Appunti - Decision Models

Alberto Filosa

30/6/2020

Indice

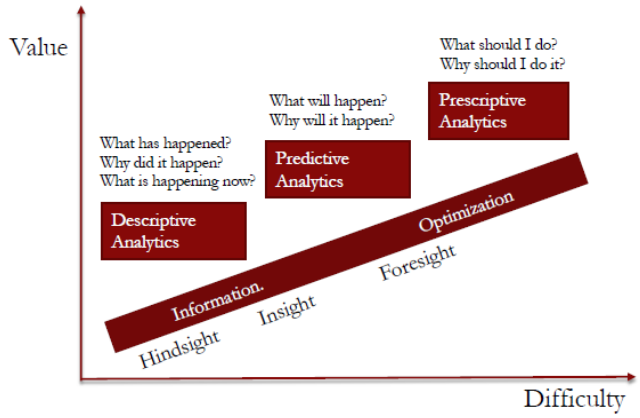
1 Decision Making

Un importante e difficile compito in una azienda di business è quella di prendere una decisione incerta, soprattutto dettata dalla presenza di svariati fattori da prendere in considerazione. Per ***decision making*** si intende un processo di scelta logica tra diverse opzioni e affidarsi successivamente a future azioni. Quando si prende una decisione è necessario considerare tutte le alternative e pesare gli aspetti positivi e negativi della scelta appena compiuta.

I problemi possono essere *strutturati* e *programmati*, di conseguenza gli obiettivi sono chiari, ricorrenti e facilmente definiti, oltre che ripetitivi. Viceversa, essi possono essere non strutturati e non programmati, di conseguenza unici con le informazioni ambigue ed incomplete. Nelle decisioni incerte si prevede la probabilità di un eventuale risultato con il corrispondente fattore di rischio.

Per prendere una decisione si deve utilizzare un ***decision model*** che predice l'output di una decisione per capire o controllare un problema. Il modello deve anche predire cosa succede se viene presa una certa azione, in modo da massimizzare i fattori desiderabili e minimizzare quelli non desiderabili. Il computer esamina le informazioni utilizzando metodi matematici per trovare dei pattern nei dati. Le *analytics* studiano dati storici per cercare possibili trend nei dati e predire i dati del futuro. Essa prende in considerazione tre step:

1. Analisi Descrittiva (anche chiamata esplorativa), che estrae informazioni dai dati calcolando indici descrittivi quali moda, media, mediana, varianza, deviazione standard, ecc;
2. Analisi Predittiva, che costruisce modelli di previsione quali serie storiche, modelli lineari, ecc;
3. Analisi Prescrittiva, che determina cosa deve verificarsi e come far accadere l'evento, ad esempio la *what-if analysis*.



2 Linear Programming

I modelli sono uno strumento essenziale per risolvere problemi decisionali, in quanto non tutti possono essere modellati tramite semplici modalità. Quando si compie una decisione si deve scegliere una modalità tra le alternative, principalmente secondo la logica della vincita-perdita: questo fenomeno è chiamato *Frame Effect*. I modelli decisionali aiutano le aziende nel compiere ottime decisioni, ma non garantiscono che il risultato sia sempre positivo; comunque è sempre meglio rispetto a prendere decisioni azzardate .

I modelli devono essere molto accurati in modo da ottenere risultati consistenti, per cui avviene un processo di *ottimizzazione* per ottenere il miglior risultato utilizzando una parte limitata di risorse. L'ottimizzazione presenta le seguenti caratteristiche:

- Bisogna definire una *funzione obiettivo*, che deve essere minimizzata o massimizzata (in base al tipo di studio);
- Si utlizzano un numero limitato di *variabili di decisione* che controllano il valore della funzione obiettivo;
- Si definiscono i *vincoli* che specificano il range di accettabilità dei valori delle variabili coniderate nel modello.

In definitiva, il problema di ottimizzazione consiste nel trovare il valore delle variabili di decisione del modello che minimizza o massimizza la funzione obiettivo, $f_0(X_1, X_2, \dots, X_n,)$, e che soddisfa anche i vincoli imposti:

$$f_k(X_1, X_2, \dots, X_n,) = / \leq / \geq b_k \ \forall k = 1, \dots, m$$

$$\begin{array}{lrcl} \max \text{ (o min)} & f_0(X_1, X_2, \dots, X_n) & & \\ \text{rispetto a} & f_1(X_1, X_2, \dots, X_n) & \leq & b_1 \\ & \vdots & & \\ & f_k(X_1, X_2, \dots, X_n) & = & b_k \\ & \vdots & & \\ & f_m(X_1, X_2, \dots, X_n) & \geq & b_m \end{array}$$

Se tutte le funzioni in un processo di ottimizzazione sono lineari (sia funzioni obiettivo che vincoli imposti), si avrà un problema di ottimizzazione lineare. Si presentano le fasi di formulazione dei problemi LP:

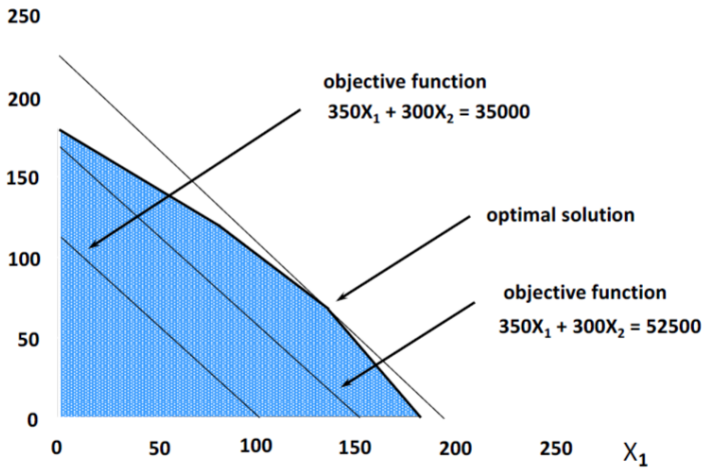
1. Comprendere il problema;
2. Identificare il numero e le varibili da considerare ed inserire nel modello;
3. Stabilire la funzione obiettivo come combinazione lineare delle variabili;
4. Stabilire i vincoli come combinazione lineare delle variabili;
5. Identificare ogni limite superiore o inferiore alle variabili di decisione.

2.1 Risoluzione Grafica

Oltre alla risoluzione matematica, è possibile risolvere i problemi LP tramite un approccio grafico: i vincoli definiscono una regione accettabile, dato che definiscono delle rette, ed il miglior punto nello spazio è la soluzione ottimale del problema.

Nel caso di un problema lineare con due variabili, i passaggi da seguire sono i seguenti:

1. Rappresentare graficamente i confini dei vincoli imposti dal modello;
2. Identificare la regione di accettazione;
3. Localizzare la soluzione ottimale tramite le curve di leva o identificando i punti estremi.



In un problema lineare si possono presentare anche delle anomalie: si possono ottenere soluzioni alternative, ovvero più valori ottimali; vincoli ridondanti, che non contribuiscono a delineare la regione di accettazione (si potrebbero rimuovere, ma non è consigliato); soluzione illimitata, nella quale la regione di accettazione non presenta limiti; infattibilità, nella quale due vincoli non si intersecano e non generano alcuna regione di accettazione.

2.2 Assunzioni

Dato che la funzione obiettivo del modello lineare è anch'esso lineare, il modello ed i vincoli saranno *proporzionali*, ovvero il contributo della funzione obiettivo da ogni variabile è proporzionale al valore della variabile di decisione, e *additivi*, ovvero il contributo della funzione obiettivo è indipendente dai valori di altre variabili di decisione. Ovviamente queste assuzioni vanno ad influire anche sui vincoli imposti dal modello. Inoltre, il modello è *divisibile*, ovvero ogni variabile di decisione permette valori in frazione, e *certo*, ovvero ogni parametro è noto.

2.3 Simplex Method

Il ***simplex method*** è un algoritmo utilizzato per individuare la soluzione ottimale per il problema lineare. Inanzitutto, è necessario convertire tutte le disequazioni del linear programming in equazioni aggiungendo delle variabili chiamate *slack*. In base al tipo di disequazione ($+s_k$ se c.l $\leq$, altrimenti $-s_k$) si somma/sottrae la suddetta variabile alla combinazione lineare:

$$a_{k1}X_1+\cdots+a_{kn}X_n \leq b_k \implies a_{k1}X_1+\cdots+a_{kn}X_n+S_k = b_k$$

$$a_{k1}X_1+\cdots+a_{kn}X_n \geq b_k \implies a_{k1}X_1+\cdots+a_{kn}X_n-S_k = b_k$$

In un sistema con n variabili in m equazioni (con $n \geq m$) si selezionano tutte le m variabili per risolvere il sistema di equazioni, mentre le restanti $n-m$ variabili vengono fissate a 0.

L'algoritmo identifica ogni punto estremo del problema, chiamato *basic feasibile solution*, e successivamente si muove ad un altro punto estremo adiacente in modo da migliorare il valore della funzione obiettivo. Questo processo avviene scambiando una *basic variable* con una non basica al fine di crearne delle nuove. Nel momento in cui alcun punto estremo adiacente ha un valore della funzione obiettivo maggiore, il processo si stoppa e l'ultimo valore individuato sarà quello ottimale.

2.4 Sensitivity Analysis

Quando si risolve un problema di ottimizzazione lineare si pensa di assumere che i valori dei coefficienti predetti dal modello siano certi, in realtà raramente ciò avviene. La *sensitivity analysis* aiuta nel capire quanto è sensibile il cambiamento dei coefficienti del modello individuando la soluzione ottimale. Gli approcci possono essere principalmente due:

- Cambiare i dati e rifar partire il modello (alcune volte è l'unico approccio utilizzabile);
- Utilizzare il *simplex method* che implementa al suo interno report sulla sensibilità del modello costruito.

In questo modo è possibile rispondere a domande quali quanto è possibile modificare la funzione obiettivo senza il cambiamento della soluzione ottimale, l'impatto del valore della funzione obiettivo sui cambiamenti dei vincoli e delle variabili di decisione.

Se le variabili slack (S) dei vincoli sono nulli allora i vincoli sono *binding* in modo da prevenire valori troppo elevati della funzione obiettivo. Il valore della variabile *slack* indica la differenza tra gli estremi dei vincoli. Questi range vengono definiti ottimali poichè il valore della soluzione ottimale non si modifica cambiando i parametri della funzione obiettivo, assumendo che tutti i coefficienti rimangano costanti.

Ad esempio, si considera un parametro di una funzione obiettivo (C1) con valore 350; esso può variare tra $300 \leq C1 \leq 450$ senza che si modifichi la soluzione ottimale, assumendo che i rimanenti coefficienti rimangano costanti.

Il prezzo contabile, in inglese *shadow price*, indica l'ammontare del cambiamento del valore della funzione obiettivo all'aumentare di una unità della right hand side del vincolo, assumendo che tutti i coefficienti rimangano costanti. Cambiando il valore della RHS per un vincolo *binding*, la regione di accettazione e la soluzione ottimale si modificano.

I costi ridotti indicano la differenza dei profitti marginali ed il valore per unità delle risorse consumate. In base al tipo di problema di ottimizzazione ed al valore ottimale della variabile di decsione, il valore ottimane dei costi ridotti sarà maggiore, minore o uguale a zero:

2.5 Integer Linear Programming

Se una o più variabili in un problema lineare devono assumere valori interi, si incorre in un *integer linear pro-*

gramming, ovvero un problema lineare con un ulteriore vincolo, chiamato vincolo di integrità. Variabili intere permetto di costruire modelli più accurati, dato che alcune variabili si riferiscono ad oggetti fisici.

In un problema lineare, alcune volte si ottengono valori interi per la costruzione della regione di accettazione. Una prima soluzione, non consigliata, è quella di arrotondare all'intero più vicino il valore; non sempre, però, è affidabile in quanto la soluzione arrotondata può essere non accettabile, per i vincoli imposti dal modello, o subottimale. Una seconda soluzione è utilizzare l'algoritmo *Branch & Bound*, che teoricamente può risolvere il problema ILP, ma praticamente richiede un alto sforzo computazionale. L'algoritmo computa la soluzione di una serie di problemi lineari, chiamati anche *candidate problem*.

Siccome l'algoritmo B&B necessita di un enorme sforzo in termini computazionali e quindi di tempo, esistono altri metodi che specificano un fattore di tolleranza subottimale, in modo da fermare l'algoritmo quando una soluzione intera è stata trovata (chiamata *stopping rule*) in una percentuale della soluzione ottimale in termini globali. In particolare, è possibile calcolare la soluzione ottimale per un problema LP, che definisce dei limiti per la il valore della funzione obiettivo, e utilizzarla per risolvere il problema ILP. Per problemi di massimizzazione, la funzione obiettivo ottimale del problema LP è un limite superiore del valore ottimale intero, mentre per problemi di minimizzazione è un limite inferiore.

L'algoritmo Branch & Bound lavora nei seguenti step:

1. Fase di inizializzazione, nella quale si rivolve il problema LP senza la condizione di integralità. Se i risultati sono lineari, la si utilizza come soluzione ottimale intera. Altrimenti, se si vuole risolvere un problema di massimizzazione si assegna al valore della funzione obiettivo della miglior soluzione intera $Z_{best} = -\infty$, altrimenti $Z_{best} = +\infty$;
2. Fase di Branching: si definisce X_j la variabile che viola la condizione di integralità, b_j il suo valore non intero. Si approssima il valore della funzione obiettivo al più vicino intero $\text{int}(b_j)$, ma sempre minore b_j. In questo modo si risolvono due problemi lineari, la prima aggiungendo come vincolo $X_j \leq \text{int}(b_j)$, mentre per il secondo $X_j \geq \text{int}(b_j) + 1$, aggiungendoli in una lista di problemi candidati da risolvere;
3. Fase di bounding, risolvendo i sub-problemi definiti al punto precedente. Si presentano diversi modi:

   a. Se la lista dei candidati è vuota, l'algoritmo si stoppa; altrimenti, si risolve uno dei problemi candidati;
   b. Se la soluzione non è accettabile, si ritorna al punto 3a; altrimenti, si assegna Z_{cp} il valore della funzione obiettivo come problema candidato;
   c. Se $Z_{cp} \nleq Z_{best}$ per un problema di massimizzazione, o $Z_{cp} \ngeq Z_{best}$ per un problema di minimizzazione, si ritorna al punto 3a;
   d. Se la soluzione attuale non rispetta la condizione di integralità e $Z_{cp} \leq / \geq Z_{best}$, si ritorna al punto 2;
   e. Se la soluzione soddisfa la condizione di integralità, è stata trovata una soluzione intera migliore e si assegna $Z_{best} = Z_{cp}$; si ritorna al punto 3;

4. Fase di stop, nella quale è stata trovata la soluzione ottimale.

2.6 Network Model

Numerosi problemi di business possono essere rappresentati graficamente tramie la costruzione di *Network Flow*, definiti come una collezione di nodi connessi tramite archi. Esistono 3 tipi di nodi:

- Offerta (classificato come numeri negativi);
- Domanda (classificato come numeri positivi);
- Trasbordo.

Un primo modello è chiamato *Transhipment Problem*, legata al trasporto di prodotti in più negozi. Una volta definita la funzione obiettivo da minimizzare (o massimizzare), è necessario definire i vincoli per ogni nodo. Per un Minimum Cost Network, in base al numero totale di domanda e al numero totale di offerta si utilizza la regola di bilancio chiamata Balance of Flow applicata ad ogni nodo:

Il *Shortest Path Problem* si concentra sul minizzare il percorso dato un numero di tappe da compiere. è un caso specifico del Transhipment Problem dove è presente un nodo di offera con valore (-1), un nodo di domanda con valore (+1) e tutti gli altri nodi hanno un valore di domanda/offerta (0). Esistono due tipi di funzioni obiettivo per risolvere il problema:

- Trovare il tragitto più veloce, minimizzando il tempo di viaggio;
- Trovare il tragitto panoramico più lungo, massimizzando i punti scenici.

Una volta definita la funzione obiettivo, si costruiscono i vincoli come visto precedentemente.

L'*Equipment Replacement Problem* determina il periodo di tempo nella quale è necessario cambiare l'attrezzatura. In base al tipo di offerta, è possibile sottoscrivere contratti che minimizzano le spese; può essere modellato come un Shortest Path Problem.

Alcuni problemi di network flow non necessitano di avere nodi di trasporto, ma solo nodi di domanda ed offerta, perciò trasporti da punto a punto; questi vengono chiamati *Transportation & Assignment Problems*. Una volta definita la funzione obiettivo, che solitamente si vuole minimizzare, si definiscono i vincoli riferiti alla massima capienza alla domanda di un singolo luogo.

I *Generalized Network Flow Problems* sono dei problemi nella quale si verificano nell'aumento (o diminuzione) di prodotti lungo gli archi. Questi modelli richiedono modifiche alla modellazione ; si formulano come i Transhipment Problem.

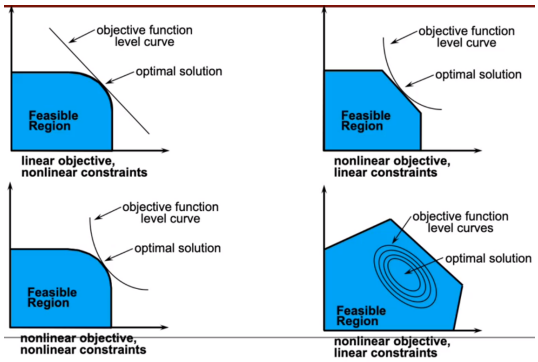
Il *Maximal Flow Problem* ha l'obiettivo di determinare la massima quantità di flusso in un network. Il problema si risolve come il Transhipment Problem.

Nel caso di un network con n nodi, si utilizza il *Spanning Tree Problem*, con $n-1$ archi che connettono tutti i nodi e che non contenogono dei loop. Il *Minimal Spanning Tree Algorith* risolve il problema di determinare la connessione di tutti gli archi al minimo costo. L'algoritmo segue i passaggi:

1. Selezionare un nodo, chiamata subnetwork;
2. Aggiungere al subnetwork corrente l'arco meno costoso che connette i nodi. Se sono presenti archi con stesso costo, se ne scene uno in arbitrariamente;
3. Se tutti i nodi sono nel subnetwork, si ottiene la soluzione ottimale. Altrimenti si ritorna al passo 2.

3 Non Linear Programming

Fino ad adesso si è considerato solamente un problema di ottimizzazione lineare, ma non sempre è possibile risolvere questi tipi di problemi. Di conseguenza, si introduce il concetto di *Non Linear Programming* che presenta una funzione obiettivo ed una o più vincoli non lineari. Il procedimento di risoluzione è analogo a quelli visti in precedenza, definendo perciò le decision variables ed i vincoli, ma le differenze principali con i problemi lineari stanno nell'utilizzo di differenti algoritmi e nella difficoltà di risoluzione, quindi un maggior tempo di risoluzione nel risolvere i problemi NLP.



Non sempre la soluzione ottimale è individuata all'estremo della regione di accettazione come accadeva nei problemi LP, ma anche al suo interno. Di conseguenza, si dovranno utilizzare differenti algoritmo per individuare la optimal solution.

L'algoritmo inizia in un punto appartenente alla regione di accettazione, di solito nell'origine. Successivamente si muove nella parte della regione nella quale si ha il più veloce miglioramento, fino a quando non si trova la soluzione ottimale migliore. Non sempre la soluzione è la migliore in termini globali, ma alcune volte può essere una soluzione ottimale locale. Non è consigliato partire dall'origine, ma da un valore approssimato della stessa grandezza del valore ottimale.

3.1 Bisection Method

Il *Bisection Method* è un algoritmo utilizzato per risolvere i problemi di ottimizzazione non lineare. Prima di introdurre i passaggi di risoluzione, si introducono dei concetti matematici.

> Una equazione $f(x) = 0$, con $f(x)$ continua su $\mathbb{R}$, ha almeno una soluzione tra due punti x_i e x_u se $f(x_l) \times f(x_u) < 0$

Unimodality:

> Una funzione unimodale consiste in esattamente un aumento crescente e decrescente di una parte di funzione
>
> Se la funzione $f(x)$ cambia segno tra due punti x_l e x_u, esiste almeno una soluzione per $f(x) = 0$ tra di essi.

L'algoritmo si formula nei seguenti passaggi:

1. Scegliere due punti x_l e x_u tale che $f(x_l) \times f(x_u) < 0$ (ovvero che cambia segno tra i due punti);

2. Stimare la soluzione x_m come il punto medio tra x_l e x_u

$$x_m = \frac{x_l + x_u}{2}$$

3. Ridurre la grandezza dell'intervallo:

a. Se $f(x_l) \times f(x_u) < 0$, la soluzione non è vera tra x_l e x_m e si assegna $x_u = x_m$;
b. Se $f(x_l) \times f(x_u) > 0$, la soluzione non è vera tra x_m e x_u e si assegna $x_l = x_m$
c. Se $f(x_l) \times f(x_u) = 0$, la soluzione è x_m. (STOP);

4. Stimare la nuova soluzione x_m e trovare l'errore relativo approssimato in termini assoluti:

$$|\varepsilon_a| = \left| \frac{x_m^{new} - x_m^{old}}{x_m^{new}} \right| \times 100$$

5. Comparare $|\varepsilon_a|$ con l'errore iniziale ε_s:

a. Se $|\varepsilon_a| > \varepsilon_s$, ritornare al punto 2 utilizzando nuovi punti;
b. STOP.

I vantaggi principali di questo algoritmo sono la convergenza sempre garantita e che all'aumentare delle ierazioni ci si avvicina sempre di più allo zero. Tuttavia, la convergenza è molto lenta, anche se il valore iniziale è molto vicino allo zero.

3.2 Newton Method

Un altro metodo di risoluzione per i problemi non lineari è chiamato *Newton Method*, che genera una sequenza di punti convergenti alla soluzione ottimale. Esistono due tipi di algoritmi:

- *Dicotomo*, che individua la soluzione della equazione della derivata pari a 0 ed ad ogni iterazione si riduce l'intervallo;
- Bisection Method (visto in precedenza).

I criteri di termine servono per terminare l'algoritmo, che si conclude se la soluzione è accurata ad un certo livello di tolleranza, se la soluzione ha un piccolo miglioramento, se si raggiunge il numero massimo di iterazione, se la soluzione diverge, se la soluzione è in un loop.

Il Newton Method adatta una soluzione quadratica a $f(x)$ utilizzando sia il gradiente che l'informazione di curvatura su x. I passaggi sono i seguenti:

1. Utilizzare l'approssimazione di Taylor:

$$f(x+h) = f(x) + f'(x) \times h + 1/2 f''(x) h^2$$

2. Trovare la soluzione di f':

$$f'(x+h) = f'(x) + f''(x)h \implies h = -\frac{f'(x)}{f''(x)}$$

3. Si assume che $x_{k+1} = x_k + h_k = x_k - \frac{f'(x)}{f''(x)}$

I vantaggi principali di questo algoritmo è che non necessita di trovare la soluzione perfetta e si ha una convergenza quadratica, ovvero l'accuratezza ottimale raddoppia ad ogni iterazione. Tuttavia, la convergenza globale non è ottima e molte volte fallisce se il valore è troppo lontano dal minimo.

3.3 Gradient Method

Il *Gradient Method* è un algoritmo di ottimizzazione non lineare per problemi multivariati. Per tutti metodi di ottimizzazione si sceglie la direzione d_k da seguire partendo da un punto x_k; successivamente, si massimizza (o minimizza) lungo la direzione il valore al fine di trovare un nuovo punto $x_{k+1} = x_k + \alpha_k d_k$, con k il numero di iterazione e α_k chiamata *Step Size*. Esso viene calcolato minimizzando la seguente funzione (con x_k e $\nabla f(x_k)$ note): $\max f(x_{k+1}) = f(x_k + \alpha_k \nabla f(x_k))$ e la si risolve utilizzando la derivata parziale per α_k:

$$\frac{\partial f(x_k + \alpha_k \nabla f(x_k))}{\partial \alpha_k} = 0$$

N.B.: dato che il gradiente è definito come il cambiamento della funzione in quel punto, il gradiente è utilizzato come direzione di ricerca in modo da ridurre il numero di iterazioni.

Il *Steepest Descent Method* segue questi step:

1. Scegliere un punto iniziale x_0;
2. Calcolare $\nabla f(x_k)$ alla k-esima iterazione;
3. Calcolare il vettore di ricerca $d_k = \pm \nabla f(x_k)$;
4. Calcolare il punto successivo $x_{k+1} = x_k \pm \alpha_k d_k$;
5. Utilizzare un metodo di risoluzione univariata per ottenere α_k;
6. Determinare la convergenza utilizzando una tolleranza $|f(x_{x+1}) - f(x_k)| < \varepsilon_1$, oppure $||\nabla f(x_{k+1})|| < \varepsilon_2$.

3.4 Newton Method

Il *Newton Method* esiste anche per i NLP multivariati. Essa approssima una funzione $f(x)$ con un funzione quadratica nella vicinanza del punto utilizzando la espansione di Taylor:

$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)\Delta x + \Delta x H(x_k)\Delta x$$

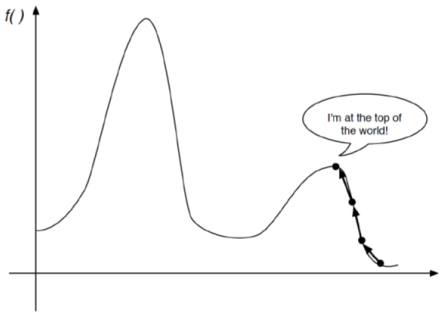
Si vuole risolvere la seguente equazione:

$$\frac{\partial f(x_k + \Delta x)}{\partial \Delta x} = 0 \implies \Delta x = -H(x_k)^{-1}\nabla f(x_k)$$

L'algoritmo segue questi step:

1. $K = 0$;
2. Scegliere un punto d'inizio x_k;
3. Calcolare $\nabla f(x_k)$ e $H(x_k)$;
4. Calcolare il Newton Step $x_{k+1} = x_k - H(x_k)^{-1}\nabla f(x_k)$;
5. Se $H(x_k)$ è definita positiva, si arriva alla convergenza; altrimenti, ritornare al punto 2 con $K + 1$.

A differenza del Gradient Method, il metodo di Newton utilizza sia il gradiente che la matrice Hessiana. Inoltre, riduce il numero di iterazioni richieste, ma aumenta il peso computazionale; di conseguenza, non è consigliato per funzioni molto complesse.



Esistono due modi per evitare la convergenza locale: la prima è aumentare lo step size in modo tale da identificare non solo la prima cima del massimo locale, ma potenzialmente anche la successiva. Non è un metodo consigliato in quanto l'algoritmo potrebbe non convergere. In alternativa, è possibile far girare il metodo in un grande loop, ogni volta che inizia da un punto e che trova il valore ottimale; così facendo, è possibile identificare la soluzione ottimale e confrontarla con le altre. Questo *Try Hard* non è efficiente in quanto alcuni algoritmi hanno delle assunzioni molto stringenti. Le metaeuritiche sono utili in quanto hanno poche o nessuna assunzione; sono metodi generali, ma spesso utili come ultima soluzione in modo da trovare la soluzione ottimale approssimata.

Per trovare le possibili soluzione, è necessario seguire questi passaggi:

- Avere delle possibili soluzioni iniziali, chiamata in inglese *initialization procedure*;
- Valutare la qualità della soluzione, chiamata in inglese *assessment procedure*;
- Copiare una soluzione candidata e modificare leggermente la soluzione con procedure leggermente diverse, chiamata in inglese *modification procedure*.

4 Metaheuristics

Le *Metaeuristiche* sono delle procedure utilizzate dagli algoritmi per approssimare il valore nella quale la funzione converge; è spesso utilizzato con algoritmi molto complessi. Una volta che la funzione converge, non sempre è garantita la soluzione ottimale, che sia un punto di minimo o massimo globale; molte volte accade che sia un punto di ottimizzazione locale. Per risolvere questo problema, esistono dei meccanismi che cercano di evitare di essere intrappolati nello spazio di ricerca. È spesso utilizzato per risolvere problemi molto complessi perchè esplora lo spazio di ricerca in modo efficacie per identificare le soluzioni possibili. Dato che non sempre è garantita la convergenza, le metaeuristiche sono utilizzate assieme ai metodi esatti.

Il termine metaeuristica è utilizzato per descrivere un ramo della ottimizzazione stocastica, ovvero una generica classe di algoritmi e tecniche che utilizza procedure casuali per individuare la soluzione ottimale (o l'approssimazione) per problemi molto complessi. Inoltre, sono utilizzati per risolvere problemi con poche informazioni, ad esempio non si conosce che tipo di soluzione si vuole ottenere, non si conosce la direzione per il valore ottimale e la soluzione forza bruta richiede troppo spazio computazionale.

4.1 Hill Climbing

Una delle procedure euristiche più utilizzate è chiamata *Hill Climbing*, che consiste nella iterazione di un punto

fino alla ricerca del punto ottimale locale. Esso è strettamente legato al metodo Gradient Ascent, ma non richiede la conoscenza del gradiente o la direazione di ottimizzazione. Iterativamente identifica la soluzione candidata e viene sostituita se ne trova una migliore. La procedura segue questi passaggi:

1. Assegnare ad S la soluzione iniziale candidata;
2. Modificare leggermente la soluzione S (diventa R);
3. Se Qualità$_R$ > Qualità$_S$, allora si assegna ad S il valore della leggera modifica e si ritorna al punto 2.

Se la soluzione è quella ottimale, o se si ha raggiunto il limite massimo computazionale, allora si stoppa l'algoritmo.

Esiste un algoritmo, chiamato *Steepest Ascent Hill-Climbing*, che campiona n trasformazioni desiderate per individuare i candidati ottimali. Il procedimento è molto simile al precedente, la differenza principale è che si sceglie il numero di trasformazioni desiderate per campionare il gradiente e si assegna a W il valore migliore identificato.

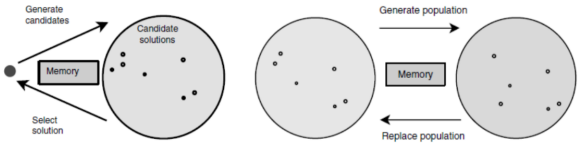
Le soluzioni possibili possono essere un vettore, una lista di oggetti di arbitraria lunghezza, un albero, un grafo o una combinazione di esse. Una semplice e comune rappresentazione per una soluzione possibile è un vettore di valori reali.

Uno dei problemi principali di questo metodo è che se il random noise aggiunto è molto piccolo, non sarà in grado di identificare il valore ottimale globale, ma solo locale. Dall'altro lato, se il random noise aggiunto è troppo grande, l'algoritmo avrà molta difficoltà per convergere e passerà da un picco all'altro senza identificare il valore ottimale globale.

Un algoritmo è ottimizzato globalmente se riesce ad identificare il valore ottimo globale. L'algoritmo più semplice è chiamato *Random Search*, estremo nella esplorazione (e quindi nella ottimizzazione globale) a differenza del Hill Climbing, estremo nell'utiizzo (e quindi nella ottimizzazione locale).

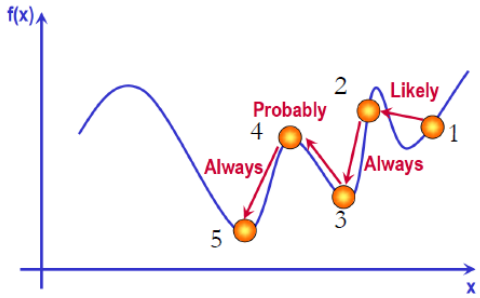
Esistono due tipologie principali di metaeuristiche:

- *Trajectory Based*: da una soluzione iniziale s_0 si generano le soluzioni possibili. Se la soluzione scelta è migliore di quella iniziale a partire da un intorno, la si utilizza come soluzione migliore ed il ciclo si termina quando si è individuata la soluzione ottimale;
- *Population Based*: da un insieme di soluzioni P_0, si generano altre popolazioni di candidati e se sono migliori di quella iniziale, le si utilizzano come soluzioni migliori.



4.2 Simulated Annealing

Il *Simulated Annealing* è un metodo di ottimizzazione stocastico senza vincoli: da un punto iniziale, al posto di ridurre la funzione di costo ad ogni iterazione come i metodi di ottimizzazione locali, considera accettando o rifiutando iterativamente punti candidati. Individuando la soluzione migliore, il metodo converge.



Il vantaggio principale è che questo metodo controlla in modo intelligente il grado di casualità aggiunta al metodo stocastico di ricerca: inizialmente la casualità della funzione di valutazione è molto alta e successivamente diminuisce in base ad un *annealing schedule*.

L'algoritmo segue questi passaggi:

1. Scegliere un punto iniziale $X = X_0$ $(K = 0)$;
2. Valutare la funzione di costo $F = f(X_k)$;
3. Spostare casualmente X_k ad una nuova soluzione X_{k+1};
4. Se $f(X_{k+1}) < F$, allora accettare la nuova soluzione e la nuova soluzione sarà $X = X_{k+1}$ e $F = f(X_{k+1})$;
5. Se $f(X_{k+1}) > F$, allora si accetta la nuova soluzione con una certa probabilità e la nuova soluzione sarà $X = X_{k+1}$ e $F = f(X_{k+1})$ se un numero casuale è minore di ε, che può essere costante o casuale. ;
6. Si passa allo step successivo $K = K + 1$ e si ritorna al punto 2.

Il valore dell'errore è definito dal *Metropolis criterion probability of acceptance*, che sfrutta la distribuzione di Boltzmann per calcolare la probabilità:

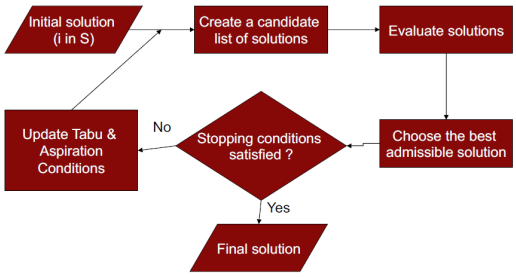
$$\varepsilon = \exp\left[-\frac{f(X_{k+1}) - F}{T_{k+1}}\right]$$

con T_{k+1} un parametro che diminuisce gradualmente. Il valore avrà una alta temperatura se cerca di accettare tutte le soluzioni anche se il numeratore è grande, mentre avrà una bassa temperatura se accetta solamente la nuova soluzione dove il numeratore è piccolo.

4.3 Tabu Search

Il *Tabu Search* è un metodo che salta alcune zone dello spazio di ricerca e che ricerca il valore ottimale utilizzando penalità e bonus. In particolare, utilizza una struttura che esplora la storia di ricerca; è una via di mezzo tra il Branch & Bound ed il Simulated Annealing. Esso utilizza una memoria extra per evitare dei loop e per diversificare la ricerca; inoltre, sfrutta la memoria per classificare un sottinsieme di movimenti in un intorno come **tabù**, creandone una lista.

Questo metodo ha una importante eccezione, chiamata *aspiration criterion*: se nello spostamento esiste una soluzione migliore, la classificazione viene riscritta. I passaggi da seguire sono raffigurati nella seguente immagine:



Si presentano le condizioni di stop se l'algoritmo non converge:

- Se $N(i, K+1) = 0$, ovvero se non è presente alcuna soluzione nell'intorno della soluzione i;
- K è più grande del numero massimo di iterazioni;
- Il numero di interazioni dell'ultimo miglioramento i^* è più grande di un numero specifico;
- L'evidenza porta ad un risultato migliore rispetto alla soluzione ottimale.

4.4 Genetic Algorithm

Basato sulla idea della evoluzione biologica Darwiniana, il *Genetic Algorithm* è un metodo utilizzato per risolvere vari problemi difficili.

L'algoritmo segue questi passaggi:

1. Generare un insieme di soluzioni randomiche;
2. Testare ogni soluzione in un insieme e classificarle;
3. Rimuovere le soluzioni non ottimali;
4. Duplicare alcune soluzioni ottime e compiere delle piccole modifiche ad alcune di esse;
5. Ritornare al punto 2.

L'algoritmo si conclude quando la soluzione è ottimale, che dipende dal tipo di problema. Questo algoritmo alcune volte cifra le soluzioni come una stringa di bit; ognuno di esso rappresenta un aspetto per la soluzione al problema ed è necessario testare ogni stringa per ottenere un punteggio sulla bontà della soluzione.

In questo metodo lo spazio di ricerca può essere visto come una superficie, in questo caso l'altezza e ogni possibile soluzione è definito come punto nello spazio. L'algoritmo cerca di muoversi nei punti con un alto fitness nello spazio.

5 Decision Trees

Prima di definire cosa sono i Decision Tree, è necessario introdurre il concetto di *Decision Making*, definito come la possibilità di prendere una buona decisione in base ad una serie di analisi compiute. Esistono due tipologie di prolemi:

- Problemi strutturati, nella quale gli obiettivi sono chiari, dato che precedentemente accaduti e le informazioni a riguardo sono noti e completi;
- Problemi non strutturati, nuovi o inusuali nella quale l'informazione è ambigua, incompleta che generi una unica risposta.

Le condizioni per prendere una decisione possono essere *certe*, nella quale tutte le possibili risposte sono note ed è possibile prendere accurate decisioni. Tuttavia, molte volte le condizioni sono *incerte* in quanto non si conoscono tutte le possibili risposte. Se una possibile soluzione è nota, allora è possibile stimare la verosimiglianaza della risposta certa, chiamata rischio.

La Decision Analysis è necessaria per prendere delle accurate decisioni ed è caratterizzata da diverse azioni;

Orderability: per ogni coppia di risposte iniziali, si preferisce la decisione A a B, B ad A oppure A e B sono entrambe preferibili.

Transitivity: se la decisione A è meglio di B e B meglio di C, allora A è meglio di C.

Continuity: se si preferisce la decisione A a B e B a C, allora esiste una certa probabilità nella quale si preferisce utilizzare questa lotteria.

Substitutability: se si preferisce la decisione A a B, allora date due identiche lotterie, in una presente A e nell'altra B, si preferisce quella che contiene A.

Monotonicity: se si preferisce la decisione A a B, e se la probalità di A pari a p è più grande di quella di B pari a q, allora si preferisce una lotteria che preferisce A su B con alta probabilità.

Decomposability: data una lotteria a due stadi, nella prima A con probabilità p e nella seconda B con probabilità q e C con probabilità $1-q$, è equivalente costruire una lotteria a singolo stadio con tre possibile risposte: A e B con stessa probabilità di prima e C con probabilità $(1-p)(1-q)$.

Se vengono soddisfatte queste assunzioni, allora esiste una funzione U tale che se si preferisce A a B, allora $U(A) > U(B)$ mentre se sono entrambe preferibili, allora $U(A) = U(B)$.

L'utilità della lotteria è definita come il valore atteso della utility della risposta:

$$U(L) = pU(A) + (1-p)U(B)$$

Il *Decision Tree* è definito come il modello analitico utilizzato nella Decision Analysis. Più è complessa l'analisi, più sarà complessa la decisione da prendere. La risposta non è certa, in quanto esistono numerosi fattori da prendere in considerazione; è necessario raccogliere informazioni per ridurre l'incertezza, ma comunque l'attitudine al rischio impatta le decisioni prese.

Esistono tre tipi di nodi:

- Decision Node, che presenta le possibili alternative;
- Chanche Node, che presenta le possibili riposte di sviluppo;
- End Point, che rappresenta la fine del problema.

Il valore atteso è una ottima misura utilizzata nel lungo periodo ed è definita come il valore medio che ci si aspetta selezionando una alternativa:

$$EV = \sum_{i}^{k} P(X_i)X_i$$

5.1 Risk Aversion

Il valore di una alternativa rischiosa può essere differente rispetto al valore atteso. La certezza equivalente per una alternativa è definita come una quantità certa preferibile all'alternativa (viene anche chiamata prezzo di vendita).

L'attitudine al rischio può essere di tre tipologie:

- Risk Averse: $CE < EV$
- Risk Neutral: $CE = EV$
- Risk Seeking: $CE > EV$

Una Utility Function traduce il risultato in numeri tali per cui il valore atteso dei numeri di utility possono essere utilizzati per calcolare la Certainty Equivalent per le alternative in termini consistenti con una attitudine del decision maker tendente al rischio.

La Exponential Utility Function è una famosa funzione utilizzata per calcolare il rischio di tolleranza; la funzione è definita come $u(x) = 1 - e^{-x/R}$, con $R > 0$ e nota. Se il rischio di tolleranza R aumenta la Utility Function sarà meno avversa al rischio. Per ogni utility function esiste una specifica certanty equivalente; in questo caso, $CE = -Rln(1 - E[U])$.

La differenza tra Expected Value ed il Certainty Equivalent può essere definito come il livello di rischio nel prendere una decisione rispetto ad un altra.

5.2 Value of Information

L'informazione perfetta rimuove tutta l'incertezza riguardo il risultato delle alternative ed è definita come l'estremo superiore del valore della informazione. Un parametro importante è chiamato valore atteso della informazione, definito come il massimo profitto atteso con informazione sommato al costo della consulenza sottratto al massimo profitto atteso senza informazione. Di conseguenza, è utile avere più informazione possibile in modo da migliorare il valore previsto di profitto.