Dispensa del corso di Decision Models

[0]P. C. Valenti, P. Lindia, CDLM Data Science, Università di Milano-Bicocca

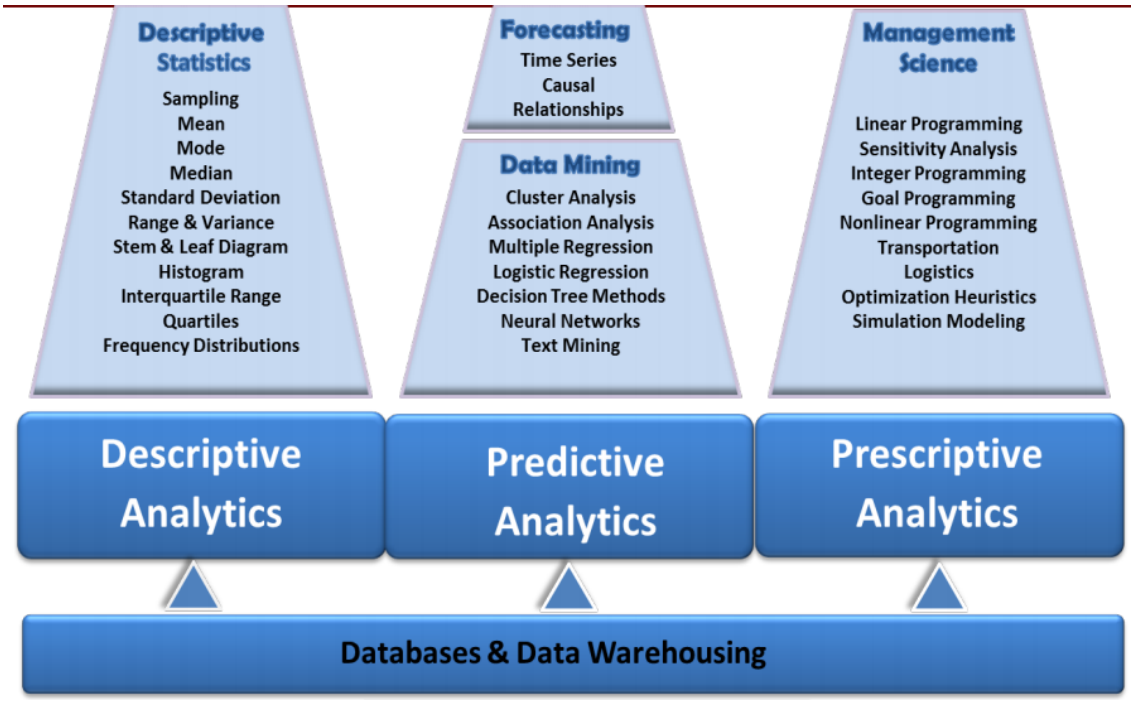
Indice

Introduzione

1.1 *Analytics*

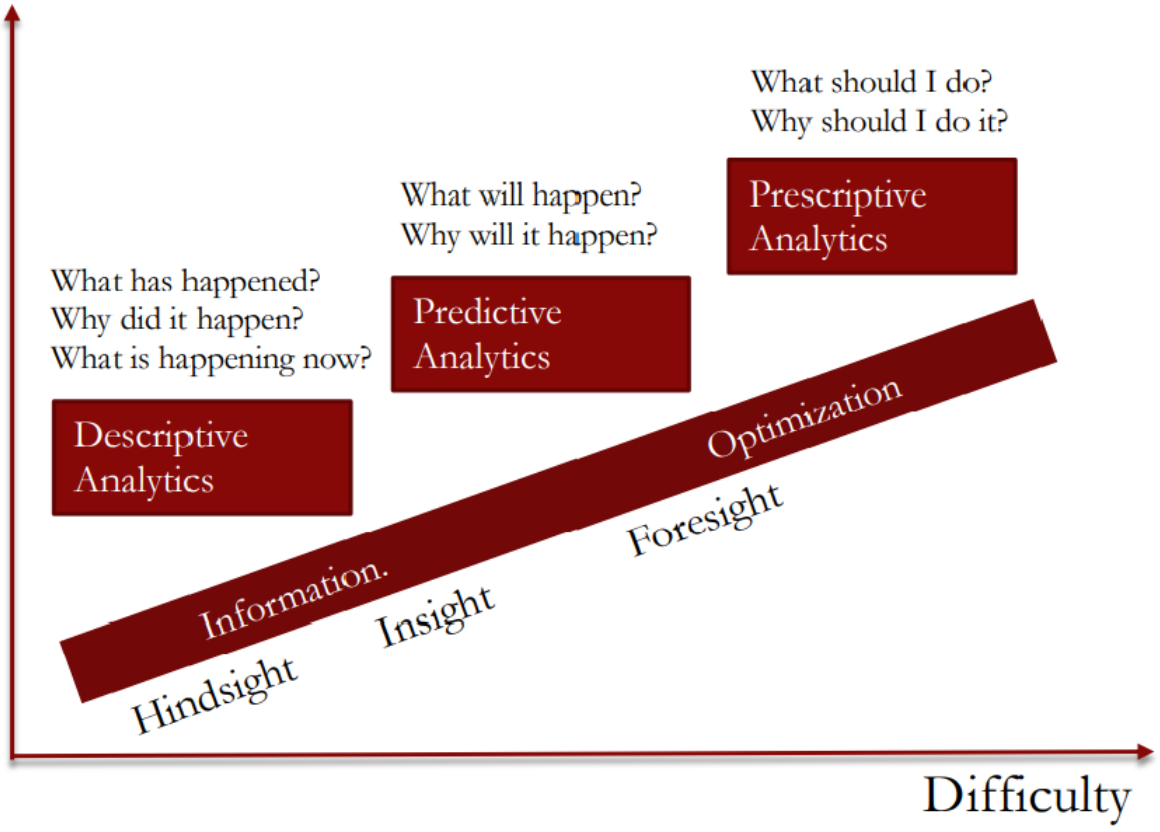
Con il termine **Analytics** si intende il processo di scoperta, interpretazione e comunicazione di *patterns* significativi all'interno di un set di dati; tipicamente viene applicata in campo statistico, *computer science* e ricerca operativa, mentre come tecniche alla base vi sono metodi computazionali e matematico-statistici. Lo scopo dell'*analytics* è quindi quello di studiare dati rappresentativi del passato, al fine di studiarne le tendenze e gli effetti, così da garantire previsioni e nuova conoscenza del fenomeno in questione.

Quando si parla di *analytics* si fa sempre riferimento al concetto di modello, ovvero l'utilizzo di un complesso (o semplice) insieme funzioni statistico-matematiche, il quale ha lo scopo di aiutare il *decision maker* a prendere le migliori decisioni possibili. Anche se negli ultimi anni si sta sempre più diffondendo il concetto di **Big Data**, a seconda del fenomeno in esame possono risultare ottimali set di dati più "piccoli" e settoriali (**Small Data**).



L'*Analytics*, come appena descritto, è il procedimento che si torva in mezzo tra i dati e la nuova conoscenza generata; poichè il corpo centrale della fase operativa risulta la scelta del modello, strettamente collegata al fenomeno in esame e al set di dati, la finalità dello studio andrà ad identificare diverse branche dell'*analytics*. Nell'immagine precedente vengono prese in considerazione le 3 principali:

- **Descriptive Analytics**: vengono impiegati i dati del passato per lo studio di un fenomeno passato. I principali strumenti dell'analisi descrittiva sono le statistiche e lo studio delle distribuzioni (statistiche e grafici).
- **Predictive Analytics**: vengono impiegati i dati del passato per lo studio del futuro di un fenomeno (previsioni). L'assunzione di base è che vi sia un collegamento tra ciò che è accaduto e ciò che deve accadere; si serve di modelli statistici (regressioni) e di *machine learning* (alberi decisionali, reti neurali...).
- **Prescriptive Analytics**: vengono impiegati metodi di ottimizzazione, di simulazione e metodi euristici al fine di specificare le soluzioni/decisioni ottimali al fenomeno in analisi.



Il grafico esemplificativo mostra le principali domande e la difficoltà delle varie tipologie di *analytics*; solitamente in uno studio approfondito tali fasi sono consequenziali.

1.1.1 *Descriptive analytics*

Lo scopo dell'analisi descrittiva è quello di descrivere le principali variabili (*features*) che riguardano i dati disponibili (riferiti al fenomeno in esame). A tale fine vengono impiegate diverse tecniche, tipicamente afferenti alla statistica descrittiva, le quali possono essere sia metodologiche che grafiche; le principali risultano: campionamento, studio delle statistiche descrittive (media, moda, mediana, deviazione, ...), delle distribuzioni (frequenze, quantili, asimmettrie, ...) e grafici (*box plot*, istogrammi, ...).

I risultati sono solitamente rappresentati da tabelle statistiche e/o grafici rappresentativi.

Un esempio di analisi descrittiva può essere fornito dalla seguente *business task*:

- Fandango, azienda leader nella vendita di biglietti cinematografici, vuole sapere i film preferiti dagli spettatori durante lo scorso anno, utilizzando la base dei dati che riguarda i milioni di biglietti venduti. A tale scopo possono essere impiegate diverse tecniche descrittive, a partire dal campionamento (utile a ridurre la numerosità), studio di genere (donna/uomo), i film più popolari (incassi oltre una certa soglia della distribuzione) e molte altre a seconda dell'interesse dell'analisi. Tale tipologia di analisi può essere utile a Fandango per definire il prezzo dei biglietti, le promozioni, gli orari, i cinema migliori dove proiettare lo spettacolo.

1.1.2 *Predictive analytics*

Tipicamente si serve inizialmente dei *patterns* individuati in fase di analisi descrittiva, estratti da set di dati molto "grandi" (*features selection* e *dimensionality reduction*). Successivamente è necessario determinare alcune correlazioni, le quali permettono di legare il fenomeno nei periodi passati (e nel presente); infine, mediante l'impiego delle correlazioni più significative, si possono generare stime e previsioni sui possibili (e probabili) valori che assumerà il fenomeno in analisi. Riprendendo l'esempio di Fandango potrebbe voler determinare quanto incasserà un determinato film in un determinato periodo (conoscendo lo storico di film simili per genere, attori, ...).

1.1.3 *Prescriptive analytics*

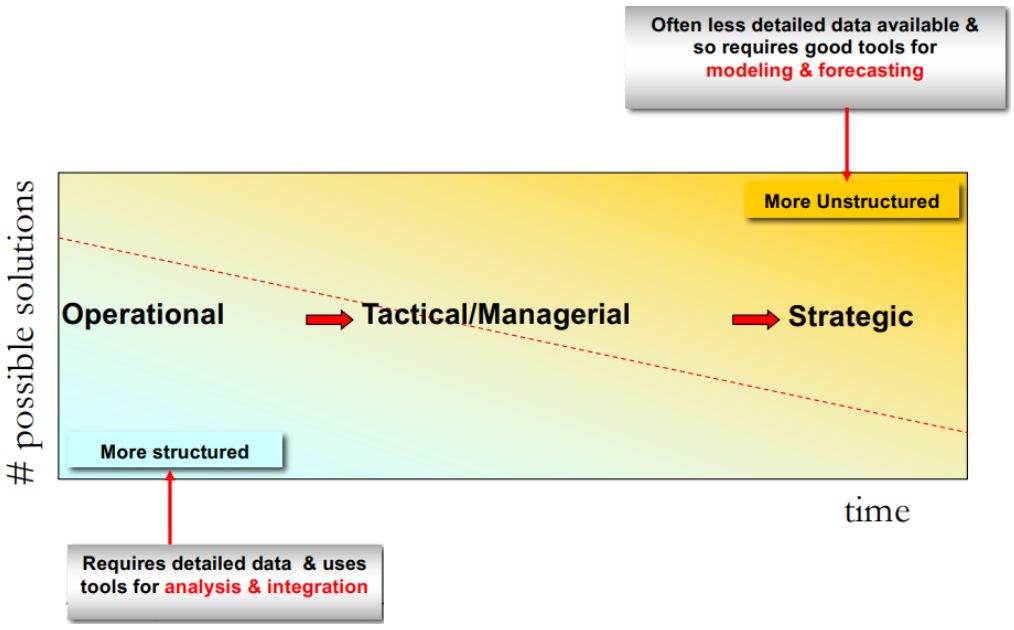
Tale fase si costruisce in generale sulla base dei risultati dell'analisi predittiva, ovvero si propone di studiare il miglior sistema decisionale, a partire dalle previsioni sull'evoluzione del fenomeno, che possa portare al risultato desiderato. Il procedimento di base è quindi un'ottimizzazione locale o globale basata sull'analisi **What-If**: riprendendo l'esempio di Fandango, l'azienda potrebbe, grazie all'ausilio di un analisi prescrittiva, riuscire a stabilire il prezzo ottimale dei biglietti, nei vari giorni e fasce orarie in cui vengono proiettati gli spettacoli, con il *target* della massimizzazione del profitto. In questo esempio risulta evidente come la *prescriptive analytics* possa determinare, attraverso l'impiego di un sufficiente numero di dati e analisi predittive

accurate, il raggiungimento dell'ottimizzazione dei risultati desiderati, guidando le decisioni degli addetti.

Una caratteristica che rende ancora più efficiente l'analisi prescrittiva è la possibilità di un aggiornamento dei modelli in **Real-Time**, ovvero ottenendo i dati utili in maniera continua (oraria, giornaliera, ...) in modo da strutturare un processo di ottimizzazione che si aggiorni in base alle nuove informazioni. Un caso tipico di tale condizione viene fornito dall'Industria 4.0, per esempio quando si ha a che fare con dati provenienti dai sensori degli impianti; tali dati vengono solitamente generati di continuo (o comunque in tempi molto brevi), fornendo così una moltitudine di informazioni aggiornate costantemente. Un altro caso, simile per caratteristiche, è quello dei sensori per il monitoraggio del traffico. Nel primo caso l'analisi prescrittiva potrebbe essere utile per prendere decisioni riguardo la *maintenance* dei sensori (quali si romperanno, come agire etc), mentre nel secondo può suggerire agli agenti del traffico dove vi saranno flussi consistenti, e come agire a riguardo.

1.2 *Decision Making* e *Decision Models*

Con **Decision Making** si intende il processo mediante il quale un addetto (*decision maker*) prende una decisione ottimale; ciò avviene vagliando tutte le possibili alternative, per le quali vengono considerati e pesati sia gli aspetti positivi, sia gli aspetti negativi derivanti dalla decisione in esame. E' chiaro che tale procedimento richieda di avere a disposizione sufficienti dati e informazioni sul fenomeno in esame, e soprattutto che il *decision maker* sia in grado di predire i diversi risultati che possono derivare da ciascuna decisione (esperto di *predictive analysis*); tali peculiarità rendono la *prescriptive analysis* estremamente complicata. Infine è anche necessario che chi prende le decisioni sia in grado di valutare quanto un possibile effetto possa risultare negativo (o positivo), e che quindi il *decision maker* sia un esperto di dominio del fenomeno in esame.



I problemi decisionali possono essere di diverso tipo, per esempio strutturati o non strutturati (immagine precedente); chiaramente le situazioni più delicate riguardano i problemi non strutturati.

- **Problemi non strutturati**:
  - Sono problemi decisionali non ordinari, e per i quali si hanno a disposizione informazioni da più fonti, parziali o ambigue.
  - Richiedono soluzioni *custom-made*, le quali non seguono procedure predefinite e precedentemente verificate.
  - Sono quasi sempre problemi non programmati, i quali richiedono quindi decisioni uniche, le quali generano risposte uniche.
  - Hanno un elevato livello di incertezza
- **Problemi strutturati**:
  - Hanno obiettivi chiari e predetermiati.
  - Sono ordinari, quindi hanno a disposizione procedure e modelli precedentemente testati.
  - Hanno a disposizione dati e informazioni ben strutturati (completi, non ambigui, ...).
  - Sono problemi programmati (di *routine*).

In secondo luogo possiamo valutare i problemi di *decision making* in base al loro livello di incertezza (**Uncertainty**), il quale può derivare sia dalla massiccia presenza di possibili risultati, sia dalla conoscenza parziale degli stessi; un elevato grado di incertezza si riflette sulla valutazione dei possibili risultati conseguenti ad una decisione. Solitamente in tale condizione si ha necessità di una distribuzione di probabilità (stimata) riferita ai possibili risultati; in questo modo è possibile stimare la verosimiglianza (*likelihood*). I concetti di *complexity* e di *risk* del problema decisionale risultano strettamente legati all'incertezza.

La definizione di **Decision Model** passa per l'assunzione che il modello in analisi fornisca una rappresentazione semplificata e simbolica della realtà del fenomeno in esame; ne esistono di diverso tipo (mentali, grafici, fisici, ...), ma per il corso vengono considerati principalmente i modelli matematici. Dal momento che lo scopo primo di un modello decisionale è la selezione della strategia ottimale, la quale passa per la previsione dei possibili risultati, affinché sia efficiente deve mettere in risalto le relazioni tra le informazioni in entrata e le previsioni in uscita. L'utilità nell'applicare un modello decisionale a sostegno del *decision making* risiede nella possibilità di elevare la propria conoscenza del fenomeno, nel miglioramento del processo decisionale (riduzione bias), nella possibilità di dimostrare e spiegare con metodo scientifico la propria strategia e infine nella definizione di un sistema riproducibile/riutilizzabile per decisioni future (il quale può essere aggiornato).

L'**Ottimizzazione** è il cuore pulsante di un processo di *decision making*, in ogni campo in cui esso possa essere applicato (*business*, sociale, personale, ...). La definizione matematica di ottimizzazione richiama la minimizzazione/massimizzazione di

un determinata funzione, la quale risulta legata ad un set di possibili dati, soluzioni e vincoli; tale funzione permette la comparazione di diverse scelte, e di conseguenza la selezione di quella ottimale.

L'immagine sottostante mostra in generale un buon processo di analisi dei dati, il quale passa per analisi descrittiva, predittiva e prescrittiva, e porta a prendere una decisione ottimale.

Per la strutturazione di un modello decisionale efficiente spesso non basta la sola capacità tecnica nella costruzione matematica dello stesso: esistono infatti numerosi aspetti dei fenomeni che non possono essere catturati dal modello, e che quindi necessitano di intuizione e giudizio del *decision maker*, tuttavia non sempre razionali e che quindi non devono prevaricare sulle risposte del modello. Le principali criticità nella decisione discrezionale degli esseri umani derivano da due effetti:

- **Anchoring Effect**→ deriva dall'influenza che pregiudizi legati alla realtà locale del *decision maker*, la quale genera una distorsione dovuta al pensiero di partenza riguardo un problema. Il *decision maker* ha quindi il compito di mitigare tale effetto, tenendone conto e aggiustando le decisioni.
- **Framing Effect** → deriva dalla visione che ha il *decision maker* riguardo le diverse opzioni decisionali, e quindi ai concetti di guadagno e perdita derivanti da ciascuna. Un esempio di *framing effect* è dato dall'avversione al rischio di un investitore nei mercati finanziari.

Entrambi gli effetti possono condurre a decisioni errate, basate su ragionamenti e giudizi irrazionali, e che ovviamente conducono spesso a risultati non ottimali.

Non sempre ad una buona decisione segue un buon risultato; un modello decisionale efficiente può solo garantire una buona decisione, ma non un risultato ottimale.

1.3 Ottimizzazione

La pratica dell'**ottimizzazione** (per esempio di un modello o di una funzione) ha come scopo il trovare la migliore soluzione; basandosi sul concetto di migliore è quindi sensibile alla definizione che si da a tale situazione (il "migliore" non è univoco). Il

problema di ottimizzazione può consistere sia nella massimizzazione che nella minimizzazione, a seconda della funzione obiettivo impiegata e/o del fenomeno studiato; solitamente quando si parla di ottimizzazione applicata al *decision making* si fa riferimento a quella matematica. Un esempio di processi che si servono dell'ottimizzazione è la catena dei rifornimenti (*supply-chain*) e la logistica: l'azienda Amazon si serve di diversi sistemi di ottimizzazione per studiare la disposizione ideale delle merci nei suoi magazzini, il tragitto dei corrieri,... tutto al fine di minimizzare i costi (di produzione, di trasporto, ...). Altri esempi di settori riguardano il *machine learning*, trasporto, le consegne a domicilio, il settore energetico, le decisioni mediche, il settore finanziario... tutti con problemi di ottimizzazione caratterizzati da probabilità ignote (incertezza). Il *machine learning* utilizza spesso processi di ottimizzazione: per esempio un algortitmo di *clustering* utilizza una funzione obiettivo (distanza), al fine di massimizzare l'omogeneità interna (e l'eterogeneità esterna) ai cluster, in alcuni casi sotto il vincolo esclusivo di appartenenza di ogni osservazione ad un solo cluster.

Il problema che rende necessaria l'ottimizzazione dei processi decisionali riguarda la limitata presenza di risorse; ogni settore ha infatti una certa quantità di mezzi di produzione, capitali, materie prime o altri tipi di risorse (capacità computazionale e *storage*, personale, spazio fisico, ...). Il problema di ottimizzazione assume quindi il fine di trovare la situazione ottimale, o perlomeno più efficiente, nel quale la scarsità di risorse debba essere allocata; possiamo dunque definire il processo di ottimizzazione come **Mathematical Programming**.

E' possibile scomporre le componenti di un problema di ottimizzazione, identificandone 3 principali:

- **Funzione Obiettivo**→ rappresenta lo scopo matematico dell'ottimizzazione, ovvero la funzione che deve essere minimizzata/massimizzata.
- **Decisioni**→sono la parte ignota del problema e rappresentano la possibilità dei *decision maker* di ottenere i risultati desiderati (sulla base della funzione obiettivo e sotto i vincoli del fenomeno). Dal punto di vista del problema rappresentano i valori ignoti delle variabili fornite alla funzione.
- **Vincoli**→consentono di escludere determinate decisioni, rendendone verosimili altre.

L'ottimizzazione diventa quindi: "trovare il valore delle variabili (decisione) il quale sia in grado di massimizzare/minimizzare la funzione obiettivo, quando i vincoli siano rispettati".

Linear Programming Problem (LP)

Dal punto di vista matematico il problema di ottimizzazione assumerà quindi la seguente forma:

$$\max_{\{f(\underline{X})\}} \; : \; f_0(X_1, ..., X_n) \;\; | \, f_1(X_1, ...X_n) <= b_1 \; ; \; ... \; ; \; f_m(X_1, ..., X_n) > b_m$$

f_0 rappresenta la funzione obiettivo da minimizzare (costi di trasporto) o da massimizzare (profitto).In caso tutte le funzioni f_i siano lineari, si dice che il problema di ottimizzazione è un **Linear Programming Problem** (LP); esistono 5 passaggi per formulare un LP:

1. Comprendere il problema a fondo.
2. Identificare le variabili decisionali.
3. Strutturare la funzione obiettivo, nel caso di un LP come combinazione lineare delle variabili decisionali ($\max_{\{f(\underline{X})\}} \; : \; \alpha_1 \cdot X_1 \; + \; ... \; + \; \alpha_n \cdot X_n$)
4. Trovare i vincoli.
5. Formulare il problema di massimizzazione tenendo conto dei precedenti punti.

L'esempio sottostante mostra una possibile soluzione di un caso di studio.

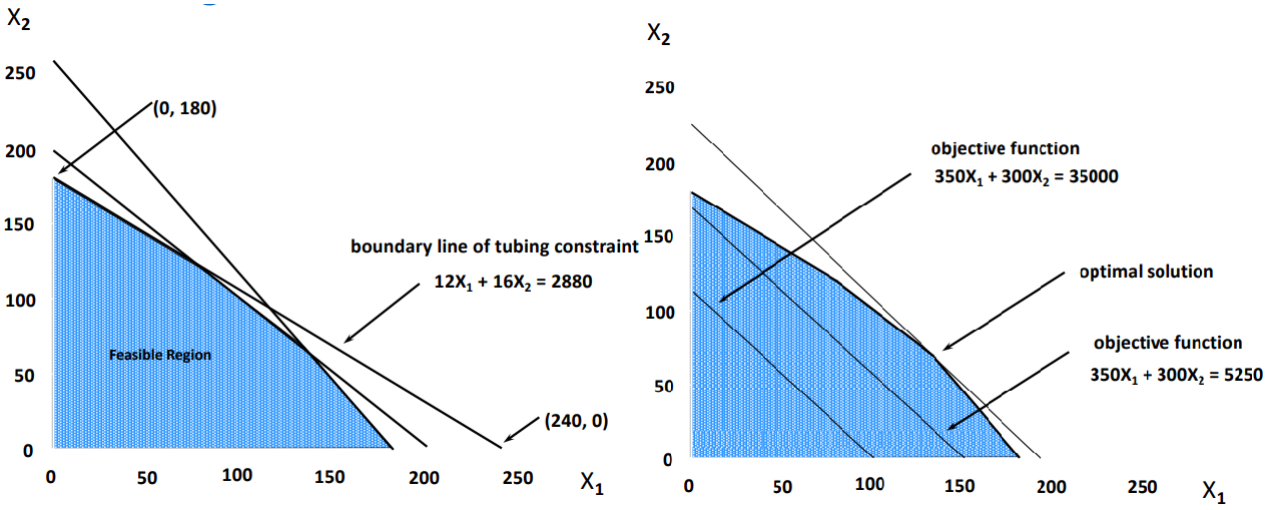
Solving LP Problems: An Intuitive Approach

MAX: $350X_1 + 300X_2$
S.T.: $1X_1 + 1X_2 <= 200$
$9X_1 + 6X_2 <= 1566$
$12X_1 + 16X_2 <= 2880$

$X_1 >= 0$
$X_2 >= 0$

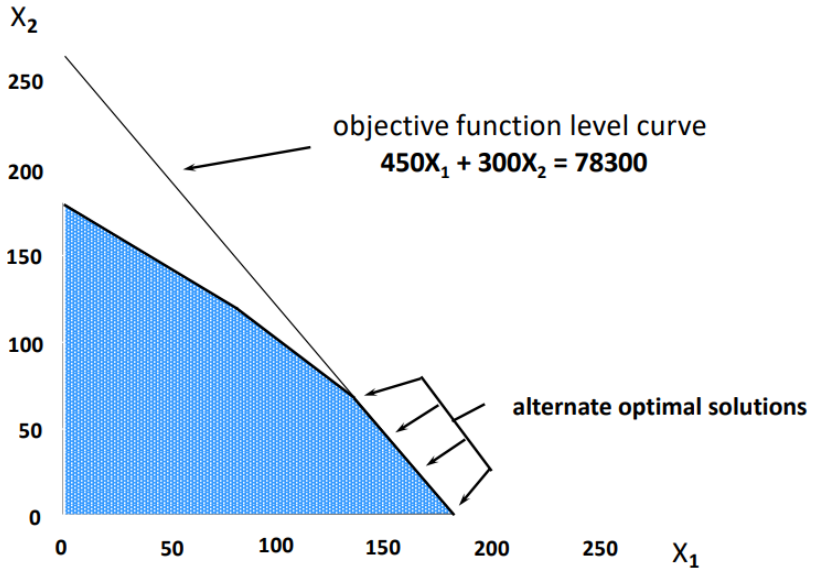
- Idea: Each Aqua-Spa (X_1) generates the highest unit profit ($350), so let's make as many of them as possible!
- How many would that be?
  - Let $X_2 = 0$
    - 1st constraint: $1X_1 <= 200$
    - 2nd constraint: $9X_1 <= 1566$ or $X_1 <= 174$
    - 3rd constraint: $12X_1 <= 2880$ or $X_1 <= 240$
- If X_2=0, the maximum value of X_1 is 174 and the total profit is $350*174 + $300*0 = $60,900
- This solution is *feasible*, but is it *optimal?*

Le funzioni che descrivono i diversi limiti evidenziano il dominio entro cui è possibile risolvere il problema di massimizzazione, ed è all'interno di questo spazio che è da ricercare il punto ottimale. In caso di modelli semplici (univariati o bivariati) la regione in cui esistono le soluzioni può essere identificata anche graficamente, mediante l'utilizzo di piani cartesiani (in R^2 o R^3). Nell'esempio precedente il dominio della funzione è dato dall'intersezione dei tre vincoli, come in figura sottostante.



I due grafici mostrano sia il dominio (con evidenziati i 3 vincoli), sia diverse funzioni obiettivo; il punto ottimale sarà quindi il massimo punto raggiungibile dalla funzione all'interno del dominio.

Generalmente le soluzioni non sono sempre così semplici; vi sono numerosi casi speciali in cui il problema esce dagli schemi appena descritti. Il primo esempio è quello di una funzione che identifichi più punti di soluzione ottimale all'interno del dominio. Tale condizione si ha quando la funzione obiettivo risulti parallela ad uno dei vincoli, come in figura:



Un altro caso particolare è quello dei vincoli inutili/ridondanti, ovvero la presenza di vincoli i quali non contribuiscono in alcun modo alla costruzione del dominio (quindi privati del loro scopo). Solitamente non è corretto tuttavia eliminare i vincoli ridondanti, poiché essi mantengono il loro significato logico anche in assenza di contributo alla struttura del dominio. Un ultimo caso è riferito ai domini non limitati, in cui purtroppo non è possibile identificare un punto ottimale.

I modelli LP si basano su una serie di **proprietà** che li rendono particolarmente adatti alla *sensitivity analysis*:

- **Proporzionalità**: il contributo che ciascuna variabile decisionale porta alla funzione obiettivo è proporzionale al valore della variabile stessa.
- **Additività**: il contributo che ciascuna variabile decisionale porta alla funzione obiettivo è indipendente dalle altre variabili.
- **Assenza di economie di scala**
- **Continuità**: ogni variabile decisionale può assumere valori decimali.
- **Certezza dei coefficienti**: tutti i coefficienti c_j delle variabili decisionali e b_i delle funzioni di vincolo sono noti con certezza.

2.1 Esempi di LP

2.1.1 Esempio LP: problema di produzione

Prendiamo in considerazione un azienda manifatturiera leader nella produzione di componenti nel campo dell'elettricità: la Electro-Poly Corporation. L'azienda ha da poco ricevuto un ordine di $750,000$ \$ per produrre un certo numero di componenti, in particolare scegliendo la combinazione ottimale di tre differenti modelli. Per ciascun modello si hanno le seguenti informazioni:

	Modello 1	**Modello 2**	**Modello 3**
Numero Ordini	3000	2000	900
Cablaggio ore/unità	2	1.5	3
Sfruttamento ore/unità	1	2	1
Costo di produzione	\$ 50	\$ 83	\$ 130
Costo di acquisto	\$ 61	\$ 97	\$ 145

La compagnia non ha tuttavia la capacità produttiva di soddisfare l'intero ordine; è infatti soggetta a due vincoli riguardanti la capacità produttiva, ovvero ha a disposizione 10,000 ore di cablaggio e 5,000 di sfruttamento. Non volendo perdere l'ordine la compagnia è disposta a passare una parte dell'ordine ad un suo *competitor*, ovviamente sostenendo dei costi maggiori rispetto alla produzione propria (costo di produzione $<$ costo di acquisto). La decisione che la compagnia dovrà prendere

riguarderà la quantità di componenti da produrre per ciascun modello, e sopratutto la quantità di componenti da produrre in proprio o da acquistare. Vi saranno quindi 6 variabili decisionali, le quali rappresentano le quantità per i 3 modelli secondo le due metodologie di ottenimento (produzione propria $\rightarrow M_1,\ M_2,\ M_3$; acquisto $\rightarrow B_1,\ B_2,\ B_3$). L'**ottimizzazione** del processo verrà quindi interpretata come una minimizzazione della funzione di costo, rappresentata dalla seguente forma:

$$\min_{\{F_c\}} : F_c = 50 \cdot M_1 + 83 \cdot M_2 + 130 \cdot M_3 + 61 \cdot B_1 + 97 \cdot B_2 + 145 \cdot B_3$$

Il problema deve esistere entro i **vincoli**:

- Vincoli di risorse
  - Cablaggio $\rightarrow 2 \cdot M_1 + 1.5 \cdot M_2 + 3 \cdot M_3 \leq 10,000$
  - Sfruttamento $\rightarrow 1 \cdot M_1 + 2 \cdot M_2 + 1 \cdot M_3 \leq 5,000$
- Vincoli di domanda:
  - Modello 1 $\rightarrow M_1 + B_1 = 3000$
  - Modello 2 $\rightarrow M_2 + B_2 = 2000$
  - Modello 3 $\rightarrow M_3 + B_3 = 900$
- Non negatività: $M_1, M_2, M_3, B_1, B_2, B_3 \geq 0$

La soluzione ottimale al problema è data da:

	Modello 1	**Modello 2**	**Modello 3**
Produzione	3000	550	900
Acquisto	0	1450	0

Tale soluzione ha costo pari a $\$453,300$, soddisfa il vincolo di domanda (e di non negatività), soddisfa il vincolo di risorse (Cablaggio: 9,525; Sfruttamento: 5,000)

2.1.2 Esempio LP: problema di investimento

Un investitore ha a disposizione $\$750,000$ da ripartire tra 6 diversi titoli; tali bonds hanno le seguenti caratteristiche:

Compagnia	**Guadagno**	**YTM**	**Rating**
Acme Chemical	8.65%	11	1-Excellent
DynaStar	9.50%	10	3-Good
Eagle Vision	10.00%	6	4-Fair
Micro Modelling	8.75%	10	1-Excellent
OptiPro	9.25%	7	3-Good
Sabre Systems	9.00%	13	2-Very Good

In tal caso il problema di ottimizzazione assume la forma di massimizzazione del profitto e minimizzazione del rischio. I vincoli del problema sono:

- Bisogna investire tutto il capitale: $\sum_{i=1}^{6} X_1 = 750,00$
- Non si può investire più del 25% in un'unica compagnia (differenziazione del portafoglio): $X_i \leq 187,500$; $\forall i = 1, ..., 6$
- Non si può investire più del 50% in bond con scadenze lunghe (YTM≥10 anni): $X_1 + X_2 + X_3 + X_4 \geq 375,000$
- Non si può investire più del 35% in DynaStar, Eagle Vision o OptiPro (alto rischio): $X_2 + X_3 + X_5 \leq 262,500$
- Non negatività: $X_i > 0$; $\forall i = 1, ..., 6$

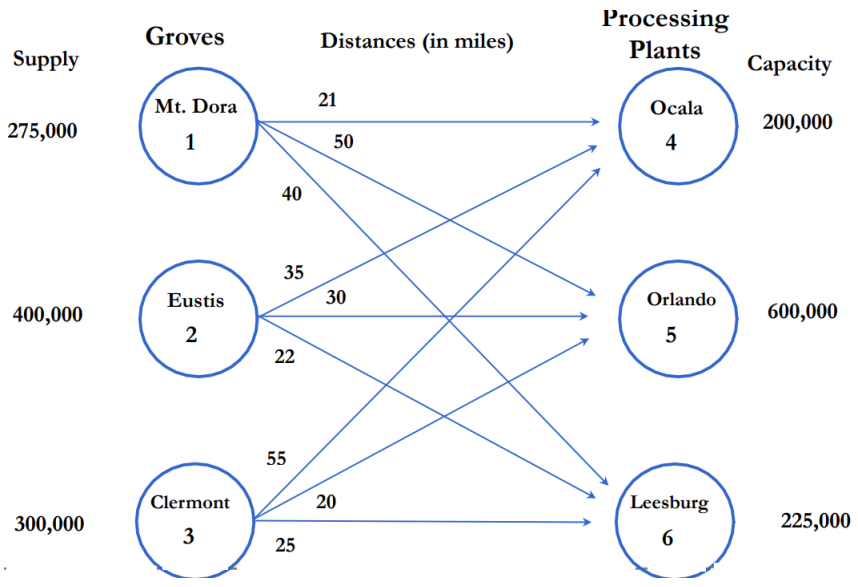
Il problema si comporrà quindi di 6 variabili decisionali, una per la quantità di capitale da allocare in ciascuna compagnia (X_i con $i = 1, 2, ..., 6$), e il problema di ottimizzazione avrà forma:

$$\max_{\{F(\underline{X})\}} : F(\underline{X}) = 0.865 \cdot X_1 + 0.95 \cdot X_2 + 0.100 \cdot X_3 + 0.875 \cdot X_4 + 0.925 \cdot X_5 + 0.900 \cdot X_6 +$$

In questo caso la soluzione ottimale sarà: $X_1 = \$187,500$, $X_2 = \$75,000$, $X_3 = \$187,500$, $X_4 = \$187,500$, $X_5 = \$0$, $X_1 = \$187,500$, la quale rispetta tutti i vincoli imposti e garantisce un guadagno di $\$75,375$

2.1.3 Esempio LP: problema di trasporto

Un terzo esempio di problema di ottimizzazione può riguardare il trasporto; immaginiamoci che vi siano 3 piantagioni e 3 impianti di lavorazione, come mostra l'esempio sottostante.



Ogni piantagione è collegata a tutti e 3 gli impianti di lavorazione, attraverso delle tratte con distanza variabile (come riportato in figura); inoltre ciascuna piantagione ha una certa quantità di risorse prodotte, mentre ogni impianto una certa quantità di risorse che possono essere lavorate. Dal momento che il problema riguarda quanto spedire, da dove e verso quale impianto, il numero di variabili decisionali coincide con il numero di possibili tratte, ovvero 9. Data $\underline{X}$ variabile multivariata riguardante le tratte, avremo che le variabili decisionali avranno forma X_{ij} (numero risorse da i a j) con $i = 1, 2, 3$ e $j = 4, 5, 6$. Il problema avrà quindi la seguente funzione obiettivo:

$$\min_{F(\underline{X})} : 21 \cdot X_{14} + 50 \cdot X_{15} + 40 \cdot X_{16} + 35 \cdot X_{24} + 30 \cdot X_{25} + 22 \cdot X_{26} + 55 \cdot X_{34} + 20 \cdot X_{35} + 25 \cdot X_{36}$$

la quale prevede la minimizzazione delle distanze percorse sotto i seguenti vincoli:

- **Vincoli di capacità**:
  - $X_{14} + X_{24} + X_{34} \leq 200,000$ (Ocala)
  - $X_{15} + X_{25} + X_{35} \leq 600,000$ (Orlando)
  - $X_{16} + X_{26} + X_{36} \leq 225,000$ (Leesburg)
- **Vincoli di risorse**:
  - $X_{14} + X_{15} + X_{16} \leq 275,000$ (Mt. Dora)
  - $X_{24} + X_{25} + X_{26} \leq 400,000$ (Eustis)
  - $X_{34} + X_{35} + X_{36} \leq 300,000$ (Clermont)
- **Non Negatività**: $X_{ij} \geq 0$, $\forall i, j$

La soluzione ottimale sarà data da:

	Ocala	**Orlando**	**Leesburg**
Mt. Dora	200,000	0	75,000
Eustis	0	250,000	150,000
Clermont	0	300,000	0

2.2 *Symplex Method*

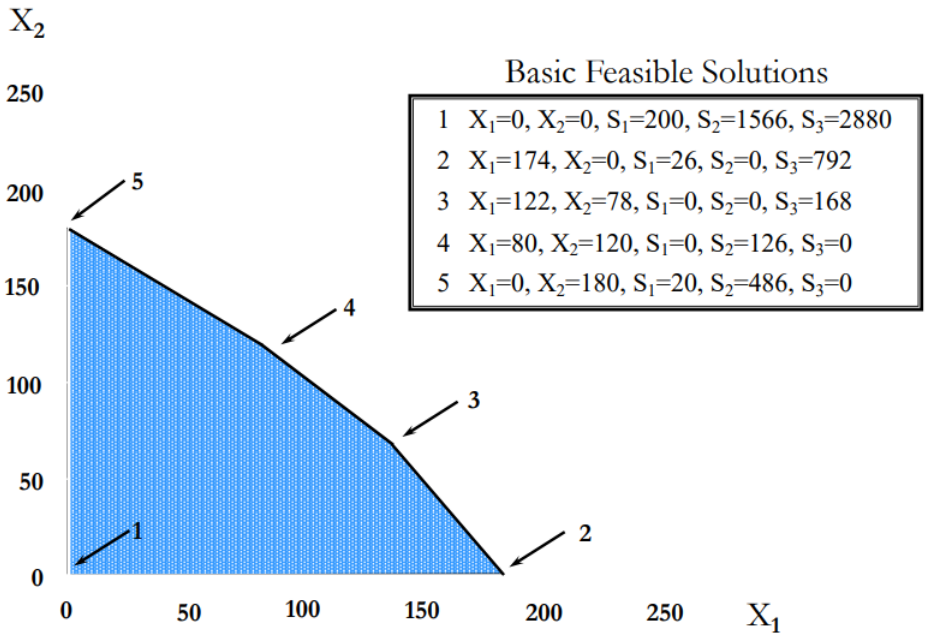
Il primo passo per l'applicazione del **symplex method** nella risoluzione di problemi di ottimizzazione lineari consiste nel trasformare ogni vincolo in un uguaglianza; avremo quindi:

$$\sum_{i=1}^{n} a_{ki} \cdot X_i \leq b_k \rightarrow \sum i = 1^n (a_{ki} \cdot X_i) \ + \ S_k = b_k$$

$$\sum_{i=1}^{n} a_{ki} \cdot X_i \geq b_k \rightarrow \sum i = 1^n (a_{ki} \cdot X_i) \ - \ S_k = b_k$$

La variabile (S_k) che viene utilizzata per riportare il vincolo ad un uguaglianza viene denominata *Slack Variable*. In secondo luogo si valuta il numero di variabili da impiegare, all'interno dei vincoli, per il problema di ottimizzazione: per esempio avendo a disposizione n variabili in un sistema di m equazioni vincolo (con $n \geq m$), sarà sufficiente impiegare m variabili e settare le rimanenti a 0. Le m variabili selezionate vengono chiamate *basic variables*, le $n - m$ settate a 0 invece *non-basic variables*. Nel caso si riesca a trovare una soluzione al sistema, mediante l'utilizzo di m *basic variables*, la soluzione verrà denominata *basic feasible solution*. Poiché settando a 0 un certo numero di variabili ci si sposta ai margini (*corners*) dell'area del dominio (*feasible region*), sapremo che in uno di tali punti vi sarà la soluzione ottimale al problema: Il problema di ottimizzazione consisterà quindi nel trovare il set ottimale di *basic variables*.

Un esempio grafico della posizione delle *basic feasible solutions*, rispetto al dominio, viene fornito dal grafico sottostante (nel quale vengono considerate 2 variabili decisionali e 3 *slack variables*).



Il *symplex method* non garantisce la via più veloce per la soluzione del problema di ottimizzazione, tuttavia porta ad identificare le soluzioni ottimali. Il procedimento può essere riassunto in 3 passaggi:

1. Identificare le **basic feasible solutions**, selezionando m variabili dal sistema di vincoli (composto da m equazioni), e settare le rimanenti $m - n$ a 0.
2. Spostarsi da una soluzione all'altra e valutare il valore della funzione obiettivo.
3. Nel caso nessun'altra soluzione adiacente garantisca un valore migliore della funzione obiettivo, tale soluzione sarà ottimale.

2.3 *Sensitivity Analysis*

Fin'ora le possibili soluzioni presentate si sono basate sull'assunzione che i coefficienti fossero noti a priori; chiaramente tale proprietà è molto stringente, e raramente si verifica. La *Sensitivity Analysis* prova a risolvere tale situazione concentrandosi sulla sensibilità con il quale cambiano le soluzioni ottimali alla variazione dei coefficienti. Dato il problema:

- $\max\limits_{F(\underline{X})} \; / \; \min\limits_{F(\underline{X})} \; : \; F(\underline{X}) \; = \; c_1 \cdot X_1 + ... + c_n \cdot X_n$
- soggetto ai vincoli: $a_{i1} \cdot X_1 + ... + a_{ij} \cdot X_j + ... + a_{in} \cdot X_n = b_i$

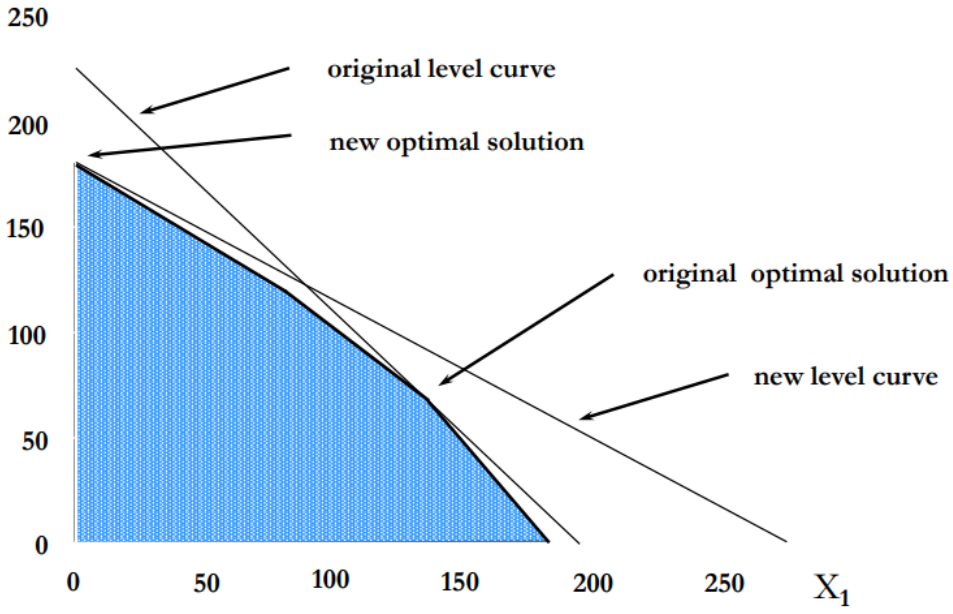
sarà necessario determinare la sensibilità delle soluzioni al cambio di c_i, a_{ij} e/o b_i.

Un esempio molto esplicativo riguarda una certa compagnia che voglia studiare come cambierebbero le decisioni al variare del prezzo di un bene o del costo di produzione.

L'**approccio** più diretto consiste nel far variare i coefficienti e rivalutare il modello e le soluzioni ottimali, per esempio mediante *symplex method*; tale metodo risulta l'unico utilizzabile in caso si vogliano valutare i cambiamenti di più parametri contemporaneamente, in particolare quando la soluzione del modello non richieda molto tempo. In generale lo studio della *sensitivity* può rispondere alle seguenti **task**:

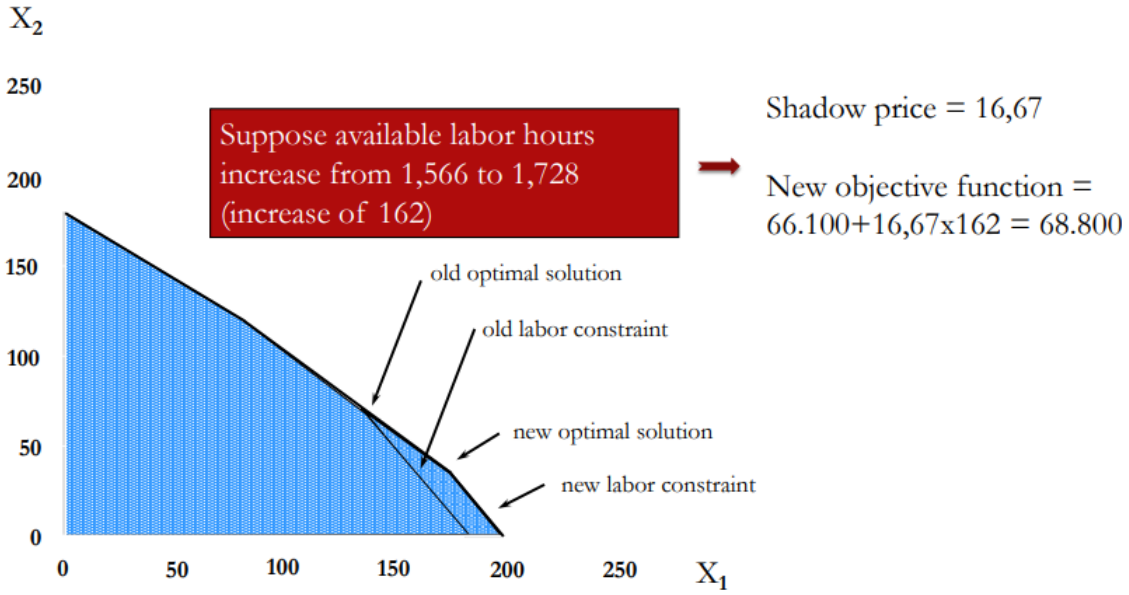
- Di quanto possono variare i coefficienti senza che vari la soluzione ottimale.
- Qual'è l'impatto delle variazioni dei coefficienti (vincoli e/o varabili decisionali) sulla funzione obiettivo.
- L'impatto che ha la variazione dei coefficienti sulla soluzione ottimale.

Graficamente viene mostrato un esempio di come possano variare soluzioni ottimali a fronte di una variazione dei coefficienti della funzione obiettivo.



Con **Range ottimale** si intende il campo di variazione dei coefficienti, all'interno del quale viene mantenuta la stessa soluzione ottimale.

Quando si hanno a disposizione vincoli nei quali la variabile *slack* è nulla ($S_k = 0$), allora si avrà un valore della *right hand side* del vincolo non nulla ($RHS = b_k \neq 0$). Definita tale situazione risulta interessante studiare la variazione nel risultato della funzione obiettivo a fronte di una variazione unitaria del RHS, sotto l'assunzione che gli altri coefficienti rimangano invariati; tale valore viene definito **Shadow Price**, ed è particolarmente utile quando vi siano cambiamenti nel valore finale dei vincoli, per esempio un aumento/diminuzione di risorse (ore macchina, ore lavoro, ...). Solitamente insieme all'ammissione di una variazione nel RHS si definisce anche un *range* in cui tale variazione risulti possibile. Un esempio di come possa variare la *feasible region* a seguito di una variazione del RHS, viene fornita dall immagine sottostante, nel quale ci si immagina un aumento di 162 unità nella disponibilità di ore lavorative.



Nell'esempio lo *shadow price* è pari a 16.67, ovvero pari all'aumento della funzione obiettivo a fronte di un aumento unitario del RHS; l'aumento totale sarà quindi dato da $16.67 \cdot 162 \simeq 2,700$ $(66,100 \rightarrow 68,800)$. Ovviamente nello studio dello *shadow price* bisogna sempre tenere conto del punto, all'interno della *feasible region*, su cui giace la soluzione ottimale; facendo infatti variare di poco un vincolo su cui non poggia direttamente la soluzione ottimale (*non-binding*), non vi saranno variazioni nella soluzione ottimale.

Lo *shadow price* rappresenta inoltre il limite superiore del costo cui siamo disposti a far fronte per ottenere nuove risorse; se il costo fosse maggiore avremmo infatti una decrescita nella funzione obiettivo. La differenza tra lo *shadow price* e i costi sostenuti viene denominato **Reduced Cost**.

Quanto esposto fin'ora è applicabile solo in caso le variazioni coinvolgano un solo coefficiente; se si vuole studiare la *sensitivity* quando variano simultaneamente più coefficienti si applica la **100% Rule**.

Si possono distinguere due casi:

1. Tutte le variabile con coefficienti variati hanno *reduced costs* non nulli.
2. Almeno una variabile con coefficiente variato ha *reduced cost* nullo.

Nel primo caso la soluzione ottimale rimane tale, sempre sotto ipotesi che le variazioni dei coefficienti siano all'interno del *range* consentito. Nel secondo caso bisogna invece calcolare, per ogni variabile, la seguente quantità:

$$r_j = \begin{cases} \frac{\triangle c_j}{I_j}, & \text{se } \triangle c_j \leq 0 \\ \frac{-\triangle c_j}{D_j}, & \text{se } \triangle c_j < 0 \end{cases}$$

Se la somma di tutti gli r_j (con $j = 1, ..., K$, K numero coefficienti variati) è minore o uguale a 1, allora la soluzione rimane ottimale, mentre in caso contrario l'ottimalità non è garantita.

Integrer Programming

Quando una o più variabili all'interno di un problema LP risultano discrete, si passa da una situazione di **Integer Programming** (ILP); alcuni esempi di applicazione dei modelli ILP riguardano la selezione degli orari di lavoro dei dipendenti e la fabbricazione di aeroplani. La formulazione di un ILP è simile a quella dei problemi lineari, con la differenza che almeno una variabile deve essere discreta:

- Original ILP

  MAX: $2X_1 + 3X_2$

  S.T.: $X_1 + 3X_2 <= 8.25$

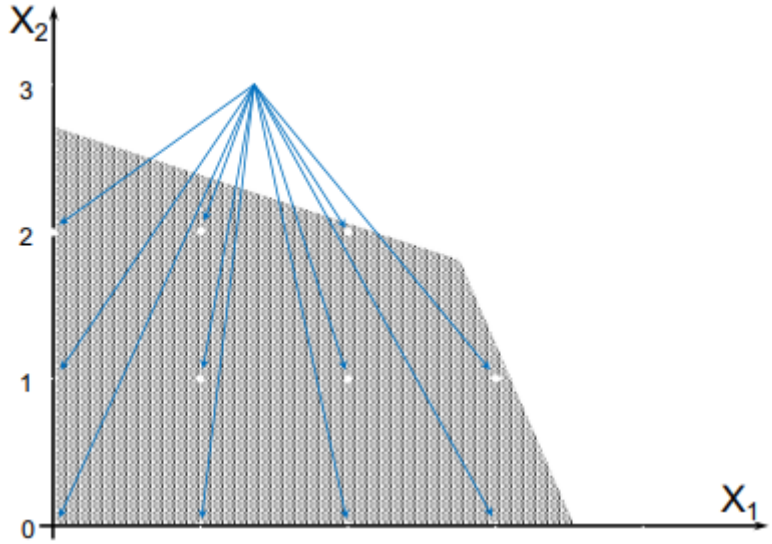
  $2.5X_1 + X_2 <= 8.75$

  $X_1, X_2 >= 0$

  X_1, X_2 must be integers

Tale formulazione implica che i modelli ILPs possano contenere anche variabili continue, con l'unico vincolo di averne una intera. Il primo passo per la risoluzione di un problema di programmazione intera consiste nello studio di un modello LP (chiamato *LP relaxation* del modello ILP), ovvero un classico problema lineare, al quale vengono sottratti i vincoli sulle variabili discrete, e che quindi può essere risolto con i metodi analizzati nelle precedenti sezioni.

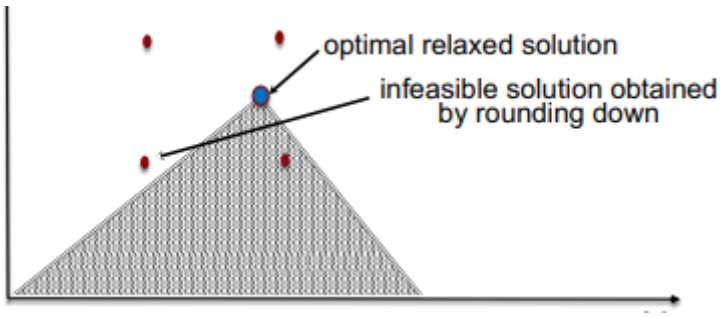
Uno dei principali effetti di un problema con variabili discrete riguarda la struttura della *feasible region*: per loro natura infatti vincoli che si servono di variabili discrete andranno a determinare dei domini rappresentati da singoli punti, e non più da aree come nel caso continuo.



La criticità di tale condizione risulta evidente, infatti fin'ora i punti di soluzione del problema sono sempre stati sui confini (*boundaries*) del dominio della funzione; poiché essi in un ILP non sono più definiti, la soluzione si troverà in altro modo.

Definita la *feasible region* di un ILP è ora possibile comprendere il perchè dell'utilizzo della *LP relaxation* (e del relativo *symplex mathod*): eliminando il vincolo sulle variabili discrete si identificherà una regione di dominio che conterrà sicuramente i punti che compongono la *feasible region* discreta (ovvero del problema ILP originale). Nel più fortunato dei casi uno dei punti appartenenti al dominio delle soluzioni del ILP coinciderà con un punto che risolve il modello *relaxed*; in tal caso LP e ILP avranno la stessa soluzione. Nella maggior parte dei casi tuttavia le soluzioni non coincideranno, e vi sarà la necessità di utilizzare metodi appositi per domini discreti.

Una prima semplice soluzione consiste nel determinare la soluzione del modello *relaxed* e arrotondarla alla soluzione intera più vicina: le problematiche derivano tuttavia dalla non garanzia che tale soluzioni risultino possibili, né tanto meno ottimali (come nell'immagine sottostante).



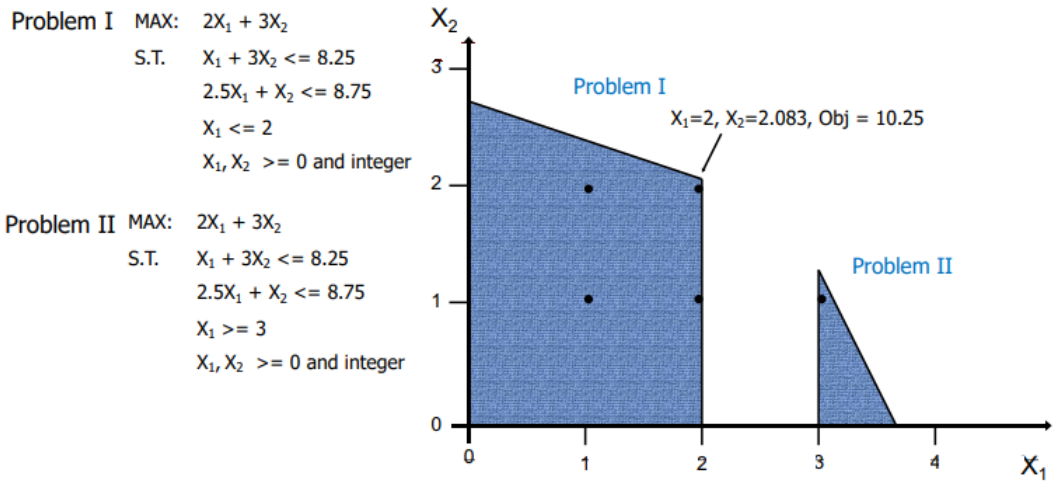
La soluzione dei ILP solitamente viene ricondotta ad un problema di candidati (*candidate points*); tale procedura può tuttavia essere particolarmente onerosa in termini di tempo e capacità computazionale, e per tale ragione è solitamente possibile specificare un *suboptimality tolerance factor*, un indice che misura il grado di accettazione di soluzioni non ottimali. In quest'ottica risultano fondamentali i *boundaries* identificati dal modello con vincoli rilassati.

Bisogna a questo punto distinguere i due casi dei problemi di ottimizzazione:

- **Massimizzazione**: il valore dell'*objective function* per la soluzione ottimale del modello rilassato (LP) sarà **limite superiore** della soluzione del modello discreto (ILP).
- **Minimizzazione**: il valore dell'*objective function* per la soluzione ottimale del modello rilassato (LP) sarà **limite inferiore** della soluzione del modello discreto (ILP).

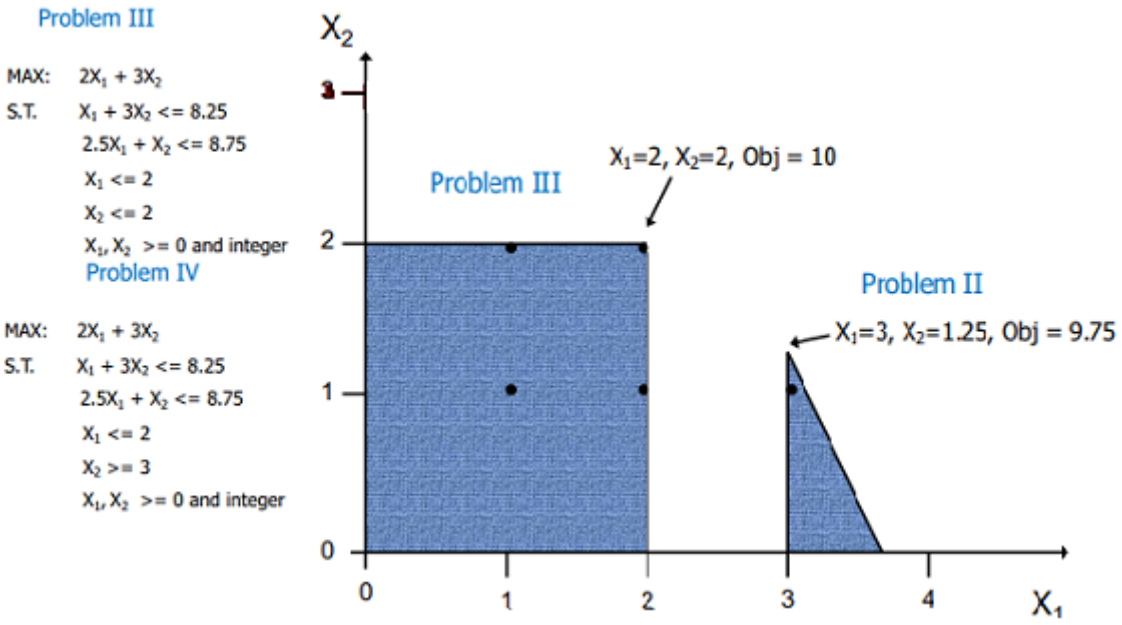
3.1 *Branch and Bound algorithm*

A partire dal problema di ottimizzazione in analisi si inizia risolvendo il modello con vincoli rilassati, ottenendo in tal modo la *feasible region* del modello LP, la quale conterrà i punti che risolvono il problema discreto. A questo punto è necessario escludere dal dominio tutte le soluzioni caratterizzate da valori non interi (poichè tali punti non potranno essere soluzione di un problema discreto). Il metodo *branch and bound* si serve di una *branching variable* (variabile ramificata), la quale ha lo scopo di eliminare i valori non interi: assumendo che una certa variabile X_1 abbia come soluzione, per il modello con vincoli rilassati, pari a 2.7, il passo successivo consisterà nel dividere il problema in due, uno nel quale viene inserito il vincolo $X_1 \leq 2$, mentre nel secondo verrà aggiunto $X_1 \geq 3$ (identificando una *feasible region* come in figura).



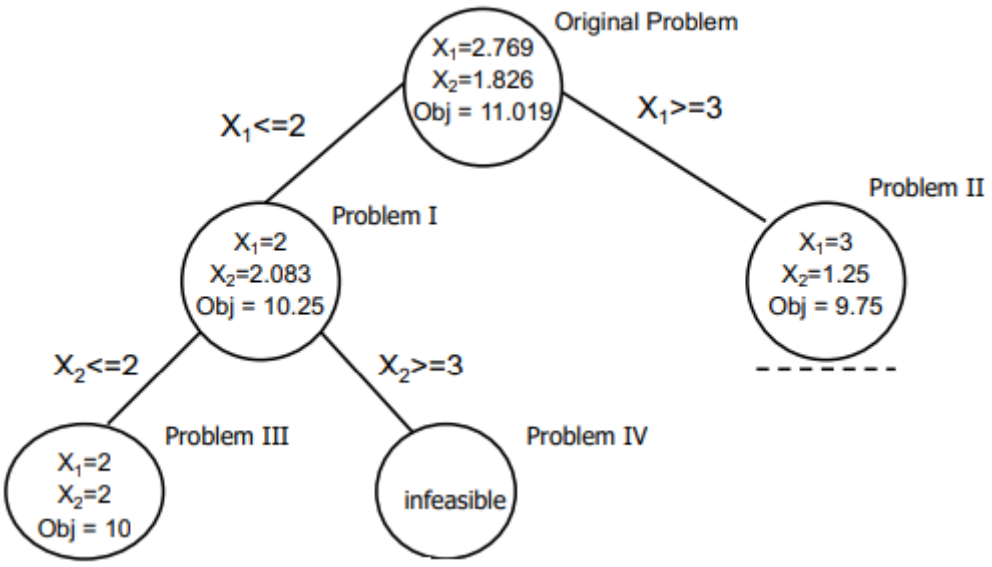
Come visibile la *feasible region* viene divisa in due, una per ciascun problema. A questo punto si applica la stessa procedura a X_2, e si divide una delle porzioni di do-

minio precedentemente individuate (quella contenente il punto ottimale) nuovamente in due:



Con questo procedimento si è arrivati a un doppio problema di ottimizzazione con due *feasible regions*; come si può vedere in figura le suddivisioni hanno permesso di modificare i confini della regione di dominio fino ad ottenere un *integer corner point* (ovvero una soluzione ottimale del problema discreto). A questo punto si confrontano i valori dell'*objective function* per i due diversi problemi e si identifica quale ne massimizza il valore.

Il procedimento descritto sarà caratterizzato da una struttura ad albero (*Branch and Bound Tree*), dove i rami rappresentano le suddivisioni delle *brenching variables*, mentre i nodi rappresentano i diversi problemi causati da tali suddivisioni.

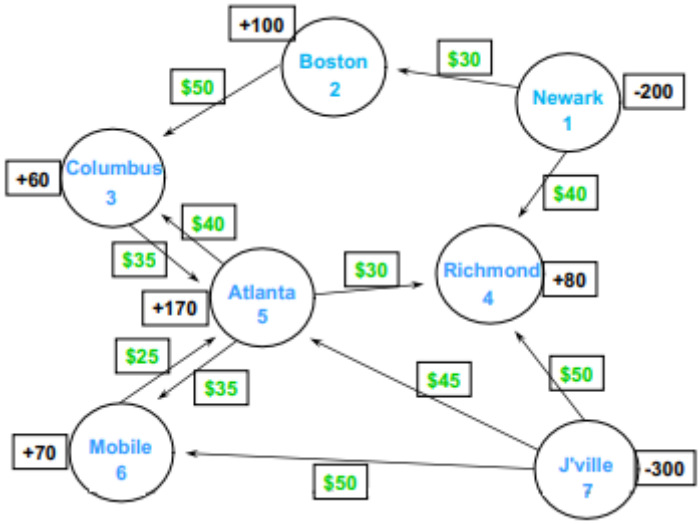


3.2 *Network Models*

Esistono diversi problemi di *business* rappresentabili da strutture a rete, e altrettanti algoritmi per risolverli; alcuni esempi sono i problemi di trasporto, di *shortest path*, di flusso massimo e molti altri. Una rete (*network flow*) è rappresentabile come una collezione di nodi collegati tra loro da rami/archi; possiamo identificare 3 diverse tipologie di nodi:

- *Supply nodes*
- *Demand nodes*
- *Transshipment nodes*

Un esempio di problema di trasporto rappresentabile da un struttura a rete è quello mostrato in figura (Compagnia manifatturiera di macchine di lusso):



Tipicamente i *supply nodes* verranno identificati con numeri negativi (Newark ha a disposizione 200 macchine da spedire), mentre i *demand nodes* da numeri positivi (Mobile necessita di 70 macchine). Gli archi che collegano i nodi di una rete sono direzionali, e un nodo che abbia archi sia in entrata, sia in uscita, viene definito *transshipment node* (nell'immagione lo è Atlanta). Possiamo considerare come variabili decisionali X_{ij}, ovvero l'ammontare di prodotti spediti dal nodo i al nodo j; il numero di variabili sarà uguale quindi al numero di archi. Il problema nell'esempio assumerà forma:

$$\begin{aligned} \text{MIN: } & 30X_{12} + 40X_{14} + 50X_{23} + 35X_{35} \\ & +40X_{53} + 30X_{54} + 35X_{56} + 25X_{65} \\ & + 50X_{74} + 45X_{75} + 50X_{76} \end{aligned}$$

ovvero un problema di minimizzazione dei costi di spedizione, il quale prenderà il nome *minimum cost network flow problem.*

Tali tipologie di problemi sono soggetti alle seguenti restrizioni legate ai flussi iin entrata e in uscita:

For Minimum Cost Network Flow Problems Where:	Apply This Balance-of-Flow Rule At Each Node:
Total Supply > Total Demand	Inflow-Outflow >= Supply or Demand
Total Supply < Total Demand	Inflow-Outflow <=Supply or Demand
Total Supply = Total Demand	Inflow-Outflow = Supply or Demand

Che nel caso dell'esempio precedente diventa:

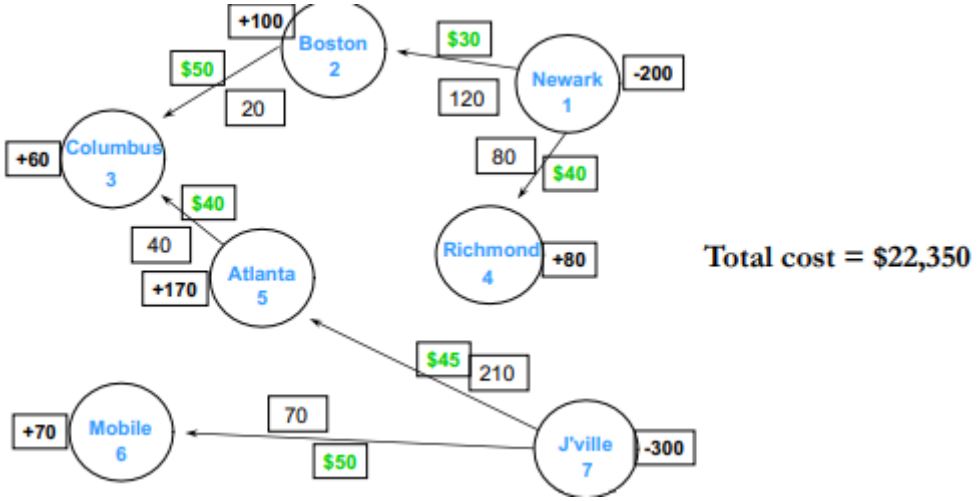
- Flow constraints

$-X_{12} - X_{14} >= -200$ } node 1
$+X_{12} - X_{23} >= +100$ } node 2
$+X_{23} + X_{53} - X_{35} >= +60$ } node 3
$+ X_{14} + X_{54} + X_{74} >= +80$ } node 4
$+ X_{35} + X_{65} + X_{75} - X_{53} - X_{54} - X_{56} >= +170$ } node 5
$+ X_{56} + X_{76} - X_{65} >= +70$ } node 6
$-X_{74} - X_{75} - X_{76} >= -300$ } node 7

- Nonnegativity conditions

$X_{ij} >= 0$ for all ij

Sotto tali vincoli la soluzione ottimale, che minimizza i costi, può essere rappresentata dalla seguente rete:

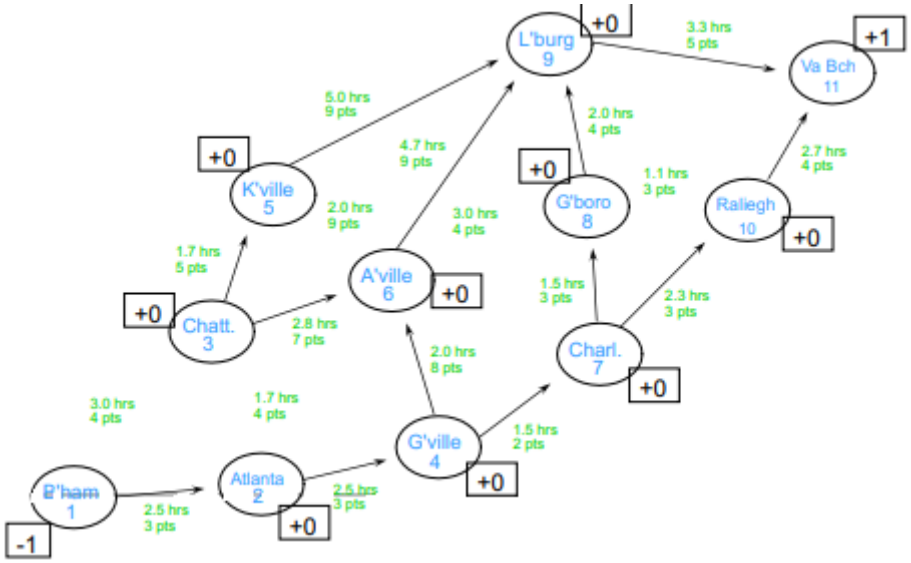


3.2.1 *Shortest path problem*

Dall'esempio presentato nella sezione precedente si può intuire l'importanza si determinare il percorso più breve, e meno costoso in termini di denaro, per lo spostamento di merci e prodotti; avendo una struttura a rete lo *shortest path* consisterà nell'insieme degli archi che appartengono alla soluzione ottimale. In un problema di trasporto, per esempio, è fondamentale che lo *shortest path* risulti il più efficiente in termini di costo degli spostamenti e, in tutti i casi in cui vi sia una tabella da

rispettare, in termini di tempistiche; quale dei due vincoli debba prevalere dipende dalle necessità dell'azienda e dalle peculiarità del problema.

Prendendo un esempio molto semplice, in cui vi siano un nodo con un bene di scorta e un altro nodo che domanda tale bene (tutti gli altri non possiedono ne domandano beni); la rete potrebbe essere rappresentata come segue:



All'interno di tale flusso sono segnate le tempistiche per il passaggio da un nodo all'altro e un punteggio per il paesaggio (più alto è più affascinante sarà il tratto); esistono due possibili problemi di ottimizzazione che possono essere risolti:

- Un problema di massimizzazione del punteggio al paesaggio.

Select Route?	From		To		Driving Time	Scenic Rating
1.0	1	Birmingham	2	Atlanta	2.5	3
0.0	1	Birmingham	3	Chattanooga	3.0	4
1.0	2	Atlanta	3	Chattanooga	1.7	4
0.0	2	Atlanta	4	Greenville	2.5	3
1.0	3	Chattanooga	5	Knoxville	1.7	5
0.0	3	Chattanooga	6	Asheville	2.8	7
0.0	4	Greenville	6	Asheville	2.0	8
0.0	4	Greenville	7	Charlotte	1.5	2
1.0	5	Knoxville	6	Asheville	2.0	9
0.0	5	Knoxville	9	Lynchburg	5.0	9
0.0	6	Asheville	8	Greensboro	3.0	4
1.0	6	Asheville	9	Lynchburg	4.7	9
0.0	7	Charlotte	8	Greensboro	1.5	3
0.0	7	Charlotte	10	Raleigh	2.3	3
0.0	8	Greensboro	9	Lynchburg	2.0	4
0.0	8	Greensboro	10	Raleigh	1.1	3
1.0	9	Lynchburg	11	Virginia Beach	3.3	5
0.0	10	Raleigh	11	Virginia Beach	2.7	4
				Total	15.9	35

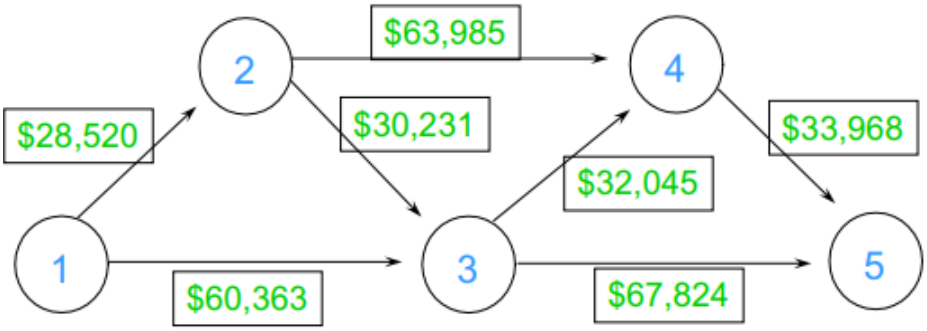
- Un problema di minimizzazione delle tempisiche.

Select Route?	From		To		Driving Time	Scenic Rating
1.0	1	Birmingham	2	Atlanta	2.5	3
0.0	1	Birmingham	3	Chattanooga	3.0	4
0.0	2	Atlanta	3	Chattanooga	1.7	4
1.0	2	Atlanta	4	Greenville	2.5	3
0.0	3	Chattanooga	5	Knoxville	1.7	5
0.0	3	Chattanooga	6	Asheville	2.8	7
0.0	4	Greenville	6	Asheville	2.0	8
1.0	4	Greenville	7	Charlotte	1.5	2
0.0	5	Knoxville	6	Asheville	2.0	9
0.0	5	Knoxville	9	Lynchburg	5.0	9
0.0	6	Asheville	8	Greensboro	3.0	4
0.0	6	Asheville	9	Lynchburg	4.7	9
0.0	7	Charlotte	8	Greensboro	1.5	3
1.0	7	Charlotte	10	Raleigh	2.3	3
0.0	8	Greensboro	9	Lynchburg	2.0	4
0.0	8	Greensboro	10	Raleigh	1.1	3
0.0	9	Lynchburg	11	Virginia Beach	3.3	5
1.0	10	Raleigh	11	Virginia Beach	2.7	4
				Total	11.5	15

Le tecniche di *shortest path* possono essere applicate anche a problemi che non prevedano degli spostamenti fisici; un esempio concreto riguarda la scelta tra diversi contratti di *leasing*. Un problema inerente è il seguente:

- Compu-Train provides hands-on software training.
- Computers must be replaced at least every two years.
- Two lease contracts are being considered:
  - Each requires $62,000 initially
  - Contract 1:
    - Prices increase 6% per year
    - 60% trade-in for 1 year old equipment
    - 15% trade-in for 2 year old equipment
  - Contract 2:
    - Prices increase 2% per year
    - 30% trade-in for 1 year old equipment
    - 10% trade-in for 2 year old equipment

A questo punto, prendendo in analisi il primo contratto, la struttura a rete assume la seguente forma:



Dove i nodi rappresentano l'anno e i collegamenti il periodo (massimo di 2 anni) trascorso prima del *trade-in*; per esempio dal nodo 1 (anno di partenza del contratto) partono due collegamenti, uno all'anno due (un anno prima del *trade-in*), e uno all'anno tre (due anni prima del *trade-in*), e i costi evidenziati rappresentano le spese (considerando quella iniziale di 62.200 e gli interessi del 6%) meno il *trade-in*. Applicando lo *shortest path* è possibile determinare quali siano le decisioni per la gestione del *leasing* in maniera tale da arrivare al quinto anno avendo sostenuto il costo minimo. La soluzione ottimale per tale contratto sarebbe cambiare i computer ogni anno:

Select	From	To	Cost
1.0	1	2	$28,520
0.0	1	3	$60,363
1.0	2	3	$30,231
0.0	2	4	$63,985
1.0	3	4	$32,045
0.0	3	5	$67,824
1.0	4	5	$33,968
		Total Cost	$124,764

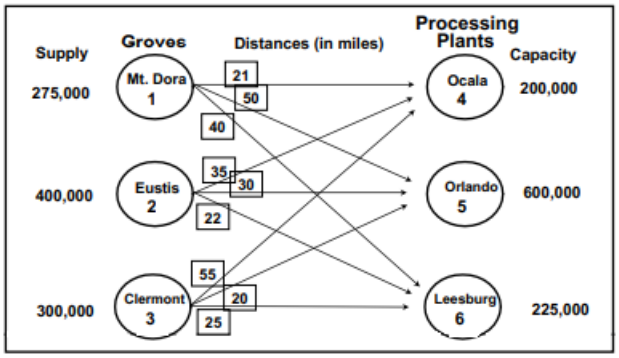
Confrontando tuttavia con la soluzione ottimale del secondo contratto si può notare come quest'ultimo risulti più conveniente:

Select	From	To	Cost
0.0	1	2	$44,640
1.0	1	3	$58,305
0.0	2	3	$45,533
0.0	2	4	$59,471
0.0	3	4	$46,443
1.0	3	5	$60,660
0.0	4	5	$47,372
		Total Cost	$118,965

3.2.2 *Transportation and Assignment problem*

Molto spesso si verifica che un problema venga rappresentato da una rete senza *trans-shipment nodes*, ovvero in cui vi sono solo nodi di domanda e approvigionamento, come nell'esempio che segue.

- Some network flow problems don't have trans-shipment nodes; only supply and demand nodes.



- Capacity constraints

  $X_{14} + X_{24} + X_{34} <= 200{,}000$ } Ocala
  $X_{15} + X_{25} + X_{35} <= 600{,}000$ } Orlando
  $X_{16} + X_{26} + X_{36} <= 225{,}000$ } Leesburg

- Supply constraints

  $X_{14} + X_{15} + X_{16} = 275{,}000$ } Mt. Dora
  $X_{24} + X_{25} + X_{26} = 400{,}000$ } Eustis
  $X_{34} + X_{35} + X_{36} = 300{,}000$ } Clermont

- Nonnegativity conditions

  $X_{ij} >= 0$ for all i and j

X_{ij} = # of bushels shipped from node i to node j

Specifically, the nine decision variables are:

X_{14} = # of bushels shipped from Mt. Dora (node 1) to Ocala (node 4)
X_{15} = # of bushels shipped from Mt. Dora (node 1) to Orlando (node 5)
X_{16} = # of bushels shipped from Mt. Dora (node 1) to Leesburg (node 6)
X_{24} = # of bushels shipped from Eustis (node 2) to Ocala (node 4)
X_{25} = # of bushels shipped from Eustis (node 2) to Orlando (node 5)
X_{26} = # of bushels shipped from Eustis (node 2) to Leesburg (node 6)
X_{34} = # of bushels shipped from Clermont (node 3) to Ocala (node 4)
X_{35} = # of bushels shipped from Clermont (node 3) to Orlando (node 5)
X_{36} = # of bushels shipped from Clermont (node 3) to Leesburg (node 6)

Minimize the total number of bushel-miles.

$$\text{MIN: } 21X_{14} + 50X_{15} + 40X_{16} + 35X_{24} + 30X_{25} + 22X_{26} + 55X_{34} + 20X_{35} + 25X_{36}$$

3.2.3 *Generalized Network Flow problems*

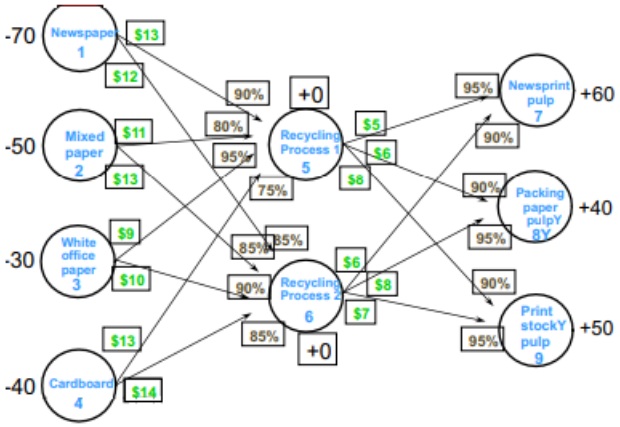
In alcune particolari *business task* si ha a che fare con problemi di *gain-loss*, ovvero attraverso gli archi transitano funzioni di perdita e di guadagno; un esempio potrebbe essere il trasporto del gas attraverso a condutture che ne disperdono un certa quantità. In tal caso si utilizzano le *generalized network flow*.

Un esempio di una condizione *gain-loss* riguarda un'azienda operante nel settore del riciclo della carta; vi sono diversi processi in grado di ottenere carta riciclata da

prodotti usati, ciascuno con un proprio costo e tasso di riciclo. Di seguito vengono riportati alcuni dati per tale esempio.

	Process 1		Process 2		
Material	Cost	Yield	Cost	Yield	Supply
Newspaper	$13	90%	$12	85%	70 tons
Mixed Paper	$11	80%	$13	85%	50 tons
White Office Paper	$9	95%	$10	90%	30 tons
Cardboard	$13	75%	$14	85%	40 tons

	Newsprint		Packaging Paper		Print Stock	
Pulp Source	Cost	Yield	Cost	Yield	Cost	Yield
Recycling Process 1	$5	95%	$6	90%	$8	90%
Recycling Process 2	$6	90%	$8	95%	$7	95%
Demand	60 tons		40 tons		50 tons	



- Raw Materials
  - $-X_{15} - X_{16} >= -70$ } node 1
  - $-X_{25} - X_{26} >= -50$ } node 2
  - $-X_{35} - X_{36} >= -30$ } node 3
  - $-X_{45} - X_{46} >= -40$ } node 4
- Recycling Processes

$+0.9X_{15}+0.8X_{25}+0.95X_{35}+0.75X_{45}- X_{57}- X_{58}-X_{59} >= 0$ } node 5

$+0.85X_{16}+0.85X_{26}+0.9X_{36}+0.85X_{46}-X_{67}-X_{68}-X_{69} >= 0$ } node 6

- Paper Pulp
  - $+0.95X_{57} + 0.90X_{67} >= 60$ } node 7
  - $+0.90X_{57} + 0.95X_{67} >= 40$ } node 8
  - $+0.90X_{57} + 0.95X_{67} >= 50$ } node 9

MIN: $13X_{15} + 12X_{16} + 11X_{25} + 13X_{26} + 9X_{35} + 10X_{36} + 13X_{45} +$
$+ 14X_{46} + 5X_{57} + 6X_{58} + 8X_{59} + 6X_{67} + 8X_{68} + 7X_{69}$

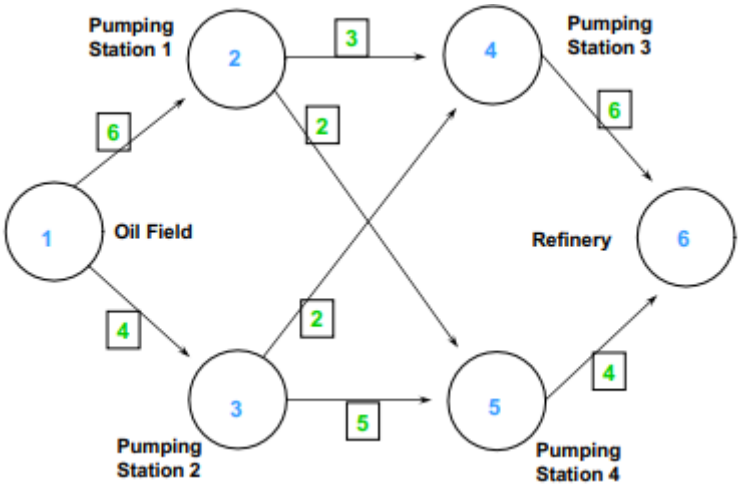
A differenza dei precedenti esempi nel *generalized network flow* ad ogni arco viene associata una certa quota di prodotto perso (in particolare nell'esempio rappresentata dalla parte di carta persa durante i processi). La soluzione del problema sarà la seguente:

Flow From Node			Yield	Flow Into Node			Cost
43.4	1	Newspaper	0.90	39.1	5	Process 1	$13
26.6	1	Newspaper	0.85	22.6	6	Process 2	$12
50.0	2	Mixed Paper	0.80	40.0	5	Process 1	$11
0.0	2	Mixed Paper	0.85	0.0	6	Process 2	$13
30.0	3	White Office	0.95	28.5	5	Process 1	$9
0.0	3	White Office	0.90	0.0	6	Process 2	$10
0.0	4	Cardboard	0.75	0.0	5	Process 1	$13
35.4	4	Cardboard	0.85	30.1	6	Process 2	$14
63.2	5	Process 1	0.95	60.0	7	Newsprint	$5
44.4	5	Process 1	0.90	40.0	8	Packaging	$6
0.0	5	Process 1	0.90	0.0	9	Print Stock	$8
0.0	6	Process 2	0.90	0.0	7	Newsprint	$6
0.0	6	Process 2	0.95	0.0	8	Packaging	$8
52.6	6	Process 2	0.95	50.0	9	Print Stock	$7
						Total Cost	$3,149

3.2.4 *Maximal Flow Problem*

Esistono alcuni problemi nel quale è necessario determinare la massima quantità che può passare attraverso un determinato arco; quando ci si trova in tale situazione è necessario risolvere un *maximal flow problem* nel quale, a ciascun arco, corrisponderà sia un limite superiore, sia un limite inferiore. Alcuni esempi riguardano la quantità massima di acqua che può essere trasportata da delle tubature, oppure il flusso di automobili che possono attraversare la città, o ancora il traffico dati massimo che può essere gestito da una rete.

Un esempio di tale problema riguarda una compagnia petrolifera che deve trasportare del petrolio dalla zona di estrazione fino alla raffineria, potendo scegliere un determinato percorso attraverso tubature (di diversa dimensione) che passano per 4 pompe. La stuttura della rete sarà dunque la seguente



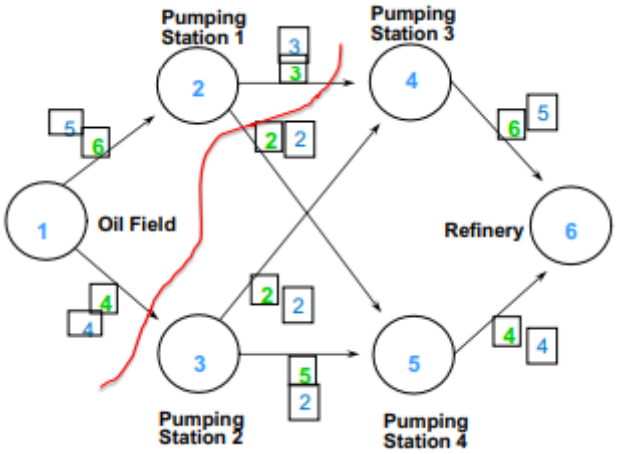
Per risolvere tale problema, in maniera analoga al precedente *transshipment problem*, bisogna aggiungere un arco alla rete, in particolare per collegare il nodo 6 al nodo 1 (X_{61}); a questo punto si assegna valore 0 alla domanda di tutti i nodi e si massimizza il flusso attraverso gli archi. Servendosi della nuova struttura la massima quantità di petrolio che potrà essere trasportato da 1 a 6 sarà equivalente alla quantità di petrolio che da 6 può arrivare a 1 (X_{16}) e tornare, attraversando la rete che passa per gli altri nodi. Bisogna inoltre fissare i limiti superiori, corrispondenti alle capacità degli archi (i numeri in figura), e i vincoli (*balance of flow constraints*)

$$
\begin{aligned}
&\text{MAX:} && X_{61} \\
&\text{Subject to:} && +X_{61} - X_{12} - X_{13} = 0 \\
& && +X_{12} - X_{24} - X_{25} = 0 \\
& && +X_{13} - X_{34} - X_{35} = 0 \\
& && +X_{24} + X_{34} - X_{46} = 0 \\
& && +X_{25} + X_{35} - X_{56} = 0 \\
& && +X_{46} + X_{56} - X_{61} = 0
\end{aligned}
$$

with the following bounds on the decision variables:

$$
\begin{array}{lll}
0 <= X_{12} <= 6 & 0 <= X_{25} <= 2 & 0 <= X_{46} <= 6 \\
0 <= X_{13} <= 4 & 0 <= X_{34} <= 2 & 0 <= X_{56} <= 4 \\
0 <= X_{24} <= 3 & 0 <= X_{35} <= 5 & 0 <= X_{61} <= \text{inf}
\end{array}
$$

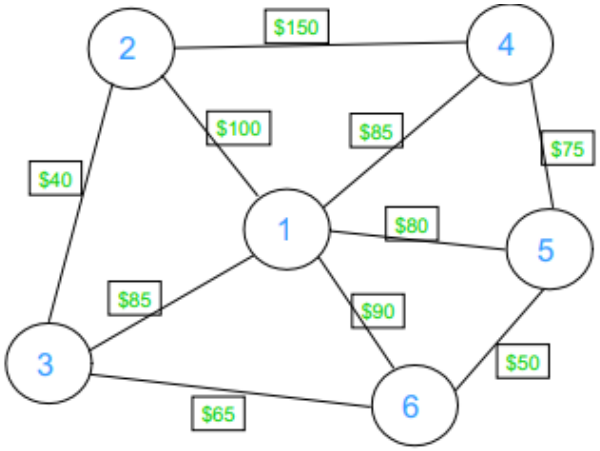
Come si può notare il problema è diventato la massimizzazione della variabile X_{16}, sotto i vincoli che ciò che arriva a ciascun nodo deve uguagliare ciò che esce (il primo vincolo afferma che ciò che arriva a 1 deve essere uguale a ciò che esce verso 2 e 3). Si nota subito la differenza dai precedenti esempio di *transshipment problems* in cui i vincoli di *balance of flows* non erano rappresentati da equazioni, ma da disuguaglianze. I limiti inferiori sono pari a 0 (ovvero nel caso in cui non vi sia transito di petrolio), mentre i limiti superiori sono pari alle capacità massime degli archi; per la variabile X_{61} il limite superiore risulta $+\infty$ poichè è la quantità da massimizzare. La soluzione sarà la seguente



dove in verde sono segnate le capacità massime, mentre in blu la quantità di prodotto che transita in ciascun arco nella soluzione ottimale. E' possibile, come avviene in figura, dividere i nodi in due *subset*, in maniera tale da dividere gli archi che utilizzano la capacità massima (*saturated arcs*) da quelli che non la raggiungono.

3.2.5 *Minimal Spanning Tree problem*

Dato un set di n nodi un algoritmo di *Minimal Spanning Tree* ha l'obbiettivo di identificare gli $n-1$ archi che consentano la connessione di tutti i nodi, senza tuttavia generare cicli; in tal caso il problema di ottimizzazione consiste nel determinare l'insieme di archi che minimizzi i costi o le distanze (a seconda del problema). Prendendo come esempio la rete in figura

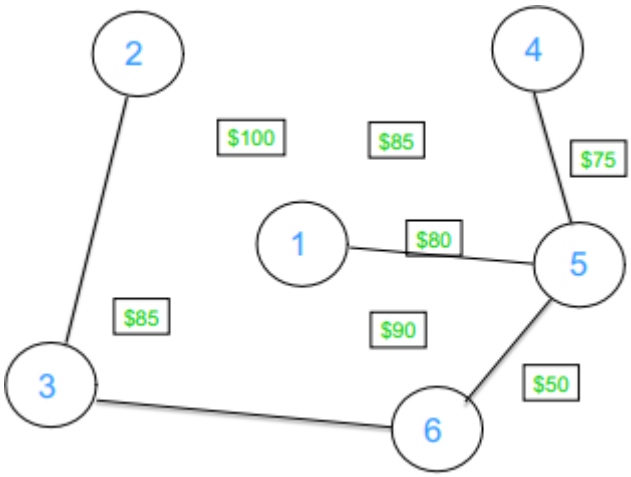


nel quale ciascun nodo rappresenta un computer, mentre gli archi rappresentano le connessioni. L'obbiettivo finale è che ciascuno computer sia collegato agli altri, in maniera tale da non escluderne nessuno, e che la somma dei costi di connessione (rappresentati dalle *label* verdi della rete) sia la minima possibile.

Il procedimento è differente rispetto a tutti i precedenti, e non si basa sull'utilizzo di LP, ma di un algoritmo apposito; il suo funzionamento è il seguente:

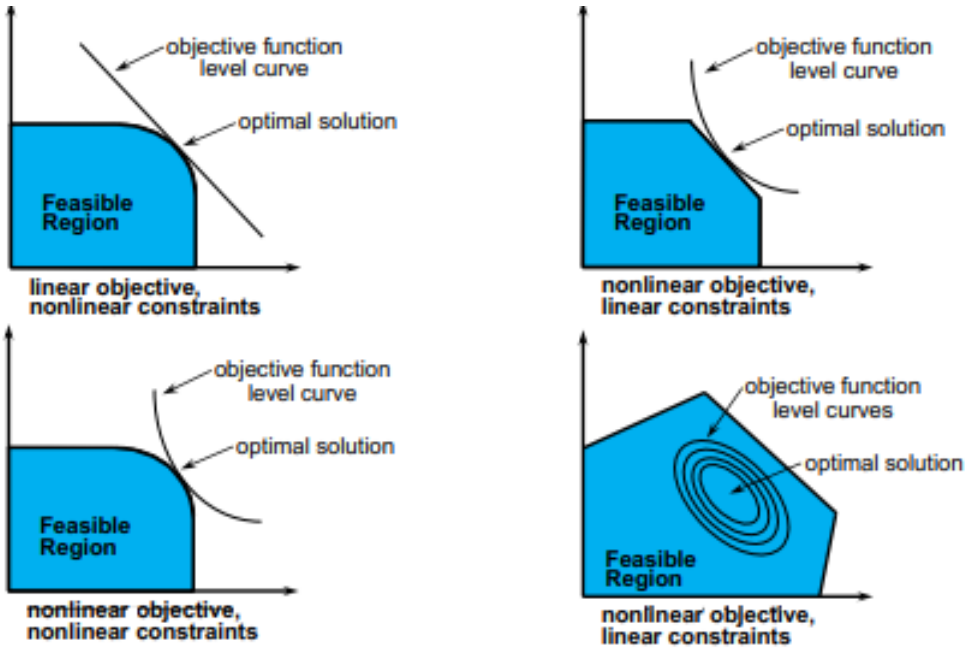
1. Decido di fissare un nodo iniziale, il quale rappresenterà la sotto-rete di partenza.
2. Collego tale nodo ad un altro, scegliendolo in base al costo minimo delle possibili connessioni; a questo punto la sotto-rete conterrà due elementi.
3. Itero il processo fino a che tutti i nodi saranno contenuti nella sotto-rete (la quale coinciderà con la rete ottimale).

La soluzione ottimale che minimizza i costi è la seguente:



Non-Linear Programming

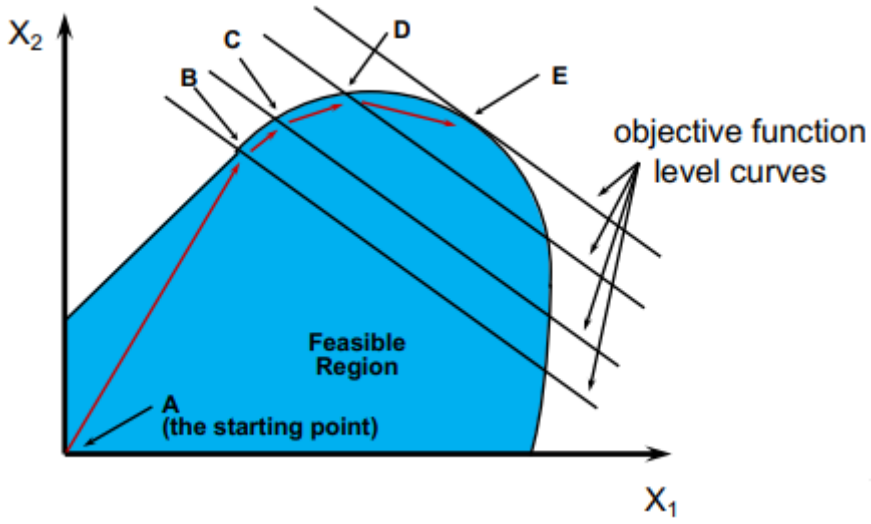
I concetti fin'ora presentati hanno come fondamento l'assunzione di problemi lineari sia nella funzione obiettivo, sia nei vincoli; quando tuttavia si analizzano determinati problemi non sempre si verificano le condizioni per utilizzare un approccio lineare nei modelli decisionali, e di conseguenza bisogna ricorrere a metodi non-lineari o **Non-Linear Programming** (NLP). Gli NLP consentono di superare l'assunzione di linearità e di conseguenza consentono di trattare problemi con relazioni non lineari nella funzione obiettivo e/o nei vincoli. Nonostante la formulazione e l'implementazione degli NLP sia analoga al caso lineare, la parte matematica coinvolta è differente e porta con se alcune criticità. Alcuni esempi di come si possono presentare dei problemi non lineari sono i seguenti:



La prima differenza viene subito messa in risalto dai grafici: la soluzione ottimale (e quindi il punto che ottimizza la funzione obiettivo) non cade più necessariamente nei *boundaries* della *feasible region*. Di conseguenza la filosofia di applicazione del *symplex method*, ovvero identificare i boundaries del dominio, non risulta più applicabile.

Esistono diversi metodi di risoluzione di problemi NLP, tuttavia hanno tutti in comune l'**iteratività dei processi**: il funzionamento di base consiste nella selezione

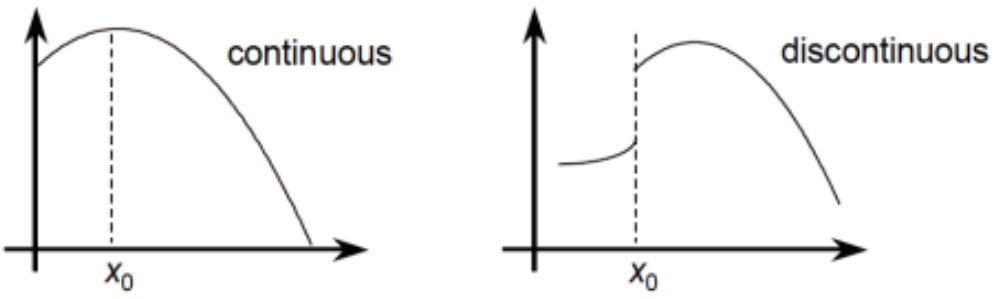
di un punto iniziale, solitamente all'interno della *feasible region*, dal quale ci si sposta iterativamente fino a che non è sia più possibile, ovvero fino al raggiungimento del confine del dominio (come mostra la figura sottostante). Il numero di spostamenti/iterazioni che l'algoritmo NLP compie prima di giungere ad una soluzione viene definito *step size*.



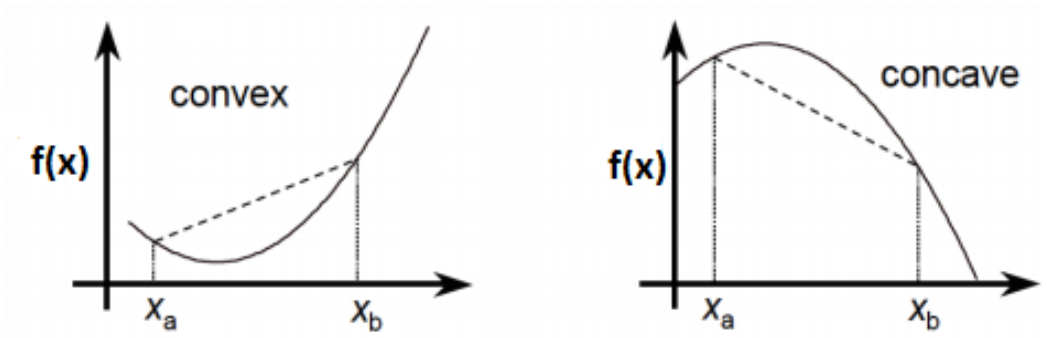
La principale criticità di tale metodo è l'assenza di garanzia che il punto identificato al termine delle iterazioni rappresenti un punto di ottimo globale, ovvero che venga individuato un punto che ottimizza la funzione solo localmente. In tale contesto la soluzione ottimale viene fortemente influenzata dalla selezione del punto iniziale; solitamente sarebbe preferibile escludere l'origine degli assi come punto iniziale, ma invece scegliere punti che siano della grandezza del valore ottimale atteso.

Vi sono alcune proprietà distintive delle funzioni da analizzare per studiare un problema NLP:

- **Continuità** di una funzione: una funzione f si dice continua in x_0 sse $f(x_0)$ esiste e se $\lim_{x \to x_0^{(+)}} f(x_0) = \lim_{x \to x_0^{(-)}} f(x_0) = x_0$



- **Convessità** di una funzione: una funzione f si dice convessa sse $f(\theta \cdot x_a + +[1-\theta]) \cdot x_b \leq \theta \cdot f(x_a) + [1-\theta] \cdot f(x_b)$, con x_a, x_b punti della funzione e $\theta \in [0,1]$ parametro noto. In caso la condizione non sia rispettata la funzione sarà concava



- **Punto di minimo**: un punto x^* viene definito di minimo locale sse la funzione, in un intorno di x^* $[x_a, x_b]$ assume valore minimo in x^*, ovvero $f(x^*) \leq f(x)$, $\forall x \in [x_a, x_b]$. In caso f sia convessa, x^* sarà anche punto di minimo globale; ragionamento analogo ma contrario per i punti di massimo locali (dove f concava implica x^* massimo globale).

Un punto ottimale di una funzione dovrà di conseguenza annullare la derivata prima (punto stazionario); lo studio della derivata seconda ci fornisce informazione sulla tipologia di ottimo, ovvero $\delta^2 f < 0$ punto di massimo, $\delta^2 f > 0$ punto di minimo. In caso la derivata seconda sia uguale a 0 nel punto ottimale si studiano le derivate di ordine superiore $\delta^n f$, fino a quando non se ne trova una non nulla.

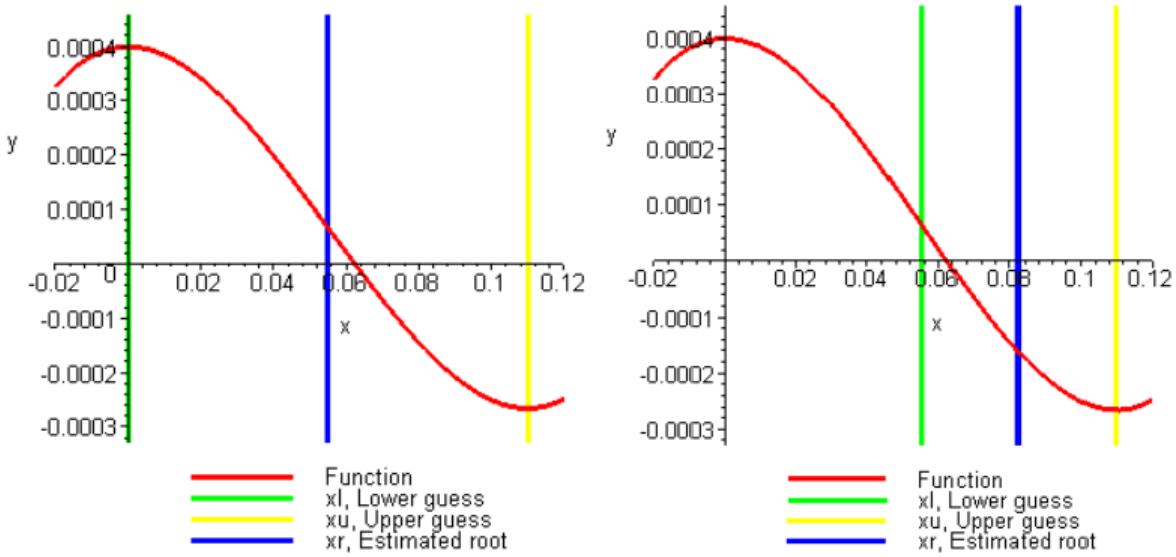
Il metodo appena mostrato, ovvero la soluzione analitica dell'*objective function*, funziona solo in caso di funzioni semplici; vi sono tuttavia molti i casi in cui non è possibile risolvere analiticamente un problema NLP, soprattutto quando si è in presenza di funzioni molto complesse. In tali situazioni si applicano procedure numeriche come il *Bisection Method*.

4.1 NLP univariati: *Bisection Method*

Data una funzione $f(x)$ (continua, reale e che cambia segno), $f(x) = 0$ avrà almeno una radice tra x_1 (con $f(x_1) < 0$) e x_2 (con $f(x_2) > 0$) se $f(x_1) \cdot f(x_2) < 0$; tale condizione è sufficiente ma non necessaria. Il metodo *bisection* si basa proprio sull'identificazione delle radici della funzione $f(x) = 0$ nel caso di NLP univariati. Il funzionamento dell'algoritmo è semplice:

1. Si scelgono due punti x_1, x_2 tali che la funzione cambi segno nell'intervallo compreso.
2. Si stima la radice di $f(x) = 0$ con il punto mediano $x_m = \frac{x_1+x_2}{2}$.
3. In caso $f(x_1) \cdot f(x_m) < 0$ la radice si troverà tra x_1, x_m, mentre se $f(x_1) \cdot f(x_m) > 0$ la radice si troverà tra x_m, x_2; in caso il prodotto si annulli, x_m rappresenta una radice di $f(x) = 0$.

4. Se si è verificata una delle prime due condizioni si stima nuovamente la radice come punto mediano tra i due nuovi punti (x_1, x_m o x_m, x_2). E' possibile stimare l'errore approssimato ad ogni iterazione i come $|\epsilon_i| = |\frac{x_m^i - x_m^{i-1}}{x_m^i}| \cdot 100$.
5. Viene fissata una soglia dell'errore ϵ, e in caso l'errore $\epsilon_i < \epsilon$ l'algoritmo si ferma; un'alternativa possibile a tale *stopping rule* può riguardare il numero massimo di iterazioni.



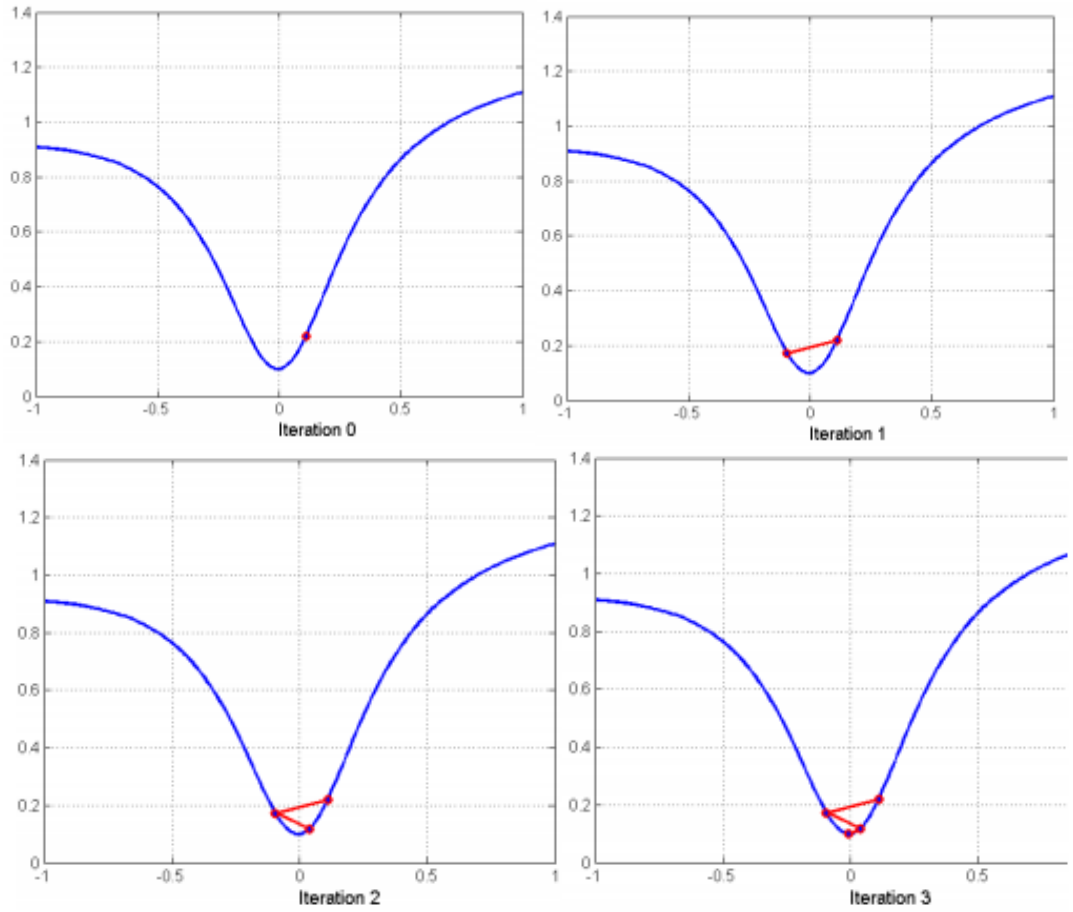
Tra i principali vantaggi nell'impiego del *bisection method* per trovare le soluzioni di un NLP univariato vi è la garanzia di convergenza, mentre lo svantaggio riguarda la lentezza con cui l'algoritmo converge (in particolare se la prima stima è molto vicina alla radice reale).

4.2 NLP univariati: *Newton's Method*

Il metodo di soluzione di Newton per NLP viene spesso utilizzato per la soluzione di problemi univariati, con tuttavia la possibilità di estensione anche al caso multivariato. Si serve di approssimazioni quadratiche della funzione utilizzando informazioni circa il gradiente e la curvatura; in tal senso utilizza approssimazioni di Taylor $f(x+h) = f(x) + f'(x) \cdot h + \frac{1}{2} f''(x) \cdot h^2$ con h un piccolo incremento della funzione. A questo punto si definisce la derivata prima della funzione quadratica $f'(x+h) = f'(x) + f''(x) \cdot h + o(h^2)$ e si trova la sua radice; possiamo omettere $o(h^2)$ poiché irrilevante per la soluzione. Lo spostamento della funzione sarà uguale ad $h = -\frac{f'(x)}{f''(x)}$; a questo punto il nuovo punto che approssimerà la soluzione ottimale sarà dato da $x_{k+1} = x_k + h_k = x_k - \frac{f'(x_k)}{f''(x_k)}$.

Tra i principali vantaggi vi sono la non necessità di conoscere il supporto della radice e la convergenza quadratica (l'accuratezza decimale raddoppia ad ogni iterazione); le criticità riguardano la non garanzia di convergenza e i fallimenti dovuti alla troppa distanza tra il punto ottimale e la stima.

Una descrizione del funzionamento viene mostrata nei seguenti grafici



4.3 NLP multivariati: *Gradient Method*

Passando dal caso univariato al multivariato lo studio delle derivate si trasforma nello studio del **gradiente** della funzione: data una funzione f e un vettore di variabili $\underline{x} = (x_1, ..., x_n)$, il gradiente ∇ sarà il vettore delle derivate parziali di ciascuna $f(x_i)$ $(\nabla f(\underline{x}) = (\frac{\partial f(\underline{x})}{\partial x_1}), ..., \frac{\partial f(\underline{x})}{\partial x_n}))$. Per lo studio delle derivate seconde parziali si impiegherà la *matrice Hessiana*, la quale avrà dimensione n^2:

$$\nabla^2 f = H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{bmatrix}$$

Tutti i metodi multivariati di risoluzione per NLP seguono un funzionamento di base comune: dato un punto iniziale $\mathbf{x}_k$, un vettore di direzione $\mathbf{d}_k$ e uno scalare

a_k (denominato *step size*), si ottimizza iterativamente $\mathbf{x}_{k+1} = \mathbf{x}_k + a_k\mathbf{d}_k$, con k numero iterazione. Come vettore di direzione $\mathbf{d}_k$ si utilizza il gradiente (in caso di massimizzazione) o il negativo del gradiente (in caso di minimizzazione), ovvero $\mathbf{x}_{k+1} = \mathbf{x}_k \pm a_k\nabla f(\mathbf{x}_k)$.

Definito il metodo si può passare alla specificazione dello *step size* a_k; un possibile metodo passa per l'ottimizzazione della funzione $f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k + a_k\nabla f(\mathbf{x}_k))$, della quale si calcola e si annulla la derivata parziale rispetto a a_k; a questo punto si risolve la risultante per a_k.

Gli step dell'algoritmo basato sul *gradient method* sono i seguenti:

1. Scegliere un punto iniziale $\mathbf{x_0}$; la scelta è molto importante poiché se si seleziona un punto vicino all'ottimo globale si può evitare l'individuazione di un ottimo locale.

2. Si calcola il gradiente $\nabla f(\mathbf{x}_k)$ con k iterazione.

3. Si ricava il vettore direzionale $\mathbf{d}_k = \pm\nabla f(\mathbf{x}_k)$.

4. Si calcola il successivo $\mathbf{x}_{k+1} = \mathbf{x}_k \pm a_k\nabla f(\mathbf{x}_k)$

5. Si applica un'ottimizzazione per determinare a_k.

6. Si valuta la convergenza utilizzando una certa tolleranza ϵ e valutando $|f(\mathbf{x}_{k+1} - f(\mathbf{x}_k)| < \epsilon$ oppure $||\nabla f(\mathbf{x}_{k+1})|| < \epsilon$

4.4 NLP multivariati: *Newton's Method*

Come nel caso univariato si utilizza un'approssimazione quadratica basata sull'espansione della serie di Taylor della funzione f, e si trova la soluzione ottimizzando tale approssimazione. L'espansione di Taylor avrà forma $f(\mathbf{x}_k + \triangle\mathbf{x}) \approx f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)\triangle\mathbf{x} + \triangle\mathbf{x}\mathbf{H}(\mathbf{x}_k)\triangle\mathbf{x}$ (dove $\mathbf{x}$ rappresenta un vettore di variabili e $\triangle\mathbf{x}$ un suo spostamento). Essendo $\mathbf{x}_k$ il punto in cui ci si trova all'iterazione k sarà elemento noto, insieme a $\nabla f(\mathbf{x}_k)$ e a $\mathbf{H}(\mathbf{x}_k)$; bisogna tuttavia determinare $f(\mathbf{x}_k + \triangle\mathbf{x})$, risolvendo $\dfrac{\partial f(\mathbf{x}_k + \triangle\mathbf{x})}{\partial\triangle\mathbf{x}} = 0$ da cui si ricava la relazione $\triangle\mathbf{x} = -\mathbf{H}(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$.

Le stime alle varie iterazioni avranno quindi forma $\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$; in caso la matrice Hessiana sia definita positiva ($\mathbf{x}'\mathbf{H}\mathbf{x}$ è positiva per ogni $\mathbf{x} \neq 0$), sarà necessaria una sola iterazione affinché l'algoritmo raggiunga il punto di ottimo, qualsiasi sia il punto di partenza. L'utilizzo della matrice Hessiana consente di non distinguere tra problema di massimizzazione e di minimizzazione; infatti se la componente del secondo ordine sarà convessa minimizzeremo, in caso contrario massimizzeremo.

Possiamo riassumere il funzionamento del metodo di Newton multivariato come segue:

1. Si sceglie un punto di partenza $\mathbf{x}_k$ ($k = 0$).
2. Si calcolano $\nabla f(\mathbf{x}_k)$ e $\mathbf{H}(\mathbf{x}_k)$.

3. Si calcola la stima all'iterazione successiva utilizzando la forma

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$$

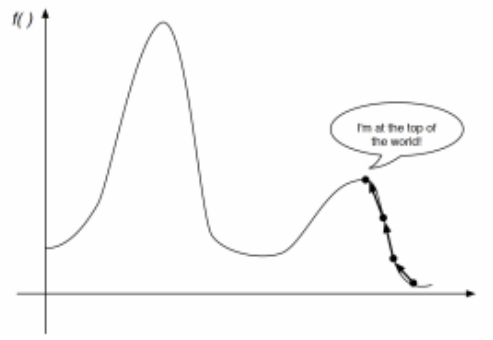
4. Si valuta la convergenza secondo un determinato criterio; in caso di convergenza si ha la soluzione ottimale.

Rispetto al *gradient method* il metodo di Newton risulta più elaborato, in quanto utilizza sia il gradiente si la matrice Hessiana; tale caratteristica consente di ridurre il numero di iterazioni necessarie alla convergenza, ma richiede una maggiore capacità in termini computazionali, soprattutto in caso di funzioni molto complesse.

Meta-heuristics

5.1 *Metaheuristics Global Optimization*

Le metaeuristiche sono procedure di alto livello responsabili del coordinamento di algoritmi di ottimizzazione locale più semplici e/o regole euristiche; lo scopo principale consiste nell'approssimare soluzioni per problemi computazionalmente complessi e onerosi (come problemi di ottimizzazione combinatoria o di ottimizzazione globale). Gli algoritmi iterativi fin'ora presentati hanno una criticità comune: poiché il loro funzionamento si basa su un processo iterativo che stima il punto ottimale con una serie di spostamenti da un punto iniziale, la scelta di tale punto può portare ad individuare minimi locali. Nessuno dei metodi prima presentati è in grado di discernere tra un punto di ottimo globale e uno locale, di conseguenza non vi è alcuna garanzia dell'individuazione di una soluzione ottima globale. Considerando un metodo classico di ottimizzazione come l'algoritmo del Gradiente, possiamo vedere come in questo caso viene individuato come ottimo globale un punto che non lo è; infatti, il metodo del Gradiente, ma anche quello di Newton, sono algoritmi che ricercano i punti di massimo e minimo, ma non ci danno nessuna garanzia che questi siano ottimi globali.



Uno dei metodi per risolvere il problema degli ottimi locali consiste nell'aumentare lo *step size*, così facendo si aumenta lo spazio in cui si può muovere l'algoritmo, con la possibilità che raggiunga i "picchi" successivi della funzione. Un'alternativa è invece mettere l'algoritmo del Gradiente all'interno di un ampio ciclo, nel quale viene fatto variare lo *starting point*, in maniera tale da esplorare la funzione su tutta la *feasible region*; successivamente vengono confrontati i risultati e stimato l'ottimo globale.

La maggior parte delle tecniche classiche per la soluzione di problemi NLP si basa su assunzioni teoriche rigide: un primo esempio riguarda la derivata prima della funzione nel punto ottimale, la quale deve annullarsi. E' chiaro che in una funzione matematica molto regolare (continua, derivabile, nota, concava/convessa) tale ipotesi non risulta inverosimile, tuttavia nel caso di problemi reali (in cui le funzioni sono spesso ignote/approssimate, discontinue, non derivabili...) potrebbe risultare molto stringente. Per ovviare a tali restrizioni vengono impiegate le **Meta-Euristiche**, le quali risultano metodi molto più generali in quanto necessitano di un numero limitato di assunzioni le quali, in certi casi, potrebbero anche non essere necessarie. Per tale peculiarità le Meta-Euristiche spesso risultano più efficienti dei metodi classici, soprattutto in presenza di funzioni ignote, non regolari e/o complesse.

L'impiego di Meta-Euristiche garantisce lo svolgimento di **3 fasi principali**:

- Fornisce in via preliminare una o più soluzioni candidate ad ottimo globale, durante il processo denominato **initialization procedure**

- Nel processo di **assessment procedure** verifica la qualità di ciascun candidato.

- Vi è infine la **modification procedure**, durante la quale vengono create delle copie dei candidati; a tali copie vengono successivamente applicati dei rumori, i quali generano casualmente nuovi candidati diversi da quelli di partenza. In aggiunta alle precedenti vi è la **selection procedure**, mediante il quale l'algoritmo seleziona i candidati da tenere e quelli da scartare, sulla base di determinati criteri (come ad esempio dei test).

Le Meta-Euristiche fanno parte di una più ampia famiglia di metodi di ottimizzazione, denominati **Stochastic Optimization methods**: la peculiarità di tali metodi viene catturata dal nome stesso, in quanto sono caratterizzati dalla presenza di un certo livello di casualità/incertezza. Le Meta-Euristiche sono tuttavia i metodi stocastici più generali e come conseguenza sono applicabili ad un grande numero di problemi di ottimizzazione, soprattutto in caso essi siano complessi o si abbiano a disposizione pochi strumenti e/o informazioni.

5.2 *Metaheuristics Stochastic Optimization*

Un metodo che ha un funzionamento simile a quello appena presentato è l'*Hill Climbing*, il quale risulta iterativo e svolge le 3 fasi dei metodi stocastici.

Il funzionamento è il seguente:

1. Viene selezionato una soluzione preliminare (*initialization procedure*), e successivamente vengono ripetuti i seguenti punti.
2. La soluzione preliminare S viene copiata e le viene applicato un rumore (*modification procedure*), ottenendo così la nuova candidata R
3. Viene impiegato un indicatore di qualità (*assessment procedure*) per valutare quale dei due candidati sia migliore.
4. I caso la qualità di R sia migliore della qualità di S, sostituiremo il primo al secondo.
5. Si ripetono i punti 1, 2, 3 fino a convergenza o fino al soddisfacimento di un determinato *stopping criterion*.
6. Si torna al punto 1.

Tale metodo ha un funzionamento simile al *gradient ascend* precedentemente presentato, tuttavia non richiede una forte conoscenza del gradiente, né per ciò che concerne l'intensità, né per quanto riguarda la direzione; semplicemente vengono testati iterativamente nuovi candidati e trovati i migliori.

Il processo può tuttavia essere molto lungo, poiché ad ogni iterazione vi sono buone probabilità di identificare candidati peggiori rispetto a quello di partenza, rallentando significativamente la convergenza dell'algoritmo.

Steepest Ascent Hill-Climbing

Per risolvere tale criticità vi è tuttavia la possibilità di rendere l'algoritmo più complesso richiedendo che in fase di *modification* vengano create e disturbate $n > 1$ copie, e successivamente selezionata la migliore tra esse; l'algoritmo così definito prende il nome di **Steepest Ascent Hill-Climbing**, il quale ha il seguente funzionamento:

Steepest Ascent Hill-Climbing

```
1: n number of tweaks desired to sample the gradient
2: S ← some initial candidate solution .
3: repeat
4: R ← Tweak(Copy(S)) .
5: for n − 1 times do
6:         W ← Tweak(Copy(S))
7:         if Quality(W) > Quality(R) then
8:         R ← W
9:         if Quality(R) > Quality(S) then.  S ← R
10: until S is the ideal solution or we have run out of time
11: return S
```

Una variazione famosa dello **Steepest Ascent Hill-Climbing** è quella con sostituzione che viene chiamata **Steepest Ascent Hill-Climbing with Replacement**, la differenza è che questo algoritmo sostituisce S direttamente con R, vedi la l'istruzione numero 10 dell'algoritmo nell'immagine, e così facendo abbiamo il rischio di perdere la nostra soluzione migliore; inoltre utilizziamo un'altra variabile *Best* in cui allochiamo la soluzione trovata finora.

```
1: n number of tweaks desired to sample the gradient
2: S ← some initial candidate solution .
3: Best ← S
4: repeat
5: R ← Tweak(Copy(S)) .
6: for n − 1 times do
7:         W ← Tweak(Copy(S))
8:         if Quality(W) > Quality(R) then
9:                 R ← W
10:        S ← R
11:        if Quality(R) > Quality(Best) then. Best ← R
12: until Best is the ideal solution or we have run out of time
13: return Best
```

La rappresentazione delle soluzioni può avvenire in diversi modi, tra i quali troviamo vettori, liste di oggetti, alberi, grafi e collezione di oggetti o una combinazione.

Exploration vs Explotation è la vera domanda che bisogna affrontare quando si progetta un algoritmo di ottimizzazione stocastico o meta-hururistics. Una parte fondamentale per la definizione dell'algoritmo è la selezione di un livello di rumore adeguato. In caso il rumore sia troppo piccolo sarà difficile che i candidati passino da un 'collina' all'altra, rischiando così di identificare ottimi locali; d'altro canto un rumore troppo elevato provocherà un rimbalzo elevato e quando sarà vicino alla cime di una collina avrà difficoltà a convergere verso la cima, poiché la maggior parte delle sue mosse sarà così grande da superare il picco. Quindi bisogna trovare un *trade off* tra ***exploration*** della nostra *feasible region* e ***exploitation*** che consiste nello sfruttamento della posizione attuale del punto scelto per andare su per la collina.

Gli algoritmi che apportano miglioramenti in gran parte locali sfruttano il gradiente locale (exploitation), mentre gli algoritmi che vagano per lo più in modo casuale esplorano lo spazio (exploration). Regola Generale: gli algoritmi *exploitation* sono veloci e vengono utilizzati quando la *feasible region* è più complessa, ovvero ha molti 'picchi'; gli algoritmi *exploration* sono più lenti e vengono utilizzati quando la *feasible region* è meno complessa, ovvero ha pochi 'picchi'.

Un algoritmo di ottimizzazione globale è quello che, se lo eseguiamo abbastanza a lungo, alla fine troverà l'ottimale globale. Esistono molti modi per costruire un algoritmo di ottimizzazione globale.

Il più semplice è: **Ricerca casuale**. La ricerca casuale è l'estremo nell'esplorazione (e nell'ottimizzazione globale); al contrario, Hill-Climbing può essere visto come l'estremo nello sfruttamento (e nell'ottimizzazione locale). Esistono modi per ottenere uno sfruttamento ragionevole e avere ancora un algoritmo globale: come ad esempio Hill-Climbing con riavvii casuali che è una combinazione dei due approcci (Random search e Hill-Climbing).

5.3 *Metaheuristics*

I metodi euristici sono procedure che vengono utilizzate per trovare l'ottimo in modo più efficiente, tuttavia esso non garantisce il raggiungimento della soluzione ottimale, ma di una soluzione molto buona. L'euristica può essere raggruppata in due grandi classi, ovvero metodi di costruzione e metodi di ricerca locale. In entrambe i casi la *feasible region* non è esplorata in modo sistematico, infatti i metodi euristici sono caratterizzati dal modo in cui viene attraversato il dominio della soluzione. Ciò si ottiene trattando l'accuratezza, la completezza, l'ottimalità, la precisione e la velocità. L'euristica può essere considerata come una scorciatoia per raggiungere una buona soluzione in tempi minori.

Cos'è la meta-euristica? Meta significa un'astrazione da un altro concetto, quindi qualcosa che va oltre, a un livello superiore e viene utilizzato per completare o aggiungere. Viene usato ad esempio quando parliamo di "meta-dati" sono "dati sui dati". Quindi la meta-euristica è una procedura di alto livello progettata per trovare, generare o selezionare un algoritmo di ricerca euristica o parziale che fornisca una soluzione sufficientemente buona a un problema di ottimizzazione, soprattutto se abbiamo una capacità computazionale limitata e/o limitata conoscenza del problema. La metaeuristica è un campo relativamente nuovo (iniziato negli anni '80 o giù di lì) e diventa possibile perché possiamo ora offrono enormi quantità di capacità di calcolo. Sono ispirati dall'intelligenza artificiale, piuttosto che dalla "matematica pura". A volte vi è una mancanza di rigore teorico in questi algoritmi (nessuna prova, teorema, ecc.). I metodi meta-euristici in sostanza campionano una serie di soluzioni che è troppo grande per essere campionata completamente. Questi metodi solitamente incorporano meccanismi per evitare di rimanere intrappolati in aree ristrette dello spazio di ricerca, ciò si ottiene esplorando le regioni promettenti e trovando un *trade-off* tra esplorazione e sfruttamento. Questi metodi sono ispirati ai processi naturali, quindi non sono deterministici; inoltre non sono specifici di una determinata tipologia di problemi ma sono generali. I metodi metaeuristici sono in grado di utilizzare una qualche forma di memoria che gli permette di guidare meglio la ricerca, mantenendo le scelte o i passaggi precedenti al fine di evitare di far rimanere il ciclo tra un numero chiuso di soluzioni. A differenza dei metodi esatti, sono agnostici rispetto allo spazio fattibile inesplorato; non hanno una metrica di "bontà" di soluzione così spesso la meta-euristica si interrompe a causa di un limite di tempo esterno o per qualsiasi limite di iterazione e non perché sa di aver raggiunto la soluzione ottimale. La meta-euristica non si basa su alcuni modelli algebrici a differenza dei metodi esatti. Ma a causa della loro efficacia nell'esplorazione dello spazio di ricerca e al fine di trovare un'ottima soluzione, vengono spesso utilizzati insieme a un metodo esatto, quindi ad esempio possiamo usare la meta-euristica per trovare una buona soluzione e quindi sfruttare la buona soluzione come punto di partenza per un metodo esatto al fine di trovare una soluzione migliore o almeno una soluzione ottimale locale.

Due sono le tipologie di algoritmo metaeuristico:

- ***Trajectory-based S-metaheuristics*** è un metodo iterativo, il quale prende in input una soluzione s_0 esplora i punti più vicini alla soluzione e li seleziona

come candidati, il quale diventerà la nuova soluzione se nel punto trovato la soluzione è migliore di quella che abbiamo; questa soluzione viene utilizzata come punto di partenza per la prossima iterazione. Se prendiamo in considerazione l'esempio sottostante possiamo vedere che l'algoritmo parte dal punto verde, il quale viene preso come soluzione di partenza, viene definito il vicinato, ovvero i punti vicini alla soluzione, successivamente viene selezionata tra il vicinato una nuova soluzione (punto giallo) che verrà rimpiazzata alla soluzione corrente; il ciclo termina quando viene rispettato lo *stopping criteria*.

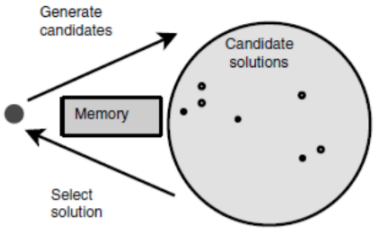
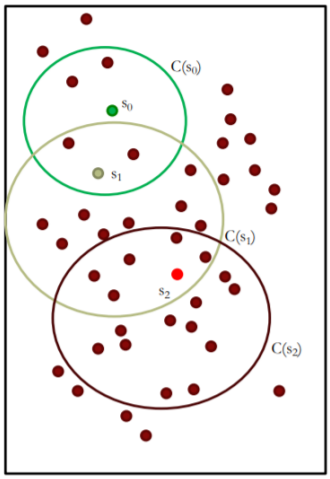


FIGURE 2.1 Main principles of single-based metaheuristics.

Algorithm 2.1 High-level template of S-metaheuristics.

Input: Initial solution s_0.
$t = 0$;
Repeat
  /* Generate candidate solutions (partial or complete neighborhood) from s_t */
  Generate($C(s_t)$) ;
  /* Select a solution from $C(s)$ to replace the current solution s_t */
  s_{t+1} = Select($C(s_t)$) ;
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution found.



Stop because of no improvement in region $C(s_2)$

Può accadere che durante la procedura di ricerca casuale abbiamo perso il punto ottimale e finiamo in un altro. Quindi la traiettoria giusta sarebbe un'altra e non quella che abbiamo noi; non abbiamo alcuna sicurezza sul fatto che l'algoritmo converga al punto ottimale.

- ***Population-based (P-metaheuristics)*** è un metodo iterativo, il quale ad ogni iterazione genera una popolazione di possibili soluzioni al nostro problema, sotto il vincolo che la numerosità della popolazione rimanga fissa ad ogni iterazione. L'algoritmo inizia selezionando alcuni punti che andranno a formare la popolazione di partenza (P1), alla prima iterazione verrà costituita una nuova popolazione (P2); per far sì che l'algoritmo sia efficiente, viene fissata una numerosità che deve rimanere fissa ad ogni iterazione, a questo proposito quindi verranno selezionati alcuni elementi di P1 e alcuni di P2 che andranno a generare la popolazione che verrà utilizzata alla seconda iterazione come popolazione di partenza. Il ciclo termina quando viene soddisfatto il criterio di stop. *stopping criteria*.

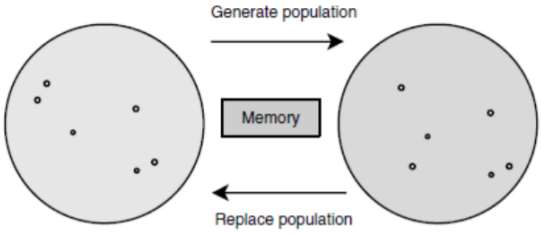
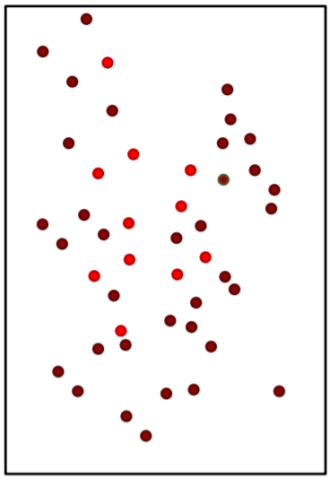


FIGURE 3.1 Main principles of P-metaheuristics.

Algorithm 3.1 High-level template of P-metaheuristics.

$P = P_0$; /* Generation of the initial population */
$t = 0$;
Repeat
  Generate(P'_t); /* Generation a new population */
  P_{t+1} = Select-Population($P_t \cup P'_t$); /* Select new population */
  $t = t + 1$;
Until Stopping criteria satisfied
Output: Best solution(s) found.



Again, optimum may or may not have been sampled

- Typically, the incumbent always remains in the population, so need only focus on last generation

Anche in questo caso non abbiamo la sicurezza che la soluzione trovata sia quella ottimale, poiché essa dipende dalla popolazione selezionata.

Esistono molti algoritmi Meta-euristici, tra i più utilizzati ci sono: *Genetic Algorithms, Simulated Annealing e Tabu Search.*

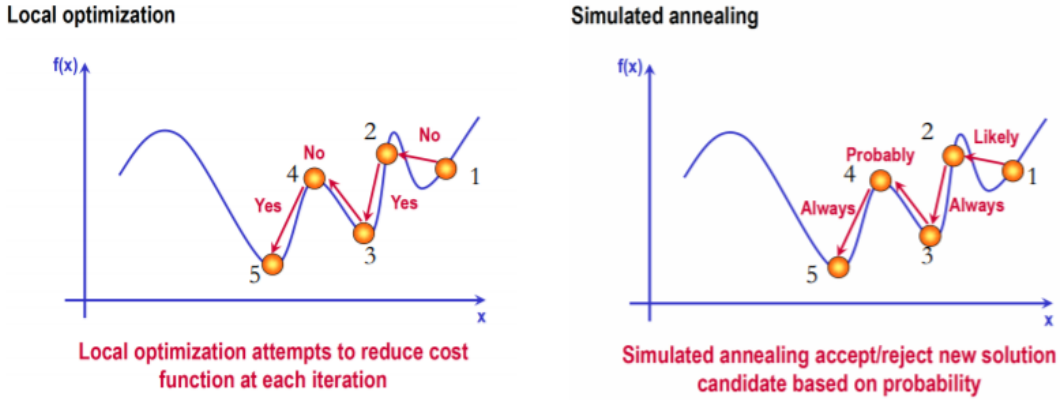
5.4 *Simulated Annealing*

Simulated Annealing è un metodo di ottimizzazione vincolato, che può essere applicato sia ad una funzione continua che ad una discreta. Il *Simulated Annealing* inizia, come tutti i metodi di ottimizzazione numerica, da un punto iniziale, per poi considerare ripetutamente vari nuovi punti di soluzione. La peculiarità della ricottura simulata è che accetta o rifiuta alcune di queste soluzioni candidate con un determinato criterio casuale. *Simulated Annealing* è stato migliorato per convergere alla soluzione ottimale. *Simulated Annealing* è stata sviluppata da vari ricercatori a metà degli anni ottanta, ma la versione originale è arrivata da un progetto di Metropolis e altri gruppi sviluppato nel 1953. Si basa su "somiglianze" e "analogie con il modo in cui le leghe riescono a trovare un livello di energia minimo quasi globale quando vengono raffreddati lentamente", infatti è simile alla ricottura nei metalli, in cui si colpisce il metano allo stato solido ad alta temperatura, quindi si raffredda molto lentamente in base a un programma specifico e, se la temperatura di contatto è sufficientemente alta da assicurare uno stato casuale e il processo di raffreddamento è abbastanza basso da garantire un equilibrio termico, gli atomi verranno posizionati in uno schema che corrisponde al minimo globale di energia nel cristallo perfetto. Quindi le idee che stanno dietro a questo metodo provengono da questo processo. Il *Simulated Annealing* è un metodo di ottimizzazione stocastica, infatti utilizza la casualità in modo strategico per esplorare lo spazio delle soluzioni,

per sfuggire ai minimi locali e ciò aumenta le possibilità di ricerca vicino all'ottimale globale.

Local Optimization vs Simulated Annealing

Quando consideriamo algoritmi di ottimizzazione locale come il *Gradient method* o il *Newton Method*, abbiamo che essi partono dal punto iniziale e attraverso delle ripetizioni considerano vari nuovi punti di soluzione e, molto importante, riducono la funzione di costo ad ogni iterazione e alla fine convergono in un soluzione ottimale. Il *Simulated Annealing* è molto simile ad un'ottimizzazione locale come struttura, infatti parte dal punto iniziale e considera ripetutamente vari nuovi punti di soluzione ma, invece di ridurre la funzione di costo ad ogni iterazione, utilizza la probabilità per accettare o rifiutare una nuova soluzione, anche se questa soluzione sta peggiorando la funzione di costo, ed eventualmente in fine converge nella soluzione ottimale. Supponiamo quindi che vogliamo minimizzare questa funzione, vogliamo trovare il minimo globale, ovvero il punto 5. Nell'immagine a sinistra supponiamo che stiamo usando un algoritmo di ottimizzazione locale, partiamo dal punto 1, generiamo il punto 2 ma non accettiamo il punto 2, perché non migliora la funzione obiettivo perché sta aumentando la funzione obiettivo, generiamo il punto 3; questa mossa sarà accettata, la mossa da 3 a 4 non sarà accettata e lo spostamento da 4 a 5 sarà accettato. Quindi, ciò non ci dà la certezze che sarà molto facile sfuggire ad esempio dal punto 3 perché vogliamo accettare di andare al punto 4. Mentre il *Simulated Annealing* non rifiuta il passaggio da 1 a 2 ma accetterà 2 con una data probabilità (*'likely'*), successivamente accetterà lo spostamento da 2 a 3 perché sta migliorando la funzione obiettivo (*'always'*), in seguito accetterà lo spostamento da 3 a 4 (*'probably'*) e pertanto, se accetta questa mossa, arriverà al punto 5 e converge verso l'ottimale globale.



La particolarità del *Simulated Annealing* è che controlla in modo intelligente il grado di casualità aggiunto ai metodi di ricerca stocastica, quindi la casualità non è solo casualità ma è casualità controllata, infatti inizialmente la casualità aggiunta alle valutazioni delle funzioni è grande, quindi la 'temperatura' è grande, essa viene lentamente abbassato secondo un "piano di ricottura" prestabilito; quindi all'inizio *Simulated Annealing* è più esplorativa che di sfruttamento, perché esplora maggiormente l'area delle soluzioni; mentre avanza con l'iterazione e quando si raggiunge una buona area, ovvero l'area intorno all'ottimale globale, la temperatura si raffredda e il comportamento della ricottura simulata diventa più sfruttatore.

Andiamo ora a vedere il funzionamento dell'algoritmo. Nello *step 1* vi è l'assegnazione del punto d'inizio $X = X_0$ e del contatore $K = 0$; nello *step 2* abbiamo la valutazione della funzione di costo $F = f(X_k)$, in cui il valore che gli viene assegnato e quello della funzione nel punto di partenza X_0; nello *step 3* ci muoviamo casualmente dal punto X_k verso la nuova soluzione X_{k+1}; nello *step 4* e nello *step 5* viene confrontato il valore della funzione obiettivo nella nuova soluzione X_{k+1} con quello della soluzione iniziale F, se $f(X_{k+1}) < F$ allora accettiamo la nuova soluzione e andiamo a sostituire $X = X_{k+1}$ e $F = f(X_{k+1})$, se $f(X_{k+1}) \geq F$ e inoltre se estraiamo un numero casuale con la funzione $rand()$ e questo risulta essere minore di ε ,quindi $rand() < \varepsilon$, allora accettiamo la nuova soluzione con una data probabilità e andiamo a sostituire $X = X_{k+1}$ e $F = f(X_{k+1})$ con una data probabilità ε, altrimenti rifiutiamo; infine nello *step 6* andiamo ad incrementare $K = K + 1$ e iteriamo partendo dallo *step 2*.

Step 1: start from an initial point $X = X_0$ & $K = 0$
Step 2: evaluate cost function $F = f(X_K)$
Step 3: randomly move from X_K to a new solution X_{K+1}
Step 4: if $f(X_{K+1}) < F$, then
Accept new solution
$X = X_{K+1}$ & $F = f(X_{K+1})$
End if
Step 5: if $f(X_{K+1}) \geq F$, then
Accept new solution with certain probability
$X = X_{K+1}$ & $F = f(X_{K+1})$ iff $rand() < \varepsilon$
End if
Step 6: $K = K + 1$ & go to Step 2

Se ci soffermiamo sullo *step 5* possiamo vedere come la probabilità ε sia ignota; due sono le possibilità:

- assegniamo ad ε una costante;
- facciamo decrescere ε ad ogni iterazione

Se scegliamo la seconda strada, un criterio molto utilizzato per decrementare la probabilità è quello di *Metropolis* che consiste nell'assegnare a ε la distribuzione di *Boltzmann*:

$$\varepsilon = exp\left[-\frac{f(X_{k+1} - F)}{T_{k+1}}\right]$$

dove T_{k+1} è la "temperatura", un parametro che diminuisce gradualmente
(Es: $T_{k+1} = \alpha * T_k$ dove $0 < \alpha < 1$).

Così facendo lo *step 5* diventa:

Step 5: if $f(X_{K+1}) \geq F$, then
Accept new solution with certain probability
$X = X_{K+1}$ & $F = f(X_{K+1})$ iff rand() <= $exp\left[-\frac{f(X_{k+1})-F}{T_{k+1}}\right]$
End if

Così che per alti livelli di temperatura si tenta di accettare tutte le nuove soluzioni anche se $[f(X_{k+1} - F]$ è grande, mentre per bassi livelli di temperatura si accettano solo le nuove soluzioni in cui $[f(X_{k+1} - F]$ è piccolo.

Vi sono altri settaggi per la temperatura quali:

Temperature Annealing schedules

- Logarithmic annealing schedule

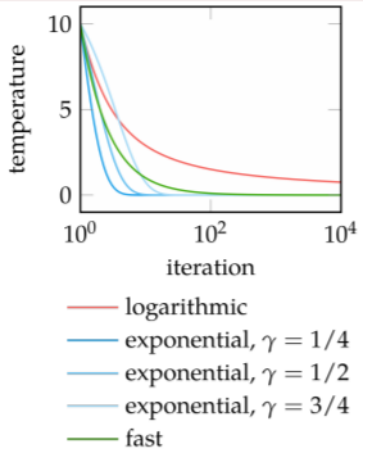
$$t^{(k)} = t^{(1)}\frac{\ln(2)}{\ln(k+1)}$$

- Exponential annealing schedule

$$t^{(k+1)} = \gamma t^{(k)}$$

- Fast annealing

$$t^{(k)} = \frac{t^{(1)}}{k}$$



Varie decisioni devono essere prese quando si applica SA:

- Rappresentazione e generazione della soluzione
- Temperatura iniziale T?
- Programma della temperatura?
- Quante iterazioni con lo stesso valore?
- Stop criteri?

Questioni pratiche

- La temperatura iniziale deve essere tale che inizialmente si accetti circa il 50% delle soluzioni in peggioramento
- Il programma di raffreddamento dovrebbe essere lento, ad esempio il 10%
- La temperatura finale dovrebbe essere tale da non accettare soluzioni in peggioramento, vale a dire $T \approx 0$

Nota: si può dimostrare teoricamente che SA converge asintoticamente all'ottimale globale. In pratica, tuttavia, la sua velocità di convergenza è fortemente influenzata dal programma di raffreddamento

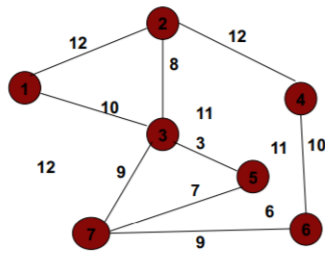
SA non garantisce l'ottimale globale, tuttavia, cerca di evitare un gran numero di minimi locali; spesso fornisce una soluzione migliore dell'ottimizzazione locale. SA non è deterministica, l'accettazione o il rifiuto di una nuova soluzione è casuale ed è possibile ottenere risposte diverse da più esecuzioni. SA è più costosa dell'ottimizzazione locale, questo è il prezzo che devi pagare per ottenere una soluzione ottimale migliore.

5.4.1 *Simulated Annealing for TSP*

Abbiamo visto fin qui l'applicazione dell'algoritmo SA per i problemi di ottimizzazione non lineare, ora andiamo a vedere il SA applicato al problema del commesso viaggiatore (*Travelling Salesman Problem*), questo è un problema di calcolo combinatorio su grafo, ovvero: dato un insieme di città, e note le distanze tra ciascuna coppia di esse, trovare il tragitto di minima percorrenza che un commesso viaggiatore deve seguire per visitare tutte le città una ed una sola volta e ritornare alla città di partenza. In questo caso il nostro obiettivo è implementare e adattare lo schema SA a tale problema. Prima che la soluzione era costituita da un solo punto, ma ovviamente in questo caso non è così semplice, poiché abbiamo che la soluzione per TSP è una sequenza di nodi. Possiamo rappresentare questa soluzione come un vettore, come un elenco di numeri interi associati ai nodi del grafo; la nostra soluzione sarà la sequenza del numero di nodi visitati in quella particolare soluzione. Ad esempio se abbiamo un grafo con 6 nodi la soluzione possibile è data da una sequenza di 7 nodi, in cui il primo è uguale all'ultimo. Ogni soluzione ammissibile è eleggibile come soluzione iniziale; le soluzioni vicine sono ottenute scambiando 2 o più interi all'interno della sequenza. Il meccanismo rimane quello di selezionare in modo casuale, ovvero: i punti, iniziale e final, di una sottosequenza del vettore della soluzione vengono selezionati in modo casuale e il successivo viene scambiato (purché sia ammissibile la soluzione del nuovo candidato). Viene fissata nuovamente la 'temperatura di cottura' e si itera fin quando non viene rispettato il criterio di stop.

Simulated Annealing: results

Iterazione	T	Soluzione	f(x)	
0		1-2-3-4-5-6-7-1	69	
1	13.8	1-3-2-4-5-6-7-1	68	
2	13.8	1-2-3-4-5-6-7-1	69	
3	13.8	1-3-2-4-5-6-7-1	68	
4	13.8	1-3-2-4-6-5-7-1	65	
5	13.8	1-2-3-4-6-5-7-1	66	
6	6.9	1-2-3-4-5-6-7-1	69	
7	6.9	1-3-2-4-5-6-7-1	68	
8	6.9	1-2-3-4-5-6-7-1	69	
...	...	...	...	
14	3.43	1-3-5-7-6-4-2-1	63	Min
15	3.43	1-3-7-5-6-4-2-1	66	
16	1.725	1-3-5-7-6-4-2-1	63	Min
...	...	...	...	
25	0.8625	1-3-7-5-6-4-2-1	66	



This is a new simulation of the SA starting from the same initial solution 1-2-3-4-5-6-7-1

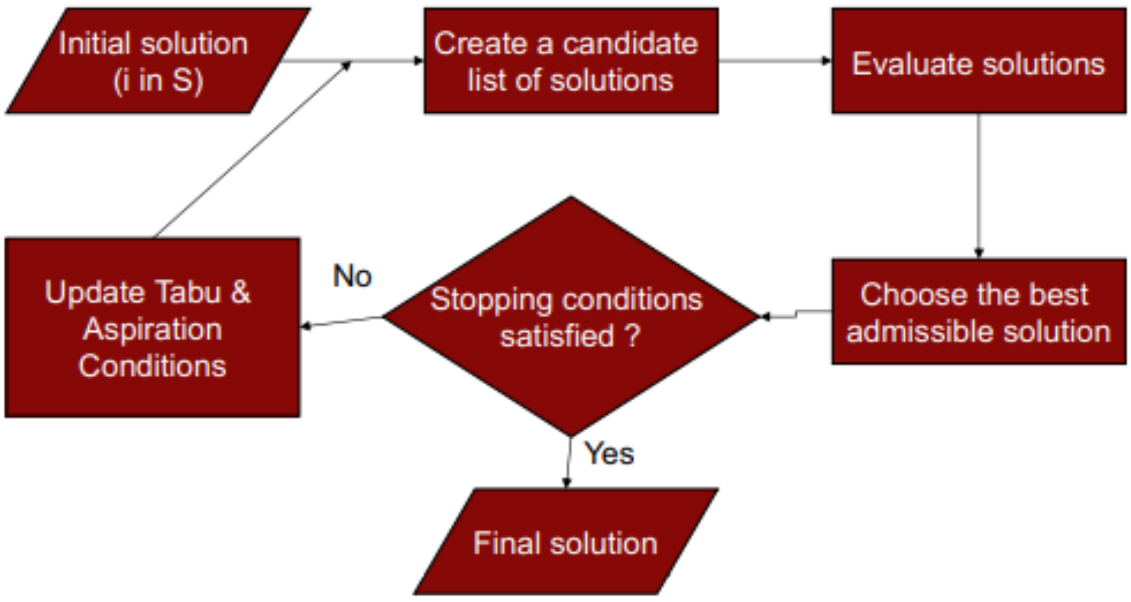
Note that we generate the optimal soltion at iterations 14 and 16

39

5.5 *Tabu Search*

Uno dei concetti fondamentali legato all'algoritmo *Tabu Search* risiede nel nome stesso; con tabu infatti indichiamo un elemento il quale non va considerato poiché dannoso per l'analisi (semanticamente si riferisce ad azioni o comportamenti deprecabili secondo una certa morale). In tale ottica il *Tabu Search* applica una suddivisione dello spazio, volta a guidare l'esplorazione della funzione in determinate direzioni, scartandone altre (ovvero i tabu); per fare ciò è necessario che vi sia un sistema di bonus e penalità volte ad indicare gli spazi in cui deve muoversi l'algoritmo. Per fare ciò necessità di una componente essenziale, la quale risulta invece assente nel *Simulated Annealing*, ovvero una **memoria storica**. Tale componente consente all'algoritmo di esplorare la funzione in determinati spazi, evitando di creare dei *searching loop* su aree già scartate.

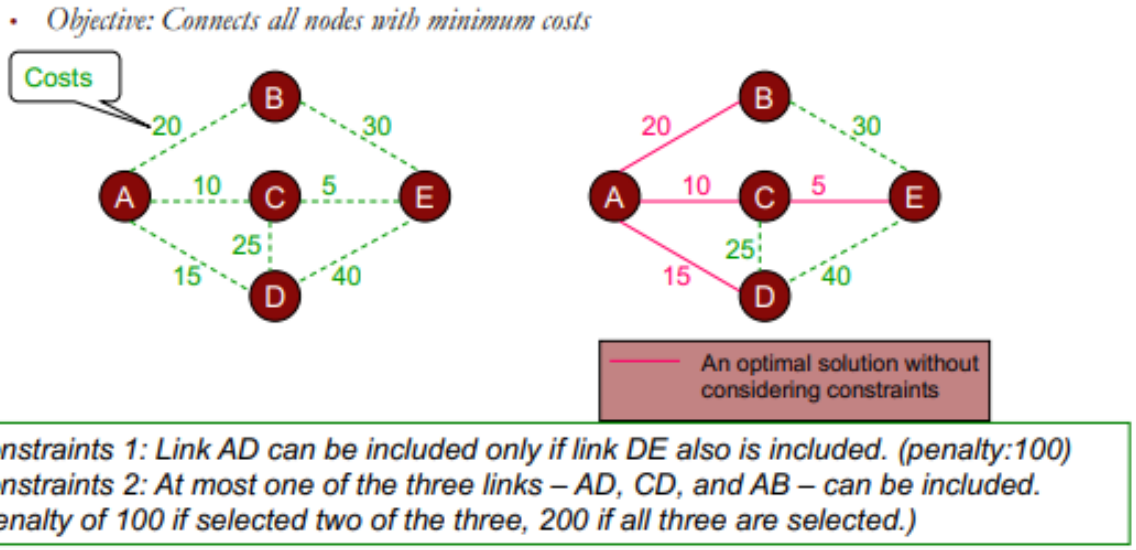
L'algoritmo *Tabu Search* ha molte caratteristiche in comune con il *Simulated Annealing*: nonostante siano entrambi basati su una logica *Local Search* (LS) ammentono degli spostamenti *non-improving*, i quali consentono all'algoritmo di "uscire" dagli ottimi locali. Il TS tuttavia, grazie alla sua memoria storica parziale, permette di evitare effetti di *looping* nelle stesse aree (non garantito dal SA), diversificando in tal modo lo spazio di ricerca. Ad ogni iterazione vengono quindi definite delle **Tabu moves** nell'intorno del punto di partenza, e inserite all'interno della *tabu list*. Dal momento che determinati spostamenti potrebbero risultare convenienti (in termini di miglioramento della soluzione) durante iterazioni successive, il TS consente l'applicazione dell'**aspiration criterion**, il quale consente di sovrascrivere o eliminare degli elementi dalla *tabu list*. Il funzionamento generale dell'algoritmo viene riassunto nel seguente *flowchart*:



Prima di applicare l'algoritmo è necessario definire alcuni parametri la cui scelta, poiché può influenzare molto i risultati, deve essere attentamente ponderata. Tra le principali vi sono:

- **Rappresentazione delle soluzioni e degli intorni** (*neighborhood*).
- **Definizione della Tabu Memory**: quanto deve essere lunga la lista delle *tabu moves* e per quante iterazioni va mantenuta.
- **Selezione della candidate list**: definire il metodo con il quale vengono selezionati i candidati a nuova soluzione durante le iterazioni dell'algoritmo.
- **Definizione della funzione obiettivo** (*evaluation function*), la quale consente di verificare presenza e livello dei miglioramenti tra un'iterazione e l'altra.
- **Definizione delle stopping condition**: si tratta di una fase essenziale, nel quale si definiscono i criteri di stop. Dal momento che la convergenza a ottimo globale non viene garantita dal TS è necessario individuare dei buoni criteri di stopping; alcuni sono:
  - Non vi sono *feasible solutions* nell'intorno della soluzione all'iterazione K (N(i,K+1)=0).
  - L'iterazione K raggiunta super la soglia massima $K_{\max}$.
  - Il numero d'iterazioni senza miglioramenti supera una soglia i.
  - Vi sono alcuni indizi che si sia raggiunto un ottimo globale.

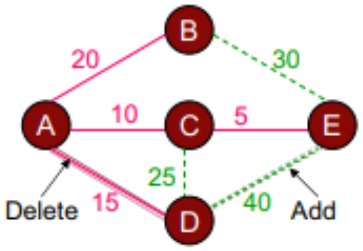
Un'applicazione tipica per gli algoritmi TS riguarda i problemi di *Minimum Spanning Tree* (MSP) in presenza di vincoli (i quali non consentono l'applicazione del metodo descritto nelle precedenti sezioni). Un esempio di MSP con vincoli viene proposto nella seguente immagine:



Constraints 1: Link AD can be included only if link DE also is included. (penalty:100)
Constraints 2: At most one of the three links – AD, CD, and AB – can be included. (Penalty of 100 if selected two of the three, 200 if all three are selected.)

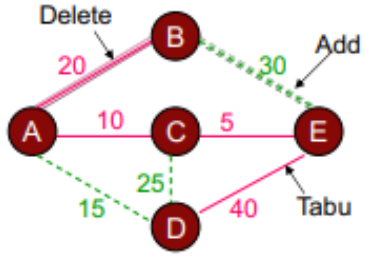
Nelle seguenti immagini vengono mostrati i risultati delle prime 3 iterazioni:

Iteration 1
Cost=50+200 (constraint penalties)



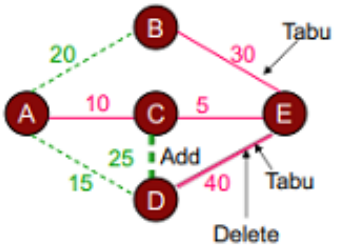
Add	Delete	Cost
BE	CE	75+200=275
BE	AC	70+200=270
BE	AB	60+100=160
CD	AD	60+100=160
CD	AC	65+300=365
DE	CE	85+100=185
DE	AC	80+100=180
DE	AD	**75+0=75**

Tabu list: DE
Iteration 2 Cost=75



Add	Delete	Cost
AD	DE*	Tabu move
AD	CE	85+100=185
AD	AC	80+100=180
BE	CE	100+0=100
BE	AC	95+0=95
BE	AB	**85+0=85**
CD	DE*	60+100=160
CD	CE	95+100=195

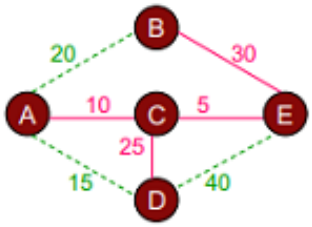
Tabu list: DE & BE
Iteration 3 Cost=85



Add	Delete	Cost
AB	BE*	Tabu move
AB	CE	100+0=100
AB	AC	95+0=95
AD	DE*	60+100=160
AD	CE	95+0=95
AD	AC	90+0=90
CD	DE*	**70+0=70**
CD	CE	105+0=105

A tabu move will be considered only if it would result in a better solution than the best trial solution found previously (Aspiration Condition)

Iteration 4 new cost = 70 Override tabu status



Optimal Solution
Cost = 70
Additional iterations only find inferior solutions

5.6 *Genetic Algorithms*

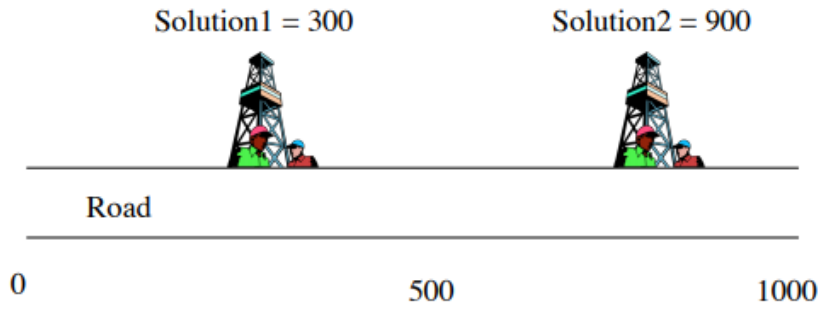
Un ultima tipologia di algoritmi che fanno riferimento ai metodi metaeuristici sono i **Genetic Algorithms** (GA): a differenza degli altri appena presentati (SA e TS), i quali sono metodi *trajectory-based* (basati sulla direzione di spostamento per passare ai candidati successivi), i GA sono metodi **population-based** i quali non seguono logiche di *local-search*. L'idea di base nasce dalla teoria dell'Evoluzione Darwiniana e risulta molto utile per risolvere una grande varietà di problemi, difficilmente trattabili con metodi esatti o altri metodi stocastici meno raffinati. Al fine di comprendere il funzionamento è necessario definire alcuni termini afferenti al campo della genetica e dell'evoluzione, i quali vengono spesso "presi in prestito" nella terminologia dell'algoritmo:

- **Cromosomi**: stringhe di DNA contenute in tutte le cellule viventi.
- **Geni**: blocchi di DNA contenuti nei cromosomi; ciascuno determina alcuni aspetti degli organismi.
- **Genotipo**: collezione di geni
- **Fenotipo**: collezione di aspetti.
- **Mutazioni**: insieme alla ricombinazione dei geni dei genitori determinano le caratteristiche degli organismi.
- **Adattamento** (*fitness*): quanto un organismo può riprodursi prima di morire.

L'idea di base parte dalla generazione casuale di un set di soluzioni, le quali vengono inserite in un ciclo all'interno del quale ciascun elemento viene testato e successivamente copiato, in caso di soluzione ritenuta buona, o rimosso, in caso di cattiva soluzione; le copie delle soluzioni "buone" vengono successivamente perturbate in maniera tale da ottenere delle piccole modifiche. Il ciclo termina quando si incontra una soluzione considerata sufficientemente "buona" da soddisfare determinati *stopping criterion*.

Tipicamente i set di soluzioni vengono codificati come stringhe binarie (per esempio [101001...]): ciascun bit rappresenta alcuni aspetti (**geni**) delle soluzioni (**genotipi**) proposte, alle quali sono associati determinati valori (**fenotipi**). Ciascuna stringa deve essere quindi testata affinché si possa associare un determinato *score* sulla bontà della soluzione.

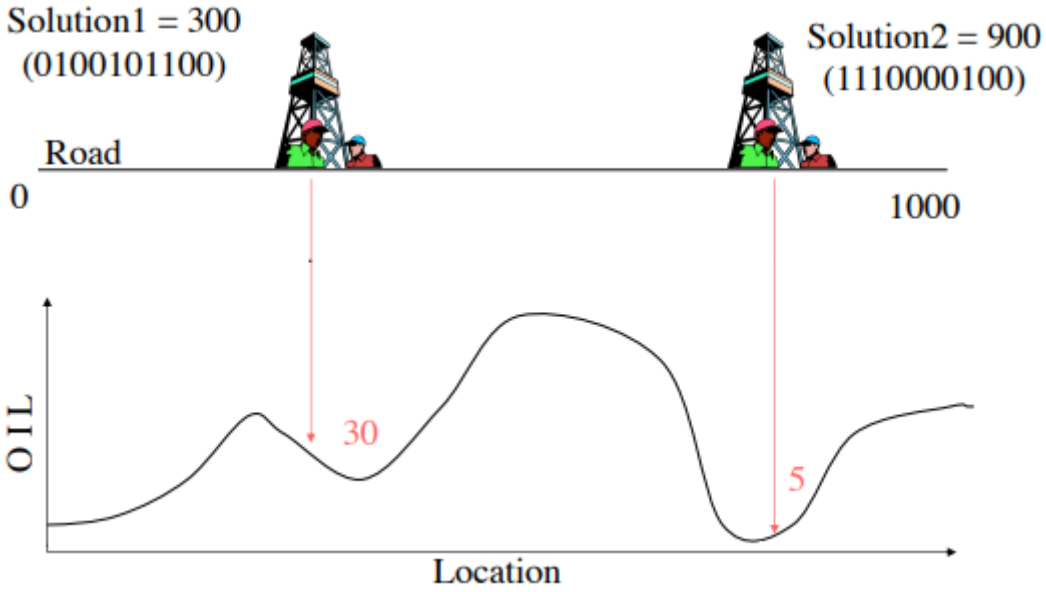
Un esempio di applicazione di un GA viene fornita dal seguente problema: una compagnia petrolifera deve decidere dove posizionare in maniera ottimale una pompa per l'estrazione di greggio su un determinato percorso (per semplicità lungo 1 Km).



Nell'immagine precedente vengono mostrate due possibili soluzioni, una a 300 metri e una a 900, delle quali bisogna tuttavia determinare la bontà. L'insieme di tutte le possibili soluzioni ([0,1000]) viene chiamato *searching space*; per la rappresentazione in stringhe di bit saranno sufficienti 10 bit (come mostrato in tabella).

	512	256	128	64	32	16	8	4	2	1
900	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
300	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**
1023	1	1	1	1	1	1	1	1	1	1

A seconda della posizione scelta per la pompa vi sarà la possibilità di estrarre un certo quantitativo di greggio: lo scopo del problema sarà quindi determinare il punto ottimale in cui sia possibile estrarre la maggiore quantità.



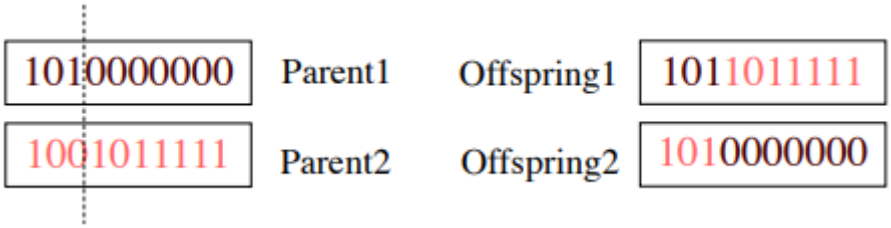
L'*objective function* f viene quindi definita dalla quantità di greggio estraibile nella posizione x (espressa in codice binario); valutando la funzione in una determinata soluzione si stabilisce la bontà della soluzione stessa (*fitness function*).

Anche se nell'esempio viene mostrato il funzionamento del GA nel caso di funzioni univariate, il ragionamento è applicabile anche al caso multivariato, nel quale il *searching space* assumerà la forma di iperpiano.

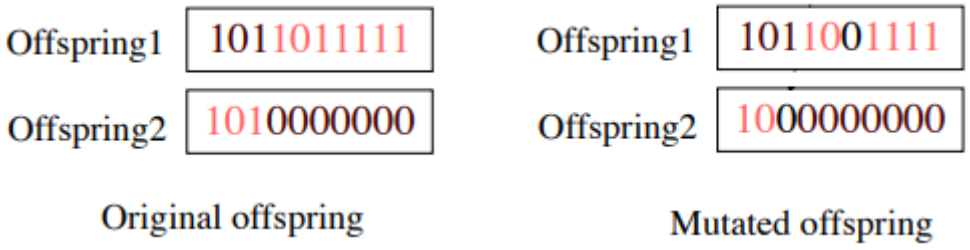
Dal momento che l'algoritmo risulta ancora troppo semplice, si è pensato di inserire un elemento di complessità per garantire una maggiore "mutazione casuale" delle soluzioni; viene quindi introdotto il concetto di **genere** (*sex*), il quale consente di selezionare due "genitori" durante il processo riproduttivo e di combinare i loro geni. Il nuovo funzionamento seguo lo schema precedente, tuttavia vengono selezionate secondo una certa distribuzione di probabilità due soluzioni ritenute "buone"

e successivamente combinate per ottenerne due nuove (*offspring*); ciascun *offspring* può dunque subire mutazioni casuali. Il metodo per determinare la probabilità di selezione è basato sul *fitness* delle soluzioni stesse; tipicamente viene impiegato il **Roulette Wheel**: si inizia addizionando le *fitness* di ciascun cromosoma, si genera casualmente un numero R nel *range* della somma totale, infine si estrae il primo cromosoma che (dopo aver sommato tutti gli altri) restituisce una valore somma uguale o maggiore di R.

Un altro metodo da decidere prima di passare all'applicazione dell'algoritmo è la **tecnica di cross-over**. Tipicamente si applica una **ricombinazione** delle stringhe dei genitori, selezionando una parte dalla prima e una dalla seconda e ricombinandole per comporre gli *offspring*.



Tipicamente per rendere ancora più casuale il processo di riproduzione si applica una *mutazione* agli *offspring* appena generati, cambiando casualmente alcuni geni (in maniera molto lieve per non sconvolgere il *cross-over*)



Per riassumere il **funzionamento dell'algoritmo GA**:

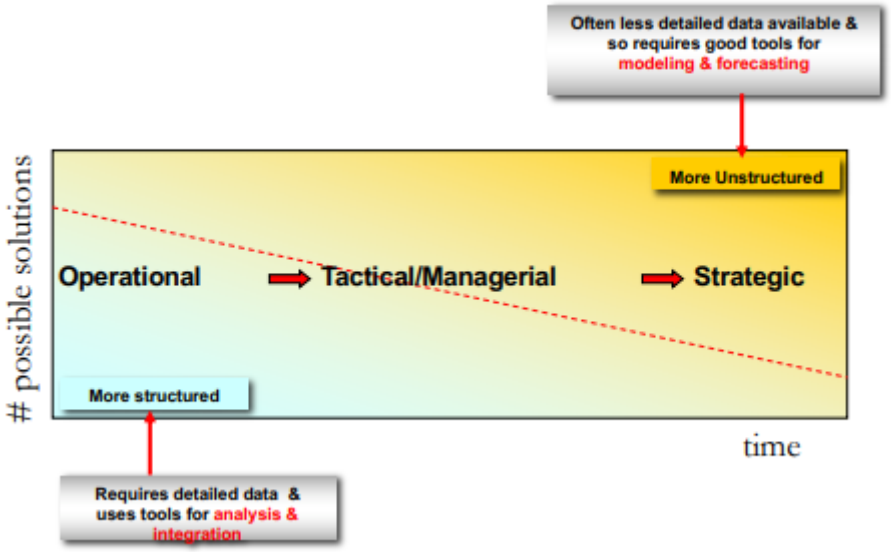
Generate a *population* of random chromosomes
Repeat (each generation)
  Calculate fitness of each chromosome
  Repeat
    Use roulette selection to select pairs of parents
    Generate offspring with crossover and mutation
  Until a new population has been produced
Until best solution is good enough

Decision Trees

6.1 *Uncertainty, Utility, Risk*

Fin'ora sono stati descritti problemi le cui caratteristiche risultano ben definite e per i quali si hanno a disposizione molte informazioni circa la funzione obiettivo, le variabili decisionali e gli eventuali vincoli, ovvero *well-structured problems*. Nella realtà tuttavia il *decision making* è soggetto a un certo livello di incertezza, e spesso si ha a che fare con **unstructured problems**, problemi decisionali non usuali e per i quali può essere necessaria una procedura sviluppata ad hoc. Strettamente collegato a tali problemi vi è il concetto di *non-programmed decisions*, ovvero decisioni non ricorrenti le quali forniscono risposte uniche, e che di conseguenza sono difficilmente riconducibili procedura presviluppate.

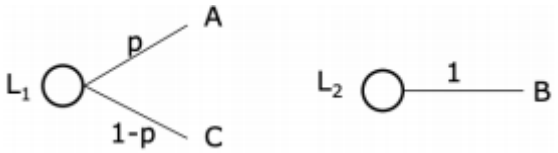
Il seguente grafico mostra come, rispetto a possibili soluzioni e tempo, i problemi non strutturati si posizionino nella angolo destro:



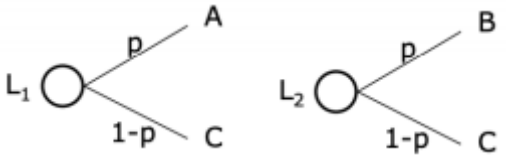
L'**incertezza** di un problema decisionale risiede nella non conoscenza a priori di tutti i possibili risultati di una decisione, e per tale motivo si fa spesso ricorso a strutture probabilistiche legate agli stessi risultati. Tale caratteristica distingue fortemente i problemi non strutturati da quelli ben strutturati (i quali risultano infatti deterministici); un esempio molto comune di *decision problem* caratterizzato

da incertezza riguarda le transazioni finanziarie, per le quali è possibile stimare i diversi possibili risultati solamente in relazione ad una certa funzione di probabilità. I possibili risultati possiedono le seguenti proprietà:

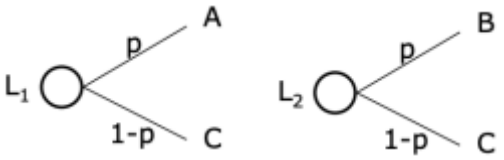
- **Ordinabilità**: Per ogni possibile coppia di risultati A e B, data una funzione di utilità u, possono verificarsi solamente tre situazioni:
  - Si preferisce A a B $u(A) > u(B)$
  - Si preferisce B a A $u(A) < u(B)$
  - A e B sono indifferenti $u(A) = u(B)$
- **Transitività**: Se $u(A) > u(B)$ e $u(B) > u(C)$, allora varrà $U(A) > u(C)$.
- **Continuità**: date le seguenti lotterie L_1, L_2:



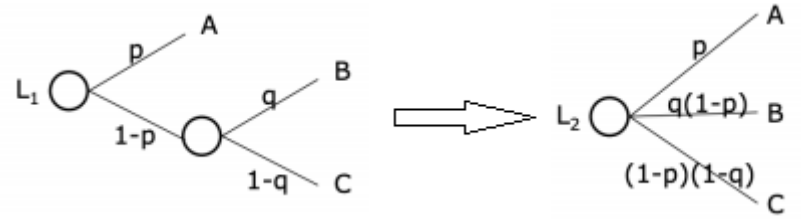
  in cui $u(A) > u(B)$ e $u(B) > u(C)$, esisterà una certa probabilità che $u(L_1) = u(L_2)$
- **Sostituibilità**: date due lotterie con la stessa funzione di probabilità, le quali differiscono per la presenza di A in L_1 e B in L_2 (con $u(A) > u(B)$), allora la lotteria contenente A sarà preferibile.



- **Monotonicità**: date le seguenti lotterie

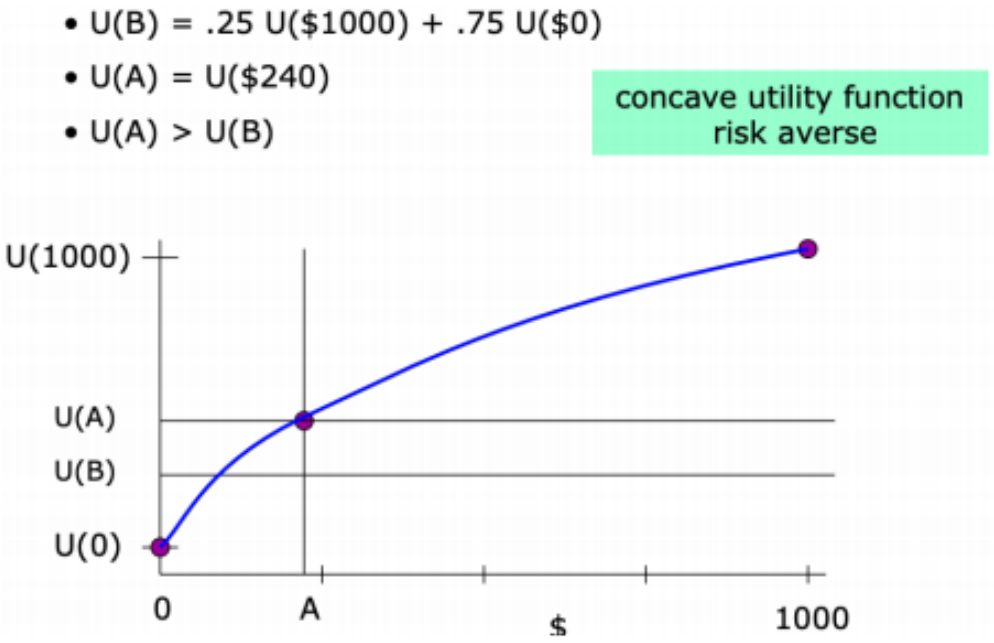


  se $u(A) > u(B)$ e $p < q$, allora la lotteria L_1 sarà preferibile a L_2.
- **Scomponibilità**: data lotteria a più stati è possibile applicare la seguente trasformazione:



- **Utilità totale**: l'utilità totale di una lotteria L sarà data dal valore atteso dei possibili risultati, ovvero dalla somma delle utilità dei singoli risultati pesati per la probabilità che si verifichino $\rightarrow \sum_{k=1}^{n} p_k * E_k$ (E_K risultato k-esimo e p_k probabilità associata).

Quanto detto fin'ora circa l'incertezza di una decisione si lega ai concetti di **utilità e rischio**, infatti sono questi due fattori a determinare le scelte di un individuo quando si trova davanti diverse lotterie. Studi sociali affermano che la maggior parte delle persone (circa 85%) quando si trova a dover scegliere tra 2 lotterie tende a preferire quella meno rischiosa; in caso vi siano due lotterie una con un profitto alto ma incerto e una con un profitto minore ma certo, le persone tenderanno quindi a scegliere la seconda. Tale situazione viene definita *avversità al rischio*, ed è identificabile nel grafico dell'utilità quando si è in presenza di una funzione concava



Vi sono inoltre altri due casi, indifferenza al rischio e inclinazione al rischio, le quali hanno rispettivamente una funzione di utilità lineare o convessa.

Le principali criticità dei *non-structured problems*, legati all'incertezza e al rischio, sono i seguenti:

- I possibili risultati non possono essere previsti/stimati in maniera deterministica, ovvero con certezza.

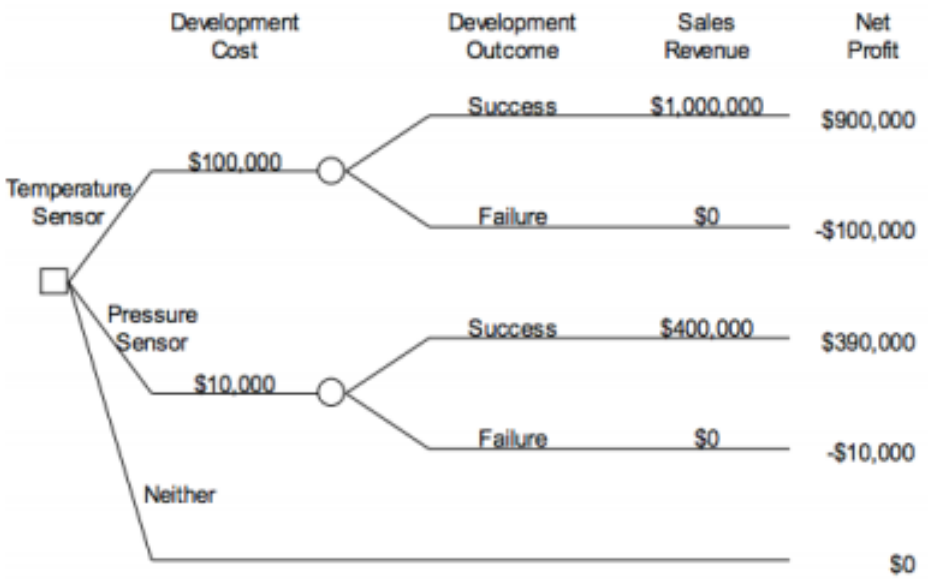
- La struttura non ben definita fa sì che possano essere presi in considerazione un insieme molto ampio di fattori (variabili decisionali).
- La ricerca di nuove informazioni che riducano l'effetto dell'incertezza può richiedere molto risorse.
- L'attitudine al rischio può impattare fortemente sulle decisioni degli individui, fino anche allo sviluppo di irrazionalità nel processo di scelta.

6.2 *Decision Trees*

Tra i principali strumenti utilizzati a sostegno del processo decisionale, in presenza di incertezza e problemi non strutturati, vi sono gli **Alberi Decisionali**. Un esempio di un problema decisionale e della relativa risoluzione ad albero viene fornita nel sottostante esempio.

Product decision. To absorb some short-term excess production capacity at its Arizona plant, Special Instrument Products is considering a short manufacturing run for either of two new products, a temperature sensor or a pressure sensor. The market for each product is known if the products can be successfully developed. However, there is some chance that it will not be possible to successfully develop them.

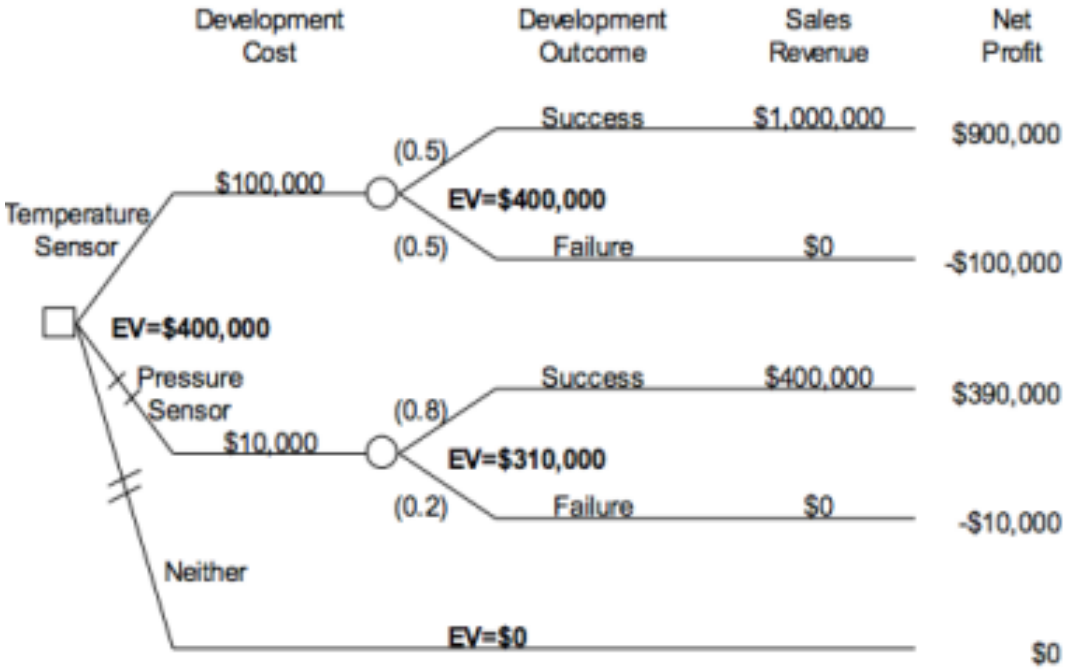
Revenue of $1,000,000 would be realized from selling the temperature sensor and revenue of $400,000 would be realized from selling the pressure sensor. Both of these amounts are net of production cost but do not include development cost. If development is unsuccessful for a product, then there will be no sales, and the development cost will be totally lost. Development cost would be $100,000 for the temperature sensor and $10,000 for the pressure sensor.



Vi sono tre tipologie di nodi all'interno di una struttura ad albero:

- **Nodo decisionale**: rappresentano quei nodi che corrispondono agli *split* conseguenti ad una decisione.
- **Nodi Chance**: sono i nodi che corrispondono agli *split* collegati a possibili risultati derivanti dalla decisione.
- **Nodi finali**: rappresentano i nodi dopo i quali non vi sono altri *split*, ovvero che rappresentano i possibili risultati finali.

Solitamente si associano agli archi che partono dagli *chance nodes* le probabilità che ciascun risultato possa verificarsi; con tali probabilità è possibile calcolare il **valore atteso** collegato all'evento rappresentato dall'albero, ovvero il risultato atteso. Possiamo dunque considerare l'*expected value* come un criterio di individuazione del ramo ottimale, e di conseguenza come indicatore di risoluzione per problemi decisionali (nell'immagine sottostante il primo arco decisionale sviluppa un valore atteso di 400 mila, maggiore del valore atteso degli altri due archi).



Quanto detto fin'ora circa il valore atteso non tiene conto di un concetto fondamentale quando si parla di risultati incerti: l'**avversione al rischio**. L'*expected value* (EV) rappresenterà un buono strumento decisionale solo quando coincide con l'utilità totale, e di conseguenza in situazione di indifferenza al rischio; l'utilità di una decisione incerta può essere infatti molto differente dal suo valore atteso. Possiamo considerare quindi un altro indicatore che viene definito **certainty equivalent** (CE), ovvero il guadagno sicuro che porta una persona a rifiutare un determinato valore atteso non sicuro (se fossi disposto a cedere una lotteria con valore atteso 2500 per 500, tale sarebbe il *certainty equivalent*).

Possiamo quindi definire 3 situazioni:

- Se $CE < EV \rightarrow$ **avversione al rischio**
- Se $CE = EV \rightarrow$ **neutralità al rischio**
- Se $CE > EV \rightarrow$ **propensione al rischio**

Tali situazioni si possono tradurre attraverso le seguenti funzioni nel grafico guadagno/utilità:

La **funzione di utilità** consente dunque di tradurre in valore i possibili risultati di un problema decisionale, tenendo inoltre conto dell'avversione al rischio degli individui. Tipicamente l'utilità viene rappresentata da funzioni esponenziali: definendo *risk tolerance* un indicatore $R > 0$, il quale crescendo mitiga l'avversione al rischio, possiamo definire una funzione di utilità $u(x) = 1 - e^{-\frac{x}{R}}$. Si può inoltre definire il *centainty equivalent* per una funzione esponenziale come $CE = -R \cdot ln(1 - E(U))$. La differenza tra EV e CE rappresenta un indice del rischio della scelta.

Durante la fase di studio del problema decisionale può risultare di grande interesse per i *decision makers* l'**acquisto di informazioni** che possano ridurre l'incertezza delle scelte; è infatti vero che, come mostrato fin'ora, la natura non deterministica di alcuni problemi decisionali porta ad una significativa riduzione del valore atteso dei risultati, rendendo estremamente difficile la loro previsione. Ovviamente risulta molto difficile sia valutare quanto un informazione sia accurata, sia quanto effettivamente si è disposti a pagare (due problemi che sono fortemente legati tra loro); per tale motivo può risultare conveniente definire un *upper bound* del valore dell'informazione, il quale si ha ovviamente in caso di informazione perfetta (si pagherà il massimo cui si è disposti in caso di informazioni che annullino l'incertezza).

E' di fondamentale importanza cercare di stimare il valore che ha un'informazione, problema che risulta spesso molto difficile; come esposto in precedenza il valore di un informazione è strettamente legato all'**accuratezza** della stessa, variando da 0 al valore dell'informazione perfetta, e di conseguenza al fine di determinare tale valore è necessario stimare la probabilità che tale informazione risulti corretta (ovvero il suo livello di accuratezza). Uno dei metodi possibili consiste nell'applicazione di **regole bayesiane**, ovvero basate sulle probabilità condizionate.