xg$, mentre per le camicie $15xc$.

$$\begin{cases} 10Xp + 20Xg + 15Xc \leq 120 \\ 15Xp + 20Xg + 20Xc \leq 1080 \\ 10Xp + 15Xg + 20Xc \leq 96 \end{cases}$$

La funzione obiettivo sarà $25Xp + 35Xg + 20Xc$ se voglio minimizzare i costi. Se voglio massimizzare il ricavo voglio il max di $45Xp+50Xg+35,5Xc$. Per massimizzare il profitto : $20Xp+35Xg+15,50Xc$. Diciamo di voler massimizzare il profitto, scegliendo così la terza funzione obiettivo. Una qualsiasi possibile decisione soddisfa i vincoli del mio problema e viceversa : tutte le mie possibili soluzioni rappresenta una decisione ammissibile. Esistono soluzioni di questo sistema che non rappresentano nessuna possibile soluzione nel mondo reale? Chiaramente si, le soluzioni negative. Quindi devo aggiungere i vincoli di non negatività, ovvero porre tutte le variabili del problema $\geq 0$.
Nel minimizzare i costi complessivi potrei pensare di non produrre niente per minizzare i costi, il che non è logico ovviamente. Negli altri casi vedremo un algoritmo per ottenere la soluzione ottima.

# 2 problemi di ottimizzazione lineari

Ipotizziamo che un'industria debba costruire un silos, ovvero un serbatoio di forma cilindrica. Tale cilindro deve essere posto internamente ad un magazzino con un pavimento rettangolare di $20x10$ metri e ha un tetto aspiovente con lato lungo 10 metri e altezza massima di 5 metri, con un minimo di 3. Per costruire tale silos è necessario utilizzare del materiale plastico, tagliato e modellato a piacere. Tuttavia abbiamo a disposizione un massimo di $200m^2$ di tale materiale.
Come deve esser fatto questo cilindro in maniera tale da entrare nel magazzino e rispettare la specifica, contenendo la massima quantità di liquido possibile?

Come prima cosa mi devo chiedere di quanti numeri io necessiti per descrivere il mio problema. Naturalmente in questo caso dovrò tener conto delle dimensioni del cilindro, ovvero il raggio/diametro

e l'altezza. Stabiliamo dunque due variabili decisionali : $X$ sarà la lunghezza del raggio di base, mentre $Y$ sarà l'altezza del cilindro. A questo punto bisogna definire una funzione obiettivo, che in tal caso sarà proprio il volume del cilindro. Perciò il vostro obiettivo sarà massimizzare la funzione volume:

$$maxV(x) = \pi X^2 Y$$

Il nostro dunque è un problema sicuramente non lineare, in quanto cubico. Devo ora preoccuparmi di inserire i vincoli, poiché non posso pretendere di avere $X$ e $Y$ tendenti ad infinito. Come prima cosa il cilindro deve poggiare sulla base del mio magazzino e dunque la sua base deve entrare nello spazio del pavimento.
Il primo vincolo da imporre è che il diametro sia minore o uguale del lato più piccolo della base del magazzino:

$$2X \leq 10$$

Sappiamo inoltre che per costruire questo cilindro devo utilizzare del materiale plastico, naturalmente limitato, con una fornitura di $200m^2$. Il secondo vincolo sarà dunque relativo alla superficie del cilindro:

$$2\pi X^2 + 2\pi XY \leq 200$$

L'ultimo vincolo è relativo all'altezza. Questa adesso è in funzione del diametro, in quanto se il cilindro sarà abbastanza largo dovrà avere una altezza di non più di 3 metri. Tanto più stretta è la base del cilindro e tanto maggiore potrà essere l'altezza, fino a $5m$.

$$Y <= 5 - \frac{2}{5}X$$

Notiamo che qualsiasi soluzione deve essere implementabile. E allora vuol dire che devo aggiungere i vincoli di fisica realizzabilità. Dunque sia $X$ che $Y$ devono essere maggiori di 0.

$$X \geq 0$$

$$Y \geq 0.$$

Impostato il modello possiamo utilizzare un qualsiasi strumento di risoluzione per ottenere la soluzione al nostro problema. Un tool per risolvere problemi di programmazione matematica semplici, è microsoft excel. Utilizzando un risolutore del problema, siamo in grado di determinare i valori di $X = 4.22$ e $Y = 3.31$, con un volume di 185.59 .

## 2.1 Programmazione lineare

Un problema di programmazione matematica in cui sia la funzione obiettivo che i vincoli sono lineari viene detto *problema di programmazione lineare* (PL). Un problema di PL di $n$ variabili ed $m$ vincoli può essere rappresentato in **forma canonica** come:

$$\min c_1 x_1 + \cdots + c_n x_n$$

$$\begin{array}{cccc} a_{1,1}x_1 & a_{1,2}x_2 & \cdots & a_{1,n}x_n \leq b1 \\ a_{2,1}x_1 & a_{2,2}x_2 & \cdots & a_{2,n}x_n \leq b2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1}x_1 & a_{m,2}x_2 & \cdots & a_{m,n}x_n \leq bm \end{array}$$

Possiamo riscrivere il problema in forma vettoriale utilizzando i vettori $c$ e $a_i$, come:

$$\min c^T x$$
$$a_i^T x \leq b_i$$

Utilizzando la matrice $A$ in dimensione m x n il problema può esser riscritto come

$$\min c^T x$$
$$Ax \leq b$$

Tali problemi sono facili, poiché esistono algoritmi con complessità polinomiale in grado di risolverli, anche con un numero di variabili $n$ molto elevato. Tra gli algoritmi presenti vedremo *l'algoritmo del simplesso*, che nonostante abbia una complessità esponenziale, si comporta meglio di alcuni algoritmi a complessità polinomiale, che sono quindi relegati solo ad una importanza teorica più che pratica.

## 2.2 problemi di allocazione ottima di risorse

In un problema del genere definiamo un insieme di risorse $R1 \cdots Rm$, dove ogni risorsa è disponibile in una quantità prefissata $b_i$. È presente poi un insieme di prodotti $P \cdots Pn$, a ciascuno del quale è associato un profitto $c_i$. È quindi possibile definire una matrice avente sulle righe $R_i$ e sulle colonne $P_j$, detta *matrice dei tassi di assorbimento*,

che ha come generico elemento $a_{ij}$, che indica la quantità di risorsa i necessaria a produrre una unità di prodotto j.Massimizzare allora vuol dire trovare:

$$\max c^T x$$

$$Ax \leq b$$

$$x \geq 0$$

dove $c$ è il vettore dei profitti e $b$ il vettore delle quantità, mentre $A$ è la *matrice dei tassi di assorbimento.*

Facciamone un esempio a riguardo. Consideriamo di avere una industria dolciaria che procude quattro tipi di confezioni di cioccolatini, $T1, T2, T3, T4$. In figura possiamo vedere la matrice, seppur trasposta, di assorbimento.

| | Latte | Fondente | Bianco |
|---|---|---|---|
| T1 | 10 | 20 | 10 |
| T2 | 20 | 5 | 5 |
| T3 | 5 | 20 | 5 |
| T4 | 5 | 5 | 20 |

Di seguito invece possiamo vedere le quantità che abbiamo a disposizione in totale e infine il prezzo di vendita.

| | Latte | Fondente | Bianco |
|---|---|---|---|
| Quantità | 5000 | 7000 | 3500 |

| | T1 | T2 | T3 | T4 |
|---|---|---|---|---|
| Prezzo | 20 | 20 | 25 | 25 |

Dunque possiamo ora formalizzare il problema. Con $X_i$ posso indicare le confezioni, mentre $P_i$ è il prezzo di vendita di ciascuna confezione e $Q_j$ le quantità disponibili per tipo di cioccolatino, e infine $r_{ij}$ è il numero di cioccolatini di tipo j per confezioni di tipo $T_i$. Possiamo dunque schematizzare il nostro problema secondo il seguente modello matematico.

- $x_1$ confezioni prodotte del tipo $T1$
- $x_2$ confezioni prodotte del tipo $T2$.

- $x_3$ confezioni prodotte del tipo $T3$
- $x_4$ confezioni prodotte del tipo $T4$

$$\max 20x_1 + 20x_2 + 25x_3 + 25x_4$$

$$\begin{gathered} 10x_1 + 20x_2 + 5x_3 + 5x_4 \leq 5000 \\ 10x_1 + 5x_2 + 5x_3 + 5x_4 \leq 7000 \\ 10x_1 + 5x_2 + 5x_3 + 20x_4 \leq 3500 \end{gathered}$$

$$x_i \geq 0$$

La soluzione può essere trovata sempre con il risolutore di excel, che ci rivela che il numero di confezioni prodotte per poter massimizzare il profitto è : $1, 160, 294, 61$, con un profitto massimo di 12.095. Soluzione analoga può essere ovviamente ottenuta con altri tool di risoluzione, come ad esempio Xpress IVE.

Vediamo ora una *variante* di problema di allocazione ottima di risorse, che tenga in conto anche il fattore **temporale**. In questo caso prendiamo in considerazione una produzione *multiperiodo*. Lo scopo di questo esempio è capire come usare al meglio delle risorse limitate in ciascuno dei sottoperiodo in cui è diviso l'orizzonte temporale relativo al problema; ciò varia sensibilmente il problema di allocazione classicamente definito in precedenza.

Vogliamo produrre due tipi di pneumatici $A$ e $B$, ed è necessario prendere una decisione su *base trimestrale* della produzione. Per i successivi 3 mesi si hanno a disposizione conoscenze circa la richiesta dei due prodotti $A$ e $B$. Per ognuno dei tre mesi e per ognuno dei prodotti siamo a conoscenza dunque delle richieste. Ad ottobre per esempio saranno richieste 4500 unità di $A$ e 3000 di $B$ e così via.

| | Tipo A | Tipo B |
|---|---|---|
| Ottobre | 4500 | 3000 |
| Novembre | 5000 | 3500 |
| Dicembre | 2000 | 8000 |

Per la produzione di questi prodotti la fabbrica ha a disposizione due macchinari $M1$ ed $M2$. Sia $A$ che $B$ possono esser prodotti su entrambe le macchine, tuttavia il tempo di produzione pneumatico/ora

rispetto alle macchine, per prodotto, è differente.

| | M1 | M2 |
|---|---|---|
| Tipo A | 0.10 | 0.12 |
| Tipo B | 0.18 | 10.15 |

Ogni macchina possiede inoltre un massimo tempo di esecuzione mensile, visibile in tabella

| | M1 | M2 |
|---|---|---|
| Ottobre | 1000 | 800 |
| Novembre | 1500 | 1800 |
| Dicembre | 600 | 1100 |

Inoltre il costo di lavorazione per ora è di 6 euro, per entrambe le macchine. È chiaramente presente anche un costo per la materia prima, che è di 2.5 euro per il prodotto di tipo $A$ e 4 per quello di tipo $B$ . La produzione in eccesso deve essere immagazzinata per il mese successivo,con conseguenti costi di immagazzinamento sono di 0.35 euro per ciascun pneumatico. Assumiamo che all'inizio del trimestre i magazzini siano vuoti, e che alla fine del trimestre i magazzini debbano essere vuoti. Dunque a questo punto proviamo a realizzare un modello per la pianificazione trimestrale, espresso in programmazione lineare, con l'obiettivo di *minimizzare il costo complessivo*.

Stabiliamo l'insieme di variabili, procedendo poi con la scrittura della funzione obiettivo e dei vincoli. Naturalmente la scelta delle variabili *non è univoca*, in quanto è possibile identificare più insiemi di variabili differenti. Ciò che invece è univoco è il numero minimo di variabili da utilizzare per modellare interamente il problema.

Ciò che interessa sapere, e che al momento non è specificato, è il numero di pneumatici di tipo $A$, o $B$, prodotto su macchina $Mi$, per ogni mese. Un insieme di variabili che possono essere utilizzate per formulare il problema possono essere definite come $X_{ijk}$ dove l'indice$i$ varia sull'insieme dei prodotti, l'indice $j$ varia sui mesi, e l'indice $k$ varia sulle macchine utilizzate. Tale insieme è sufficiente, ma potrebbe tornar comodo l'impiego delle variabili $S_{ij}$, ovvero la quantità di prodotto $i$ conservata alla fine del mese $j$. Tuttavia questa informazione potrebbe

essere direttamente derivata dai valori di $X$.

Per quanto ne concerne la funzione obiettivo, sarà necessario fare uso del numero di ore di utilizzo delle macchine, e il loro costo per ora.

$$\min 6 \begin{pmatrix} 0.10(x_{A,ott,M1} + x_{A,nov,M1} + x_{A,dic,M1})+ \\ 0.12(x_{A,ott,M2} + x_{A,nov,M2} + x_{A,dic,M2})+ \\ 0.18(x_{B,ott,M1} + x_{B,nov,M1} + x_{B,dic,M1})+ \\ 0.15(x_{B,ott,M2} + x_{B,nov,M2} + x_{B,dic,M2})+ \end{pmatrix} + 2.5 \begin{pmatrix} x_{A,ott,M1} + x_{A,nov,M1} + x_{A,dic,M1}+ \\ x_{A,ott,M2} + x_{A,nov,M2} + x_{A,dic,M2} \end{pmatrix}$$

$$+4 \begin{pmatrix} x_{B,ott,M1} + x_{B,nov,M1} + x_{B,dic,M1}+ \\ x_{B,ott,M2} + x_{B,nov,M2} + x_{B,dic,M2} \end{pmatrix} + 0.35 \begin{pmatrix} S_{A,ott} + S_{A,nov} + S_{A,dic}+ \\ S_{B,ott} + S_{B,nov} + S_{B,dic} \end{pmatrix}$$

Il primo termine di costo è relativo alla quantità di prodotto realizzato per macchina; il secondo termine è il costo di produzione per pneumatici di tipo $A$, il terzo per quelli di tipo $B$, mentre l'ultimo termini è relativo al costo di immagazzinamento.

A questo punto è necessario solo stabilire i vincoli del problemi. È fondamentale produrre nel mese di ottobre almeno 4500 unità di $A$ e 3000 di $B$, e così via per gli altri mesi, come specificato nelle tabelle precedentemente presentate. Tenendo conto anche dei materiali immagazzinati mese per mese, otteniamo :

$$X_{A,ott,M1} + X_{A,ott,M2} = 4500 + S_{A,ott}$$
$$X_{A,nov,M1} + X_{A,nov,M2} + S_{A,ott} = 5000 + S_{A,nov}$$
$$X_{A,dic,M1} + X_{A,dic,M2} + S_{A,nov} = 2000$$
$$X_{B,ott,M1} + X_{B,ott,M2} = 3000 + S_{B,ott}$$
$$X_{B,nov,M1} + X_{B,nov,M2} + S_{B,ott} = 4500 + S_{B,nov}$$
$$X_{B,dic,M1} + X_{B,dic,M2} + S_{B,nov} = 8000$$

Manca ancora l'insieme di vincoli relativi alla disponibilità massima di risorsa $M1$ ed $M2$ nei tre mesi segnalati. Nel mese di ottobre $M1$ può esser utilizzata per 1000 ore, mentre $M2$ per 800, e così via per gli altri mesi. L'utilizzo della macchina $Mi$ è dovuto alla produzione di $A$ o $B$, naturalmente. Continuando così il discorso, ne discende che

$$0.10X_{A,ott,M1} + 0.18X_{B,ott,M1} \leq 1000$$
$$0.10X_{A,nov,M1} + 0.18X_{B,nov,M1} \leq 1500$$
$$0.10X_{A,dic,M1} + 0.18X_{B,dic,M1} \leq 600$$
$$0.12X_{A,ott,M2} + 0.15X_{B,ott,M2} \leq 800$$
$$0.12X_{A,nov,M2} + 0.15X_{B,nov,M2} \leq 1800$$
$$0.12X_{A,dic,M2} + 0.15X_{B,dic,M2} \leq 1100$$

Da queste ultime relazioni siamo in grado di notare la dipendenza tra le variabili di immagazzinamento $S_{i,j}$ con le generiche $X_{ijk}$. Tuttavia i calcoli risultano essere in questa maniera più agevol e leggibili.

## 2.3 problemi di miscelazione

Un problema di miscelazione nella sua forma generale è definito da un insieme di *sostanze da miscelare* $S1, S2 \cdots Sn$ e di ciascuna sostanza $S_i$ si conosce il costo unitario $c_i$.
Abiamo poi un insieme di *componenti utili* dette $C1, C2 \cdots Cm$ e ciascun componente $C_i$ deve esser presente nella miscela finale in una quantità almeno pari a $b_i$. Dopodichè definiamo la *matrice di Componenti-Sostanze*, che ha tante righe quanti sono le componenti C, tante colonne quanto sono le sostanze S, con il generico elemento $a_{ij}$ indicante la *quantità di componente* $i$ presente in una unità di sostanza $j$.

$$\min c^T x$$
$$Ax \geq b$$
$$x \geq 0$$

Visualizziamo un esempio; il problema della dieta.

Ipotizziamo di avere un atleta, che in prossimità di una gara, deve perdere peso senza perdere massa muscolare. Dunque il suo regime alimentare deve prevedere l'assunzione di carni, legumi ,pasta ed olio. Le sostanze da mescolare per creare la dieta sono dunque carne, legumi, pasta ed olio e le componenti di cui tener presente sono Grassi, Carboidrati, Proteine e Calorie.

| | Carne | Legumi | Pasta | Olio | Richiesta Giornaliera |
|---|---|---|---|---|---|
| Grassi | 2.6 | 1.5 | 1.5 | 100 | 30 |
| Carboidrati | 0 | 60.7 | 74.7 | 0 | 90 |
| Proteine | 20.2 | 22.3 | 13 | 0 | 60 |
| Calorie | 110 | 337 | 371 | 884 | |

Questi numeri nella tabella indicano il contenuto dei macronutrienti per ciascun elemento. Quale sarà il regime dietetico giornaliero? Associamo una variabile a ciascuna delle sostanze Si. Avrò una variabile $x_1$ rappresentante la carne, $x_2$ i legumi, $x_3$ la quantità di pasta, e $x_4$ la quantità di olio
Volendo minimizzare le calorie, la funzione obiettivo sarà:

$$110x_1 + 337x_2 + 371x_3 + 884x_4.$$

Avrò un vincolo ovviamente per ogni sostanza, in quanto abbiamo un numero limitato di quantitativo da assumere. Possiamo allora impostare il nostro modello matematico, come segue:

$$\min 110x_1 + 337x_2 + 371x_3 + 884x_4.$$

$$\begin{gathered} 2.6x_1 + 1.5x_2 + 1.5x_3 + 100x_4 \geq 30 \\ 60.7x_2 + 74.7x_3 \geq 90 \\ 20.2x_1 + 22.3x_2 + 13x_3 \geq 60 \\ x_1, x_2, x_3, x_4 \geq 0 \end{gathered}$$

Introduciamo un ulteriore **esempio:**

Una raffineria produce quattro tipi di benzine grezze ($B_1, B_2, B_3, B_4$) e le miscela allo scopo di ottenere due carburanti diversi, $C_1$ e $C_2$. Per ogni benzina grezza abbiamo il numero di *ottani*, gli *ettolitri richiesti* e il *costo*. È inoltre presente una tabella indicante il numero minimo di ottani e prezzo per tipo di carburante.
Il mercato inoltre è in grado di assorbire non più di 25000 ettolitri al giorno del carburante $C_1$, mentre richiede almeno 10000 ettolitri al giorno di$C_2$. Se le benzine grezze non vengono usati per miscelare i carburanti, possono essere rivendute ad un prezzo di 270 euro per ettolitro se il numero di ottani è inferiore a 80, e a 250 altrimenti. Per esempio $B_1$ e $B_4$ possono esser vendute a 280 euro per ettolitro.

| | B1 | B2 | B3 | B4 |
|---|---|---|---|---|
| n. di ottani | 90 | 73 | 79 | 86 |
| ettolitri | 3500 | 6000 | 4500 | 5200 |
| costo | 260 | 210 | 190 | 220 |

| | C1 | C2 |
|---|---|---|
| Min di n. di ottani | 80 | 85 |
| Prezzo | 350 | 520 |

Occorre inoltre pianificare la produzione giornaliera della raffineria, cioè le quantità e le composizioni delle due miscele, massimizzando il profitto ottenuto dalla vendita dei prodotti. Assumiamo naturalmente che il numero di ottani di ciascuna miscela dipenda linearmente dalle gradazioni delle benzine componenti. Ciò significa che in caso di miscela di $B_1$ e $B_2$, la miscela risultante avrà ottani pari ad 88, ovvero la media pesata.

L'obiettivo è massimizzare il profitto ottenuto dalla vendita dei prodotti, ovvero le miscele, e le 4 benzine.
Stabiliamo comeprima cosa le variabili decisionali del problema.Sicuramente siamo interessati a capire quante benzine vogliamo produrre, e quante parti di queste utilizzare per creare le miscele $C_1$ e $C_2$. Potremo allora definire 4 variabili, una per ogni benzina grezza, e altre 8 indicanti ,per ciascuna benzina grezza, quanta ne viene utilizzata per le due miscele. In questa maniera siamo in grado di descrivere completamente la pianificazione della produzione

la quantità di benzina grezza di tipo $B_i$ possiamo modellarla attraverso una variabile $y_i$ con $i = 1 \cdots 4$, mentre invece la quantità di benzina grezza di tipo $B_i$ presente nella miscela di tipo $C_j$, è possibile modellarla attraverso la variabile a doppio indice $x_{i,k}$ con $i = 1 \cdots 4 \quad j = 1, 2$

La funzione obiettivo in questo caso consiste nel *massimizzare* il profitto, e dunque avrò una *componente di ricavo*, ottenuto dalla vendita sia delle benzine grezze, che delle due miscele, meno una *componente di costo* dovuto alla produzione delle benzine grezze.

$$\max \begin{pmatrix} 280(y_1 - x_{11} - x_{12}) + \\ 250(y_2 - x_{21} - x_{22}) + \\ 250(y_3 - x_{31} - x_{32}) + \\ 250(y_4 - x_{41} - x_{42}) \end{pmatrix} + \begin{pmatrix} 350(x_{11} + x_{12} + x_{31} + x_{41}) + \\ 520(x_{21} + x_{22} + x_{32} + x_{42}) \end{pmatrix} - \begin{pmatrix} 260y_1 + 220y_2 + \\ 290y_3 + 220y_4 \end{pmatrix}$$

Per quanto ne concerne i *vincoli*, dobbiamo ricordare di avere una quantità massima di benzina grezza producibile. Inoltre è anche presente una informazione di richiesta del mercato, conseguentemetne all quale $C_1$ può esser prodotta al massimo per 25000 unità, mentre invece di $C_2$ è necessario averne almeno 10000. Infine è presente un numero minimo di ottani per benzina grezza. Inoltre il numero di ottani delle miscele può essere ottenuto come *media pesata* del numero di ottani delle benzine grezze.

Vincoli di mercato:

$$x_{11} + x_{21} + x_{31} + x_{41} \leq 25000$$
$$x_{12} + x_{22} + x_{32} + x_{42} \geq 10000$$

Vincoli sulle quantità massima di benzine:

$$y_1 \leq 3500 \qquad y_3 \leq 4500$$
$$y_2 \leq 6000 \qquad y_4 \leq 5200$$

Vincolo sul numero minimo di ottani:

$$10x_{11} - 7x_{21} - x_{31} + 6x_{41} \geq 0$$
$$5x_{12} - 12x_{22} - 6x_{32} + x_{42} \geq 0$$

Relazione tra le variabili y ed x:

$$y_1 \geq x_{11} + x_{12} \qquad y_2 \geq x_{21} + x_{22}$$
$$y_3 \geq x_{31} + x_{32} \qquad y_4 \geq x_{41} + x_{42}$$

(**NB vincoli sul numero minimo di ottani**: sono divisi per la somma degli $x_{ij}$ e imposti $\leq$ di 80. Moltiplico e faccio le sottrazioni, così ottengo la prima riga, e così via)

## 2.4 problemi di trasporto

Un problema di trasporto tipicamente tratta di un prodotto generato in *m stazioni di origine* che deve essere trasportato averso *n stazioni*

*di destinazione*, dove dovrà essere utilizzato. Per ogni nodo di origine abbiamo una *disponibilità massima* di merce $B_i$ dalla stazione origine i-esima. Allo stesso modo per ogni stazione destinazione abbiamo una richiesta $D_i$ per ogni stazione destinazione i-esima.

Possiamo così definire una *matrice dei costi di trasporto*, dove i coefficienti rappresentano il costo di trasposto di una unità di merce dall'origine i alla destinazione j.

Dove $O_i$ rappresenta la stazione origine, mentre $D_i$ rappresenta la stazione destinazione.

$$\begin{array}{cccccc} & D_1 & D_2 & \cdots & D_n \\ O_1 & C_{1,1} & C_{1,2} & \cdots & C_{1,n} \\ O_2 & C_{2,1} & C_{2,2} & \cdots & C_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \\ O_m & C_{m,1} & C_{m,2} & \cdots & C_{m,n} \end{array}$$

Voglio minimizzare il costo di trasporto. Questo problema può essere rappresentato in maniera molto intuitiva utilizzando un *grafo bipartito*.

Considero quindi un insieme di $m$ nodi, che rappresentano le origini di cui sopra. A ciascun nodo è associata una disponibilità $B_i$, come detto prima. Allo stesso modo esiste un altro insieme di nodi, quello destinazione, ad ognuno dei quali viene associata una richiesta. Alla fine si possono effettuare tutti i collegamenti e possiamo associare ad ogni arco la quantità di merce che va da $O_i$ a $D_j$, che chiameremo $X_{ij}$. Possiamo dunque stabilire la seguente funzione obiettivo:

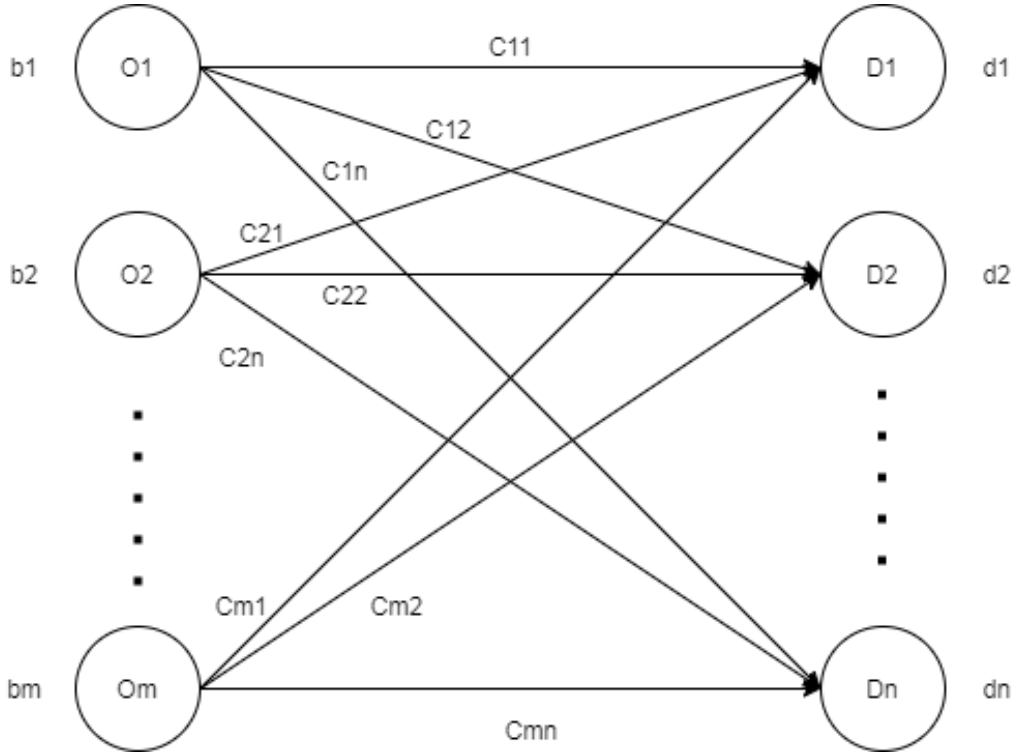
$$\sum \sum C_{ij} X_{ij}$$

Per quanto ne concerne i vincoli, la quantità di merce uscente dal generico nodo $O_i$ non deve eccedere la quantità $B_i$. allora vale che, il flusso uscente non può eccedere le quantità disponibili, e dunque:

$$\sum_{j=1}^{n} X_{ij} \leq B_i$$

$$i = 1 \cdots m$$

Lo stesso ragionamento va fatto sui nodi destinazione.in questo caso però, è necessario che la quantità di merce entrante sia almeno


b1
O1
C11
D1
d1
C12
C1n
C21
b2
O2
D2
d2
C22
C2n
Cm1
Cm2
bm
Om
Dn
dn
Cmn

pari a quella richiesta. Svolgo dunque la sommatoria su tutti gli archi entranti in j . vale allora che

$$\sum_{i=1}^{m} X_{ij} \geq D_j$$

$$j = 1 \cdots n$$

E infine naturalmente è necessario che le variabil $X_{ij}$, ovvero la quantità di merce trasportata dall'origine i al magazzino j, siano maggiori di 0.

$$Xij \geq 0$$

$$\forall i, j \in n, m$$

Naturalmente se $\sum B_i \leq \sum d_j$ allora il problema è detto *inammissibile*, poiché ci sono meno risorse disponibili rispetto alle richieste. In caso contrario il sistema ammetterà infinite soluzioni e quindi potremo trovare la migliore.

Introduciamo ora un ulteriore **esempio di modellazione circa il problema di trasporto**.

Un'industria produce un preparato chimico usando due impianti di produzione $I1$ ed $I2$. Da questi impianti tutto il preparato chimico prodotto viene trasportato in due magazzini $M1$ ed $M2$ distinti tra loro. in questi magazzini una parte viene venduta direttamente all'ingrosso, mentre il resto viene spedito verso dei centri di distribuzione $D1, D2, D3, D4$. Questi centri necessitano rispettivamente di almeno $150, 190, 220, 170$ quintali di preparato chimico, che vendono rispettivamente a $300, 280, 200, 270$ euro al quintale.

data la consegna del problema, possiamo cominciare a rappresentare in forma tabulare sia i costi necessari per trasportare un quintale di preparato da ciascun impianto a ciascun magazzino, sia i costi necessari per trasportare un quintale di preparato da ciascun magazzino a ciascun centro di distribuzione.

| | M1 | M1 |
|---|---|---|
| I1 | 21 | 25 |
| I2 | 27 | 22 |

| | D1 | D2 | D3 | D4 |
|---|---|---|---|---|
| M1 | 33 | 31 | 36 | 30 |
| M2 | 27 | 30 | 28 | 31 |

Inoltre, l'impianto di produzione $I1$ può fabbricare al più 3000 quintali di preparato, mentre $I2$ ne può fabbricare al più 2000. Per quanto ne concerne i prezzi di vendita all'ingrosso, questi sono rispettivamente 150 euro al quintale per $M1$, e 170 per $M2$, con una richiesta di vendita all'ingrosso di almeno 500 quintali. è inoltre fondamentale che non ci siano rimanenze in magazzino, e che tutto il preparato venga venduto all'ingrosso e smistato nei centri di distribuzione.

Un altro modo per rappresentare il nostro modello, può essere fatto attraverso un *grafo a livelli*. Tale grafo mostra come nodi gli impianti di produzione, i magazzini $Mi$ e le stazioni di distribuzione $Di$, mentre presenta sugli archi il costo necessario a trasportare quintali di materiali da una stazione ad un'altra. è dunque necessario, al fine di mantenere coerenza con il modello, modellare la vendita all'ingrosso nei magazzini, come un trasporto a costo zero verso delle nuove stazioni, che chiamereo $Z1$ e $Z2$. Per ogni stazione è inoltre specificato il prezzo di vendita al quintale, la richiesta minima e, nel caso degli impianti, la disponibilità massima.

L'obiettivo del modello è *massimizzare il profitto totale*, ovvero il ricavo totale ottenuto dalla vendita del materiale, meno i costi di trasporto. Cerchiamo adesso di formulare il nostro problema mediante tecniche di programmazione lineare. Per prima cosa è necessario stabilire l'insieme delle variabili da utilizzare per poter poi definire funzione obiettivo e vincoli.
*L'insieme delle variabili* deve essere tale che una volta assegnati dei valori a ciascuna variabile venga univocamente determinata una possibile soluzione del problema. Potrei associare ad ogni arco del grafo una variabile. Se utilizzassi queste come variabili, ovvero il numero di prodotti che attraversano un arco, potrei stabilire le decisioni del problemai. una possibile soluzione infatti deve stabilire quanto deve esser prodotto in un generico impianto di produzione $Ii$, che sarà uguale alla somma dei flussi uscenti.

Dunque i flussi in ingresso ai magazzini sono indicativi di quanto sia stato prodotto negli impianti $Ii$. È poi necessario stabilire quanto deve esser venduto su $Z1$ e $Z2$ e a quel punto bisogna decidere come

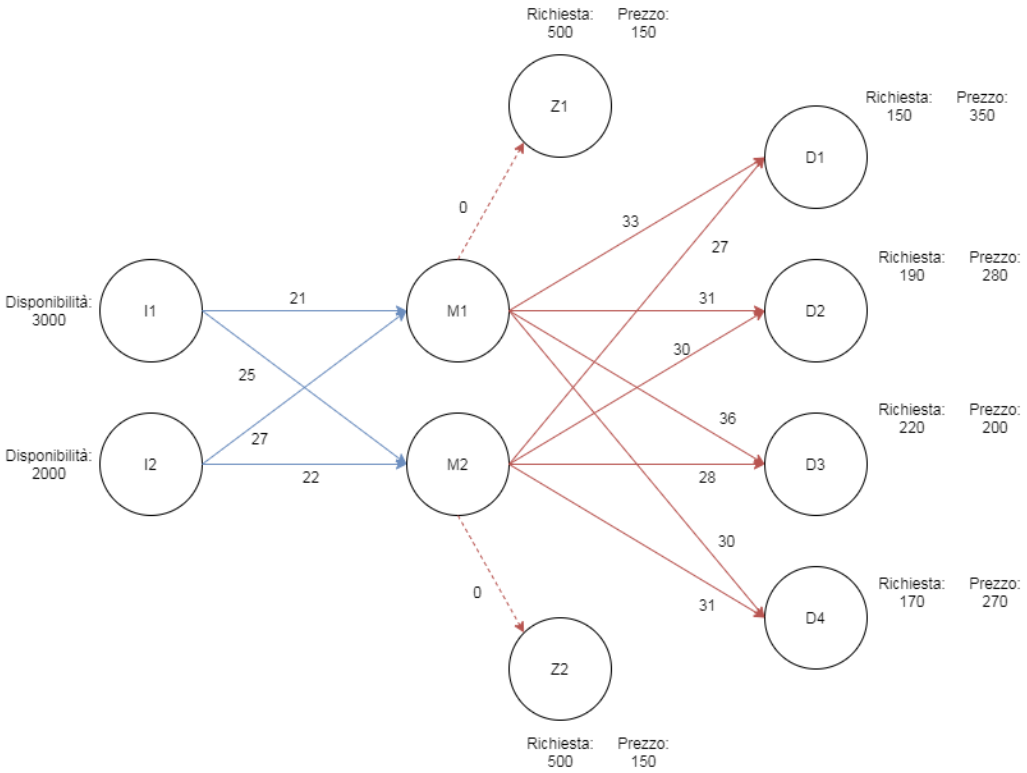



Figure 1: Grafo a livelli

trasportare il materiale nei centri $Di$ a partire da $Mi$, e ciò anche è conoscibile tramite le informazioni degli archi uscenti da $Mi$.

In conclusione possiamo utilizzare come insieme di variabili l'insieme che corrisponde agli archi del grafo. Stabiliamo quindi le variabili decisionali $X_{ij}$, associate agli archi del primo livello (impianti - magazzini), ed $Y_{ij}$ invece gli archi del secondo livello, ovvero i collegamenti magazzini-distribuzioni. Avremo allora 4 variabili sul primo livello e 8 sul secondo. Mancano da introdurre però le variabili associati alla vendita all'ingrosso, che chiameremo $Z_j$.

La funzione obiettivo dunque sarà la massimizzazione della differenza tra profitti e costi.

$$\max[350(Y_{1,1}+Y_{2,1})+280(Y_{1,2}+Y_{2,2})+200(Y_{1,3}+Y_{2,3})+270(Y_{1,4}+Y_{2,4})+$$

$$150Z_1 + 170Z_2 - (25X_{1,1} + 25X_{1,2} + 27X_{2,1} + 22X_{2,2})-$$

$$(33Y_{1,1} + 31Y_{1,2} + 36Y_{1,3} + 30Y_{1,4} + 27Y_{2,1} + 30Y_{2,2} + 28Y_{2,3} + 31Y_{2,4})]$$

I profitti sono dati dalla vendita del materiale nei centri $Di$ e nei magazzini $Mi$, mentre invece i costi sono i costi di trasporto, posti trascurabili quelli di produzione. Per quanto ne concerne i vincoli, è chiaro che per ogni impianto di produzione esiste una quantità massima di prodotto. Dunque ciò che viene inviato verso i magazzini $Mi$ non deve eccedere la quantità massima possibile.

$$X_{1,1} + X_{1,2} \leq 3000$$

$$X_{2,1} + X_{2,2} \leq 2000.$$

Allo stesso modo abbiamo ipotizzato che non deve rimanere giacenza nei magazzini $Mi$. Sul nodo $M1$ allora il flusso entrante deve essere uguale al flusso uscente.Introduciamo così il vincolo detto di *conservazione di flusso.*

$$X_{1,1} + X_{2,1} = Z_1 + \sum Y_{1,j}$$

$$X_{1,2} + X2,2 = Z_2 + \sum Y_{2,j}$$

$$Z_1 \geq 500$$

$$Z_2 \geq 500$$

Infine vanno fissati i vincoli su $Z_i$, in quanto è richiesta una minima quantità da vendere all'ingrosso. Alla stessa maniera in $D$ è richiesta una minima quantità $Qi$ di prodotto. Il flusso entrante in $Di$ deve essere dunque maggiore o uguale di $Qi$, la quantità minima richiesta in $Di$. È chiaro che le variabili di flusso$X, Y, Z$ devono essere maggiori o uguali di zero. Una volta impostato il modello, è possibile utilizzare, come sempre, un risolutore, come Xpress.

## 2.5 Modelli risolutivi di programmazione lineare

Come già accennato in precedenza, problemi lineari sono particolarmente semplici da risolvere anche all'aumentare del numero di variabili, cosa non vera per problemi non lineari, in base all quale non linearità, tali problemi diventano intrattabili all'aumentare delle variabili. Cercheremo di analizzare alcuni modelli di risoluzione lineare, e come trasformare semplici problemi non lineari in problemi lineari.

Si consideri un generico *modello in programmazione matematica* in cui i vincoli sono tutti lineari, ma la funzione obiettivo sia del tipo:

$$\max \min e_1(x), e_2(x), \cdots, e_q(x)$$

con le $e_i(x)$ funzioni anch'esse lineari. è sempre possibile trasformare un modello del genere, detto **max-min** in uno equivalente lineare, introducendo una variabile $y = \min e_1(x), e_2(x), \cdots, e_q(x)$, trasformando la funzione obiettivo in $\max y$. Vanno inoltre aggiunti i vincoli di disequazioni $y \leq e_i(x), \quad \forall i \in 1, \cdots q$

Ipotizziamo di voler realizzare un certo prodotto finale, composto di tre parti che possono essere lavorate su quattro linee differenti di produzione. Ogni linea è dotata di una limitata capcità di ore di produzione.

| Linea | Capacità | Produttività Pt 1 | Produttività Pt 2 | Produttività Pt 3 |
|---|---|---|---|---|
| 1 | 100 | 10 | 15 | 5 |
| 2 | 150 | 15 | 10 | 5 |
| 3 | 80 | 20 | 5 | 10 |
| 4 | 200 | 10 | 15 | 20 |

Si vuole determinare il *numero di ore di lavorazione* di ciascuna parte su ciascuna linea di produzione, in modo da massimizzare il numero di unità complete del prodotto finale.

Le variabili decisionali saranno le $x_{ij}$, ovvero il numero di ore di lavorazione della parte $j$ sulla linea $i$, mentre l'unico vincolo presente, è relativo al numero di ore per linea:

$$\begin{gathered} x_{11} + x_{12} + x_{13} \leq 100 \\ x_{21} + x_{22} + x_{23} \leq 150 \\ x_{31} + x_{32} + x_{33} \leq 80 \\ x_{41} + x_{42} + x_{43} \leq 200 \end{gathered}$$

L'obiettivo sarà massimizzare il numero minimo tra il numero di pezzi di parte 1,2 e 3.

$$\max \min \begin{bmatrix} (10x_{11} + 15x_{21} + 20x_{31} + 10x_{41}); \\ (15x_{12} + 10x_{22} + 5x_{32} + 15x_{42}); \\ (5x_{13} + 5x_{23} + 10x_{33} + 20x_{43}) \end{bmatrix}$$

In accordo a quanto detto prim, aggiungo una variabile fittizia $y$, rappresentante il numero di pezzi completi che riesco a produrre, trasformando la funzione obiettivo nella massimizzazione di y. Mi occorrono però adesso nuovi vincoli, che uniscano la nuova variabile $y$ al resto del problema.

$$\begin{gathered} y \leq 0x_{11} + 15x_{21} + 20x_{31} + 10x_{41} \\ y \leq 15x_{12} + 10x_{22} + 5x_{32} + 15x_{42} \\ y \leq 5x_{13} + 5x_{23} + 10x_{33} + 20x_{43} \\ x_{11} + x_{12} + x_{13} \leq 100 \\ x_{21} + x_{22} + x_{23} \leq 150 \\ x_{31} + x_{32} + x_{33} \leq 80 \\ x_{41} + x_{42} + x_{43} \leq 200 \end{gathered}$$

Un altro modello di trasformazione da non lineare a lineare, è quello di **min abs**, dove la funzione obiettovo è del tipo $\min |e(x)|$ con $e(x)$ funzione lineare. Anche in questo caso parliamo di una non linearità semplice.Il costo da pagare nel caso precedente era quello di aggiungere una variabile e un insieme di vincoli e situazione analoga si presenta in questo caso.

Ricordiamo che $|e(x)| = \max e(x); -e(x)$, e allora trasormato il modulo secondo questa formula, ottengo una funzione obiettivo del

tipo min max, che risolvo secondo lo schema precedente.

$$y = \max e(x); -e(x) \qquad y \geq e(x) \qquad y \geq -e(x).$$

Vediamone un esempio: Ipotizziamo di dover pianificare l'esecuzione di 5 lotti su di una macchina rispettivamente di 5,7,4,7 e 10 minuti per lotto. La sequenza di esecuzione 1-2-3-4-5 è data e non ci può essere sovrapposizione temporale tra i lotti. Il primo lotto ha come ora di consegna desiderata le 10:32, il secondo le 10:38, il terzo le 10:42, il quarto le 10:52 e il quinto le 10:57. Sia l'*errore di un lotto* pari al valore assoluto della differenza tra il suo tempo di fine lavorazione e l'ora di consegna. Si vuole determinare il tempo di inizio di ciascuna lavorazione al fine di minimizzare la somma degli errori dei lotti.

Sia $i_j$ il tempo di lavorazione del lotto $j$, e siano i vincoli di precedenza tra i lotti i seguenti:

$$\begin{aligned} i_1 &\geq 0 \\ i_2 &\geq i_1 + 5 \\ i_3 &\geq i_2 + 7 \\ i_4 &\geq i_3 + 4 \\ i_5 &\geq i_4 + 7 \end{aligned}$$

Poiché i lotti non devono sovrapporsi, allora l'inizio di produzione del lotto successivo deve avvenire un certo tempo *dopo* il tempo di inizio di lavorazione del lotto precedente. $i_n \geq i_{n-1} + \alpha_i$ Per ottenere la funzione obiettivo bisogna tenere in conto dell'errore temporale sulla produzione dei lotti. L'errore del lotto 1 è dato dalla distanza di $i_1$ 5, ovvero quando termina il lotto 1, meno 632, che sarebbero le 10 e 32 espresso in minuti. Tale ragionamento può essere reiterato per ogni lotto, ottenendo la funzione obiettivo:

$$\min |i_1 + 5 - 632| + |i_2 + 7 - 638| + |i_3 + 4 - 642| + |i_4 + 7 - 652| + |i_5 + 10 - 657|$$

Dunque l'idea è quella di introdurre variabili fittizie $y_j$, una per ogni lotto, e minimizzarne la somma degli errori; aggiungendo il vincolo che $y_j \geq e(x)$ e $y_j \geq -e(x)$, completo il modello:

$$\min y_1 + y_2 + y_3 + y_4 + y_5$$

$$y_j \geq \pm(i_{j-1} + \alpha_j - t_j) \qquad \forall j = 1, \cdots, 5$$

con $\alpha$ già introdotto in precedenza, e $t_i$ l'orario di inizio della produzione

## 2.6 Rappresentazione grafica di problemi lineari

Quando il vettore x delle variabili di decisione è bidimensionale, il problema di programmazione lineare può essere facilmente risolto per *via grafica* nel piano cartesiano, mediante l'equazione lineare $ax_1 + bx_2 = c$. la retta individuata al crescere di $c$ si muove parallelamente a se stessa nella direzione del vettore $(a, b)^T$. Graficamente un vincolo quindi rappresentabile in un grafico $n-$dimensionale.

Ogni vincolo infatti, essendo una disequazione lineare, individua una regione nel piano; dati dunque $n$ vincoli, identifico $n$ regioni la cui intersezione definisce la **regione di ammissibilità**. Graficamente si può anche notare che alcuni vincolo possono esser ridondanti, in quanto non influiscono sul dominio di ammissibilità.

Per quanto ne concerne la funzione obiettivo,nel caso bidimensionale, essa sarà una funzione nello *spazio*, e quindi potrebbe essere opportuno tracciarne le *di livello*. Data una funzione qualsiasi, posso sezionarla tramite un piano parallelo al piano $xy$, nel caso di funzioni in due variabili. è chiaro che nel caso di funzione $n - dimensionale$, la funzione obiettivo avrà dimensione $n + 1$, e sarà quindi necessario tagliarla con un iperpiano. L'intersezione tra il piano e la funzione, proiettata su xy, definisce la *curva di livello*. E dunque è un insieme di punti in cui la funzione obiettivo assume gli stessi valori. Ciò significa, analiticamente, data una funzione obiettovo, fissarne y.

$$y = ax_1 + bx_2 = c$$

Essendoci posti nel caso di funzioni obiettivo lineari, la curva di livello, nel nostro caso, sarà una retta nel piano $xy$. Potendo tagliare il piano a qualsiasi altezza,le rette che posso individuare sono infinite e tutte parallele tra loro. Parlo allora di *fasci di linee di livello*, che rappresentano la funzione obiettivo. Se la mia funzione è del tipo $z \leq 5x + 10y$ è possibile fissare l'altezza $z$ a 2, e disegnare la prima linea di livello. Al variare di $z$, per esempio per $z = 4$ ottengo un'altra retta, parallela alla precedente, e caratterizzata dall' avere tutti i punti della funzione obiettivo a questa altezza pari a 4.

Graficamente se ci sono punti di ammissibilità del dominio relativo che si intersecano con una qualsiasi linea di livello, esse rappresentano i valori della funzione obiettivo che soddisfano i vincoli. Dunque la

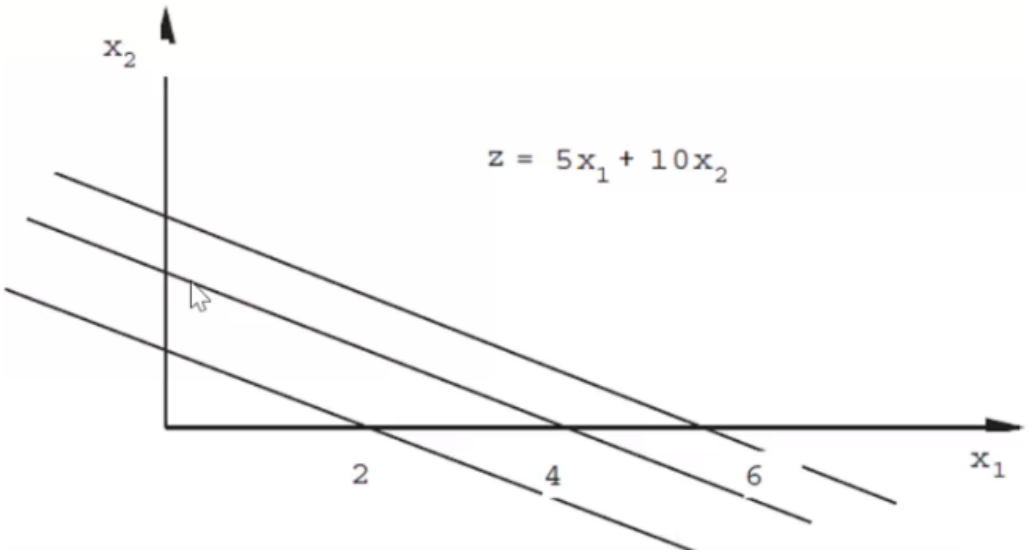



Figure 2: Linee di livello della funzione obiettvo

soluzione analitica si può ottenere effettuando l'intersezione tra la linea di livello e il dominio di ammissibilità.

È intuitivo notare che un qualsiasi problema di programmazione lineare in due dimensioni, possa avere un dominio di ammissibilità pari ad un poliedro dato dall'intersezione di semipiani. In un problema a massimizzare è chiaro che il punto di ottimo si troverà sulla **frontiera** del dominio di ammissibilità. Ciò è vero in 2 variabili, ma anche in $N$ variabili. Così facendo la ricerca va fatta solo su tutti i punti della frontiera.
È possibile però che la retta si sovrapponga perfettamente ad una parte della frontiera. Ciò implica allora l'avere *soluzioni non uniche*. Tuttavia alcune di queste infinite soluzioni ottime, si troveranno sui vertici del poliedro, che potremo provare essere *ottime tra le ottime*. Così facendo, l'**algoritmo del simplesso**, effettua una ricerca solo sui vertici di tale poliedro, limitando la ricerca solo ai vertici della frontiera.

Altra casistica plausibile è che il problema sia *inammissibile*. È infatti possibile che i vincoli siano in disaccordo tra loro a tal punto, da non individuare un dominio di ammissibilità. Ciò si traduce grafi-

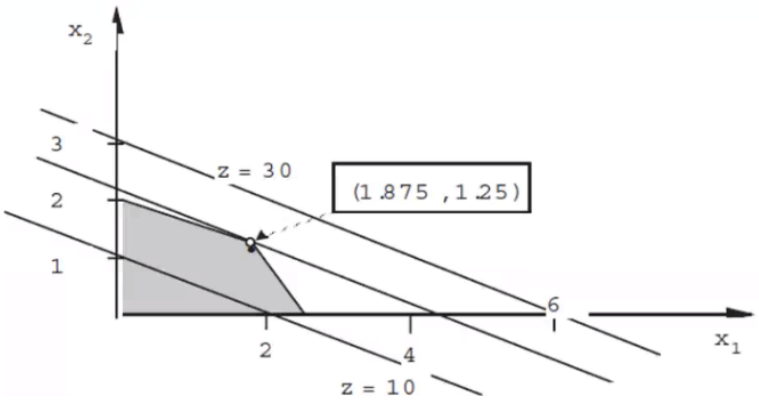



Figure 3: Soluzione grafica ad un problema lineare

camente in una regione ammissibile vuota, in quanto le disequazioni relative ai vincoli, non vanno ad identificare una regione intersezione.

Giocando sui vincoli è possibile ottenere quindi differenti configurazioni, come, al contrario di prima, una regione ammissibile *illimitata*. Ciò può accadere per esempio se una delle due variabili decisionale non è vincolata ad essere strettamente positiva. Questo significa che *non esiste una circonferenza di raggio finito in grado di contenere questa regione*. Ciò nonostante, la soluzione ottima al problema *non è illimitata*. Infatti, prendendo in considerazione le linee di livello, è possibile comunque notare che l'intersezione ottima rimane sempre unica.

In alcuni casi è però possibile sia che la regione ammissibile sia illimitata, sia che non esiste soluzione. Un esempio potrebbe essere il calcolo del minimo in caso di regione ammissibile illimitata, come mostrato nella figura rappresentante una regione illimitata. In questo caso il problema assume soluzioni infinte, in quanto è possible portare ad un valore arbitrariamente piccolo la funzione obiettivo.

Dato dunque un problema di PL si possono in generale presentare alcuni casi:

- La regione ammissibile è *vuota*, e quindi il problema non ammette soluzione
- La regione ammissibile è *illimitata*, e lo è anche la funzione obiettivo (superiormente in un problema a massimizzare o inferiormente in un problema a minimizzare) e anche in tal caso non ho soluzione.

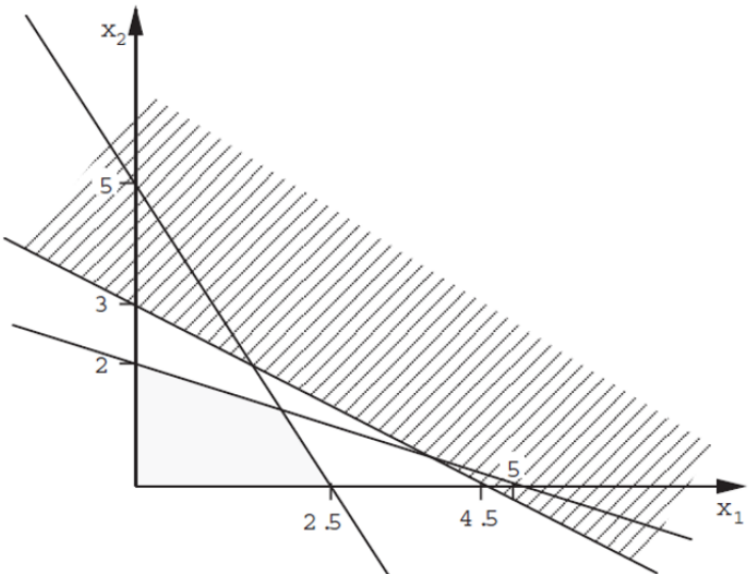



Figure 4: Problema inammissibile

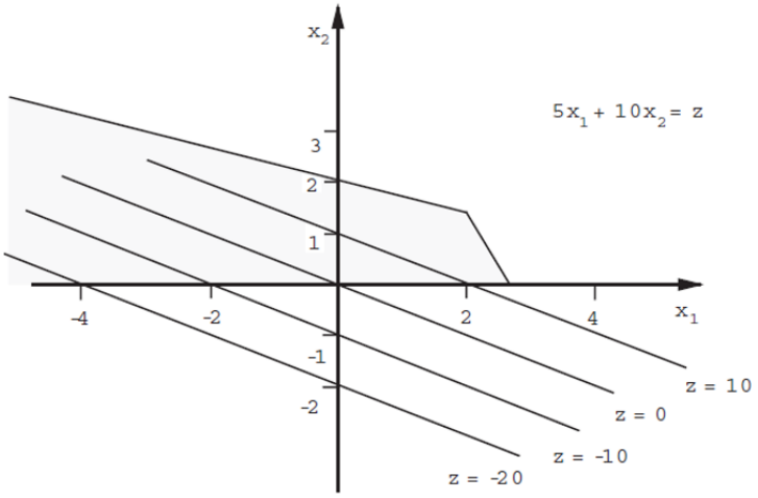



Figure 5: Regione di ammissibilità illimitata

- Il problema può ammettere una *soluzione ottima* , unica o non unica (infinite), ma almeno una di queste soluzioni ottime si trova in corrispondenza di uno dei vertici del poliedro.

Questa analisi è in generale vera per un problema di PL in $n$ variabili. Se questo è vero, se il problema ammette soluzioni, allora bisogna semplicemente stabilire in quale dei vertici del dominio di ammissibilità si trova una soluzione ottima del problema.

Introduciamo ora il **teorema fondamentale della programmazione lineare**. Dati $x, y \in \Re^n$, e considerato uno scalare $\lambda \in [0,1]$, si dice **combinazione convessa** di x e y un qualunque punto ottenuto come

$$\lambda x + (1-\lambda)y \qquad \lambda \in [0,1]$$

Si definisce così il segmento $n$-dimensionale che unisce x e y.
Un insieme è detto **convesso** *se comunque presi due punti appartenenti agli insiemi, il segmento che li congiunge appartiene interamente all'insieme.*Utilizzando questa definizione è possibile dimostrare che *l'intersezione di un numero finito di insiemi convessi è un insieme convesso.*
Sia a questo punto $a$ un vettore di $\Re^n$ e $b$ un numero reale, l'insieme

$$H = \{x \in \Re^n : a^T x = b\}$$

è detto *iperpiano*, definito dall'equazione $a^T x = b$.

Definiamo adesso due nuovi insiemi, detti **semispazi chiusi**, definiti da due disequazioni:

$$S^{\leq} = \{x \in \Re^n : a^T x \leq b\} \qquad S^{\geq} = \{x \in \Re^n : a^T x \geq b\}$$

Preso quindi un semispazio chiuso, questo è certamente insieme convesso.prendendo infatti due qualsiasi punti $x, y \in S^{\leq}$ e considerarndone una combinazione convessa $z = \beta x + (1-\beta)y$, posso dimostrare che z sia appartenente ad $S^{\leq}$, convesso.

$$a^T z = a^T(\beta x + (1-\beta)y) = \beta a^T x + (1-\beta)a^T y \leq \beta b + (\ -\beta)b = b$$

$$a^T z \leq b \implies z \in S^{\leq} \implies S^{\leq} \quad \textit{convesso}$$

Essendo il dominio di ammissibilità di un PL, l'intersezione di un numero finito di semispazi chiusi, o iperpiani, il dominio di ammissibilità è allora un insieme convesso. In particolare esso assume la forma di una figura detta **poliedro**

Altro concetto fondamentale nella teoria della programmazione lineare è rivestito dal concetto di **vertice**. L'insieme in figura possiamo notare essere sicuramente un insieme convesso. Preso infatti qualsiasi punto A,B appartenente alla figura, qualsiasi segmento sarà ancora interno alla figura stessa.
Un vettore x appartenente ad un insieme convesso C è detto vertice di C se :

$$\nexists \quad x_1, x_2 \in C : x \neq x_1, x \neq x_2 \quad \forall x \in [x_1, x_2]$$

In figura per esempio, il punto A si trova sul segmento BC, mentre magari il punto D si trova sul segmento che congiunge F con E. se seleziono il punto F però, non esiste nessun segmento i cui estremi siano interni all'insieme e checontengano il punto F.
Notiamo inoltre che tutti i punti di un arco di circonferenza sono, per definizione, vertici. Ma allora l'esempio in figura ha infiniti vertici, il che ovviamente non andrebbe a limitare una ricerca di programmazione lineare. Tuttavia un poliedro costituito dall'intersezione di un numero finito di semispazi chiusi non sarà mai caratterizzato da una frontiera arcuata. Infatti essendo un poliedro costituito dall'intersezione di regioni determinate da disequazioni lineari, ovvero sempiani, non è possibile, a meno di avere infiniti vincoli, ottenere una frontiera arcuata in un problema lineare.

Considerando invece un insieme dalla forma a fascia di piano, tale insieme non presenta alcun vertice, poichè comunque presi A e B nella regione, riusciamo sempre a trovare un segmento che li includa
.

Se il problema ammette soluzioni ottime e il poliedro che definisce la regione ammissibile ha dei vertici, allora, *almeno una soluzione ottima cade su di un vertice.*
Sia P l'insieme ammissibile di un problema lineare di minimizzazione. Sia $x^*$ una soluzione ottima del problema e sia $z^*$ il valore che la funzione obiettivo assume in corrispondenza del punto $x^*$.

$$z^* = c^T x^*$$

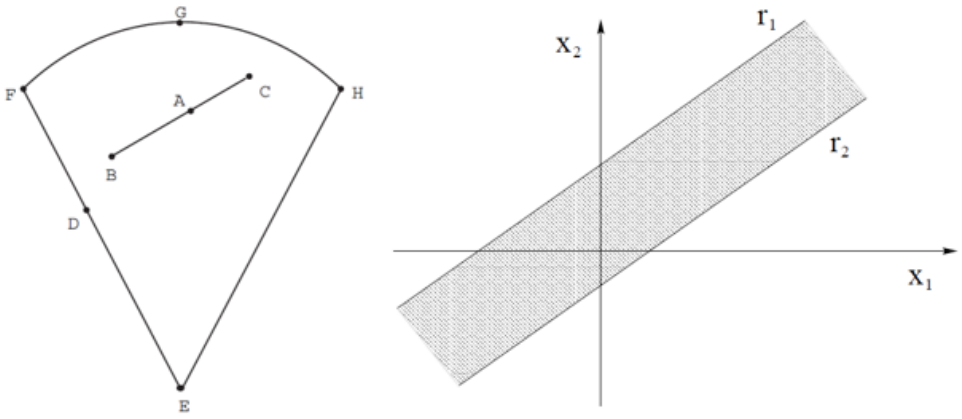



Figure 6: Regioni di ammissibilità: Arcuata e Fascia di piano

Sia $SOL$ un insieme convesso dato dall'intersezione tra il dominio di ammissibilità P e l'insieme di tutte le soluzioni ottime del problema, ovvero tutti i punti x per cui la funzione obiettivo assume il valore ottimo $z^*$.

$$SOL = P \cap \{x \in \Re^n : c^T x = z^*\}$$

SOL definisce un poliedro convesso poiché intersezione di due insiemi convessi. Per dimostrare il teorema, occorre dimostrare che se v è un vertice di $SOL$, allora è anche vertice del poliedro P.
È possibile realizzar una dimostrazione per assurdo; Dato un problema di minimizzazione, ipotizziamo per *assurdo* che v sia un vertice di SOL, ma *non* un vertice di P. allora vuol dire che devono esistere due punti di P, x,y tali che v sia combinazione convessa di x e y. In altre parole, esiste un segmento.

$$\exists \quad x, y \in P : v = \beta x + (1 - \beta)y \quad \beta \in ]0, 1[$$

Se x e y sono due punti generici di P, il valore della funzione di x e y deve essere maggiore o uguale di $z^*$, altrimenti sarebbero punti ottimi.

$$c^T x \geq z^* \quad c^T y \geq z^*$$

Ma $z^* = c^T v$ e posso perciò riscriverlo come combinazione convessa.

$$z^* = c^T v = \beta c^T x + (1-\beta) c^T y$$

Ma allora mettendo insieme queste condizioni, x ed y appartengono a $SOL$, e quindi *soluzioni ottime*.

Introduciamo un *esempio di analisi grafica*. Ipotizziamo di avere una azienda chimica in grado di produrre due tipi di fertilizzanti. Ogni quintale del fertilizzante A contiene 0.1 quintali di azoto e 0.3 quintali di potassio per un prezzo di vendita di 300 euro.
Ogni quintale di B invece contiene 0.2 quintali di azoto e 0.1 quintali di potassio, al prezzo di 400 euro. L'azienda dispone di 9 quintali di azoto e 10 di potassio.Si vuole conoscere quali sono le produzioni giornialiere espresse in quintali di A e B che rendono massimo il ricavo.

$$\max z = 300x_1 + 400x_2$$

Naturalmente questo è un problema di allocazione ottima di risorse, dove tali risorse sono azoto e potassio, e i prodotti da realizzare sono i fertilizzanti di tipo A e B. Dunque stabilite risorse e prodotti, possiamo definire i vincoli del problema:

$$\begin{cases} 0.1x_1 + 0.2x_2 \leq 9 \\ 0.3x_1 + 0.1x_2 \leq 10 \\ x_1, x_2 \geq 0 \end{cases}$$

è dunque possibile disegnare graficamente le rette corrispondenti ai vincoli, le cui regioni definite, ed intersecate, vanno a stabilire il dominio di ammissibilità del problema.

è possibile a questo punto valutare il valore della funzione obiettivo da massimizzare nei punti dei vertici A,B,C,O. Il valore massimo, sarà il valore ottimo del problema.

## 2.7 Metodo del simplesso

Un algoritmo risolve un problema di PL se esso è capace di determinare correttamente se il problema dato è vuoto,illimitato oppure, se nessuno di questi due casi riuslta verificato, sia capace di individuare una soluzione ottima.
Il **metodo del simplesso** è stato il primo algoritmo pratico per la risoluzione di problemi di PL ed è tuttora il più usato. Poiché è vero

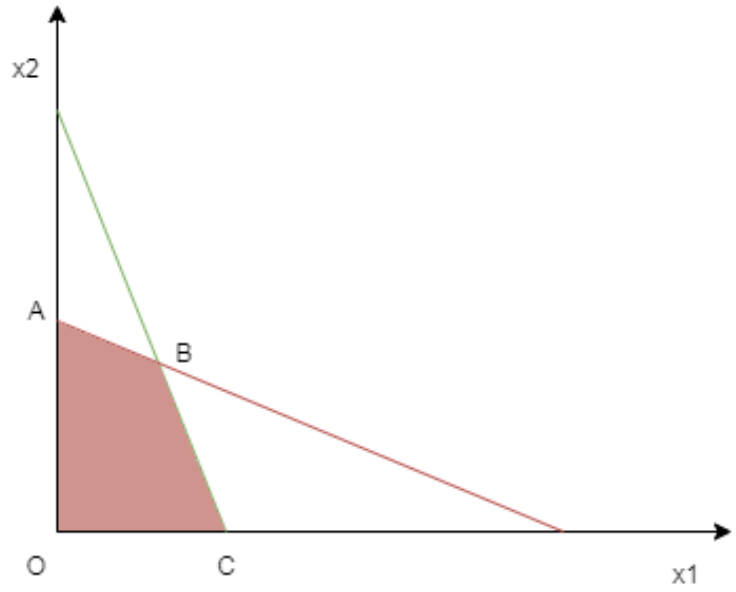



Figure 7: Esempio: Dominio di ammissibilità del problema

che un poliedro ha sempre un numero di vertici finiti, allora è anche vero che un dominio di ammissibilità ammette un finito numero di soluzioni ottime. Tipicamente un poliedro con n variabili decisionali identifica nell'iperspazio $2^n$ vertici.
L'algoritmo del simplesso inizia identificando i vertici del poliedro, in maniera enumerativa. Un algoritmo enumerativo valuterebbe la funzione obiettivo nei punti corrispondenti ai vertici, prendendo il migliore. Un approccio a forza bruta è un esempio di algoritmo enumerativo che va però a valutare ogni possibile permutazione dei valori di ingresso.

Il simplesso invece effettua questa valutazione in maniera "intelligente". Riesce infatti a stabilire il vertice ottimo anche senza aver elencato tutti i vertici del problema. Dunque determina un vertice iniziale $v$ di $P$ e valuta se $v$ sia una soluzione ottima. Se $v$ non è una soluzione ottima allora determina in maniera intelligente un nuovo vertice $v$ del poliedro $P$ e verifica di nuovo se essa sia una soluzione ottima, e così via fino a convergenza del problema.

Algoritmi come L'algoritmo del simplesso, sono detti *algoritmi a*

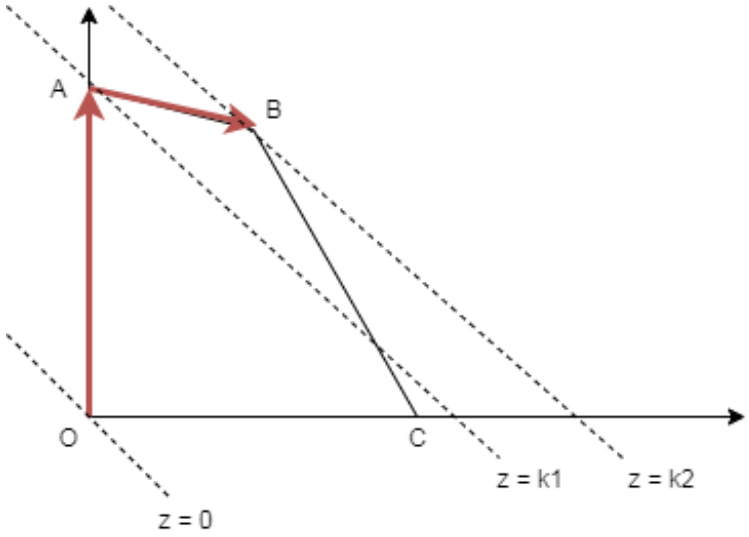



Figure 8: Esempio: Percorso sul dominio di ammissibilità

*direzione ammissibile*,vale a dire che generano una successione di punti attraverso la relazione $x^{k+1} = x^k + \theta_k d^k$

Con $\theta_k$ *lunghezza dello spostamento*, e $d^k$ *direzione di spostamento ammissibile e di miglioramento*. Possiamo graficamente immaginare queste quantità come dei vettori con una lunghezza e direzione. I termini $x^k$ sono invece i vertici del poligono.

Preso il dominio di ammissibilità in figura 8, ipotizziamo di voler risolvere un problema lineare a massimizzare.. Partiamo allora dalla linea di livello (tratteggiate in figura) relativa all'origine del piano $x_1$,$x_2$. Se il punto di ottimo esiste si trova certamente in corrispondenza di un vertice, e quindi tra tutte le possibili direzioni interne al poliedro, scegliamo *quelle sulla frontiera* che puntano verso un vertice adiacente.
Dunque cerchiamo di capire cosa accade se decidiamo di spostarci verso A piuttosto che verso C. Tali spostamenti sono coincidenti allo svolgimento di una derivata parziale lungo la direzione scelta.
L'algoritmo del simplesso sceglie tra le due derivate direzionali quella più conveniente. Ipotizziamo di scegliere come direzione di spostamento quella lungo l'asse $x_2$, che congiunge il punto O al punto A. L'andamento lungo $x_2$ è un andamento lineare in quanto la funzione

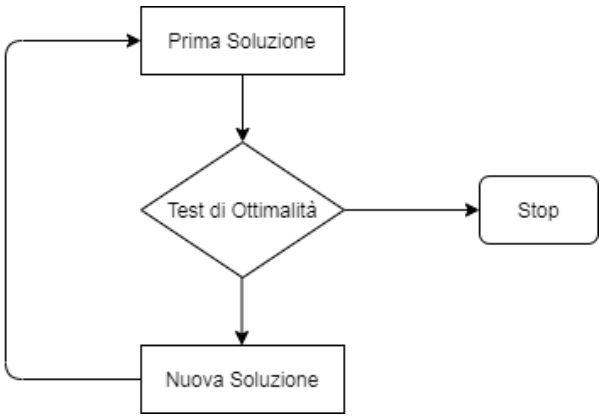



Figure 9: Schema logico dell'algoritmo del simplesso

obiettivo è anch'essa lineare, ed in virtù di ciò stiamo certamente migliorando la funzione obiettivo. Continuando con questo ragionamento potrei valutare la derivata lungo la direzione di B, che sarà certamente positiva, in quanto le linee di livello relative a B si trovano più in alto rispetto a quelle del punto A. Raggiunto il vertice B, spostandomi lungo A o C, otterrei un peggioramento della funzione obiettivo, poiché si "abbasserebbero" le linee di livello.
L'algoritmo del simplesso tuttavia non fa direttamente uso delle derivate direzionali, Infatti basta valutare soltanto i valori, dato O, della funzione in A e C. partendo da un vertice infatti, mi basta controllare che il valore della funzione obiettivo sia il più grande rispetto a quello calcolato in tutti i vertici adiacenti.

Si parte quindi da una prima soluzione, ovvero un vertice, e si cerca di capire se è il vertice ottimo. Se lo è, l'algoritmo termina, altrimenti si passa ad una nuova soluzione, ovvero uno dei vertici adiacenti.

La soluzione ottima di un problmea di programmazione lineare, se esiste si trova in corrispondenza di un vertice del dominio di ammissibilità. Si può inoltre dimostrare che i vertici del dominio di ammissibilità di un problema PL corrispondono a particolari soluzioni, dette **basiche**, del sistema di equazioni ottenuto a partire dai vincoli del problema *trasformati tutti in vincoli di uguaglianza.*

E allora il primo passo dell'algoritmo consiste nel trasformare il

problema originario in una forma *standard*, ovvero con tutti vincoli di uguaglianza. Immaginiamo di avere una delle variabili, $x_i$, non ristretta nel segno, ovvero tale da poter assumere valori minori o uguali a zero; con variabili negative non sarebbe possibile utilizzare l'algoritmo del simplesso ed in tal caso $x_i$ può essere espresso come differenza di due variabili $x_i$' e $x_i$", dove entrambe queste variabili sono considerate positive.

Per poter effettuare la trasformazione sopra descritta è necessario fare uso di una variabile ausiliaria o **variabile Slack**, $y_i$. A questo punto, un vincolo di tipo $\leq$ si trasforma in un vincolo di tipo $=$ aggiungendo al termine di sinistra la variabile slack, altrimenti in caso di tipo $\geq$, trasformiamo la relazione in una di uguaglianza *sottraendo* la variabile slack.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \pm y_1 = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \pm y_2 = b_2 \\ \cdots + \cdots + \cdots + \cdots + \cdots = \cdots \\ a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n \pm y_i = b_i \\ \cdots + \cdots + \cdots + \cdots + \cdots = \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \pm y_m = b_m \end{cases}$$

Facendo riferimento alla figura, è possibile notare come un qualsiasi punto P appartenente alla frontiera sia caratterizzato dall'avere variabile slack $y_i = 0$. Preso invece un generico punto H, interno al dominio, la variabile slack assumerà valore diverso da 0. $y_i$ rappresenta infatti la *distanza* tra un punto interno al dominio di ammissibilità e la sua frontiera.
Ma un punto H in uno spazio bidipensionale, può essere descritto da una quaterna $(x_1, x_2, y_1, y_2)$, parliamo allora di **spazio esteso**.

In generale dato un sistema di m equazioni in n variabili con $n > m$, definiamo **Soluzione basica ammissibile** del sistema, una soluzione che presenta n-m variabili nulle ed m variabili non negative. Una soluzione basica ammissibile può essere *Degenere* nel caso in cui essa presenti $n-m$' variabili nulle ed m' variabili non negative, con $m' > m$. Ma allora il numero delle soluzioni basiche ammissibili di un sistema di equazioni è al più pari al numero di combinazioni di m ed n, ovvero:

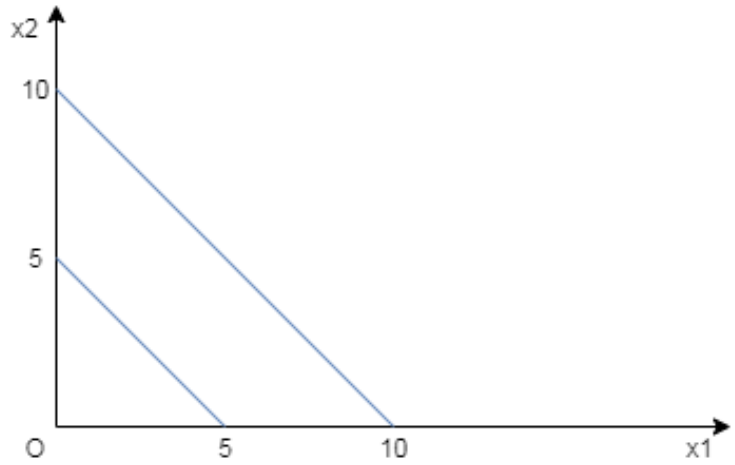



Figure 10: Esempio: Dominio di ammissibilità non comprendente l'origine

$$\frac{n!}{m!(n-m)!}$$

Dal punto di vista geometrico l'algoritmo del simplesso non fa altro, ad ogni iterazione, che spostarsi di vertice in vertice. Esiste inoltre una corrispondenza biunivoca tra vertici del dominio di ammissibilità e soluzioni basiche ammissibili del sistema di equazioni del problema in forma standard.

Consideriamo un problema in due variabili non negative e tre vincoli il cui dominio di ammissibilità è visibile in figura 11. Possiamo stilare in tabella la corrispondenza tra i vertici del poligono e le variabili dello spazio esteso, nella forma standard.

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|---|
| Vertice O | $= 0$ | $= 0$ | $\geq 0$ | $\geq 0$ | $\geq 0$ |
| Vertice A | $= 0$ | $\geq 0$ | $= 0$ | $\geq 0$ | $\geq 0$ |
| Vertice B | $\geq 0$ | $\geq 0$ | $= 0$ | $= 0$ | $\geq 0$ |
| Vertice C | $\geq 0$ | $\geq 0$ | $\geq 0$ | $= 0$ | $= 0$ |
| Vertice D | $\geq 0$ | $= 0$ | $\geq 0$ | $\geq 0$ | $= 0$ |

Nel vertice O possiamo notare come $x_1$ ed $x_2$ siano le variabili non basiche, in quanto nulle, e sono in numero n-m, come previsto dalla teoria. Il fatto che tutte le variabili slack siano maggiori di zero è

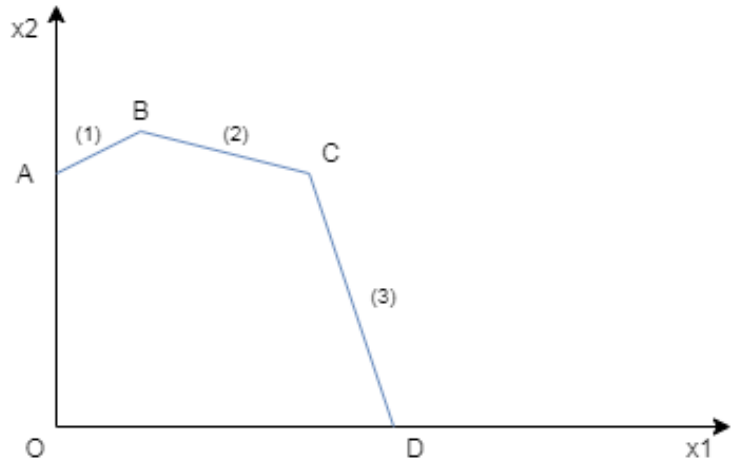



Figure 11: Esempio: vertici del dominio di ammissibilità e soluzioni basiche ammissibili

dovuto al fatto che il punto O, non appartiene a nessuno dei vincoli. Ad O dunque corrisponde una soluzione basica, date 3 variabili basiche e 2 non basiche.
Anche per il vertice A vale lo stesso discorso, poichè infatti continuiamo ad avere 3 variabili basiche e 2 non basiche. tuttavia in questo caso la variabile $x_2$ è diventata una variabile basica, mentre $y_1$ è diventata nulla.

Da questo esempio si nota che qualsiasi vertice del poliedro corrisponde, in questo caso, esattamente ad una soluzione basica ammissibile. L'adiacenza geometrica, si traduce algebricamente in una coincidenza delle variabili basiche. Infatti notiamo che dal vertice O al vertice A, le variabili basiche $y_2$ e $y_3$ rimangono entrambe basiche, perchè relative al segmento BC e CD, mentre la variabile slack $y_1$ afferisce al segmento AB.
Ancora, nel vertice B abbiamo $x_1$,$x_2$ e $y_3$ come variabili basiche mentre Il vertice C, essendo adiacente, mantiene 2 delle tre precedenti variabili basiche.
Possiamo allora concludere che tra vertici adiacenti, vi sarà sempre una variazione di una singola variabile basica e che se il vertice non appartiene agli assi, allora sarà una delle variabili slack a diventare non basica, ovvero ad annullarsi

### 2.7.1 Vincoli di tipo Minore Uguale

Poniamoci nel caso più elementare in cui i vincoli del problema siano tutti di tipo $\leq$, quindi con variabili slack positive. In questo caso trovare una variabile basica ammissibile è semplice , in quanto la prima soluzione è sempre l'origine, che mi stabilisce un punto di partenza per l'algoritmo. Ciò si traduce nel dire che la prima soluzione basica ammissible, la possiamo ottenere semplicemente *trasformando il problema in forma canonica*, ovvero aggiungendovi le variabili slack, e trasformando i vincoli di $\leq$ in vincoli di uguaglianza.

$$\max z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + y_1 = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + y_2 = b_2 \\ \cdots + \cdots + \cdots + \cdots + \cdots = \cdots \\ a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n + y_i = b_i \\ \cdots + \cdots + \cdots + \cdots + \cdots = \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + y_m = b_m \end{cases}$$

dove la prima soluzione basica ammissibile sarà, come già detto, coincidente con l'origine degli assi:

$$x_1 = x_2 = \cdots = x_n \quad y_i = b_i \geq 0 \qquad \forall i = 1, \cdots, m$$

Possiamo adesso provare a schematizzare l'intero problema in un formato tabluare:

| | $x_1$ | $x_2$ | $\cdots$ | $x_j$ | $\cdots$ | $x_n$ | $y_1$ | $y_2$ | $\cdots$ | $y_i$ | $\cdots$ | $y_n$ | -z | b |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $y_1$ | $a_{11}$ | $a_{12}$ | $\cdots$ | $a_{1j}$ | $\cdots$ | $a_{1n}$ | 1 | 0 | $\cdots$ | 0 | $\cdots$ | 0 | 0 | $b_1$ |
| $y_2$ | $a_{21}$ | $a_{22}$ | $\cdots$ | $a_{2j}$ | $\cdots$ | $a_{2n}$ | 0 | 1 | $\cdots$ | 0 | $\cdots$ | 0 | 0 | $b_2$ |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $y_i$ | $a_{i1}$ | $a_{i2}$ | $\cdots$ | $a_{ij}$ | $\cdots$ | $a_{in}$ | 0 | 0 | $\cdots$ | 1 | $\cdots$ | 0 | 0 | $b_i$ |
| $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $y_m$ | $a_{m1}$ | $a_{m2}$ | $\cdots$ | $a_{mj}$ | $\cdots$ | $a_{mn}$ | 0 | 0 | $\cdots$ | 0 | $\cdots$ | 1 | 0 | $b_m$ |
| | $c_1$ | $c_2$ | $\cdots$ | $c_j$ | $\cdots$ | $c_n$ | 1 | 0 | $\cdots$ | 0 | $\cdots$ | 0 | 1 | 0 |

Avremo dunque una colonna per ogni variabile $x_i$, una per ogni slack $y_i$, una colonna extra per $-z$, la funzione obiettivo, e una per i

termini noti $b$. Le righe invece saranno relative ai coefficienti moltiplicativi delle singole variabili citate, e rappresenteranno le variabili in *base*; fa eccezione la funzione obiettivo. Infatti per completare il sistema è necessario anche inserire una riga per la funzione obiettivo, esplicitata in questo caso come:

$$0 = -z + c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

dove generico coefficiente $c_j$ rappresenta il valore della derivata parziale nella direzione di $x_j$.
Ogni volta che viene selezionata una soluzione, è necessario effettuare un test di ottimalità, il quale risulta essere decisamente semplice.
Infatti se il problema è a massimizzare, la soluzione corrente è **ottima** se $c_j \leq 0 \quad \forall j = 1, \cdots, n$.
Viceversa se il problema è a minimizzare, a soluzione corrente è **ottima** se $c_j \geq 0 \quad \forall j = 1, \cdots, n$.

A questo punto per ottenere una *nuova soluzione basica ammissibile* occore determinare la nuova **variabile entrante** in base e la nuova **variabile uscente** dalla base. In particolare la variabile entrante deve essere quella cui corrisponde il valore minimo/massimo del coefficiente di costo $c_j$. Sia $x_j$ la variabile entrante, è allora possibile calcolare il valore delle variabili slack all'aumentare di $x_j$ come:

$$y_i = b_i - a_{ij}x_j$$

Inoltre le variabili slack non possono assumere valori negativi, ne discenede allora che

$$y_i \geq 0 \implies x_j \geq \frac{b_i}{a_{ij}} \quad \forall i : a_{ij} > 0$$

Individuiamo la variabile $\hat{i}$-esima uscente come quella che si annulla per prima all'aumentare di $x_j$, e dunque

$$\frac{b_{\hat{i}}}{a_{\hat{i}j}} = \min\left\{ \frac{b_i}{a_{ij}} \quad \forall i \in 1, \cdots, m : a_{ij} > 0 \right\}$$

Date queste premesse cercheremo di sfruttare le proprietà elementare su di un sistema di equazioni lineari. Infatti ricordiamo che due sistemi lineari con n incognite si dicono *equivalenti* se e solo se hanno lo stesso insieme di soluzioni. è infatti possibile trasformare

un sistema in un suo equivalente scambiando due equazioni, moltiplicando un'equazione per un numero diverso da zero e sostituendo ad una equazione la somma della stessa con un'altra moltiplicata per un numero diverso da zero.

L'algoritmo fa quindi largamente uso di queste operazioni; dopo aver determinato la variabile entrante $x_s$ (colonna) e la variabile uscente $y_r$ (riga), la configurazione della nuova soluzione basica ammissibile è nota. la trasformazione conseguente del sistema prende il nome di **pivoting**, e consiste nel dividere tutti gli elementi della riga r-esima per l'elemento pivot $a_{rs}$, individuato dalla colonna della variabile entrante e e dalla riga della variabile uscente.

$$a'_r = \frac{a_r}{a_{rs}} = \left(\frac{a_{1s}}{a_{rs}}; \frac{a_{2s}}{a_{rs}}; \cdots; 1; \cdots; \frac{a_{ns}}{a_{rs}}\right)$$

Dopodichè si sottrae a ciascuna riga i-esima diversa dalla riga pivot, la nuova riga pivot moltiplicata per l'elemento $a_{is}$

$$a'_i = a_i - a_{is}a'_r = (a_{i1} - a_{is}a'_{r1}; a_{i2} - a_{is}a'_{r2}; \cdots; 0; \cdots; a_{n1} - a_{is}a'_{rn})$$

Tale set di operazioni va effettuata anche sulla riga relativa alla funzione obiettivo; di conseguenza tale funzione, a seguito delle operazioni di pivot, potrà essere espressa come:

$$\sum_j c'_j x_j + \sum_i c'_{n+i} y_i - b'_{m+1} = z$$

Terminato questo generico passo algoritmico, è necessario effettuare il **test di ottimalità**. Se il problema è a massimizzare, la soluzione corrente è ottima se

$$c'_j \leq 0 \quad \forall j = 1, \cdots, n+m$$

viceversa se il problema è a minimizzare, la soluzione corrente risulta ottima solo se

$$c'_j \geq 0 \quad \forall j = 1, \cdots, n+m$$

Per comprendere al meglio questo caso dell'algoritmo del simplesso, introduciamo un esempio: sia il nostro un problema a *massimizzare*, e sia la modellazione del problema tale da ottenere il seguente sistema di equazioni:

$$\begin{cases} \max z = 11x_1 + 10x_2 \\ -5x_1 + 4x_2 \leq 20 \\ x_2 \leq 6.5 \\ 3x_1 + 4x_2 \leq 36 \\ 2x_1 + x_2 \leq 16 \end{cases}$$

Portiamo a questo punto il sistema in forma canonica, aggiungendo le variabili slack $y_i$:

$$\begin{cases} \max z = 11x_1 + 10x_2 \\ -5x_1 + 4x_2 + y_1 = 20 \\ x_2 + y_2 = 6.5 \\ 3x_1 + 4x_2 + y_3 = 36 \\ 2x_1 + x_2 + y_4 = 16 \end{cases}$$

Trovandoci nel caso in cui tutti i vincoli sono di tipo minore o uguale, possiamo con certezza affermare che l'origine sia una soluzione basica ammissibile. Tale soluzione sarà perciò il *punto di partenza* dell'algoritmo.

$$y_1 = 20 \quad y_2 = 6.5 \quad y_3 = 36 \quad y_4 = 16 \quad x_1 = 0 \quad x_2 = 0 \quad z = 0$$

Costruiamo ora la tabella relativa all'algoritmo, in maniera tale da visualizzarne gli step.

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | -z | b |
|---|---|---|---|---|---|---|---|---|
| $y_1$ | -5 | 4 | 1 | 0 | 0 | 0 | 0 | 20 |
| $y_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 6.5 |
| $y_3$ | 3 | 4 | 0 | 0 | 1 | 0 | 0 | 36 |
| $y_4$ | 2 | 1 | 0 | 0 | 0 | 1 | 0 | 16 |
| $-z$ | 11 | 10 | 0 | 0 | 0 | 0 | 1 | 0 |

La variabile entrante da scegliere dovrà essere $x_1$ in quanto corrispondente al valore massimo attuale della funzione obiettivo, ovvero 11. è ora necessario selezionare la variabile uscente dalla base.
per fare ciò ricordiamo, che è necessario, data la prima colonna, trovare il minimo tra i rapporti $\frac{b_1}{a_{i1}} \quad \forall i \in 1, \cdots, m : a_{i1} > 0$ .
Dunque la prima riga, relativa ad $y_1$ deve essere scartata in quanto il coefficiente $a_{11} = -5$ è negativo. Tra tutti i rapporti, il minore è

chiaramente quello relativo ad $y_4$, che possiamo designare come *variabile uscente*.

Possiamo ora dividere tutti gli elementi della riga 4 per l'elemento pivot $a_{4,1} = 2$, dopodichè sottraiamo a ciascuna riga i diversa dalla riga pivot, la nuova riga pivot $a'_4$ moltiplicata per l'elemento $a_{i1}$. Otteniamo così la seguente tabella:

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | -z | b |
|---|---|---|---|---|---|---|---|---|
| $y_1$ | 0 | 6.5 | 1 | 0 | 0 | 2.5 | 0 | 60 |
| $y_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 6.5 |
| $y_3$ | 0 | 2.5 | 0 | 0 | 1 | -1.5 | 0 | 12 |
| $x_1$ | 1 | 0.5 | 0 | 0 | 0 | 0.5 | 0 | 8 |
| $-z$ | 0 | 4.5 | 0 | 0 | 0 | -5.5 | 1 | -88 |

Notiamo che essendo i termini di costo della funzione obiettivo $c_ij$ non tutti minori o uguali di zero, tale soluzione basica ammissibile non passa il test di ottimalità. è quindi necessario reiterare il passo algoritmico, individuando una nuova variabile entrante, ed una nuova uscente. Nel secondo passo è chiaro che la variabile entrante, corrispondente al valore massimo di funzione obiettivo, sarà $x_2$, mentre quella uscente, data dal minimo rapporto visto in precedenza, sarà $y_3$.

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | -z | b |
|---|---|---|---|---|---|---|---|---|
| $y_1$ | 0 | 0 | 1 | 0 | -2.6 | 6.4 | 0 | 28.8 |
| $y_2$ | 0 | 0 | 0 | 1 | -0.4 | 0.6 | 0 | 1.7 |
| $x_2$ | 0 | 1 | 0 | 0 | -0.4 | -0.6 | 0 | 4.8 |
| $x_1$ | 1 | 0 | 0 | 0 | -0.2 | 0.8 | 0 | 5.6 |
| $-z$ | 0 | 0 | 0 | 0 | 0 | -1.8 | -2.8 | -109.6 |

Notiamo che la nuova soluzione soddisfa ora il test di ottimalità, per cui la **soluzione ottima** sarà:

$$y_1 = 28.8 \quad y_2 = 1.7 \quad x_2 = 4.8 \quad x_1 = 5.6 \quad y_3 = 0 \quad y_4 = 0 \quad z = 109.6$$

### 2.7.2 Vincoli di tipo Maggiore Uguale

Passiamo ora ad analizzare una casistica più complessa, ovvero il caso in cui i vincoli siano di tipo **Uguali o Maggiori uguali**. In questo caso il problema in forma canonica non contiene la matrice identità,

in quanto alcune delle variabili slack saranno prese con segno positivo (caso di vincoli minori o uguali), mentre altre saranno prese con segno negativo (caso di vincoli maggiori o uguali).

Come abbiamo già presentato, in questo caso particolare l'origine non sarà una delle soluzioni basiche ammissibili, dunque il problema prevederà una prima fase di ricerca di una prima soluzione basica ammissibile, per poi reiterare l'algoritmo come già visto in precedenza.

Per risolvere tale inconveniente è possibile aggiungere una nuova variabile ausiliaria $h_i$, detta **variabile artificiale**, per ciascun vincolo di tipo uguale o maggiore uguale. Ognuna di queste variabili sarà caratterizzata dall'avere coefficiente unitario e segno positivo. Il sistema a questo punto presenterà m equazioni ad (n + p + q + r) variabili, di cui:

- **n** variabili originarie
- **p** variabili slack con segno positivo, relative ai vincoli di $\leq$
- **q** variabili slack con segno negativo, relative ai vincoli di $\geq$
- **r** variabili artificiali con segno positivo, relative ai vincoli di = e $\geq$

Ovviamente aggiungere nuove variabili artificiali si traduce nel definire un problema esteso; l'obiettivo è trovare una soluzione che sia *ammissibile anche per il problema originario.*
Il primo metodo che andiamo a presentare è il **Metodo del bigM**, che si basa sulla modifica della funzione obiettivo z del problema originario, introducendo le variabili artificiali $h_i$ con segno *negativo* se la funzione obiettivo è a massimizzare, viceversa se è a minimizzare.

In questo modo si favorisce l'annullamento delle variabili artificiali e quindi la loro uscita dalla base. Per accelerare il processo le variabili $h_i$ vengono quindi moltiplicate per un coefficiente positivo molto elevato, ovvero M.

Per poter innescare l'algoritmo è necessario rimettere il sistema in forma canonica rispetto alle variabili artificiali. A tale scopo bisogna effettuare preliminarmente una operazione di pivoting per ogni colonna di variabile artificiale. è possibile che l'algoritmo non riesca ad eliminare le variabili artificiali dalla base: ciò implica che il problema

originario è inconsistente.

Facciamo dunque un esempio relativo a tale metodo; poniamo il seguente sistema come modellazione di un problema lineare, già reso in forma canonica

$$\begin{cases} \max z = x_1 + 2x_2 - Mh_1 \\ 2x_1 + 5x_2 + y_1 = 100 \\ 5x_1 + 2x_2 + y_2 = 100 \\ x_1 + x_2 - y_3 + h_1 = 10 \end{cases}$$

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $h_1$ | -z | b |
|---|---|---|---|---|---|---|---|---|
| $y_1$ | 2 | 5 | 1 | 0 | 0 | 0 | 0 | 100 |
| $y_2$ | 5 | 2 | 0 | 1 | 0 | 0 | 0 | 100 |
| $h_1$ | 1 | 1 | 0 | 0 | -1 | 1 | 0 | 10 |
| $-z$ | 1 | 2 | 0 | 0 | 0 | -M | 1 | 0 |

Come prima cosa prendiamo la riga relativa ad $h_1$, e previa una moltiplicazione per M, la sommiamo alla riga della funzione obiettivo z, ottenendo la nuova tabella

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $h_1$ | -z | b |
|---|---|---|---|---|---|---|---|---|
| $y_1$ | 2 | 5 | 1 | 0 | 0 | 0 | 0 | 100 |
| $y_2$ | 5 | 2 | 0 | 1 | 0 | 0 | 0 | 100 |
| $h_1$ | 1 | 1 | 0 | 0 | -1 | 1 | 0 | 10 |
| $-z$ | 1+M | 2+M | 0 | 0 | -M | 0 | 1 | 10M |

A questo punto possiamo procedere individuando la variabile entrante, e quella uscente. Chiaramente dato M arbitrariamente grande, la variabile entrante sarà $x_2$ e quella uscente sarà invece $h_1$. Siamo così in grado di far uscire la variabile artificiale dalla base, e ottenere la seguente tabella, a valle delle operazioni algoritmiche:

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $h_1$ | -z | b |
|---|---|---|---|---|---|---|---|---|
| $y_1$ | -3 | 0 | 1 | 0 | 5 | -5 | 0 | 50 |
| $y_2$ | 3 | 0 | 0 | 1 | 2 | -2 | 0 | 80 |
| $x_1$ | 1 | 1 | 0 | 0 | -1 | 1 | 0 | 10 |
| $-z$ | -1 | 0 | 0 | 0 | 2 | -2-M | 1 | -20 |

possiamo così continuare effettuando il test di ottimalità, ed in seguito identificando variabili entranti ed uscenti. L'algoritmo può così continuare come visto in precedenza, in quanto la variabile artificiale non tornerà più in base. Si giunge dunque ad una soluzione ottima del tipo $y_1 = \frac{-8}{21} \quad x_1 = \frac{-1}{21} \quad x_2 = -M$.

Alla fine utilizzando le variabili artificiali come soluzioni basiche ammissibili, stiamo ipotizzando di lavorare su di uno spazio esteso, talvolta in più dimensioni. Dunque possiamo immaginare di star percorrendo un poligono lungo i suoi lati, individuando un set di soluzioni ottime, di cui solo una sottoparte sarà corrispondente a soluzioni dello spazio originale.

Implementare l'algoritmo del big M tuttavia non è una cosa elementare, in quanto dato l'alto valore in modulo di M, rischiamo di incorrere in errori di tipo numerico, quali errori di round off e overflow. Dunque presentiamo un secondo metodo risolutivo chiamato **Metodo delle due fasi**. Ci poniamo sempre l'obiettivo di determinare una prima soluzione basica ammissibile e eliminare le variabili artificiali dalla soluzione del problema.

Si costituisce dunque una *nuova funzione obiettivo ausiliaria* **w**, definita come la somma delle variabili artificiali.

$$w = \sum_{i=1 \cdots k} h_i$$

la funzione obiettivo deve essere sempre *minimizzata.* se il minimio della funzione w è pari a 0, raggiungiamo l'annullamento di tutte le variabili artificiali. Naturalmente aggiungere la riga w al sistema, ci fa perdere la forma canonica di equazioni; tuttavia se si esprime w in funzione delle variabili originarie, il sistema resta in forma canonica rispetto alle variabili artificiali.
Per comprendere al meglio tale metodo, come al solito procederemo con un esempio:

$$\begin{cases} \max z = x_1 + 2x_2 \\ x_1 + y_1 = 6 \\ x_2 + y_2 = 4 \\ 2x_1 + 5_x 2 - y_3 + h_1 = 10 \\ 4x_1 + 2x_2 - y_4 + h_2 = 8 \end{cases}$$

Andiamo dunque ora a riscrivere le variabili artificiali in funzione delle variabili del problema e delle variabili slack, così da poter esprimere w come funzione delle suddette variabili.

$$w = h_1 + h_2 = -6x_1 - 7x_2 + y_3 + y_4 + 18$$

Siamo ora pronti a costruire la tabella. Questa prima parte della risoluzione prende il nome di **Prima Fase**.

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $h_1$ | $h_2$ | -z | -w | b |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $y_1$ | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 |
| $y_2$ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |
| $h_1$ | 2 | 5 | 0 | 0 | -1 | 0 | 1 | 0 | 0 | 0 | 10 |
| $h_2$ | 4 | 2 | 0 | 0 | 0 | -1 | 0 | 1 | 0 | 0 | 8 |
| $-z$ | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $-w$ | -6 | -7 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | -18 |

Dunque la prima soluzione basica ammissibile si trova in corrispondenza di z = 0 e w = 18. Possiamo identificare la variabile entrante, ovvero $x_2$.La variabile uscente invece sarà relativa ad $h_2$. Continuando così è possibile far uscire dalla base entrambe le variabili artificiali, portandoci così ad una soluzione che preveda in base solo le variabili slack e quelle del problema. La soluzione basica ammissibile appena ottenuta è la *soluzione ottima rispetto a w*, che infatti risulterà essere w = 0.

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $h_1$ | $h_2$ | -z | -w | b |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $y_1$ | 0 | 0 | 1 | 0 | $\frac{-1}{8}$ | $\frac{5}{16}$ | $\frac{1}{8}$ | $\frac{-5}{16}$ | 0 | 0 | $\frac{19}{4}$ |
| $y_2$ | 0 | 0 | 0 | 1 | $\frac{1}{4}$ | $\frac{-1}{8}$ | $\frac{-1}{4}$ | $\frac{1}{8}$ | 0 | 0 | $\frac{5}{2}$ |
| $x_2$ | 0 | 1 | 0 | 0 | $\frac{-1}{4}$ | $\frac{1}{8}$ | $\frac{1}{4}$ | $\frac{-1}{8}$ | 0 | 0 | $\frac{3}{2}$ |
| $x_1$ | 1 | 0 | 0 | 0 | $\frac{1}{8}$ | $\frac{5}{16}$ | $\frac{-1}{8}$ | $\frac{5}{16}$ | 0 | 0 | $\frac{5}{4}$ |
| $-z$ | 0 | 0 | 0 | 0 | $\frac{1}{8}$ | $\frac{3}{16}$ | $\frac{-1}{8}$ | $\frac{-3}{16}$ | 1 | 0 | $\frac{-11}{4}$ |
| $-w$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

Con questo passo termina la prima fase ed è dunque possibile cancellare le colonne $h_1$,$h_2$, -w e la riga -w.

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | -z | b |
|---|---|---|---|---|---|---|---|---|
| $y_1$ | 0 | 0 | 1 | 0 | $\frac{-1}{8}$ | $\frac{5}{16}$ | 0 | $\frac{19}{4}$ |
| $y_2$ | 0 | 0 | 0 | 1 | $\frac{1}{4}$ | $\frac{-1}{8}$ | 0 | $\frac{5}{2}$ |
| $x_2$ | 0 | 1 | 0 | 0 | $\frac{-1}{4}$ | $\frac{1}{8}$ | 0 | $\frac{3}{2}$ |
| $x_1$ | 1 | 0 | 0 | 0 | $\frac{1}{8}$ | $\frac{5}{16}$ | 0 | $\frac{5}{4}$ |
| $-z$ | 0 | 0 | 0 | 0 | $\frac{1}{8}$ | $\frac{3}{16}$ | 1 | $\frac{-11}{4}$ |

Inizia ora la **Seconda fase**, che coincide perfettamente con l'algoritmo presentato precedentemente nel caso di minore uguale. Procedendo quindi con l'identificazione di variabili entranti/uscenti è possibile giungere alla soluzione ottima

$$y_3 = 22 \quad x_2 = 4 \quad y_4 = 24 \quad x_1 = 6 \quad y_1 = 0 \quad y_2 = 0$$

### 2.7.3 Circolazione e soluzioni degeneri

Con questo possiamo considerare conclusa la trattazione basilare sull'algoritmo del simplesso. Cerchiamo ora di analizzare però il caso di **soluzione basica ammissibile degenere.**

Una soluzione basica ammissibile degenere è una *soluzione* che ha più di n-m variabili uguali a zero, e ha quindi meno di m variabili positive. In caso di *degenerazione* uno o più termini noti $b'_i$ sono nulli e il risultato dell'operazione di pivoting sarà il passaggio ad una nuova soluzione basica in cui la variabile entrante entra in base con valore nullo. Può infatti capitare, nella tabella del simplesso, che uno dei termini noti sia nullo. In questo caso parliamo di soluzione basica ammissibile degenere, in cui almeno un delle variabili in base assume valore zero.

Da un punto di vista geometrico, ciò si traduce nell'avere una sovrapposizione di soluzioni basiche ammissibili su di uno stesso vertice. In figura possiamo infatti vedere un dominio definito a partire da due vincoli di minore uguale che individuano i segmenti BC e CD, ed uno di maggiore uguale che individua il segmento AD.
Una soluzione basica ammissibile è in questo caso una soluzione che abbia almeno 2 variabili uguali a zero e 3 maggiori o uguali di zero. Nel vertice D tuttavia la variabile slack individuata dal segmento CD,

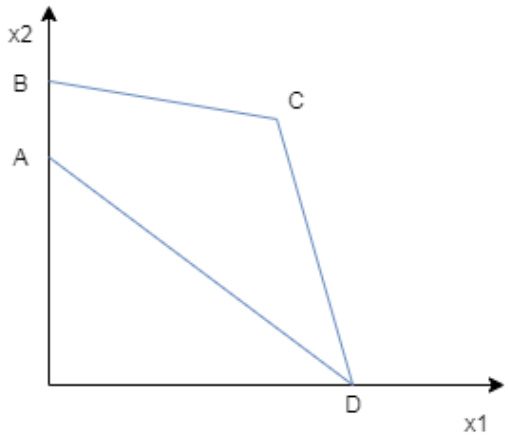



Figure 12: Esempio: Dominio di ammissibilità nel caso di soluzioni degeneri

oltre che quella individuata dal segmento AD, sarà nulla. Ma anche la variabile $x_2$ sarà nulla, poichè il punto D si trova sulla frontiera dei due segmenti, ma anche sull'asse x1.
Allora avremo solo 2 variabili strettamente positive! ciò implica che nel vertice D sono associate più soluzioni basiche ammissibili, e dunque *degeneri*.

perciò se nel corso dell'algoritmo si determina una soluzione degenere, c'è la possibilità che dopo un certo numero di soluzioni degeneri si passi una seconda volta per una soluzione già *generata in precedenza*
tale fenomeno prende il nome di **circolazione**, ed è una situazion in cui l'algoritmo non riesce a convergere a soluzione ottima. è dunque necessario stabilire delle regole di *anticircolazione* che eliminino questo problema catastrofico; tale scelte, come vedremo di qui a poco, si baseranno sullo scegliere opportunamente le variabili entranti e uscenti dalla base.

La regola di **Bland**, attraverso una scelta diversa delle variabili da far entrare e uscire dalla base, consente di evitare *possibili loop*. Tra le variabili non in base con *coefficiente di costo modificato negativo* (o positivo in caso di problema a massimizzare), si scegliere la variabile con indice di colonna più basso, e dunque la prima che si incontra scorrendo il vettore da sinistra verso destra.

Può capitare che il valore minimo del rapporto:

$$\min\left\{\frac{b_i}{a_{ij}} \quad \forall i \in 1, \cdots, m : a_{ij} > 0\right\}$$

si ottenga in corrispondenza di *più valori di indice i*, e allora in tal caso si sceglie come *variabile uscente* quella relativa alla riga di indice più basso.

### 2.7.4 Revisione dell'algoritmo del Simplesso

Immaginiamo di star risolvendo un problema molto grande, ovvero con molte variabili, e di volere applicare l'algoritmo del simplesso. Quando dobbiamo selezionare una variabile entrante, andiamo a consultare il vettore dei coefficienti modificati. A questo punto, stabilito il coefficiente i-esimo, si seleziona la colonna i-esima relativa alla variabile entrante. L'altro passo fondamentale consiste nello scegliere la variabile uscente, ma per far questo è necessario conoscere solo i valori della colonna dei termini noti, e della variabile entrante.

Successivamente viene effettuato il *pivotin* in maniera tale da avere la matrice identità in corrispondenza delle nuove variabili in base. Quindi le uniche informazioni utili all'algoritmo saranno l'ultima riga, e i due vettori colonna dei termini noti e della variabile entrante, e poi naturalmente le variabili in base.

Lavorando solo su queste informazioni si riesce a risparmiare molto in termini di calcolo, ottimizzando di fatto l'algoritmo così come presentato in precedenza. Diventa a questo punto però fondamentale ricavarsi queste tre informazioni nella maniera più efficiente possibile. Formalizziamo i tre vettori da dover trovare:

$$c'_{nb} = [c_{1nb}, c_{2nb}, \cdots, c_{(m-n)nb}]$$
$$b' = [b_1, b_2, \cdots, b_m]$$
$$p'_s = [a_{1s}, a_{2s}, \cdots, a_{ms}]$$

con $c'_{nb}$ *Coefficienti di costo modificati relativi alle variabili non basiche*, $b'$ *Vettore colonna dei termini noti* e $p'_s$ *Vettore dei coefficienti della colonna relativa alla variabile entrante.*

è allora necessario revisionare l'algoritmo e formalizzarlo, al fine di definire correttamente questi 3 vettori presentati. Sia $\hat{A}\hat{x} = \hat{b}$ il sistema $Ax = b$ con in aggiunta la riga della funzione obiettivo $cx - z = 0$. valgono allora le seguenti relazioni:

$$\hat{A} = \begin{bmatrix} A & 0 \\ c & 1 \end{bmatrix} \qquad \hat{x} = \begin{bmatrix} x \\ -z \end{bmatrix} \qquad \hat{b} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

Cerchiamo ora di configurare il sistema mettendo in evidenza le componenti basiche e le componenti non basiche relative ad un generico passo algoritmico.

$$A = \begin{bmatrix} N_B & B \end{bmatrix} \qquad \hat{x} = \begin{bmatrix} x_{nb} \\ x_b \end{bmatrix} \qquad c = \begin{bmatrix} c_{nb} & c_b \end{bmatrix}$$

Quindi il sistema $\hat{A}\hat{x} = \hat{b}$ potrà essere riscritto come:

$$\begin{bmatrix} N_B & B & 0 \\ c_{nb} & c_b & 1 \end{bmatrix} \begin{bmatrix} x_{nb} \\ x_b \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

Facciamo ora ulteriori considerazioni sul risultato appena ottenuto; siano

$$\hat{N_B} = \begin{bmatrix} N_B \\ c_{nb} \end{bmatrix} \qquad \hat{B} = \begin{bmatrix} B & 0 \\ c_b & 1 \end{bmatrix}$$

possiamo individuare l'inverso della matrice $\hat{B}$ come:

$$(\hat{B})^{-1} = \begin{bmatrix} B^{-1} & 0 \\ -c_n B^{-1} & 1 \end{bmatrix}$$

Possiamo perciò riscrivere il sistema $\hat{A}\hat{x} = \hat{b}$ presentato in precedenza come:

$$\begin{bmatrix} \hat{N_B} & \hat{B} \end{bmatrix} \begin{bmatrix} x_{nb} \\ x_b \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

Completiamo i passaggi *premoltiplicando* al sistema la matrice $\hat{B}$ inversa, ottenendo:

$$(\hat{B})^{-1}\hat{N_B}x_{nb} + (\hat{B})^{-1}\hat{B}x_b = (\hat{B})^{-1} \begin{bmatrix} b \\ 0 \end{bmatrix}$$

ma per definizione il prodotto della matrice $\hat{B}$ e la sua inversa, restituisce la matrice identità. otteniamo dunque il seguente sistema:

$$(\hat{B})^{-1}\hat{N_B}x_{nb} + I_{m+1}x_b = (\hat{B})^{-1}\begin{bmatrix} b \\ 0 \end{bmatrix}$$

Essendo

$$(\hat{B})^{-1}\hat{N_B} = \begin{bmatrix} B^{-1}N_B \\ c_{nb} - c_b B^{-1}N_B \end{bmatrix}$$

$$(\hat{B})^{-1}\begin{bmatrix} b \\ 0 \end{bmatrix} = \begin{bmatrix} B^{-1}b \\ -c_b B^{-1}b \end{bmatrix}$$

Il sistema può essere in conclusione riscritto come:

$$\begin{bmatrix} B^{-1}N_B \\ c_{nb} - c_b B^{-1}N_B \end{bmatrix} x_{nb} + I_{m+1}x_b = \begin{bmatrix} B^{-1}b \\ -c_b B^{-1}b \end{bmatrix}$$

Il sistema appena ottenuto è il sistema in forma canonica che ci permette di ottenere i tre vettori rilevati per lo sviluppo dell'algoritmo del simpesso. Infatti il vettore dei coefficienti di costo modificati, relativi alle variaibli non basiche, può essere calcolato come:

$$c'_{nb} = c_{nb} - c_b B^{-1}N_B \implies c'_j = c_j - c_b B^{-1}p_j$$

Il *Vettore della colonna dei termini noti*

$$b' = B^{-1}b$$

e il *Vettore dei coefficienti della colonna relativa alla variabile entrante*

$$p'_s = B^{-1}p_s$$

ciò significa che ruolo fondamentale acquisisce l'inversa della matrice B, la cui premoltiplicazione realizza di fatto l'operazione di pivoting; tale matrice va calcolata ad ogni iterazione dell'algoritmo, in maniera tale da poter ricavare, iterazione per iterazione, i tre vettori richiesti. Diventa quindi ora fondamentale il calcolo della matrice inversa di B

Consideriamo il sistema in forma canonica *senza* la funzione obiettivo, in cui le variabili di base sono relative alle *ultime colonne della matrice*. abbiamo allora che:

$$\begin{bmatrix} A & I \end{bmatrix} x = b$$

per ottenere un sistema equivalente in forma canonica rispetto ad un insieme diverso di variabili di base, occorre premoltiplicare per l'inversa della matrice di base, ovvero B.

$$B^{-1} \begin{bmatrix} A & I \end{bmatrix} x = B^{-1}b \implies \begin{bmatrix} D & B^{-1} \end{bmatrix} x = B^{-1}b$$

con $D = B^{-1}A$. è allora chiaro che ad ogni iterazione dell'algoritmo, la matrice $B^{-1}$ si trovi in corrispondenza delle ultime m colonne *corrispondenti alla mtrice identità iniziale*.

Proceduiamo adesso illustrando un esempio finale sull'algoritmo del Simplesso revisionato. Immaginiamo di voler risolvere il seguente problema a massimizzare:

$$\max z = x_1 + 2x_2 - x_3 + x_4 + 4x_5 - 2x_6$$

$$\begin{cases} x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 6 \\ 2x_1 - x_2 - 2x_3 + x_4 \leq 4 \\ x_3 + x_4 + 2x_5 + x_6 \leq 4 \end{cases}$$

Come prima cosa deve essere scritta la prima tabella dell'algoritmo del simplesso. In corrispondenza delle variabili in base, ovvero $x_7, x_8, x_9$ (var slack), avremo la matrice identità. Perciò l'inversa di questa matrice sarà la matrice identità stessa. (B inversa).

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | -z | b |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_7$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 6 |
| $x_8$ | 2 | -1 | -2 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| $x_9$ | 0 | 0 | 1 | 1 | 2 | 1 | 0 | 0 | 1 | 0 | 4 |
| $-z$ | 1 | 2 | -1 | 1 | 4 | -2 | 0 | 0 | 0 | 1 | 0 |

In questo caso, consideriamo solo la colonna $x_5$ relativa alla variabile entrante, la colonna dei termini noti b e la riga dei coefficienti di costo modificati, oltre che la matrice B delle variabili in base. Dunque effettuiamo il pivoting sulla variabile entrante e modifichiamo quindi la matrice B. Così facendo siamo in grado di ottenere la segiemte tabella:

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | -z | b |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_7$ | | | | | 0 | | 1 | 0 | $-\frac{1}{2}$ | | |
| $x_8$ | | | | | 0 | | 0 | 1 | 0 | | |
| $x_9$ | | | | | 1 | | 0 | 0 | $\frac{1}{2}$ | | |
| $-z$ | | | | | | | | | | | |

Notiamo che presa la nuova base $\{x_7 \quad x_8 \quad x_5\}$, la nuova matrice B di base, costruita prendendo in esame le colonne relative alla nuova base dalla prima tabella, diventerebbe:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

L'inversa di questa matrice, è la matrice che otteniamo ora , nella nuova tabella, in corrispondenza della precedente matrice di base

$$B^{-1} = \begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix}$$

Calcoliamo ora i coefficienti di costo modificati, secondo la formula presentata in precedenza:

$$c'_{nb} = c_{nb} - c_b B^{-1} N_B \implies c'_j = c_j - c_b B^{-1} p_j$$

Dalla tabella iniziale possiamo infatti considerare che i valori $c_b = [c_7, c_8, c_5] = [0; \quad 0; \quad 4]$. Il vettore $c_{nb}$ sarà invece composto da tutti i coefficienti delle variabili non in base, e quindi $[1; 2; -1; 1; -2; 0]$. Successivamente avremo bisogno della matrice $N_B$, composta dai vettori $p_j$ delle variabili non in base. Ottenendo così la matrice:

$$N_B = [p_1, p_2, p_3, p_4, p_6, p_9] = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 2 & -1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

data quindi la matrice B inversa, presentata prima, possiamo calcolare il nuovo $c'_{nb}$ secondo la formula, ottenendo:

$$c'_{nb} = [c'_1, c'_2, c'_3, c'_4, c'_6, c'_9] =$$

$$[1; 2; -1; 1; -2; 0] - [0; 0; 2] \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 2 & -1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} = [1; 2; -3; -1; -4; -2]$$

Preso il coefficiente non basico più grande, ovvero 2, determiniamo la nuova variabile entrante, ovvero $x_2$. Calcoliamo quindi il vettore dei coefficienti della colonna relativa alla variabile entrante, ovvero $p'_s = B^{-1}p_s$. Dato il vettore $p_2 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$, e B inversa la stessa di prima, ottenendo il vettore della nuova variabile in base $p'_2$, come:

$$p'_2 = \begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

L'ultima informazione necessaria sarà il vettore dei termini noti b, che sarà fondamentale per determinare la variabile uscente. Ricordando che tale vettore può essere ottenuto come $b' = B^{-1}b$, vale che:

$$b' = \begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 6 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 2 \end{bmatrix}$$

Abbiamo quindi con successo determinato tutte le informazioni necessarie, ricapitolate nella tabella:

| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | -z | b |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_7$ | | 1 | | | 0 | | 1 | 0 | $-\frac{1}{2}$ | | 4 |
| $x_8$ | | -1 | | | 0 | | 0 | 1 | 0 | | 4 |
| $x_9$ | | 0 | | | 1 | | 0 | 0 | $\frac{1}{2}$ | | 2 |
| $-z$ | 1 | 2 | -3 | -1 | | -4 | | | -2 | | |

è chiaro ora che la variabile uscente sarà la slack $x_7$. A questo punto è possibile reiterare il ragionamento mostrato fino ad ora, consci del fatto che la nuova variabile intorno alla quale fare pivoting sarà $x_2$. Sarà quindi necessario, una volta effettuato il pivoting, stabilire l'entità della nuova matrice di base B inversa.
Date perciò le variabili in base $\{x_2 \quad x_8 \quad x_5\}$, otteniamo come matrice di base dalla prima tabella:

$$\begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

mentre come nuova matrice di base inversa:

$$\begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 1 & 1 & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} \end{bmatrix}$$

È possibile adesso ricostruirsi i vettori $c'_{nb}$, $c'_b$ e la matrice $N_B$ come fatto nel caso precedente. si perviene così al vettore dei coefficienti di costo modificati:

$$c'_{nb} = [c'_1, c'_3, c'_4, c'_6, c'_7, c'_9]$$

Notiamo che dati i coefficienti ottenuti, essendo il problema a massimizzare, siamo giunti alla soluzione ottima. Dunque l'algoritmo è arrivato al suo ultimo passo, ma devo necessariamente calcolarmi la colonna b per ottenere il risultato desiderato, ovvero i valori delle variabili in base, per poi calcolare il valore della funzione obiettivo z.

$$b' = B^{-1}b = \begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 1 & 1 & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 6 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 2 \end{bmatrix}$$

$$-z = -c_b B^{-1} b = -[2; 0; 4] \begin{bmatrix} 1 & 0 & -\frac{1}{2} \\ 1 & 1 & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 6 \\ 4 \\ 4 \end{bmatrix} = -16$$

la soluzione ottima sarà quindi data da:

$$x_2 = 4 \quad x_8 = 8 \quad x_1 = x_3 = x_4 = x_6 = x_7 = x_9 = 0 \quad z = 16$$

Sarebbe possibile, in caso di molti vincoli, trasformare il problema in un corrispondente problema duale, riducendo il numero di vincoli aumentando il numero di variabili. Tuttavia non tratteremo questa problematica.

## 2.8 Analisi post ottimale

Fino ad ora abbiamo risolto problemi di programmazione lineare in presenza di un unico decisore, un unico obiettivo e assenza di incertezza nelle grandezze in gioco. Tuttavia nella maggioranza dei

problemi reali, tali dati possono presentare un certo grado di *aleatorietà*. Ci chiediamo quanto possa esser stabile un problema, ovvero se data la variazione di parametri di input, la soluzione resti ottima o meno.

Dunque l'**Analisi di stabilità** determina i campi di variazione dei parametri che consentono di mantenere inalterata la configurazione della soluzione basica ammissibile ottima determinata. L'**Analisi parametrica** invece studia l'andamento della soluzione al variare dei parametri oggetto di studio.

Dunque con l'analisi post ottimale trattiamo due possibili campi di indagine : come cambia la soluzione al *variare dei termini noti*, e come cambia la soluzione al *variare del valore dei coefficienti di costo della funzione obiettivo*. Si potrebbero analizzare anche i coefficienti della matrice dei vincoli, ma è una tecnica poco usata in quanto più complessa.

Dato un problema di programmazione lineare scritto in forma standard, sia B una *base ottima ammissibile*, ovvero tale che $x_b = B^{-1}b$. Possiamo dunque determinare la **Condizione di ammissibilità** come:

$$B^{-1}b \geq 0$$

e al contempo possiamo stabilire la **Condizione di ottimalità** per la base B come:

$$c'_j = c_j - c_b B^{-1} p_j \leq 0 \quad \forall j \text{non basico}$$

Dunque variazioni dei valori dei termini noti incidono sulla condizione di ammissibilità, mentre variazioni sui valori dei coefficienti della funzione obiettivo, influenzano la condizione di ottimalità.

Facciamo un esempio di analisi di stabilità: prendiamo in esame il seguente problema:

$$\max z = 11x_1 + 10x_2$$

$$\begin{cases} -5x_1 + 4x_2 \leq 20 \\ x_2 \leq 6.5 \\ 3x_1 + 4x_2 \leq 36 \; 2x_1 + x_2 \leq 16 \end{cases}$$

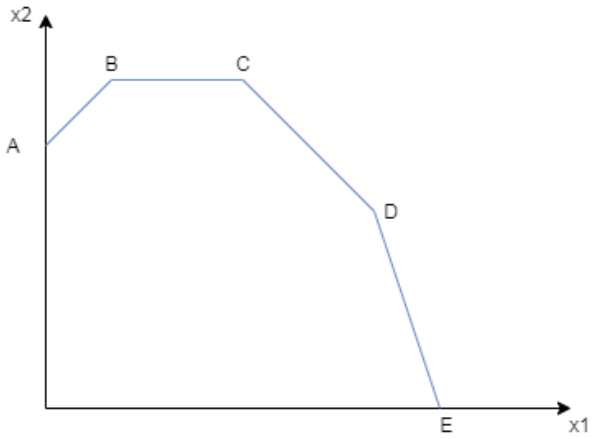



Figure 13: Dominio di ammissibilità per Analisi post ottimale

Graficamente possiamo subito individuare la soluzione basica ammissibile ottima. Al momento non ci interessa sapere esattamente il valore delle variabili e dei termini noti nel punto D, ma ci interessa solo comprendere cosa accade al variare dei valori dei termini noti. Ipotizziamo infatti che il termine noto $b_3$ sia affetto da errore.
Questo si traduce nella trasformazione del dominio di ammissibilità, che andrà ad individuare una *nuova regione*, che presenterà un nuovo vertice D', differente dal precedente D.
La base tuttavia rimarrà la stessa in termini di variabili. Una perturbazione sufficiente potrebbe farmi ottenere anche una soluzione degenere, come è possibile vedere in figura. Se la perturbazione dovesse ancora aumentare la composizione delle variabili di base andrebbero a cambiare.

Dunque all'aumentare di $b_3$, fino al valore limite in corrispondenza del quale il vincolo diventa ridondante, la configurazione della s.b.a ottima *resta immutata*.
È naturalmente anche possibile che il termine $b_3$ diminuisca. Anche qui però la composizione delle variabili in base non va a cambiare.Se si prosegue nel far decrescere $b_3$, il terzo vincolo arriverà ad esser rappresentato dal segmento BD, facendo passare la soluzione ottima a D"'. La composizione di base nuovamente resterà la stessa, fino ad un certo punto limite.
Infatti la configurazione della s.b.a ottima resta immutata al diminuire

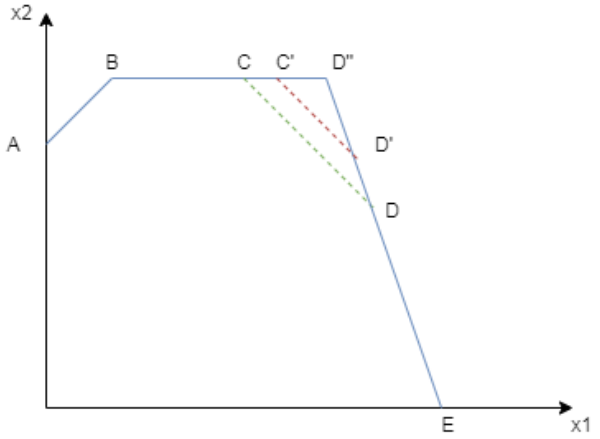



Figure 14: Dominio di ammissibilità con variazione positiva sui termini noti

di $b_3$ dino ad un valore limite posto in corrispondenza del vertice E.

Se provassimo a diminuire il valore del termine noto $b_1$, lo spigolo A-B si sposterebbe verso il basso. In corrispondenza di una posizione tale che il segmento AB si trova a sovrapporsi a quello AC, il vincolo relativo a BC dventerebbe ridondante, tuttavia la configurazione delle s.b.a ottima rimarrebbe *invariata*.
Tuttavia se il termine $b_1$ continua a diminuire fino a raggiungere e superare la posizione limite di un segmento AD, rendendo ridondante anche il vincolo 3 relativo al segmento CD, allora la configurazione s.b.a ottima *varierebbe*.

Consideriamo perciò il caso in cui un unico termine noto $b_i$ incrementa il suo valore di una quantità pari a $\Delta b_i^+$, con $(b_i^+ = b_i + \Delta b_i^+)$. Il valore della soluzione basica ammissibile varia secondo la seguente espressione:

$$x_b^+ = B^{-1}[b + \Delta b_i^+ u_i]$$

Dove $u_i$ è l'i-esima colonna della matrice identità. Sviluppando i calcoli, e ponendo $B^{-1}u_i = \beta_i$, possiamo stabilire che :

$$x_b^+ = B^{-1}b + \Delta b_i^+ \beta_i$$

Ne consegue dunque che una soluzione basica ammissibile **resta ottima** fintanto che

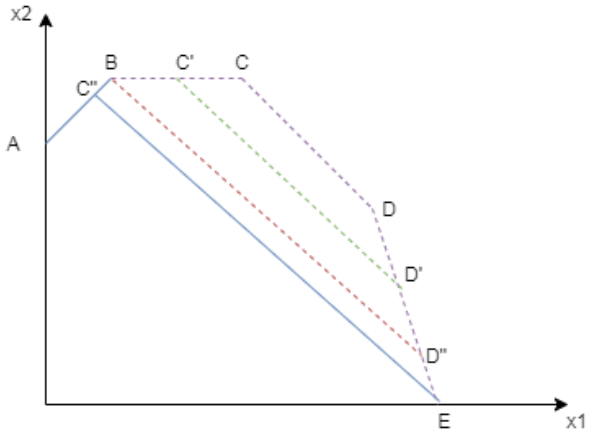



Figure 15: Dominio di ammissibilità con variazione negativa sui termini noti

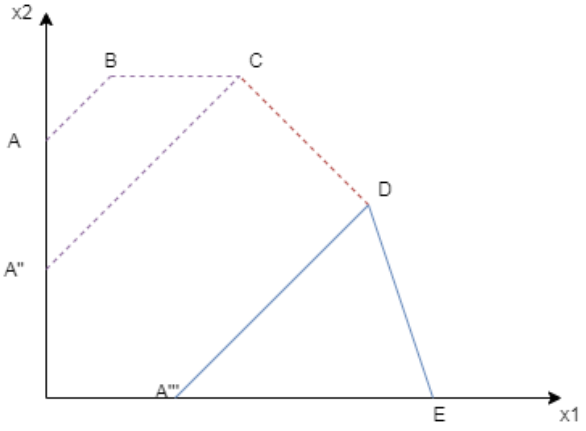



Figure 16: Dominio di ammissibilità con perturbazione sul termine noto $b_1$

$$x_b^+ = B^{-1}b + \Delta b_i^+ \beta_i \geq 0$$

possiamo quindi stabilire il massimo valore di perturbazione positivo sul generico termine noto tale da mantenere l'ottimalità della soluzione:

$$x_{kb} + \Delta b_i^+ \beta_{ki} \geq 0 \qquad k = 1 \cdots m$$

$$\max(\Delta b_i^+) = \min_k \left[ \frac{x_{kb}}{-\beta_{ki}} \right]$$

Discorso analogo può esser fatto per il caso di perturbazione negativa $\Delta b_i^-$ Accenniamo al fatto che la colonna $\beta_i$ sarà quella relativa alla variabile slack $y_i$ se i è un vincolo di tipo $\leq$, altrimenti sarà relativa alla variabile artificiale $h_i$ in caso di tipo $=$ o $\geq$

Facciamo ora un esempio e analizziamo la stabilità di un problema.

| | $x_1$ | $x_2$ | $y_1$ | $y_2$ | $y_3$ | $y_4$ | $-z$ | $b$ |
|---|---|---|---|---|---|---|---|---|
| $y_1$ | 0 | 0 | 1 | 0 | -2.6 | 6.4 | 0 | 28.8 |
| $y_2$ | 0 | 0 | 0 | 1 | -0.4 | 0.6 | 0 | 1.8 |
| $x_2$ | 0 | 1 | 0 | 0 | 0.4 | -0.6 | 0 | 4.8 |
| $x_1$ | 1 | 0 | 0 | 0 | -0.2 | 0.8 | 0 | 5.6 |
| $-z$ | 0 | 0 | 0 | 0 | -1.8 | -2.8 | 0 | -109.6 |

Possiamo immediatamente notare che la tabella è già stata posta in maniera tale da mostrare la soluzione ottima:

$$y_1 = 28.8 \quad y_2 = 1.7 \quad x_2 = 4.8 \quad x_1 = 5.6 \quad y_3 = y_4 = 0 \quad z = 109.6$$

A questo punto possiamo determinare la generica perturbazione positiva/negativa del termine noto come:

$$\Delta b_i^+ = \min_k \left[ \frac{x_{kb}}{-\beta_{ki}} \right] \quad \forall \beta_{ki} < 0$$

$$\Delta b_i^- = \min_k \left[ \frac{x_{kb}}{-\beta_{ki}} \right] \quad \forall \beta_{ki} > 0$$

è quindi a questo punto possibile calcolare le massime e le minime perturbazione sui termini noti tali da far rimanere il problema "stabile". Facciamo l'esempio del calcolo di $\Delta b_3^+$ e $\Delta b_3^-$.

$$\Delta b_3^+ = min\left[\frac{28.8}{2.6}; \frac{1.7}{0.4}; \frac{5.6}{0.2}\right] = 4.25$$

$$\Delta b_3^- = min\left[\frac{4.8}{0.4}\right] = 12$$

Andiamo adesso ad analizzare la stabilità sui **coefficienti di costo**. Ipotizziamo di avere il seguente problema, caratterizzato dal dominio di ammissibilità in figura:

$$\max z = 11x_1 + 10x_2$$

$$\begin{cases} -5x_1 + 4x_2 \leq 20 \\ x_2 \leq 6.5 \\ 3x_1 + 4x_2 \leq 36 \ 2x_1 + x_2 \leq 16 \end{cases}$$

Come prima ipotizziamo che il problema ammetta soluzione ottima nel vertice D. ci chiediamo come varia la composizione della soluzione basica ammissibile ottima al variare del valore del coefficiente di costo $c_1$.
Se per esempio il coefficiente $c_1$ aumenta, la direzione della funzione obiettivo *ruota* in senso orario, ovvero si ottiene una rotazione delle linee di livello. Può quindi capitare che le linee di livello diventino *parallele* alla front
iera del dominio di ammissibilità. Fintanto che questo non accade, il vertice D *resta ottimo*. Per valori di spostamento superiori la soluzione ottima invece si sposta nel vertice E.

Se invece il coefficiente *diminuisce*, le linee di livello ruotano in senso *antiorario*. Analogamente a prima la soluzione D rimane quella ottima fintanto che la linea di livello non diventa parallela al segmento CD, fino a far diventare C la soluzione ottima.

Perciò l'entità di variazione, positiva o negativa, di $c_1$ è proporzionale all'ampiezza dell'angolo tra la direzione della funzione obiettivo e lospigolo al quale essa va a diventare parallela. Naturalmente questo discorso può essere applicato anche agli altri coefficienti di costo $c_i$.

Vogliamo quindi determinare il **campo di variazione** di un generico coefficiente di costo che mantenga *inalterata* la configurazione della soluzione basica ammissibile ottima. Ricordiamo che dovrà esser rispettata la condizione di ottimalità :

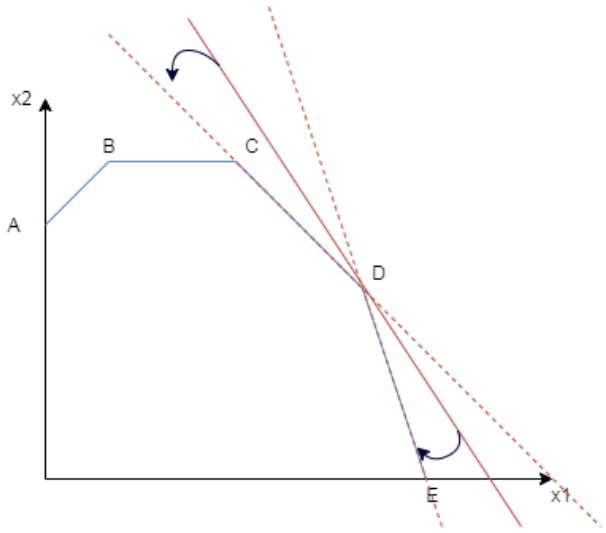



Figure 17: Dominio di ammissibilità con perturbazione sul coefficiente di costo

$$c'_j = c_j - c_b B^{-1} p_j \leq 0 \quad \forall j \text{non basico}$$

Mentre definiamo il campo di variazione come la minima/massima perturbazione applicabile al coefficiente di costo, tale che la soluzione basica ammissibile ottima resti tale.

$$c_k = c_k \pm \Delta c_k^-$$

Distinguiamo due possibili casi nello scenario di problema a massimizzare. Nel primo ipotizziamo che la variabile $x_k$ non sia basica e dunque all'ottimo possono valere le seguenti condizioni:

$$c_k^* = (c_k + \Delta c_k^+) - c_b B^{-1} p_k \leq 0$$
$$\Delta c_k^+ = c_b B^{-1} p_k - c_k \implies \Delta c_k^+ = -c'_k$$

Nel caso di perturbazione negativa, capita invece che:

$$c_k^* = (c_k - \Delta c_k^-) - c_b B^{-1} p_k \leq 0 \implies \Delta c_k^- = \infty$$

perciò se nella soluzione ottima la variabile $x_k$ non è in base, diminuendo il coefficiente di costo relativo, risulterà sempre conveniente

lasciare la variabile a 0. nel secondo caso invece ipotizziamo che la variabile $x_k$ sia effettivamente basica. Data la condizione di ottimo:

$$c_j^* = c_j - c_b^* B^{-1} p_j \leq 0 \quad \forall j \text{non basico}$$
$$c_j - c_b p'_j - \Delta c_k^+ u_k p'_j \leq 0$$
$$c'_j - \Delta c_k^+ a'_{kj} \leq 0$$

il *valore limite* di $\Delta c_k^{\pm}$ sarà quindi dato da:

$$\Delta c_k^+ = \min_j \left\{ \frac{-c'_j}{-a'_{kj}} \right\} \quad \forall a'_{kj} < 0 \quad j \text{ non basico}$$

$$\Delta c_k^- = \min_j \left\{ \frac{-c'_j}{a'_{kj}} \right\} \quad \forall a'_{kj} > 0 \quad j \text{ non basico}$$

sviluppando quindi l'esempio, tabulandolo come visto in precedenza, possiamo calcolare la generica perturbazione massima/minima come appena mostrato. A titolo esemplificativo mostriamo il calcolo della perturbazione su $c_1$:

$$\Delta c_1^+ = \frac{1.8}{0.2} = 9$$
$$\Delta c_1^- = \frac{2.8}{0.8} = 3.5$$

Individuiamo quindi il campo di variazione:

$$[c_1^- = c_1 - \Delta c_1^-; c_1^+ = c_1 + \Delta c_1^+] = [7.5; 20]$$

# 3 Programmazione Lineare Intera

In problemi di programmazione lineare intera consideriamo un insieme di variabili e una funzione obiettivo lineare, della quale vogliamo calcolare una soluzione. L'insieme delle soluzioni ammissibili è pero un sottoinsieme S di $\mathscr{Z}^n$, ovvero un sottoinsieme di vettori le cui componenti sono *valori interi*.

È necessario costruire un insieme di vincoli che definiscano un poliedro P che contenga tutte e sole le soluzioni ammissibili del problema di programmazione lineare intera presa in considerazione. parliamo allora di **formulazione lineare** del problema come un insieme

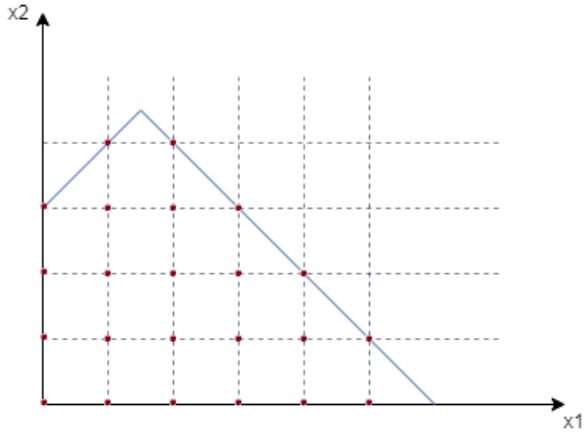



Figure 18: Dominio di ammissibilità P di un problema PLI

di vincoli lineari che ci permettono di separare Il dominio di ammissibilità S da tutti gli altri vettori interi

$$P = \{x \in \mathscr{R}^n : Ax \leq b\}$$
$$S = P \cap \mathscr{Z}^n$$

## 3.1 Branch and Bound

Immaginiamo di avere un problema in due variabili e che l'insieme S, ovvero il dominio di ammissibilità discreto, sia costituito dai punti in figura.

$$\max -x_1 + 2x_2$$
$$\begin{cases} 4x_1 + 6x_2 \leq 9 \\ x_1 + x_2 \leq 4 \\ x_1, x_2 \geq 0 \\ x_1, x_2 \in \mathscr{Z} \end{cases}$$

Nel caso di problema non intero, la soluzione ottima sarebbe individuata in corrispondenza del vertice; tuttavia è possibile notare come questo punto di ottimalità non appartenga all'insiem S. Approcci risolutivi al problema possono essere di svariata natura. La prima cosa che possiamo considerare, è un **arrotondamento** : risolvo il problema continuo, ottenendo i valori di x che danno la soluzione per poi

arrotondare tali punti $x_i$ all'intero più vicino. Tuttavia questo processo non ci da alcuna*garanzia* di ottimalità o ammissibilità.

È possibile allora **Rafforzare** la formulazione, mediante l' algoritmo dei *piani di taglio*, che consiste in un miglioramento progressivo della formulazione, aggiungendo vincoli validi fino ad ottenere una soluzione del rilassamento ammissibile del problema originario.

Altra possibilità è quella **enumerativa**. La maggior parte dei problemi di PL intera hanno un numero di soluzioni ammissibili finite e quindi si potrebbe valutare la funzione obiettivo punto per punto per poi scegliere la soluzione migliore. Ovviamente l'enumerazione *totale* richiederebbe tempi di calcoli *improponibili*, ragion per cui si opta per una tecnica di enumerazione *intelligente*, ovvero enumerazione *implicita*, che prende il nome di **Branch and Bound**.

Per capire cosa si intenda per enumerazione implicita, possiamo inizialmente considerare il più semplice problema di PLI, detto problema dello zaino, o *knapsack problem.*
Ipotizziamo di avere un zaino da riempire con un insieme di oggetti. Non tutti gli oggetti entrano nello zaino, e ogni oggetto è caratterizzato da un peso e da un valore; è quindi necessario selezionare l'insieme di oggetti in maniera tale che lo zaino abbia il massimo valore possibile in termini di oggetti. Lo zaino ha una capacità massima b e ognuno degli i oggetti è caratterizzato da un valore$v_i$ e da un peso $p_i$. Va inoltre specificato che l'informazioe sarà *binaria*, in quanto un oggetto potrà appartenere oppure no allo zaino.
Possiamo quindi definire la funzione obiettivo come la massimizzazione del valore degli oggeti presenti nello zaino:

$$\max \sum_{i=1}^{n} v_i x_i$$

Per quanto ne concerne i vincoli, l'unico da soddisfare è che gli oggetti da mettere nello zaino non debbano superare la capacità dello zaino.

$$\sum_{i=1}^{n} p_i x_i \leq b \qquad x_i \in \{0,1\} \quad i = 1 \cdots n$$

Ipotizziamo di avere 3 oggetti A,B,C e schematizziamo il problema

come:

$$\max 40x_A + 20x_B + 15x_C$$
$$\begin{cases} 15x_A + 10x_B + 8x_C \leq 20 \\ x_A, x_B, x_C \in \{0, 1\} \end{cases}$$

L'oggetto A ha un peso di 15 e un valore di 40, B un valore di 20 e un peso di 10, C un valore di 15 e un peso di 8 e la capacità massima dello zaino è pari a 20.
Nel caso dell'enumerazione implicita posso enumerare per partizioni. Osservo che l'insieme iniziale è composto da $2^3$ possibili soluzioni. Bisognerebbe elencare infatti tutte le possibibili combinazioni binarie del problema, che conferiranno all' insieme S cardinalità pari a 8.
Dividiamo l'insieme su due sottoinsiemi, così da ridurre la dimensione del problema. Definisco S1 ed S2 partizioni dell'insieme S. la soluzione ottima in S è quindi data dalla migliore tra la soluzione ottima in S1 e la soluzione ottima in S2.
Per costruire le due partizioni, in S1 metterò tutti gli x tali che $x_A$ sia pari a 1, e dunque presente nello zaino, mentre invece S2 sarà composto da tutti gli x in S tali che A non appartenga allo zaino, ovvero $x_A = 0$.
Sarebbe possibile partizionare ancora S1 ed S2, reiterando il ragionamento. Il discorso termina quando si arriva a tutti i sottoinsiemi di cardinalità 1.

Dunque il problema può essere schematizzabile tramite un albero binario. Ad ogni foglia verrà però aggiunto il vincolo $x_A = 1$ o $x_A = 0$.
Prima osservazione che possiamo fare circa SA è che sicuramente L'oggetto A è presente nello zaino. Ciò significa che per il sottoproblema SA ho trovato un **Lower Bound** di 40, ovvero il limite inferiore rispetto alla variabile scelta. Lo zaino avrà infatti, posto A al suo interno, almeno un valore di 40 al momento.
In $S\overline{A}$ si può fare un altro ragionamento. Poiché sicuramente A non c'è nello Zaino, possiamo stabilire che l'**Upper bound** sarà 35 qualora inserissimo sia B che C. ma poiché il LowerBound di $SA >$ UpperBound in $S\overline{A}$, allora possiamo procedere solo con il sottoproblema SA.

Posso allora partizionare allo stesso modo $SA$, ipotizzando questa volta di inserire l'oggetto B, o assumere di non farlo. In questo caso

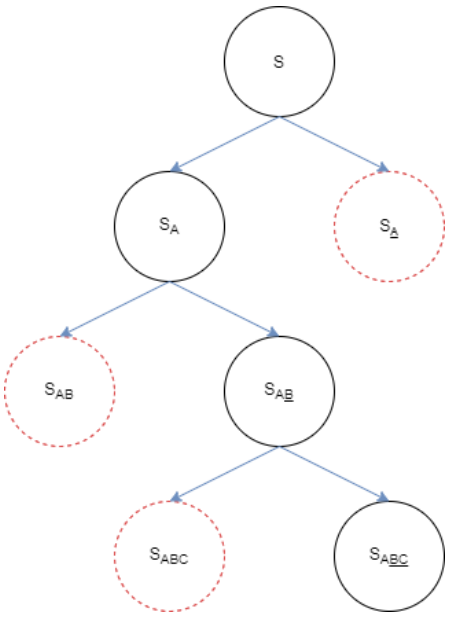



Figure 19: Grafo del problema dello zaino

però, inserito B vado a superare la capienza dello zaino, e quindi prenderò in consegna solo il sottoproblema $SA\overline{B}$.
Continuando troveremmo che aggiungere l'oggetto C farebbe eccedere la capacità dello zaino, portandoci all'unica soluzione ottima possibilea con $x_A = 1 \quad x_B = 0 \quad x_C = 0$. La soluzione ottima allora coincide con il Lower Bound trovato in precedenza.

Dunque il Branch and Bound cerca di rendere intelligente l'enumerazione, cercando di lavorare sui limiti superiori ed inferiori, eliminando quindi alcune partizioni senza andare ad enumerarle. Si eliminano dunque i problemi *inammissibili* o quelli per i quali l'upperbound UB è inferiore all'ottimo corrente , che coincide al Lower Bound nel caso di problemi a massimizzare.
Viene poi aggiornato *l'ottimo corrente* se si trovano soluzioni ammissibili migliori di quella corrente, oppure partizionando ulteriormente i sotto-problemi che non si è in grado di risolvere.

Dunque le componenti fondamentali di questo metodo saranno le procedure di Bounding e di Brancching.
Nel **Bounding** si determina un valore limite, detto *Bound*, di fun-

zione obiettivo per un generico sotto problema.(upper bound per un problema a massimizzare, lower bound per uno a minimizzare.
Il **Branching** invece partiziona in sotto problemi e genera nuovi sottoproblemi più semplici.

Il bounding può esser fatto per *rilassamento continuo*, eliminando i vincoli di interezza ottenendo un problema continuo. Altrimenti si possono rilassare vincoli difficili, ottenendo un sottoproblema più facile.
Se risolvo il rilassamento di un problema a massimizzare ottengo un upper bound per quel problema. Sia I l'insieme di ammissibilita del problema intero e $Z_i$ il valore della sua soluzione ottima. Sia R l'insieme di ammissiblità del problema rilassato (quello di interezza) e sia $Z_r$ il valore della sua soluzione ottima. L'insieme I sarà quindi certamente contenuto nell'insieme R.

È possibile esser fortunai, per cui lavorando sull' insieme R ottengo il punto di ottimo appartenente anche all'insieme delle soluzioni intere I. perciò $Z_r$ coincide con $Z_i$. Tuttavia nella maggior parte dei casi ciò non accade. Otterrei un punto appartenente ad R ma non ad I. allora $Z_r$ sarà un limite superiore per il mio problema.

Per quanto ne concerne il Branching invece, se ho a che fare con un problema 01, ovvero binarii, come nell'esempio dello zaino, un modo banale per fare branching è quello di partizionare in maniera binaria, fissando a 0 o 1 una variabile.
Tuttavia in un caso generico solitamente non si ricorre allo schema ad albero visto in precedenza. Considerando il dominio di ammissibilità "triangolare" visto in precedenza, la soluzione ottima era in corrispondenza del vertice, che definival'upper bound. Un modo facile per partizionare è quello di prendere una delle componenti $x_1, x_2$ che non abbia valore intero. In questo caso possiamo scegliere $x_1$ e da questo momento in poi si risolverà da un lato il problema P1 e dall'altro P2.effettuando da un lato una approssimazione per difetto, dall'altra per eccesso.

In questo caso però P1 e P2 potrebbero non sembrare partizioni in quanto la loro unione non darebbe P. tuttavia in quella fascia eliminata non ci sono soluzioni intere, quindi non sto effettivamente perdendo soluzioni.

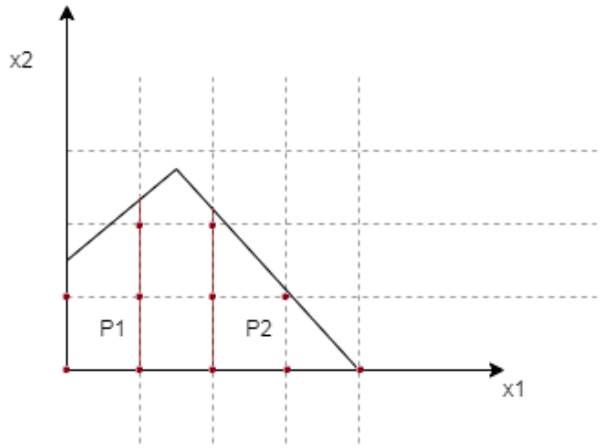



Figure 20: Partizionamento di un problema grafico

Dunque posto P il sotto problema e x la soluzione del rilassamento, sia $x_k = v \notin \mathscr{X}$, una componente della soluzione di valore frazionario. è possibile ottenere le due paritizioni come:

$$P_1 = P \cap \{x : x_k \leq |v|\}$$
$$P_2 = P \cap \{x : x_k \geq |v|\}$$

Poniamo dunque lo schema risolutivo di un generico problema PLI. In generale, per risolvere il problema PLI, si inizializza una *lista* con unico elemento il problema da risolvere. Viene inoltre posta indefinita la attuale migliore soluzione intera calcolata $\tilde{x}$ e posto a $-\infty$ il lower bound corrente $\tilde{z}$.

Ad ogni passo iterativo sarà dunque presente una lista problemi da risolvere, ovvero le foglie dell'albero di enumerazione.
Dunque iterazione per iterazione verranno aggiornate queste informazioni.Ad ogni passo viene estratto un sottoproblema dalla lista L e si risolve il rilassamento sul problema $S_i$. Se il problema è inammissibile si torna alla lista per valutare il successivo sottoproblema, altrimenti si effettua il calcolo dell'*Upper bound* $U_i$ . se questo upper bound è minore del lower bound attuale $\tilde{z}$, allora si torna alla lista per prelevare il successivo sottoproblema.
Se così non è però e se la soluzione attuale $y_i$ è intera,viene aggior-

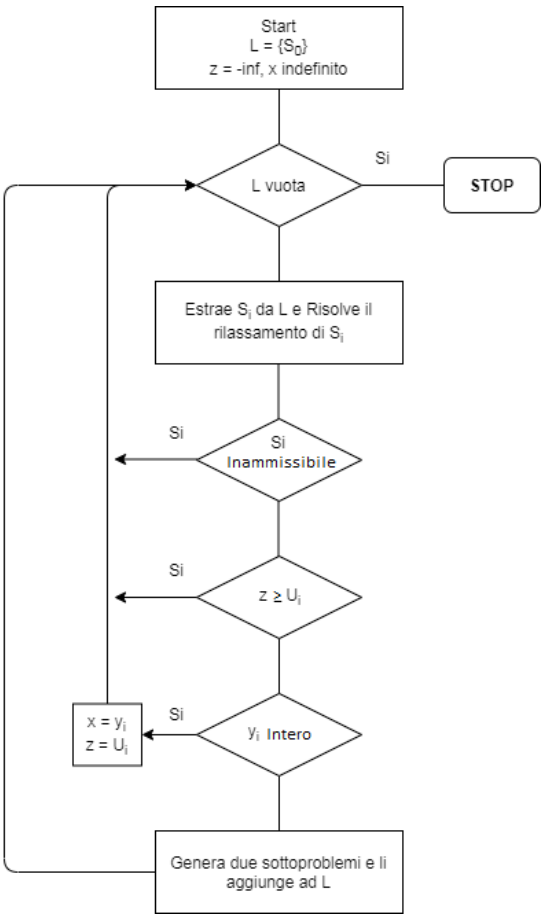



Figure 21: Schema grafico del metodo Branch and Bound

nata la soluzione corrente, e il nuovo upper bound, altrimenti vengono generati due nuovi sottoproblemi e aggiunti ad L.

$$\tilde{x} = y_i$$

$$\tilde{z} = U_i$$

Il B&B è un algoritmo *esatto* e cioè garantisce di trovare l'ottimo, se questo esiste. Naturalmente bound migliore portano a costruire alberi di dimensioni ridotte,tuttavia per calcolare buoni bound sono necessarie maggiori risorse di calcolo, il che ci porta ad un trade off tra qualità dei bound calcolati e dimensione dell'albero di enumerazione.

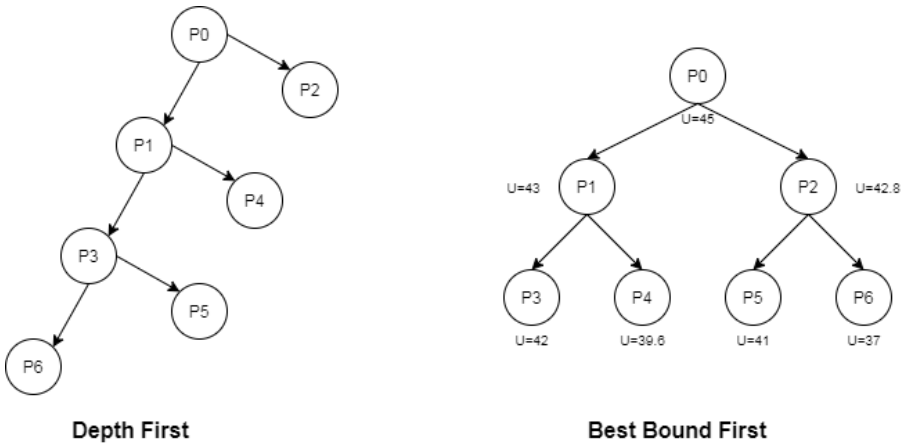



Figure 22: Strategie di ispezione grafica nel Branch and Bound

Sarebbe possibile utilizzare anche delle *euristiche* per trovare soluzioni ammissibili di lower bound.
Possiamo inoltre presentare due strategie diverse di esplorazione dell'albero; la prima è una ricerca del tipo textbfDepth first in cui si parte dal problema iniziale P0 e si generano i sottoproblemi figli P1 e P2. Si andrà a prelvare l'ultimo sottoproblema inserito, quindi P1, dal quale saranno poi generati altri sottoproblemi P3 e P4. Con questo ragionamento l'albero di enumerazione procede verso il basso fino ad arrivare ad un problema $Pn$ non più ispezionabile in quanto diventato magari inammissibile o comunque non ulteriormente scomponibile.
Una volta costruito l'albero, esso verrà percorso in senso inverso, ispezionando a ritroso e risalendo l'albero.

L'altra strategia è di tipo **Best bound**, nella quale ogni volta si va a prelevare il nodo più promettente, ovvero con un bound migliore. Si parte da P0 e si generano P1 e P2 e per ciascuno dei due sottoproblemi viene calcolato un upper bound. Se il problema è a massimizzare è più promettende quello con l'upper bound maggiore. In questo caso l'albero sarà sviluppato più in larghezza che in profondità.

Nonostante non ci sia una chiara supremazia di una strategia rispetto ad un'altra, i solutori fanno uso di una strategia Depth first, in maniera tale da arrivare ad un certo punto ad una soluzione ammissibile, per poi passare ad un Best bound, così da raffinare la soluzione.

Proviamo adesso a svolgere l'esempio portato avanti fino ad ora, applicando però il metodo *Branch and Bound*.

$$\max -x_1 + 2x_2$$

$$\begin{cases} -4x_1 + 6x_2 \leq 9 \\ x_1 + x_2 \leq 4 \\ x_1, x_2 \geq 0 \\ x_1, x_2 \in \mathscr{Z} \end{cases}$$

Come già detto, in corrispondenza del vertice del dominio di ammissibilità, abbiamo la soluzione $y_0$ del problema *rilassato*. possiamo quindi sostituire il valore di tale soluzione nella funzione obiettivo, ottenendo il nostro primo *Upper Bound* $U_0$.

$$y_0 = \left(\frac{3}{2}, \frac{5}{2}\right)^T$$

$$U_0 = \frac{7}{2}$$

La soluzione attualmente ottenuta però non è una soluzione intera, e quindi non ammissibile. decidiamo di partizionare il nostro dominio in 2 parti, una fissando la prima componente a 1, ed un'altra fissando la prima componente della soluzione a 2. Siamo così in grado di ottenere due nuovi sottoproblemi S1 ed S2, con le rispettive soluzioni e gli upper bound:

$$y_1 = \left(1, \frac{13}{6}\right)^T \qquad U_1 = \frac{10}{3}$$

$$y_2 = \left(2, 2\right)^T \qquad U_2 = opt_2 = 2$$

Notiamo che poichè la seconda soluzione $y_2$ è una soluzione intera, e quindi appartenente a , questa verrà presa come soluzione ottima attuale, fissando un Lower Bound. Possiamo ora continuare a dividere il sottoproblema S1 in due sottoproblemi S3 ed S4. In questo caso dovremo approssimare la seconda componente di $y_1$, una volta a 2, e l'altra a 3. ottenendo:

$$y_3 = \left(\frac{3}{5}, 2\right)^T \qquad U_3 = \frac{13}{4}$$

Notiamo che nel caso di S4 il vincolo $x_2 \geq 3$ appena introdotto, ci porta al di fuori del dominio di ammissibilità, rendendo il problema inammissibile. non avendo ancora raggiunto una nuova soluzione intera, possiamo suddividere ancora S3 in S5 ed S6, ottenendo le seguenti soluzioni:

$$y_5 = \left(0, \frac{3}{2}\right)^T \qquad U_5 = 3$$

$$y_2 = \left(1, 2\right)^T \qquad U_2 = opt_6 = 3$$

la nuova soluzione intera ottenuta $y_6$ fornisce un upper bound pari a 3, migliore quindi della precedente soluzione ottima, a cui corrispondeva un UB pari a 2. Poichè l'attuale $U_5$ è uguale al corrente Opt, l'algoritmo ci porterebbe al di sotto di questo nuovo LB. Dunque l'algoritmo è arrivato a corvengenza determinando la soluzione ottima.

Proviamo ora ad applicare l'algoritmo di Branch and Bound al problema dello zaino. Poichè è necessario ovviamente risolvere il rilassamento continuo del problem, un valore frazionario della variabile $x_i$ significherebbe mettere nello zaino uno porzione dell'oggetto, il che ovviamente ha poco senso. Tuttavia ai fini del rilassamento continuo possiamo assumere che gli oggetti possano esser partizionati, e prioritizzati tramite un *valore specifico* dato dal rapporto valore-prezzo $\frac{v_i}{p_i}$.

Volendo stabilire un ordine di priorità in base a tale rapporto, ordiniamo gli oggetti secondo valore specifico decrescente. Si preleva quindi il primo elemento dalla lista ordinata e se l'oggetto entra nello zaino esso viene inserito e si torna a prelevare un nuovo oggetto, altrimenti se l'oggetto prelevato non entra per intero nello zaino viene calcolata la massima porzione dell'oggetto che può essere inserita nello zaino.

Questo algoritmo è quello che si può utilizzare per risolvere il rilassamento continuo del problema per ogni nodo dell'albero di enumerazione del BB. Poniamo un problema di zaino continuo a 5 oggetti:

$$\max 32x_A + 36x_B + 15x_C + 20x_D + 5x_E$$

$$\begin{cases} 16x_A + 9x_B + 5x_C + 8x_D + 5x_E \leq 32 \\ 0 \leq x_A, x_B, x_C, x_D, x_E \leq 1 \end{cases}$$

Schematizziamo ora in tabella i l'ordinamento decrescente per valore specifico:

| Oggetto | V/P | V | P |
|---|---|---|---|
| $B$ | 4 | 36 | 9 |
| $C$ | 3 | 15 | 5 |
| $D$ | 2.5 | 20 | 8 |
| $A$ | 2 | 32 | 16 |
| $E$ | 1 | 5 | 5 |

I tre oggetti B,C e D entrano interamente nello zaino, per un peso totale di 22. L'oggetto A, che è il successivo in lista, entra solo per i suoi $\frac{10}{16}$ nello zaino, portandoci a definire una soluzione ottima del problema continuo:

$$x_B = x_C = x_D = 1 \quad x_A = \frac{10}{16} \quad x_E = 0$$

Possiamo così definire un nuovo Upper bound tramite la soluzione ottima del problema, ottenendo $z_1 \leq 91$. A questo punto una soluzione ammissibile del problema la si può ottenere eliminando dalla soluzione ottima del problema rilassato la componente frazionaria, ponendo quindi $x_A = 0$ . Al contempo possiamo decidere di realizzare un sottoproblema nel quale $x_A = 1$, ottenendo nel primo caso una soluzione ottima composta solo da valori interi delle variabili, con UB = LB = 76.

Nel sottoproblema in cui $x_A = 1$, $x_D$ sarà pari a $\frac{1}{4}$, dandoci un UP pari a 88 e un LV pari a 83.
Non avendo una soluzione ammissibile per il problema intero, continuiamo a partizionare in sottoproblemi, ponendo una volta $x_D = 1$ e un'altra volta $x_D = 0$
Nel primo caso stiamo ipotizzando di mettere sia l'oggetto A che l'oggetto D all'interno dello zaino,lasciando una capacità residua di 8. L'oggetto che riesce ad entrare in percentuale maggiore è B, che riesce ad entrare per i suoi $\frac{8}{9}$. Otterremo così un UP di 84 e un LB di 52.
Nel caso in cui D non sia inserito nello zaino, sarebbe possibile riuscire

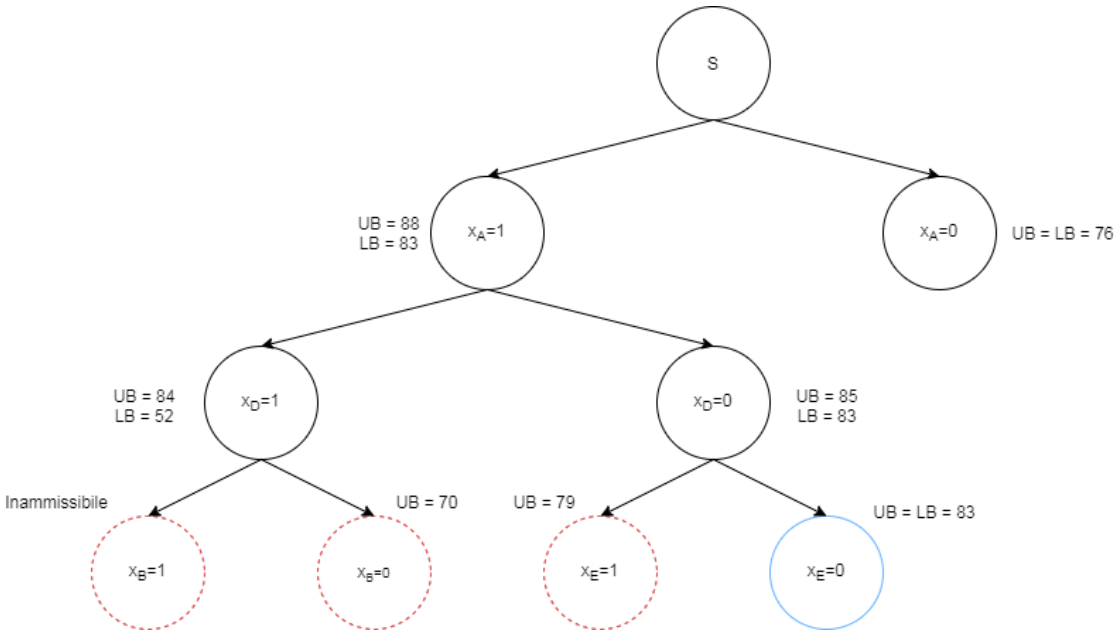



Figure 23: Metodo Branch and Bound applicato al problema dello zaino

a mettere nello zaino A,B,C e i $\frac{3}{5}$ di E per un UB = 85 e un LB = 83.

Provando a partizionare ulteriormente a partire dal sottoproblema con D nello zaino, otterrei il caso in cui $x_B$ venga interamente messo nello zaino, ma ciò eccederebbe la dimensione dello zaino stesso, definendo tale soluzione inammissibile. Al contempo potrei valutare di non mettere B nello zaino, ma otterrei un UB pari a 70, inferiore all'attuale LB.

L'unica alternativa è perciò quella da continuare a partire dal caso in cui l'oggetto D non sia inserito nello zaino. L'unica valutazione che resta da fare è quella di mettere o meno $x_E$ nello zaino. Nel caso in cui $x_E = 0$, otterremo un UB=LB=83, che sarà la soluzione ottima al problema intero.

$$x_A = x_C = x_B = 1 \quad x_E = 0 \quad x_D = 0$$

### 3.1.1 Carico Fisso

Può capitare talvolta di imbattersi in problemi con un *costo iniziale fisso*, legato per esempio all'inizio di una attivit. tale Costo prosegue poi con un andamento *proporzionale* al livello dell'attività. Per ogni variabile $x_i$ che rappresenta un bene a carico fisso, si introduce una nuova variabile binaria $y_j$ ed un limite superiore M al livello massimo dell'attività. Dunque possiamo definire il nuovo vincolo $x_i \leq My_i$ che indica l'attivazione della variabile $x_i$ solo qualora sia attiva anche la variabile $y_i$.

è quindi necessario aggiungere alla funzione obiettivo il termine $f_i x_i$ con $f_i$ **costo fisso** della variabile $x_i$. Per poter chiarire meglio questo problema, facciamone un esempio:

Un azienda produce tonno all'olio, tonno al vapore e tonno agli aromi, e la loro produzione richiede un macchinario, che deve essere affittato ad un costo fisso di 200 euro a settimana per tonno all'olio, 150 euro a settimana per il tonno al vapore, e 100 euro a settimana nel caso di aromi. Schematizziamo nelle seguenti tabelle le richieste di quantità di materia prima e le ore di lavoro, le spese e i guadagni:

| Prodotto | tempi di lavorazione | quantità di materia prima |
|---|---|---|
| Olio | 3 | 4 |
| Vapore | 2 | 3 |
| Aromi | 6 | 4 |

| Prodotto | Ricavo (euro/scatola) | Spesa(euro/scatola) |
|---|---|---|
| Olio | 12 | 6 |
| Vapore | 8 | 4 |
| Aromi | 15 | 8 |

Messe a disposizione 150 ore di lavoro a settimana e 160kg di tonno, si vuole definire un *piano di produzione settimanale*. Possiamo allora stabilire come variabili decisionali $x_i$ le quantità di prodotto da realizzare, e con $y_i$ la variabile rappresentativa del carico fisso, qualora $x_i > 0$. La funzione obiettivo da massimizzare sarà allora:

$$\max(12-6)x_1 + (8-4)x_2 + (15-8)x_3 - 200y_1 - 150y_2 - 100y_3$$

Introduciamo ora i vincoli del problema relativi alla Disponibilità di ore settimanali, quelli della materia prima e naturalmente i vincoli introdotti dal carico fisso, nonchè quelli di interezza:

$$\begin{cases} 3x_1 + 2x_2 + 3x_3 \leq 150 \\ 4x_1 + 3x_2 + 4x_3 \leq 160 \\ x_i \leq M_i y_i \quad \forall i = 1 \cdots 3 \\ x_i \in \mathscr{Z}^+ \quad \forall i = 1 \cdots 3 \\ y_i \in \{0, 1\} \quad \forall i = 1 \cdots 3 \end{cases}$$

In alternativa è possibile modellare due vincoli $g_1(x) \leq 0$ e $g_2(x) \leq 0$ che non possono essere contemporaneamente soddisfatti. Si introduce quindi una nuova variabile binaria y e una quantità M arbitrariamente grande, in maniera tale che l'alternativa tra i due vincoli possa essere modellata nella seguente forma:

$$g_1(x) \leq My$$
$$g_2(x) \leq M(1 - y)$$

A tal proposito introduciamo un esempio: una azienda produce tre tipi di auto: utilitarie, berline e station wagon. Supponiamo che risorse, richeiste e guadagni siano sintetizzate nella seguente tabella:

| Auto | Acciaio (tonnellate) | Lavoro(ore) | Profitto(euro) |
|---|---|---|---|
| utilitarie | 1.5 | 30 | 2000 |
| berline | 3 | 25 | 3000 |
| station wagon | 5 | 40 | 4000 |

Ipotizziamo inoltre di avere a disposizione 6000 tonnellate di acciaio, 60000 ore di lavoro totali e di essere a conoscenza che se si produce un tipo di veicolo, allora se ne devono produrre almeno 1000 unità. L'obiettivo è formulare un piano di produzione nel quale massimizzare il profitto in euro:

$$\max 2000x_1 + 3000x_2 + 4000x_3$$

Per quanto ne concerne i vincoli, dovremmo imporre come prima cosa dei vincoli relativi alla disponibilità di materia prima e di ore di lavoro. Dunque vale che:

$$1.5x_1 + 3x_2 + 5x_3 \leq 6000$$

$$30x_1 + 25x_2 + 40x_3 \leq 60000$$

Adesso possiamo impostare i due vincoli mutuamente esclusivi che rappresentino l'alternativa tra non produrre affatto un determinato tipo di vettura, oppure produrne almeno 1000. tale coppia di vinoli andrà impostata per ogni tipo di automobile prevista dal problema.

$$\begin{cases} x_1 \leq 4000y_1 \\ 1000 - x_1 \leq 4000(1 - y_1) \\ x_2 \leq 2000y_2 \\ 1000 - x_2 \leq 2000(1 - y_2) \\ x_3 \leq 1200y_3 \\ 1000 - x_3 \leq 1200(1 - y_3) \\ x_1, x_2, x_3 \in \mathscr{Z}^+ \quad y_1, y_2, y_3 \in \{0, 1\} \end{cases}$$

I valori scelti per M, ovvero 4000, 2000 e 1200 sono i limiti superiori al valore massimo di produzione di ogni tipo di auto. Per ottenere questi valori, ci basta considerare di produrre unicamente una tipologia di auto. Ipotizziamo di andare a produrre solo le auto associate a $x_3$, ciò significa che nel vincolo relativo alla quantità di acciaio, il massimo numero producibile, date le risorse, sarà di 1200 vetture.

### 3.1.2 Modelli ti tipo logico

Immaginiamo di avere un problema che comprenda una data implicazione logica del tipo *IF THEN ELSE*.
Stiamo supponendo quindi che data una coppia di variabili $x_i$ e $y_i$, se $x_i > 0$ allora deve valere che $y_i = 1$, altrimenti $y_i = 0$.
Tale modello non è un modello lineare, e dobbiamo quindi capire come impostarlo in termini di vincoli al fine di poterlo descrivere come un problema lineare.

$$\begin{cases} x_i \leq My_i \\ x_i \geq 0 \\ y_i \in \{0, 1\} \end{cases}$$

Supponiamo di avere 2 variabili $x_i$ e $x_j$ maggiori di zero, rappresentanti di due attività *incompatibili* tra loro. è allora possibile aggiungere due variabili binarie $y_i$ ed $y_j$, indicanti se una attività sia svolta o meno, uguali a zero se le corrispondenti variabili x sono anch'esse uguali a zero.
L'incompatibilit tra le due condizioni può essere espresso mediante il vincolo $y_i + y_j \leq 1$. in base a tale vincolo, è possibile affermar che può esser svolta una sola delle due alternative, o al più nessuna delle due.

$$\begin{cases} x_i \leq My_i \qquad x_j \leq My_j \\ y_i + y_j \leq 1 x_i \geq 0 \qquad x_j \geq 0 \\ y_i \in \{0,1\} \qquad y_j \in \{0,1\} \end{cases}$$

Anche in questo caso possiamo introdurre un esempio: Una agenzia finanziaria deve investire 1 milione di euro di un suo cliente in fondi di investimento. Il mercato offre cinque tipi di fondi, che sono riassunti nella tabella:

| Nome | Tipo | Moody's rating | Durata in anni | Rendita alla maturazione |
|---|---|---|---|---|
| A | privato | Aa | 9 | 4.5% |
| B | pubblico | A | 15 | 5.4% |
| C | stato | Aaa | 4 | 5.1% |
| D | stato | Baa | 3 | 4.4% |
| E | privato | Ba | 2 | 4.1% |

si sa che i fondi pubblici e dello stato sono tassati del 50% alla fine del periodo. Il cliente chiede di riservare almeno il 40% del capitale ai fondi pubblici o dello stato, e ha richiesto che il tempo medio della durata dell'investimento non debba superare i 5 anni. Inoltre, il cliente esige che al massimo uno tra i fondi di investimento C e D sia attivato. Infine, trasformando il Moddy0s rating in una scala numerica, il valore medio dell'investimento non deve superare 1.4. (Aaa = 1, Aa = 2, A = 3, Baa = 4, Ba = 5).
L'obiettivo è massimizzare la rendita dell'investimento, il che ci porta alla seguente funzione obiettivo:

$$\max 4.5x_A + 2.7x_B + 2.55x_C + 2.2x_D + 4.1x_E$$

dove le variabili decisionali da A ad E indicano la quantità di euro investiti nei corrispettivi fondi. è necessario introdurre delle variabili

binarie $y_C$ e $y_D$ tali che la prima sia 1 se l'investimento su C è attivo, altrimenti 0. Discorso analogo può esser fatto per D.
Per quanto ne concerne i vincoli, è necessario in primo luogo porne uno per la dimensione dell'investimento, in maniera tale da non spendere più del milione a disposizione. Inoltre va imposto il vincolo sul Moody's rate, nonchè il tempo medio della durata dell'investimento richiesto. Infine possiamo impostare il vincolo sul capitale destinato ai fondi pubblici e dello stato, ottenendo il seguente sistema:

$$\begin{cases} x_A + x_B + x_C + x_D + x_E \leq 1000000 \\ 2x_A + 3x_B + x_C + 4x_D + 5x_E \leq 1.4(x_A + x_B + x_C + x_D + x_E) \\ 9x_A + 15x_B + 4x_C + 3x_D + 2x_E \leq 5(x_A + x_B + x_C + x_D + x_E) \\ x_B + x_C + x_D \geq 400000 \\ x_C \leq My_C \qquad x_D \leq My_D \geq 0 \\ y_C + y_D \leq 1 \\ y_C, y_D \in \{0, 1\} \end{cases}$$

# 4 Problemi di ottimizzazione definiti su grafo

Nell'ambito dei problemi di flussi di reti, il più semplice tra questi è il problema del *cammino minimo*, o shortest path problem.
Dato un grafo orientato G = (V,A), con V insieme dei vertici e A l'insieme degli archi, si vuole trovare un percorso di costo minimo da un nodo sorgente $s \in V$ ad un noto destinazione $t \in V$, assumendo che ciascun arco $(i, j) \in A$ sia caratterizzato da un costo $c_{ij}$.

Nonostante l'interpretazione grafica, Questo problema è modellabile come un problema di programmazione lineare, ed è di grande interesse in quanto largamente applicato sia nei protocolli di rete che in applicazione di uso comune come i navigatori GPS.

La prima formulazione che andiamo ad analizzare riguarda il cammino minimo da **una origine ad una destinazione**, nel quale il problema può esser formulato come un problema di flusso a costo minimo in cui i termini noti b(x) assumono valori 1 in caso della sorgente, -1 in caso della destinazione e 0 in tutti gli altri casi.

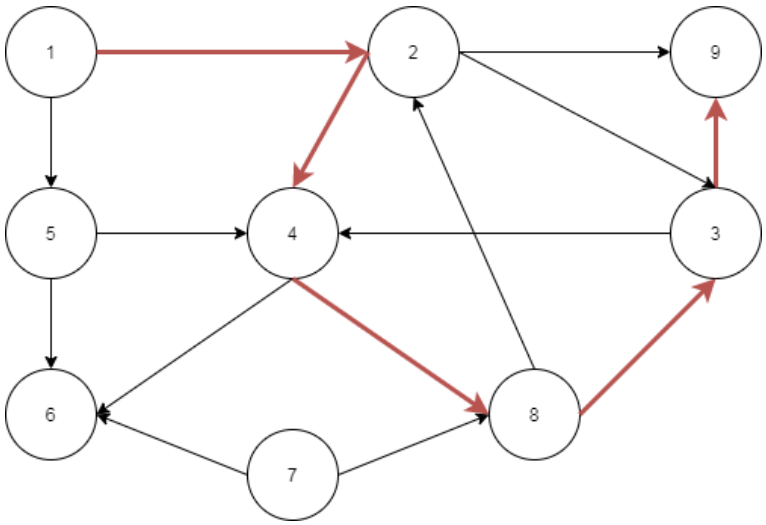



Figure 24: Problema del cammino minimo da una origine a una destinazione

Posto $c_{ij}$ il costo dell'arco (i,j) e P il percorso che viene costruito, possiamo stabilire che il determinato nodo $x_{ij}$

$$x_{ij} = \begin{cases} 1 & \text{se} \quad (i,j) \in P \\ 0 & \text{altrimenti} \end{cases}$$

La funzione obiettivo in questi problemi dovrà essere la minimizzazione della somma dei nodi presenti nel percorso P , pesati per gli archi:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

Per quanto ne concerne i vincoli invece, se il nodo preso in considerazione è il nodo origine, il vincolo relativo a questo nodo deve imporre che almeno uno degli archi *uscenti* dall'origine deve appartenere alla soluzione. Se invece il nodo valutato è quello destinazione, dobbiamo imporre che almeno uno degli archi *entranti* appartenga alla soluzione. Infine, nel caso di un qualsiasi altro nodo appartenente all'insieme V, vale praticamente la legge di Kirchoff ai nodi.

Possiamo quindi impostare due sommatorie, con i fissato e j variabile. Nel caso della prima sommatoria, j è appartenente alla *stella uscente dal nodo* i ($\delta(i)^+$), ovvero l'insieme dei nodi j destinazione di

archi con origine in i. Per esempio in figura, la stella uscente da 1 è fatta dai nodi 2 e 5.
La seconda sommatoria invece farà variare j su $\delta(i)^-$,ovvero l'insieme di tutti i nodi che sono origine di archi con destinazione nel nodo i, perciò relativa agli archi entranti.
Per tutti i nodi diversi da sorgente e destinazione,il vincolo pone le sommatorie uguali a zero, e quindi il flusso entrante in un generico nodo deve essere uguale al flusso dei nodi uscenti.

$$\sum_{j \in \delta(i)^+} x_{ij} - \sum_{j \in \delta(i)^-} x_{ji} = \begin{cases} 1 & \text{se} \quad i = s \\ 0 & \text{se} \quad i \in V - \{s, t\} \\ -1 & \text{se} \quad i = t \end{cases}$$

è possibile a questo punto utilizzare le formule mostrate per ogni nodo del grafo, ottenendo una matrice **unimodulare**. Ciò significa che presa una qualsiasi sottomatrice quadrata essa avrà determinante pari a 1, -1 o 0. Quando accade questo, la soluzione ottima che si ottiene risolvendo il problema continuo con l'algoritmo del simplesso ad esempio, è già intera e allora è anche la soluzione ottima del problema originario intero.

Analizziamo ora il caso di un problema di cammino minimo da **una origine a tutti gli altri nodi**. In questo caso stiamo considerando un nodo sorgente s e tutti gli altri nodi del grafo G come destinazioni.

Il problema può perciò esser visto come un problema di flusso a costo minimo in cui si vuole mandare una unità di flusso dal nodo origine s a tutti gli altri nodi della rete.
Possiamo immaginare di avere n-1 pacchi da consegnare, uno per ogni nodo destinazione, e voler consegnarli in maniera tale che ognuno di essi faccia il percorso più breve possibile.
In questo caso non possiamo più fare uso di variabili binarie, in quanto la generica variabile $x_{ij}$ non darà più semplicemente l'informazione se un arco faccia parte del percorso o meno, ma dovrà portare informazioni relative al numero di pacchetti che verranno fatte passare per quell' arco. Dunque ogni variabile associata ad un arco sarà intera e non più binaria.

La funzione obiettivo rimane la stessa, di prima, e quindi:

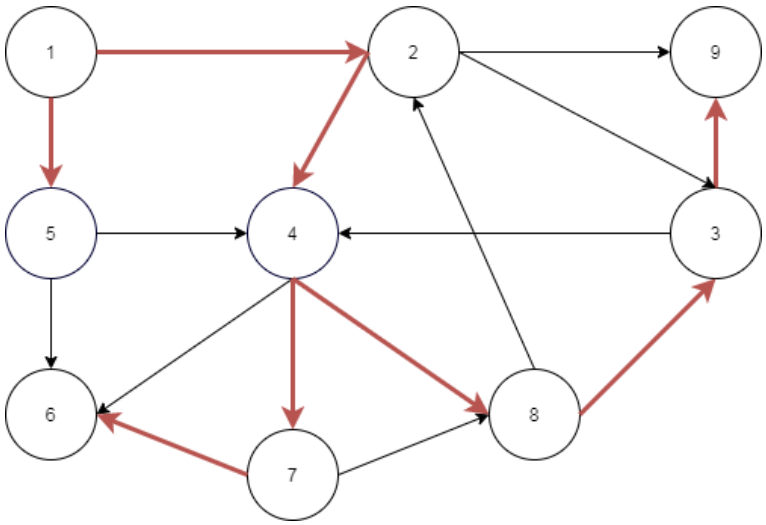



Figure 25: Problema del cammino minimo da una origine ad n estinazioni

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

Per quanto riguarda i vincoli invece, l'infrastruttura che realizzeremo sarà molto simile a quella precedentemente esposta, prevedendo quindi un vincolo per ogni nodo del grafo.
Tuttavia in questo caso stiamo dicendo che dal nodo sorgente devono partire n-1 pacchetti, mentre in ogni altro nodo deve entrare almeno un pacchetto, e tutti quelli in eccesso devono essere scartati. I vincoli potranno quindi esser espressi come:

$$\sum_{j\in\delta(i)^+} x_{ij} - \sum_{j\in\delta(i)^-} x_{ji} = \begin{cases} n-1 & \text{se} \quad i = s \\ -1 & \text{se} \quad i \in V - \{s\} \end{cases}$$

Anche in questo caso la matrice ottenuta sarà unimodulare, quindi la soluzione ottima del problema continuo coinciderà con quella del problema intero.

Durante tutte queste ipotesi abbiamo però fatto implicitamente una assunzione: tale modello è corretto solo se il grafo è **aciclico**. Se il grafo fosse ciclico, al fine di avere un modello corretto e risolvibile, sarebbe necessario che sul grafo non si presentino cicli di *costo totale*

*negativo.*
Se ciò avvenisse allora un qualsiasi algoritmo andrebbe in loop, in quanto continuerebbe a percorrere lo stesso tracciato, *migliorando* ad ogni iterazione il suo costo. Se infatti cambiassimo il costo di un arco da un valore apositivo ad uno negativo, quando l'algoritmo andrà a valutare un eventuale ciclo contenete quell'arco, continuerà a percorrerlo all'infinito.

Questo è un problema **NP completo**, per il quale non esiste alcun algoritmo di soluzione con complessità polinomiale.

## 4.1 Problemi di cammino minimo

Gli approcci algoritmici per la soluzione del problema del cammino minimo vengono quindi suddivisi in due possibili gruppi, entrambi basati sulle iterazioni. Entrambi assegnano ad ogni iterazione delle etichette di tentativo ai nodi, dove ogni etichetta rappresenta una stima, o upper bound del valore del cammino minimo. quando l'algoritmo termina, l'etichetta associata a ciascun nodo indica il valore del cammino minimo dal nodo sorgente al nodo cui l'etichetta è associata. Il modo in cui le due tipologie variano, sta nel modo in cui vengono aggiornate le etichette.

- **Label Setting** : Algoritmi di questo tipo ad ogni iterazione rendono *permanente* una etichetta, e sono applicabili quando il grafo è aciclico, oppure non sono presenti archi di costo negativo
- **Label Correcting**: Algoritmi di questa tipologia considerano tutte le etichette come *temporanee* e le rendono permanenti solo arrivati all'ultima iterazione. Essi sono quindi applicabili anche in presenza di archi negativi, puchè il costo totale di eventuali cicli non sia negativo.

Prendiamo prima di tutto in considerazione il caso di grafo aciclico. Possiamo quindi introdurre il concetto di *numerazione topologica* dei nodi di un grafo : ciò significa assegnare dei numeri ai nodi del grafo in modo tale che $i < j$ per ogni arco $(i, j) \in A$. Se il grafo è aciclico siamo sempre in grado di trovare una numerazione topologica, altrimenti non è possibile farlo.

Possiamo quindi presentare un algoritmo per realizzare tale numerazione, partendo dall'assunto che se il grafo totale è aciclico, allora anche un suo sottografo di sarà aciclico. Inoltre in un grafo aciclico esiste sempre almeno un nodo senza archi entranti. Poste queste ipotesi, l'algoritmo per la numerazione topologica di un grafo G , definito dalla coppia di Nodi V e archi A, ragiona iterativamente come di seguito:

1. Si pone $\hat{G} = G$ e i = 1, con $\hat{G}$ il grafo temporaneo
2. se tutti i nodi di $\hat{G}$ hanno archi entranti l'algoritmo si arresta, poichè il grafo è *ciclico*
3. Si assegna il valore i ad un nodo di $\hat{G}$ senza archi entranti
4. si elimina il nodo i e i suoi archi uscenti dal grafo $\hat{G}$, e si pone i = i+1
5. se $\hat{G}$ non è vuoto si ritorna al passo due, altrimenti l'algoritmo termina correttamente

La complessità computazione di questo algoritmo sarà pari al numero di archi del grafo m.

Una volta che il grafo è stato *ordinato*, l'algoritmo per il calcolo dei cammini minimi non fa altro che etichettare i nodi seguento il loro ordine topologico. Dunque possiamo definire un algoritmo per il calcolo dei cammini minimi sul grafo G(V,A) ordinato topologicamente come:

1. Si pone d(1) = 0, p(a) = -1 e j = 2, dove d saranno le etichette e p definito *predecessore*
2. si calcola f(i) = $\min_{i<j}(f(i)+w_{ij})$ e $p(j) = arg\min_{i<j}(f(i)+w_{ij})$
3. si pone j = j+1
4. se $j > n$ allora l'algoritmo termina, altrimenti si torna al passo 2

Anche in questo caso l'algoritmo presenterà una complessità del tipo O(m) in quanto dovrà essere analizzato ogni arco.
Facciamo quindi un esempio relativo a tale algoritmo, facendo riferimento al grafo in figura.
Sia 1 il nodo origine, e gli venga attribuita una etichetta d(1) = 0, con predecessore p(1) = -1, in modo tale da indicare il nodo come sorgente.

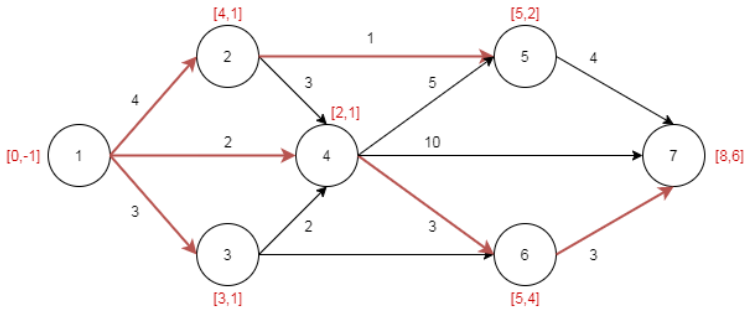



Figure 26: Risoluzione di un grafo topologicamente ordinato con un algoritmo per il calcolo dei cammini minimi

Essendo stato ordinato topologicamente, il cammino da 1 a 2 sarà quello minimo per arrivare a 2, che verrà quindi etichettato dalla coppia costo/predecessore [4,1]. Allo stesso modo è possibile stabilire una etichetta [3,1] per il nodo 3.

Il nodo 4 invece sarà raggiungibile da diverse fonti, ovvero i nodi 1,2 e 3. Possiamo quindi calcolare il cammino minimo da 1 a 4, secondo la formula del passo 2:

$$d(4) = min(d(1) + 2; d(2) + 3; d(3) + 2) = 2 \implies p(4) = 1$$

Così facendo, sono in grado di stabilire i valori di tutte le altre etichette. Grazie a queste etichette sarà possibile stabilire non solo il costo per giungere ad ogni nodo a partire da una stessa sorgente, ma sarà anche possibile ricostruire il percorso a ritroso.

### 4.1.1 Algoritmo di Dijkstra

Per poter risolvere problemi modellati tramite grafi ciclici,è possibile utilizzare un algoritmo Labeled Setting chiamato **Algoritmo di Dijkstra**, ipotizzando che sugli archi non ci siano costi negativi. L'algoritmo prevede che ad ogni iterazione sarà calcolato uno dei cammini minimi.
Ad ogni passo dell'algoritmo abbiamo un insieme di nodi S con le etichette già correttamente assegnate. Il nuovo nodo da mettere in S è il nodo che ha distanza minima da 1 col vincolo di passare solo

attraverso nodi di S.L'algoritmo termina in un numero di passi pari al numero di nodi. Possiamo quindi descrivere l'algoritmo secondo i seguenti passi:

1. si pongono

$$S = \{s\} \qquad d(s) = 0 \qquad p(s) = -1$$

$$U = V - \{S\} \qquad d(j) = +\infty \qquad \forall j \in U \qquad i = s$$

2. Per ogni arco (i,j) appartenente alla stella $\delta^+(i)$ con $j \in U$, se $d(j) < d(i) + w_{ij}$, allora si aggiorna l'etichetta temporanea di j ponendo $d(j) = d(i) + 2_{ij}$ e p(i) = i
3. Si pone $i = arg\min_{i \in U} d(i)$ e $S = S \cup \{i\} \qquad U = U - \{i\}$
4. se U è diverso dall'insieme vuoto si torna al passo 2, altrimenti si termina l'algoritmo

L'algoritmo appena presentato avrà quindi una complessità computazionale dell'ordine di $O(n^2)$. per reti sparse si può utilizzare una implementazione con code di priorità, riducendo la complessità ad $O(m \log n)$ con m numero di archi ed n numero di nodi del grafo.

Proviamo a presentare un esempio circa l'algoritmo di Dijkstra:

Nello step di inizializzazione poniamo S = {1}, d(1) = 0, p(1) = -1. Abbiamo così definito il nodo sorgente come primo nodo risolto. Poniamo successivamente U, l'insieme dei nodi non ancora risolti, pari a {2,3,4,5,6,7}, con ognuno di questi nodi avente costo d(j) = $+\infty \qquad \forall j \in U$.

Alla prima iterazione si esamina la stella uscente dal nodo 1, contenete quindi gli archi (1,2), (1,3) e (1,4). Essendo i costi d(j) di ognuno di questi nodi pari a $+\infty$, verranno fatte le seguenti assegnazioni:

$$d(2) = d(1) + 10 = 10 \qquad p(2) = 1$$

$$d(3) = d(1) + 7 = 7 \qquad p(3) = 1$$

$$d(4) = d(1) + 5 = 5 \qquad p(4) = 1$$

Poichè l'unico arco per raggiungere 4 è stato esaminato, possiamo aggiungere il nodo 4 all'insieme dei nodi risolti S, e rimuoverlo dall'insieme

U.

Nella seconda iterazione possiamo analizzare la stella uscente dal nodo 4, e dunque gli archi (4,3) e (4,6). Varranno dunque le seguenti valutazioni:

$$d(3) = 7 > d(4) + 1 = 6$$
$$d(6) = +\infty > d(5) + 9 = 14$$

Possiamo quindi inserire il nodo 3 in S e rimuoverlo dall'insieme U.

è possibile continuare l'algoritmo con una terza iterazione nella quale analizzare la stella uscente da 3, così da individuare un nuovo percorso minimo per il nodo 2, aggiungerlo ad S e rimuoverlo da U. L'algoritmo terminerà quando tutti i nodi saranno stati inseriti in S e rimossi da U.

Come già accennato però, è possibile che un arco presenti un costo negativo. In situazioni reali ciò è possibile quando lo spostarsi su di un arco può portare un guadagno in termini di costo, mentre spostarsi su di altri può portare uno svantaggio.
Gli algoritmi label setting tuttavia non si prestano alla risoluzione di questo tipo di problema, mentre quelli label correcting riescono ad aver successo, in quanto stabiliscono l'etichetta definitiva per ogni nodo solo al termine di tutte le iterazioni dell'algoritmo.

### 4.1.2 Algoritmo di Bellman-Ford

Gli algoritmi laberl correcting si basano su alcune *condizioni di ottimalità*: dato un grafo orientato G e un insieme di etichette d, che indichino la lunghezza di un percorso dal nodo sorgente s al nodo j, si può dimostrare che le etichette d(j) rappresentano le lunghezze dei cammini minimi dal nodo s se e solo se esse soddisfano la condizione :

$$d(j) \leq d(i) + c_{ij} \qquad forall(i,j) \in A$$

Un generico algoritmo*Label correcting*, lavora seguendo questi passi elaborativi:

1. si pone
$$S = \{s\} \qquad d(s) = 0 \qquad p(s) = -1$$
$$U = V - S \qquad d(j) = +\infty \qquad \forall j \in U$$

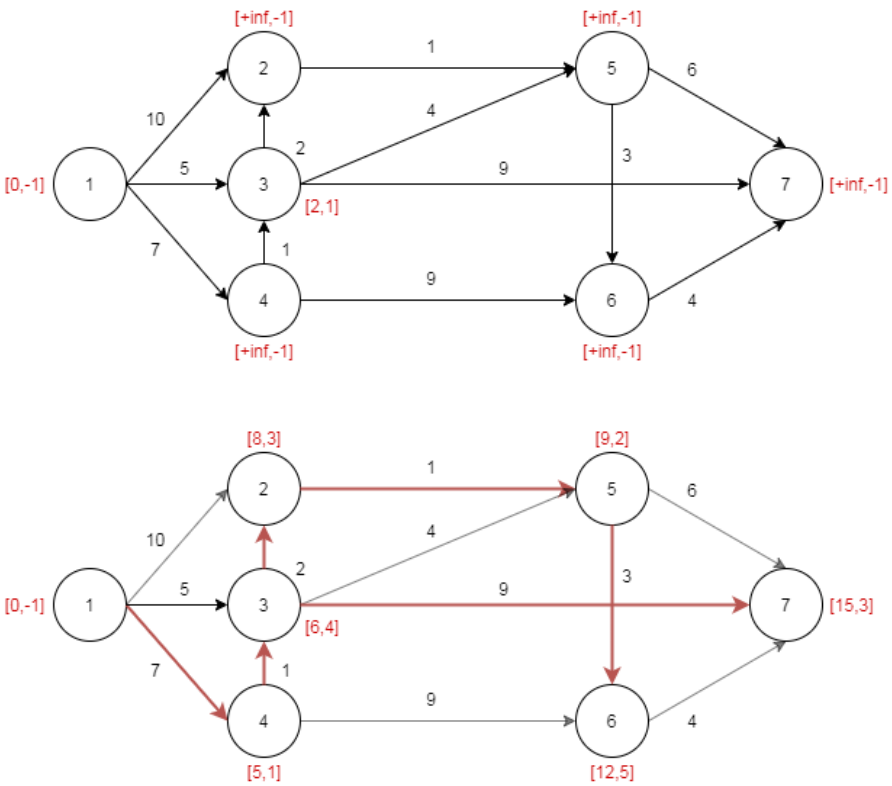



Figure 27: Risoluzione di un grafo con l'algoritmo di Dijkstra

2. se per tutti gli archi $(i, j) \in A$ vale la condizione di ottimalità, allora l'algoritmo termina ; è stata trovata la soluzione ottima
3. sia (i,j) un arco per il quale $d(j) > d(i) + c_{ij}$, allora si pone $d(j) = d(i) + c_{ij}$ e p(j) = i.
4. si ritorna allo step 2.

è in questo contesto che nasce 'algoritmo di **Bellman Ford** che al primo passo individua i cammini minimi tra il nodo sorgente e gli altri nodi con il vincolo che i cammini contengano al più 1 arco del grafo. Al secondo passo si trovano i cammini minimi tra il nodo sorgente e gli altri nodi, con il vincolo che i cammini contengano al più 2 archi. Si itera il procedimento fino al valore massimo di archi possibili di un cammino, ovvero n-1. L'algoritmo così termina con un numero di iterazioni pari a n-1, con $n = |V|$.

Posta quindi $d^k(j)$ l'etichetta del nodo j alla iterazione k, e $BS(j) = \{i \in V : \exists (i, j) \in A\}$ l'insieme dei nodi che possono precedere j nel grafo G, l'algoritmo di Bellman Ford può essere così descritto:

1. si pongono $d^0(s) = 0 \qquad p(s) = -1 \qquad d^0(j) = +\infty \quad forall j \in U; k = 1$
2. Per ogni nodo $j \in V$ si calcola u = $arg \min_{i \in BS(j)}(d^{k-1}(i) + c_{ij})$. Se $d^{k-1}(j) > d^{k-1}(u) + c_{uj}$, allora $d^k(j) = d^{k-1}(u) + c_{uj}$, con p(j) = u, altrimenti $d^k(j) = d^{k-1}(j)$
3. nel caso in cui $d^k(j) = d^{k-1}(j)$, l'algoritmo terminerà, in quanto sarà stata trovata la soluzione ottima.
4. se k = n l'algoritmo terminerà ugualmente in quanto sarà stato trovato un ciclo a costo totale negativo
5. Altrimenti k = k+1 e si torna al passo 2.

La complessità computazionale totale di questo algoritmo sarà del tipo O(nm).
Presentiamo anche in questo caso un esempio dimostrativo dell'algoritmo. Durante il passo di inizializzazione la distanza $d^0(1) = 0$ e p(1) = -1, mentre tutte le altre distanze saranno poste a $+\infty$.

Durante il primo passo iterativo saranno valutati i cammini minimi con al più un arco. E allora :

$$d^1(2) = 1 \qquad p(2) = 1$$

$$d^1(3) = 2 \qquad p(3) = 1$$

Durante la seconda iterazione saranno valutati i cammini minimi con al più 2 archi. Potremo così determinare i percorsi a costo minimo per i nodi 4 e 6. Verrà nuovamente calcolato il percorso a costo minimo verso 2, provando a passare per 3.Tuttavia, il costo per arrivare a 2 partendo da 1, ma passando per 3, sarà maggiore rispetto a quello precedentemente impostato, lasciando inalterata l'etichetta su 2. Potremo però definire le seguenti nuove etichette:

$$d^2(4) = d^1(2) - 1 = 0 \qquad p(4) = 2$$
$$d^2(6) = d^1(3) - 4 = -2 \qquad p(6) = 3$$

Nella terza iterazione sarà possibile arrivare al nodo 5, e fissarne la sua etichetta [-3,6]. L'algoritmo tuttavia non terminerà ancora, non avendo raggiunto le condizioni di ottimalità. si procederà dunque con una quarta iterazione, durante la quale si proverà a raggiungere 2 seguendo il percorso 1-3-6-5-2, e il nodo 4 seguendo il percorso 1-3-6-5-4.

$$d^4(2) = d^3(5) + 3 = 0 \qquad p(2) = 5$$
$$d^4(4) = d^3(5) + 2 = -1 \qquad p(4) = 5$$

L'algoritmo potrà così terminare.

## 4.2 Problemi di massimo flusso

Altro problema classico nell'ambito delle reti è relativo alla distribuzione di un dato prodotto da una sorgente, o origine, ad una destinazione, detta anche pozzo.
Dato un grafo orientato G, ogni arco è caratterizzato da una *capacità massima* $u_{ij}$ di flusso che può scorrere in quel dato arco.
Fulcro del problema è l'identificazione della massima quantità di flusso che è possibile inviare dalla sorgente s alla destinazione t attraverso il grafo G, definito dalla coppia V (insieme dei nodi) e A (insiem degli archi).

È possibile risolvere anche problemi di flusso nel caso di più sorgenti $s_i$ e più destinazioni $t_i$, riconducendoli all caso di unica sorgente e unica destinazione, aggiungendo al grafo di partenza due nodi fittizi aggiuntivi : una sorgente s* collegata con dei nuovi archi a tutti i

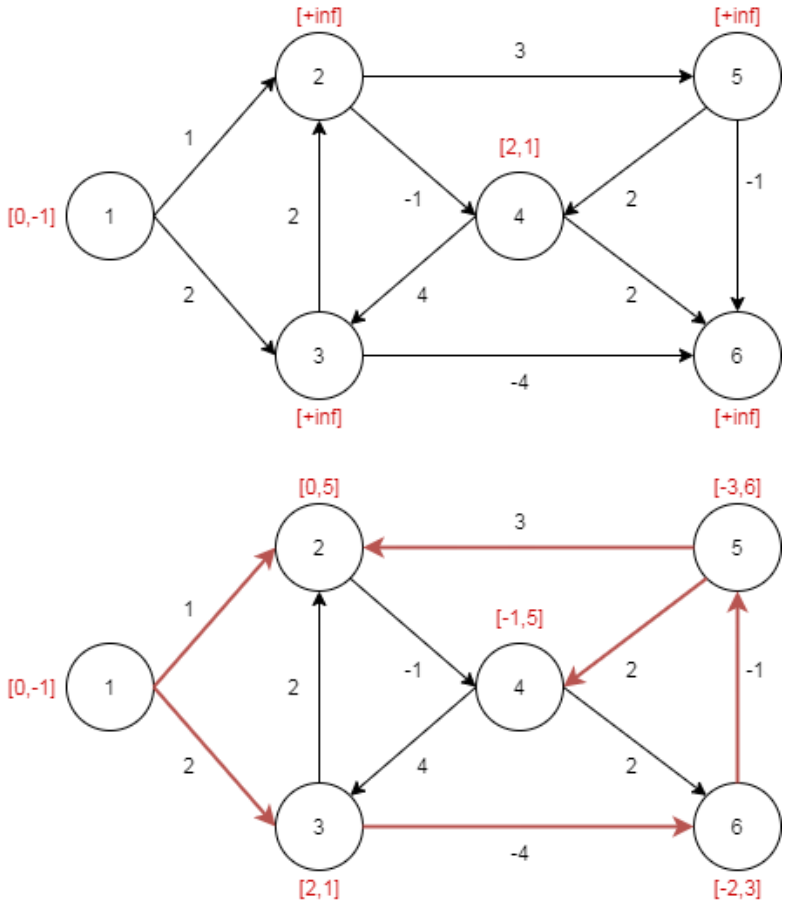



Figure 28: Risoluzione di un grafo con l'algoritmo di Bellman Ford

nodi $s_i$ con capacità illimitata, e un nodo destinazione s* collegato a ciascuno dei nodi $t_i$ con archi a capacità illimitata.

Risolvere il problema del massimo flusso da s* a t* equivale a risolvere il problema di partenza con più origini e più destinazioni.

Per modellare il problema è necessari ora stabilire le *variabili decisionali* corrispondenti alla quantità di flusso che attraversa l'arco (i,j) appartenente ad A, che chiameremo $x_{ij}$. tale flusso fovrà necessariamente essere maggiore di 0, e minore della capacità massima dell'arco $u_{ij}$.
Possiamo considerare in aggiunta una variabile f, misurante il flusso totale inviato dalla sorgente alla destinazione, in modo tale da tener traccia del flusso ttoale.

Banalmente allora la funzione obiettivo sarà

$$\max f$$

Per quanto ne concerne i vincoli invece, come accadeva nei problemi di cammino minimo, considereremo le stelle uscenti ed entranti nel nodo. Ogni nodo può quindi essere attraversato o meno dal flusso; Nel caso di nodi sorgente sappiamo che il flusso sarà interamente uscente dal nodo, e viceversa nel caso di nodo destinazione. Nel caso di nodi intermedi invece il flusso entrante dovrà coincidere con quello uscente, ragion per cui:

$$\sum_{j \in \delta(i)^+} x_{ij} - \sum_{j \in \delta(i)^-} x_{ji} = \begin{cases} f & \text{se} \quad i = s \\ 0 & \text{se} \quad i \in V - \{s, t\} \\ -f & \text{se} \quad i = t \end{cases}$$

è allora possibile impostare a partire da un grafo, impostare un problema lineare, risolvibile con l'algoritmo del simplesso. è tuttavia possibile individuare soluzioni e algoritmi più efficienti.

### 4.2.1 Problema di matching

Supponiamo di disporre di m ingegneri ed n lavori da svolgere con $n > m$.
A ciascun ingegnere deve quindi essere associata una lista di lavori

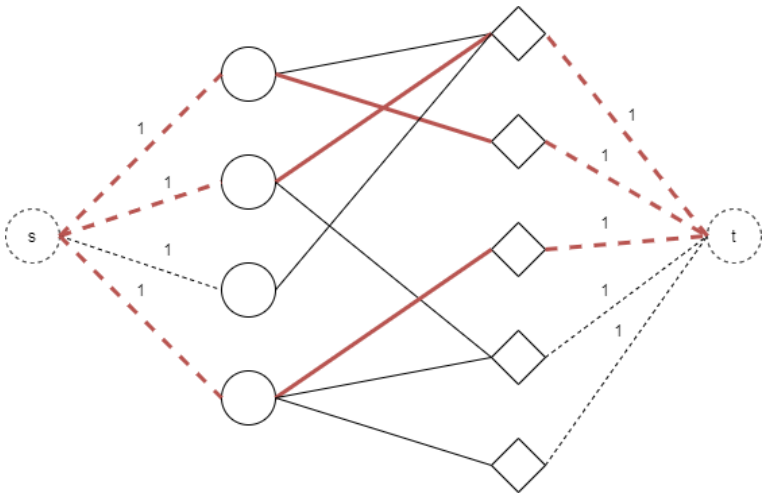



Figure 29: Grafo Bipartito in un problema di Matching

che quell'ingegnere è in grado di svolgere. Inoltre i lavori devono esser assegnati in maniera tale che ogni ingegnere svolga al massimo un lavoro ed ogni lavoro sia assegnato a non più di 1 ingegnere, avendo il numero di ingegneri occupati, ovvero di lavori eseguiti, massimizzato.

Il problema di assegnazione, o **Matching**, consiste nel determinare un accoppiamento con un numero massimo di archi sul grafo bipartito. A partire dal grafo bipartito, si possno aggiungedere dei nodi fittizi s e t che fungano da sorgente e destinazione. Ciascun ingegnere è quindi collegato al nodo sorgente, e ciascun lavoro è collegato alla destinazione.
Associamo inoltre ad ogni arco, sia fittizio che non, una capacità pari ad 1.
Una modellazione del problema come problema di flusso permette di giungere ad una soluzione, andando a selezionare solo gli archi intermedi.

### 4.2.2 Taglio di un grafo

Un altro problema relativo al flusso attraversante un grafo, è quello del **Taglio di un grafo**.
Un taglio è una partizione dell'insieme V dei nodi del grafo orientato G in due sottoinsiemi, il primo contenente il nodo s e il secondo t,

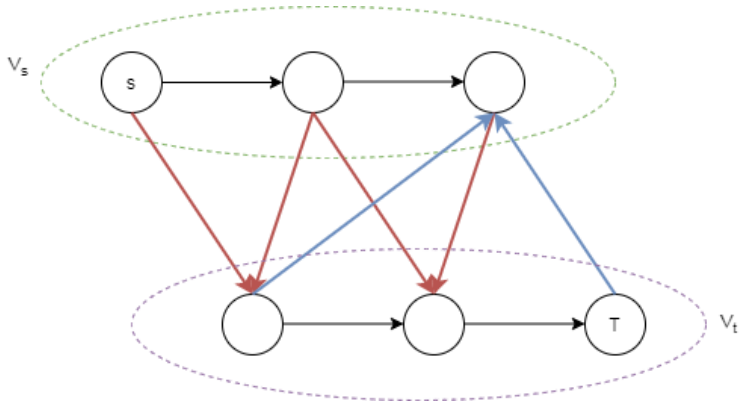



Figure 30: Taglio di un grafo orientato

definiti come:

$$V = V_s \cup V_t \qquad V_s \cap V_t = \emptyset \qquad s \in V_s \quad t \in V_t$$

Ciò significa che date le due partizioni, esisterà un insieme di archi direzionati che collegherà i due insiemi di taglio, così da ricostruire il grafo originale G. Possiamo distinguere tale insieme di archi in base alla *direzionalità*. Gli archi diretti da $V_s$ a $V_t$ prenderanno nome di **Archi diretti**, mentre quelli da $V_t$ a $V_s$ **Archi Inversi**.

$$\text{Archi Diretti:} \qquad A^+_{st} = \{(i,j) \in A : i \in V_s, j \in V_t\}$$
$$\text{Archi Inversi:} \qquad A^-_{st} = \{(i,j) \in A : i \in V_t, j \in V_s\}$$

Possiamo definire la **Capacità del taglio** come la somma delle capacità degli archi diretti del taglio, e quindi:

$$u(V_s, V_t) = \sum_{(i,j) \in A^+_{st}} u_{ij}$$

Mentre invece definiremo il valore del *flusso sul taglio*, come la differenza tra la somma dei flussi sugli archi diretti e la somma dei flussi sugli archi inversi:

$$x(V_s, V_t) = \sum_{(i,j) \in A^+_{st}} x_{ij} - \sum_{(i,j) \in A^-_{st}} x_{ij}$$

è inoltre importante aggiungere che il flusso sul taglio deve esser minore o uguale della capacità dello stesso taglio, e quindi vale la seguente relazione:

$$x(V_s, V_t) = \sum_{(i,j)\in A_{st}^+} x_{ij} - \sum_{(i,j)\in A_{st}^-} x_{ij} \leq \sum_{(i,j)\in A_{st}^+} x_{ij} \leq \sum_{(i,j)\in A_{st}^+} u_{ij} = u(V_s, V_t)$$

Tuttavia è possibile identificare una moltitudine di tagli ammissibili. Il problema del minimo taglio richiede infatti di determinare, tra tutti i possibili tagli del grafo, quello di capacità minima.

Presentiamo a tal proposito un esempio applicativo del problema di minimo taglio:
Ipotizziamo di avere a disposizione 2 processori sui quali esegua un programma suddiviso in 4 Thread. L'obiettivo è assegnare i thread ai processori in modo tale da minimizzare il costo totale, ovvero il costo di calcolo più il costo di comunicazione. Supponiamo di conoscere:

$\alpha_i \implies$ costo di esecuzione del modulo i sul primo processore, con i da 1 a 4

$\beta_i \implies$ costo di esecuzione del modulo i sul secondo processore, con i da 1 a 4

$c_{ij} \implies$ costo di comunicazione se i moduli i e j sono assegnati a processori diversi

Questo problema può esser ricondotto a quello di determinare un taglio di capacità minima su un particolare grafo orientato. e quindi per trovare un taglio di costo minimo bisogna effettuare un assegnamento dei thread ai due processori di costo totale minimo.

Per affrontare questi problemi è necessario introdurre il **Teorema debole del massimo flusso - minimo taglio**.
sia f(s,t) il valore del massimo flusso e $u(V_s, V_t)$ il valore del minimo taglio. Sommando i vincoli di bilancio dei nodi appartenenti all'insieme $V_s$ otteniamo che il massimo flusso è pari al valore del flusso sul taglio:

$$f(s,t) = \sum_{(i,j)\in A_{st}^+} x_{ij} - \sum_{(i,j)\in A_{st}^-} x_{ij}$$

Dal momento che il valore del flusso è non superiore alla capacità del taglio, allora ne consegue che:

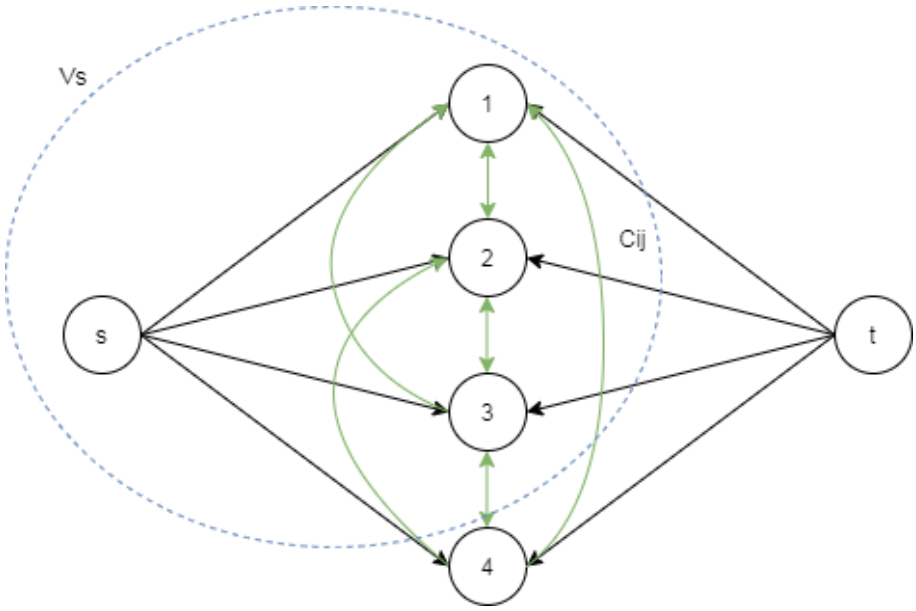



Figure 31: Problema di taglio applicato al sistema Processore-Moduli

$$f(s,t) \leq u(V_s, V_t)$$

è possibile però dimostrare un teorema che dia conclusioni più stringenti rispetto al teorema debole, che afferma che la soluzione ottima del problema del massimo flusso corrisponde al valore della soluzione ottima del minimo taglio.

Prima di poter dimostrare tale teorema, dobbiamo presentare un nuovo concetto, ovvero quello di **Grafo Residuo**.
Dato un flusso ammissibile x, il grafo residuo $G_x$ è un grafo con gli stessi nodi di G, mentre gli archi e le loro capacità sono definite come:

$$(i,j) \in A \qquad x_{ij} < u_{ij} \implies (i,j) \in A' \qquad r_{ij} = u_{ij} - x_{ij}$$

$$(i,j) \in A \qquad x_{ij} > 0 \implies (j,i) \in A' \qquad r_{ij} = x_{ij}$$

possiamo così definire $r_{ij}$, ovvero la **capacità residua**, come la differenza tra il massimo flusso e il flusso attualmente passante su quell'arco. Quindi a partire da un grafo G(V,A) è possibile costruire

un grafo resuduo G(V,A'), utilizzando gli stessi nodi del grafo originario, ma archi che rispettino i vincoli precedentemente posti, detti anche *archi non saturi*

Possiamo così definire il concetto di **Cammino aumentante**. Dato il grafo residuo $G_x$, un cammino aumentante è un cammino orientato da s a t sul grafico residuo. Sia $\delta$ il valore minimo delle capacità degli archi di un cammino aumentante P(s,t), il valore del flusso ammissibile può essere aumentato ponend:

$$x_{ij} = x_{ij} + \delta \qquad \text{se } (i,j) \in P \quad (i,j) \in A$$
$$x_{ij} = x_{ij} - \delta \qquad \text{se } (i,j) \in P \quad (j,i) \in A$$
$$x_{ij} = x_{ij} \qquad \text{se } (i,j) \notin P$$

Nel primo caso parleremo di *Archi concordi*, mentre nel secondo caso parleremo di *Archi discordi*.

Possiamo così definire il **Teorema del cammino aumentante**: Il flusso x su G ammissibile è ottimo se e solo se nel grafo residuo $G_x$ non esiste alcun cammino aumentante da s a t.

Infatti se esistesse un percorso P aumentante, potrei aumentare il valore di flusso sul grafo originario. Quindi si potrebbe incrementare del valore $\delta$ il valore del flusso, portandoci a concludere che x non sia ottima.
Posto inoltre $V_s$ l'insieme dei nodi raggiungibili da s nel grafo residuo e $V_t$ l'insieme dei restanti nodi, per definizione non può esserci sul grafo residuo nessun arco che vada da $V_s$ a $V_t$.
Sul grafo originario noi sappiamo che ogni arco (i,j) con $i \in V_s$ e $j \in V_t$ ha $xij = u_{ij}$, mentre ogni arco con $i \in V_t$ e $j \in V_s$ ha $x_{ij} = 0$. Il flusso sul taglio $(V_s, V_t)$ coincide con la capacità del taglio e quindi, per il teorema del massimo flusso minimo taglio in forma debole, coincide con la soluzione ottima.

Perciò il **Teorema forte del massimo flusso-minimo taglio** stabilisce che il flusso massimo da s a t è uguale alla capacità del taglio minimo

$$f(s,t) = \min_{V_s}(u(V_s, V - V_s))$$

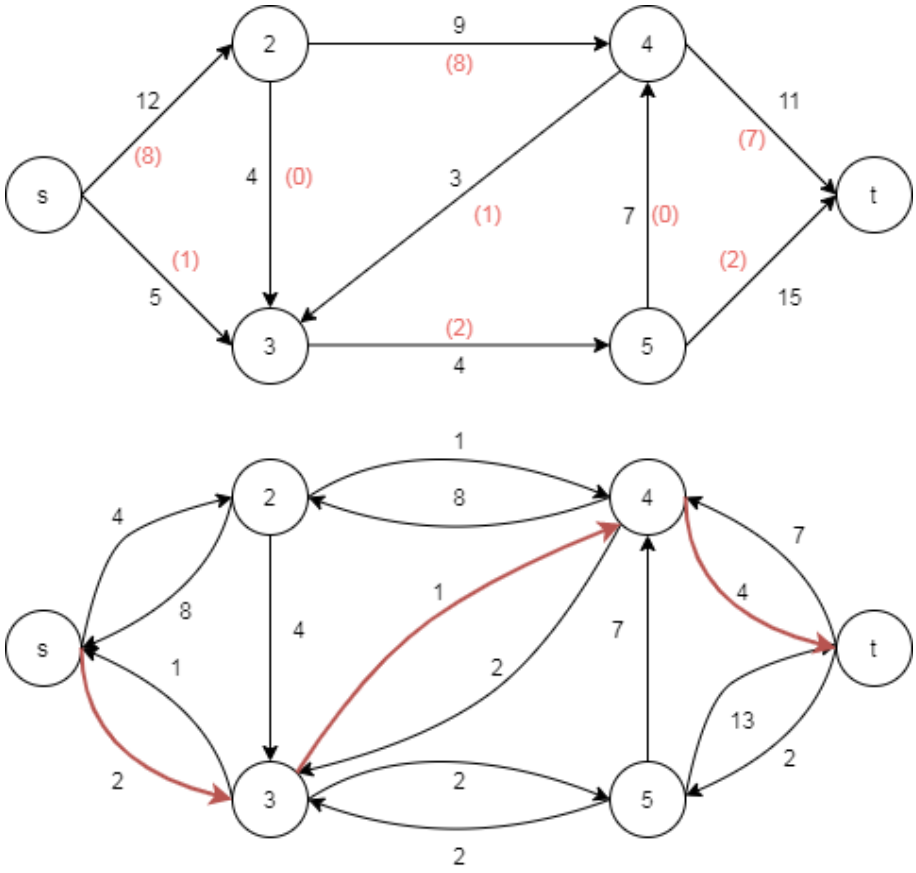



Figure 32: Grafo residuo e cammino aumentante

sia f(s,t) il valore del massimo flusso e sia invece u il valore del minimo taglio. Per il teorema del massimo flusso-minimo taglio in forma debole si ha la relazione che $f(s,t) \leq u(V_s, V_t)$.
Per il teorema del cammino aumentante visto in precedenza, al flusso x che definisce il massimo flusso f(s,t) deve esser associato un grafo residuo in cui non esiste alcun percorso aumentante. Allora il flusso ammissibile x deve saturare gli archi di un taglio, e perciò deve valere che :

$$f(s,t) = u(V_s, V_t)$$

### 4.2.3 Algoritmo di Ford-Fulkerson

Algoritmo efficiente per la risoluzione di problemi di massimo flusso, è l'algoritmo di **Ford-Fulkerson**.
Poniamo il flusso x pari a zero e costruiamo il grafo residuo $G_x$. L'algoritmo prevede i seguenti passi iterativi:

1. valuta se esista un cammino aumentante C sul grafo residuo $G_x$. Se non esiste termina l'algoritmo.
2. Calcola $\delta = \min_{(i,j)\in C} r_{ij}$ per ogni arco (i,j)$\in A$.
3. Se l'arco (i,j) appartiene al percorso C, allora incrementa il flusso $x_{ij}$ di $\delta$
4. Se l'arco (j,i) appartiene al percorso C, allora decrementa il flusso $x_{ij}$ di $\delta$
5. Aggiorna il grafo residuo $G_x$ e torna al punto 1.

L'algoritmo di Ford-Fulkerson quindi termina quando sul grafo residuo non esistono più percorsi che vadano da s a t. Se indichiamo con $V_s$ i nodi raggiungibili da s e con $V_t = V - V_s$ i restanti nodi, allora il taglio ottenuto $(V_s, V_t)$ è il taglio di capacità minima.Se le capacità massime degli archi sono numeri interi, allora l'algoritmo converege in un numero finito di iterazioni.

Nonostante sia garantita la convergenza per problemi interi, tale convergenza potrebbe esser *lenta*.
L'algoritmo prevede infatti di incrementare o decrementare il valore del flusso sul grafo originale del più piccolo valore di flusso presente, chiamato $\delta$. tuttavia può capitare che tale valore sia molto più piccolo rispetto a quello degli altri archi, portando la convergenza ad esser

estremamente lenta.

Per migliorare la convergenza dell'algoritmo è possibile usare un algoritmo diverso per il calcolo, ad ogni iterazione, del cammino aumentante sul grafo residuo, detto algoritmo di **Edmonds-Karp**, che calcola tra tutti i possibili cammini aumentanti quello con il minor numero di archi.

1. L'algoritmo effettua una visita a ventaglio del grafo residuo a partire dal nodo s
2. Fa uso di una lista Q dei nodi raggiunti nell'esplorazione del grafo e di un vettore p di nodi predecessori dei nodi raggiunti. Se un nodo presenta p = -1, allaro quel nodo non è stato raggiunto.
3. Ad ogni passo si estrae il primo nodo da Q e si aggiungono in fondo a Q tutti i nodi appartenenti alla stella uscente del nodo i-esimo selezionato, purchè essi non siano stati ancora raggiunti, ovvero non siano presenti in Q.
4. L'algoritmo termina o quando t entra in Q o quando Q si svuota. Nel primo caso il vettore p dei predecessori individua un cammino aumentante, mentre nel secondo caso non esiste un cammino aumentante, e quindi individuiamo un taglio di capacità minima, dato da:

$$V_s = \{i \in V : p_i \neq -1\} \qquad V_t = \{i \in V : p_i = -1\}$$

Presentiamo ora un esempio relativo ad un problema di massimo flusso, relativo al grafo in figura. In questo caso vogliamo risolvere il problema dal nodo 1 al nodo 7 applicando i due algoritmi presentati poc'anzi.
Inizialmente il grafo residuo $G_x$ corrisponde al grafo di partenza. Si applica quindi l'algoritmo di Edmonds-Karp, che eseguirà le seguenti iterazioni:

1. Viene inserito il nodo sorgente s nella coda. Q = {1}.
2. Successivamente si valuta la stella uscente da 1 e si inseriscono i nodi 2,3,4. I loro predecessori vengono quindi messi ad 1. Q = {2,3,4}
3. Alla terza iterazione si procede dal nodo 2 e si controlla la stella uscente dal nodo. Viene così rimosso il nodo 2 dalla coda e

aggiunto il nuovo nodo raggiunto, ovvero 5, il cui predecessore è posto a 2. Q = {3,4,5}

4. Alla quarta iterazione si analizzerà il nodo 3. La stella uscente comprenderà i nodi 2 e 5, già appartenenti alla coda. L'unico nodo che verrà aggiunto sarà il nodo 7, che essendo la destinazione, farà terminare l'algoritmo.

Avendo identificato un cammino aumentante 1-3-7, possiamo cercare in questo camminimo il minimo $\delta$, pari ad 8. è quindi possibile incrementare di 8 il flusso passante per i suddetti archi nel grafo originario e aggiornare così il grafo residuo $G_x$. Con lo stesso ragionamento si può applicare nuovamente l'algoritmo di Edmonds-Karp, che identificherà un nuovo percorso aumentante 1-2-5-7, con $\delta = 7$. Anche in questo caso verrà aumentato il valore del flusso sul percorso 1-2-5-7, per poi aggiornare il grafo residuo.

Continuando così, arrivati alla quinta iterazione, l'algoritmo di Edmonds Karp non sarà in grado di identificare un cammino aumentante, in quanto la coda Q si svuoterà prima di aver aggiunto il nodo 7. definendo così il valore di flusso massimo pari a 27 e l'insieme $V_s = \{1, 2\}$

## 4.3 Conclusioni

Entrambi i problemi visti fino ad ora si possono vedere come casi particolari di problemi di flusso detti **single commodity**: Sia dato un grafo G(N,A) in cui per ogni arco (i,j) si definisce $c_{ij}$ come il costo per unità di flusso che attraverso l'arco i,j,
$l_{ij}$ come la capacità inferiore dell'arco, ovvero un lower bound, e $u_{ij}$ la capacità superiore dell'arco, ovvero l'upper bound.

Ad ogni nodo del grafo è poi associato un peso $b_i$ che potrà assumere valore negativo qualora il nodo i-esimo richiedesse una quantità di flusso (destinazione), oppure positivo qualora il nodo i-esimo producesse una quantità di flusso (sorgente). Infine tale termine noto pul avere valore nullo nel caso in cui il flusso possa solo attraversare il nodo.
Nei problemi di questo tipo si vuole trasportare nel modo più conveniente, ovvero a costo minimo, flusso dai nodi origine ai nodi destinazione, rispettando i limiti di capacità sugli archi e le disponibilità dei nodi origine e delle richieste dei nodi destinazione.

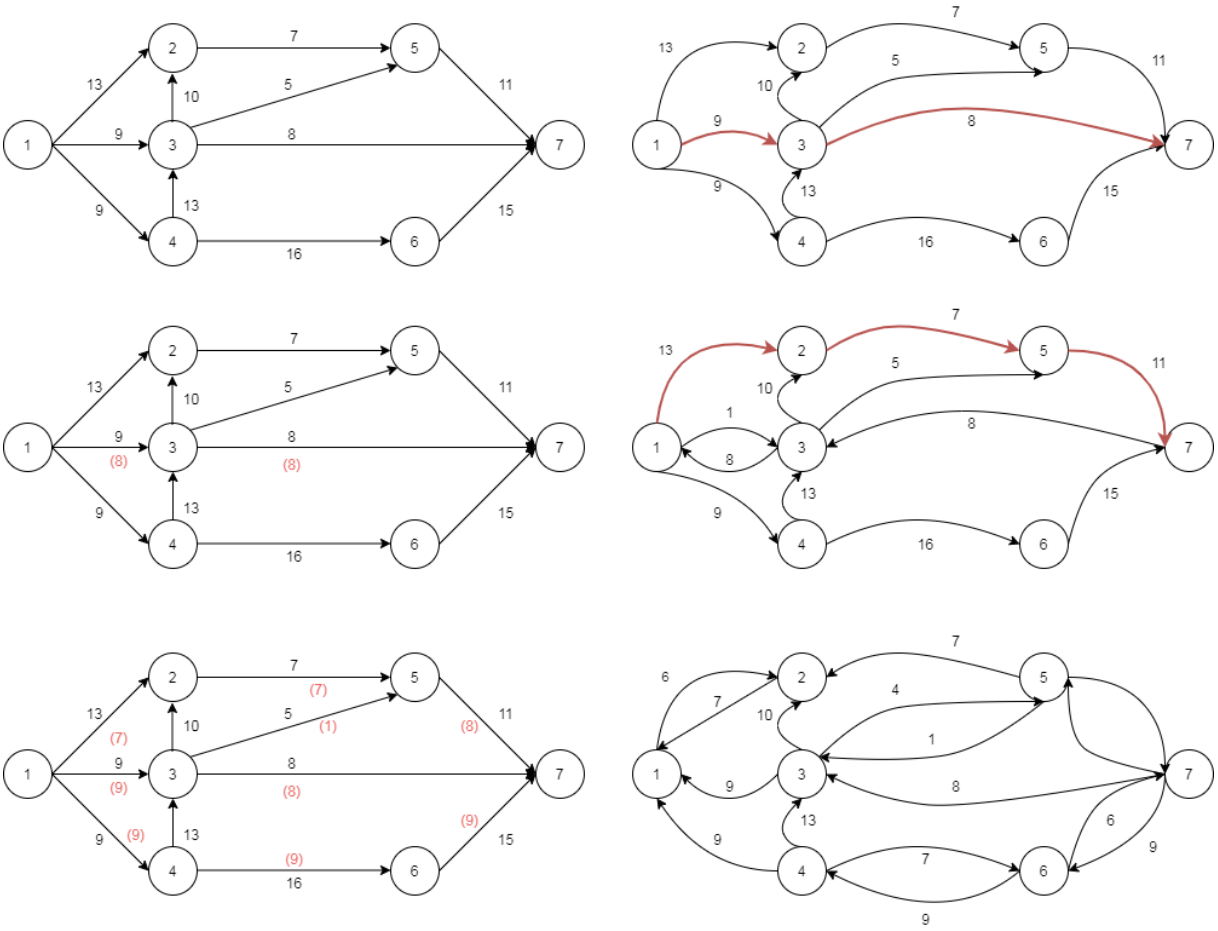



Figure 33: Iterazioni 1,2 e 5 dell'algoritmo di Ford-Fulkerson con Edmonds-Karp

Tali problemi possono in linea teorica esser rappresentati anche in forma matriciale o tabellare, dando una funzione di minimizzazione:

$$\min c^T x$$

e il sistema

$$Ex = b \qquad 0 \leq x \leq u$$

Di questi tipi di problemi abbiamo visto sia i problemi a cammino minimo che quelli a massimo flusso. è possibile però distinguere anche problemi di flusso detti a **Multi-commodity**. Quando abbiamo descritto il problema di flusso a costo minimo infatti abbiamo dato per scontato che nella rete viaggiasse un *singolo tipo di prodotto*. in realtà in molte applicazioni si ha che lungo la rete, rappresentata con un grafo orientato, viaggiano contemporaneamente più tipi di prodotti. E allora tali problemi altro non sono che l'estensione naturali di quelli presentati fino ad ora.

Supponiamo di aver r¿1 commodity, e che per ogni commodity k $\in \{1, \cdots, r\}$ vi sia un unico nodo sorgente, indicato con $s^k$ con valore associato $b^k$ e una unica destinazione $d^k$ con valore associato $-b^k$. Considereremo costo di trasporto $c_{ij}$ lungo gli archi di A a costo unitario, ed una capacità $u_{ij}$ assegnata.

Un esempio classico di problema multi-commodity è quello dei pacchetti su rete IP. In questo caso il protocollo adottato per la gestione della comunicazione è **Open Shortest Path First**-OSPF, che si occupa di determinare e assegnare i percorsi all'interno della rete alle avarie connessioni attive. I pacchetti che vadano da una stessa origine ad una stessa destinazione possono attraversare la rete seguendo percorsi diversi, viaggiando su flussi detti *splittable*.

Infatti, poste le premesse precedenti, risolvere un problema di flusso multi-commodity si traduce nel trovare il flusso per ogni commodity da inviare su ogni arco, in modo da soddisfare la domanda e minimizzare i costi complessivi, senza però violare la capacità degli archi stessi.
Il problema può quindi essere interpretato come n problemi di flusso, accoppiati dal vincolo di capacità su arco.

Matematicamente quindi potremmo modellare la variabile decisionale $x_{ij}^k \geq 0$ come la quantità di flusso relativo alla commodity k che attraversa l'arco ij. I vincoli potremo esprimerli, per la commodity k-esima, come nel caso della modellazione di un problema di flusso, e quindi:

$$\sum_{j \in \delta^+(i)} x_{ij}^k - \sum_{j \in \delta^-(i)} x_{ij}^k = \begin{cases} b^k & \text{se } i = s^k \\ -b^k & \text{se } i = d^k \\ 0 & \text{altrimenti} \end{cases} \qquad \forall i \in V, k \in \{1, 2, \cdots, r\}$$

$$\sum_{k=1}^{r} x_{ij}^k \leq u_{ij} \qquad \forall (i,j) \in A$$

possiamo infine definire la funzione obiettivo come:

$$\min \sum_{(i,j) \in A} \left( c_{ij} \sum_{k=1}^{r} x_{ij}^k \right)$$

# 5 Metodi Euristici

Nei casi in cui la ricerca della soluzione ottima possa essere troppo onerosa o addirittura impossibile, cercheremo di individuare di buona qualità in tempi ragionevoli, seppur non l'ottima.
Il classico problema di questo tipo è quello del **commesso viaggiatore**, o TSP.
Proveremo a modellare tale problema secondo l'approccio di programmazione lineare intera visto fino ad ora, per poi passare ad un approccio euristico. Il TSP è infatti il problema di ottimizzazione combinatoria più conosciuto e studiato in letteratura, ed è quindi il più adatto ad introdurre una ottimizzazione combinatoria.

## 5.1 Travelling Salesman Problem

Un commesso viaggiatore deve visitare un certo numero di città partendo dalla sua città di residenza e ritornando alla fine del suo giro dalla città da cui era partito. Vuole effettuare tutte le visite e tornare nella città di partenza percorrendo la lunghezza complessiva minima.Nonostante questo problema è estremamente semplice da definire,

risulta molto difficile da risolvere al crescere del numero di città.

Dato un grafo G(V,E), un **ciclo hamiltoniano** è un ciclo che attraversa tutti i nodi del grafo una ed una sola volta. Se ad ogni arco del grafo associamo un peso positivo $d_{uv}$, ad ogni ciclo hamiltoniano si può associare un costo dato dalla somma dei pesi degli archi che lo compongono.
Risolvere il problema TSP significa cercare il circuito hamiltoniano di costo minimo.

Se il grafo è pieno, il numero di circuiti costruibili è molto grande. Infatti dati n archi, il numero totale di percorsi sarebbe dato dalle permutazioni di tali archi , ovvero n!
A seconda delle caratteristiche del grafo parliamo di tsp *simmetrico* se il grado è non orientato, altrimenti *asimmetrico* se il grafo è orientato.

Come prima cosa partiamo col definire le variabili decisionali del problema. Possiamo definire la variabile $x_{ij} = 1$ se l'arco $(i, j) \in A$ appartiene al circuito hamiltoniano minimo, altrimenti essa varrà 0. L'obiettivo è chiaramente quello di minimizzare il costo del percorso:

$$\min \sum_{(i,j)\in A} d_{ij} x_{ij}$$

Per quanto ne concerne i vincoli c'è bisogno che in ogni nodo j entri un arco e che in ogni nodo j ne esca uno.

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad j \in V$$

$$\sum_{(j,i)\in A} x_{ji} = 1 \qquad j \in V$$

Questi vincoli prendono il nome di *vincoli di assegnamento*, che però non sono gli unici. è infatti fondamentale che il circuito hamiltoniano non ammetta sottogiri. è infatti possibile che e i vincoli d'assegnamento vengano rispettati, ma che al contempo non vengano rispettati i vincoli di assenza di sottogiri.

Dato un grafo orientato, è possibile che questo grafo contenga al suo interno dei cicli separati tra loro, il che porta ad avere soluzioni non

ammissibili. Quindi i vincoli di assenza di sottogiri si rendono necessari per tagliare fuori dall'insieme delle soluzioni ammissibili soluzioni che soddisfino i vincoli di assegnamento ma non ammissibili. È necessario imporre che il numero di archi che hanno origine e destinazione sui nodi sia minore o uguale della cardinalità dell'insieme dei nodi, meno 1. Così facendo siamo in grado di eliminare i sottogiri:

$$\sum_{i \in S, j \in S, (i,j) \in A} x_{ij} \leq |S| - 1 \qquad \forall S \subset V : 2 \leq |S| \leq |V| - 1$$

è chiaro che questo discorso vada fatto per ogni sottoinsieme di V, escluso ovviamente V.
Perciò se n è la cardinalità di V, si può costruire una stringa di n bit, dove un possibile sottinsieme è dato da una sottostringa contenente degli 1. Ne consegue che il numero di sottoinsiemi possibili sia pari a $2^n$, definendo perciò un numero esponenziale di vincoli di assenza di sottogiri.

Data perciò la complessità esponenziale, una strategia intelligente potrebbe essere quella di determinare dei *lower bounds*.
Un possibile Lower bound della soluzione è ottenibile semplicemente considerando una formulazione in cui non vi sia alcun vincolo di sottogiro.In tal caso la soluzione è ammissibile e ottima per il TSP se la soluzione non contiene sottogiri, altrimenti si ottiene solo un lower bound.

Sperimentalmente, si nota che dei $2^n$ vincoli , non tutti sono necessarii, ma è sufficiente aggiurne solo alcuni alla formulazioni per ottenere una soluzione ammissibile soddisfacente. Questa osservazione permette di pensare ad un algoritmo esatto per la soluzione del problema TSP asimmetrico, ovvero ATSP.

Parliamo di **algoritmo a generazioni di vincoli**, o *row generation*.

1. si eliminano tutti i vincoli di sottogiro e i vincoli di interezza e si risolve il corrispondente problema di assegnamento
2. si reintroducono nel modello i vincoli di interezza delle variabili

3. se la soluzione corrente non contiene sottocicli allora abbiamo trovato la soluzione ottima, quindi l'algoritmo termina
4. in caso contrario si individuano uno o più sottocicli nella soluzione corrente
5. si aggiungono al modello i vincoli di eliminazione di almeno uno dei sottocicli individuati
6. si risolve nuovamente il problema corrente e si ritorna al passo 3.

Perciò nel caso peggiore sarà necessario introdurre tutti i $2^n$ vincoli, e ad ogni risolvere un problema di PLI.
Naturalmente per capire se una soluzione sia ammissibile o meno, il modo più semplice è usare un algoritmo di ispezione di un grafo, che a partire da un nodo cerca di visitare tutti gli altri nodi, come nel caso di DSF.

## 5.2 Formulazione di problemi di ottimizzazione combinatoria

Un problema di ottimizzazione combinatoria riesce ad individuare B soluzioni, viste come un insieme finito detto insieme di base o **ground set**

$$B = \{b1, \cdots, b_n\}$$

Sia invece $\Sigma$ , una famiglia di sottoinsiemi di B detta **subset system**

$$\Sigma = \{s_1, \cdots, s_m\}$$

la funzione obiettivo invece associerà a ciascun insieme S di $\Sigma$ un numero reale w(S) .

$$\min_{S \in \Sigma} w(S) \qquad w : \Sigma \implies \mathscr{R}$$

Se riusciamo a definire questi oggetti parliamo allora di problema di ottimizzazione combinatoria, che solitamente è la minimizzazione di w(S).

Partendo dal presupposto che un qualsiasi problema di cammino minimo può essere rappresentato come un problema di ottimizzazione combinatorio, anche il problema TSP può esser visto in questa maniera. Il ground set B corrisponderebbe con E, ovvero l'insieme degli archi del grafo, necessario per definire le soluzioni del problema. Il sub set system sarebbe invece H, ovvero l'insieme di tutti i circuiti Hamiltoniani, che è a sua volta l'insieme dei sottoinsiemi di B. se il grafo G è pieno, m sarà circa n!
Alla fine la funzione obiettivo sarà:

$$w(H) = \sum_{u,v \in H} d_{uv}$$

Altro problema di ottimizzazione combinatoria molto discusso è quello del **Clustering**.
Dato un grafo non orientato G = (V,E) , e $w_{uv}$ il peso associato agli archi $\in E$, si vuole trovare la partizione di V in k classi, dette cluster, con k prefissato, che minimizzi la somma dei pesi degli archi incidenti in nodi appartenenti ad una stessa classe.

Data una partizione $\pi = \{C_1, C_2, \cdots, C_k\}$, si ha che:

$$\bigcup_{i=1\cdots k} C_i = V \qquad C_i \cap C_j = \emptyset \qquad \forall i,j = 1 \cdots k \quad i \neq j$$

dove il costo di un singolo cluster è definito come:

$$w(C_i) = \sum_{u,v \in C_i} w_{uv}$$

Mentre il costodella partizione può esser visto come:

$$w(\pi) = \sum_{i=1}^{k} w(C_i)$$

Dovendo trovare le partizioni $C_k$ dell'insieme dei nodi a costo minimo, (ovvero i singoli cluster), il problema del partizionamento di un grafo si può vedere come un problema di ottimizzazione combinatoria.

Il ground set è l'insieme delle coppie v,i dove v è un nodo appartenente a V, mentre invece i è il cluster i-esimo in cui viene inserito il nodo V.

$$B = \{(v,i) : v \in V, i = 1 \cdots k\} \qquad B = V \times \{1 \cdots k\}$$

Perciò il subset system è un sottoinsieme di V con determinate caratteristiche. Sicuramente ogni elemento di v deve appartenere ad una unica coppia dell'insieme Sj. Esiste perciò, ed è unico, il cluster tale che la coppia (v,i) appartenga ad $S_j$.

$$S_j \subset V : \quad \forall v \in V, \exists! i \in \{i \cdots k\} \quad (v,i) \in S_j$$

Alla fine la cardinalità del subset System $\Sigma$ sarà $m \approx k^n$
mentre la funzione obiettivo:

$$w(S) = \sum_{i=1}^{k} \sum_{(u,i) \in S (v,i) \in S} w_{uv}$$

## 5.3 Euristiche

La quasi totalità dei problemi di *ottimizzazione combinatoria* è di tipo *NP-hard* e quindi i tempi di calcolo per aver una soluzione ottima sono richiesti tempi di calcolo esponenziali.
Gli algoritmi euristici sono approssimanti e cercano una buona soluzione che non sia necessariamente ottima, ma abbia tempi di calcolo più contenuti.
La bontà di uan soluzione viene quindi valutata in termini di scarto percentuale tra la soluzione ottenuta e quella individuabile attraverso una tecnica esatta.

Perciò l'efficacia di una tecnica euristica viene valutata considerando due criteri in antitesi tra loro, ovvero la qualità della soluzione e i tempi di calcolo necessari per ottenerla.
Dunque il progetto di una euristica efficiente deve ricercare un giusto compromesso tra la velocità di calcolo e la bontà della soluzione, oltre a dover avere una bassa difficoltà implementativa, ed una buona flessibilità, intesa come la adattabilità dell'euristica all'applicazione in problemi diferenti.

Posto I una istanza di un dato problema P, sia EUR(I) il valore della soluzione fornita dall'euristica, e OPT(I) la soluzione ottima determinata da un algoritmo esatto. Come detto in precedenza, è

possibile stabilire un errore percentuale relativo, ovvero un *gap*, calcolabile come lo scostamento percentuale relativo tra la soluzione ottima e quella sub ottima ottenuta con una euristica.

$$gap = \frac{|OPT(I) - EUR(I)|}{|OPT(I)|}100$$

A questo punto gli algoritmi euristici possono essere classificati in primo luogo secondo l'errore. Algoritmi ad **errore massimo garantito** sono quegli algoritmi per i quali è possibile fornire un limite massimo all'errore, e hanno sostanzialmente una importanza prevalentemente teorica.

$$gap < \epsilon \quad \forall I$$

Algoritmi euristici con **stima dell'errore** sono invece algoritmi che data una istanza del problema ,forniscono una soluzione ammissibile ed una stima per eccesso, o upper bound,della distanza della soluzione fornita da quella ottima. è questo il caso delle *Euristiche lagrangiane*.

Per le euristiche che non danno alcuna stima dell'errore, la loro valutazione può essere effettuata generando casualmente una serie di istanze del problema di dimensioni diverse e andando a vlautare le *statistiche dell'errore* su queste istanze. In assenza di algoritmi esatti la valutazione dell'errore si può fare utilizzando procedure che forniscono *limiti alla soluzione ottima*.

Per esempio in un problema a minimizzare se si indica con LB(I) il valore di lower bound della soluzione ottima, si può ottenere una stima dell'errore per eccesso, come:

$$gap \leq \frac{|LB(I) - EUR(I)|}{|LB(I)|}100$$

Possiamo a questo punto suddividere gli algoritmi euristici in due classi principali: Quelli **costruttivi** che costruiscono gradualmente la soluzione attraverso il passaggio per soluzioni parziali, secondo un approccio *iterativo*. Abbiamo poi degli algoritmi **migliorativi**, che sono algoritmi che partendo da una soluzione ammissibile del problema, tentano di modificarla, migliorandola.

### 5.3.1 Algoritmi Greedy

Gli **Algoritmi Greedy**, o algoritmi *avidi*, sono algoritmi che ad ogni iterazione aggiungono un pezzo alla soluzione, classificandosi quindi come algoritmi costruttivi. Ad ogni iterazione, tra tutte le possibile aggiunte che possono contribuire alla *soluzione parziale*, viene aggiunto quello di costo inferiore, senza preoccuparsi della struttura complessiva della soluzione. Capita quindi che le ultime iterazioni risultino inefficienti dal momento che le possibilità di scelta si siano notevolmente ridotte.

Dato un problema di ottimizzazione combinatoria, definiamo l'insieme di base, B, il subset System S e la funzione obiettivo w.
Una **soluzione parziale** è un sottoinsieme T dell'insieme di base B che può essere ricondotto ad una soluzione ammissibile appartenente all'insieme S soltanto aggiungendo elementi di B.
Ma allora un algoritmo greedy non fa altro che costruire una sequenza di soluzioni parziali fino ad arrivare ad una soluzione ammissibile.
Una euristica greedy può quindi essere schematizzata come:

1. Inizializzazione in cui si pone $T_0 = \emptyset$ e i=1.
2. Si sceglie un elemento $e \in B - T_{i-1}$ tale che l'insieme $T_{i-1}$ con l'aggiunta dell'elemento e, sia una soluzione parziale e che w($Ti-1 \cup \{e\}$) sia *minimo*
3. Si pone $T_i = (Ti-1 \cup \{e\})$
4. Se $T_i$ appena identificato è una soluzione ammissibile, l'algoritmo termina
5. SAltrimenti, si pone i = i+1 e si ritorna al passo 2.

è chiaro che al passo 2 sia possibile che esistano più elementi che minimizzino il costo della nuova soluzione parziale, ed in questo caso è necessario una regola nel caso di pareggi, detta *Tie Breaking Rule*. A volte si sceglie semplicemente a caso, mentre altre volte può essere opportuno definire nuove funzioni di costo.

Volendo è possibile applicare una euristica greedy al caso del problema del viaggiatore. L'obiettivo è quello di trovare, ricordiamo, un ciclo hamiltoniano, al fine di avere una soluzione ammissibile.
Infatti, dato un grafo G, l'insieme di archi $S \subset A$ è un ciclo hamiltoniano se in ogni nodo di G incidono esattamente 2 archi di S e S non

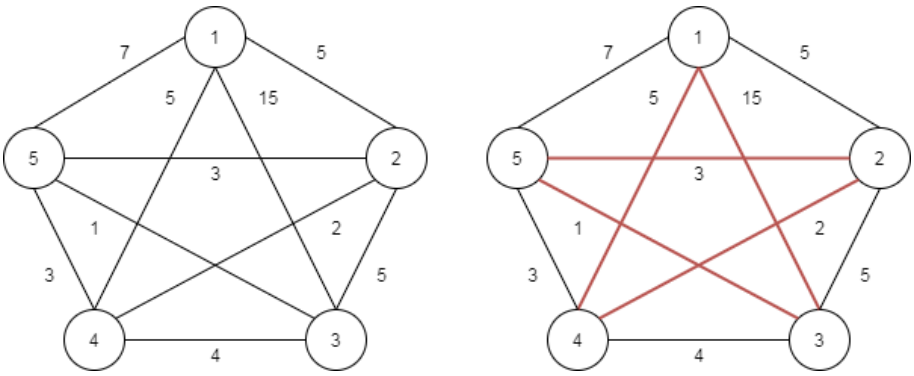



Figure 34: Problema del TSP con euristica Greedy

contiene cicli di cardinalità minore di V, l'insieme dei nodi.

Per applicare l'euristica Greedy bisogna definire una soluzione parziale T per il problema del TSP. In questo caso, una soluzione parziale quindi è tale se in ogni nodo di G incidono al più 2 archi di S e T non contiene cicli di cardinalità $< |V|$.

Dato un grado non orientato in figura, la prima soluzione parziale che rispetti le proprietà elencate, la si ottiene scegliendo l'arco con il costo più basso, che in figura è quello 3-5. La nuova soluzione parziale viene aggiunta alla sottosoluzione, con in aggiunta il suo costo.

$$T = \{(3,5)\} \qquad W = 1$$

Ad una seconda iterazione si procede scegliendo dall'insieme degli archi, quello di costo minore, in maniera tale che rispetti le proprietà che definiscano la soluzione parziale. Banalmente l'arco 2-4 può essere aggiunto a T, incrementando il costo di 2.

$$T = \{(3,5),(2,4)\} \qquad W = 4$$

Ad una successiva iterazione è ancora banale l'aggiunta dell'arco 2-5 di costo 6.
La situazione comincia a complicarsi a partire dalla quarta iterazione, quando L'arco 5-4 , che ha un costo minore, non può essere aggiunto alle soluzioni parziali, poiché chiuderebbe un ciclo,oltre a determinare tre archi incidenti nel nodo 5.
Allora si passa a selezionare il secondo più conveniente, ovvero l'arco

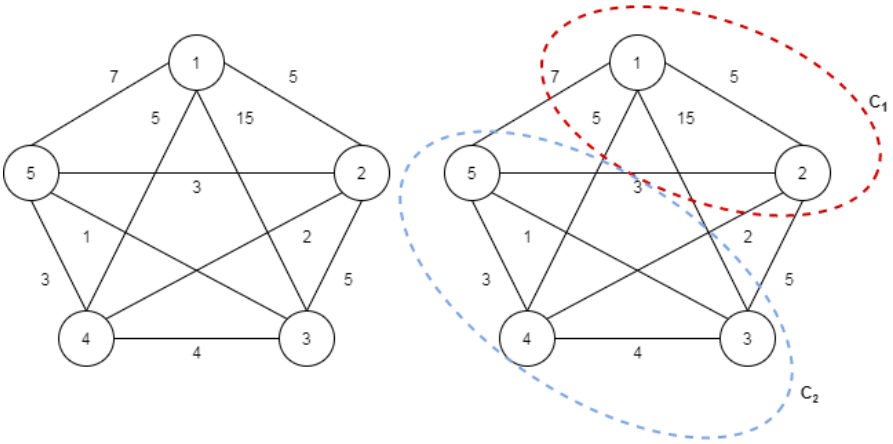



Figure 35: Problema del 2-clustering con euristica Greedy

4-3, ma anche qui si giungerebbe alla chiusura di un ciclo.
Restano perciò gli archi di valore 5, dei quali sono l'arco 1-4 è valido per la costruzione della soluzione parziale. è possibile quindi continuare il ragionamento fintanto che la soluzione ottenuta non sia una soluzione ammissibile, ovvero un ciclo hamiltoniano.

$$T = \{(3,5),(2,4),(2,5),(1,4),(1,3)\} \qquad W = 26$$

Non possiamo però stimare quanto "buona" sia questa soluzione. infatti nonostante i tempi di calcolo siano piuttosto veloci, non vi è alcuna garanzia che questa sia la soluzione ottima, ne tantomeno quanto questa si discosti dalla soluzione ottima.

Altro problema nel quale è possibile applicare euristiche greedy è quello del *2-Clustering*, modellabile come un problema di ottimizzazione combinatoria.
Definiamo l'insieme delle soluzioni parziali T come insieme vuoto.Ad ogni iterazione l'insieme verrà costruito aggiungendo delle coppie *nodo-cluster*, dove nel problema specifico considereremo solo 2 cluster, $C_1$ e $C_2$

Nella prima iterazione è possibile aggiungere un qualunque nodo al cluste $C_1$, senza alterare il costo della soluzione parziale, in quanto

ancora nessun nodo arco sarà contenuto nel cluster.possiamo allora scegliere di aggiungere il nodo 1 nel cluster 1, aggiungendo a Tla coppia 1-1.
Nella seconda iterazione è possibile nuovamente prendere un qualsiasi altro nodo e metterlo nel cluster 2, lasciando il costo a zero.
In questo caso però la scelta sarà più oculata. Converrebbe infatti aggiungere al cluster 2 il nodo collegato al Nodo 1 già presente in $C_1$ avente arco a peso maggiore. Mettendo in $C_2$ il nodo 3, infatti, si evita la possibità di avere nello stesso cluster l'arco con peso 15.

Arrivati alla terza iterazioen sarà necessario scegliere l'arco a costo minimo da aggiungere al cluster specifico.
Si può quindi aggiungere 5 nel cluster 2, così da aumentare il costo della funzione obiettivo solo di 1.
Successivamente aggiungere il nodo 2 al cluster 2, farebbe aumentare sia di 3 che di 5 il valore della funzione obiettivo, a causa degli archi 2-5 e 2-3. Aggiungendo invece il nodo 2 al primo cluster, l'incremento sarebbe solo di 5, e ciò sarebbe più vantaggioso. Procedendo con questo ragionamento si può inserire 4 nel cluster 2, ottenendo così una soluzione ammissibile in quanto ogni nodo del grafo sarà stato aggiunto nel Cluster.

Anche in questo caso però, non vi è garanzia del fatto che la soluzione individuata sia ottima per il problema.

### 5.3.2 Algoritmi Migliorativi

Le euristiche di **Ricerca locale** si basano su concetti di **intorno** di una soluzione ammissibile corrente.
Algoritmo di questo tipo restituisce in output un minimo locale del problema, che non è detto sia l'ottimo. Perciò gli algoritmi di ricerca , data una soluzione corrente $H_0 \in S$, costruisconoun intorno di $H_0$ tramite soluzioni vicine ad $H_0$.
Si ispeziona quindi un intorno di $H_0$ e si calcola la migliore soluzione $H_1$ appartenente all'intorno.
se $H_1$ coincide con $H_0$ l'euristica termina, altrimenti si itera il procedimento ripartendo da $H_1$.

In particolare il punto di partenza è fondamentale poiché potrebbe

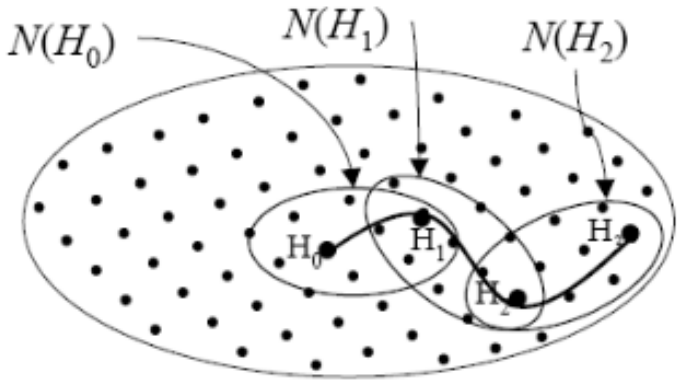



Figure 36: Visualizzazione insiemistica di una ricerca locale

influire sul punto di convergenza dell'euristica. Al contempo la scelta della grandezza dell'intorno è determinante, in quanto consentirebbe di identificare un set di punti maggiore, e quindi eventualmente punti migliori. Tuttavia Allargare l'intorno si traduce nell'ispezionare più soluzioni ammissibili, e quindi la singola iterazione diventa computazionalmente più onerosa.

Poniamo due definizioni fondamentali intermanete ad una euristica di ricerca locale.
Una **mossa** è una operazione che a partire dalla soluzione corrente consente di generare altre soluzioni ammissibili per il problema di partenza con alcune caratteristiche simili alla soluzione di partenza.
Un **intorno** di una soluzione H è invece l'insieme constituito da tutte le soluzioni ottenibili applicando una determinata mossa alla soluzione H.

Allora possiamo strutturare una euristica di ricerca locale come:

1. Si sceglie una soluzione iniziale H detta di innesco, per il processo di ricerca
2. Si definisce un intorno N(H) della soluzioen corrente ottenuto applicando una data mossa alla soluzione corrente
3. Si individua una nuova soluzione $H' \in S$ cui corrisponde il minimo della funzione obiettivo.
   $H0 : w(H') = \min w(H) \quad \forall H \in S$

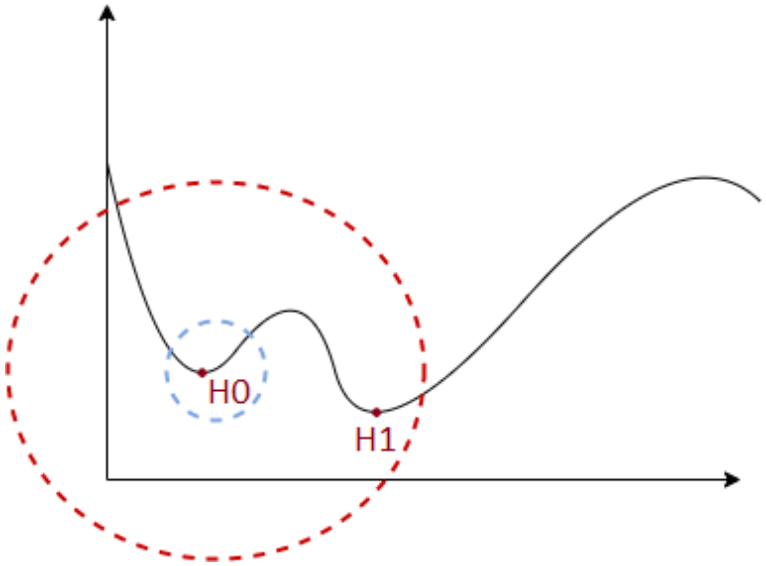



Figure 37: Visualizzazione grafica di una ricerca locale

4. se $w(H') < w(H)$, in un problema a minimizzare, si sostituisce H con H', per poi ritornare al passo 2, altrimenti si assume la soluzione H come soluzione finale.

Volendo applicare tale euristica per il problema del TSP, è necessario definire la mossa del **2 scambio**.
Data una soluzione ammissibile per il TSP, se rimuoviamo due archi non adiacenti la soluzione parziale può essere completata in due modi distinti, uno solo dei quali produce un nuovo ciclo.
Sia $H_0 = \{(u_1, u_2), \cdots, (u_{q-1}, u_q), (u_q, u_1))\}$ un generico ciclo hamiltoniano. La mossa di 2-scambio non fa altro che scegliere una coppia di archi non adiacenti $(u_i, u_{i+1}), (u_j, u_{j+1})$, rimuovere la coppia di archi dal ciclo, e aggiungere due nuovi archi $(u_i, u_j), (u_{i+1}, u_{j+1})$

Allora per il TSP bisogna in primo luogo scegliere un ciclo hamiltoniano $H_0 \in T$ e imporre i = 1. Alla iterazione i-esima si considera $H_i$ il più corto ciclo hamiltoniano ottenibile a partire da $H_{i-1}$ con il 2-scambio. Se la funzione $w(H_i) \geq w(H_{i-1})$ allora l'algoritmo termina e il ciclo $H_{i-1}$ è il migliore trovato fino a questo punto, altrimenti si incrementa i e si reitera l'algoritmo.

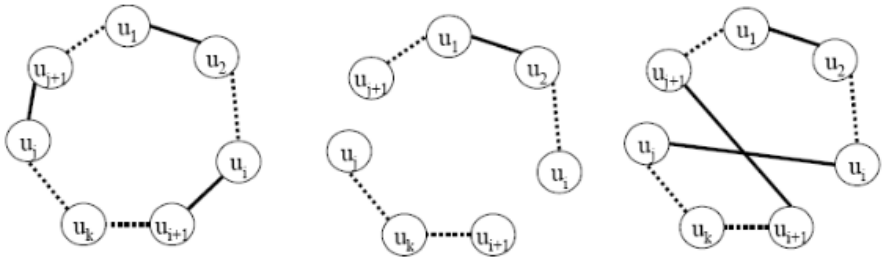



Figure 38: Mossa del 2 scambio

Per migliorare la qualità delle soluzioni di una euristica di ricerca locale è quindi possibile applicare più volte l'algoritmo a partire da diverse soluzioni iniziali, oppure, come già anticipato, aumentare la dimensione degli intorni. Nel primo caso parliamo di **Approccio Multi-start**, dove le soluzioni iniziali possono essere costruite in modo casuale, oppure in modo guidato, al fine di distribuirle su tutto lo spazio delle soluzioni. Se il problema presenta numerosi punti di minimo locale, ed un unico minimo globale (nascosto dai molti minimi locali vicini tra loro), diventa difficile raggiugnere questo picco, a meno di non scegliere una opportuna soluzione iniziale.

In generale all'aumentare della *dimensione degli intorni* considerati migliora quindi la qualità delle soluzioni, ma peggiora il tempo di calcolo. Nel caso del TSP, una 2-op creerà una dimensione do ogni intorno del tipo O($n^2$), mentre nel caso di 3-op avremo intorni di dimensione dell'ordin di O($n^3$).
Ciò significa banalmente che per un grafo di 100 nodi, tipicamente piccolo, ogni esplorazione dell'intorno 3-opt richiederà 1000000 di valutazioni della funzione obiettivo!

naturalmente sarebbe possibile generalizzare ad il caso di k op, in maniera tale che l'intorno contenga tutti i possibili circuiti Hamiltoniani. In tal caso è ovvio allora che il tempo di calcolo sarebbe del tipo O(n!), assolutamente inaccettabile.

e allora bisogna stabilire come ottenere un trade off tra *accuratezza* e *tempo di calcolo*. Si può pensare di **Diversificare**, esplorando regioni diverse tra loro, come accade ad esempio nell'approccio multi start, oppure **Intensificare**, aumentando la dimensione degli intorni

ed esplorando a fondo una unica regione.
Naturalmente la scelta tra l'una e l'altra dipende dalle performance dell'algoritmo per il problem affrontato, il che ci porta a dire che non esiste una regola assoluta per effettuare una decisione, e che quindi ci saranno lunghe fasi di tuning dei parametri.

Dunque per poter scegliere la soluzione corrente in maniera efficiente, si possono adottare due politiche diverse.
La prima politica, detta di **First improvement** prevede di effettuare lo scambio tra la soluzione corrente e la prima soluzione migliroativa trovata. Ciò significa naturalmente che solo l'ultima iterazione ispezionerà per intero l'intorno.
In alternativa si può adottare una politica di **Best improvement**, che prevede l'ispezione di un intorno della soluzione corrente,valutando funzione obiettivo in tutti i punti di questo intorno, per poi prendere il punto in cui corrisponde il valore minimo di funzione obiettivo.

### 5.3.3 Tabu Search

L'euristica migliorativa **Tabu Search** per cercare di superare un minimo locale, vero tallone d'achille degli algoritmi migliorativi, può scegliere, iterazione per iterazione, la migliore soluzione dell'intorno anche se caratterizzata da un valore *peggiore* della funzione obiettivo, rispetto alla soluzione corrente.
Una soluzione del genere permette infatti di potersi spostare ulteriormente nell'insieme di definizione della funzione, cercando per eventuali minimi relativi inferiori.

La criticità di tale soluzione è da ricercare nel fatto che solitamente, dato H2 il punto "peggiorativo" ed H1 il precedente punto migliore, H1 cadrà nell'intorno di H2. Quindi ad una successiva iterazione H1 potrebbe essere riselezionato, causando una oscillazione dell'algoritmo. Bisogna inoltre stabilire dei criteri d'arresto, per evitare che l'algoritmo proceda arbitrariamente a cercare soluzioni peggiori.

Dunque bisogna stabilire dopo quante iterazioni di ricerca della soluzione peggiore l'algoritmo possa convergere, e bisogna inoltre conservare una *memoria delle soluzioni visitate*, in modo da evitare di ripassare per tali soluzioni.

Seguendo quest criteri, l'euristica Tabu Search è in grado di migliorare l'efficienza del classico algoritmo di ricerca locale, memorizzando informazinoi che riguardano il percorso già effettuato.
Proponiamo lo schema relativo alla ricerca appena presentata, molto simile al caso di minimo locale.

1. Si sceglie una soluzione iniziale S di innesco dell'algoritmo, cosi da inizializzare il procedimento
2. Alla generica iterazione k si definisce un intorno $N(S, k)$ delal soluzione corrente S
3. Si individua una nuova soluzione S' appartenente all'intorno, tale che sia il migliore sulla base di uno *stimatore* E[N(S,k)], e si sostituisce S con S'
4. Se viene soddisfatto un dato criterio d'arresto l'algoritmo termina e restituisce la migliore soluzione calcolata, altrimenti k viene incrementato e si torna al passo 2.

Lo **stimatore** è una funzione, spesso identica alla funzione obiettivo, o al più una sua versione semplificata, che permetta una valutazione rapida della qualità della soluzione.
Lo stimatore perciò rappresenta un criterio di valutazione delle soluzioni che ad ogni iterazione permette di scegliere la soluzione migliore, facendolo in un tempo minore rispetto a valutazioni classiche.

È poi importante il modo in cui viene creato l'intorno e come da un intorno si elimini la soluzione già ispezionata. Infatti abbiamo già visto che all'aumentare della dimensione dell'intorno ,la valutazione diventa molto più dispendiosa.
Inoltre per evitare le oscillazioni è necessario salvare in delle liste, dette **Liste Tabu**, tutte le soluzioni esaminate ad ogni iterazione, così da confrontarle con la soluzione corrente. Una soluzione che è stata eliminata dall'intorno, al fine di evitare oscillazione, prende il nome di soluzione **Tabu**, da cui il nome dell'euristica.

Però in un problema come il TSP ogni soluzione è una permutazione, e quindi con tempi di calcolo irragionevoli. E allora invece di memorizzare completamente tutte le soluzioni già ispezionate, si potrebbero memorizzare solo alcune caratteristiche di queste soluzioni,

sintetiche a sufficienza. Tali dettagli vengono chiamati *attributi*.

Consideriamo un problema di TSP relativo a 10 città ed S la soluzione corrente con gni soluzione permutazione delle 10 città. Ipotizziamo che valutando l'intorno di S si scopra che la mossa migliore da fare sia quella di swappare il nodo 3 con il 7.
Tale soluzione andrebbe inserita nella lista tabu : la mossa di swap tra 3 e 7 diventerà adessso Tabù.

Tale scelta è però più approssimativa, in quanto non sarà solo S ad esser eliminata dal prossimo intorno, ma anche altre soluzioni che non verranno ispezionate, potenzialmente migliori di S.
Perciò gli attributi memorizzati nella lista relativi alla mossa/soluzione saranno:

- A1 - La posizione del primo nodo dello swap
- A2 - La posizione del secondo nodo dello swap
- A3 - Tutte le mosse che coinvolgono il nodo in A1, oppure quello in A2
- A4 - Tutte le mosse che coinvolgono il nodo in A1 e quello in A3

Tali liste sono anche dette *Memorie a breve termine*, in quanto queste non potranno contenere un numero arbitrariamente alto di valori.
Per evitare di non visitare intere regioni dello spazio delle soluzioni infatti, le liste tabu vengono implementate come code FIFO di lunghezza TL. Se un attributo entra in lista all'iterazione k, esso ne uscirà all'iterazione k+TL.

Il valore di TL allora è ceramente un parametro critico e dipenderà dalla dimensione del problema da risolvere. Il TL può esser scelto **Staticamente**, e quindi rimanere lo stesso per tutta la durata dell'algoritmo, oppure **dinamicamente**, variando in corso d'opera.

Possiamo così procedere nel definire un **Criterio d'aspirazione**, ovvero una condizione che quando si verifica autorizza l'esecuzione di mosse tabu.
Ad esempio se una mossa tabu dovesse consentire il raggiungimento di un valore di funzione obiettivo migliore della soluzione esaminata,

essa verrebbe eseguita, anche se tabu.

IMMAGINE SCHEMA TABU SEARCH

Così come accadeva nel caso di ricerca locale, anche in tabu search, dopo un certo numero di iterazioni, potrebbe non esser possibile trovare soluzioni migliori di quelle già ispezionate. Al fine di migliorare le prestazione si potrebbe adottare sia una strategia di ricerca in intorni già visitati, ma con attributi meno restrittivi, (intensificazione), oppure spostarsi in regioni di ricerca nuove, del tutto assimilabile al multi-start (diversificazione) .

Possiamo quindi descrivere gli *elementi fondamentali* per una tabu search.

1. Si sceglie una soluzione iniziale S e si definisce un criterio per valutare le soluzioni
2. Si definisce la mossa che individui l'intorno N
3. Si definiscono le caratteristiche della struttura di memoria a breve termine, come la sua lunghezza.
4. Si definiscono i metodi di innesco e di implementazione di eventuali fasi di intensificazione e diversificazione
5. Si determina il criterio di arresto

### 5.3.4 Algoritmi Threshold e Simulated Annealing

Gli algoritmi Threshold, o a **Soglia**, prevedono di partire da una soluzione iniziale, considerando , iterazione per iterazione, la migliore soluzione appartenente ad un intorno mediante una soglia.
La soglia $T_k$, viene modificata ad ogni iterazione, e all'inizio dell'algoritmo essa è presa abbastanza grande.
Tale scelta è dovuta al fatto che nelle prime iterazioni non bisogna subito scende verso un minimo locale, ma bensì capire in quale regione del dominio di ammissibilità si trovino le soluzioni migliori.
Può infatti capitare, specialmente nelle prime iterazioni, che la soluzioni peggiore sia scelta come soluzione corrente, col semplice scopo di spostarsi nel dominio di ammissibilità.

Tale euristica prevede quindi di generare una soluzione $x' \in N(x)$ e se la distanza tra la funzione valutata in x (soluzione corrente) e

x' (soluzione candidata) è minore della soglia, allora x' è scelta come nuova soluzione.

$$f(x') - f(x) < t_k \implies x = x'$$

l'algoritmo prosegue fintanto che un criterio di arresto non è soddisfatto.
nonostante $t_k$ parta da un valore abbastanza grande, essa, iterazione per iterazione, diventerà sempre più piccola, tendendo a zero.

$$\lim_{k \to \infty} t_k = 0$$

Se invece la valutazione di tale soglia non avviene in maniera deterministica, ma solo con una certa *probabilità*, ci stiamo riferendo ad un algoritmo naturale che prende nome di **Simulated annealing**, dove per algoritmi naturali intendiamo algoritmi di ottimizzazione chesimulano degli eventi che accadono normalmente in natura, come ad esempio gli algoritmi genetici.

Come nell'algoritmo a soglia, è necessario definire la successione $t_k$ di soglia che permetta di scegliere una nuova soluzione , seppur peggiore , della precedente.
Nella simulated,però, tale successione è probabilistica
Parleremo quindi *probabilità di accettazione di soluzione*. Se nell'intorno di x viene individuata una soluzione x', la nuova soluzione sarà tale solo con una certa probabilità.

$$P(x') = \begin{cases} 1 & f(x') \leq f(x) \\ e^{-\frac{f(x')-f(x)}{t_k}} & f(x') > f(x) \end{cases}$$

Inizialmente il valore di $t_k$ sarà elevato a sufficienza, e diminuirà iterazione per iterazione, così da selezionare con certezza la soluzione migliore solo qualora il numero di iterazioni fosse sufficientemente alto.

Essendo questo un algoritmo naturale, come già detto, è possibile trovarne un applicazione nel mondo reale, come nel caso di *Processo di tempra di un solido.*
Preso un metallo che si riscaldi fino al punto di fusione, le particelle di questo metallo si distribuiscono in manieradel tutto casuale con l'innalzarsi della temperatura.
Se il raffreddamento successivo avviene in tempi ampi, le particelle

del solido hanno il tempo di raggiungere il loro stato di equilibrio termico.
Al contrario, se il raffreddamento è veloce, le particelle non si disporranno bene per mancanza di tempo, e si formerà perciò un cristallo imperfetto.
Un algoritmo simulativo di tale fenomeno fu proposto nel 1953, detto **Algoritmo di Metropolis**.

A partire dallo stato corrente S alla temperatura T, definito dalla posizione delle molecole, viene applicata una perturbazione spostando casualmente una molecola in una nuova posizione. Il sistema raggiunge così il nuovo stato S', caratterizzato da un diverso valore di energia.
Il nuovo stato viene accettato come stato corrente con una probabilità data da:

$$P(x') = \begin{cases} 1 & \Delta E \leq 0 \\ e^{-\frac{\Delta E}{KT}} & \Delta E > 0 \end{cases}$$

Nell'algoritmo di Metropolis la temperatura T viene abbassata gradualmente e l'algoritmo termina quando il valore di T è tale da rendere inaccettabile una qualsiasi perturbazione peggiorativa.
Il valore di energia del solido svolge quindi il ruolo di funzione obiettivo, mentre lo stato corrente del solito sarà la soluzione corrente del problema.
Lo spostamento di una molecola in una nuova posizione, e quindi la perturbazione, rappresenta invece la mossa che individua la nuova soluzione, rendendo l'ultimo parametro, ovvero la un valore di controllo dell'algoritmo. (mentre K è la costante di Boltzman)

La simulated Annealing applica l'algoritmo di metropolis a problemi di ottimizzazione combinatoria, stabilendo le seguenti fasi:

1. Si individua una soluzione iniziale S di innesco
2. Si definisce l'operazione che permetta di individuare casualmente una soluzione S' nell'intorno della soluzione corrente S
3. Si verifica la probabilità di accettazione della mossa, stabilendo quindi la nuova soluzione verrà sostituita dalla S', con probabilità

$$P(x') \leq \begin{cases} 1 & \Delta f \leq 0 \\ e^{-\frac{\Delta f}{T}} & \Delta f > 0 \end{cases}$$

Naturalmente data la temperaturail parametro di soglia, e stando simulando un problema di raffreddamento, è necessario stabilire un insieme di regole che definiscano tale processo.
L'insieme di queste regole prende il nome di **Cooling schedules**, che stabilisce il valore di temperatura iniziale $T_0$ (molto alta in quanto modellante un metallo fuso), quello finale $T_f$ e il numero di transizioni $L_k$ da effettuare per ogni valore $T_k$, nonche una legge di decremento di T.

Il valore del parametro $T_0$ deve essere scelto in modo tale che, nella fase iniziale dell'algoritmo, tutte le transizioni siano accettate. Deve dunque risultare che

$$e^{-\frac{\Delta f}{T_0}} \approx 1 \qquad \forall \Delta f > 0$$

Per garantire questa condizione si può considerare il valore di funzione obiettivo $f_0$ della soluzione iniziale e porre un valore di riferimento per la massima variazione della funzione obiettivo, ad esempio $\delta f_{max} = |f_0/2|$.
Ciò significa accettare funzioni peggiorative che non superino il 50% del valore di $f_0$.
Si può quindi stabilire un valore di $T_0$ pari a $10\Delta f_{max}$,in modo tale da assicurare, per transizioni caratterizzate da un incremento di funzione obiettivo pari al 50% del valore iniziale, che la probabilità di accettazione sia praticamente unitaria.

L'altra decisione riguarda invece il valore di destinazione $T_f$. Arrivati alla fine del processo la probabilità di accettazione di soluzioni peggiorative deve essere ovviamente circa nulla

$$e^{-\frac{\Delta f}{T_0}} \approx 0 \qquad \forall \Delta f > 0$$

Per garantire tale condizione si pone nuovamente un valore di riferimento per la minima variazione della funzione obiettivo, ad esempio $\Delta f_{min} = 10^{-3}|f_0|$.
A questo punto $T_f$ può esser posto pari a $10^{-4}|f_0|$, in modo tale che alla fine la probabilità di accettazione che presenti incremento di funzione obiettivo obiettivo pari allo 0.1 % del valore iniziale sia praticamente nulla.

Per concludere sarà necessario stabilire come passare da un valore $T_i$ di temperatura a quello successivo
La regola più frequentemente utilizzata per decrementare T dalla generica iterazione k alla successiva è $T_{k+1} = \alpha T_k$, con $\alpha < 1$.
I falori di $L_k$ relativi alle transizioni invece, vanno scelti conseguentemente. Infatti se $\alpha$ è prossimo ad 1 allora il decremento sarà lento lento, con un $L_k$ relativamente basso, altrimenti se $\alpha$ è basso (minore di 0.8 già può esser considerato basso) $L_k$ sarà più elevato.
$L_k$ rappresenta in particolare il numero di mosse o soluzioni generate da accettare o meno, per uno stesso valore di $T_k$, ripristinando le condizioni di equilibrio termico.

E allora una possibile soluzione è quella di considerare $L_k = L$ costante e correlato alle dimensioni del problema N, con L ad esempio pari proprio ad N. In altre proposte $L_k$ viene scelto in modo da assicurare che per ogni valore di $T_k$ il numero delle transizioni accettate sia almeno pari ad un certo valore $\mu_{min}$.

### 5.3.5 Algoritmi Genetici

Gli **Algoritmi Genetici** si basano su una analogia tra la soluzione di problemi di ottimizzazione combinatoria ed i meccanismi di selezione naturale in campo genetico.
L'idea è quella di considerare una popolazione di soluzioni che evolva in accordo con un meccanismo di selezione naturale per produrre soluzioni con buoni valori di funzioni obiettivo.

Viene quindi simulato il processo di evoluzione naturale, in cui le caratteristiche di un organismo sono determinate dai *geni* presenti nei suoi *cromosomi*. Ciascun gene può assumere *alleli* diversi che producono differenze delle caratteristiche associate a quel gene.
L'insieme dei geni è detto *Genotipo*,mentre il *fitness*, dipendente dal genotipo, indica la capacità dell'individuo di adattarsi all'ambiente esterno.

Perciò di generazione in generazione tali caratteristiche evolveranno, alterando le caratteristiche genetiche, affinchè possano adattarsi meglio al proprio ambiente.

L'evoluzione di una popolazione è inoltre legata al processo di *riproduzione*. Durante la loro vita gli individui, detti genitori, si accoppiano producendo nuovi individui, figli, il cui patrimonio genetico è combinazione di quello dei genitori.
Tali figli, una volta generati, subiscono delle mutazioni oltre che ereditare il patrimonio genetico. Infatti la *legge di selezione naturale* si basa sul principio che, tra i nuovi individui, hanno maggiore probabilità di sopravvivere e generare nuovi individui quelli che possiedono una migliore fitness, e dunque una maggiore capacità di adattamento.

Gli algoritmi genetici altro non fanno che simulare il processo di selezione naturale partendo da una soluzione iniziale ed applicando **Operatori genetici**. Possiamo infatti sancire delle corrispondenze tra i termini relativi all'evoluzione genetica e quelli utilizzati da un algoritmo genetico:

| Evoluzione genetica | Algoritmo genetico |
|---|---|
| Gene | Variabile decisionale |
| Allele | Valore delle variabili decisionali |
| Cromosoma | Insieme delle variabili decisionali |
| Genotipo | Soluzione ammissibile |
| Fitness | Funzione obiettivo |
| Accoppiamento | Crossover |
| Influenza dell'ambiente | Altri operatori genetici |
| Selezione Naturale | Algoritmo |

Per quanto ne concerne l'algoritmo ,questo passa per diverse fasi elaborative.
In primo luogo è necessario **Codificare il problema**, e quindi stabilire i geni del problema, ovvero le variabili di decisioni. Si rappresenta così una soluzione del problema in termini di stringa di variabili decisionali, indicando i possibili valori di ciascuna variabile

A questo punto bisogna generare un insieme di possibili soluzioni che formi la popolazione iniziale, per poi associare ad ogni individuo un certo **valore di fitness**.
La difficoltà di questa selezione consiste nel trovare un set di soluzioni ammissibili iniziali dal quale far evolvere il sistema, il che non è banale.

Successivametne, nel corso dell'algoritmo, si selezionano **coppie di soluzioni** della popolazione, tipicamente a maggior valore di fitness,

alle quali applicare gli operatori genetici, ovvero crossover e mutazione. A partire da tale coppia si genereranno nuove soluzioni, ovvero i figli. Possono allora esser imposti alcuni vincoli relativi alla *crescita della popolazione*, come ad esempio il sostituire le soluzioni a basso fitness esistenti della popolazione con le soluzioni prodotte nella fase di generazione.

Nella codifica del problema, la stringa generata potrebbe esser composta da variabili binari, ciascuno delle quali rappresentante un allele; naturalmente le codifiche sono realizzate ad hoc, e devono tener conto delle caratteristiche del problema. Ad esempio una variabile binaria potrebbe esser sufficiente a descrivere il sesso di un individuo, ma non lo sarebbe per determinare il colore degli occhi dello stesso individuo.

Ad esempio nel problema del TSP ad N città, una soluzione può esser rappresentata da una stringa di n elementi che indichi l'ordine secondo il quale verranno visitate le città.

Per poter selezionare gli individui a maggior fitness si ricorre solitamente ad un metodo probabilistico, detto **Simulazione Montecarlo**, nelal quale si calcola per ciascun elemento i della popolazione, il *valore cumulato di fitness* $F_i$, a partire da quello cumulato totale F.

$$F_i = \frac{\sum_{k=1}^{i} f_i}{F} \qquad F = \sum_{i=1}^{n} f_i$$

Si generano quindi due valori casuali a e b compresi tra 0 e 1. e si selezionano gli elementi s e t della popolazione che rispettino le seguenti condizioni.

$$F_{s-1} \leq F_s \qquad F_{t-1} \leq b \leq F_t$$

Geometricamente tale selezione è accomunabile ad una circonfrerenza di raggio unitario e suddivisa in archi di dimensione diversa, rappresentanti la probabilità di selezionare un dato individuo. Tanto maggiore è la fitness, tanto maggiore sarà lo spicchio di circonferenza associato a quell'individuo.

Nel caso di problemi a minimizzare, occorre semplicemente sostituire $f_i$ con il suo reciproco $1/f_i$, lasciando il problema inalterato.

A ciascuna di queste coppie selezionate saranno poi applicati i due operatori relativi alla riproduzione, quello di **Crossover** e quello di **Mutazione**.
L'operatore di crossover accoppia due soluzioni generandone altre due che presentano un *patrimonio genetico*, e quindi valori delle variabili decisionali, dedotti da quelli dei genitori.

Se i due genitori sono rappresentati da stringhe di valori, l'operatore di crossover può scegliere casualmente un **punto di taglio** all'interno delle due stringhe rappresentanti i due individui, detto **Crosspoint**.
Una volta fatto ciò, per generare l'individuo, l'operatore di crossover unisce le due sottostringhe tagliate.
In generale è possibile implementare l'operatore di crossover usando m crosspoint , in modo tale da generare più soluzioni combinando in modo diverso le m+1 sottostringhe individuate.

Altro operatore fondamentale è quello di mutazione,che prevede di modificare casualmente il valore di una variabile decisionale all'interno della stringa. Tale scelta corrisponde in binariad un set/reset del bit, mentre in caso contrario, ad una scelta di unnuovo valore di allele tra i possibili valori che esso può assumere.

La necessità della mutazione è necessaria in quanto si potrebbe averee, in poche iterazioni, individui simili tra loro, ottenendo il fenomeno di *convergenza prematura*. Questo significa che non vi è stato tempo di ispezionare tutta la regione di ammissibilità, il che per un algoritmo genetico non è l'ideale.
Bisogna per questo stabilire una certa probabilità di mutazione, che sia ragionevole al punto da non generare solo valori casuali, perché altrimenti l'intera struttura genetica con crossover tra soluzioni ottime andrebbe a risultare superflua, e al contempo per non arrivare a convergenza prematura.

Quindi si fissa una probabilità di mutazione per ogni gene e si applica la simulazione montecarlo per verificare se ciascun gene debba esser modificato, posti valori di probabilità di mutazione sufficientemente bassi da non perdere patrimonio genetico.

è possibile prevedere un altro operatore, detto **filtro**, che è un

**Single Crossover**

$$\begin{array}{l|l} (0\ 0\ 0\ 0 & 1\ 1\ 1) \\ (0\ 1\ 0\ 1 & 0\ 1\ 0) \end{array} \quad \Rightarrow \quad \begin{array}{l} (0\ 0\ 0\ 0\ 0\ 1\ 0) \\ (0\ 1\ 0\ 1\ 1\ 1\ 1) \end{array}$$

**Double Crossover**

$$\begin{array}{l|l|l} (0\ 0 & 0\ 0 & 1\ 1\ 1) \\ (0\ 1 & 0\ 1 & 0\ 1\ 0) \end{array} \quad \Rightarrow \quad \begin{array}{l} (0\ 0\ 0\ 1\ 1\ 1\ 1) \\ (0\ 1\ 0\ 0\ 0\ 1\ 0) \end{array}$$

**Mutazione**

$$\begin{array}{l} (0\ 1\ 0\ 0\ 0\ 1\ 0) \\ \qquad\ \downarrow \\ (0\ 1\ 1\ 0\ 0\ 1\ 0) \end{array}$$

Figure 39: Crossover e Mutazione

operatore di *mutazione particolare* che trasforma uan soluzione inammissibile in una ammissibile, e si rende necessario quando un operatore di mutazione o crossover genera una soluzione non ammissibile.

Volendo fare un esempio per il TSP: l'operatore filtro cotruisce un discendente scegliendo due punti di taglio *casualmente*. Per esempio i due genitori vengono tagliati per il terzo e il quarto elemento e tra il settimo e l'ottavo.

$$p_1 = (123|456|89) \qquad p_2 = (452|1876|93)$$

La parte compresa tra i tagli viene scambiata:

$$\sigma_1 = (- - - - |1876| - -) \qquad \sigma_2 = (- - - - |4567| - -)$$

Questo passaggio definisce delle corrispondenze tra le città scambiate 1-4, 8-5, 7-6, 6-7, e dunque i discendenti saranno:

$$\sigma_1 = (423187659) \qquad \sigma_2 = (182456793)$$

Altro operatore da poter presentare è quello di **inversione**, che data una stringa e scelti casualmente due crosspoint, inverte l'ordine degli alleli presenti nella sottostringa individuata. Qualora vengano generate stringhe non ammissibili si può ricorrere ad un filtro, oppure stabilire di accettare soluzioni non ammissibili nella popolazione, sottraendo alla loro fitness un *termine di penalità*.

A questo punto, come già detto, la popolazione deve evolvere attraverso il progressivo rinnovo degli elementi. A tal fine infatti è necessario individuare gli individui da sostituire, scegliendo o per fitness più bassa, o applicando il metodo montecarlo per eliminare con maggior probabilità gli individui con bassa fitness. L'algoritmo quindi converge quando gli elementi della popolazione sono tutti più o meno simili. In tal caso l'oepratore dicrossover cessa di produrre nuovi genotimi e l'algoritmo rimane confinato all'attuale insieme limitato della regione ammissibile.

La formulazione dell'algoritmo introdotta fino ad ora è chiaramente solo una rappresentazione basilare per algoritmi genetici.
Potrebbe esser infatti possibile inserire numerosi altri fattori di modifica, dovuti ad esempio a fenomeni non naturali, come interventi culturali, educativi ecc.. simulabili magari tramite altre euristiche di

ricerca locale, aumentando la fitness del singolo individuo.
Combinazione di algoritmi genetici e algoritmi di ricerca locale, finalizzati ad incrementare la fitness, prendono il nome di **algoritmi memetici**.

## 5.4 Problemi di Vehicle Routing

Un problema fondamentale nell'ambito dell'ottimizzazione combinatoria riguarda il problema di **Vehicle Routing**, o problema di *distribuzione*.
Tale problema presenta molteplici formulazioni, che si possono sostanzialmente vedere come una generalizzazione del TSP oppure come una fusione tra il TSP e il clustering.

Tali problemi di distribuzione consistono nell'utilizzo di una flotta di veicoli con capacità limitata per prelevare o consegnare merci o persone in determinati punti.
Obiettivi possono esser la minimizzazione dei costi di trasporto, come la dstanza percorsa o il consumo di carburante, o anche minimizzare il numero di veicoli utilizzati. Dal lato massimizzazinoe si potrebbe invece massimizzare la qualità del servizio, come la velocità di consegna.

Il problema può esser quindi rappresentato su di un grafo *orientato e pieno* G di nodi V ed archi E.
Nell'insieme dei nodi, uno di questi giocherà ruolo particolare, ovvero quello di **Nodo deposito**, a partire dal quale la flotta inizia e termina il suo percorso. Tutti gli altri nodi sono detti **Punti vendita**, e sono quei nodi per i quali transiteranno i veicoli della flotta.

Ad ogni arco, rappresentante il collegamento esistente tra due nodi, è associata una *distanza*, o costo, $c_j$ di valore non negativo. Inoltre ogni punto vendita $v_i$ sarà caratterizzato da una certa domanda di merci $d_i$, fatta eccezione per il nodo deposito, che chiaramente non avrà alcuna domanda associata.

Infine per effettuare le consegne si avranno a disposizione m *veicoli*, caratterizzati da una *portata massima*, $Q_i$ ovvero la quantità massima di materiale che un veicolo i può trasportare. Per risolvere questo problema bisogna anzitutto stabilire come assegnare i punti vendita ai veicoli e dunque clusterizzare i clienti.

Ogni cluster verrà perciò associato ad un veicolo che servirà quello specifico insieme seguendo il percorso migliore tra tutti i possibili percorribili.
Identificare tale percorso minimo internamente ad un cluster coincide con il risolvere un problema del TSP relativo al veicolo servente del cluster specifico.

A questo punto una soluzione ammissibile a tale problema è rappresentato da una famiglia di sottoinsieme dei nodi di V, ovvero i Cluster.

$$\mathscr{C} = \{C_1, \cdots, C_m\}$$

Dato un sottoinsieme di nodi $C_i$, ogni punto di vendita potrà essere incluso una sola volta in un unico cluster. Fa eccezione il nodo deposito, che dovendo essere punto di arrivo e di ritorno per ogni cluster, esso dovrà appartenere ad ogni sottoinsieme.
Per ogni sottoinsieme $C_h \in \mathscr{C}$ assegnato al veicolo h, la domanda del punto vendita in $C_h$ non dovrà eccedere la portata massima del veicolo $Q_h$

### 5.4.1 Risoluzione esatta del Vehicle Routing

Fatta questa premessa possiamo pensare di modellare il problema secondo un approccio esatto prima ancora di usare mete euristiche.
Posta C la famiglia di cluster, sia $L(C_h)$ la *lunghezza del minimo ciclo hamiltoniano* su $C_h$. Il valore della funzione obiettivo allora sarà:

$$Z(C) = \sum_{h=1}^{m} L(C_h)$$

che consiste nel risolvere m volte il problema del TSP. Per modellare il problema bisogna stabilire come prima cosa le variabili decisionali, che prenderanno il nome di **Variabili d'incidenza**.Tali variabili possono esser modellate a doppio indice come $Y_{ik}$, che assumano valore 1 qualora il nodo i-esimo sia associato al cluster k-esimo, 0 altrimenti.
Si può successivamente definire il **Vettore di incidenza** $y_k$ costituito dalle singole variabili di incidenza nel cluster k-esimo:

$$y_k = (y_{1k}, y_{2k}, \cdots, y_{nk})$$

Cioè i nodi di V che devono esser percorsi dal veicolo k, con $y_{1k}$ il nodo deposito.
Posta la lunghezza del circuito hamiltoniano minimo $L(C_k) = f(y_k)$, ovvero funzione del vettore di incidenza, e quindi dei nodi di V appartenenti al cluster k-esimo, si può modellare il problema come un problema di minimizzazione:

$$\min \sum_{k=1}^{m} f(y_k)$$

Bisogna ora stabilire i vincoli di questo problema: Come detto in precedenza, il deposito deve appartenere a tutti i sottoinsiemi di C, ciò significa che la somma di tutte le variabii di incidenza relative al deposito negli m-cluster deve esser pari ad m, in quanto in ogni cluster sarà presente il nodo deposito.

$$\sum_{k=1}^{m} y_{1k} = m$$

Al contempo deve accadere che ogni variabile incidente debba apparire una sola volta per cluster, da cui

$$\sum_{k=1}^{m} y_{ik} = 1 \qquad i \in \{1, \cdots, m\}$$

Infine, come già detto, la somma sulle richieste nei singoli nodi deve essere maggiorata dalla portata disponibile dal veicolo k esimo che serve quel cluster.

$$\sum_{i=2}^{n} d_i y_{ik} \leq Q_k \qquad k \in \{1, \cdots, m\}$$

Questo modello tuttavia non è lineare in quanto la funzione obiettivo non è lineare, poichè per definirne il valore è necessario risolvere gli m problemi di ottimizzazione di TSP.
Per linearizzare il problema sarà necessario porre al posto di ciascuna di $f(y_k)$ il modello del TSP corrispondente.
A tal fine si può far riferimento a delle nuovi variabili decisionali referenti all'arco appartenente ad un circuito hamiltoniano, internamente al cluster k-esimo.:

$$\begin{cases} x_{ijk} = 1 & \text{se l'arco (i,j) è percorso dal veicolo k} \\ x_{ijk} = 0 & \text{altrimenti} \end{cases}$$

è allora possibile esplodere la funzione $f(y_k)$, ovvero il costo del circuito hamiltoniano minimo sul sottoinsieme $C_k$:

$$f(y_k) = min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ijk}$$

Per quanto ne concerne i vincoli invece, è ovvio che se i deve esser servito dal veicolo k, allora un arco uscente dal nodo i deve esser necessariamente attraversato dal veicolo k.

$$\sum_{j=1}^{n} x_{ijk} = y_{ik} \qquad i = 1, \cdots, n$$

Il secondo vincolo è logicamente uguale, ma in questo caso si riferisce ai nodi entranti, e quindi eve esistere un arco entrante nel nodo i percorso dal veicolo k, se il nodo i appartiene al cluster $C_k$

$$\sum_{j=1}^{n} x_{jik} = y_{ik} \qquad i = 1, \cdots, n$$

L'ultima cosa da imporre cono i vincoli relativi all'assenza di sottogiri. Per ogni sottoinsieme di V che non contenga il nodo deposito, la somma degli archi con origine e destinazione in questo sottoinsieme non può superare la cardinalità di S-1, altrimenti si avrebbero dei cicli.

$$\sum_{i,j \in S} x_{ijk} \leq |S| - 1 \qquad S \subset V - \{v_1\}, \quad |S| \geq 2$$

$$x_{ijk} \in \{0, 1\} \qquad i, j \in \{1, \cdots, n\} \quad k = \{1, \cdots, m\}$$

Combinando le due rappresentazioni appena presentate possiamo pervenire ad una forma risolutiva del problema di distribuzione in forma esatta, secondo una risoluzione PL:

$$\min \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ijk}$$

$$\sum_{i,j \in S} x_{ijk} = \sum_{j=1}^{n} x_{jik} \qquad i = 1, \cdots, n \quad k = 1, \cdots, m$$

$$\sum_{i,j \in S} x_{ijk} \leq |S| - 1 \qquad S \subset V - \{v_1\}, \quad |S| \geq 2, \quad k = 1, \cdots, m$$

$$\sum_{k=1}^{m} \sum_{j=2}^{n} x_{1jk} = m \qquad \sum_{k=1}^{m} \sum_{j=1}^{n} x_{ijk} = 1 \qquad i = 2, \cdots, n$$

$$\sum_{i=2}^{n} \sum_{j=1}^{n} d_i x_{ijk} \leq Q_k \qquad k = 1, \cdots, m$$

$$x_{ijk} \in \{0, 1\} \qquad i, j \in \{1, \cdots, n\} \quad k = \{1, \cdots, m\}$$

### 5.4.2 Varianti sugli approcci esatti

Esistono molte varianti a questo problema che tengono in conto di diversi fattori ; in ciascun punto vendita abbiamo ipotizzato la consegna di un bene, ma in realtà si potrebbero prendere in considerazione anche casi di prelievo o di consegna e prelievo contemporaneamente. Ciò ovviamente si traduce nel dover verificare per ogni nodo il soddisfarsi del vincolo di capacità del veicolo, che preleverà e depositerà in ogni nodo (*Pickup and Delivery*).

Altre varianti sono relative alle caratteristiche della flotta, che potrebbe avere un numero variabile di elementi con una funzione obiettivo non fissa. La stessa flotta potrebbe avere anche veicoli con portata differente, e quindi alcune zone potrebbero esser raggiungibili da alcuni veicoli e altre da veicoli differenti.
Infine altra caratteristica plausibile è quella relativa al numero di depositi, che può essere singolo, oppure multiplo.

Altri vincoli da poter prendere in considerazione sono le strutture delle rotte dei veicoli. Infatti si può stabilire una distanza massima percorribile da un veicolo, causata, ad esempio, dai vincoli di numero di ore di attività di un autista, ma anche precedenze di percorrenze tra nodi, specialmente in pickup and delivery.
Potremmo tra l'altro considerare vincoli di carattere temporale, in qui bisogna servire un cliente durante specifiche finestre temporali. In tali problemi il tempo di percorrenza $t_{ij}$ di ogni arco ed il tempo di servizio

$t_i$ presso ogni nodo diventano elementi fondamentali del problema.

A questo punto le funzioni obiettivo possibili sono relative non solo alla minimizzazione di distanza/tempo di percorrenza, ma anche al numero di veicoli o la sicurezza del trasporto.

Immaginiamo ad esempio di avere un numero di veicoli variabili : il vincolo che stabilisce che dal deposito escano m veicoli, diventa un vincolo di minore uguale e non più di uguaglianza.
Partiranno allora al massimo m veicoli e la funzione obiettivo dovrà prevedere un termine di penalità per ogni veicolo utilizzato.

$$\sum_{k=1}^{m}\sum_{j=1}^{n} x_{ijk} \leq m$$

$$\sum_{k=1}^{m} f_k \sum_{j=1}^{n} x_{1jk}$$

Posto $f_k$ il costo di utilizzo del generico veicolo k, Tale sommatoria è pari a 1 se dal nodo 1 verso il nodo j partirà un veicolo k. la sommatoria interna varrà zero se non ci sarà alcun veicolo che rispetti tale vincolo.

Potrebbe esser necessario aggiungere un vicolo relativo alla durata massima di ogni percorso. Un autusta k-esimo può infatti lavorare solo per un numero $T_k$ limitato di ore.
E allora posti $t_i$ il tempo di servizio nel nodo $v_i$ e $t_{ij}$ il tempo di attraversamento dell'arco (i,j) vale il seguente vincolo:

$$\sum_{i=1}^{n} t_i \sum_{j=1}^{n} x_{ijk} + \sum_{i=1}^{n}\sum_{j=1}^{n} t_{ij} x_{ijk} \leq T_k$$

I $t_i$ vengono sommati per ogni punto vendita che viene servito dal veicolo k, mentre, per quanto ne concerne i tempi di viaggio,se $t_{ij}$ è il tempo per percorrere l'arco $x_{ij}$, si dovrà sommare tale termine di tempo se il veicolo k percorre l'arco ij.

Una *finestra temporale* associata al generico nodo i, è un intervallo $[l_i, L_i]$ , con $l_i$ *istante minimo di servizio* ed $L_i$ *istante massimo di servizio.*
Ciò si traduce nell' aggiungere un nuovo insieme di variabili, $a_i$, che

rappresente quale istante è quello di arrivo nel nodo i-esimo, che viene quindi servito.
La durata totale T della finestra sarà calcolata come la massima tra le disponibilità di tempo $T_k$ del k-esimo autistca.

Bisogna inoltre collegare i due insiemi di variabili decisionali $a_i$ ed $x_{ijk}$

$$a_j \geq (a_i + t_i + t_{ij}) - (1 - x_{ijk})T \qquad i, j \in \{1, \cdots, n\}, \quad k = \{1, \cdots, m\}$$

$$a_j \leq (a_i + t_i + t_{ij}) + (1 - x_{ijk})T \qquad i, j \in \{1, \cdots, n\}, \quad k = \{1, \cdots, m\}$$

$$l_j \leq a_j \leq L_j$$

Il tempo necessario per giungere nel nodo j, e quindi il tempo di inizio servizio $a_j$ sarà almeno pari alll'istante in cui si è arrivati nel generico nodo i, più il tempo per servire quel nodo, più il tempo per spostarsi da i a j.
Se subito dopo aver servito i ci si fosse spostati in j, il legame sarebbe diventato di uguaglianza, e il secondo termine diventerebbe pari a zero. Se così non fosse però il veicolo k potrebbe aver servito i e j in ordine non immediatamente sequenziale, oppure i potrebbe non esser stato servito affatto.

T viene quindi scelto arbitrariamente grande, in maniera tale da imporre $a_j \geq -\infty$ , e la seconda disequazione $a_j \leq \infty$ permettendo di non imporre di fatto alcun vincolo qualora $x_{ijk} \neq 1$. Infatti il significato di questi due vincoli serve a restringere i tempi di $a_i$ e $a_j$.
Non potendo scegliere un valore arbitrariamente alto, T viene scelto solitamente pari al tempo massimo tra i $T_k$, come detto in precedenza.

Per concludere possiamo dire che, dato che il numero di vincoli aumenta con un ordine del tipo $n^2m$, in queste varianti del problema di distribuzione base il numero di vincoli diventa sempre più grande, per questo l'utilizzo di euristiche diventa fondamentale per ottenere risultati in tempi ragionevoli.

### 5.4.3 Risoluzione euristica del Vehicle Routing

Abbiamo già in precedenza accennato al fatto che il Vehicle Routing può esser visto già visto come l'ottimizzazione di un problema di clustering e uno di tsp; di conseguenza esistono diversi metodi elaborati

per poter risolvere euristicamente questo problema.

I **metodi in due fasi** risolvono separatamente la fase di problema clustering, e quella di tsp.
Verranno quindi applicate due euristiche, una per il problema di clustering che determinerà i sottoinsimi di Cluster $\mathscr{C} = \{\mathscr{C}, \cdots, \mathscr{C}\}$ ed una per il TSP che individuerà i cammini hamiltoniani minimi.
Parliamo allora di euristiche *Cluster first, route second* **CFRS** ,o il viceversa, **RFCS**, in base all'ordine di esecuzione delle due fasi.

Iniziamo presentando un paio di semplici *euristiche greedy*, che permettano di risolvere il problema.
Immaginiamo di avere un solo veicolo con una certa capacità Q che deve servire tutti i nodi. Si procede a determinare il cammino hamiltoniano e ci si ferma non appena si termina la capacità disponibile del veicolo, che tornerà al deposito.
Verrà successivamente fatto partire un nuovo veicolo che continuerà il ciclo hamiltoniano da dove il precedentemente veicolo si era fermato.
Una soluzione del genere avrà il costo di una 2opt, che è un calcolo veloce,ma non necessariamente eccelso. Si può però costruire così una serie di cammini hamiltoniani, ed in seguito clusterizzarli. Questo allora è un semplice esempio di RFCS.

Viceversa, in una euristica del tipo CFRS possiamo rappresentare le posizioni dei nodi nel piano cartesiano usando *coordinate polari*, usando come primo valore la distanza dall'origine $\rho$, e come secondo valore $\theta$, ovvero l'angolo.
Quindi al variare di theta, si percorre interamente il piano.Quando l'asse di rotazione incontrerà dei punti vendita, questi verranno aggiunti nel cluster. All'aumentare del numero di nodi nel generico cluster $C_i$ verrà aumentata la capacità di questo cluster, fino a saturare la capacità Q del veicolo.
Il cluster verrà quindi chiuso, associando alla porzione di piano attraversata dal raggio con spaziatura $\theta$, un Cluster specifico.
Successivamente viene immesso un nuovo veicolo di capacità Q, e si torna a far spaziare l'asse, identificando nuovi cluster secondo lo stesso principio.
Terminato il clustering, si potrù stabilire il ciclo hamiltoniano internamente ad ogni cluster.

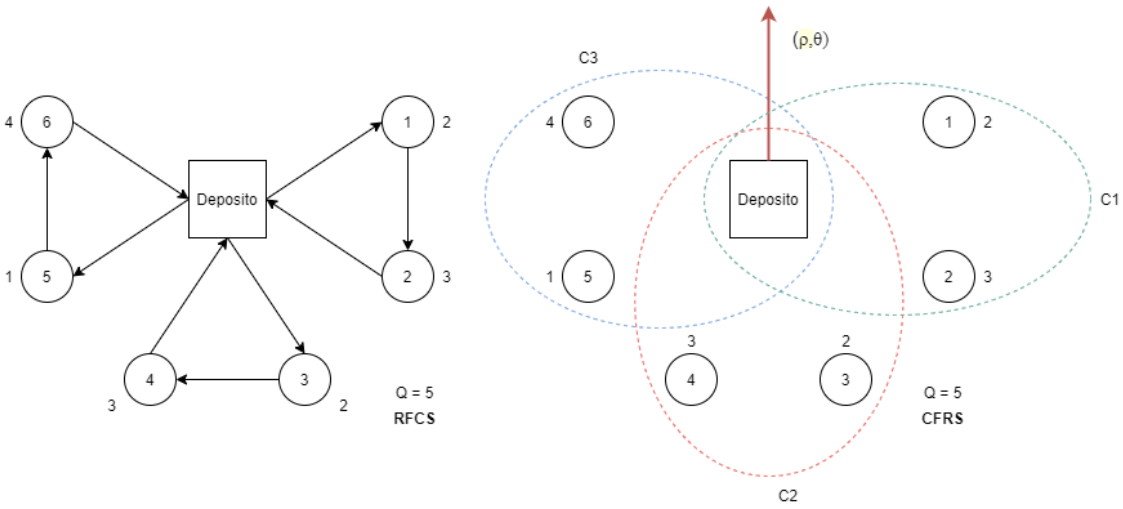



Figure 40: Euristiche Greedy per il VR

Nonostante la buona velocità di queste due euristiche, queste non presentano una buona qualità di soluzioni, ragion per cui si può passare ad altre euristiche.

Altra euristica costruttiva, di tipo greedy, molto famosa, è quella relativa all'**algoritmo di Clarke and Wright**.
Alla base di questa euristica vi è il concetto di *Saving*, ovvero un risparmio.
La soluzione ad ogni iterazione è fatta da un certo numero di *Tour*, o percorsi, e a partire da questa soluizione si cerca di unire più percorsi in uno solo. La soluzione corrente, ad ogni passo, è data da una famiglita $T = \{T_1, \cdots, T_q\}$ di cicli orientati, escluso il deposito. Ciò significa che ogni nodo apparterrà ad un ciclo, che il deposito apparterrà ad ogni ciclo e che la somma delle domande dei punti vendita associate ad un ciclo $T_h$ non supererà la capacità del veicolo $Q_h$.

Si parte perciò da un soluzione iniziale molto semplice in cui dato un deopsito, ed n punti vendita, sarà disponibile unveicolo per ogni cliente da servire. Ciò si traduce nel fatto che ogni veicolo serva un nodo per poi tornare al deposito. Si cerca quindi, a partire da questa configurazione, la coppia di tour che conviene unire.

Quindi dati due tour $T_h$ e $T_k$, composti dai nodi che costituiscono

il loro ciclo, si prova a **Fonderli**, ovvero a creare un nuovo ciclo T' eliminando due degli archi che prima chiudevano i due cicli, e inserendone uno nuovo che faccia da ponte tra i due cicli, e che ne definisca uno nuovo.
Tale fusione appena realizzata dovrà essere ammissibile e generare un risparmio. Quindi in primo luogo la domanda di questi cluster fusi deve poter esser gestita dalla capactà del singolo veicolo.

$$\sum_{v_j \in T' - \{v_1\}} d_j \leq Q$$

Successivamente possiamo chiederci se si generi un risparmio. Come già detto verranno rimossi 2 archi di percorrenza, quelli che chiudevano i due cicli precedenti, e ne sarà aggiunto uno che colleghi i due cicli in un unico ciclo. Se questo nuovo arco ha costo minore della somma degli archi eliminati allora è stato generato un risparmio.
Tale calcolo prende il nome di saving, che deve essere necessariamente maggiore di zero.

$$s_{hk} = c_{v_u(h)v_1} + c_{v_1 v_{p(k)}} - c_{v_{u(h)} v_{p(k)}} > 0$$

Il ragionamento può esser quindi reiterato e raccolto nel seguente macrocodice:

1. Si pone i O 0, $T^0 = \{T_1^0, \cdots, T_{n-1}^0\}$, con $T_{i^0 = \{v_1, v_{i+1}\}}$ per ogni i
2. si individua la coppia $\{T_h, T_k\}$ a cui corrisponde il massimo saving ammissibile
3. se il massimo saving è minore o uguale di 0 l'algoritmo termina
4. altrimenti si genera un nuovo ciclo T' dato dall'unione dei cicli $T_h$ e $T_k$ e si genera la nuova soluzione $T^{i+1} = TiU\{T'\} - \{T_h, T_k\}$
5. si incrementa i e si torna al passo 2

Un altro vincolo fondamentale è quello relativo alla cardinalità della flotta. Infatti in base a quanto detto fino ad ora non abbiamo tenuto conto della limitata cardinalità della flotta. Quindi quando l'algoritmo termina, il numero di tour individuati potrebbe eccedere il numero di veicoli necessari a disposizione. Perciò l'algoritmo deve terminare quando non si possono ottenere altri saving, ma se il numero di veicoli viene superato, si effettuano altre unioni che peggiorano la soluzione per poter rientrare nei vincoli di cardinalità.

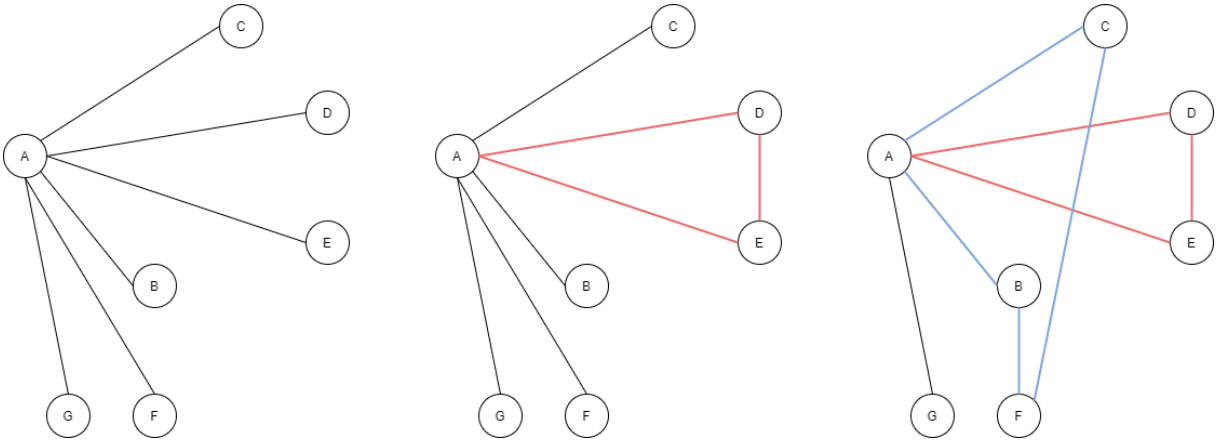



Figure 41: Euristiche Greedy per il VR

Proviamo a fare un esempio, ponendo nella seguente tabella il costo per spostarsi da un nodo ad un altro, considerando A il deposito, e usando come riferimento la figura.

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| A | 0 | 10 | 17 | 25 | 20 | 15 | 16 |
| B | 10 | 0 | 11 | 15 | 12 | 7 | 16 |
| C | 17 | 11 | 0 | 9 | 8 | 12 | 24 |
| D | 25 | 15 | 9 | 0 | 6 | 11 | 24 |
| E | 20 | 12 | 8 | 6 | 0 | 7 | 18 |
| F | 15 | 7 | 12 | 11 | 7 | 0 | 13 |
| G | 16 | 16 | 24 | 24 | 18 | 13 | 0 |

Inoltre possiamo considerare la richiesta di ogni nodo nella seguente tabella, mentre la capacità Q pari a 4.

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| d | - | 2 | 1 | 3 | 1 | 1 | 2 |

Nella prima iterazione viene selezionato il tour da AD e quello da AE e vengono fusi in un unico tour ADE. infatti il costo precedente era per AD = 50, e per AE 40. unendo i due cicli si riesce ad ottenere, per un costo totale di 20 + 25 + 6 = 51, un risparmio di 39, per una richiesta servita di 4, pari alla capacità massima Q.
Ragionando alla stessa maniera, afferendo alle due tabelle, si possono

determinare i tour ACFA prima, e ACFBA dopo.

è possibile inoltre modificare tale algoritmo tramite alcuni accorgimenti. In primo logo, data la soluzione corrente T, per ogni tour i si considera come primo nodo $v_{p(i)}$ il nodo più vicino al deposito e come ultimo nodo $v_{u(h)}$ il nodo più lontano dal deposito.
Al termine dell'algoritmo, quando non esistono più savings di valore positivo, ovvero al termine del clustering, per ogni cluster $C_i$ individuato viene generata una soluzione euristica del problema del TSP utilizzando l'euristica 2-opt.

### 5.4.4 Tabu Search in Vehicle Routing

Sia G un grafo orientato con V insieme dei nodi, e sia il nodo $v_1$ indicativo del deposito mentre ogni altro nodo dell'insieme rappresenta un punto vendita con una domanda $d_i$ maggiore di 0. Ogni arco rappresenta il collegamento esistente tra una coppia di nodi ed è caratterizzato da una distanza, o costo, non negativa $c_{ij}$.
Per effettuare le consegne viene messa a disposizione una flotta di m veicoli tutti con la stessa portata Q.

Partendo da queste basi, introdurremo l'algoritmo di **Hertz, Gendreau e Laporte** per applicare il Tabu Search sul Vehicle Routing.
Tale algoritmo fa uso del concetto di **Pre-soluzione**,ovvero una famiglia di sottoinsiemi $\mathscr{C} = \{C_1, \cdots, C_t\}$, con la proprietà che ogni punto vendita sia assegnato ad un solo sottoinsieme $C_i$. Una pre soluzione in cui la somma della domande dei punti vendita appartenenti a ciascun insieme $C_i$ è minore di Q è una *soluzione ammissibile*, altrimenti essa non sarà ammissibile, ma comunque pre-soluzione.

L'algoritmo HGL costruisce una sequenza di pre soluzioni $C_1, C_2 \cdots, C$, ciascuna nell'intorno della soluzione precedente.
Tale intorno viene costruito scegliendo un punto di vendita $v_i$ appartenente all'insieme $C_h$ e lo si inserisce in un nuovo cluster $C_k$. l'insieme dal quale viene prelvato il punto vendita, ovvero $C_h$,sarà uno qualsiasi dei cluster della pre-soluzione corrente e quindi, dato $C_k$ il cluster nel quale sarà inserito il nodo $v_i$, trattandosi di una tabu search, sarà necessario memorizzare degli attributi nella tabu list, per evitare di ritornare sui propri passi, indicando Il cluster appena privato del punto vendita come Tabu.

Infatti il reinserimento di $v_i$ in $C_h$ resterù tabu per le successive $\theta$ iterazioni, inserendo nella lista la coppia nodo, cluster.

Una mossa potrebbe quindi consistere nel prelevare un nodo dal cluster $C_h$ e assegnarlo ad un altro cluster, oppure cominciare, a partire dal nodo deposito e dal nodo vendita prelevato, la costruzione di un altro cluster, evitando così di lavorare sempre e soltanto su soluzioni che abbiano lo stesso numero di cluster.

Ogni volta che si ottiene una nuova presoluzione C generata dall'algoritmo ,essa viene ottimizzata applicando l'euristica 2 opt a ciascun insieme $C_i \in \mathscr{C}$. Poiché stiamo parlando di un problema rilassato, e quindi pre-soluzioni e non soluzioni, a ciascuna pre soluzione sono assegnate 2 funzioni obiettivo. La prima,è quella classica del vehicle routing, relativa alla somma su tutti i cluster della lunghezza del circuito hamiltoniano definito sul cluster j-esimo.

$$f_1 = \sum_{j=1}^{q} L(C_j)$$

L'algoritmo lavora però solo su pre soluzioni, e considera un problema rilassato in cui non ci si preoccupa sul vincolo di capacità.
La seconda funzione infatti terrà in considerazione il caso in cui la pre-soluzione corrente *non sia* una soluzione ammissibile.
Tale funzione sarà data dalla somma della f obiettivo originaria più un termine rappresentante la penalizzazione da considerare ogni qual volta che la pre-soluzione violii un vincolo di capacità.
Perciò, data $\alpha$ la costante di penalizzazione, per ogni cluster si considera il valore massimo tra zero e la differenza tra la somma delle domande dei nodi appartenenti al cluster j-esimo e la capactià del veicolo che serve quel cluster.
E allora se per il cluster il vincolo è soddisfatto, la differenza è negativa, e viene preso lo zero, altrimenti è preso il termine positivo. Questa penalità accresce il valore della f obiettivo e dipende da quanti vincoli vengono violati e da quanto essi vengano violati.

$$f_2 = f_1 + \alpha \sum_{j=1}^{q} \max \left\{ 0, \left( \sum_{v_i \in C_j} d_i \right) - Q \right\}$$

è allora chiaro che una pre soluzione è ammissibile se $f_1 = f_2$.

Per poter ispezionare un intorno si preleva quindi un nodo da un generico cluster e si valuta se sia possibile inserirlo in un altro cluster oppure crearne uno nuovo.
Se si applicasse questa mossa per generare l'intero intorno della pre-soluzione corrente, l'intorno sarebbe fatto da numerosissime soluzioni, con una cardinalità di n punti vendita X m veicoli.
Applicando poi la mossa di 2 opt, tale cardinalità non farebbe altro che aumentare, diventando ingestibile.
La soluzione a ciò consisterà allora nel non ispezionare completamente l'intorno, ma selezionare solo alcune soluzioni dell'intorno, effettuando un numero limitato di tentativi di mossa.

In particolare una volta che viene scelto un nodo $v_i$, estraendolo da $C_h$, non si vanno a verificare tutte le possibili pre-soluzioni in cui $v_i$ è inserito in ogni cluster, ma bensì solo nei cluster più vicini, ovvero che contengono nodi vicini al nodo $v_i$.

Bisogna poi stabilire in che posizione metter il nodo nel percorso del veicolo. Per non fare tutte le euristiche, $v_i$ viene inserito immediatamente prima e immediatamente dopo al nodo ad esso più vicino posto in $C_k$.
Ciò si traduce nel mantenere in $C_k$ lo stesso circuito hamiltoniano della soluzione corrente, a meno di una piccola modifica.

Quindi mel selezionare le soluzioni dell'intorno viene valutata soltanto la funzione $f_2$ senza applicare l'euristica 2-opt, così per ogni nodo scelto $v_i$ vengono eseguiti q tentativi di mossa.
Una volta stabilita la nuova soluzione diversa da quella corrente, poichè essa sarà stata migliorata, la coppia nodo-cluster diventa tabu. Se $f_2$ è il minore del minimo corrente, la mossa viene presa in considerazione altrimenti ignorata.

Nel corso dell'algoritmo la costante di penalizzazione $\alpha$ deve inoltre essere aggiornata.
Inizialmente essa è posta pari ad 1 e se per 10 iterazioni consecutive sono state prodotte soltanto soluzioni inamissibili, il valore di $\alpha$ viene raddoppiato, rendendo la violazione dei vincoli più pesante.
Se invece in 10 iterazioni di fila si hanno solo soluzioni ammissibili, si

sta limitando l'ambito della ricerca, e allora $\alpha$ viene dimezzato, così da rendere più stringente il vincolo di violazione di capactià.

L'algoritmo termina se si hanno più di $50 \times n$ iterazioni senza aggiornamenti del migliore valore di $f_2$, oppure se $\alpha$ diventa maggiore di $10^{30}$.

Riassumendo, l'algoritmo dovrà effettuare un passo di inizializzazione del numero minimo di veicolo da utilizzare, pari alla somma di tutte le domande divise perla capacità del veicolo, approssimato in eccesso. Inoltre andranno stabiliti p, ovvero il numero di nodi vicini al nodo corrente da considerare e che definiranno l'intorno, e q il numero di tentativi di mossa.

$$m = \left\lceil \frac{\sum_{i=2}^{n} d_i}{Q} \right\rceil \qquad p = \min(n, 50) \qquad q = \min(n, 5m) \qquad \alpha = 1$$

L'algoritmo procede calcolando una soluzione iniziale $C = \{C_1, C_2 \cdots C_t\}$ utilizzando l'algoritmo di clarke and wright modificato, così da ottenere una soluzione di partenza, e inizializzare $f_1*$ ed $f_2*$ al valore della soluzione prodotta dall'algoritmo di clarke e wright modificato.

L'algoritmo procede, iterazione per iterazione, valutando se $\alpha < 10^30$ e se $f_2$ sia stata aggiornata da meno di $50 \times n$ iterazioni (condizioni di terminazioni).
Se tali condizioni non si verificano si pone t=t+1; se le ultime iterazioni sono state tutte ammissibili allora si pone $\alpha = \alpha/2$, se sono tutte inammissibili allora $\alpha = 2\alpha$.
Si selezionano q nodi in modo casuale e si inserisce ogni nodo in un insieme diverso da quello attuale, contentente almeno uno dei p nodi più vicini e che non sia tabù, e si calcola il valore di $f_2$.
Si esegue la mossa in corrispondenza della quale si è ottenito il più piccolo valore di $f_2$; se $f_2 < f_2*$ si aggiorna la soluzione corrente $f_2*$.
Se la soluzione calcolata è ammissibile, e $f_1 < f_1*$, si aggiorna anche $f_1*$.
Se la mossa eseguita ha spostato $v_i$ da $C_h$ in un altro insieme, la coppia nodo-cluster è ora dichiarata tabu per le prossime $\theta$ iterazioni

# 6 Problemi di Localizzazione

Parleremo adesso di un altro classico problema di PL, ovvero il **Problema di Localizzazione**.
Introduciamo quindi il concetto mediante un esempio: Consideriamo dei centri di servizio da localizzare allo scopo di poter rispondere ad una domanda distribuita sul territorio. Ipotizziamo di avere una amministrazione cittadina che debba costruire centri di pronto soccorso per servire i quartieri della città. Sono stati individuati tre possibili siti in cui poter localizzare i vari pronto soccorso, e la città è stata sezionata in 4 quartieri differenti.

| | quart.1 | quart.2 | quart.3 | quart.4 |
|---|---|---|---|---|
| Sito 1 | 7 | 6 | 7 | 8 |
| Sito 2 | 10 | 10 | 1 | 1 |
| Sito 3 | 9 | 5 | 4 | 1 |

È necessario selezionare due tra questi 3 siti in cui costruire i due centri di pronto soccorso, e si vuole decidere dove costruirli, ovvero quali attivare al fine di non sfavorire troppo nessuno dei potenziali utenti, e stabilire a quale sito dovrà afferire un certo quartiere.
Come prima cosa andranno stabilite le variabili decisionali, relative per ogni quartiere a quale sito esso afferirà.
Allora saranno $x_{ij} = 1$ se il quartiere j afferisce al sito i, altrimenti zero. Possiamo inoltre introdurre le variabili $y_i$ pari 1 se il sito i è stato aperto (**Nemici dell'erede temete**), zero altrimenti, mentre $d_j$ rappresenterà la distanza del quartiere j dal sito aperto al quale esso afferisce.

Per capire meglio, $d_1$ vale 7 se il quartiere 1 afferisce al sito 1, 10 se afferisce al sito sito 2, 3 altrimenti. E allora possiamo definire la funzione obiettivo come il minimo del massimo tra i $d_j$, per poi stabilire i vincoli.

$$\min(\max_{j=1\cdots4} d_j)$$

I primi vincoli da stabilire sono relativi alla definizione delle distanze $d_j$, che avranno un determinato valore in base a quale sito essi afferiranno:

$$d_1 = 7x_{11} + 10x_{21} + 9x_{31}$$

$$d_2 = 6x_{12} + 10x_{22} + 5x_{32}$$

$$d_3 = 7x_{13} + x_{23} + 4x_{33}$$

$$d_4 = 8x_{14} + x_{24} + x_{34}$$

Successivamente bisogna imporre i vincoli relativi alle x e alle y. Potendo avere solo 2 siti attivi , è chiaro che la somma delle trevariabili y, relativi all'apertura del sito, sarà pari a 2 . Bisognerà poi stabilire i vincoli relativi alla mutua esclusività tra le afferenze ai quartieri.

$$y_1 + y_2 + y_3 = 2$$

$$\sum_{i=1}^{3} x_{ij} = 1 \qquad j = 1 \cdots 4$$

Infine è necessario stabilire i vincoli che leghino le variabili x a quelle y. Se un quartiere afferisce ad un sito specifico, tale sito dovrà esistere necessariamente, altrimenti, se il sito non è stato costruito, è ovvio che un quartiere non vi si possa riferire. Questi vincoli possono esser sintetizzati come:

$$x_{ij} \leq y_i \qquad i = 1 \cdots 3, j = 1 \cdots 4$$

Questo modello di programmazione lineare intera, che però non è propriamente lineare in quanto è un problema di min max.
Perciò invece di utilizzare le variabile $d_j$ utilizziamo un'unica variabile d che rappresenti la distanza percorsa dagli utenti del quartiere più favorevole e stabiliamo di voler minimizzare questa quantità d, ottenendo la banale funzione obiettivo:

$$\min d$$

Naturalmente in questo caso il valore dei $d_j$ sarà allora maggiore o uguale della distanza percorsa da ciascuno dei quartieri, e non più semplicemente uguale. La minimizzazione infatti porterà alla massima delle distanze.

Altro esempio che possiamo introdurre è il seguente: Una azienda produttrice di automobili ha a disposizione tre nuovi stabilimenti S1,S2 ed S3, il cui costo di attivazione è pari a 9000, 7000 e 8000 euro, rispettivamente. Si deve decidere quali stabilimenti attivare, con l'obiettivo di soddisfare la domanda annuale di 4 punti vendita P1,P2,P3,P4 pari a 150,400,200 e 300 automobili, rispettivamente.

Nella seguente tabella sono riportati i costi unitari di trasporto, espressi in euro, dagli stabilimenti ai punti vendita:

| | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| S1 | 20 | 40 | 10 | 30 |
| S2 | 30 | 60 | 60 | 40 |
| S3 | 40 | 50 | 50 | 70 |

si vuole minimizzare i costi complessivi di attivazione degli stabilimenti e di trasporto delle automobili nei punti vendita, tenendo conto che la capacità produttiva annuale dei tre stabilimenti è pari a 700,900,800 automobili, rispettivamente.

## 6.1 Formalizzazione del problema

I modelli di localizzazione ed i relativi algoritmi di soluzione, sono uno dei principali strumenti quantitativi per la *Pianificazione territoriale di reti di servizio*.
L'obiettivo di un problema di localizzazione è di definire le localizzazioni di **Centri di servizio**, come impianti di produzione, che devono soddisfare una **domanda** dispersa sul territorio da negozi o singoli centri.

La localizzazione può essere **puntuale**, ovvero si sta ricercando un punto nel grafo, o nella regione, in cui andare a localizzare uno o più centri di servizio, oppure le attività da localizzare possono avere una **estensione**, ovvero, una volta rappresentato il sistema mediante un grafo, non bisogna localizzare nei nodi del grafo, ma bensì sugli archi, visti come se avessero una propria estensione. Parliamo allora di Localizzazione **non puntuale**, che sono problemi soliti in progetti di reti.

Altre caratteristiche di questi problemi riguardano la possibilità di discretizzare o meno i dati di input. Si possono avere infatti modelli **discreti** nel caso in cui la scelta della localizzazione venga fatta all'interno di un insieme finito di potenziali localizzazioni, solitamente i grafi.
Un problema è al contrario **continuo** se è possibile localizzare in qualsiasi punto della regione il punto di servizi specifico.

Altro aspetto fondamentale è il *calcolo della distanza* tra clienti e centri di servizio. La cosa più naturale potrebbe consistere nell'utilizzare

la distanza euclidea, anche se sarebbe improprio stabilire la distanza euclidea come distanza solitamente utilizzata. Si applica infatti una generalizzazione di questa, del tipo:

$$d(P_i, P_j) = \left( (x_i - x_j)^k + (y_i - y_j)^k \right)^{\frac{1}{k}}$$

Per k = 1 si parla di metrica lineare, o di manhattan, che descrive efficacemente il caso di spostamenti su direzioni ortogonali.
Per k = 2 si ottiene invece quella che viene convenzionalmente chiamata distanza euclidea
Sperimentalmente si è verificato che su realtà territoriali urbane ed extraurbane la metrica più efficace è caratterizzata da un valore k compreso tra 1 e 2.

Occorre infine definire i **costi** da considerare nella funzione obiettivoTali costi possono essere legati all'effettivo costo di localizzazione da effettuare e possono dipendere dal punto in cui si decide di localizzare il centro e da quanto esso sia capace/grande. parliamo in questo caso di **Costi di localizzazione** o fissi.

Sono poi presenti dei **Costi variabili**, o di afferenza, che sono i costi di accesso al servizio da parte degli utenti finali.

Per la funzione obiettivo possono esserci svariate scelte da metter in campo.
Si può infatti o minimizzare i costi fissi di localizzazione, o quelli di afferenza o una combinazione dei due.Si potrebbe inoltre voler massimizzare la domanda totale coperta dal servizio, oppure ancora minimizzare il massimo costo di afferenza (min max).

Quindi gli obiettivi possono esser considerati contestualmente, minimizzando una combinazione lineare degli stessi, oppure scegliendo un obiettivo e imponendo dei vincoli sui valori ammissibili degli altri. Si potrebbe anche realizzare, volendo, una analisi multiobiettivo, cercando le soluzioni non dominate sul fronte di Pareto.

A questo punto possiamo concludere la modellazione stabilendo i vincoli del problema, oltre a quelli relativi al soddisfacimento della domanda.
Tali vincoli aggiuntivi possono essere relativi alla copertura del servizio,

ai costi da sostenere, alla capacità dei centri, alla struttura di rete e via dicendo.

## 6.2 Simple Plant Location

il problema di simple plant location è definito su di un grafo, quindi relativo ad un modello di localizzazione discreta. Dato il classico grafo G(V,A) sia I⊂V l'insieme dei nodi clienti che richiedono un dato servizio e sia $J \subset V$ l'insieme dei nodi in cui può essere localizzato un centro di servizio. (I e J potrebbero coincidere tra loro e coincidere anche con l'intero insieme V)
Il problema di simple plant location richiede di localizzare i centri di servizio in modo da minimizzare i costi totali, ovvero i costi fissi più quelli di afferenza.
Quindi non vi sono vincoli sul numero degli impianti da localizzare, che saranno invece un valore di output al problema. Tra i dati del problema vi sono inoltre i costi di afferenza in j della domanda generata da i ($c_{ij}$)e il costo di apertura del servizio in j, che simboleggeremo con $f_j$.

Le variabili decisionali invece sono relative all'afferenza di un cliente i ad un centro di servizio j, $x_{ij}$, ovviamente quelle relative ai vincoli di localizzazione del centro di servizio j $y_j$. Entrambe le tipologie di variabili decisionali saranno binarie in quanto segnaleranno se un cliente afferisca ad un centro o meno, e se un determinato centro sia stato localizzato o meno.
Possiamo di seguito definire, analiticamente, la struttura della funzione obiettivo:

$$\min \sum_{i \in I, j \in J} c_{ij} x_{ij} + \sum_{j \in J} f_i y_i$$

E successivamente i vincoli:

$$\sum_{j \in J} x_{ij} = 1 \qquad \forall i \in I$$

$$x_{ij} \leq y_i \qquad \forall i \in I, j \in J$$

$$x_{ij} \geq 0 \qquad \forall i \in I, j \in J$$

$$y_j \in \{0, 1\} \qquad \forall j \in J$$

Il primo vincolo è chiaramente relativo al fatto che ogni domanda dei cliente debba esser soddisfatta, e quindi non potrà esistere un particolare cliente a non esser servito.
Gli altri vincoli sono invece relativi alla definizione di y come binaria e x come intera. Il fatto che x sia binaria viene automaticamente assicurato dalla relazione di disuguaglianza che lega x ed y tra loro. è infatti naturale che se il centro j, rappresentato dalla variabile y, non fosse stato localizzato, nessuno dei clienti potrebbe afferire a j, in quanto avremmo la relazione $x_{ij} \leq 0$ insieme a $x_{ij} \geq 0$

data questa modellazione è allora chiaro che è possibile risolvere problemi di SPL direttamente all'ottimo usando una modellazione esatta, qualora il problema rimanesse di dimensioni contenute.Tuttavia può talvolta capitare di non disporre di solutori, o di avere problemi di grandi dimensioni. Può quindi esser necessario applicare una euristica per ottenere una soluzione in maniera più rapida, seppur meno precisa.

Data una soluzione ammissibile con alcuni centri di servizio aperti, l'apertura di un nuovo centro di servizio può migliorare la funzione obiettivo solo se la riduzione dei costi di afferenza supera il costo di apertura del nuovo centro.

Una soluzione ammissibile consiste perciò nello scegliere un impianto appartenente a J ed aprirlo.
Al passo iniziale l'algoritmo sceglie di aprire il centro di servizio cui corrisponde il minimo valore della somma tra il costo di apertura e i costi di afferenza al centro di servizio.
Alla generica iterazione, se S è l'insieme dei centri aperti, viene aperto il centro di servizio k (tra quelli non ancora aperti) cui corrisponde il massimo valore del saving, dato da:

$$s_k = \sum_{i \in I} \max(\min_{j \in S} c_{ij} - ck, 0) - f_k$$

L'algoritmo si arresta quando non vengono più identificati saving positivi.
Se oltre alla mossa di aggiunta di un impianto si considera uan mossa di sostituzione di impianto si ottiene un vero e proprio algoritmo di ricerca locale, e il termine euristica migliorativa assume un significato più corretto.

## 6.3 Capacitated Plant Location

Nel problema di **Capacitated Plant Location** i nodi clienti sono caratterizzati da un valore di domanda $d_i$ e i potenziali centri di servizio da un valore di capacità $m_j$. una volta aperto, un centro di servizio, può servire soltanto insiemi di clienti la cui domanda totale non supera la sua capacità.

Si stabiliscono in aggiunta $c_{ij}$ come il costo di afferenza in j della domanda generata da i, e $f_j$ il costo di apertura del servizio j.

Le variabili decisionali del problema saranno nuovamente le variabili di localizzazione $y_j$, come nel caso del SPL, e poi le variabili $s_{ij}$, indicanti questa volta la quantità di flusso che arriva al cliente i proveniente dal centro j.
La modellazione del problema non si discosta eccessivamente dalla precedente, presentando la medesima funzione obiettivo

$$\min \sum_{i \in I, j \in J} c_{ij} x_{ij} + \sum_{j \in J} f_i y_i$$

ma vincoli leggermente differenti. Deve infatti accadere nuovamente che ogni domanda di un cliente vada soddisfatta, ma in questo caso la domanda dei clienti rappresenterà una certa quantità di flusso da soddisfare, per cui

$$\sum_{i \in I} x_{ij} = d_i \qquad \forall i \in I$$

è inoltre necessario stabilire dei vincoli di capacità, per i quali

$$\sum_{i \in I} x_{ij} \leq m_j y_j \qquad \forall j \in J$$

infine è possibile stabilire, come prima, vincoli sulla interezza della variabile x e della variabile y, che sarà invece binaria.

## 6.4 P-mediana e P-centro

Se i costi di localizzazione sono uguali per ogni potenziale centro di servizio e si fissa il numero p di localizzazioni da determinare, in funzione obiettivo si possono trascurare i costi di localizzazione.
Un problema di **p-mediana** consiste quindi nella individuazione di p

nodi nei quali localizzare i centri di servizio allo scopo di minimizzare la somma dei costi di afferenza.
Nuovamente stabiliamo cone $c_{ij}$ il costo di afferenza in j della domanda generata da i, mentre prenderemo come variabili decisionali gli $y_j$ pari ad 1 se in j è localizzato un centro di servizio e 0 altrimenti, e le variabili $x_{ij}$ pari ad 1 se il cliente i afferisce al centro di servizio j, 0 altrimenti.

Potendo trascurare i costi di localizzazione, la funzione obiettivo diventa semplicemente:

$$\min \sum_{i \in I, j \in J} c_{ij} x_{ij}$$

I vincoli seguenti saranno del tutto simili a quelli visti nel problema del SPL, relativi al soddisfacimento delle domande, agli upper bound dati dalle variabili y e dalla interezza delle variabili decisionali. A differenza del problema dell'SPL è necessario però considerare un nuovo vincolo, relativo al numero di centri da aprire pari a p.

$$\sum_{j \in J} y_j = p$$

Per risolvere problemi di p-mediana si usa solitamente l'algoritmo di **Teits e Bart**, ovvero un algoritmo migliorativo basato sul concetto di saving dovuto alla sostituzione di un centro aperto con un altro.
Sia S l'insieme dei p centri aperti ed S' l'insieme dei centri non aperti, allora l'algoritmo prevederà, al passo iniziale, di scegliere, spesso a caso, p centri di servizio da aprire.
Alla generica iterazione, per ogni coppia $i \in S$ e $j \in S'$ viene calcolato il saving come:

$$s_{ij} = z(s) - z(S - \{i\} \cup \{j\})$$

Viene quindi effettuata la soluzione cui corrisponde il saving minore; se tutti i saving sono non negativi l'algoritmo si arresta.
Notiamo inoltre che se un nodo $j \in S'$ non genera alcun saving negativo, anche nelle succesive iterazioni potrà generare saving e quindi potrà non esser considerato.

Ultima variante che vale la penacitare è quella relativa al problema di $P - centro$. Nel caso in cui i centri di servizio siano centri emergenziali, l'obiettivo diventa quello di localizzare p centri allo scopo di

sfavorire il meno possibile gli utenti più svantaggiati. Tale problema si può ricondurre ad un problema di min max, in cui l'obiettivo è minimizzare una certa funzione z.

z è il valore della distanza massima tra il cliente e il centro aperto, ragion per cui si può definire la funzione obiettivo come:

$$\min z \qquad z \geq \sum_{i \in I, j \in J} c_{ij} x_{ij}$$

i vincoli da porre successivamente sono gli stessi vincoli già visti nel problema di p-mediana, relativi ai centri di aprire e agli upper bound e variabili intere:

$$\sum_{j \in J} y_j = p$$

$$\sum_{j \in J} x_{ij} = 1 \qquad \forall i \in I$$

$$x_{ij} \leq y_i \qquad \forall i \in I, j \in J$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, j \in J$$

$$y_j \in \{0, 1\} \qquad \forall j \in J$$