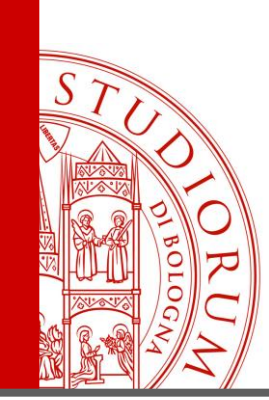


Ricerca Operativa

Supporto alle Decisioni: Introduzione all'Ottimizzazione

Marco A. Boschetti

Università di Bologna

Introduzione

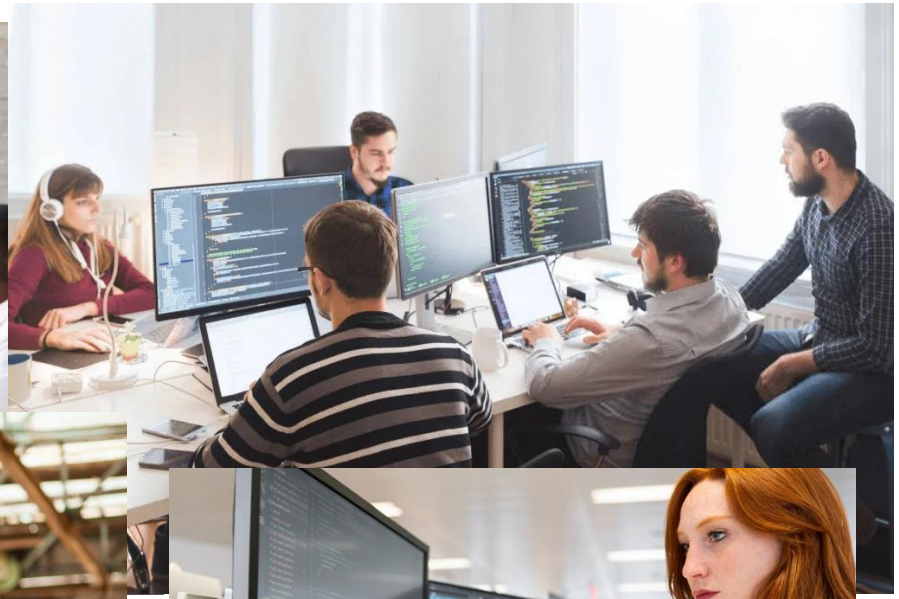
- I metodi di ottimizzazione hanno un'ampia gamma di applicazioni e tra i diversi ambiti abbiamo il Supporto alle Decisioni.
- In questa introduzione forniremo il contesto generale e degli esempi di applicazioni. Lo scopo è quello di motivare quanto vedremo nelle lezioni successive, che saranno necessariamente più impegnative.
- Il corso si concentrerà sui metodi dell'ottimizzazione, ma in questa introduzione proveremo a definire le possibili interrelazioni con altri ambiti.
- Ricordate che la matematica non è una punizione per chi si è iscritto a un scorso di laurea scientifico, ma uno strumento di lavoro molto prezioso.

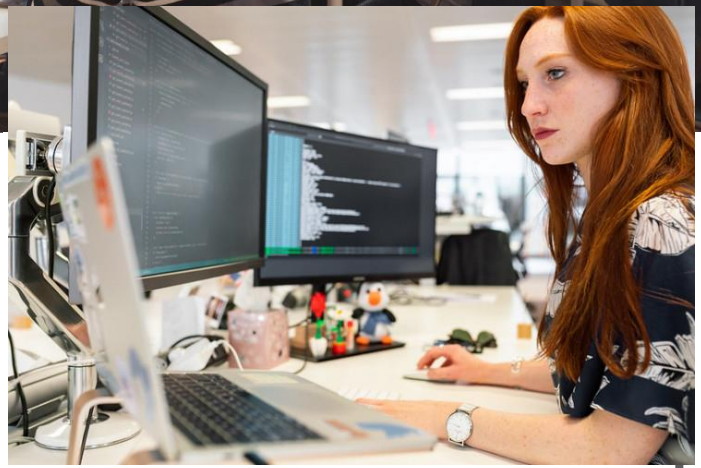
Introduzione

- Quando si parla di Information Communication Technology (ICT), spesso i non addetti ai lavori immaginano un mondo popolato da "nerd" estremamente esperti nell'uso delle tecnologie.

Introduzione

- La realtà è molto diversa e molto più articolata.

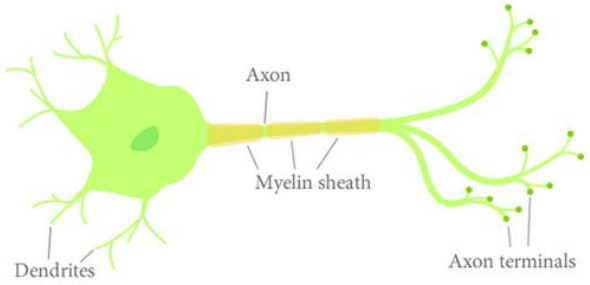




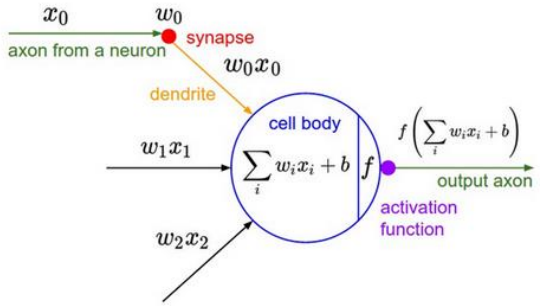
Introduzione

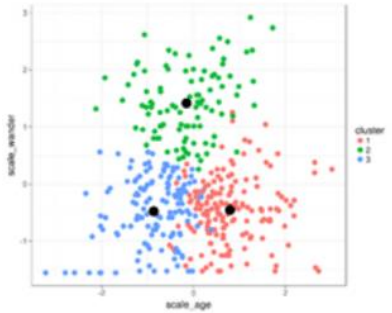
- Dietro all'Intelligenza Artificiale c'è tanta matematica… e molta ottimizzazione.

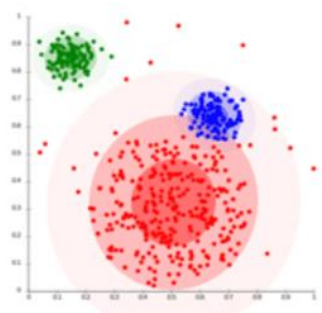
Real Neuron

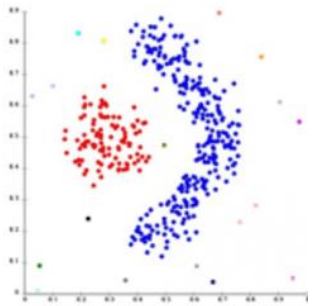


Artificial Neuron

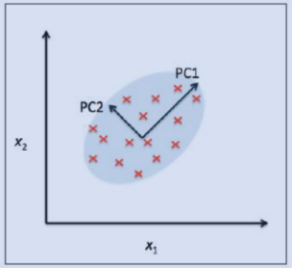


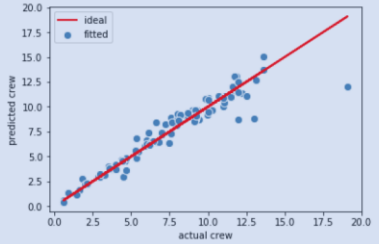






$X_1^{(1)}$	$X_2^{(1)}$	$X_3^{(1)}$	$X_4^{(1)}$
$X_1^{(2)}$	$X_2^{(2)}$	$X_3^{(2)}$	$X_4^{(2)}$
$X_1^{(3)}$	$X_2^{(3)}$	$X_3^{(3)}$	$X_4^{(3)}$
.	.	.	.
.	.	.	.
.	.	.	.
$X_1^{(n)}$	$X_2^{(n)}$	$X_3^{(n)}$	$X_4^{(n)}$





$$\sigma_{jk} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{X_j^{(i)} - \mu_j}{\sigma_j}\right)\left(\frac{X_k^{(i)} - \mu_k}{\sigma_k}\right) \qquad MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \qquad \boldsymbol{P(B/A) = \frac{P(B) \times P(A/B)}{P(A)}}$$

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} & \sigma_{14} \\ \sigma_{21} & \sigma_2^2 & \sigma_{23} & \sigma_{24} \\ \sigma_{31} & \sigma_{32} & \sigma_3^2 & \sigma_{34} \\ \sigma_{41} & \sigma_{42} & \sigma_{43} & \sigma_4^2 \end{bmatrix} \longrightarrow \tilde{\Sigma} = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 \\ 0 & 0 & 0 & \lambda_4 \end{bmatrix} \qquad \frac{1}{f_1} = \frac{1}{2}\left(\frac{1}{precison} + \frac{1}{recall}\right) \qquad \hat{y}_i = \omega_0 + \sum_{j=1}^{4} X_{ij}\,\omega_j$$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} y_i$$

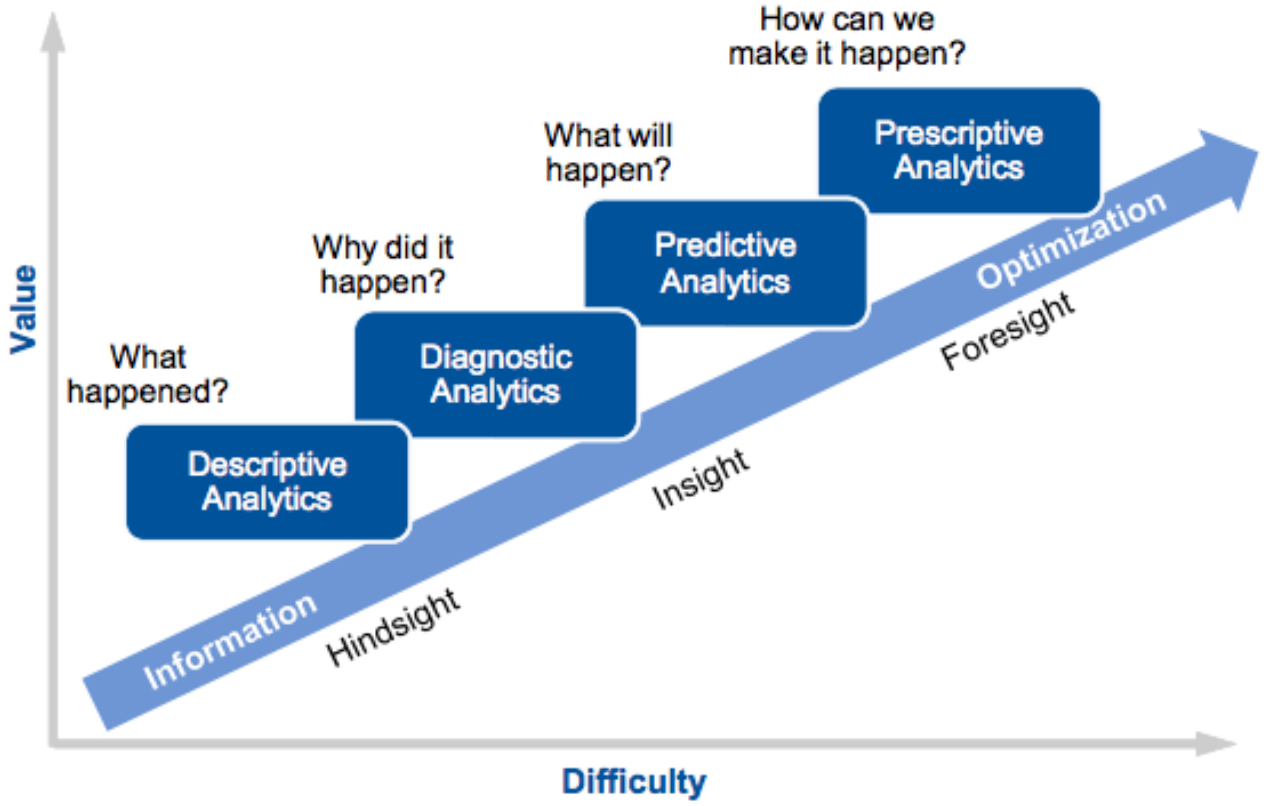
$$Sensitivity = \frac{TP}{TP + FN} \qquad \sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \mu)^2 \qquad R2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \qquad MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Introduzione

- Il processo di Trasformazione Digitale ha pervaso l'intera società e le aziende, per cui il ruolo dell'ICT è diventato centrale.
- Chi progetta e sviluppa soluzioni ICT è prima di tutto un consulente che deve saper risolvere problemi.
- Uno degli ambiti di applicazione dell'ICT è il supporto alle decisioni, che consiste nel realizzare soluzioni software che aiutano manager, operatori finanziari, medici, etc., a individuare la "miglior" decisione in ambito aziendale, finanziario, medico, etc.
- I metodi utilizzati per sviluppare soluzioni software per il supporto alle decisioni spaziano dall'ottimizzazione all'intelligenza artificiale e richiedono una conoscenza approfondita della matematica.
- L'ottimizzazione e l'intelligenza artificiale sono settori strettamente legati e spesso sovrapponibili, ma non coincidono.

Introduzione

- Fondamentale per il supporto alle decisioni è la relazione tra dati e ottimizzazione.



Source: Gartner (March 2012)

Introduzione

- In questo corso dimostreremo che per sviluppare una soluzione per il supporto alle decisioni sono necessari i seguenti passi:
  - modellare matematicamente il problema che si vuole risolvere;
  - identificare la sua complessità;
  - progettare l'algoritmo più adatto, usando le tecniche di soluzione in relazione al modello e alla complessità del problema.
- Mostreremo come problemi molto semplici da formulare possono risultare molto difficili da risolvere e come problemi molto complessi da descrivere possono essere in realtà molto facili da risolvere.
- Partiremo definendo cosa è un problema, un modello e un algoritmo.

Problema, Istanza, Modello e Algoritmo

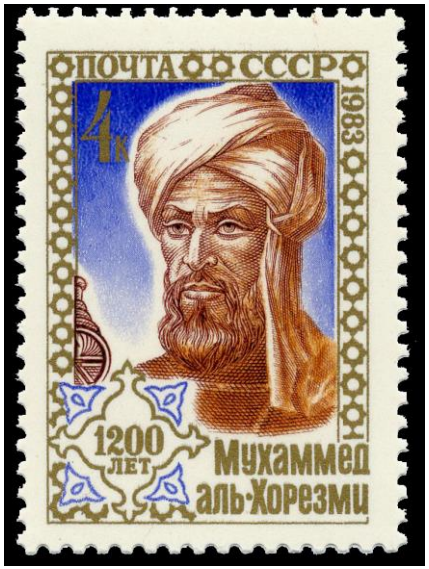
- Problema, *[pro-blè-ma] s.m. (pl. -mi)*
  - Garzanti: "*quesito con cui si chiede di trovare, con un procedimento di calcolo, uno o più dati sconosciuti, partendo dai dati noti contenuti nell'enunciato del quesito stesso*."
  - Coletti: "*in matematica e in altre scienze, domanda con cui si chiede di trovare, sulla base di dati noti ed enunciati, dati non noti, logicamente deducibili dai primi*."
- Una istanza rappresenta i dati noti del problema.
- Calcolare l'area di un triangolo è il problema, mentre il triangolo di altezza 20 cm e base 30 cm è l'istanza.
- Nel nostro contesto per risolvere un problema è necessario costruire un modello e definire un algoritmo di soluzione.

Algoritmo

- Algoritmo, da Treccani:

  "termine, derivato dall'appellativo al-Khuwārizmī ("originario della Corasmia") del matematico Muhammad ibn Mūsà del 9° sec., che designa qualunque schema o procedimento sistematico di calcolo (per es. l'a. euclideo, delle divisioni successive, l'a. algebrico, insieme delle regole del calcolo algebrico ecc.).

  Con un algoritmo si tende a esprimere in termini matematicamente precisi il concetto di procedura generale, di metodo sistematico valido per la soluzione di una certa classe di problemi."



Modelli

- I modelli sono una rappresentazione, spesso semplificata, di concetti, fenomeni, relazioni, strutture, processi, sistemi, etc.
- I modelli possono riguardare sia aspetti puramente teorici che del mondo reale.
- I modelli sono di tipo verbale, grafico, fisico, matematico, etc.
- I modelli consentono di perseguire i seguenti obiettivi:
  - facilitare la comprensione, identificando le componenti fondamentali e i meccanismi di funzionamento;
  - spiegare, controllare e predire eventi, anche sulla base delle osservazioni passate;
  - supportare i processi decisionali.

Modello Matematico

- Molto spesso ciò che deve essere rappresentato nel modello può essere composto da molte componenti le cui interrelazioni possono essere anche molto complesse. In questo caso è fondamentale applicare delle *semplificazioni* al modello, che possibilmente non "*interferiscano*" con l'utilizzo che se ne vuole fare.
- Il processo di semplificazione deve identificare le componenti di interesse e definire eventuali approssimazioni.
- Lo strumento di modellazione a cui siamo interessati è il modello matematico, che consente di rappresentare la realtà di interesse con un alto grado di astrazione per mezzo di simboli, equazioni, etc.
- Per scrivere i modelli matematici, come per analizzarli e risolverli, bisogna conoscere il linguaggio e i metodi della matematica.
- I modelli matematici che utilizzeremo prevedono l'uso di parametri che rappresentano l'input e di variabili che rappresentano l'output.

Modello Matematico

- Nel caso in cui i parametri possono essere definiti con esattezza a priori parleremo di modelli deterministici, mentre nel caso in cui i parametri non sono certi a priori parleremo di modelli stocastici.
- In molte applicazioni, come in economia e finanza, molto spesso i modelli sono di tipo stocastico.
- I modelli di tipo stocastico sono solitamente difficili da risolvere e spesso ci si riporta a modelli deterministici, ad esempio enumerando sottoinsiemi di scenari rappresentativi della realtà di interesse.
- Nella definizione e nell'uso dei modelli non bisogna dimenticarsi dell'aspetto numerico, perché l'approssimazione dei parametri e gli errori nel calcolo possono inficiare la validità dell'output.
- La matematica permette di prevedere anche la precisione con cui sarà fornito l'output. Di questo tema se ne occupa l'analisi numerica.

Approssimazione di un Modello

- **Prima del calcolo:**
  - modello;
  - misure empiriche;
  - precedenti calcoli.
- **Durante il calcolo:**
  - troncamento o discretizzazione;
  - arrotondamento.

NOTA: l'accuratezza del risultato finale dipende da tutti questi aspetti.

Errori di Troncamento e Arrotondamento

- **Errore di Troncamento**

  L'errore di troncamento è dato dalla differenza tra il "*risultato reale*" e quello prodotto dall'algoritmo. Per esempio, può essere dovuto alla terminazione di un algoritmo di soluzione di tipo iterativo prima del raggiungimento del valore reale.

- **Errore di Arrotondamento**

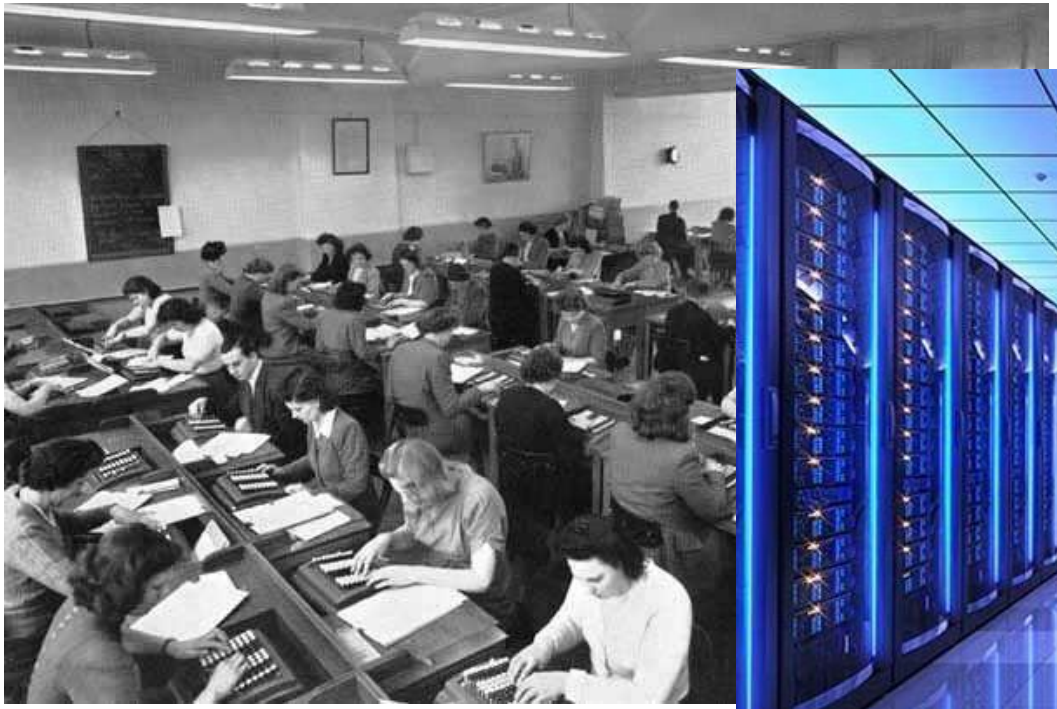
  L'errore di arrotondamento è dato dalla differenza tra il risultato prodotto da un dato algoritmo usando un'aritmetica esatta e il risultato prodotto dallo stesso algoritmo usando un'aritmetica approssimata. L'aritmetica del "*computer*" è approssimata a causa dell'inesatta rappresentazione dei numeri reali e dall'applicazione degli operatori aritmetici ad essi.

NOTA: gli errori di calcolo sono la somma di errori di troncamento e di arrotondamento.

Approssimazione Durante il Calcolo

- L'approssimazione durante il calcolo spesso è inevitabile.

Ieri…



Oggi…

Approssimazione di un Modello

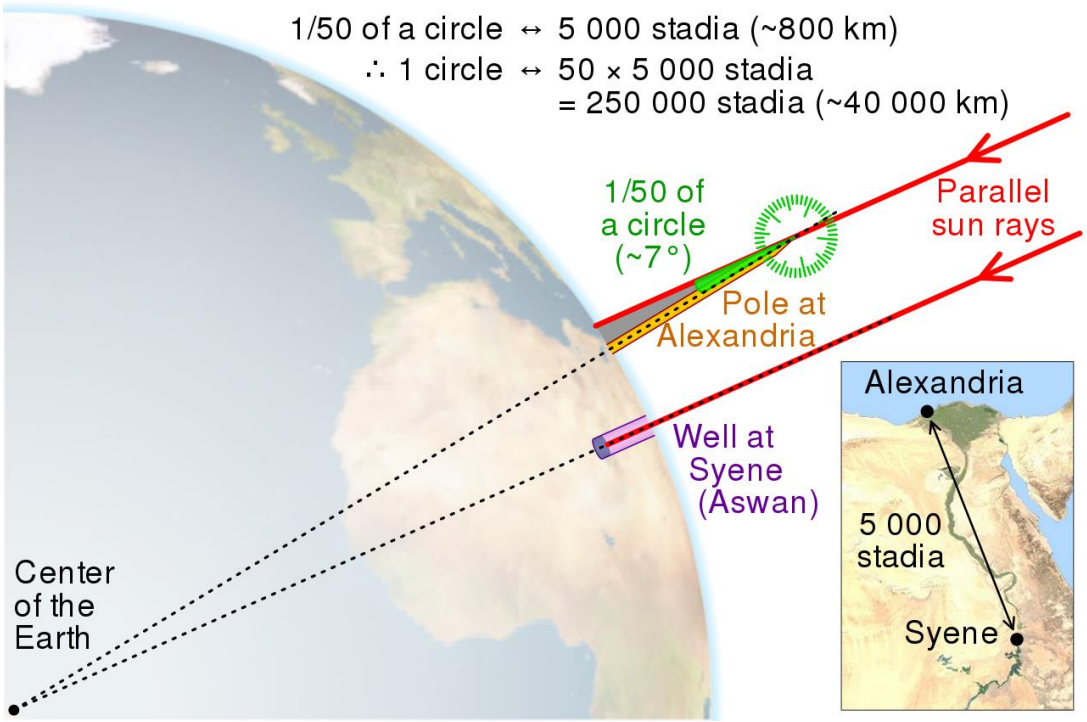
Esempio

- Il calcolo della superficie della terra usando la formula $A = 4\pi r^2$ implica le seguenti approssimazioni:
  - la terra viene modellata (o modellizzata) come una sfera;
  - il valore del raggio r è dato da misure empiriche e precedenti calcoli (come?);
  - il valore di π richiede un troncamento;
  - i valori dei dati in input e i risultati delle operazioni aritmetiche sono arrotondati dal computer.

Approssimazione di un Modello

Il calcolo del raggio terrestre:

- Eratostene (Cirene, 276 A.C. – Alessandria d'Egitto, 194 A.C.) riuscì a stimare con buona precisione il raggio terrestre:

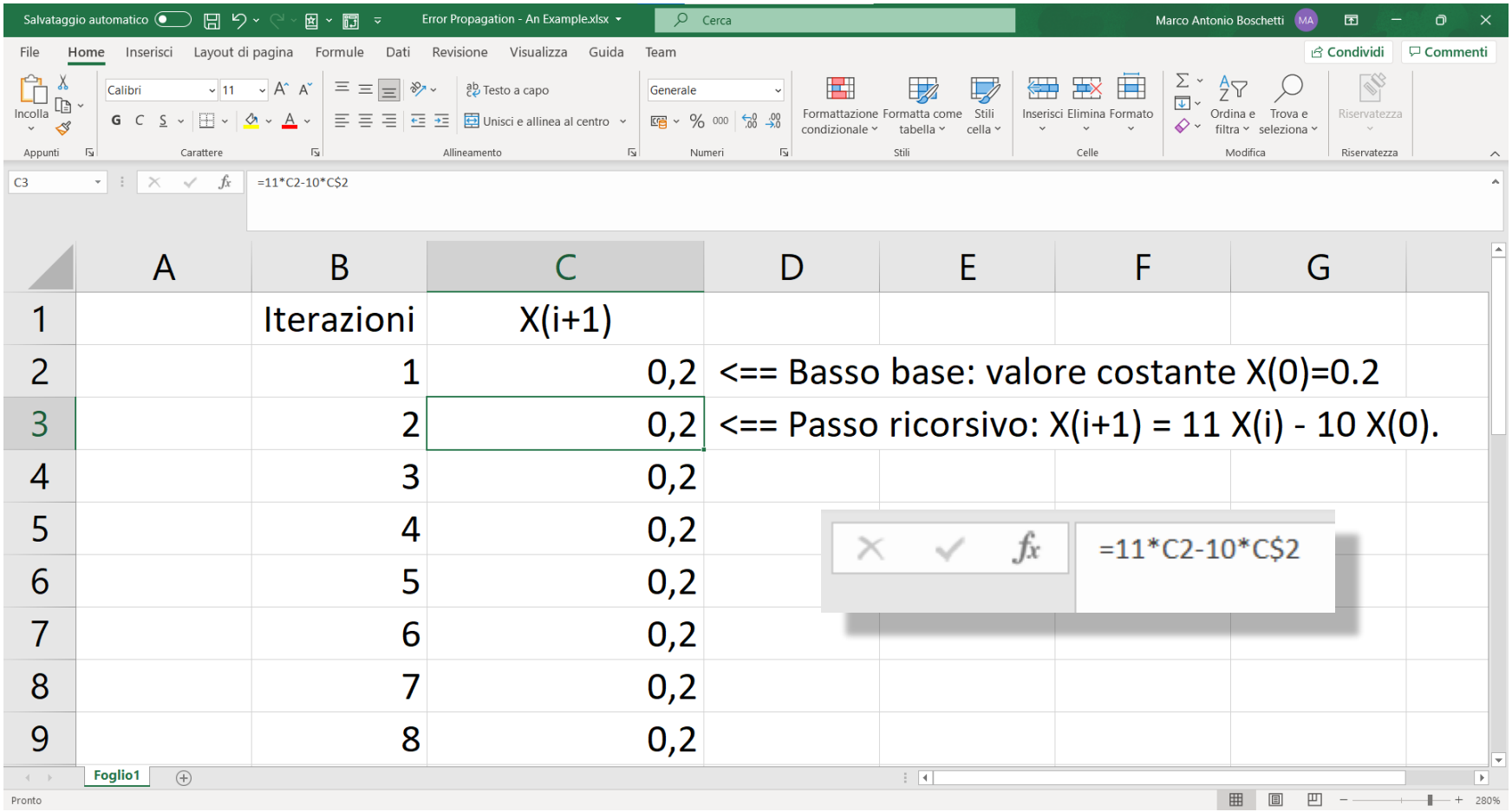


Fonte: https://en.wikipedia.org/wiki/Eratosthenes

Errori di Calcolo: un Esempio

- Gli errori di calcolo possono "*propagarsi*" in modo inaspettato e a volte disastroso. Il risultato potrebbe essere inutilizzabile.
- Vediamo un esempio molto semplice utilizzando un "*foglio elettronico*", uno strumento di lavoro molto usato in ambito scientifico/economico.
- Vogliamo calcolare con Excel (o una applicazione analoga) la seguente "*formula ricorsiva*":
  - Passo base: $x_0 = 0.2$
  - Passo ricorsivo: $x_{i+1} = 11x_i - 10x_0$
- Per induzione possiamo dedurre che $x_i = x_0$ per ogni i:
  - Passo base: $x_1 = 11x_0 - 10x_0 = x_0$.
  - Passo induttivo: se è vero fino a i, allora $x_{i+1} = 11x_0 - 10x_0 = x_0$.
- Ma cosa accade se calcolo la formula ricorsiva usando Excel? Otterremo $x_i = x_0 = 0.2$ per ogni i?

Errori: un esempio



Errori: un esempio

C3 | =11*C2-10*C$2

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		Iterazioni	X(i+1)										
2		1	0,2	<== Basso base: valore cos									
3		2	0,2	<== Passo ricorsivo: X(i+1)									
4		3	0,2										
5		4	0,2										
6		5	0,2										
7		6	0,2										
8		7	0,2										
9		8	0,2										
10		9	0,200000003										
11		10	0,200000038										
12		11	0,200000419										
13		12	0,200004607										
14		13	0,200050682										
15		14	0,200557497										
16		15	0,206132466										
17		16	0,267457121										
18		17	0,942028336										
19		18	8,362311691										
20		19	89,9854286										
21		20	987,8397146										
22		21	10864,23686										
23		22	119504,6055										
24		23	1314548,66										
25		24	14460033,26										

7	6	0,2	
8	7	0,2	
9	8	0,2	
10	9	0,200000003	
11	10	0,200000038	
12	11	0,200000419	
13	12	0,200004607	
14	13	0,200050682	
15	14	0,200557497	
16	15	0,206132466	
17	16	0,267457121	
18	17	0,942028336	
19	18	8,362311691	
20	19	89,9854286	
21	20	987,8397146	
22	21	10864,23686	
23	22	119504,6055	
24	23	1314548,66	
25	24	14460033,26	

Foglio1

Malcondizionamento di un'Istanza

- Consideriamo il problema della soluzione di un sistema di equazioni lineari:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

- Una particolare istanza può essere:

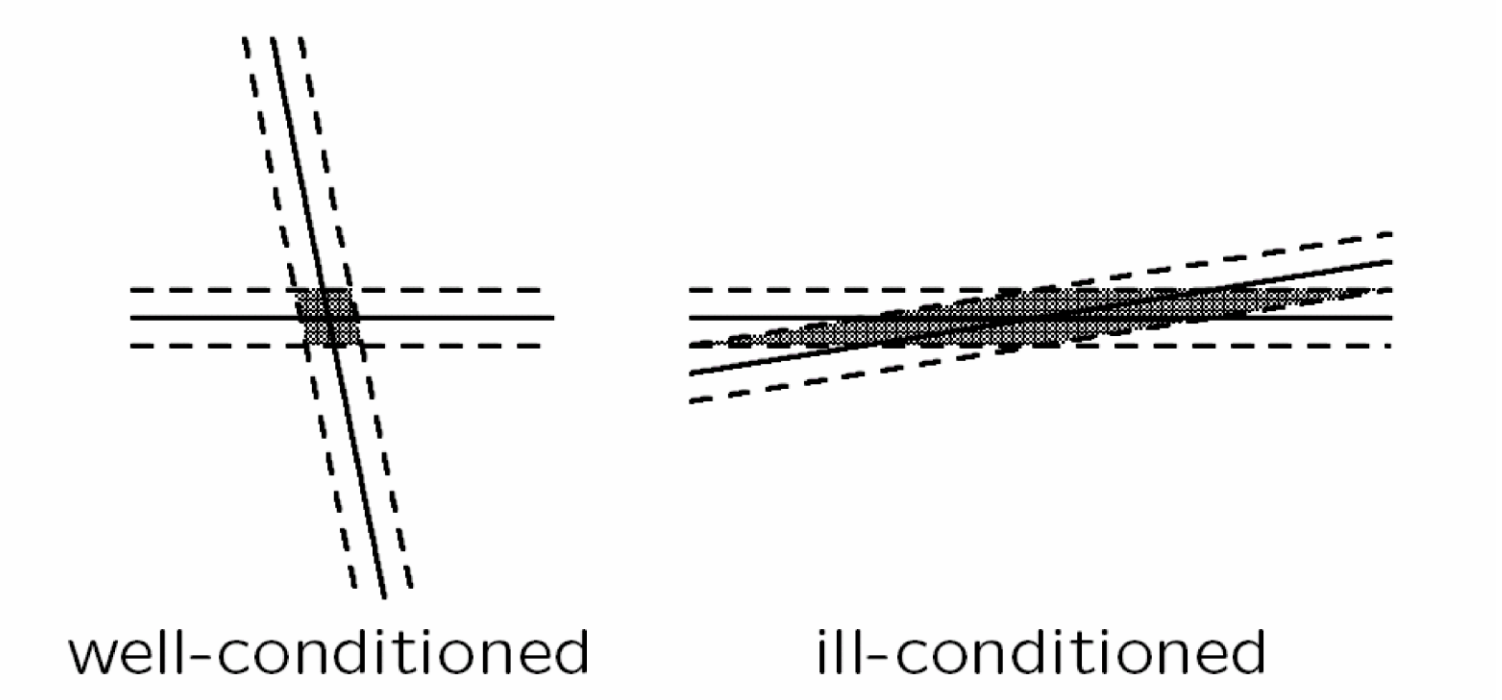
$$\begin{cases} x_1 + x_2 = 1 \\ x_1 - x_2 = 2 \end{cases}$$

  Questo sistema lineare ha un'unica soluzione che rappresenta l'intersezione delle rette rappresentate dalle due equazioni.

- Questo problema è semplice, ma per alcune istanze può essere difficile ottenere una soluzione sufficientemente accurata.

Malcondizionamento di una Istanza

- Più le rette tendono a essere parallele e più vengono amplificati gli errori (peggiora il "*condizionamento*" dell'istanza di un problema).



Tipologie di Problemi

- Distinguiamo le seguenti tipologie di problemi:
  - Problema di "*Decisione*" (o problema decisionale): data un'istanza si vuole determinare se esiste o meno una certa soluzione;
  - Problema di "*Ricerca*": data un'istanza si vuole determinare una possibile soluzione;
  - Problema di "*Enumerazione*": data un'istanza si vuole determinare tutte le possibili soluzioni;
  - Problema di "*Ottimizzazione*": data un'istanza si vuole determinare la "migliore soluzione" possibile rispetto ad una misura fissata.
- **Quali sono dei possibili esempi?**

Tipologie di Problemi

Esempio: Traveling Salesman Problem (TSP)

A un mezzo di un corriere sono stati assegnati dei pacchi da consegnare:

- Problema di "*Decisione*" : si vuole sapere se è possibile svolgere tutte le consegne in una giornata lavorativa (e.g., entro le 9 ore di lavoro);
- Problema di "*Ricerca*": si vuole determinare una sequenza di visite che consente di consegnare tutti i pacchi entro la giornata lavorativa;
- Problema di "*Enumerazione*": si vogliono enumerare tutte le possibili soluzioni che consentono la consegna di tutti i pacchi;
- Problema di "*Ottimizzazione*": si vuole determinare la sequenza delle visite che minimizza una funzione obiettivo (e.g., tempo impiegato, chilometri percorsi, etc.).

Tipologie di Problemi

Esempio: Traveling Salesman Problem (TSP)

Una soluzione del TSP è definita dall'ordine con cui si dovrà visitare le n fermate (per esempio, per permettere al corriere di consegnare i pacchi).

Il numero delle possibili soluzioni alternative per un TSP che prevede n fermate potrebbe essere pari a:

$$n! = 1 \times 2 \times 3 \times 4 \times 5 \times \cdots \times n$$

Esempi:

$n = 5 \implies 5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

$n = 10 \implies 10! = 1 \times 2 \times \cdots \times 9 \times 10 = 3.628.800$

$n = 20 \implies 20! = 1 \times 2 \times \cdots \times 19 \times 20 = 2.432.902.008.176.640.000$

$n = 100 \implies 100! = 1 \times 2 \times \cdots \times 99 \times 100 = 9{,}33262 \times 10^{157}$

$n = 200 \implies 200! = 1 \times 2 \times \cdots \times 199 \times 200 = 7{,}88658 \times 10^{374}$

Tipologie di Problemi

Problemi non decidibili

- Se consideriamo i soli problemi decisionali, che richiedono un Si o un No come risposta, tra questi ve ne sono alcuni che non sono decidibili.
- Per esempio, non riusciamo a decidere se le seguente frase è vera:

  "Questa frase è falsa"

- Questa frase deriva dal "paradosso di Epimenide", noto anche come "paradosso del mentitore".
- In origine l'affermazione di Epimenide, che era cretese, era:

  "Tutti i Cretesi sono bugiardi"

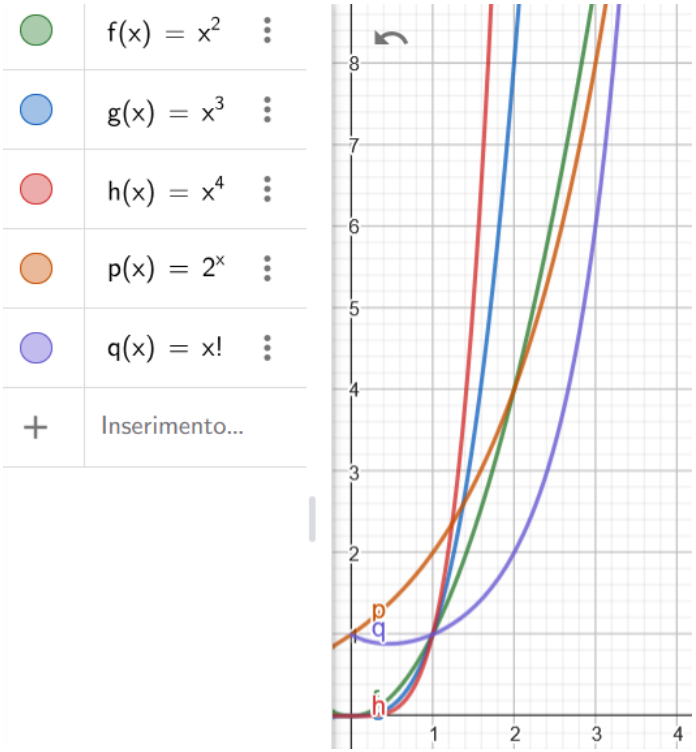
  Essendo Epimenide cretese avrebbe dovuto essere bugiardo e perciò la frase dovrebbe essere falsa poiché pronunciata da un bugiardo.

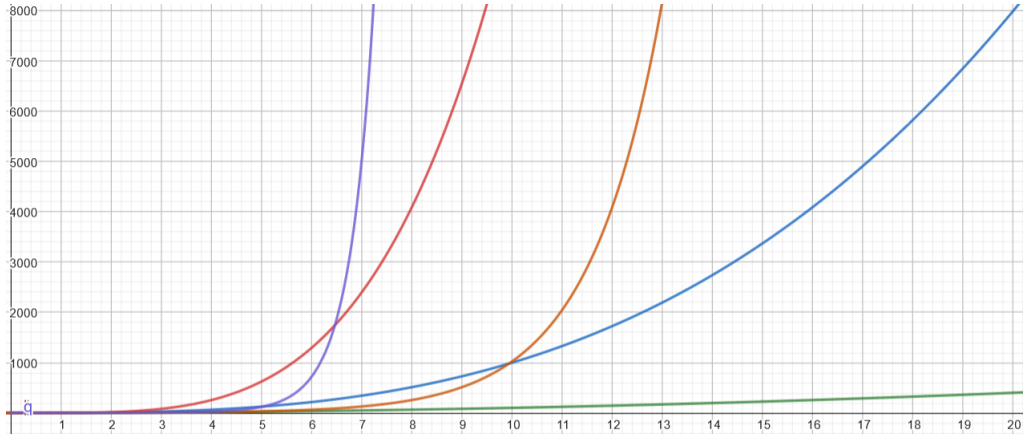
Complessità Computazionale

- La teoria della complessità computazionale è un ambito di ricerca della matematica, oggi spesso denotato come "*informatica teorica*", che ha l'obiettivo di classificare i problemi secondo la loro intrinseca complessità.
- Banalizzando, la teoria della complessità ci consente di determinare se un problema è "facile" o "difficile".
- La difficoltà di un problema è una sua caratteristica generale che non è associata a una particolare istanza. Ricordiamo che un problema è un'entità astratta, mentre un'istanza è un caso particolare.
- Potremo dire che un algoritmo risolve un problema se è in grado di generare una soluzione per ogni possibile istanza.
- Se un problema è semplice possiamo garantire la sua soluzione in un tempo e una quantità di risorse (e.g., memoria) "ragionevole".
- Se un problema è difficile non possiamo garantire la sua soluzione.

Complessità Computazionale

- Funzioni polinomiali, esponenziali e fattoriali…





Complessità Computazionale

- Volendo semplificare un argomento in verità molto più complesso, potremo dire che:
  - I problemi semplici sono quelli polinomiali (insieme $\boldsymbol{P}$).
  - I problemi difficili sono quelli "non-polinomiali" (che si trovano nell'insieme $\boldsymbol{NP}$), tra cui ci sono i problemi $\boldsymbol{NP}$**-Completi**.
- Si noti che $\boldsymbol{NP}$ significa Nondeterministic Polynomial time.
- $\boldsymbol{NP}$ è l'insieme dei problemi decisionali in cui se la risposta è "Si" lo si può verificare in tempo polinomiale utilizzando una macchina di Turing deterministica, o in modo equivalente possiamo dire che il problema può essere risolto in tempo polinomiale da una macchina di Turing non-deterministica.
- Al momento possiamo facilmente dimostrare che $\boldsymbol{P} \subseteq \boldsymbol{NP}$, ma non sappiamo se $\boldsymbol{P} \neq \boldsymbol{NP}$.

Complessità Computazionale

- Un problema decisionale $f: \mathbb{I} \longrightarrow \{0,1\}$ è *riducibile polinomialmente* a $g: \mathbb{I} \longrightarrow \{0,1\}$, i.e. $f \propto g$, se esiste una funzione h, calcolabile in tempo polinomiale, tale che per ogni x:

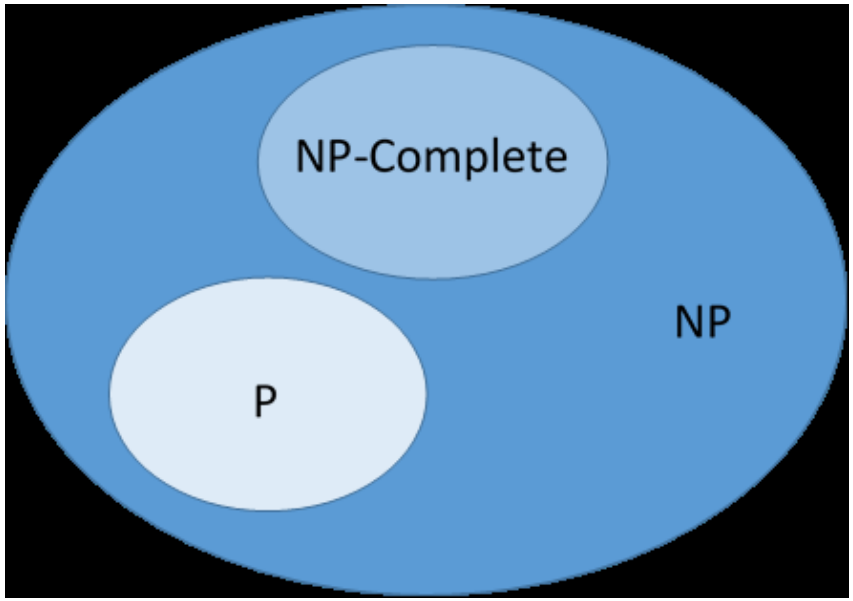
$$f(x) = g(h(x))$$

- Un problema decisionale $p: \mathbb{I} \longrightarrow \{0,1\}$ è definito ***NP*-Completo** se e solo se:
  - $p \in \boldsymbol{NP}$;
  - per ogni $f \in \boldsymbol{NP}$ si ha $f \propto p$.
- Fino ad oggi non è stato dimostrato che l'insieme dei problemi $\boldsymbol{P}$ e l'insieme dei problemi $\boldsymbol{NP}$-Completi sono disgiunti. Se così non fosse, allora $\boldsymbol{P} = \boldsymbol{NP}$.
- Un problema di ottimizzazione è ***NP*-Hard** se esiste un problema $\boldsymbol{NP}$-Completo che può essere ridotto ad esso con trasformazione polinomiale.

Complessità Computazionale

Millennium Prize Problems

Il Clay Mathematics Institute lo ha incluso tra i 7 problemi matematici del millennio per i quali è previsto un premio di 1,000,000 di dollari per chi fornirà la dimostrazione corretta per primo.



Problemi vs Algoritmi

- In letteratura può capitare che per un dato problema siano proposti più algoritmi. Perché?
- Le prestazioni di un algoritmo possono essere misurate con:
  - il tempo di calcolo richiesto per ottenere la soluzione;
  - la qualità della soluzione (i.e., quanto il risultato ottenuto approssima il risultato corretto).
- Per ogni istanza di un problema esistono algoritmi che forniscono prestazioni migliori di altri.
- Per i problemi difficili c'è un ulteriore opzione:
  - Algoritmi euristici, che trovano soluzioni di "buona qualità";
  - Algoritmi esatti, che trovano le soluzioni "ottime".
- Per la scelta dell'algoritmo è determinante definire la complessità computazionale del problema. Perché?

Modelli vs Algoritmi

- Il modello scelto per un dato problema ha un impatto determinante sulla scelta dell'algoritmo più adatto.
- Si potrebbe scegliere un modello che approssima maggiormente il problema che può essere risolto in modo molto efficiente/efficace.
- Mentre modelli che descrivono in modo molto preciso il problema potrebbero essere troppo difficili da risolvere.
- Come si fa a scegliere?
- Quali dovrebbero essere i criteri di scelta?
- La stragrande maggioranza degli algoritmi di soluzione nell'ambito dell'Intelligenza Artificiale (AI) sono classificabili come algoritmi euristici.
- Nell'ambito dell'ottimizzazione molti modelli possono essere risolti in modo esatto (algoritmi esatti).

Ricerca Operativa

- Quando dobbiamo risolvere un problema dobbiamo costruire un modello che sia risolvibile da un algoritmo adeguato alla complessità del problema e alla tipologia di istanze che dobbiamo risolvere.
- La disciplina che si occupa dei problemi di ottimizzazione è nota come operations research (ricerca operativa) e fa parte del settore decision science (scienze decisionali).
- Le keyword di moda oggi che identificano i campi di applicazione sono business analytics, data science, machine learning, etc.
- Per esempio, nell'ambito della gestione aziendale (management science) l'uso della matematica consente di definire degli algoritmi per il supporto alle decisioni.
- In questa prima introduzione vedremo alcuni esempi molto semplici di applicazione della programmazione matematica e ottimizzazione combinatoria, argomenti che poi approfondiremo meglio.

Ottimizzazione Non-Vincolata

Esempio: Domanda e Ricavo

- Una casa editrice prevede che l'equazione della domanda relativa alle vendite del suo ultimo romanzo sia:

$$q = -2000p + 150000$$

dove q è il numero di libri che si prevede di vendere ogni anno se il prezzo di vendita è di p Euro (come è stato costruito il modello?).

- Quale prezzo deve applicare la casa editrice per ottenere il massimo del ricavo annuo?
- Il ricavo è dato dal prodotto tra il prezzo e la quantità venduta:

$$R = pq$$

che sostituendo l'equazione relativa alla domanda diventa:

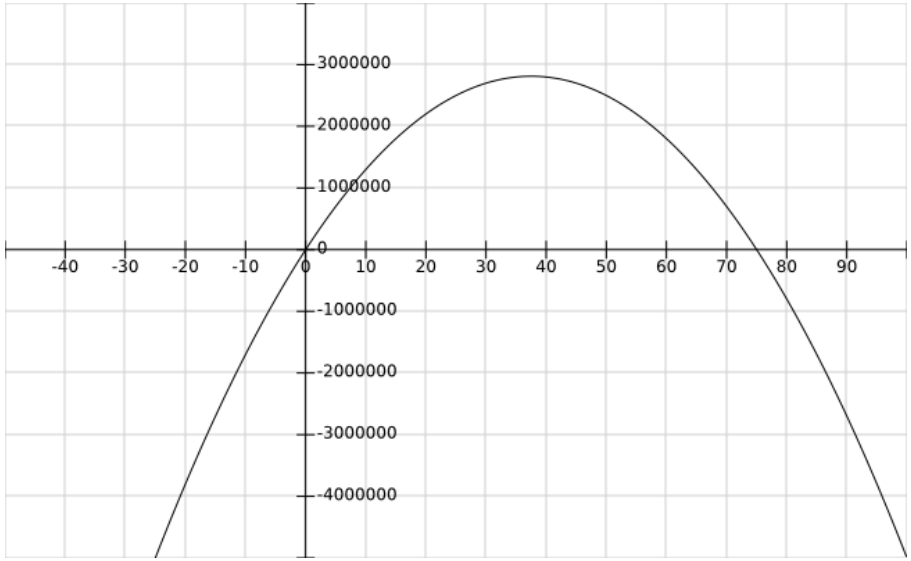
$$R = pq = p(-2000p + 150000) = -2000p^2 + 150000p$$

Ottimizzazione Non-Vincolata

- Il ricavo è in funzione del prezzo ed è dato da:

$$R(p) = -2000p^2 + 150000p$$

- Per cui, dobbiamo trovare il prezzo p che massimizza il ricavo $R(p)$.
- Possiamo notare che si tratta dell'equazione di una parabola e che il coefficiente del termine quadratico è -2000, ossia è negativo. Quindi la funzione è concava.



Ottimizzazione Non-Vincolata

- Il prezzo che bisogna applicare per ottenere il massimo ricavo annuo corrisponde al vertice della parabola (si potevano usare le derivate).
- Per cui, se l'equazione è:

$$R = ap^2 + bp + c = -2000p^2 + 150000p$$

abbiamo che $a = -2000$, $b = +150000$ e $c = 0$.

- L'ascissa del vertice è data da:

$$p = -\frac{b}{2a} = -\frac{150000}{2(-2000)} = +\frac{150000}{4000} = 37.5$$

Quindi il prezzo che consente il massimo ricavo è 37.50 Euro.

- Se vogliamo conoscere il corrispondente ricavo è sufficiente calcolare:

$$R = -2000p^2 + 150000p = -2000(37.5)^2 + 150000(37.5)$$

- La soluzione ottima del problema è $p = 37.5$ e il corrispondente ricavo $R = 2812500$. Si noti che è il massimo della funzione $R(p)$.

Ottimizzazione Non-Vincolata

- In generale cosa significa il termine "ottimizzazione non-vincolata"?
- Che cosa è il minimo relativo (o massimo relativo) di una funzione?
- Che cosa è il minimo assoluto (o massimo assoluto) di una funzione?
- Data una generica funzione $f: \mathbb{R}^n \longrightarrow \mathbb{R}$ come faccio a trovare il suo minimo/massimo relativo/assoluto?
- Il metodo da impiegare dipende dalla funzione f?
- Quale algoritmo suggerite di utilizzare per risolvere un problema di ottimizzazione non-vincolata?
- Durante il corso vedremo come fare.

Ottimizzazione Vincolata

Esempio: Miscelazione

- L'azienda Artic Juice Company produce tre miscele di succo che per ogni litro richiedono la seguente composizione:
  - PineOrange: 2 parti di succo d'ananas e 2 parti di succo d'arancia;
  - PineKiwi: 3 parti di succo d'ananas e 1 parte di succo di kiwi;
  - OrangeKiwi: 3 parti di succo d'arancia e 1 parte di succo di kiwi.
- L'azienda dispone ogni giorno di 800 parti di succo d'ananas, 650 parti di succo d'arancia e 350 parti di succo di kiwi.
- Quanti litri di ciascuna miscela deve produrre per utilizzare tutte le materie prime?
- Vogliamo modellare il problema e risolverlo.

Ottimizzazione Vincolata

- Il primo passaggio è l'identificazione delle variabili decisionali:
  - x_1: numero di litri di PineOrange prodotti in un giorno;
  - x_2: numero di litri di PineKiwi prodotti in un giorno;
  - x_3: numero di litri di OrangeKiwi prodotti in un giorno.
- Le informazioni che abbiamo a disposizione sono le seguenti:

	x_1	x_2	x_3	Totale disponibile
Succo Ananas (parti)	2	3	0	800
Succo Arancia (parti)	2	0	3	650
Succo Kiwi (parti)	0	1	1	350

- Ora bisogna scrivere il sistema lineare e risolverlo.

Ottimizzazione Vincolata

- Il sistema lineare risultante è:

$$\begin{cases} 2x_1 + 3x_2 = 800 \\ 2x_1 + 3x_3 = 650 \\ +1x_2 + 1x_3 = 350 \end{cases}$$

- Si può risolvere il sistema lineare con il metodo di eliminazione di Gauss-Jordan oppure uno dei tanti algoritmi alternativi.

- La soluzione del sistema lineare è:

$$(x_1, x_2, x_3) = (100, 200, 150)$$

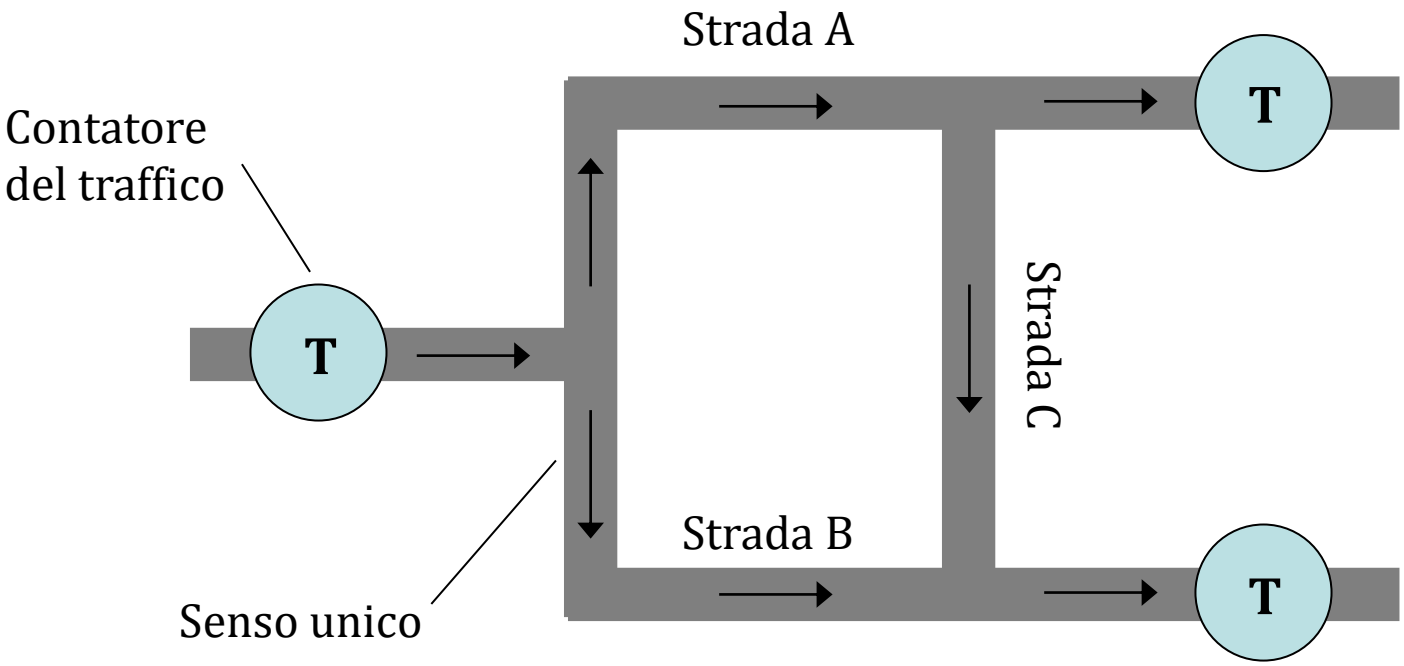
che suggerisce di produrre ogni giorno 100 litri di PineOrange, 200 litri di PineKiwi e 150 litri di OrangeKiwi per riuscire a consumare tutte le materie disponibili. Esiste una soluzione più "*conveniente*"?

- Come può essere modellato il problema per un generico numero di materie prime e di prodotti (ognuno con la sua ricetta specifica)?

Ottimizzazione Vincolata

Esempio: Flussi di Traffico

- Il traffico che attraversa un quartiere del centro di una cittadina rispetta il seguente sistema di sensi unici:

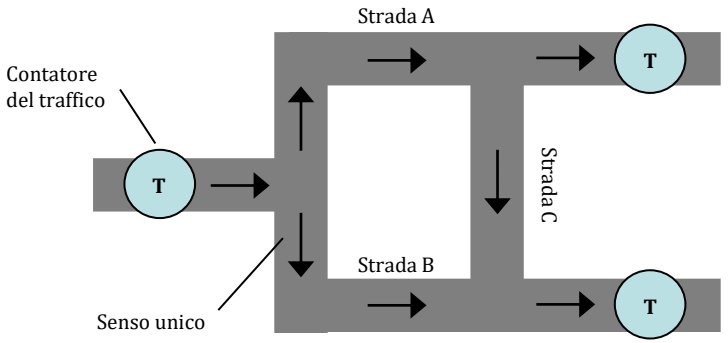


Ottimizzazione Vincolata

- I tre contatori del traffico rilevano i seguenti dati:
  - Contatore a Ovest: 200 automobili in entrata ogni ora;
  - Contatori a Est: 100 automobili in uscita ogni ora ciascuno.
- Si può notare che il numero di automobili entrante nel quartiere (contatore a ovest) è pari al numero di automobili uscenti (somma dei contatori a est).
- Possiamo ipotizzare che il numero di auto che entrano nel quartiere e parcheggiano è compensato dal numero di auto che lasciano i parcheggi ed escono dal quartiere (è ragionevole?).
- In una rete stradale si applica il principio della conservazione del flusso: il numero di auto entranti in un incrocio e pari al numero delle auto uscenti (è ragionevole?).

Ottimizzazione Vincolata

- Con le informazioni a disposizione è possibile stabilire quante auto transitano ogni ora nelle strade A, B e C?
- Il primo passaggio è l'identificazione delle variabili decisionali:
  - x_1: numero di auto che transitano ogni ora nella Strada A;
  - x_2: numero di auto che transitano ogni ora nella Strada B;
  - x_3: numero di auto che transitano ogni ora nella Strada C.
- Applicando il principio di conservazione del flusso possiamo scrivere le seguenti equazioni:
  - $x_1 + x_2 = 200$
  - $x_1 - x_3 = 100$
  - $x_2 + x_3 = 100$



- Tutte queste equazioni devono essere soddisfatte.

Ottimizzazione Vincolata

- Il sistema lineare risultante è:

$$\begin{cases} 1x_1 + 1x_2 = 200 \\ 1x_1 - 1x_3 = 100 \\ +1x_2 + 1x_3 = 100 \end{cases}$$

- Se risolviamo il sistema lineare (e.g., con il metodo di eliminazione di Gauss-Jordan) scopriremo che una delle tre equazioni è ridondante (i.e., è combinazione lineare delle altre 2). Cosa significa?
- La soluzione del sistema lineare è:

$x_1 = x_3 + 100$

$x_2 = -x_3 + 100$

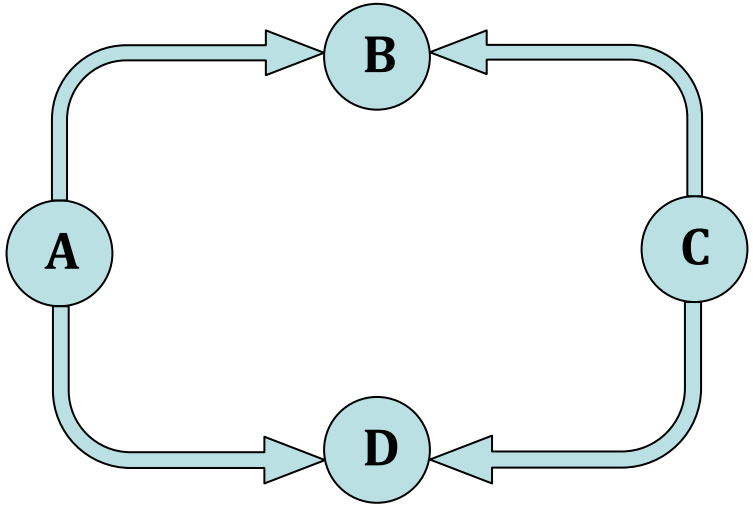
x_3 arbitrario

- I tre contatori non ci permettono di conoscere il flusso esatto sulle tre strade. Cosa potremo fare per riuscire a conoscere il flusso?

Ottimizzazione Vincolata

Esempio: Trasporti

- Un'azienda di noleggio di automobili ha quattro sedi in città:
  - Sede A: che ha 20 auto più del necessario;
  - Sede B: che necessita di 10 auto in più di quelle di cui dispone;
  - Sede C: che ha 15 auto più del necessario;
  - Sede D: che necessita di 25 auto in più di quelle di cui dispone.

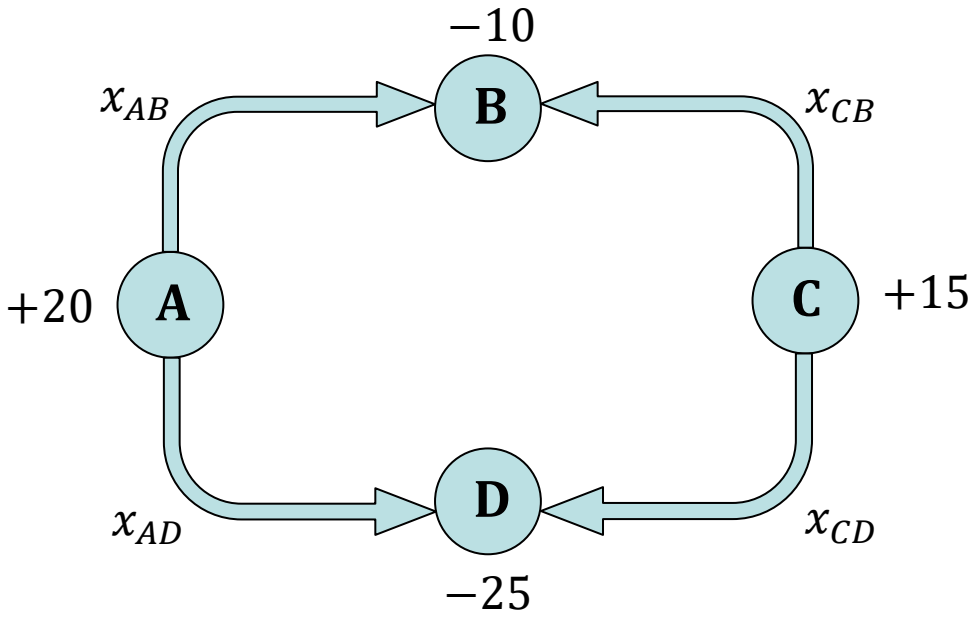


Ottimizzazione Vincolata

- Per spostare le auto da una sede all'altra si incorre in un costo dovuto al tempo speso da un dipendente alla guida e i costi di carburante e manutenzione per ogni chilometro percorso.
- I costi per spostare un'auto tra le diverse sedi sono i seguenti (come devono essere calcolati?):
  - Dalla Sede A alla Sede B costa complessivamente 10 Euro;
  - Dalla Sede C alla Sede B costa complessivamente 5 Euro;
  - Dalla Sede A alla Sede D costa complessivamente 20 Euro;
  - Dalla Sede C alla Sede D costa complessivamente 10 Euro.
- L'azienda di noleggio ha un budget per gli spostamenti di 475 Euro.
- Con il budget a disposizione è possibile spostare le auto tra le 4 sedi?

Ottimizzazione Vincolata

- Per modellare il problema possiamo definire le variabili decisionali:
  - x_{AB}: numero di auto spostate dalla Sede A alla sede B;
  - x_{AD}: numero di auto spostate dalla Sede A alla sede D;
  - x_{CB}: numero di auto spostate dalla Sede C alla sede B;
  - x_{CD}: numero di auto spostate dalla Sede C alla sede D;



Ottimizzazione Vincolata

- Sappiamo che la Sede A ha una disponibilità di 20 auto:

$$x_{AB} + x_{AD} = 20$$

- La Sede B richiede 10 auto:

$$x_{AB} + x_{CB} = 10$$

- La Sede C ha una disponibilità di 15 auto:

$$x_{CB} + x_{CD} = 15$$

- La Sede D richiede 25 auto:

$$x_{AD} + x_{CD} = 25$$

- Infine c'è il vincolo relativo al budget disponibile:

$$10x_{AB} + 20x_{AD} + 5x_{CB} + 10x_{CD} = 475$$

- Abbiamo 4 variabili e 5 equazioni. Questo cosa implica?

Ottimizzazione Vincolata

- Il sistema lineare risultante è:

$$\begin{cases} 1x_{AB} + 1x_{AD} = 20 \\ 1x_{AB} + 1x_{CB} = 10 \\ + 1x_{CB} + 1x_{CD} = 15 \\ + 1x_{AD} + 1x_{CD} = 25 \\ 10x_{AB} + 20x_{AD} + 5x_{CB} + 10x_{CD} = 475 \end{cases}$$

- Se risolviamo il sistema lineare, scopriremo che una equazione è ridondante e la soluzione è:

$$(x_{AB}, x_{AD}, x_{CB}, x_{CD}) = (5, 15, 5, 10)$$

- Nel caso la soluzione fosse frazionaria, come possiamo utilizzarla?
- Cosa potrebbe accadere se il budget fosse diverso?
- Come potremo risolvere il problema se volessimo spostare tutte le auto, ma minimizzando il budget necessario?

Ottimizzazione Vincolata

- Se il budget fosse diverso, la soluzione sarebbe cambiata e qualche variabile sarebbe potuta diventare frazionaria o negativa.
- Nel caso la soluzione fosse frazionaria, le auto non possono essere frazionate, per cui la soluzione non sarebbe applicabile. E quindi?
- Se volessimo spostare tutte le auto, ma minimizzando il budget necessario, dovremo eliminare il vincolo di budget e risolvendo otterremo la soluzione:

$$x_{AB} = x_{CD} - 5$$
$$x_{AD} = 25 - x_{CD}$$
$$x_{CB} = 15 - x_{CD}$$
x_{CD} è arbitrario.

Questa soluzione può essere sostituita nella funzione di costo.

Ottimizzazione Vincolata

- Siccome la funzione di costo è la seguente:

$$c(\mathbf{x}) = 10x_{AB} + 20x_{AD} + 5x_{CB} + 10x_{CD}$$

Se sostituiamo la $\mathbf{x}$ con la soluzione $\mathbf{x}'$ del sistema lineare:

$$\mathbf{x}' = (x_{AB}, x_{AD}, x_{CB}, x_{CD}) = (x_{CD} - 5, 25 - x_{CD}, 15 - x_{CD}, x_{CD})$$

Ottenendo la seguente espressione:

$$c(\mathbf{x}') = 10(x_{CD} - 5) + 20(25 - x_{CD}) + 5(15 - x_{CD}) + 10x_{CD}$$

$$c(\mathbf{x}) = (10 - 20 - 5 + 10)x_{CD} + (-50 + 500 + 75)$$

$$c(\mathbf{x}) = -5x_{CD} + 525$$

- Il costo diminuisce all'aumentare di x_{CD}. Siccome il massimo valore di x_{CD} è 15 (il numero di auto in più presso la Sede C), allora la "soluzione ottima" sarà $\mathbf{x}^* = (10, 10, 0, 15)$ e il costo corrispondente è $c(\mathbf{x}^*) = 100 + 200 + 0 + 150 = 450$ Euro.

Ottimizzazione Vincolata

- **Esempio:** L'azienda Acme produce succo di mela in due diverse concentrazioni:
  - Tipo 1: un litro è composto da 30 parti di acqua e 2 parti di succo di mela concentrato;
  - Tipo 2: un litro è composto da 20 parti di acqua e 12 parti di succo di mela concentrato.

  Ogni giorno l'azienda dispone di 30000 parti d'acqua e di 3600 parti di concentrato.

  Se l'azienda vuole usare tutta l'acqua e tutto il concentrato, quanti litri di ciascun tipo di succo deve produrre?

- **Primo passo:** dobbiamo identificare le incognite (i.e., le variabili).

  x_1: litri di succo di tipo 1 prodotti ogni giorno.

  x_2: litri di succo di tipo 2 prodotti ogni giorno.

Ottimizzazione Vincolata

- **Secondo passo:** dobbiamo definire i vincoli (in forma di equazioni) a cui sono soggette le variabili x_1 e x_2 utilizzando tutte le informazioni fornite nel testo:
  - Le parti d'acqua complessivamente disponibili sono 30000. Per ogni litro di succo di tipo 1 se ne usano 30 e per ogni litro di succo di tipo 2 se ne usano 20:
    $$30x_1 + 20x_2 = 30000$$
  - Le parti succo di mela concentrato complessivamente disponibili sono 3600. Per ogni litro di succo di tipo 1 se ne usano 2 e per ogni litro di succo di tipo 2 se ne usano 12:
    $$2x_1 + 12x_2 = 3600$$

Il sistema di equazioni lineari risultante è il seguente:

$$\begin{cases} 30x_1 + 20x_2 = 30000 \\ 2x_1 + 12x_2 = 3600 \end{cases}$$

Ottimizzazione Vincolata

- Per risolvere il sistema di equazioni lineari:

$$\begin{cases} 30x_1 + 20x_2 = 30000 \\ 2x_1 + 12x_2 = 3600 \end{cases}$$

possiamo iniziare svolgendo delle "*normalizzazioni*" dividendo il primo vincolo per 10 e il secondo per 2:

$$\begin{cases} 3x_1 + 2x_2 = 3000 \\ x_1 + 6x_2 = 1800 \end{cases}$$

- La soluzione è:

$$\begin{cases} x_2 = 150 \\ x_1 = 900 \end{cases}$$

che possiamo "*leggere*" come segue:

  - In un giorno devono essere prodotti 900 litri di succo di tipo 1;
  - In un giorno devono essere prodotti 150 litri di succo di tipo 2.

Ottimizzazione Vincolata

- Cosa sarebbe accaduto se avessimo ammesso la possibilità di non consumare tutta l'acqua e tutto il succo di mela concentrato?
- Il sistema di equazioni sarebbe diventato un sistema di disequazioni:

$$\begin{cases} 30x_1 + 20x_2 \leq 30000 \\ 2x_1 + 12x_2 \leq 3600 \end{cases}$$

- Il sistema di disequazioni:

$$\begin{cases} 30x_1 + 20x_2 \leq 30000 \\ 2x_1 + 12x_2 \leq 3600 \end{cases}$$

  Richiede l'ulteriore vincolo di "non negatività" delle variabili x_1 e x_2, $x_1 \geq 0$ e $x_2 \geq 0$. Perché?
- Come lo si risolve?

Ottimizzazione Vincolata

- Le operazioni elementari utilizzate per risolvere i sistemi lineari possono essere usate per un sistema di disequazioni?

  La risposta è NO.

- Può essere trasformato in un sistema di equazioni aggiungendo delle "variabili di scarto" s_1 e s_2:

$$\begin{cases} 30x_1 + 20x_2 + s_1 = 30000 \\ 2x_1 + 12x_2 + s_2 = 3600 \end{cases}$$

  Anche per le variabili di scarto s_1 e s_2 è necessario il vincolo di non negatività, $s_1 \geq 0$ e $s_2 \geq 0$. Perché?

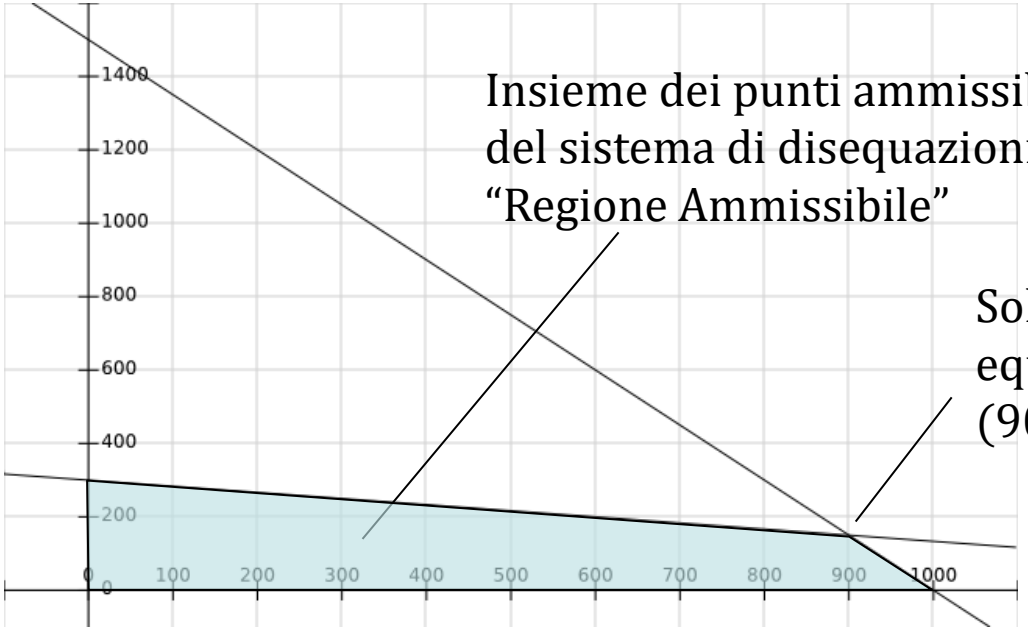
- Ma i vincoli di non negatività non complicano le cose?

  La risposta è NO.

Ottimizzazione Vincolata

- Il sistema di disequazioni:

$$\begin{cases} 30x_1 + 20x_2 \leq 30000 \\ 2x_1 + 12x_2 \leq 3600 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$$



Insieme dei punti ammissibili del sistema di disequazioni: "Regione Ammissibile"

Soluzione sistema equazioni lineari (900,150)

Ottimizzazione Vincolata

- Il sistema di disequazioni definisce l'insieme delle "soluzioni ammissibili" del problema, che chiameremo "Regione Ammissibile".
- In altri termini, tutti i punti (soluzioni) della regione ammissibile soddisfano i vincoli del problema.
- Potremo stabilire un criterio per decidere quale punto della regione ammissibile scegliere.
- Per esempio, se associamo al succo di mela di tipo 1 e tipo 2 due profitti diversi, allora potremo pensare di scegliere la soluzione ammissibile che massimizza il profitto.
- Questo ambito è noto come "Programmazione Matematica".
- Quali sono gli strumenti matematici necessari per affrontare questo argomento?

Ottimizzazione Vincolata

- Se ipotizziamo che il profitto per ogni litro di succo di mela di tipo 1 è 0.25 Euro e per quello di tipo 2 è 0.75, il corrispondente problema di "*programmazione lineare*" si potrebbe scrivere come segue:

$$z = \text{Max } 0.25x_1 + 0.75x_2$$

Funzione Obiettivo

tale che:

$$30x_1 + 20x_2 \leq 30000$$
$$2x_1 + 12x_2 \leq 3600$$
$$x_1 \geq 0$$
$$x_2 \geq 0$$

Vincoli

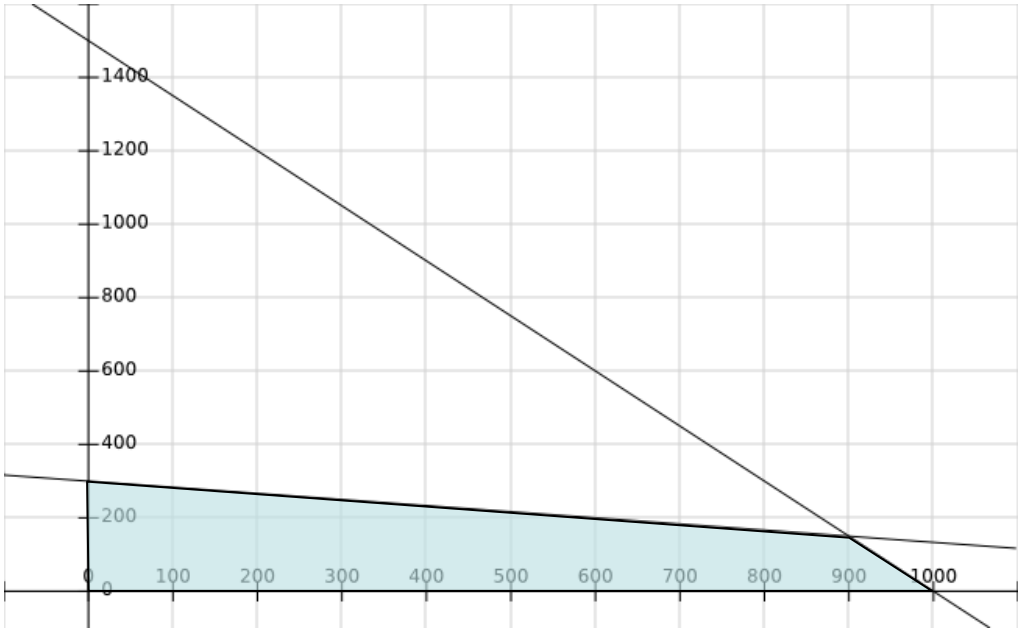
- Per risolvere i problemi di programmazione lineare possono essere applicati numerosi algoritmi. Il più noto è il *Metodo del Simplesso*.
- Il "*metodo del simplesso formato tableau*" usa le operazioni elementari come quelle usate per la soluzione dei sistemi lineari, per esempio, dal Metodo di Eliminazione di Gauss-Jordan.

Ottimizzazione Vincolata

- La *soluzione ottima* del problema di programmazione lineare è $(x_1, x_2) = (900,150)$, dove la funzione obiettivo è $z = 337.5$:

$$z = \text{Max } 0.25x_1 + 0.75x_2 = 337.5$$

Perché?



Ottimizzazione Vincolata

Esempio: un generico problema di produzione

- Come possiamo generalizzare il modello per un generico problema in cui un'azienda deve produrre n prodotti con a disposizione m materie prime in un dato orizzonte temporale?
- Ogni prodotto i ha una ricetta che indica per ogni materia prima j la quantità a_{ij} da utilizzare per produrre una unità.
- Ogni unità di prodotto i genera un profitto p_i e la disponibilità di ogni materia prima j è pari a b_j.
- Se vogliamo trovare le quantità di prodotto che dobbiamo produrre per massimizzare il profitto, come possiamo scrivere un modello di programmazione lineare?
- Quali sono i parametri e quali sono le variabili?
- Come costruisco la funzione obiettivo e i vincoli?

Ottimizzazione Vincolata

- Per ogni prodotto i definiamo una variabile decisionale x_i che indica le unità prodotte.
- Il corrispondente problema di "*programmazione lineare*" si potrebbe scrivere come segue:

$$z = \max \sum_{i=1}^{n} p_i x_i$$
$$s.t. \sum_{i=1}^{n} a_{ij} x_i \leq b_j , \qquad j = 1, \dots, m$$
$$x_i \geq 0, \qquad i = 1, \dots, n$$

- Per ogni istanza dobbiamo definire i parametri p_i, a_{ij} e b_j.
- La soluzione è data dai valori x_i.
- Il problema di programmazione lineare continua è "facile".

Modelli Matematici per l'Ottimizzazione

- Il primo passo per determinare l'algoritmo di ottimizzazione per un problema consiste nel definire il modello matematico.
- Un modello matematico si può rappresentare come segue:

$$
\begin{aligned}
(P) \qquad z_P = \min\, & f(\mathbf{x}) \\
s.t.\ \ & g_i(\mathbf{x}) \leq b_i, && i = 1, \dots, n \\
& h_j(\mathbf{x}) = d_j, && j = 1, \dots, m \\
& \mathbf{x} \geq \mathbf{0}
\end{aligned}
$$

- La funzione $f(\mathbf{x})$ è detta funzione obiettivo, le espressioni $g_i(\mathbf{x}) \leq b_i$ e $h_j(\mathbf{x}) = d_j$ rappresentano i vincoli e $\mathbf{x} = (x_1, \dots, x_\ell)$ sono le variabili.
- L'espressione $\mathbf{x} \geq \mathbf{0}$ rappresenta i vincoli di non negatività.
- Se le funzioni $f(\mathbf{x})$, $g_i(\mathbf{x}) \leq b_i$ e $h_j(\mathbf{x}) = d_j$ sono lineari parliamo di programmazione lineare continua.

Modelli Matematici per l'Ottimizzazione

- Se abbiamo il vincolo aggiuntivo che la soluzione **x** debba avere le componenti intere, allora parliamo di programmazione lineare intera.
- Se solo alcune componenti di **x** devono essere intere, parliamo di programmazione lineare mista intera.
- Dato un problema di programmazione lineare P di "*minimo*", un valido lower bound z_{LB} è una stima per difetto del valore della soluzione ottima z_P (i.e., $z_{LB} \leq z_P$). Le procedure per calcolare i lower bound sono dette procedure di bounding.
- Dato un problema di programmazione lineare P di "*minimo*", una soluzione ammissibile corrisponde a un valido upper bound z_{UB} ed è, quindi, una stima per eccesso del valore della soluzione ottima z_P (i.e., $z_P \leq z_{UB}$). Le procedure per calcolare soluzioni ammissibili sono dette euristici.

Modelli Matematici per l'Ottimizzazione

- Dato un problema di programmazione lineare *P*, un algoritmo esatto "garantisce" (compatibilmente con le risorse di memoria e tempo calcolo disponibili) la determinazione della soluzione ottima di *P*.
- Per capire che cosa sono gli algoritmi di bounding, euristiche ed esatti, consideriamo l'esempio del problema dello zaino (Knapsack Problem).

Knapsack Problem

- Il problema del knapsack (i.e., "dello zaino") consiste nel determinare quale degli n oggetti di peso w_i e profitto p_i devono essere inseriti nel knapsack di capacità W, per massimizzare il profitto complessivo.
- Se si ipotizza che per ogni oggetto si ha una sola copia, allora si parla del problema del knapsack (0–1).
- E' un problema facile o difficile?
- Come si risolve?
- Considerate la seguente istanza:
  - $n = 4$ e $W = 10$
  - $w_1 = 6, w_2 = 4, w_3 = 5, w_4 = 2.$
  - $p_1 = 12, p_2 = 4, p_3 = 15, p_4 = 3.$

  Qual è la soluzione ottima?

Knapsack Problem

- Il problema del knapsack può essere modellato matematicamente (formulato) come segue:

$$\max z_{KP} = \sum_{i=1}^{n} p_i x_i$$

$$s.t. \sum_{i=1}^{n} w_i x_i \leq W$$

$$x_i \in \{0,1\}, \qquad i = 1, \dots, n$$

- Che per la nostra istanza di esempio diventa:

$$\max z_{KP} = 12x_1 + 4x_2 + 15x_3 + 3x_4$$

$$s.t. \quad 6x_1 + 4x_2 + 5x_3 + 2x_4 \leq 10$$

$$x_1, x_2, x_3, x_4 \in \{0,1\}$$

- Come faccio a scegliere gli oggetti di peso complessivo minore o uguale di 10 che massimizzano il profitto?

Knapsack Problem

- Calcoliamo per ogni oggetto i il suo profitto per unità di capacità:

$$r_i = \frac{p_i}{w_i}$$

- L'oggetto che ha il valore r_i più alto è quello che ci fa guadagnare di più per ogni unità di capacità impiegata.
- Per cui, potremo ordinare gli oggetti per valori di r_i decrescenti:

$$r_3 = \frac{15}{5} = 3, \qquad r_1 = \frac{12}{6} = 2, \qquad r_4 = \frac{3}{2} = 1.5, \qquad r_2 = \frac{4}{4} = 1$$

- Se inseriamo nel knapsack il primo oggetto della lista ordinata (il 3) otteniamo un profitto di $p_3 = 15$ e occupiamo $w_3 = 5$ unità di peso.
- Il secondo più conveniente è l'oggetto 1, ma siccome pesa 6 unità non ci sta nelle 5 unità residue.
- Cosa faccio?

Knapsack Problem

- Se l'oggetto 1 fosse divisibile (per esempio, è una partita di grano) posso prenderne solo la frazione che ci sta (5 unità su 6) ottenendo l'occupazione di tutto il knapsack con il profitto "ottimo" pari a:

$$z'_{KP} = 12x_1 + 4x_2 + 15x_3 + 3x_4 = 12 \times \frac{5}{6} + 0 + 15 \times 1 + 0 = 25$$

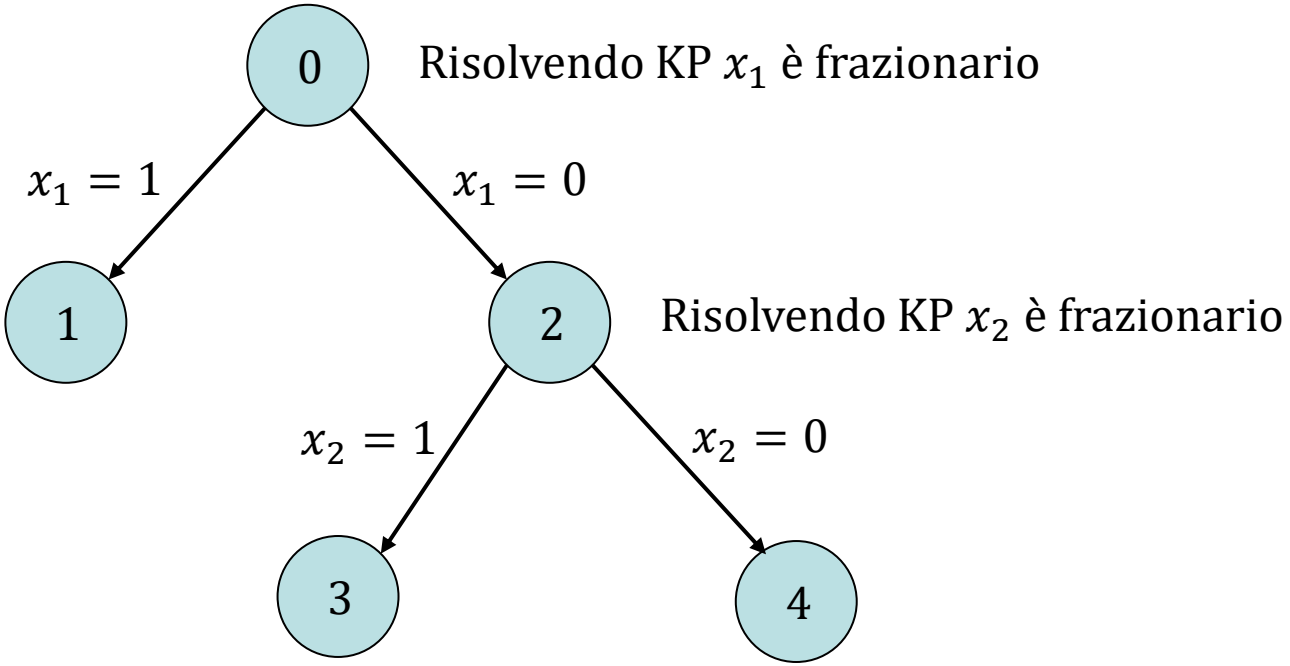
- Se però gli oggetti non sono divisibili, allora il valore $z'_{KP} = 25$ è solo una stima per eccesso (upper bound) al valore della soluzione ottima.
- Potevamo scegliere di inserire il primo oggetto più conveniente tra i rimanenti che ci stava. Sarebbe stato il 4, ottenendo una soluzione di profitto $z''_{KP} = p_3 + p_4 = 18$ e che occupa $w_3 + \mathrm{w}_4 = 7$ unità di peso.
- Quella ottenuta è una soluzione ammissibile del problema e z''_{KP} è una stima per difetto (lower bound) al valore della soluzione ottima.
- Ora sappiamo che il valore della soluzione ottima è compreso tra 18 e 25. Ma come possiamo determinare la soluzione ottima?

Knapsack Problem

- Il problema in cui ammettiamo anche una soluzione frazionaria, è detto rilassamento lineare (sostituiamo $x_i \in \{0,1\}$ con $0 \leq x_i \leq 1$).
- Quando risolviamo il rilassamento lineare del problema del knapsack abbiamo sempre una soluzione in cui avremo al più una sola variabile frazionaria, che chiameremo "splitting item".
- Se la soluzione è intera, allora non potremo mai trovare una soluzione migliore.
- Nel caso una variabile x_i sia frazionaria, allora possiamo generare due nuovi problemi e la soluzione ottima sarà la migliore delle due:
  - Un problema in cui si impone l'oggetto i in soluzione: $x_i = 1$.
  - Un problema in cui si vieta l'oggetto i in soluzione: $x_i = 0$.
- A loro volta i nuovi problemi possono generare soluzioni frazionarie, quindi a loro volta genereranno due nuovi problemi.

Knapsack Problem

- L'approccio appena descritto è noto come branch and bound.



- Quando in un nodo dell'albero di ricerca (tree search) la soluzione è intera ci fermiamo e se migliora la soluzione ottima emergente la sostituisce.

Knapsack Problem

- L'upper bound calcolato per ciascun nodo è utilizzato anche per:
  - Eliminare i nodi che sicuramente non conducono alla soluzione ottima (se l'upper bound è minore del lower bound).
  - Scegliere il nodo da espandere (esistono strategie alternative).
- Il problema del knapsack corrisponde a molti problemi reali o a loro approssimazioni. Esistono varianti del problema del knapsack.
- Molti modelli e algoritmi per risolvere altri problemi decisionali hanno il problema del knapsack come sottoproblema.
- Quando tutti gli oggetti hanno i profitti p_i uguali ai pesi w_i, allora tutti i rapporti r_i sono uguali a 1 e il branch and bound è poco efficiente.
- Per risolvere il problema del knapsack esistono anche altri algoritmi alternativi.
- Una delle alternative più efficienti è la programmazione dinamica.

Knapsack Problem

- La Programmazione Dinamica per risolve il problema del knapsack (0–1) prevede n stadi (quanti sono gli oggetti) e ad ogni stadio un numero di stati pari a W (la capacità del knapsack).
- Ad ogni stadio $j \in \{1, \dots, n\}$ e per ogni stato $w \in \{0, \dots, W\}$ si risolve il seguente sottoproblema:

$$\left(KP_j(w)\right) \quad z_j(w) = \max \sum_{i=1}^{j} p_i x_i$$

$$s.t. \sum_{i=1}^{j} w_i x_i \leq w$$

$$x_i \in \{0,1\}, \qquad i = 1, \dots, j$$

- La Programmazione Dinamica di fatto è una enumerazione parziale delle soluzioni (i.e., considera solo un sottoinsieme delle soluzioni), tra le quali però c'è quella ottima.

Knapsack Problem

- Risolvere per ogni stato w dello stadio j i problemi $KP_j(w)$ equivale a utilizzare la seguente recursione:

  (a) Inizializza $KP_0(w) = 0$, per ogni $w \in \{0, \ldots, W\}$;

  (b) Ad ogni stadio $j \in \{1, \ldots, n\}$ e per ogni stato $w \in \{0, \ldots, W\}$, calcola la seguente recursione:

$$z_j(w) = \begin{cases} z_{j-1}(w), & \text{se } w < w_j \\ \max\{z_{j-1}(w), z_{j-1}(w - w_j) + p_j\}, & \text{se } w \geq w_j \end{cases}$$

- L'algoritmo di programmazione dinamica qui proposto ha complessità $O(nW)$. Quindi si dice che è "pseudopolinomiale".
- Un algoritmo di programmazione dinamica alternativo per il knapsack (0–1) poteva essere ottenuto definendo uno stadio per ogni $w \in \{0, \ldots, W\}$ e uno stato per ogni $j \in \{1, \ldots, n\}$.

Travelling Salesman Problem

- Consideriamo il problema di un autista che deve fare delle consegne visitando una e una sola volta ciascun cliente (cosa significa?).
- Possiamo rappresentare il problema utilizzando un grafo orientato $G = (V, A)$, dove V è l'insieme dei vertici e A è l'insieme degli archi.
- I vertici $i \in V$ rappresentano i clienti da visitare e il deposito da cui partire e rientrare.
- Gli archi $(i, j) \in A$ rappresentano il tragitto dal vertice i al vertice j, a cui è associato un costo c_{ij} (e.g., i chilometri da percorrere, il tempo di guida, etc.). Per semplicità consideriamo il grafo completo (cosa significa?).
- Si noti che in genere si ha che $c_{ij} \neq c_{ji}$ (perché?), per cui parleremo di Asymmetric Travelling Salesman Problem (ATSP, Problema del Commesso Viaggiatore Asimmetrico).

Travelling Salesman Problem

- Siano x_{ij} delle variabili binarie che sono uguali a 1 se l'arco (i,j) è nella soluzione ottima, 0 altrimenti.
- Un modello molto noto per il TSP asimmetrico è il seguente:

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

Problema dell'assegnamento

$$s.t. \sum_{j\in V} x_{ij} = 1, \qquad i \in V$$

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

$$\sum_{i\in S}\sum_{j\in S'} x_{ij} \geq 1, \qquad \forall S \subset V, S' = V\backslash S$$

$$x_{ij} \in \{0,1\}, \qquad (i,j) \in A$$

Travelling Salesman Problem

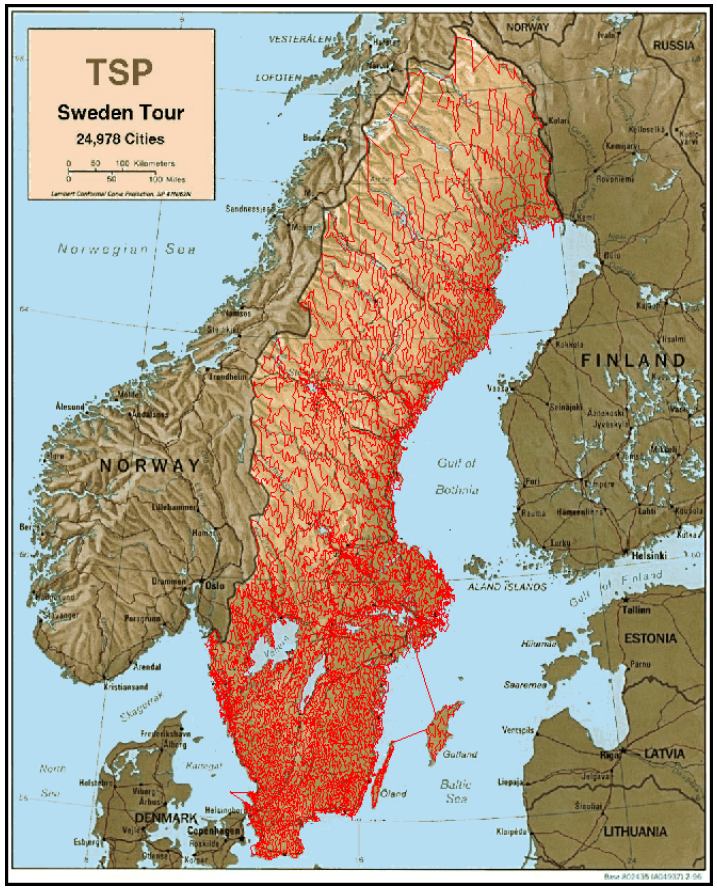
- I vincoli $\sum_{i \in S} \sum_{j \in S'} x_{ij} \geq 1, \forall S \subset V, S' = V \backslash S$, sono detti subtour elimination, perché eliminano i sottocicli e possono essere definiti anche in modo diverso.
- L'insieme degli archi $A(S, S') = \{(i, j) \in A : i \in S, j \in S'\}$ è detto taglio determinato dagli insiemi S e S'.
- I vincoli per eliminare i cicli sono in un numero esponenziale, per cui si può risolvere il problema dell'assegnamento ottenuto ignorando questi vincoli (la soluzione sarà sicuramente intera… perché?).
- Una volta risolto il problema dell'assegnamento, basta identificare se esiste un sottociclo. I vertici visitati dal sottociclo rappresentano un insieme S il cui corrispondente vincolo è violato.
- Si aggiunge il vincolo e si riottimizza il problema, che ora non è più un semplice assegnamento e il suo rilassamento lineare può avere soluzioni frazionarie.

Travelling Salesman Problem

- Anche se le soluzioni sono frazionarie, è possibile determinare se un vincolo di eliminazione dei cicli è stato violato e identificare l'insieme *S* corrispondente.
- Purtroppo, anche per trovare l'ottimo del rilassamento lineare della formulazione può essere necessario generare molti vincoli.
- Inoltre, la soluzione del rilassamento continuo può essere molto frazionaria e può avere un valore molto distante dal valore della soluzione ottima intera.
- Per cui, è necessario aggiungere anche altri vincoli (disuguaglianze valide) ridondanti nella formulazione intera originaria, ma che eliminano molte soluzioni frazionarie.
- Se aggiungiamo disuguaglianze valide durante la soluzione con un metodo branch and bound, diremo che stiamo usando un metodo branch and cut.

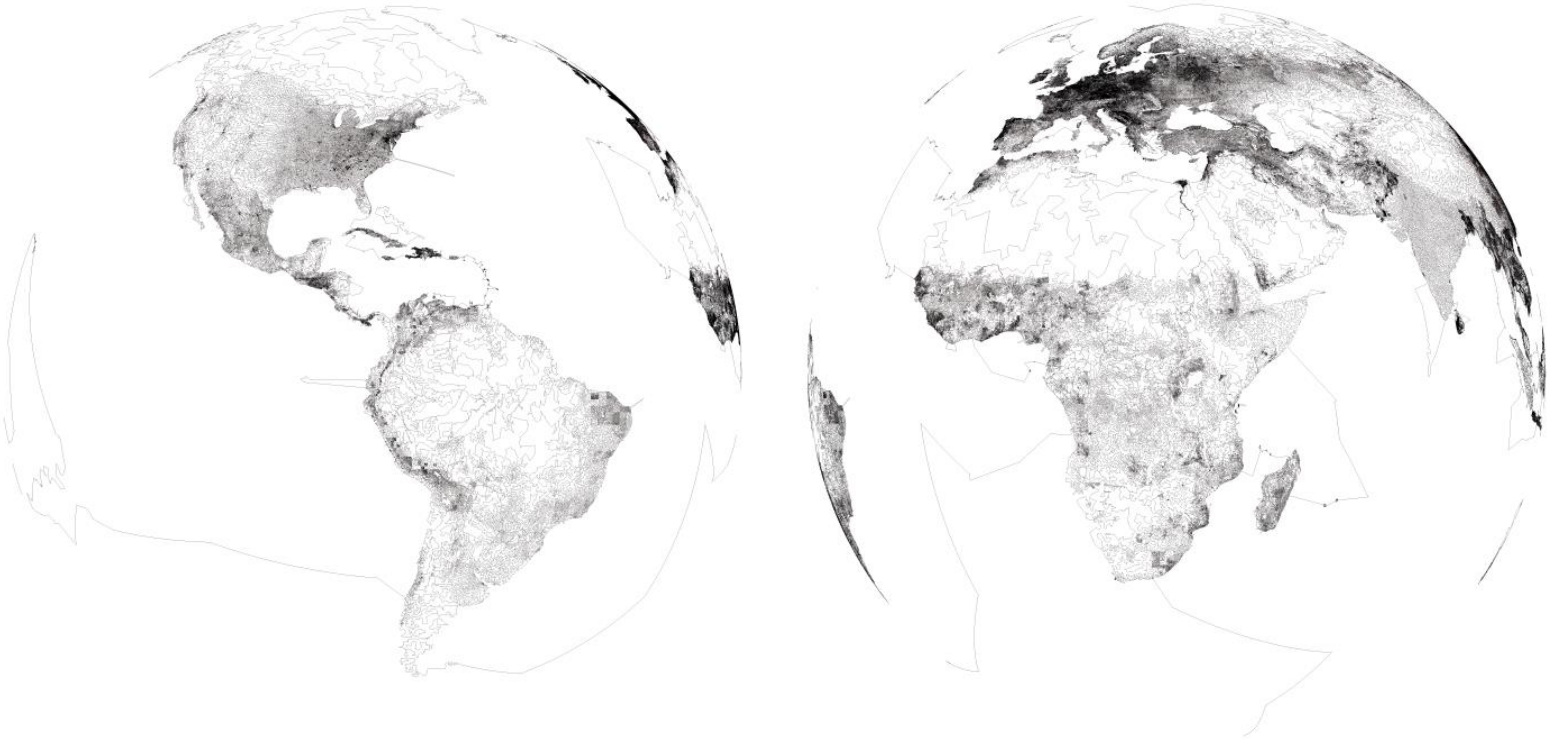
Travelling Salesman Problem

- Gli strumenti matematici sviluppati hanno permesso la soluzione di TSP di enormi dimensioni:



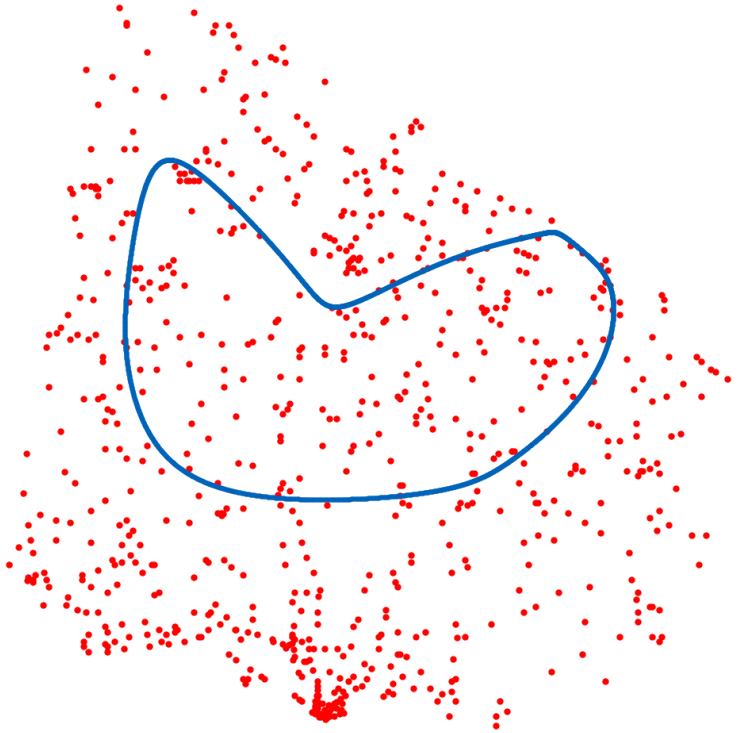
Travelling Salesman Problem

- Il World TSP Tour trovato da Keld Helsgaun nel dicembre 2003. Un lower bound fornito dal codice Concorde TSP ha dimostrato che la soluzione dista al massimo lo 0.076% dal tour ottimo che attraversa le 1,904,711 città.



Travelling Salesman Problem

- I risultati mostrati finora sono stati ottenuti con l'ottimizzazione "tradizionale" … e gli "altri" metodi dell'intelligenza artificiale?



- **Qatar**, containing 194 cities with an optimal tour of 9352.
- **Uruguay**, containing 734 cities with an optimal tour of 79114.
- **Finland**, containing 10639 cities with an optimal tour of 520527.
- **Italy**, containing 16862 cities with an optimal tour of 557315.

Instance	Iterations	Time (s)	Length	Quality
Qatar	14690	14.3	10233.89	9.4%
Uruguay	17351	23.4	85072.35	**7.5%**
Finland	37833	284.0	636580.27	22.3%
Italy	39368	401.1	723212.87	29.7%

Fonte: Using Self-Organizing Maps to solve the TSP (https://diego.codes/post/som-tsp/)

Metodi di soluzione

- I problemi di programmazione lineare (LP) continua possono essere risolti con diverse tecniche:
  - Simplesso Primale;
  - Simplesso Duale;
  - Simplesso per le reti;
  - Metodo Barrier;
  - Etc...
- I problemi di programmazione lineare intera o mista intera (PLI) possono essere risolti utilizzando i seguenti algoritmi:
  - Branch & Bound;
  - Branch & Cut;
  - Branch & Price;
  - Branch & Price & Cut;
  - Programmazione Dinamica;
  - Etc...

Metodi di soluzione

- Per risolvere i problemi di programmazione non-lineare, si possono usare metodi come il Barrier, che in alcuni risolutori commerciali consente di trattare sia funzioni obiettivo che vincoli quadratici.
- I problemi di programmazione quadratica intera o mista intera possono essere risolti utilizzando metodi di tipo Branch & Bound.
- Esistono diversi risolutori sia commerciali che open source che permettono il loro impiego a vari livelli: console interattiva, librerie da integrare in software scritto in C, C++, JAVA, etc...
- I risolutori commerciali più noti sono: IBM Ilog Cplex, Gurobi, FICO XPRESS, LINDO, etc...
- I risolutori open source più noti sono: GLPK, Coin-OR (progetti CLP, CBC e BCP), SoPlex, etc...

Linguaggi di Modellazione

- I linguaggi di modellazione servono per descrivere un problema di ottimizzazione con un buon livello di astrazione e semplicità.
- Sarà compito del linguaggio di modellazione interfacciarsi con il solutore per caricare e risolvere il problema e per accedere ai dati e alle soluzioni.
- In alternativa all'uso dei linguaggi di modellazione si potrebbero utilizzare i seguenti approcci:
  - si preparano i dati in file di formato standard (e.g., LP, MPS, etc.), che poi vengono caricati nei risolutori per mezzo della "console" (che qualche risolutore mette a disposizione degli utenti).
  - si utilizzano le librerie di un solutore che vengono "linkate" con i linguaggi di programmazione (C, C++, JAVA, Python, etc.).
  - si utilizza uno spreadsheet in cui è integrato un risolutore (Excel).

Linguaggi di Modellazione

- Sul mercato esistono diversi linguaggi di modellazione: AMPL, OPL, GAMS, LINGO, etc.
- Tra i diversi linguaggi disponibili si è scelto di usare AMPL, che però non differisce molto dagli altri.
- AMPL si può interfacciare con numerosi risolutori.
- AMPL è una soluzione commerciale.
- Sul sito web **www.ampl.com** è disponibile una versione "studente" che può essere scaricata gratuitamente. Include anche alcuni solutori, che però hanno alcune limitazioni, in particolare nelle dimensioni dei problemi che possono risolvere.

Linguaggi di Modellazione

- Impariamo a usare AMPL utilizzando degli esempi.
- Consideriamo il seguente problema di programmazione lineare relativo a un'applicazione in ambito finanziario:

$$
\begin{aligned}
z = \text{Max } & 3x_1 + 4x_2 && \textit{Rendimento} \\
\text{tale che:} & && \\
x_1 + \ & x_2 \leq 100 && \textit{Capitale} \\
2x_1 + \ & x_2 \leq 150 && \textit{Rating medio} \\
3x_1 + & 4x_2 \leq 360 && \textit{Scadenza media} \\
x_1, & x_2 \geq 0 && \textit{Non negatività}
\end{aligned}
$$

- Il modello può essere inserito direttamente utilizzando la console "ampl" (eseguibile) richiamabile da linea di comando Linux digitando quanto segue:

```
./ampl
```

Linguaggi di Modellazione

- Per inserire il problema in AMPL dobbiamo fare quanto segue:

```
ampl: option solver "./cplex";
ampl: var X1;
ampl: var X2;
ampl: maximize yield: 4*X1 + 3*X2;
ampl: subject to cash: X1 + X2 <= 100;
ampl: subject to rating: 2*X1 + X2 <= 150;
ampl: subject to maturity: 3*X1 + 4*X2 <= 360;
ampl: subject to X1_limit: X1 >= 0;
ampl: subject to X2_limit: X2 >= 0;
ampl: solve;
...
ampl: display X1;
X1 = 50
ampl: display X2;
X2 = 50
```

Linguaggi di Modellazione

- Invece di inserire il modello direttamente da console lo si può salvare su un file.
- Questo consente di salvare il lavoro fatto per poter riutilizzare e modificare il modello.
- Per esempio, possiamo salvare il modello nel file “bonds_opt.mod”. Dopodiché, per risolverlo digiteremo quanto segue:

```
ampl: option solver "./cplex";
ampl: model bonds_opt.mod;
ampl: solve;
...
ampl: display X1;
X1 = 50
ampl: display X2;
X2 = 50
```

Linguaggi di Modellazione

- Se ora si vuole generalizzare il modello dell'esempio precedente a più di due prodotti finanziari e con dati diversi.
- AMPL consente di separare il modello dai dati.
- Gli elementi fondamentali di un problema LP sono:

Dati

- Insiemi: liste di prodotti, risorse, etc.
- Parametri: input numerici come costi, consumo di risorse, etc.

Modello

- Variabili: valori che devono essere definiti dal risolutore.
- Funzione obiettivo: funzione delle variabili decisionali da minimizzare o massimizzare.
- Vincoli: funzioni delle variabili decisionali che devono rimanere entro certi limiti.

Linguaggi di Modellazione

- Possiamo salvare il modello generale nel file “bonds.mod”:

```
set bonds; # bonds disponibili

param yield {bonds}; # rendimenti
param rating {bonds}; # ratings
param maturity {bonds}; # scadenze
param max_rating; # Massimo rating medio ammesso
param max_maturity; # Massima scadenza media ammessa
param max_cash; # Capitale massimo disponibile per
l'investimento

var buy {bonds} >= 0; # Quantità da investire per ciascun
prodotto contenuto nell'insieme bonds

maximize total_yield : sum {i in bonds} yield[i] * buy[i];

subject to cash_limit : sum {i in bonds} buy[i] <= max_cash;
subject to rating_limit :
sum {i in bonds} rating[i]*buy[i] <= max_rating;
subject to maturity_limit :
sum {i in bonds} maturity[i]*buy[i] <= max_maturity;
```

Linguaggi di Modellazione

- Per una particolare istanza possiamo salvare il dati nel file "bonds.dat":

```
set bonds := A B;

param : yield rating maturity :=
    A          4        2        3
    B          3        1        4;

param max_cash := 100;
param max_rating := 150;
param max_maturity := 360;
```

Linguaggi di Modellazione

- Per risolvere il modello definito nel file “bonds.mod” con i dati salvati nel file “bonds.dat” basta digitare quanto segue dalla console:

```
ampl: model bonds.mod;
ampl: data bonds.dat;
ampl: solve;
...
ampl: display buy;
buy [*] :=
A 50
B 50
```

Linguaggi di Modellazione

- Si possono anche modificare i dati direttamente da console:

```
ampl: reset data max_cash;
ampl: data;
ampl data: param max_cash := 150;
ampl data: param max_rating := 225;
ampl data: param max_maturity := 540;
ampl data: solve;
...
ampl: display buy;
buy [*] :=
A 45
B 105
```

Linguaggi di Modellazione

- Si possono aggiungere nuovi prodotti (bonds) nell'istanza e salvare i dati nel file "bonds_ext.dat":

```
set bonds := A B C;
param : yield rating maturity :=
    A      4       2       3
    B      3       1       4
    C      5       3       2;
param max_cash := 100;
param max_rating := 150;
param max_maturity := 360;
```

Linguaggi di Modellazione

- Per risolvere la nuova istanza basterà ricaricare i dati:

```
ampl: reset data;
ampl: data bonds_ext.dat;
ampl: solve;
..
ampl: display buy;
buy [*] :=
A 0
B 85
C 15
```

Linguaggi di Modellazione

- Nella programmazione lineare abbiamo a disposizione un ulteriore strumento: le variabili duali (shadow prices).
- Consideriamo il seguente problema di produzione:

```
ampl: var X1;
ampl: var X2;
ampl: maximize profit: 3*X1 + 3*X2;
ampl: subject to hours: 3*X1 + 4*X2 <= 120000;
ampl: subject to cash: 3*X1 + 2*X2 <= 90000;
ampl: subject to X1_limit: X1 >= 0;
ampl: subject to X2_limit: X2 >= 0;
ampl: solve;
...
ampl: display X1;
X1 = 20000
ampl: display X2;
X2 = 15000
```

Linguaggi di Modellazione

- Per visualizzare variabili duali ottime:

```
...
ampl: display hours, cash;
hours = 0.5
cash = 0.5
```

- La soluzione duale ci dice che aumentando le ore di lavorazione di 2000 unità si aumenterà il profitto di 2000 × 0.5 = 1000 Euro.
- Per cui potremo decidere di pagare fino a 0.50 Euro all'ora per il lavoro in più necessario per aumentare le ore di lavorazione disponibili.
- Inoltre, si può notare che la disponibilità di denaro e ore/uomo contribuiscono equamente al costo/profitto di ogni prodotto.

Linguaggi di Modellazione

Esercizio 1

- Un'azienda deve produrre n prodotti con a disposizione m materie prime in un dato orizzonte temporale (e.g., giorno, settimana, etc.).
- Ogni prodotto i ha una ricetta che indica per ogni materia prima j la quantità a_{ij} da utilizzare per produrre una unità.
- Ogni unità di prodotto i genera un profitto p_i e la disponibilità di ogni materia prima j è pari a b_j.
- Vogliamo trovare le quantità di prodotto che dobbiamo produrre per massimizzare il profitto.
- Usare AMPL per risolvere il problema.

Linguaggi di Modellazione

- Il corrispondente problema di "*programmazione lineare*" si potrebbe scrivere come segue:

$$z = \text{Max} \sum_{i=1}^{n} p_i x_i$$

$$s.t. \sum_{i=1}^{n} a_{ij} x_i \leq b_j\,, \qquad j = 1, \dots, m$$

$$x_i \geq 0, \qquad i = 1, \dots, n$$

- I dati dell'istanza sono i seguenti:
  - Materie prime $m = 2$: $b_1 = 30000$ e $b_2 = 3600$
  - Prodotto $n = 2$:
    - $p_1 = 0{,}25, a_{11} = 30, a_{12} = 2$
    - $p_2 = 0{,}75, a_{21} = 20, a_{22} = 12$

Linguaggi di Modellazione

- La soluzione prevede il modello definito nel file “esercizio1.mod” :

```
set prod; # prodotti
set raw; # materie prime disponibili

param profit {prod}; # profitti
param qnt {raw}; # quantita’ materie prime disponibili
param recipe {prod,raw}; # ricetta

var x {prod} >= 0; # quantita’ da produrre

maximize tot_prof : sum {i in prod} profit[i] * x[i];

subject to raw_limit {j in raw}: sum {i in prod}
recipe[i,j] * x[i] <= qnt[j];
```

Linguaggi di Modellazione

- La soluzione prevede i dati salvati nel file “esercizio1.dat” :

```
set prod := T1 T2;
set raw := water apple;

param : profit :=
    T1   0.25
    T2   0.75;

param : qnt :=
    water  30000
    apple   3600;

param recipe : water apple :=
    T1      30       2
    T2      20      12;
```

Linguaggi di Modellazione

Esercizio 2

- Un'azienda ha n depositi e m punti vendita.
- Da ogni deposito $i = 1, \dots, n$ deve trasferire a_i unità di merce.
- A ogni punto vendita $j = 1, \dots, m$ devono arrivare b_j unità di merce.
- Trasportare una unità di merce dal deposito i al punto vendita j costa c_{ij}.
- Vogliamo determinare come rifornire i punti vendita dai depositi minimizzando il costo di trasporto.
- Usare AMPL per risolvere il problema.

Esercizio 3

- Come cambia il problema e il modello dell'esercizio 2 se per essere abilitato a trasportare una qualche quantità di merce dal deposito i al punto vendita j bisogna pagare un costo fisso f_{ij}?

Linguaggi di Modellazione

- Il modello di "*programmazione lineare*" potrebbe essere il seguente:

$$z = \text{Min} \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

$$s.t. \sum_{j=1}^{m} x_{ij} = a_i, \qquad i = 1, \dots, n$$

$$\sum_{i=1}^{n} x_{ij} = b_j, \qquad j = 1, \dots, m$$

$$x_{ij} \geq 0, \qquad i = 1, \dots, n, \; j = 1, \dots, m$$

- In questo caso la soluzione è una matrice $n \times m$.
- Si noti che è necessario che $\sum_{i=1}^{n} a_i = \sum_{j=1}^{m} b_j$ (perché?). Come si risolve il problema se la condizione non è rispettata?

Linguaggi di Modellazione

- I dati dell'istanza da risolvere sono i seguenti:
  - Depositi $n = 2$: $a_1 = 10$ e $a_2 = 15$
  - Punti vendita $m = 3$: $b_1 = 5$, $b_2 = 12$ e $b_3 = 8$
  - Costi trasporti:
    - Deposito 1: $c_{11} = 100$, $c_{12} = 300$, $c_{13} = 500$
    - Deposito 2: $c_{21} = 200$, $c_{22} = 400$, $c_{23} = 600$

Linguaggi di Modellazione

- La soluzione prevede il modello definito nel file “esercizio2.mod” :

```
set depot; # depositi
set pdv; # punti di vendita

param disp {depot}; # disponibilita’ ai depositi
param req {pdv}; # richieste punti di ventida
param cost {depot,pdv}; # costi di trasporto

var x {depot,pdv} >= 0; # quantità trasportata dal
deposito i al punto vendita j;

minimize tot_cost : sum {i in depot} sum {j in pdv}
cost[i,j] * x[i,j];

subject to depot_limit {i in depot}: sum {j in pdv}
x[i,j] = disp[i];

subject to pdv_limit {j in pdv}: sum {i in depot}
x[i,j] = req[j];
```

Linguaggi di Modellazione

- La soluzione prevede i dati salvati nel file “esercizio2.dat” :

```
set depot := depBO depMI;
set pdv := pdvFI pdvRM pdvNA;

param : disp :=
    depBO  10
    depMI  15;

param : req :=
    pdvFI  5
    pdvRM  12
    pdvNA  8;

param cost : pdvFI pdvRM pdvNA :=
    depBO     100   300   500
    depMI     200   400   600;
```

Linguaggi di Modellazione

- Se aggiungiamo un costo fisso per usare ciascun arco, il modello di "*programmazione lineare*" potrebbe essere il seguente:

$$z = \text{Min} \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} + f_{ij} y_{ij}$$

$$s.t. \sum_{j=1}^{m} x_{ij} = a_i, \qquad i = 1, \dots, n$$

$$\sum_{i=1}^{n} x_{ij} = b_j, \qquad j = 1, \dots, m$$

$$0 \le x_{ij} \le u_{ij} y_{ij}, \quad i = 1, \dots, n, \quad j = 1, \dots, m$$

$$y_{ij} \ge 0 \text{ intero}, \quad i = 1, \dots, n, \quad j = 1, \dots, m$$

dove u_{ij} è un limite a quanto può essere trasportato da un mezzo.

Linguaggi di Modellazione

- La soluzione prevede il modello definito nel file “esercizio3.mod” :

```
set depot; # depositi
set pdv; # punti di vendita

param disp {depot}; # disponibilita’ ai depositi
param req {pdv}; # richieste punti di ventida
param cost {depot,pdv}; # costi di trasporto
param fixcost {depot,pdv}; # costi fissi di trasporto
param cap {depot,pdv}; # capacita’ di trasporto

var x {depot,pdv} >= 0; # quantita’ trasportata da i a j;
var y {depot,pdv} integer >= 0; # quantita’ di mezzi da i a j;

minimize tot_cost : sum {i in depot} sum {j in pdv} (cost[i,j] * x[i,j]
+ fixcost[i,j] * y[i,j]);

subject to depot_limit {i in depot}: sum {j in pdv} x[i,j] = disp[i];

subject to pdv_limit {j in pdv}: sum {i in depot} x[i,j] = req[j];

subject to cap_limit {i in depot,j in pdv}: x[i,j] <= cap[i,j] * y[i,j];
```

Linguaggi di Modellazione

- La soluzione prevede i dati salvati nel file “esercizio3.dat” :

```
set depot := depBO depMI;
set pdv := pdvFI pdvRM pdvNA;

param : disp :=
    depBO  10
    depMI  15;

param : req :=
    pdvFI  5
    pdvRM  12
    pdvNA  8;

param cost : pdvFI pdvRM pdvNA :=
    depBO     100   300   500
    depMI     200   400   600;

param fixcost : pdvFI pdvRM pdvNA :=
    depBO       1000  1000  1000
    depMI       1000  1000  1000;

param cap : pdvFI pdvRM pdvNA :=
    depBO     8     8     8
    depMI     8     8     8;
```

Algoritmi euristici e metaeuristici

- Si consideri un generico problema di ottimizzazione:

$$\min f(\mathbf{x})$$
$$\mathbf{x} \in X$$

- La funzione $f: \mathbb{R}^n \longrightarrow \mathbb{R}$ è detta funzione obiettivo.
- Il vettore delle variabili decisionali è $\mathbf{x} = [x_1, x_2, \dots, x_n]^{\mathrm{T}} \in \mathbb{R}^n$.
- L'insieme X rappresenta le soluzioni ammissibili, che sono quelle che soddisfano i "vincoli del problema".
- Ogni soluzione ammissibile $\mathbf{x} \in X$ ha un costo $f(\mathbf{x})$.
- Se esiste una soluzione $\mathbf{x}^* \in X$ tale che:

$$f(\mathbf{x}^*) < f(\mathbf{x}), \qquad \forall \mathbf{x} \in X$$

allora $\mathbf{x}^* \in X$ è la soluzione ottima (o minimo globale).

Algoritmi euristici e metaeuristici

- L'obiettivo è quello di determinare tra le soluzioni ammissibili una soluzione ottima o una soluzione di "buona qualità".
- Molti problemi di ottimizzazione sono NP-difficili e spesso le istanze di interesse pratico hanno dimensioni tali da rendere proibitivo l'uso di algoritmi esatti di soluzione.
- Per risolvere quei problemi dove non si possono usare metodi esatti si possono utilizzare algoritmi euristici, che permettono di ottenere buone soluzioni in tempi di calcolo ridotti (e l'uso della memoria?).
- In generale, gli algoritmi euristici non garantiscono l'ottimalità della soluzione prodotta e di norma non sono in grado di fornire neanche una stima della distanza dalla soluzione ottima.
- L'uso della matematica nelle euristiche potrebbe consentire di avere una stima della distanza dalla soluzione ottima e/o fornire strumenti utili a migliorarne le prestazioni.

Algoritmi euristici e metaeuristici

- Gli algoritmi euristici possono essere classificati come segue:

  a) Algoritmi costruttivi e di ricerca locale: sfruttano le proprietà strutturali delle soluzioni ammissibili per ottenere rapidamente una soluzione di buona qualità.

  b) Metaeuristiche: gestiscono il trade-off tra diversificazione della ricerca, quando la ricerca è effettuata in regioni dello spazio di ricerca poco promettenti, e intensificazione nella regione dello spazio più promettente.

  c) Algoritmi basati sulla programmazione matematica: sfruttano alcuni risultati della programmazione matematica (per esempio, metodi di decomposizione, lower/upper bounds, etc.).

- Caratteristica fondamentale degli algoritmi metaeuristici e quella di fornire un framework generale che può essere facilmente utilizzato (e adattato) per risolvere problemi di tipo diverso.

Metaeuristiche

- Gli algoritmi costruttivi e di ricerca locale possono spesso funzionare molto bene, ma possono "bloccarsi" in soluzioni di scarsa qualità.
- A partire dalla metà degli anni '70 sono stati proposti nuovi approcci, chiamati metaeuristici, che possono guidare gli algoritmi costruttivi e di ricerca locale per trovare soluzioni di migliore qualità.
- In letteratura sono state proposte diverse tipologie di metaeuristiche:
  - Single Solution Metaheuristics: Simulated Annealing, Tabu Search, Iterated Local Search, Variable Neighborhood Search, GRASP, etc.
  - Population Based Metaheuristics: Algoritmi Genetici, Ant Colony Optimization (ACO, ANTS), Scatter Search, etc.
  - Matheuristics: Diving Heuristics, Very Large-Scale Neighborhood Search, Decomposition-Based Heuristics, etc.
- Anche le metaeuristiche "tradizionali" possono usare la matematica.

Ricerca Locale

- Per ogni soluzione $\mathbf{x} \in X$, si definisce l'insieme di vicinanza $N(\mathbf{x}) \subset X$, (noto anche come neighborhood), che rappresenta le soluzioni vicine alla soluzione $\mathbf{x}$.

Algoritmo Ricerca Locale

Step 1. Genera una soluzione iniziale $\mathbf{x} \in X$.

Step 2. Trova $\mathbf{x}' \in N(\mathbf{x})$ tale che $f(\mathbf{x}') = \min\{f(\mathbf{x}'') : \forall \mathbf{x}'' \in N(\mathbf{x})\}$.

Step 3. Se $f(\mathbf{x}') < f(\mathbf{x})$, allora $\mathbf{x} = \mathbf{x}'$ e vai allo Step 2.

Step 4. La miglior soluzione trovata è $\mathbf{x}^* = \mathbf{x}$.

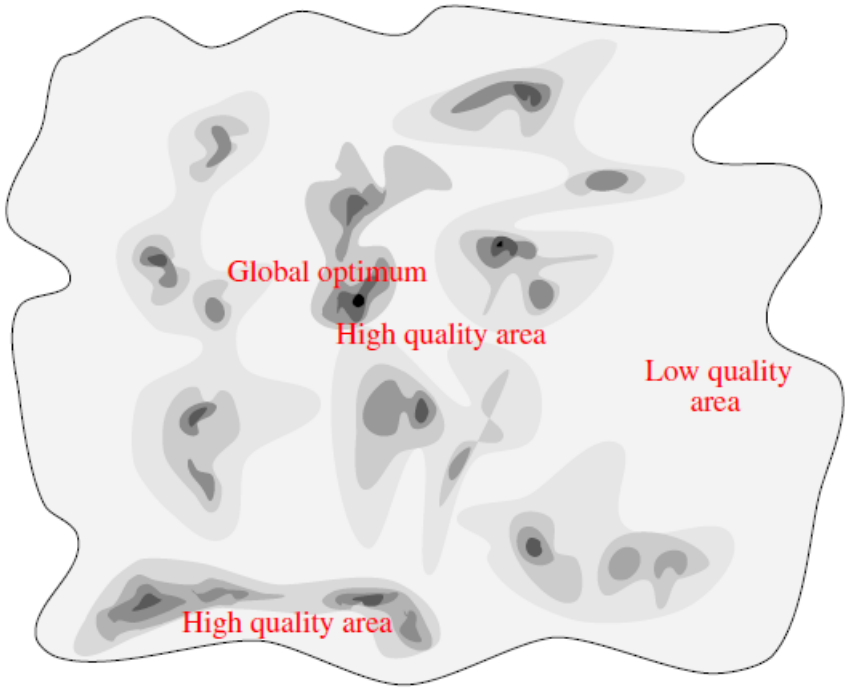
Nota: L'aggiornamento della soluzione allo Step 3 è detto spostamento o mossa da $\mathbf{x}$ a $\mathbf{x}'$. Le mosse possibili dipendono da come è stato definito il neighborhood, ossia il concetto di "soluzioni vicine".

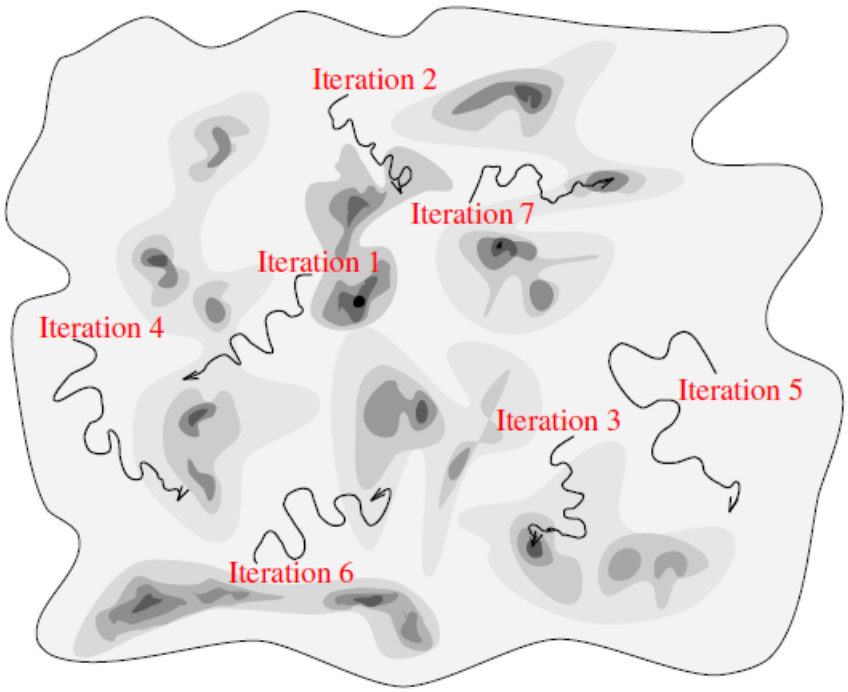
Ricerca Locale

- L'algoritmo può facilmente terminare in un minimo locale $\mathbf{x}^*$:

$$f(\mathbf{x}^*) < f(\mathbf{x}), \qquad \forall \mathbf{x} \in N(\mathbf{x}^*)$$

che può essere migliorato semplicemente applicando l'algoritmo di ricerca locale a diverse soluzioni iniziali.





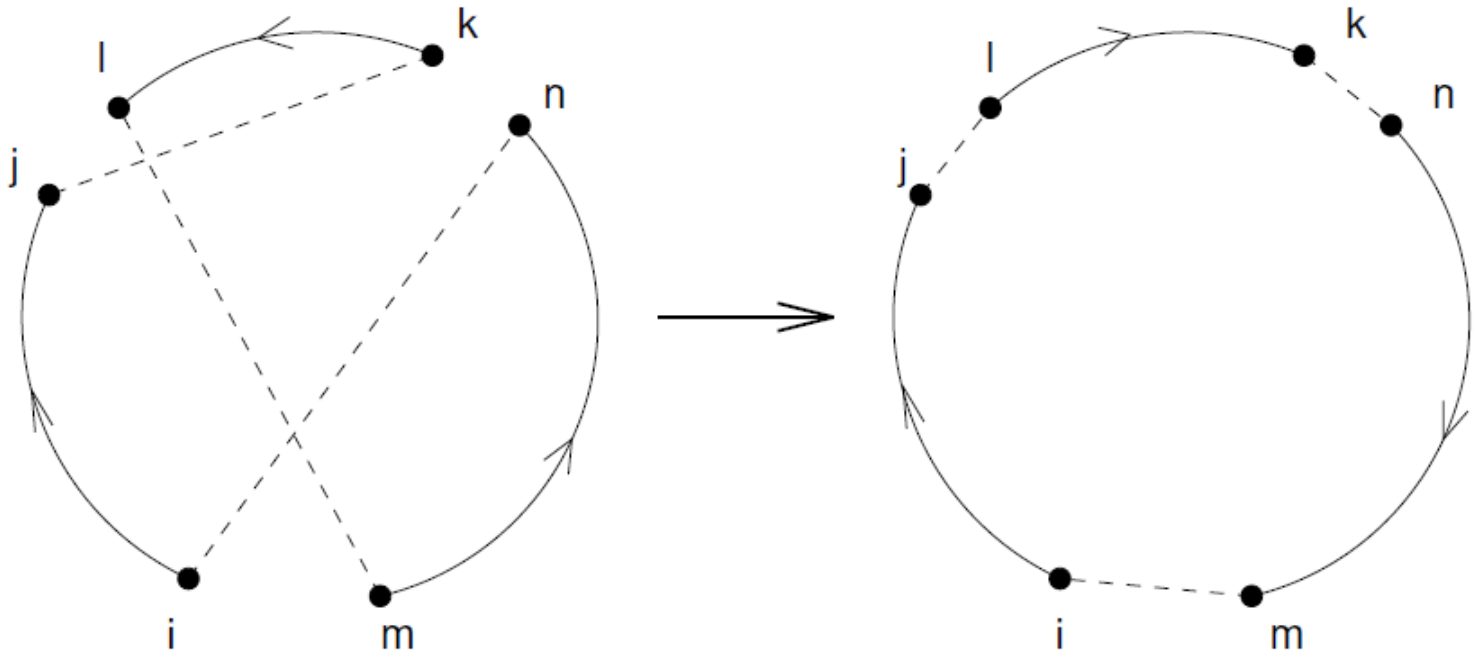
Ricerca Locale

Esempio: Travelling Salesman Problem (TSP)

- Un possibile algoritmo di ricerca locale per il TSP può svolgere i seguenti due passi principali:
  1) Costruzione di una soluzione iniziale: per esempio applicando un algoritmo "nearest neighbor" (i.e., parte da un nodo iniziale e mette in soluzione l'arco che lo congiunge al nodo più vicino, poi il principio viene riapplicato all'ultimo nodo inserito finché non si sono visitati tutti i nodi).
  2) Miglioramento della soluzione: per migliorare una soluzione **x**, partendo da quella iniziale, si potrebbero applicare degli scambi. Gli scambi che vengono considerati definiscono il neighborhood e lo spostamento viene svolto utilizzando lo scambio "migliore".
- Lo scambio migliore potrebbe essere quello che riduce il costo della nuova soluzione risultante (vi sono alternative?).

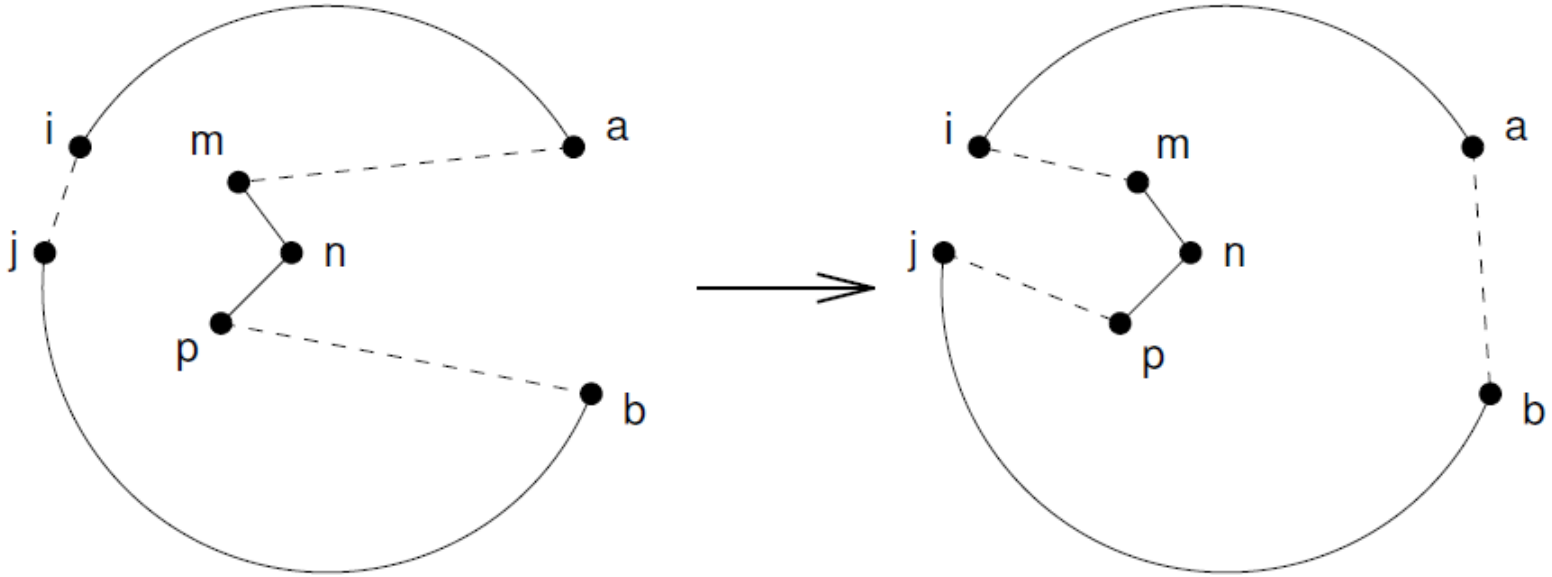
Ricerca Locale

3-opt (Lin and Kernighan (1973))



Ricerca Locale

Or-opt (Or (1976))



Tabu Search

- Il Tabu Search è stato proposto originariamente da Glover (1986).
- Il Tabu Search ad ogni iterazione, si muove nella migliore soluzione disponibile nell'intorno della soluzione corrente:

$$f(\mathbf{x}') = \min\{f(\mathbf{x}'') : \forall \mathbf{x}'' \in N(\mathbf{x})\}$$

- Il Tabu search consente di uscire dai minimi locali muovendosi anche in soluzioni peggiori di quella corrente.
- Una struttura di memoria chiamata Tabu List impedisce di tornare su soluzioni già visitate.
- La ricerca locale si modifica come segue:

$$f(\mathbf{x}') = \min\{f(\mathbf{x}'') : \forall \mathbf{x}'' \in N(\mathbf{x}), \mathbf{x}'' \notin TL\}$$

dove l'insieme TL rappresenta la tabu list.

La tabu list ha una lunghezza massima, per cui dopo un certo numero di iterazioni alcune soluzioni potrebbero essere riconsiderate.

Tabu Search

Algoritmo Tabu Search

Step 1. Genera una soluzione iniziale $\mathbf{x} \in X$.

Poni $\mathbf{x}^* = \mathbf{x}$ e inizializza la Tabu List vuota $TL = \emptyset$.

Step 2. Trova $\mathbf{x}' \in N(\mathbf{x})$, tale che:

$$f(\mathbf{x}') = \min\{f(\mathbf{x}'') : \forall \mathbf{x}'' \in N(\mathbf{x}), \mathbf{x}'' \notin TL\}$$

Step 3. Poni $\mathbf{x} = \mathbf{x}'$, $TL = TL \cup \{\mathbf{x}\}$.

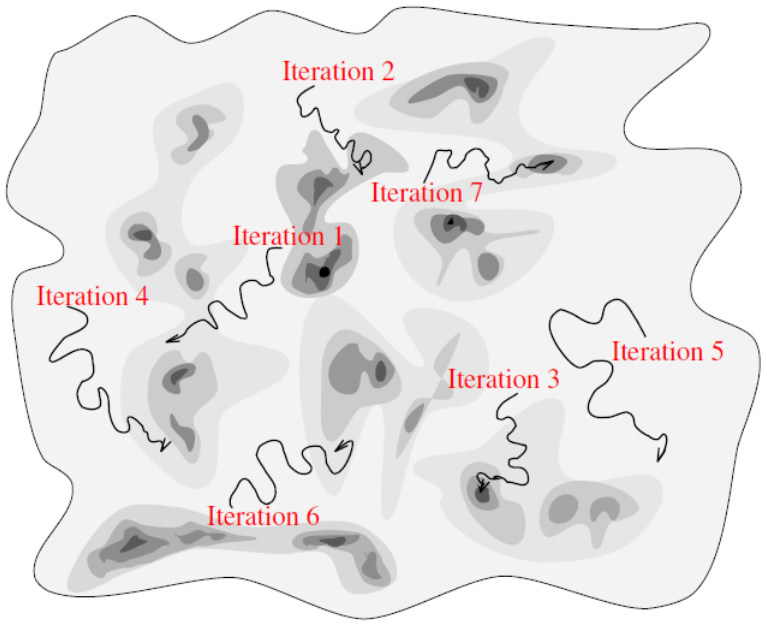
Se $f(\mathbf{x}) < f(\mathbf{x}^*)$ allora $\mathbf{x}^* = \mathbf{x}$.

Step 4. Se la condizione di terminazione non è soddisfatta goto Step 2.

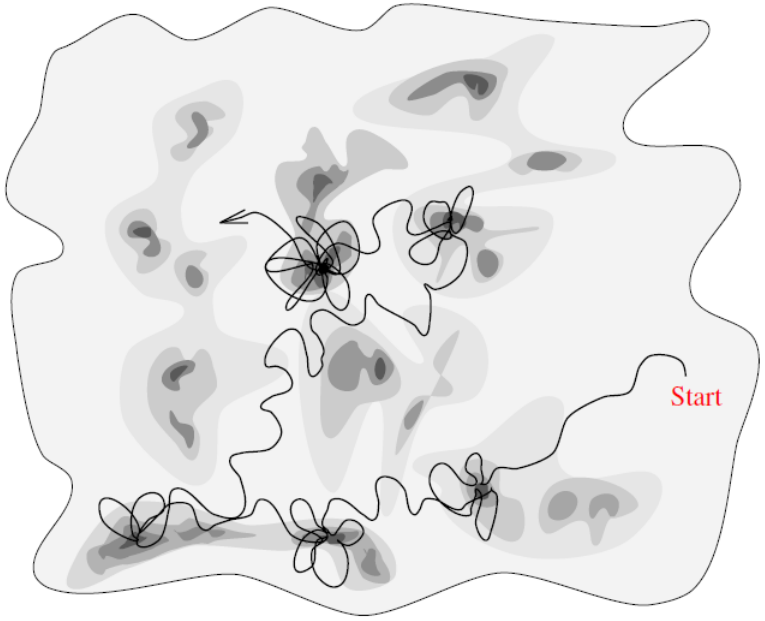
Tabu Search

Ricerca Locale vs Tabu Search

Ricerca Locale



Tabu Search



Simulated Annealing

- Il Simulated Annealing è stato proposto da Kirkpatrick et al. (1983) ed è basato sul metodo Monte Carlo di Metropolis et al. (1953).
- L'annealing corrisponde al processo termico che raggiunge stati di bassa energia libera in un materiale solido mediante ripetute fasi di riscaldamento e lento raffreddamento controllato.
- Il simulated annealing è un algoritmo che "imita" questo processo termodinamico per minimizzare una funzione obiettivo.
- Il simulated annealing è un algoritmo di "ricerca globale" che impiega un parametro "temperature" in modo tale che alle "alte temperature" la ricerca è "diversificata", mentre alle "basse temperature" la ricerca è "intensificata".
- L'algoritmo parte da una temperatura alta, che consente di peggiorare la soluzione con alta probabilità, per poi diminuire progressivamente la temperatura e ridurre la probabilità di accettare peggioramenti.

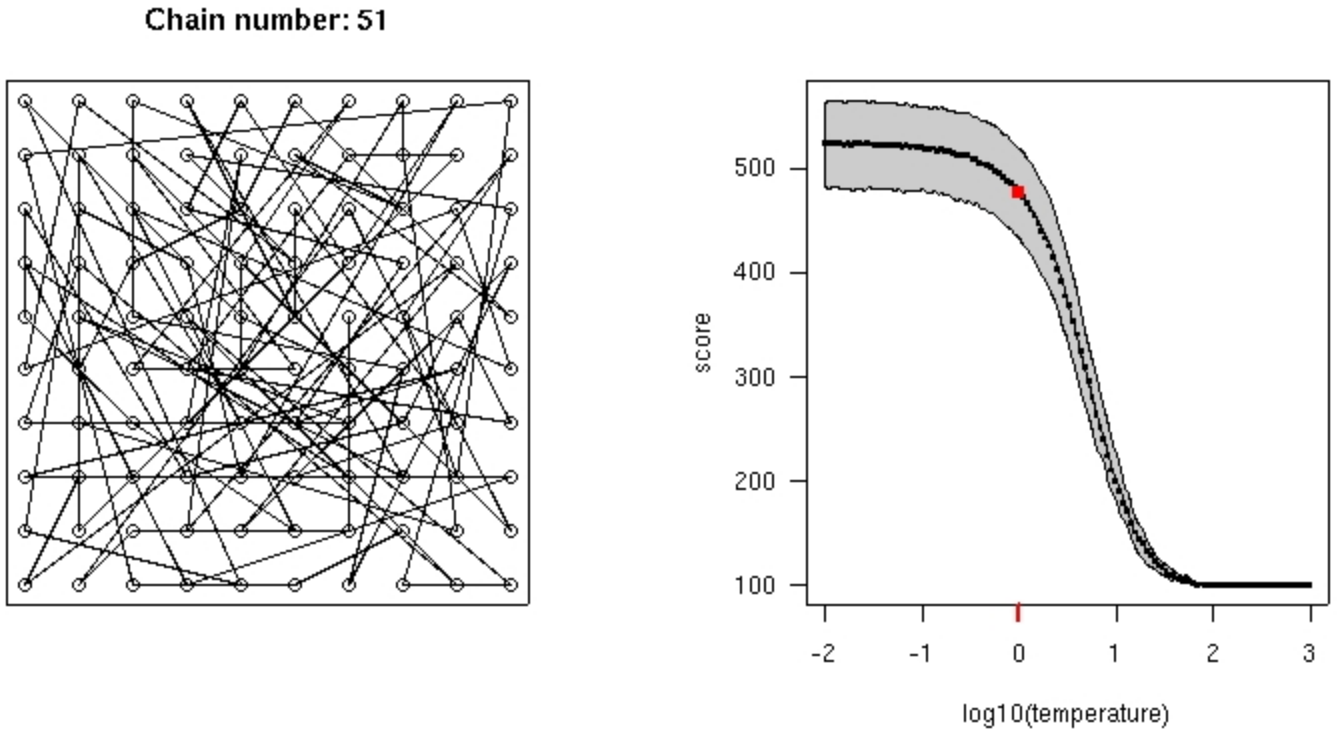
Simulated Annealing

Algorithm 6: Generic Simulated Annealing

```
1  function SimulatedAnnealing(T);
   Input  : temperature T
   Output: A feasible solution x*
2  Generate a feasible solution x; Set x* = x;
3  Generate a feasible solution x' ∈ N(x);
4  if (z(x') < z(x)) then
5  |    Set x = x';
6  |    if (z(x*) > z(x)) then x* = x;
7  else
8  |    Set x = x' with probability p = e^{-(z(x')-z(x))/(kT)};
9  end
10 if (annealing condition) then decrease T;
11 if not(terminating condition) then go to 3;
12 return x*;
```

Fonte: https://link.springer.com/book/10.1007/978-3-030-70277-9

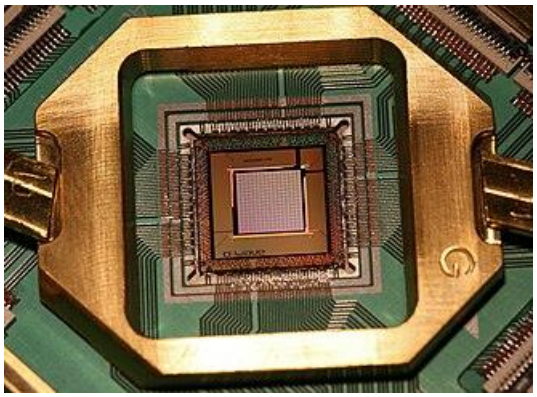
Simulated Annealing



Fonte: https://biostat.jhsph.edu/~iruczins/teaching/misc/annealing/animation.html

Quantum Computing

- Negli ultimi anni la ricerca scientifica sta lavorando intensamente nell'ambito del Quantum Computing sperando di ottenere una svolta epocale sulle capacità di calcolo utilizzando risultati della meccanica quantistica.
- Tra i risultati finora ottenuti possiamo citare il Quantum Annealing che rappresenta una generalizzazione del Simulated Annealing.
- L'industria ha già prodotto dei dispositivi hardware che eseguono algoritmi di Quantum Annealing.



Fonte: https://en.wikipedia.org/wiki/D-Wave_Systems

Algoritmi Genetici

- Gli algoritmi genetici sono stati proposti per la prima volta da Holland (1992) e si ispirano al processo evolutivo degli organismi in natura.
- Questi algoritmi definiscono un insieme di soluzioni (detti individui), che costituiscono la popolazione che ad ogni iterazione è "aggiornata".
- La popolazione è aggiornata ricombinando sottoinsiemi di individui (parent set) per ottenere nuove soluzioni. L'operazione che permette di generare un nuovo individuo è chiamata crossover.
- Sui nuovi individui è effettuata una operazione di mutazione al fine di diversificare la popolazione.
- Al termine di ogni iterazione si selezionano gli individui che faranno parte della popolazione nella prossima iterazione.
- Le soluzioni sono codificate utilizzando delle "stringhe" (cromosomi), il cui formato dipende dall'applicazione (e.g., stringhe di bit).

Algoritmi Genetici

Algoritmo Genetico

Step 1. Genera una popolazione P di soluzioni iniziali.

Step 2. Valuta il costo $f(\mathbf{x}), \forall \mathbf{x} \in P$ (funzione di fitness).

Step 3. Selezione dei genitori: seleziona un sottoinsieme G di soluzioni dall'insieme P.

Step 4. Crossover: costruisci un insieme P_G di soluzioni combinando fra loro i genitori in G.

Step 5. Mutazione: modifica casualmente alcune soluzioni in P_G.

Step 6. Selezione della popolazione: la nuova popolazione è selezionata sostituendo tutti o alcuni individui della popolazione P con gli individui nel nuovo insieme P_G utilizzando la funzione di fitness.

Step 7. Se la condizione di terminazione non è soddisfatta vai allo Step 3.

Algoritmi Genetici

Single Crossover

$$(0\ 0\ 0\ 0 \mid 1\ 1\ 1) \qquad\qquad (0\ 0\ 0\ 0\ 0\ 1\ 0)$$
$$\Rightarrow$$
$$(0\ 1\ 0\ 1 \mid 0\ 1\ 0) \qquad\qquad (0\ 1\ 0\ 1\ 1\ 1\ 1)$$

Double Crossover

$$(0\ 0 \mid 0\ 0 \mid 1\ 1\ 1) \qquad\qquad (0\ 0\ 0\ 1\ 1\ 1\ 1)$$
$$\Rightarrow$$
$$(0\ 1 \mid 0\ 1 \mid 0\ 1\ 0) \qquad\qquad (0\ 1\ 0\ 0\ 0\ 1\ 0)$$

Algoritmi Genetici

- L'operatore di mutazione consente di introdurre nella popolazione delle nuove caratteristiche che possono essere utili all'evoluzione, generando nelle future generazioni individui con fitness migliore.

Mutazione

(0 1 0 0 0 1 0)

↓

(0 1 1 0 0 1 0)

- La selezione della popolazione può essere svolta seguendo numerosi schemi che comunque dipendono dalla funzione di fitness (perché?).
- La funzione di fitness deve valutare sia il "costo" della soluzione che il suono "grado" di non ammissibilità (aggiungendo una penalità).

Set Covering Problem

- Il Set Covering Problem (SCP) è il problema di coprire le righe di una matrice $A = (a_{ij})$ di dimensioni $m \times n$ con coefficienti 0 ed 1, con un sottoinsieme di colonne di costo minimo.
- Sia x_j una variabile binaria 0-1 definita per ogni colonna come segue:

$$x_j = \begin{cases} 1, & \text{se la colonna } j \text{ di costo } c_j \text{ è in soluzione} \\ 0, & \text{altrimenti} \end{cases}$$

- Una formulazione matematica per il problema SCP è la seguente:

$$(SCP) \qquad z_{SCP} = \min \sum_{j=1}^{n} c_j x_j$$

$$s.t. \sum_{j=1}^{n} a_{ij} x_j \geq 1, \qquad i = 1, \dots, m$$

$$x_j \in \{0,1\}, \qquad j = 1, \dots n$$

Set Covering Problem

Esempio

- Si consideri il problema di Set Covering definito dai parametri che sono riassunti qui di seguito:
  - Il numero di colonne è pari a $n = 6$ e il numero di righe è $m = 3$.
  - Il vettore dei costi è $\mathbf{c} = [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6]$.
  - La matrice dei vincoli è:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

- In questo caso la soluzione ottima è $\mathbf{x}^* = [0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0]$, ossia include la colonna 2 e la colonna 3.
- Il costo della soluzione ottima è $z^*_{SCP} = 5$.

Set Covering Problem

Algoritmo Genetico per il Set Covering (Beasley and Chu, 1995)

Step 1. Genera in modo casuale una popolazione di N soluzioni iniziali.

Step 2. Seleziona due genitori: seleziona due soluzioni P_1 e P_2.

Step 3. Crossover: combina P_1 e P_2 per formare una nuova soluzione C.

Step 4. Mutazione: modifica k colonne di C in modo casuale.

Step 5. Repair: Rendi ammissibile la soluzione C.

Step 5. Se la soluzione C è già presente nella popolazione torna a Step 2.

Step 6. Sostituisci una soluzione della popolazione con la soluzione C.

Step 7. Se non è stato raggiunto il numero massimo di iterazioni, torna allo Step 2; altrimenti la soluzione è quella con il valore di fitness più piccolo.

Set Covering Problem

Rappresentazione di una soluzione

- Una soluzione è rappresentata da una stringa binaria di n bit, dove un valore a 1 del j-esimo bit indica che la colonna j è in soluzione.

1	2	3	4	5	$\cdots$	$n-1$	n
1	0	1	1	0	$\cdots$	1	0

Definizione della funzione di fitness

- La funzione di fitness f_{P_k} di una soluzione P_k è dato da:

$$f_{P_k} = \sum_{j=1}^{n} c_j s_{ij}$$

dove s_{ij} è il valore del j-esimo bit nella stringa corrispondete alla soluzione P_k.

Set Covering Problem

Selezione dei genitori

- La tecnica utilizzata per selezionare i genitori è la binary tournament selection.
- Dalla popolazione vengono selezionati in modo casuale due gruppi di soluzioni di una cardinalità prestabilita. Poi, a turno, vengono estratte due soluzioni P_1 e P_2 con fitness migliore.
- Esistono molte altre tecniche per selezionare i genitori da utilizzare nel crossover.
- Cosa proporreste come criterio di selezione?

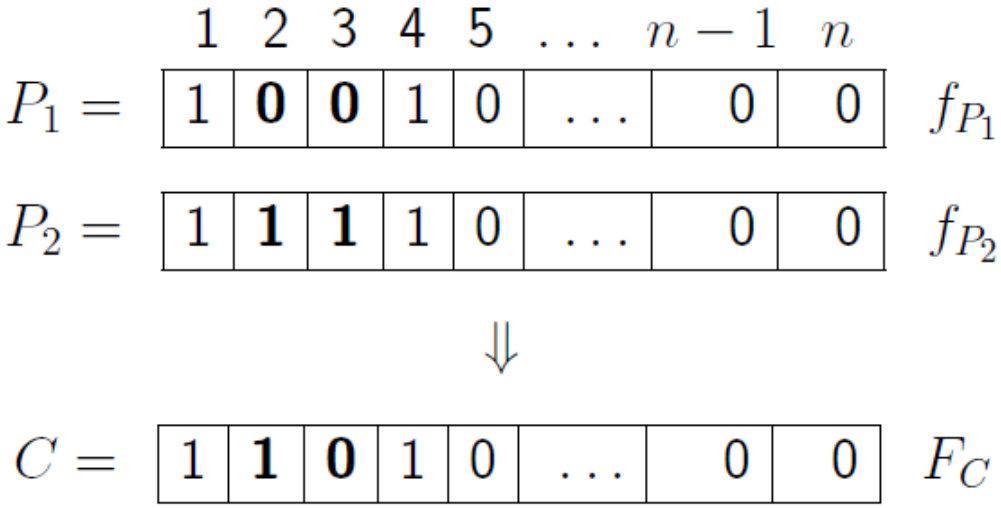
Set Covering Problem

Operazione di crossover

- Viene utilizzato un operatore di crossover, chiamato fusion crossover, che tiene conto sia della struttura sia della fitness delle soluzioni del parent set.
- L'operatore di crossover produce, a differenza degli operatori classici, una sola soluzione.
- Siano f_{P_1} e f_{P_2} i valori di fitness delle soluzioni P_1 e P_2 e sia C la nuova soluzione costruita come segue, per ogni $i = 1, \dots, n$:
  1) Se $P_1[i] = P_2[i]$, allora $C[i] = P_1[i] = P_2[i]$;
  2) Se $P_1[i] \neq P_2[i]$, allora
     a) $C[i] = P_1[i]$ con probabilità $p = f_{P_2}/(f_{P_1} + f_{P_2})$,
     b) $C[i] = P_2[i]$ con probabilità $1 - p$.

Set Covering Problem

Esempio di operazione di crossover



- Nel caso in cui $f_{P_1} = 4$ e $f_{P_2} = 6$, se $P_1[i] \neq P_2[i]$, la probabilità che $C[i] = P_1[i]$ è $p = \frac{6}{4+6} = 0.6$, mentre la probabilità che $C[i] = P_2[i]$ è $1 - p = 1 - 0.6 = 0.4$.

Set Covering Problem

Operazione di ammissibilità (Repair)

- La soluzione C generata dopo l'applicazione dell'operatore di fusion crossover può risultare non ammissibile.
- Nel caso la soluzione C non sia ammissibile possiamo applicare un operatore di ammissibilità per renderla ammissibile.
- Consideriamo i seguenti parametri:
  - I = insieme di tutte le righe;
  - J = insieme di tutte le colonne;
  - α_i = insieme delle colonne che coprono la riga $i \in I$;
  - β_j = insieme delle righe coperte dalla colonna $j \in J$;
  - S = insieme delle colonne in una soluzione;
  - U = insieme delle righe non coperte dalla soluzione;
  - w_i = numero di colonne che coprono la riga $i \in I$ in S.

Set Covering Problem

Algoritmo Operatore di Ammissibilità

Step 1. Inizializza $w_i = |S \cap \alpha_i|, \forall i \in I$.

Step 2. Inizializza $U = \{i : w_i = 0, \forall i \in I\}$.

Step 3. Per ogni riga $i \in U$:

a) Trova la prima colonna j in α_i che minimizza $c_j / |U \cap \beta_j|$;

b) Aggiungi j a S e poni $w_i = w_i + 1, \forall i \in \beta_j$, e $U = U \setminus \beta_j$.

Step 4. Per ogni colonna $j \in S$, considerati per costo c_j decrescente, se $w_i \geq 2, \forall i \in \beta_j$, poni $S = S \setminus \{j\}$ e $w_i = w_i - 1, \forall i \in \beta_j$.

Set Covering Problem

Aggiornamento della popolazione

- Dopo che una nuova soluzione C è stata prodotta, la popolazione viene aggiornata rimuovendo dalla popolazione una soluzione scelta casualmente fra le soluzioni con valore di fitness inferiore al valore medio (below-average fitness) ed inserendo C nella popolazione.
- Quali alternative potevano esserci?

Vehicle Routing Problem

- Dato un grafo $G = (V, A)$, dove il vertice 0 rappresenta il deposito e i vertici $V' = \{1, \dots, n\}$ rappresentano i clienti.
- Ad ogni cliente $i \in V'$ è associata una domanda positiva q_i.
- Ad ogni arco $(i, j) \in A$ è associato un costo non negativo c_{ij}.
- Presso il deposito sono disponibili M veicoli identici di capacità Q.
- Il costo di un viaggio (o route) è dato dalla somma dei costi degli archi che compongono il viaggio.
- Ogni veicolo deve effettuare un viaggio (route) partendo (o tornando) con un carico minore o al più uguale alla capacità Q.
- Ogni viaggio deve iniziare e terminare al deposito.
- Ogni cliente deve essere visitato una ed una sola volta (e da un solo veicolo).

Vehicle Routing Problem

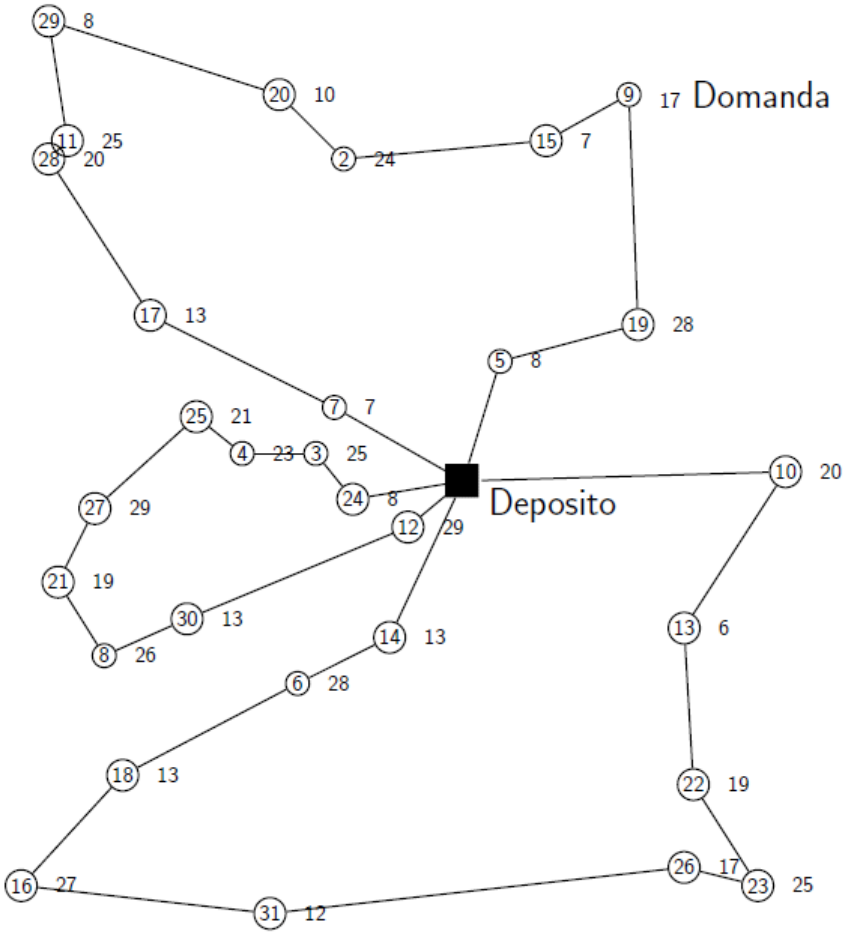
Obiettivo

- Disegnare M route, una per ogni veicolo, in modo da visitare tutti i clienti una e una sola volta e minimizzare la somma dei costi delle route.
- Il costo di ciascuna route è data dalla somma dei costi c_{ij} degli archi percorsi.
- Possiamo denotare una route come:
  - Insieme dei vertici visitati $R = \{0, i_1, i_2, \dots, i_{|R|-1}, 0\}$;
  - Insieme degli archi visitati $R = \{(0, i_1), (i_1, i_2), \dots, (i_{|R|-1}, 0)\}$.
- Il costo della route R è dato da $c(R) = \sum_{(i,j)\in R} c_{ij}$.

Vehicle Routing Problem

Esempio

$n = 30$ - $M = 3$ - $Q = 200$



Vehicle Routing Problem

Tabu Search per il Vehicle Routing Problem (Gendreau, Hertz e Laporte (1994))

- Si suppone che al deposito siano disponibili un numero infinito di veicoli.
- Sia $S = (R_1, R_2, \dots, R_k)$ una soluzione per il VRP di k route.
- Ad ogni soluzione S si possono associare le seguenti due funzioni obiettivo (funzioni di fitness):

$$F_1(S) = \sum_{r \in S} \sum_{(i,j) \in R_r} c_{ij}$$

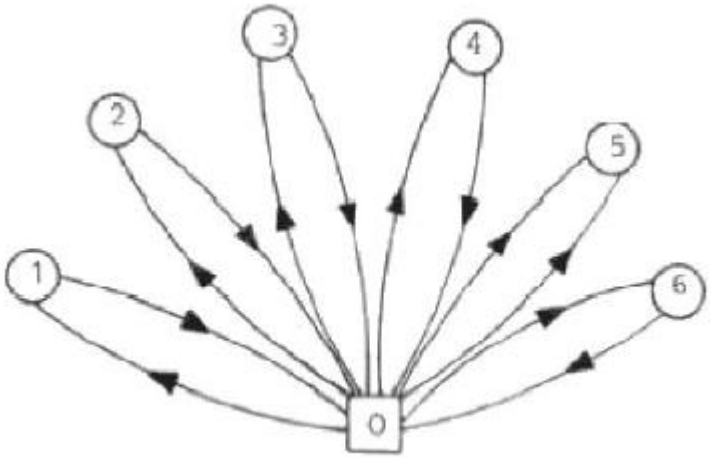
$$F_2(S) = F_1(S) + \alpha \left(\sum_{r \in S} \max \left\{ 0, \sum_{i \in R_r} q_i - Q \right\} \right)$$

- La funzione $F_2(S)$ include una penalizzazione nel caso sia violato il vincolo di capacità.

Vehicle Routing Problem

Costruzione della soluzione iniziale

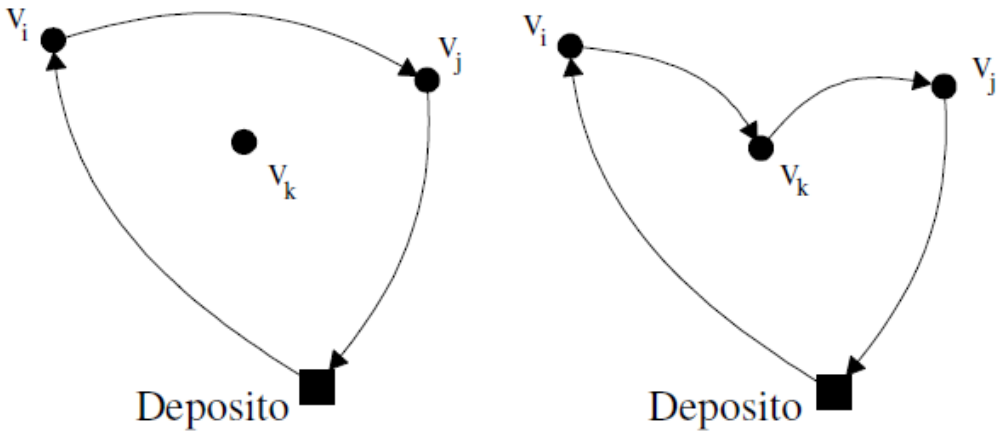
- La soluzione iniziale è costituita da n route del tipo $R_r = (0, r, 0)$, per $r = 1, \dots, n$.



Vehicle Routing Problem

Inserimento di un cliente in una route

- Un cliente può essere inserito in una route emergente utilizzando l'extra-mileage per valutare la miglior posizione d'inserimento.



A. Route iniziale B. Route finale

Extra Mileage

$$em(i,k,j) = c_{ik} + c_{kj} - c_{ij}$$

Vehicle Routing Problem

- Sia S la soluzione corrente a una data iterazione k.
- Indichiamo con (i, r) l'operazione (mossa) di inserimento del cliente i nella route r della soluzione S.
- Ora definiamo le mosse che il tabu search esegue ad ogni iterazione k, dopodiché definiremo lo schema di funzionamento generale.

Algoritmo 1: Tabu Search all'iterazione *k*

Step 1. Se $\text{mod}(k, 10) = 0$ (i.e., esegue questo step ogni 10 iterazioni):

Se le precedenti 10 soluzioni sono ammissibili, $\alpha = \alpha/2$;

Se le precedenti 10 soluzioni sono non ammissibili, $\alpha = \alpha * 2$.

(Nota: il parametro α è il peso della componente che misura la non ammissibilità di una soluzione).

Step 2. Sia W un insieme di q clienti scelti casualmente dall'insieme V'.

Vehicle Routing Problem

Step 3. Per ogni cliente $i \in W$:

- Considera tutte le mosse non tabu (i, r), dove r è una route di S che contiene almeno uno dei p clienti più vicini ad i.
  Sia Φ l'insieme delle soluzioni prodotte.
- Determina la soluzione S' tale che $F_2(S') = \min_{S'' \in \Phi} \{F_2(S'')\}$.
  Sia (i, r^*), con $r^* \in S$, la mossa che produce S'.
- Se $F_2(S') < F_2(S^*)$, allora $S^* = S'$.
- Poni $S = S'$.
- Dichiara tabu per θ iterazioni la mossa (i, r^*), dove θ è scelto nell'intervallo [5,10].

Vehicle Routing Problem

Algoritmo Tabu Search (schema generale)

Step 1. Inizializzazione

Sia $m = \lceil \sum_{i=1}^{n} q_i / Q \rceil$.

Sia S la soluzione iniziale.

Poni $S^* = S$, dove S^* rappresenta la miglior soluzione prodotta.

Poni $\alpha = 1$ e poni $k = 0$.

Step 2. Miglioramento della soluzione

Esegui $k_{max} = 50n$ iterazioni dell'Algoritmo 1 con $p = \min\{n, 5\}$ e $q = \min\{n, m\}$.

Step 3. Intensificazione

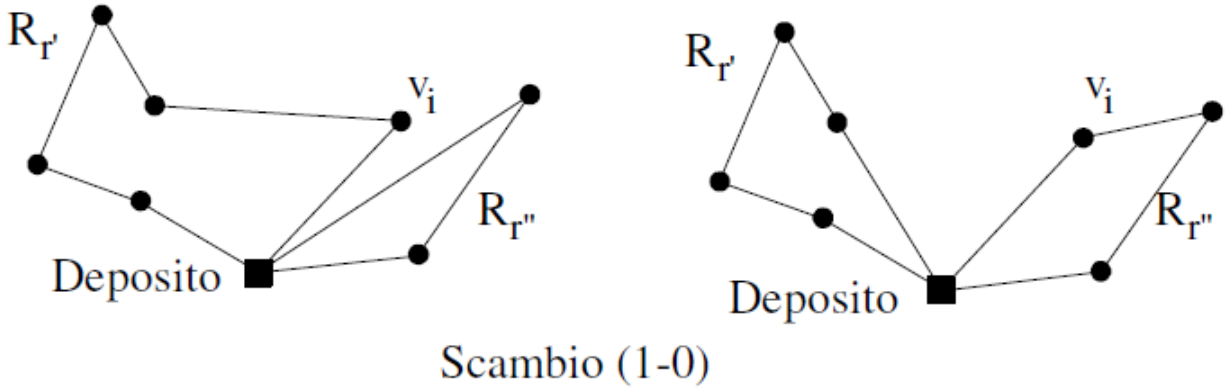
Esegui $k_{max} = 100$ iterazioni dell'Algoritmo 1 con $p = \min\{n, 10\}$ e $q = n$.

Step 3. Post-ottimizzazione della miglior soluzione

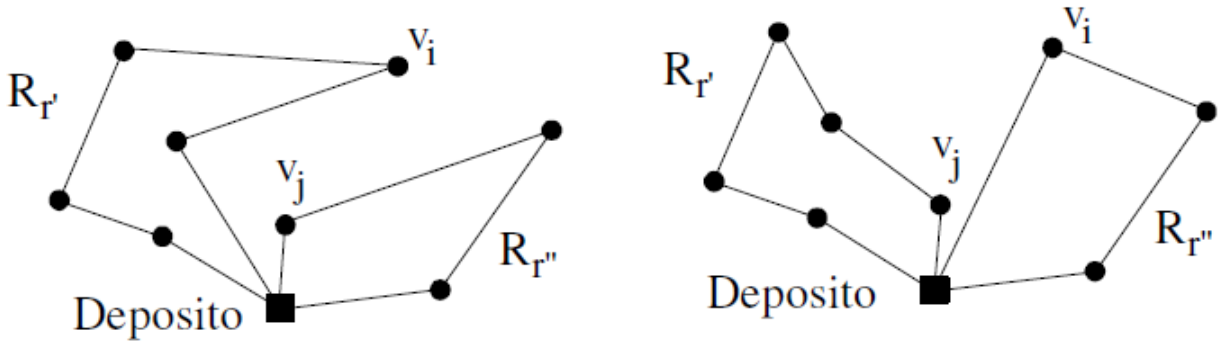
Migliora S^* con una procedura 3-opt (scambi di clienti tra route).

Vehicle Routing Problem

Post-Ottimizzazione: scambi fra route



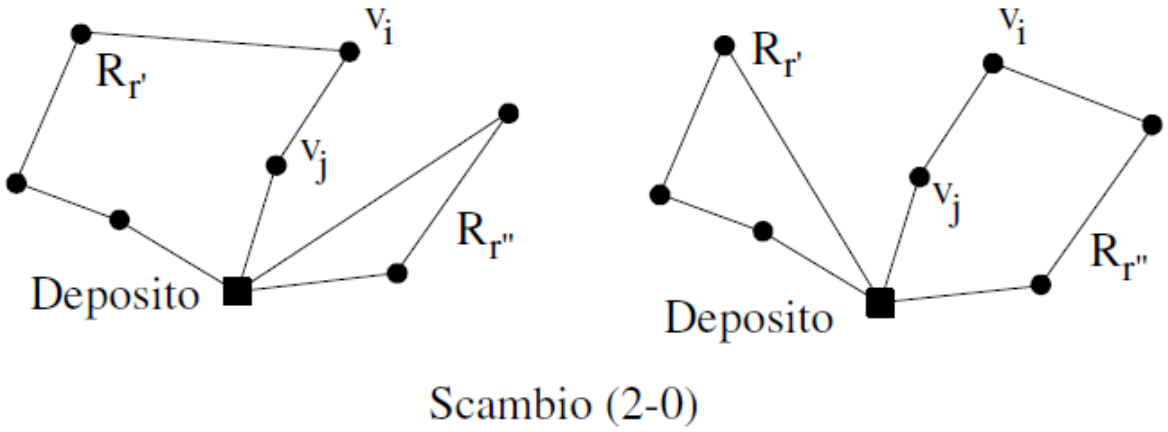
Scambio (1-0)



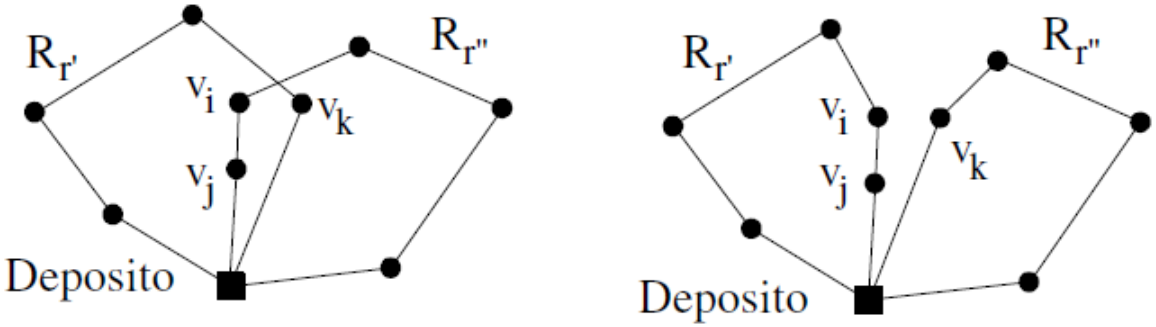
Scambio (1-1)

Vehicle Routing Problem

Post-Ottimizzazione: scambi fra route



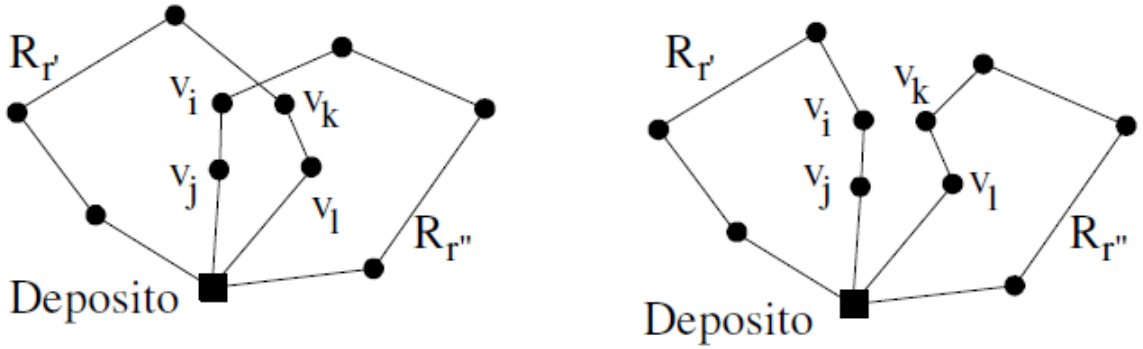
Scambio (2-0)



Scambio (2-1)

Vehicle Routing Problem

Post-Ottimizzazione: scambi fra route

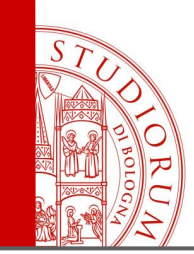


Scambio (2-2)

Quale altra mossa si potrebbe implementare?

Intelligenza Artificiale

- Al giorno d'oggi, quando si parla di supporto alle decisioni, la mente va subito all'Intelligenza Artificiale (AI).
- Come abbiamo visto finora, molti problemi relativi al supporto alle decisioni possono essere risolti con la programmazione matematica e l'ottimizzazione combinatoria.
- Ora vogliamo ricordare in cosa consiste l'Intelligenza Artificiale oggi per comprendere meglio la relazione tra ottimizzazione e AI.
- Abbiamo già detto che oggi l'AI copre l'area degli algoritmi euristici, consentendo di determinare delle soluzioni di un problema, senza riuscire a dimostrare che le soluzioni trovate siano ottime oppure fornire delle stime della distanza dalla soluzione ottima.
- Non solo, molti algoritmi di AI spesso non danno neppure la garanzia che la soluzione trovata sia "corretta".

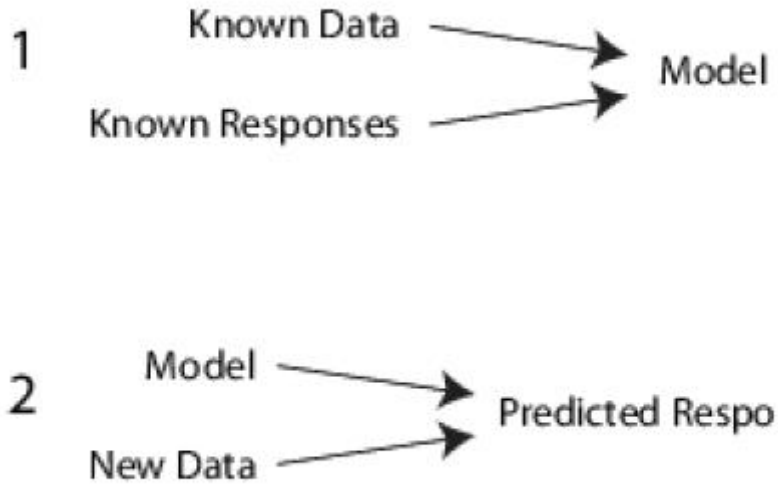


Intelligenza Artificiale

Alcuni esempi di applicazioni

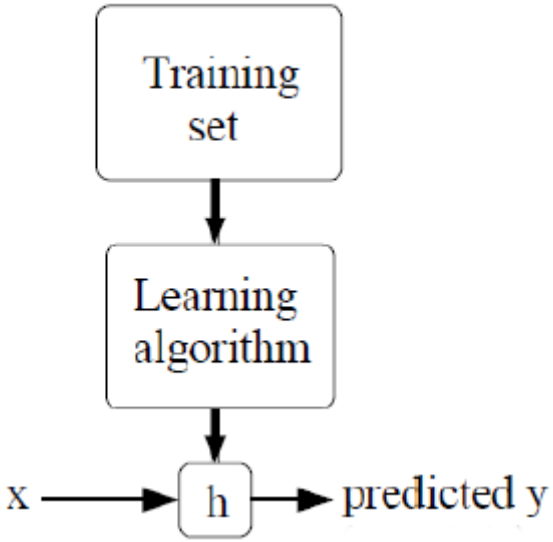
- Diagnosi medica: definire se un paziente è affetto da una patologia, etc.
- Previsioni del tempo: stabilire le condizioni meteo per domani, etc.
- Spam filtering: identificare quali messaggi sono spam oppure no.
- Riconoscimento scrittura: per digitalizzare un testo scritto a mano, etc.
- Face detection: riconoscere un volto tra quelli in un archivio, etc.
- Riconoscimento vocale: riconoscimento di un comando vocale, etc.
- Topic spotting: classificare articoli, etc.
- Customer segmentation: predire il comportamento dei clienti a fronte di una promozione, di determinati prodotti, etc.
- Fraud detection: identificare comportamenti fraudolenti, etc.
- Etc.

- Nel contesto del machine learning abbiamo due possibili approcci: supervisionato e non-supervisionato.
- Un algoritmo di apprendimento supervisionato usa un insieme di dati di input di cui è noto il corrispondente output corretto e "impara" a rispondere in modo "autonomo" quando poi si avranno nuovi dati in input.



Intelligenza Artificiale

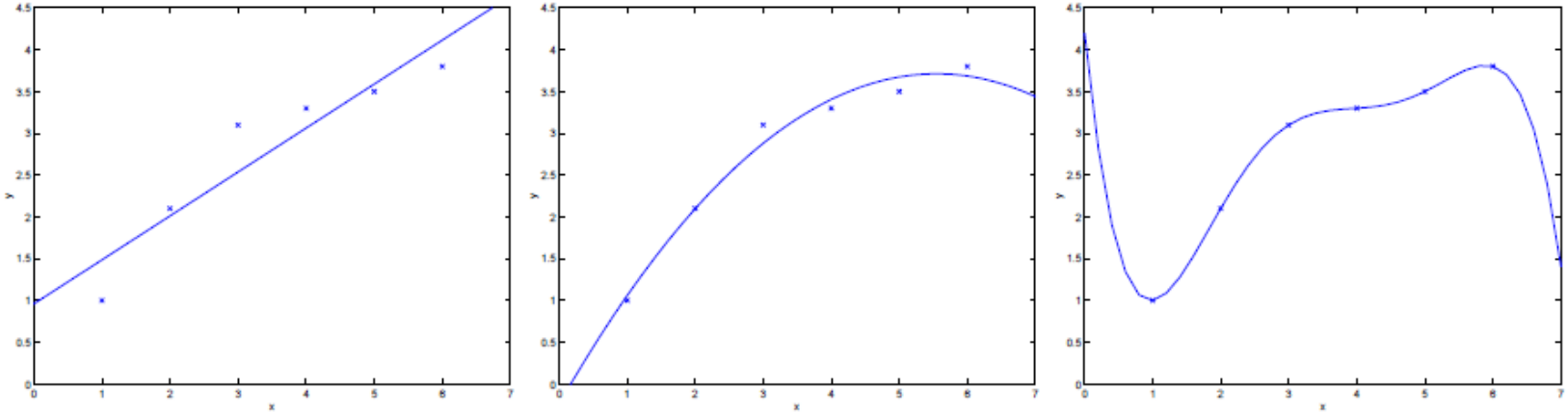
- Il processo di apprendimento può essere riassunto come segue:



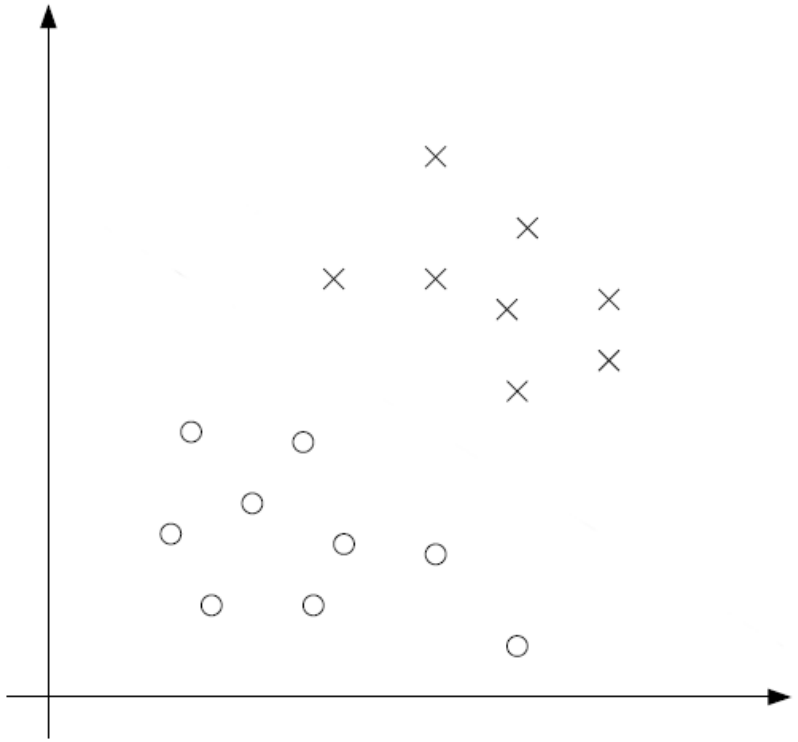
- Quando le variabile che forniscono la previsione sono continue, siamo solitamente di fronte a un problema di "regressione".
- Quando l'output può assumere solo un insieme discreto di valori è probabile che ci troviamo di fronte a un problema di "classificazione".

Intelligenza Artificiale

- Una regressione ci permette di costruire quelle funzioni che meglio dovrebbero approssimare il comportamento del fenomeno di nostro interesse.
- Per costruire la funzione possiamo usare i dati a nostra disposizione.
- Quando il modello sarà definito potremo svolgere delle previsioni rispetto ad altri possibili input.

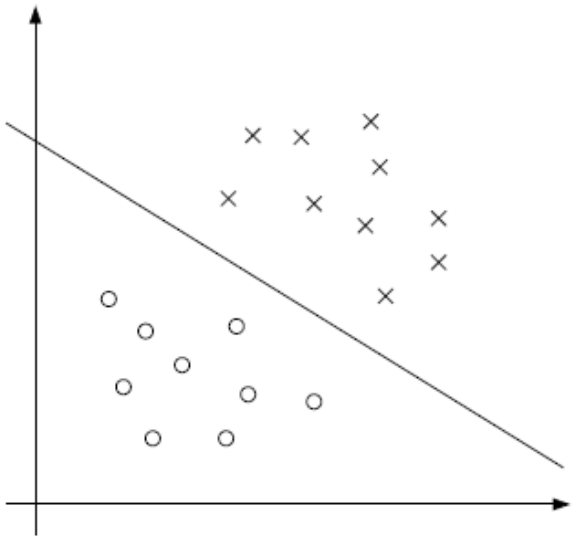


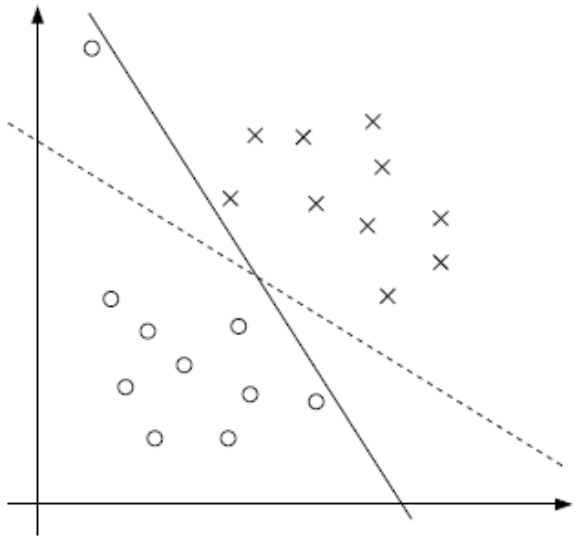
- Quando dobbiamo svolgere una classificazione, dobbiamo definire a quale elemento dell'insieme discreto di output deve corrispondere un certo input (quindi a quale "categoria" appartiene).



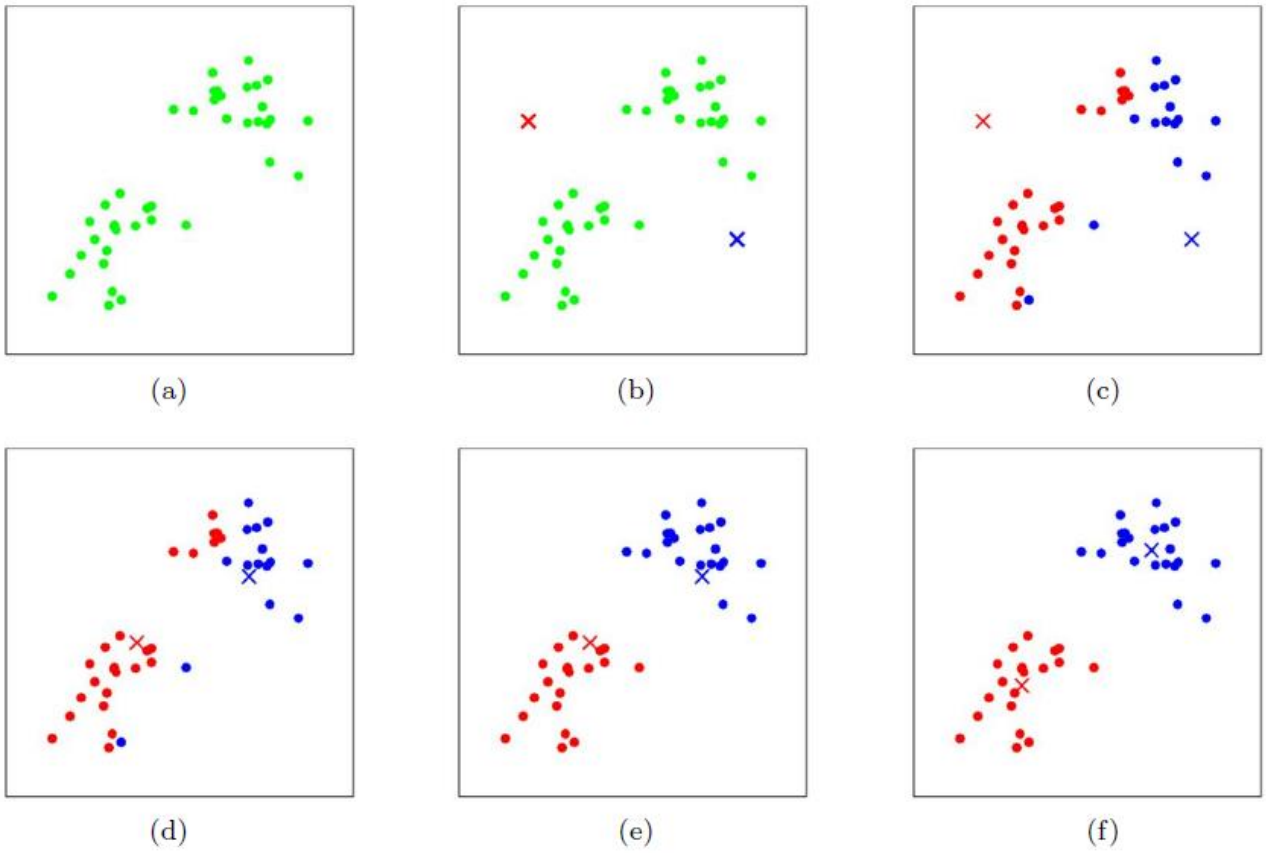
Intelligenza Artificiale

- Tra i metodi di classificazione dell'AI uno dei più efficaci è noto come Support Vector Machine (SVM), che è un metodo supervisionato.
- Le SVMs sono basate sulla definizione di iperpiani di separazione.





- Tra i metodi di classificazione non-supervisionato possiamo citare l'algoritmo k-Means Clustering.



Regressione Lineare

- Vediamo come possiamo costruire un semplice modello lineare.
- Possiamo ricavare un modello lineare "*interpolando*" due punti, ossia trovando l'equazione della retta passante per i due punti dati.
- Normalmente, però, si hanno più di due punti a disposizione e non sono quasi mai "*allineati*". In alcuni casi lo sono quasi, mentre in altri sono ben lontani dall'esserlo. In tutti questi casi, il problema è trovare l'equazione della retta che "*meglio approssima*" i punti dati.
- Perché dobbiamo avere più di due punti?
- Perché non sono allineati?
- Cosa significa che "*meglio approssima*" i punti dati? Perché ci interessa tanto calcolare questa funzione?
- Qualche idea su come determinare l'equazione della retta che "*meglio approssima*" i punti dati?

Regressione Lineare

- **Esempio:** Supponiamo di avere i dati sulle vendite di case nuove in una determinata regione nel corso di un anno, che riassumiamo nella seguente tabella:

Prezzo Vendita (migliaia di Euro)	150-169	170-189	190-209	210-229	230-249	250-269	270-289
Numero di case vendute	126	103	82	75	82	40	20

Vorremo utilizzare questi dati per costruire una funzione lineare che "*meglio approssima*" i punti dati. Perché una funzione lineare?

Un primo problema da risolvere riguarda i prezzi di vendita, che sono espressi in termini di "*intervalli*". Qual è il problema? Come lo si risolve?

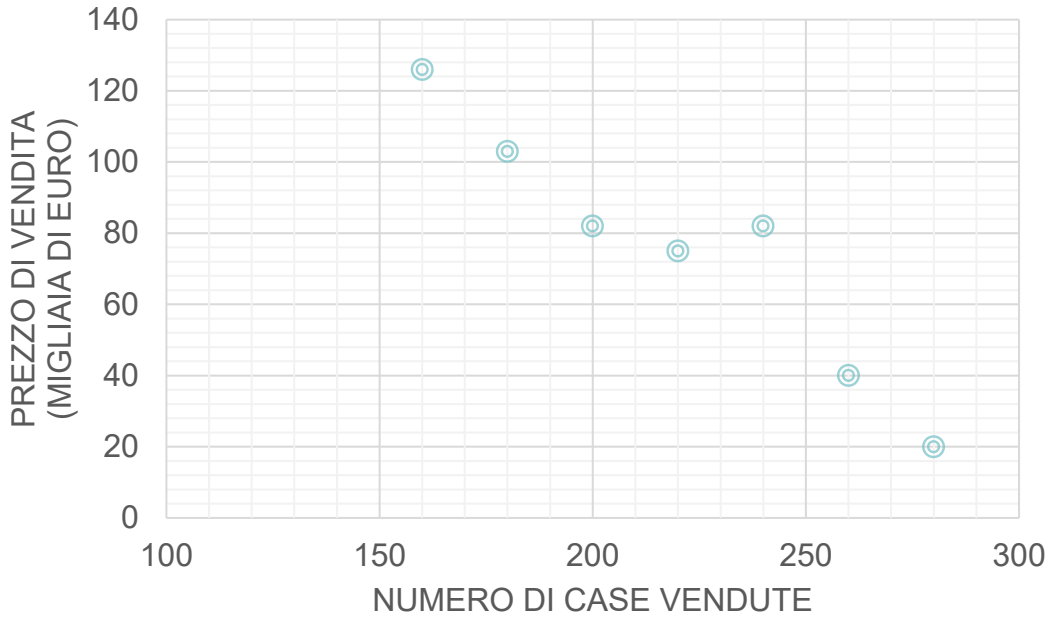
Il metodo più semplice ed efficace consiste nel sostituire ciascun intervallo con un singolo prezzo, che possiamo far corrisponde al valore al centro dell'intervallo.

Regressione Lineare

- La nuova tabella è la seguente:

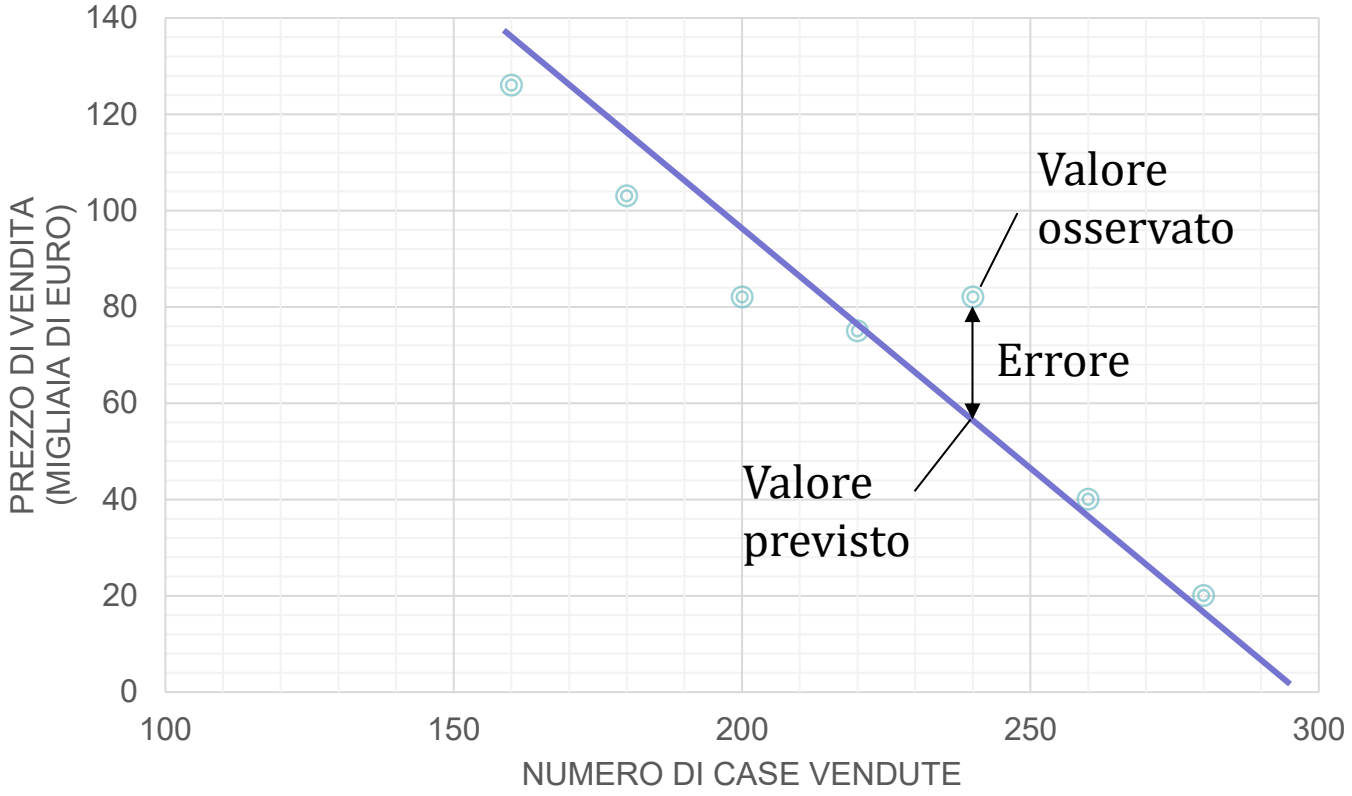
Prezzo Vendita (migliaia di Euro)	160	180	200	220	240	260	280
Numero di case vendute	126	103	82	75	82	40	20

Con un Excel possiamo "disegnare" i punti su un piano cartesiano.



Regressione Lineare

- Vorremo che la retta fosse il più vicina possibile ai dati di vendita reali (dati osservati). In altre parole, vogliamo minimizzare la differenza "complessiva" tra i valori previsti (sulla retta) e quelli osservati.



Regressione Lineare

- Una possibile retta di regressione può essere calcolata con il metodo dei minimi quadrati.
- Siano dati n punti (dati osservati): (x_1, y_1), (x_2, y_2), …, (x_n, y_n)
- La retta che meglio approssima i dati osservati in base al metodo dei minimi quadrati ha la forma:

$$y = mx + b$$

dove

$$m = \frac{n(\sum_{i=1}^{n} x_i y_i) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n(\sum_{i=1}^{n} x_i^2) - (\sum_{i=1}^{n} x_i)^2}$$

$$b = \frac{\sum_{i=1}^{n} y_i - m(\sum_{i=1}^{n} x_i)}{n}$$

- Come sono state derivate queste formule? E perché funzionano?

Regressione Lineare

- **La buona/cattiva notizia:** per poter derivare la retta di regressione dei minimi quadrati dobbiamo conoscere il calcolo differenziale.
- Ricordiamo che i metodi di regressione sono una delle metodologie di base utilizzate nell'ambito "*Data Science*" e, in particolare, nell'ambito "*Machine Learning*".
- Per determinare la retta di regressione si vuole minimizzare l'errore tra i punti forniti dalla retta e i punti relativi alle osservazioni.
- Quindi, la retta di regressione che meglio approssima i dati osservati in base al metodo dei minimi quadrati ha la forma:

$$y = mx + b \quad \Longrightarrow \quad \bar{y}_i = mx_i + b$$

- L'errore e_i per ciascun punto (x_i, y_i) relativo alle osservazioni è:

$$e_i = |\bar{y}_i - y_i| = |mx_i + b - y_i|$$

Regressione Lineare

- L'errore complessivo è dato dalla somma degli errori e_i:

$$Errore = \sum_{i=1}^{n} |\bar{y}_i - y_i| = \sum_{i=1}^{n} |mx_i + b - y_i|$$

- Il problema consiste nel determinare il coefficiente angolare m e l'intercetta b, che consentono di minimizzare l'errore. Per cui dobbiamo trovare il minimo della seguente funzione:

$$e(m, b) = \sum_{i=1}^{n} |mx_i + b - y_i|$$

- Siccome il valore assoluto rende più difficile la soluzione del problema si è preferito usare il "quadrato" (perché?):

$$e(m, b) = \sum_{i=1}^{n} (mx_i + b - y_i)^2$$

Regressione Lineare

- Per cui vogliamo trovare il punto (m, b) in cui la funzione $e(m, b)$ ha un minimo:

$$e(m, b) = \sum_{i=1}^{n} (mx_i + b - y_i)^2$$

- Si noti che la funzione $e(m, b)$ è una funzione quadratica. Per cui se ha un minimo relativo, questo è anche minimo globale.
- Per trovare il punto di minimo applichiamo le condizioni necessarie del primo ordine (i.e., trovare il punto in cui $\nabla f = \mathbf{0}$):

$$\nabla e = \begin{bmatrix} \frac{\partial e}{\partial m} \\ \frac{\partial e}{\partial b} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} 2(mx_i + b - y_i)x_i \\ \sum_{i=1}^{n} 2(mx_i + b - y_i) \end{bmatrix}$$

Regressione Lineare

- Per trovare il punto di minimo dobbiamo risolvere il seguente sistema lineare:

$$\nabla e = \begin{bmatrix} \sum_{i=1}^{n} 2(mx_i + b - y_i)x_i \\ \sum_{i=1}^{n} 2(mx_i + b - y_i) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Il sistema lo possiamo scrivere anche come segue:

$$\begin{cases} \sum_{i=1}^{n} mx_i^2 + bx_i - x_i y_i = 0 \\ \sum_{i=1}^{n} mx_i + b - y_i = 0 \end{cases} \Rightarrow \begin{cases} \sum_{i=1}^{n} mx_i^2 + \sum_{i=1}^{n} bx_i - \sum_{i=1}^{n} x_i y_i = 0 \\ \sum_{i=1}^{n} mx_i + \sum_{i=1}^{n} b - \sum_{i=1}^{n} y_i = 0 \end{cases}$$

Regressione Lineare

- Semplificando la notazione e ricavando b, abbiamo:

$$\begin{cases} m\sum x_i^2 + b\sum x_i = \sum x_i y_i \\ m\sum x_i + nb = \sum y_i \end{cases} \Rightarrow \begin{cases} m\sum x_i^2 + b\sum x_i = \sum x_i y_i \\ b = \dfrac{\sum y_i - m\sum x_i}{n} \end{cases}$$

- Se sostituiamo b nella prima equazione abbiamo:

$$\begin{cases} m\sum x_i^2 + \dfrac{1}{n}\sum x_i \sum y_i - \dfrac{1}{n} m(\sum x_i)^2 = \sum x_i y_i \\ b = \dfrac{\sum y_i - m\sum x_i}{n} \end{cases}$$

- Moltiplicando la prima equazione per n e raccogliendo m, si ha:

$$\begin{cases} m\left(n\sum x_i^2 - (\sum x_i)^2\right) = n\sum x_i y_i - \sum x_i \sum y_i \\ b = \dfrac{\sum y_i - m\sum x_i}{n} \end{cases}$$

Regressione Lineare

- Dal seguente sistema:

$$\begin{cases} m\left(n\sum x_i^2 - (\sum x_i)^2\right) = n\sum x_i y_i - \sum x_i \sum y_i \\ b = \dfrac{\sum y_i - m\sum x_i}{n} \end{cases}$$

si ricavano facilmente le espressioni per calcolare il coefficiente angolare m e l'intercetta b:

$$\begin{cases} m = \dfrac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2} \\ b = \dfrac{\sum y_i - m\sum x_i}{n} \end{cases}$$

Conclusioni

- Quando dobbiamo risolvere un problema dobbiamo costruire un modello che sia risolvibile da un algoritmo adeguato alla complessità del problema e alla tipologia di istanze che dobbiamo risolvere.
- Il problema del knapsack è solo uno degli esempi più semplici di applicazione della programmazione matematica e ottimizzazione combinatoria a problemi di ottimizzazione.
- Nell'ambito della gestione aziendale (management science) l'uso della matematica consente di definire degli algoritmi per il supporto alle decisioni.
- Le keyword di moda oggi che identificano i campi di applicazione sono business analytics, data science, etc.
- La disciplina è nota come operations research (ricerca operativa) e fa parte del settore decision science (scienze decisionali).

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI RIMINI

Marco A. Boschetti
Dipartimento di Matematica
marco.boschetti@unibo.it

https://www.unibo.it/sitoweb/marco.boschetti
http://isi-personale.csr.unibo.it/marco.boschetti