Intelligenza Artificiale

Intelligenza Artificiale: capacità di un computer di eseguire compiti comunemente associati a esseri intelligenti

Intelligenza Naturale (intelligenza = sopravvivenza)**:** se dimostriamo intelligenza, allora sopravviveremo e prospereremo nei contesti sociali e geografici in cui risiediamo e incontriamo.

Abilità primarie (classificazione dell'intelligenza)

- Fluidità verbale
- Abilità numerica e spaziale
- Inferenza
- Velocità nella percezione
- Memoria

Turing: pensiero impossibile senza coscienza **(tutti pensano). L'unico modo per essere sicuri che una macchina pensi è essere la macchina e sentirsi pensare**. L'unico modo per sapere che un uomo pensa è essere quell'uomo.

Calcolo Sub-Simbolico (subconscio)**:** ottimo per il riconoscimento delle immagini e molti problemi che non possono essere risolti algoritmicamente

- Rappresentazione della conoscenza logica
- Basato su **Reti neurali** (rete feed forward e ricorrente) e **Machine Learning**
- **Non è possibile fornire spiegazioni per le decisioni**
- Computazione (processi mentali): processo emergente di reti interconnesse di unità semplici.
- **Apprendimento profondo**
- **Intelligenza dello sciame**

Calcolo Simbolico (consapevole)**: sistemi esperti e sistemi di supporto alle decisioni (basati sulla conoscenza)**

- Risolvono problemi utilizzando la rappresentazione della conoscenza e la **ricerca** in ampi spazi (alto livello): **ottimizzazione**, ricerca in contraddittorio (giochi) e problemi di soddisfazione dei vincoli
- Costruiscono dinamicamente la soluzione e ne spiegano il motivo
- Ricercano e generano soluzioni **guidate da regole strutturate**
- **Non appropriati per problemi come il riconoscimento delle immagini (non può essere risolto algoritmicamente)**
- **Logica e programmazione dei vincoli**
- **Ontologie** (rappresentazione della conoscenza logica)

Temi e tecniche

Ragionamento: inferenza e pianificazione

- Programmazione **logica** PROLOG e dei vincoli: proposizionale, del primo ordine e modale (epistemico, temporale, deontico)
- Incerto: ragionamento probabilistico, processi decisionali di Markov, reti Bayesane

Apprendimento e analisi dei dati: supervisionato e no, rinforzo e induttivo

Comunicazione: elaborazione del linguaggio naturale, comunicazione uomo-macchina e traduzione automatica

Percezione: acustica e visiva

Robotica: sensori, attuatori e processo decisionale autonomo

Sistemi esperti: medicinale e finanza

Programmazione naturale: algoritmi genetici e informatica delle formiche

Modellazione cognitiva: stati cognitivi e emozioni

Tecniche per rappresentare la conoscenza e il ragionamento

- **Deduzione:** permette di derivare conseguenze dell'assunto
  - B può essere derivato da A se B è una conseguenza logica di A (A |= B)
  - Data la verità dell'assunzione A segue la verità della conclusione B
- **Induzione:** permette di inferire B da A dove B non segue necessariamente A
  - Le premesse sono una forte prova della verità della conclusione
  - Derivazione di principi generali da osservazioni: abbiamo visto cigni bianchi deriviamo la regola “tutti i cigni sono bianchi”
- **Abduzione:** permette di inferire A una spiegazione di B
  - La precondizione A è abdotta dalla conseguenza B
  - Data una teoria T e un'osservazione O deduciamo (abduciamo) e spieghiamo E Tale che T U E |= O (T U E è coerente)

Reti neurali

Ispirato al modello del cervello

- Molte unità computazionali (neuroni) con bassa potenza di calcolo
- Molte connessioni (ponderate)
- Controllo distribuito altamente parallelo

Approccio completamente diverso da quello simbolico: conoscenza non esplicita, incorporata nella struttura della rete e nei pesi delle connessioni.

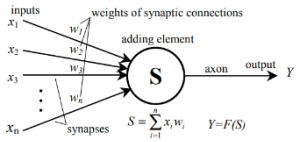
Capacità di apprendimento

- Imparare dalle differenze tra obiettivo noto e fatti osservati (I/O)
- Apprendimento delle funzioni matematiche associate al calcolo dei nodi

Il neurone artificiale

$$Y = F\left(\sum_{i=1}^{n} x_i \cdot w_i\right)$$

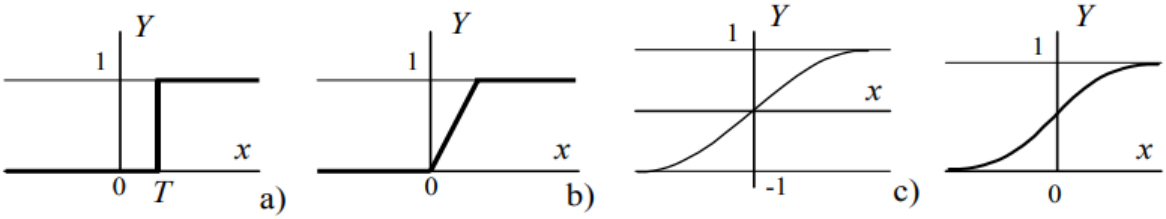
F: funzione di attivazione



Struttura generale di un neurone

La condizione per attivare un neurone è dove è $\sum_{i=1}^{n} w_i x_i - \theta \geq 0$ la soglia della funzione di trasferimento θ

Funzioni di attivazione



a) Salto b) Soglia c) Bersaglio iperbolico d) Funzione logistica

La regola dell'apprendimento percettivo

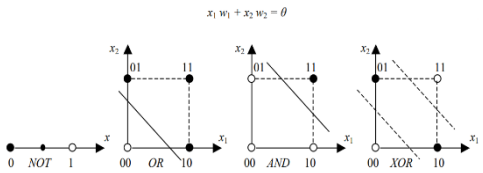
Regola utilizzata per **aggiornare i pesi** (aggiornamento per la regressione lineare):

Converge dati sono $w_i <- w_i + \alpha (y - h_{\mathbf{w}}(\mathbf{x})) * x_i$ in un separatore lineare perfetto se i punti separabili in modo lineare

Rappresentazione di funzioni booleane

NOT OR AND XOR

x_1	x_2	$\overline{x_1}$	$x_1 + x_2$	$x_1 \cdot x_2$	$x_1 \oplus x_2$
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	1	0	1	1	0



Agenti Intelligenti

Agenti e ambienti

- **Agenti:** umani, robot, soft-bot, termostati, ecc.
- **Funzione agente:** esegue il mapping dalle storie percettive alle azioni
- **Programma agente:** viene eseguito sull'**architettura** fisica per produrre f

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

Razionalità ⇒ esplorazione, apprendimento, autonomia

- **Misura delle prestazioni** fisse**:** valuta la **sequenza dell'ambiente**
- **Agente razionale:** sceglie qualsiasi azione massimizzi il valore **atteso** della misura della performance **data la sequenza percettiva fino ad oggi**
- **Razionale ≠ onnisciente:** le percezioni potrebbero non fornire tutte le informazioni rilevanti
- **Razionale ≠ chiaroveggente:** i risultati dell'azione potrebbero non essere quelli previsti
- Razionale ≠ successo

PEAS (**P**erformance measure, **E**nvironment, **A**ctuators, **S**ensors): per progettare un agente razionale, dobbiamo specificare **l'ambiente del compito**

- Taxi automatizzato
  - **Misurazioni di prestazione:** sicurezza, profitti, comodità, ...
  - **Ambiente:** strade/autostrade statunitensi, traffico, pedoni, meteo, ...
  - **Attuatori:** sterzo, acceleratore, freno, clacson, altoparlante/display, ...
  - **Sensori:** video, accelerometri, indicatori, sensori motore, GPS, ...
- Agente di acquisti su Internet

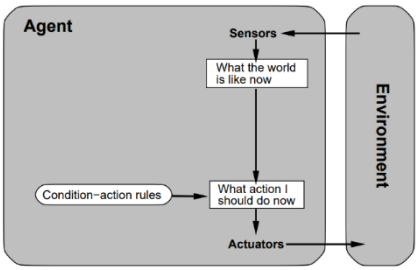
- **Misurazioni di prestazione:** prezzo, qualità, adeguatezza, efficienza
- **Ambiente:** siti WWW attuali e futuri, fornitori, spedizionieri
- **Attuatori:** mostra all'utente, segui l'URL, compila il modulo
- **Sensori:** Pagine HTML (testo, grafica, script)

Tipi di ambiente: determina in gran parte la progettazione dell'agente

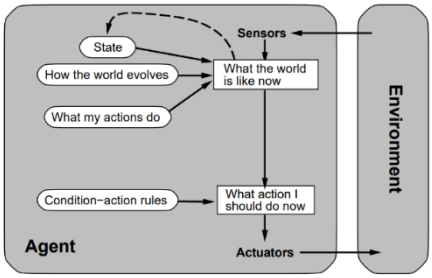
- Il mondo reale è (ovviamente) parzialmente osservabile, stocastico, sequenziale, dinamico, continuo, multi-agente

	Solitario	Backgammon	Internet shopping	Taxi
Osservabile	Si	Si	No	No
Deterministico	Si	No	In parte	No
Episodico	No	No	No	No
Statico	Si	Quasi	Quasi	No
Discreto	Si	Si	Si	No
Agente singolo	Si	No	Si (eccetto le aste)	No

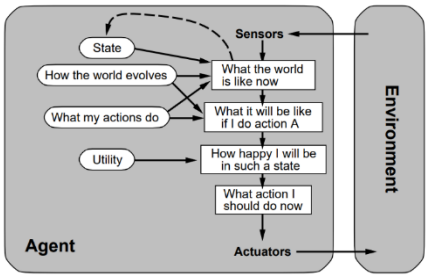
Tipi di agenti



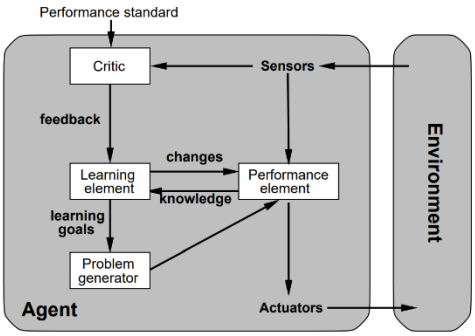
Semplici agenti riflessi



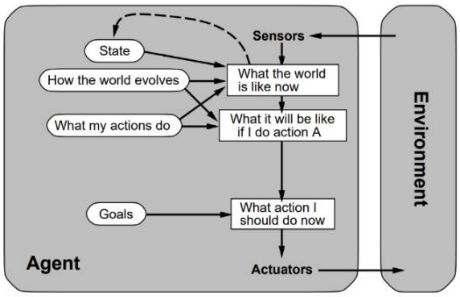
Agenti riflessi con stato



Agenti basati sugli obiettivi



Agenti basati su utility



Tutti questi possono essere trasformati in agenti di apprendimento

Gli agenti interagiscono con **gli ambienti** attraverso **attuatori** e **sensori**

La funzione agente descrive ciò che l'agente fa in tutte le circostanze

La misura delle prestazioni valuta la sequenza dell'ambiente

Un agente **perfettamente razionale** massimizza le prestazioni previste

I programmi dell'agente implementano (alcune) funzioni dell'agente

Le descrizioni di **PEAS** definiscono gli ambienti delle attività

Gli ambienti sono classificati in base a diverse dimensioni: **osservabile, deterministico, episodico, statico, discreto, unico agente**

Esistono diverse architetture di agenti di base: **riflesso, riflesso con stato, basato sull'obiettivo, basato sull'utilità**

Risoluzione dei problemi e ricerca

Agenti di problem solving

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
  action ← RECOMMENDATION(seq, state)
  seq ← REMAINDER(seq, state)
  return action
```

Tipi di problemi

Deterministico, completamente osservabile ⇒ problema a stato singolo

L'agente sa esattamente in quale stato si troverà; la soluzione è una sequenza

Formulazione del problema a stato singolo

1. **Stato iniziale:** ad esempio, "ad Arad"
2. **Funzione successore: S(x)** = insieme di coppie azione-stato
   - esempio, **S(Arad) = {hArad → Zerind,Zerindi, . . .}**
3. **Test obiettivo**, può essere
   - **Esplicito**: ad esempio, **x** = “a Bucarest”
   - **Implicito**: ad esempio, **NoDirt(x)**
4. **Costo del percorso** (additivo)**:** ad esempio, somma delle distanze, numero di azioni eseguite, ecc.
   - **c(x, a, y)** è il **costo del passo**, assunto **≥ 0**

Una **soluzione** è una sequenza di azioni che porta dallo stato iniziale a uno stato obiettivo

Non osservabile ⇒ problema conforme

L'agente potrebbe non avere idea di dove si trovi; la soluzione (se presente) è una sequenza

Non deterministico e/o **parzialmente osservabile ⇒ problema di contingenza**

Le percezioni forniscono **nuove** informazioni sullo stato attuale. La soluzione è un **piano contingente** o una **politica** (spesso la ricerca **interlacciata**) e l'esecuzione

Spazio degli stati sconosciuto ⇒ problema di esplorazione (“online”)

Il mondo reale è assurdamente complesso: lo spazio degli stati deve essere **astratto** per la risoluzione dei problemi

Stato (astratto) = insieme di stati reali

Azione (astratta) = combinazione complessa di azioni reali (ad esempio, “Arad → Zerind” rappresenta un insieme complesso di possibili percorsi, deviazioni, punti di ristoro, ecc.)

Per realizzabilità garantita, **qualsiasi** stato reale "in Arad" deve arrivare a qualche stato reale "in Zerind"

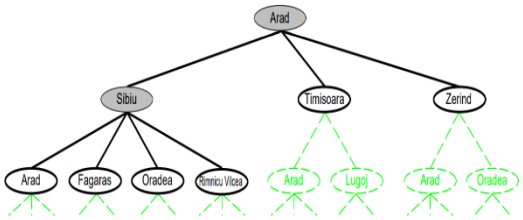
(Astratto) soluzione = insieme di percorsi reali che sono soluzioni nel mondo reale

Ogni azione astratta dovrebbe essere "più facile" del problema originale!

Algoritmi di ricerca ad albero

Idea di base: esplorazione offline e simulata dello spazio degli stati generando successori di stati già esplorati (a.k.a. stati in **espansione**)

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```



Implementazione: stati vs nodi

- Uno **stato** è una rappresentazione di una configurazione fisica: non hanno genitori, figli, profondità o costo del percorso!
- Un **nodo** è una struttura dati che costituisce parte di un albero di ricerca: include **genitore, figli, profondità, costo del percorso g(x)**
- **Expand:** crea nuovi nodi, compilando i vari campi e utilizzando il SuccessorFn del problema per creare gli stati corrispondenti.
- **Tree-Search:** una strategia è definita selezionando l'**ordine di espansione del nodo**

```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
   fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
   loop do
      if fringe is empty then return failure
      node ← REMOVE-FRONT(fringe)
      if GOAL-TEST(problem, STATE(node)) then return node
      fringe ← INSERTALL(EXPAND(node, problem), fringe)

function EXPAND( node, problem) returns a set of nodes
   successors ← the empty set
   for each action, result in SUCCESSOR FN(problem, STATE[node]) do
      s ← a new NODE
      PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result
      PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
      DEPTH[s] ← DEPTH[node] + 1
      add s to successors
   return successors
```

Strategie di ricerca

Una strategia è definita selezionando l'**ordine di espansione del nodo**

Le strategie vengono valutate lungo le seguenti dimensioni:

- **Completezza:** trova sempre una soluzione se ne esiste una?
- **Complessità temporale**: numero di nodi generati/espansi.
- **Complessità spaziale**: numero massimo di nodi in memoria.
- **Ottimale**: trova sempre una soluzione a minor costo?

La complessità del tempo e dello spazio si misura in termini di

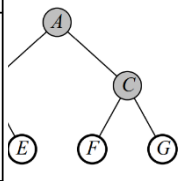
- **b**: fattore di ramificazione massimo dell'albero di ricerca.
- **d**: profondità della soluzione meno costosa.
- **m**: profondità massima dello spazio degli stati (può essere ∞).

Strategie di ricerca non informate

Le strategie **non informate** utilizzano solo le informazioni disponibili nella definizione del problema

Ricerca in ampiezza: espande il nodo non espanso più superficiale

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

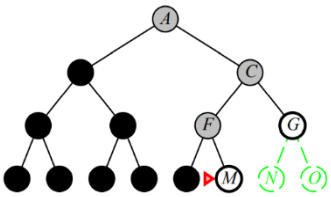


- **Implementazione: fringe** è una coda FIFO, ovvero i nuovi successori vanno alla fine
- **Completo:** Sì (se **b** è finito)
- **Tempo:** $\mathbf{1 + b + b^2 + b^3 + \cdots + b^d + b(b^d - 1) = O(b^{d+1})}$, cioè, esp. in **d**
- **Spazio:** $\boldsymbol{O(b^{d+1})}$ (mantiene ogni nodo in memoria)
  - Grande problema; può facilmente generare nodi a 100 MB/sec, quindi 24 ore = 8640 GB
- **Ottimale**: Sì (se costo = 1 per passaggio); non ottimale in generale

Ricerca a costi uniformi: espande il nodo non espanso a costo minimo

- **Implementazione: fringe** = coda ordinata per costo del percorso, prima il più basso (equivalente a larghezza se il passo ha lo stesso costo per tutti)
- **Completo:** Sì, se il costo del passaggio **≥ €**
- **Tempo:** # di nodi con **g ≤** costo della soluzione ottima**,** $\boldsymbol{O(b^{\lceil C^*/\epsilon \rceil})}$ dove d$\boldsymbol{C^*}$ è il costo della soluzione ottima
- **Spazio:** # di nodi con **g ≤** costo della soluzione ottima, $\boldsymbol{O(b^{\lceil C^*/\epsilon \rceil})}$
- **Ottimale:** Sì, nodi espansi in ordine crescente di **g(n)**

Ricerca in profondità: espande il nodo non espanso più profondo



- **Implementazione: fringe** = coda LIFO, cioè metti i successori in primo piano
- **Completezza:** No, fallisce negli spazi a profondità infinita, spazi con loop
  - Modifica per evitare stati ripetuti lungo il percorso ⇒ completo in spazi finiti
- **Tempo:** $\boldsymbol{O(b^m)}$: terribile se m è molto più grande di d, ma se le soluzioni sono dense, può essere molto più veloce di breadth-first search (ricerca in ampiezza)
- **Spazio: O(bm)**, cioè spazio lineare!
- **Ottimale:** No.

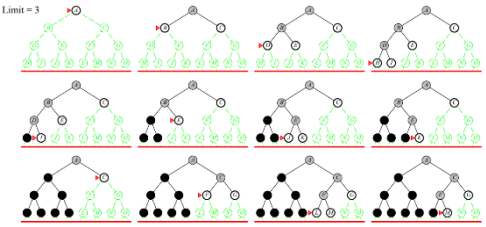
Ricerca con profondità limitata: ricerca in profondità con limite di profondità **l**, cioè i nodi alla profondità **l** non hanno successori

Implementazione ricorsiva

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)
function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST(problem, STATE[node]) then return node
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
    result ← RECURSIVE-DLS(successor, problem, limit)
    if result = cutoff then cutoff-occurred? ← true
    else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
```

Ricerca iterativa di approfondimento

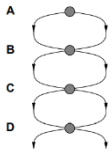
```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

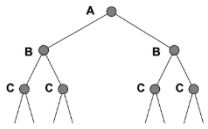


- **Completo:** sì
- **Tempo:** $(d+1)b^0 + db^1 + (d-1)b^2 + \cdots + b^d = O(b^d)$
- **Spazio:** $O(bd)$
- **Ottimale:** Sì, se il costo del passo = 1 (può essere modificato per esplorare l'albero dei costi uniformi)
- Confronto numerico per **b = 10** e **d = 5**, soluzione all'ultima anta destra:
    - IDS (Iterative Deepening Search) funziona meglio perché altri nodi alla profondità **d** non vengono espansi.
        - **N(IDS) = 50 + 400 + 3, 000 + 20, 000 + 100, 000 = 123, 450**
    - BFS (Breadth-First Search) può essere modificato per applicare il test dell'obiettivo quando viene **generato** un nodo.
        - **N(BFS) = 10 + 100 + 1, 000 + 10, 000 + 100, 000 + 999, 990 = 1, 111, 100**

Stati ripetuti: il mancato rilevamento di stati ripetuti può trasformare un problema lineare in uno esponenziale!

```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```





Ricerca nel grafico: può essere esponenzialmente più efficiente della ricerca ad albero

Algoritmi di Ricerca Informata

Best-first search (Migliore prima ricerca)

Idea: utilizzare una **funzione di valutazione** per ogni nodo

- **Stima di "desiderabilità":** espande il nodo non espanso più desiderabile

Implementazione: fringe è una coda ordinata in ordine decrescente di desiderabilità

Ricerca greedy (avida)

Funzione di valutazione **h(n)** (euristica) = stima del costo da **n** all'obiettivo più vicino

Ad esempio, $h_{SLD}(n)$ = distanza in linea retta da n a Bucarest

La ricerca greedy espande il nodo che **sembra** essere il più vicino all'obiettivo

Completo: No, può rimanere bloccato in loop

- Ad esempio: Iasi → Neamt → Iasi → Neamt → …
- Completa in uno spazio finito con controllo dello stato ripetuto

Tempo: $O(b^m)$, ma una buona euristica può dare un notevole miglioramento

Spazio: $O(b^m)$, mantiene tutti i nodi in memoria

Ottimale: No

Ricerca A*

La ricerca A* espande **g + h** più bassi: completa, ottimale ed efficiente (fino ai tie-break, per la ricerca in avanti).

Idea: evitare di espandere percorsi già costosi.

Funzione di valutazione **f(n) = g(n) + h(n)**

- **g(n)** = costo finora per raggiungere **n**
- **h(n)** = costo stimato per l'obiettivo da **n**
- **f(n)** = costo totale stimato del percorso attraverso **n** fino all'obiettivo

La ricerca A* utilizza un'euristica **ammissibile**, cioè **h(n) ≤ h*(n)** dove **h*(n)** è il **vero** costo da **n** (richiede anche **h(n) ≥ 0**, quindi **h(G) = 0** per qualsiasi obiettivo **G**).

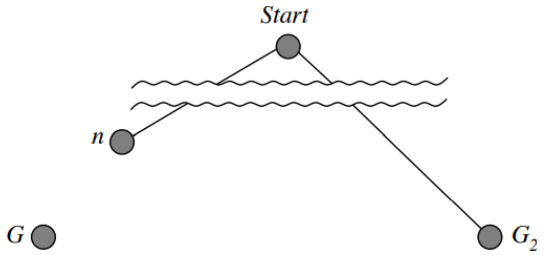
Ad esempio, $\boldsymbol{h_{SLD}(n)}$ non sopravvaluta mai la distanza effettiva

Teorema

La ricerca A* è ottimale (prova standard)

Supponiamo che un obiettivo $\boldsymbol{G_2}$ non ottimale sia stato generato e sia in coda.

Sia **n** un nodo non espanso su un cammino minimo verso un obiettivo ottimo $\boldsymbol{G_1}$.



f($\boldsymbol{G_2}$) = g($\boldsymbol{G_2}$) poiché h($\boldsymbol{G_2}$) = 0

> g($\boldsymbol{G_1}$) poiché $\boldsymbol{G_2}$ è subottimale

≥ f(n) poiché h è ammissibile

Poiché **f($\boldsymbol{G_2}$) > f(n)**, A* non selezionerà mai $\boldsymbol{G_2}$ per l'espansione

Ottimalità di A∗ (più utile)

Lemma: A* espande i nodi in ordine crescente **f** di valore *

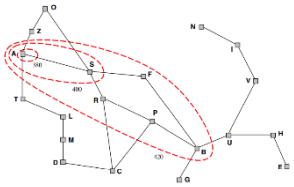
Aggiunge gradualmente "**f**-contours" di nodi (cfr. breadth-first aggiunge strati)

Il contorno **i** ha tutti i nodi con **f =** $\boldsymbol{f_i}$, dove $\boldsymbol{f_i < f_{i+1}}$

Completo: Sì, a meno che non ci siano infiniti nodi con **f ≤ f(G)**

Tempo: Esponenziale in [errore relativo in **h** × lunghezza soln.]

Spazio: Mantiene tutti i nodi in memoria



Ottimale: Sì, non è possibile espandere $\boldsymbol{f_{i+1}}$ fino al termine di $\boldsymbol{f_i}$

A* espande tutti i nodi con **f(n) < C***

A* espande alcuni nodi con **f(n) = C***

A* non espande nodi con **f(n) > C***

Un'euristica è **coerente** se h(n) ≤ c(n, a, n′) + h(n′)

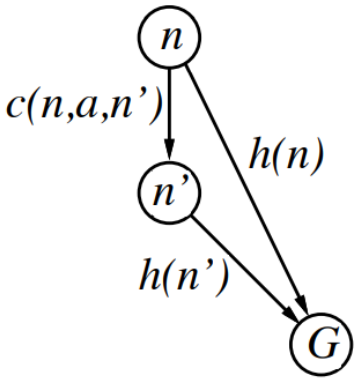
Se **h** è consistente, abbiamo:

f(n′) = g(n′) + h(n′)

= g(n) + c(n, a, n′) + h(n′)

≥ g(n) + h(n)

= f(n)



Cioè, **f(n)** non è decrescente lungo qualsiasi percorso.

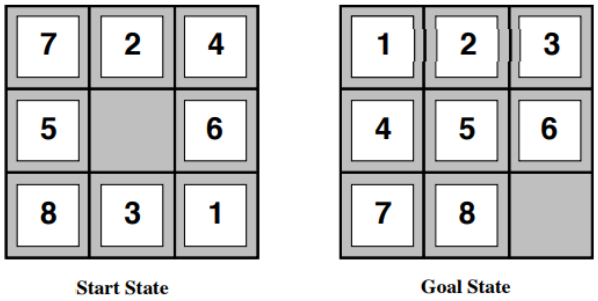
Euristiche ammissibili

Le funzioni euristiche stimano i costi dei cammini minimi e possono ridurre drasticamente i costi di ricerca.

L'euristica ammissibile può essere derivata dalla soluzione esatta di problemi rilassati.

Ad esempio, per il puzzle 8:

- $\boldsymbol{h_1(n)}$ = numero di tessere fuori posto
- $\boldsymbol{h_2(n)}$ = distanza totale di **Manhattan** (ovvero, numero di quadrati dalla posizione desiderata di ciascuna tessera)



$\boldsymbol{h_1(S)}$ = 6

$\boldsymbol{h_2(S)}$ = 4+0+3+3+1+0+2+1 = 14

Dominanza

Se $\boldsymbol{h_2(n) \geq h_1(n)}$ per tutti gli **n** (entrambi ammissibili) allora $\boldsymbol{h_2}$ **domina** $\boldsymbol{h_1}$ ed è migliore per la ricerca

Costi di ricerca tipici:

- d = 14 IDS = 3.473.941 nodi
  - A*($\boldsymbol{h_1}$) = 539 nodi
  - A*($\boldsymbol{h_2}$) = 113 nodi
- d = 24 IDS ≈ 54.000.000.000 di nodi
  - A*($\boldsymbol{h_1}$) = 39.135 nodi
  - A*($\boldsymbol{h_2}$) = 1.641 nodi

Data qualsiasi euristica ammissibile $\boldsymbol{h_a}$, $\boldsymbol{h_b}$, **h(n) = max(**$\boldsymbol{h_a}$**(n),** $\boldsymbol{h_b}$**(n))** è anche ammissibile e domina $\boldsymbol{h_a}$, $\boldsymbol{h_b}$

Problemi rilassati

L'euristica ammissibile può essere derivata dal costo **esatto** della soluzione di una versione **rilassata** del problema

Se le regole del puzzle degli otto sono allentate in modo che una tessera possa muoversi **ovunque**, allora $\boldsymbol{h_1(n)}$ fornisce la soluzione più breve

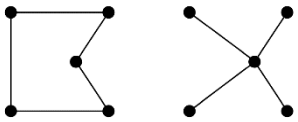
Se le regole sono allentate in modo che una tessera possa spostarsi in **qualsiasi casella adiacente**, allora $\boldsymbol{h_2(n)}$ fornisce la soluzione più breve

Punto chiave: il costo di soluzione ottima di un problema rilassato non è maggiore del costo di soluzione ottima del problema reale

Esempio ben noto: **problema del commesso viaggiatore** (TSP)

Trova il tour più breve visitando tutte le città esattamente una volta

L'albero di copertura minimo può essere calcolato in $O(n^2)$ ed è un limite inferiore nel tour (aperto) più breve.



Algoritmi di Ricerca Locale

Algoritmi di miglioramento iterativo

In molti problemi di ottimizzazione, il **percorso** è irrilevante e lo stato obiettivo stesso è la soluzione

Spazio degli stati = insieme di configurazioni "complete"; trova la configurazione **ottimale** (ad esempio TSP) o trova i vincoli che soddisfano la configurazione (ad esempio l'orario).

In tali casi, può utilizzare algoritmi di **miglioramento iterativo** e mantenere un unico stato "attuale", provando a migliorarlo.

Spazio costante, adatto alla ricerca online e offline.

Esempio: n-regine

Metti **n** regine su una tavola **n × n** senza due regine sulla stessa riga, colonna o diagonale e sposta una regina per ridurre il numero di conflitti.

Risolve quasi sempre i problemi delle **n**-regine quasi istantaneamente per **n** molto grandi, ad esempio **n = 1 milione**.

Arrampicata (o salita/discesa in pendenza)

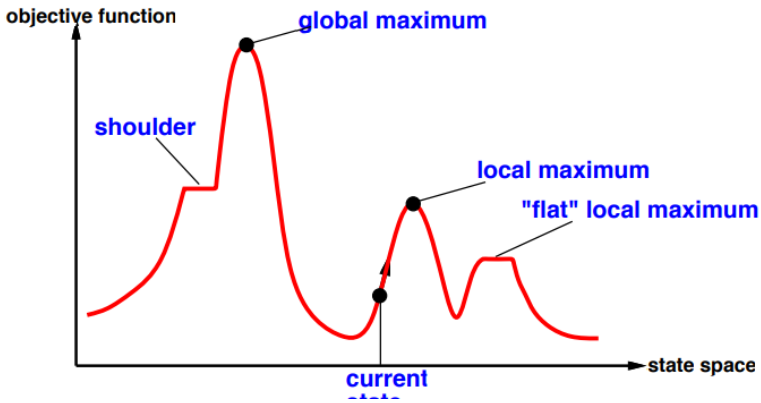
```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node
   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
   end
```

Arrampicata continua

Utile per considerare il **panorama dello spazio degli stati**.

L'arrampicata in salita con ripartenza casuale supera i massimi locali (banalmente completa)

Movimenti laterali casuali scappano dall'anello delle spalle su massimi piatti



Ricottura simulata

Idea: sfuggire ai massimi locali consentendo alcune mosse "cattive" **ma riducendone gradualmente le dimensioni e la frequenza**.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps
  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(Δ E/T)
```

A "temperatura" T fissa, la probabilità di occupazione dello stato raggiunge la distribuzione di Boltzman: $\boldsymbol{p(x) = \alpha e^{\frac{E(x)}{kT}}}$

T diminuisce abbastanza lentamente ⇒ raggiunge sempre il miglior stato **x*** perché e $\frac{e^{\frac{E(x^*)}{kT}}}{e^{\frac{E(x)}{kT}}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ per **T** piccoli.

Ricerca del raggio locale

Idea: mantieni **k** stati invece di 1; scegli il top **k** di tutti i loro successori

Non è lo stesso di **k** ricerche eseguite in parallelo!

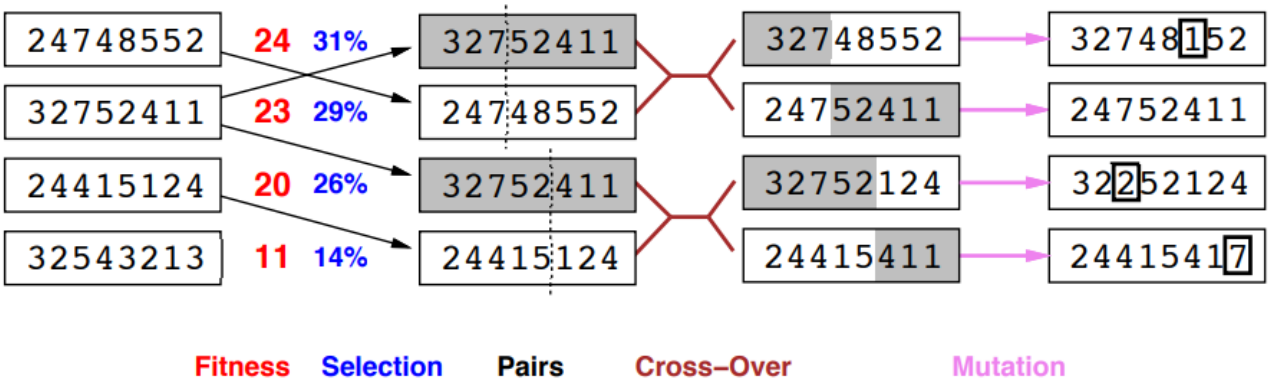
Le ricerche che trovano buoni stati reclutano altre ricerche per unirsi a loro.

Problema: abbastanza spesso, tutti i k stati finiscono sulla stessa collina locale

Idea: scegli **k** successori a caso, sbilanciati verso quelli buoni.

Osserva la stretta analogia con la selezione naturale!

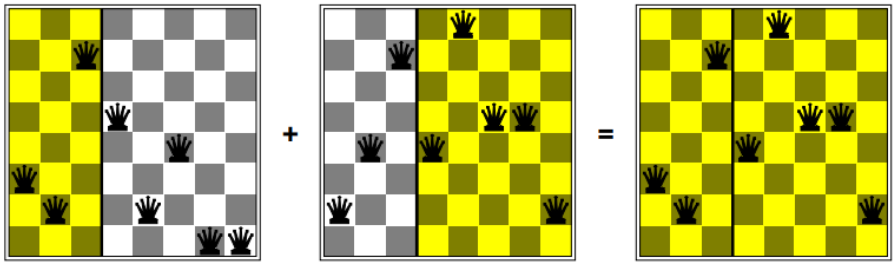
Algoritmi genetici = ricerca stocastica locale del fascio + genera successori da **coppie** di stati.



Algoritmi genetici continui

Richiedono stati codificati come stringhe (usano **programmi**)

Il crossover aiuta **se le sottostringhe sono componenti significativi**.



Algoritmi Genetici ≠ evoluzione: ad esempio, i geni reali codificano per macchinari di replicazione!

Spazi di stato continui

Supponiamo di voler collocare tre aeroporti in Romania:

- Spazio degli stati 6-D definito da $(x_1, y_2), (x_2, y_2), (x_3, y_3)$
- Funzione obiettivo $f(x_1, y_2, x_2, y_2, x_3, y_3)$ = somma delle distanze al quadrato da ogni città all'aeroporto più vicino

I metodi di **discretizzazione** trasformano lo spazio continuo in spazio discreto, ad esempio, il **gradiente empirico** considera il cambiamento di **±δ** in ciascuna coordinata

Calcolo dei metodi del **gradiente** $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3}\right)$ aumentare/ridurre **f**, ad esempio **di x ← x + α∇f(x)**

A volte può risolvere esattamente per **∇f(x) = 0** (ad esempio, con una città).

Newton-Raphson (1664, 1690) itera **x ← x −** $H_f^{-1}(x)\nabla f(x)$ per risolvere **∇f(x) = 0**, dove $H_{ij} = \partial^2 f / \partial x_i \partial x_j$

Gradiente

Per le funzioni w = f (x, y, z) abbiamo il gradiente: $grad\ w = \nabla w = \langle \frac{\partial w}{\partial x}, \frac{\partial w}{\partial y}, \frac{\partial w}{\partial z} \rangle$

Cioè, il gradiente assume una funzione scalare di tre variabili e produce un vettore tridimensionale.

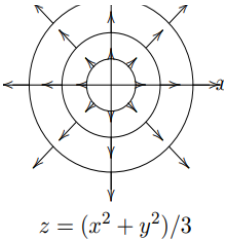
Il gradiente di una funzione scalare $f(x_1, x_2, x_3)$ definita nel sistema di riferimento cartesiano ortonormale è un campo vettoriale che associa ad ogni punto (x_0, y_0, z_0) il vettore che ha per componenti le derivate parziali prime di f calcolare nel punto.

La definizione può essere estesa a funzioni definite in uno spazio euclideo di dimensione arbitraria.

Il prodotto scalare del gradiente di f per un vettore unitario **v** è uguale alla derivata direzionale di f lungo **v** che intuitivamente ci dice quanto la funzione f cresce lungo il vettore **v** (nel punto x).

$$(\nabla f(x)) \cdot \mathbf{v} = D_{\mathbf{v}} f(x).$$

Esempio: Calcola il gradiente di w = $\frac{x^2+y^2}{3}$ e mostra che il gradiente in $(x_0, y_0) = (1, 2)$ è perpendicolare alla curva di livello attraverso quel punto.



$z = (x^2 + y^2)/3$

Risposta: il gradiente è facilmente calcolabile $\nabla w = \langle 2x/3, 2y/3 \rangle = \frac{2}{3}\langle x, y \rangle$.

In (1, 2) otteniamo $\nabla w(1, 2) = \frac{2}{3}\langle 1, 2 \rangle$. La curva di livello attraverso (1, 2) è $\frac{(x^2+y^2)}{3} = \frac{5}{3}$, che è identico a $x^2 + y^2 = 5$. Cioè è un cerchio di raggio $\sqrt{5}$ centrato nell'origine. Poiché il gradiente in (1,2) è un multiplo di $\langle 1,2 \rangle$, punta radialmente verso l'esterno e quindi è perpendicolare al cerchio.

Dimensione 3

Dimensioni maggiori

Analogamente, per w = f(x, y, z) otteniamo superfici piane f(x, y, z) = c. Il gradiente è perpendicolare alle superfici piane.

Esempio: Trova il piano tangente alla superficie $x^2 + 2y^2 + 3z^2 = 6$ nel punto P = (1, 1, 1).

Risposta: Introduci una nuova variabile $w = x^2 + 2y^2 + 3z^2$.

La nostra superficie è la superficie piana w = 6. Dire che il gradiente è perpendicolare alla superficie significa esattamente la stessa cosa che dire che è normale al piano tangente.

Calcolando $\nabla \mathrm{w} = \langle 2\mathrm{x}, 4\mathrm{y}, 6\mathrm{z} \rangle$ -> $\nabla \mathrm{w}|\mathrm{p} = \langle 2, 4, 6 \rangle$.

Usando la forma normale al punto otteniamo l'equazione del piano tangente è $2(x-1) + 4(y-1) + 6(z-1) = 0$ *o* $2x + 4y + 6z = 12$

Matrice Hessiana

La matrice Hessiana di una funzione f: R^n -> R è la matrice H(f) definita come segue (assumendo l'esistenza delle derivate seconde)

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

Ottimizzazione numerica

I metodi di ottimizzazione consentono di trovare i punti estremanti (massimi, minimi) di una funzione obiettivo f: $R^n \rightarrow$ R.

Poiché max f(x) = − min (−f (x)), è sufficiente concentrarsi sui metodi per la ricerca del $\min_{x \in \Omega \subseteq R^n} f(x)$ dove Ω è l'insieme definito dai vincoli, se presenti.

Per semplicità, in seguito supporremo l'esistenza e l'unicità del punto di x* (locale e minimo globale), e tratteremo l'ottimizzazione in assenza di vincoli.

Metodi numerici per l'ottimizzazione

Funzioni uni-variate

Ricerca sezione aurea: funzione f: R → R è **unimodale** se esiste m ∈ R tale che f è decrescente per x ≤ m e crescente per x ≥ m. m è quindi il punto di minimo (globale).

Il nome del metodo è dovuto alla proporzione aurea esistente tra le distanze dei punti scelti dagli estremi dell'intervallo: $\frac{x_2 - x_1}{x_1 - a} = \frac{x_1 - a}{b - x_1}$

Algoritmo piuttosto semplice e intuitivo per la ricerca del minimo di una funzione all'interno di un intervallo [a, b] con a < b e x* ∈ [a, b].

Ecco come funziona il **metodo ricerca sezione aurea**:

1. Si parte considerando l'intervallo $[a_0, b_0]$:= [a, b] e si fissa una tolleranza ϵ
2. Si restringe iterativamente l'intervallo a $[a_{k+1}, b_{k+1}] \subset [a_k, b_k]$ in modo che il punto di minimo x* continui a cadere al suo interno. Si determinano $x_1, x_2 \in [a_k, b_k]$ tali che $x_1 < x_2$.
3. Ci si arresta quando $|b_k - a_k| < \epsilon$, e si accetta $\frac{b_k - a_k}{2}$ come una buona approssimazione del punto di minimo x*

Poiché la funzione è unimodale e abbiamo supposto esserci un unico punto di minimo x* ∈ [a, b], se succede f (x_1) > f (x_2) allora significa x* ∈ $[x_1, b_k]$, altrimenti se f(x_1) < f(x_2) allora x* ∈ $[a_k, x_2]$.

Selezionato l'intervallo corretto, si prosegue in modo iterativo.

Pro: non richiede la conoscenza delle derivate

Contro: convergenza lenta

Esempio

$f(x) = |x - 3| + \sin x - |x - 2|$
Determinare minimi/massimi e osservare il comportamento del metodo golden section search al variare dell'intervallo di ricerca.

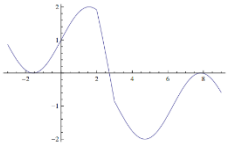


Figure: $f(x) = |x - 3| + \sin x - |x - 2|$

La funzione dell'esempio ha due minimi e due massimi nell'intervallo [−3, 9]. A seconda dell'intervallo di ricerca scelto, il metodo converge a uno dei minimi:

- [−3, 9]: due massimi, due minimi; converge a uno dei minimi
- [0, 9]: un massimo, un minimo; convergere al minimo
- [3, 6]: un minimo; convergere a quello

Metodi di discesa

Metodi iterativi che, a partire da un iterato iniziale $x_0 \in R_n$, generano una successione di punti $\{x_k\}$ k ∈ N definiti dall'iterazione $x_{k+1} = x_k + \alpha_k p_k$ dove il vettore p_k è una direzione di ricerca e lo scalare α_k è un parametro positivo chiamato lunghezza del passo (step length) che indica la distanza di cui ci si muove lungo la direzione p_k.

In un metodo di discesa, il vettore p_k e il parametro α_k sono scelti in modo da garantire la decrescita della funzione obiettivo f a ogni iterazione: f(x_{k+1}) < f(x_k) ∀k ≥ 0

In particolare, come vettore p_k si prende una direzione di discesa, cioè tale che la retta x = x_k + $\alpha_k p_k$ forma un angolo ottuso con il vettore gradiente ∇f (x_k). In questo modo è possibile garantire la decrescita di f, purché α_k sia sufficientemente piccolo.

Scelta della lunghezza del passo: esistono diverse tecniche per la scelta di α_k in modo tale da garantire la convergenza del metodo verso punti stazionari di f. Lunghezza del passo troppo piccola (può tradursi in una convergenza lenta) o troppo grande (rischio di "superare" il punto di minimo).

Metodo del gradiente discendente: $x_{k+1} = x_k - \delta f'(x_k)$

Nome dovuto al fatto che si sceglie come direzione di ricerca quella opposta al gradiente (derivata prima, nel caso uni-variato).

Direzione di ricerca (gradiente): p_k = −f ′(xk)

Lunghezza del passo: α_k = δ costante

Richiede f ∈ C^1 (derivabile con continuità) e conoscenza dell'espressione analitica della derivata prima.

Costo computazionale: 1 valutazione della derivata prima per iterazione.

Pro: converge a punti di minimo e non genericamente a punti stazionari; non richiede il calcolo della derivata seconda (matrice Hessiana nel caso multivariato)

Contro: alto numero di iterazioni e sensibilità rispetto all'iterato iniziale (se x_0 è preso troppo lontano dal minimo x* e δ è scelta piccola, il metodo può non arrivare alla soluzione in tempi ragionevoli)

Metodo di Newton-Raphson: $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$

- Direzione di ricerca (Newton): p_k = − f′(x_k)/f″(x_k)
- Lunghezza del passo: α_k = 1
- **Costo computazionale:** a ogni iterazione, valutazione di derivata prima e seconda.
- **Pro:** convergenza veloce (quadratica).
- **Contro:** convergenza locale e alto costo per iterazione.

Ha la forma del metodo di Newton per la ricerca degli zeri di una funzione, applicazione a f', in quanto determinato il punto di minimo della f funzione equivale a determinare la radice della derivata prima f′.

Il metodo Newton-Raphson è preferito rispetto al gradiente discendente, per la sua velocità, nonostante richieda f ∈ C^2, f″ ≠ 0, conoscenza dell'espressione analitica delle derivate prima e seconda, e converga indistintamente a minimi e massimi.

Se f ha un'espressione semplice, la function deriv consente di calcolarne simbolicamente le derivate prima e seconda.

Esistono varianti che rendono il metodo a convergenza globale e che abbassano il costo computazionale evitando di risolvere con metodi diretti il sistema per la determinazione di p_k.

Funzioni multivariate

Dall'espansione di Taylor nel caso multidimensionale, il **metodo di Newton-Raphson multivariato** assume la forma: $x_{k+1} = x_k - H_k(x_k)^{-1}\nabla f(x_k)$ dove ora $x_k \in R^n$ e H_f è la matrice Hessiana (derivate seconde) di f.

- Direzione di ricerca (Newton): p_k = − $H_f(x_k)^{-1}\nabla f(x_k)$

- Lunghezza del passo: α_k = 1
- La direzione di Newton è di discesa <-> H_f è definita positiva.

Il metodo di Newton-Raphson multivariato richiede f ∈ C^2 e conoscenza dell'espressione analitica del gradiente e della matrice Hessiana.

Se f ha un'espressione semplice, la function deriv consente di calcolarne simbolicamente il gradiente e la matrice Hessiana.

Costo computazionale: a ogni iterazione, valutazione di gradiente, Hessiana e risoluzione del sistema n × n

$$\mathrm{H}_f(\mathrm{x}_k)\mathrm{p}_k = -\nabla \mathrm{f}\,(\mathrm{x}_k)$$

per la determinazione di p_k (costo O(n^3)).

Gioco in corso

Giochi contro problemi di ricerca

Avversario "imprevedibile" ⇒ la soluzione è una **strategia** che specifica una mossa per ogni possibile risposta dell'avversario.

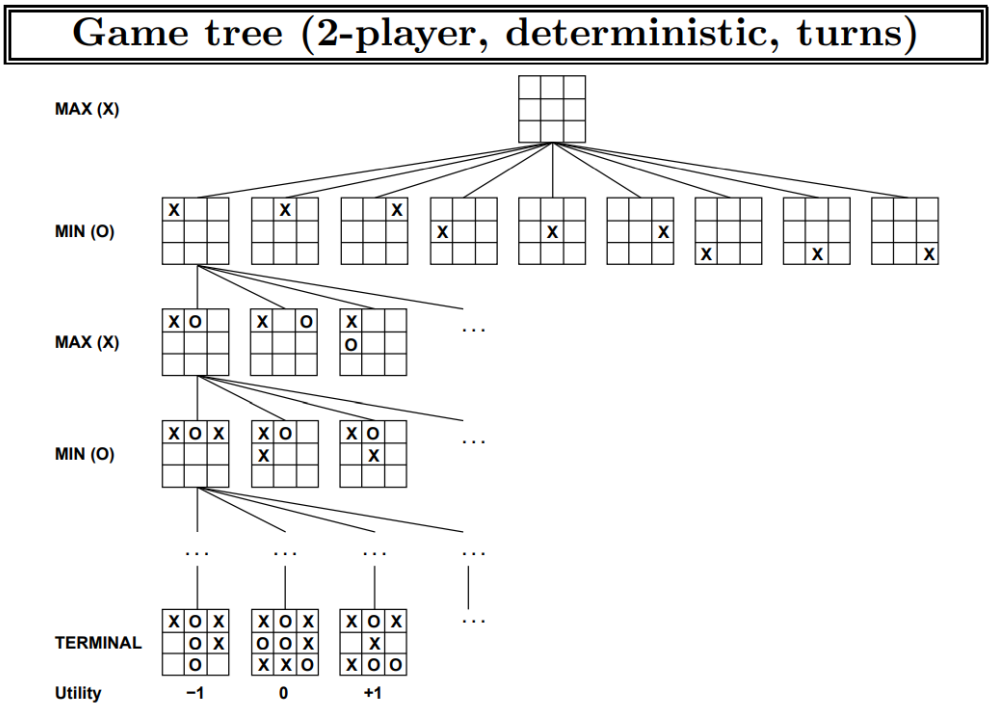
I limiti di tempo ⇒ improbabile da trovare l'obiettivo, devono approssimarsi.

Piano d'attacco:

- Il computer considera le possibili linee di gioco (Babbage, 1846)
- Algoritmo per il **gioco perfetto** (Zermelo, 1912; Von Neumann, 1944)
- **Orizzonte finito, valutazione approssimata** (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- Primo programma di scacchi (Turing, 1951)
- Apprendimento automatico per migliorare l'accuratezza della valutazione (Samuel, 1952-57)
- Potatura per consentire una ricerca più approfondita (McCarthy, 1956)

Tipi di giochi

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war



Minimax

Gioco perfetto per giochi deterministici e con informazioni perfette.

Idea: scegli il passaggio alla posizione con il valore minimax più alto = miglior payoff ottenibile contro il miglior gioco

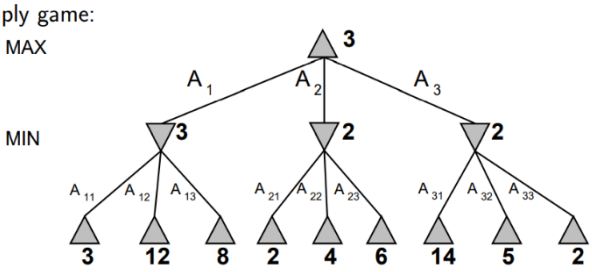
Minimax algorithm

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
  return v
```

E.g., 2-ply game:



Proprietà di minimax

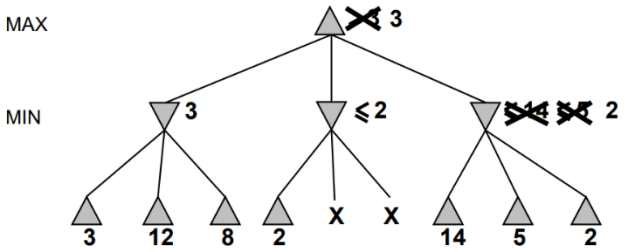
- **Completo:** Sì, se l'albero è finito (gli scacchi hanno regole specifiche).
- **Ottimale:** Sì, contro un avversario ottimale.
- **Complessità temporale:** $\boldsymbol{O(b^m)}$

- **Complessità spaziale: O(bm)** (esplorazione in profondità)

Per gli scacchi, **b ≈ 35, m ≈ 100** per giochi "ragionevoli" ⇒ soluzione esatta completamente irrealizzabile.

Ma abbiamo bisogno di esplorare ogni percorso?

α–β pruning example

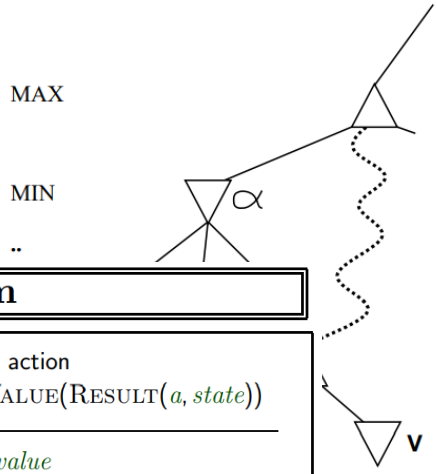


Perché si chiama α–β?

α è il miglior valore (al MAX) trovato finora al di fuori del percorso corrente.

Se **V** è peggiore di **α**, MAX lo eviterà -> pota quel ramo.

Definisci **β** in modo simile per MIN.



The α–β algorithm

```
function ALPHA-BETA-DECISION(state) returns an action
  return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
          α, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  same as MAX-VALUE but with roles of α, β reversed
```

Proprietà di α–β

- La potatura **non influisce** sul risultato finale.
- Un buon ordinamento delle mosse migliora l'efficacia della potatura.
- Con "ordinamento perfetto", complessità temporale = $O(b^{\frac{m}{2}})$ ⇒ **raddoppia** la profondità risolvibile.
- Un semplice esempio del valore del ragionamento su quali calcoli sono rilevanti (una forma di **meta-reasoning**).
- Sfortunatamente, 35^{50} è ancora impossibile.

Limiti delle risorse

Approccio standard:

- Usa Cutoff-Test invece di Terminal-Test: limite di profondità (forse aggiunge la **ricerca di quiescenza**)
- Usa Eval invece di Utility: **funzione di valutazione** che stima la desiderabilità della posizione

Supponiamo di avere **cento** secondi, esplora $\mathbf{10^4}$ nodi/secondo

⇒ $\mathbf{10^6}$ nodi per mossa ≈ $\mathbf{35^{\frac{8}{2}}}$.

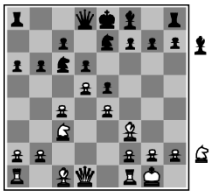
⇒ α–β raggiunge la profondità 8 ⇒ programma di scacchi abbastanza buono.

Funzioni di valutazione

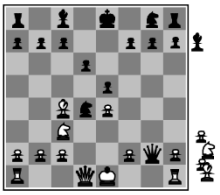
Per gli scacchi, somma di caratteristiche tipicamente **lineare** ponderata

Eval(s) = $\boldsymbol{w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)}$

Esempio, $\boldsymbol{w_1}$**= 9** con $\boldsymbol{f_1}$**(s)** = (numero di regine bianche) – (numero di regine nere), ecc

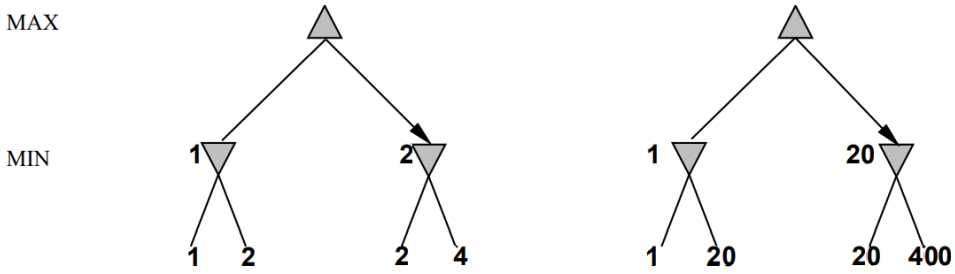


Black to move
White slightly better



White to move
Black winning

I valori esatti non contano: il comportamento è preservato sotto qualsiasi trasformazione **monotona** di Eval.



Conta solo l'ordine: il payoff nei giochi deterministici agisce come una funzione di **utilità ordinale.**

Giochi deterministici in pratica

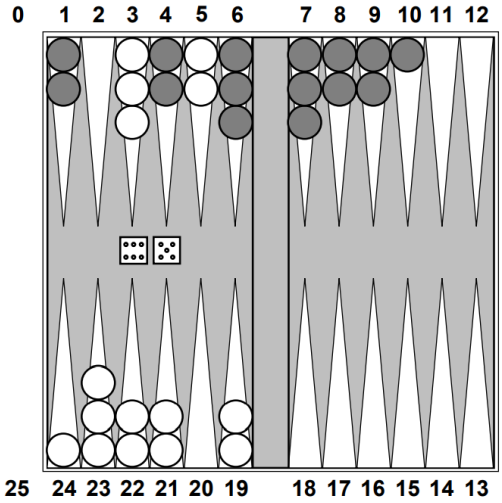
Dama: Chinook ha concluso i 40 anni di regno della campionessa mondiale umana Marion Tinsley nel 1994. Ha utilizzato un database di fine partita che definisce il gioco perfetto per tutte le posizioni che coinvolgono 8 o meno pezzi sulla scacchiera, per un totale di 443.748.401.247 posizioni.

Scacchi: Deep Blue ha sconfitto il campione del mondo umano Gary Kasparov in una partita di sei partite nel 1997. Deep Blue cerca 200 milioni di posizioni al secondo, utilizza una valutazione molto sofisticata e metodi non divulgati per estendere alcune linee di ricerca fino a 40 ply.

Otello: i campioni umani si rifiutano di competere con i computer, che sono troppo bravi.

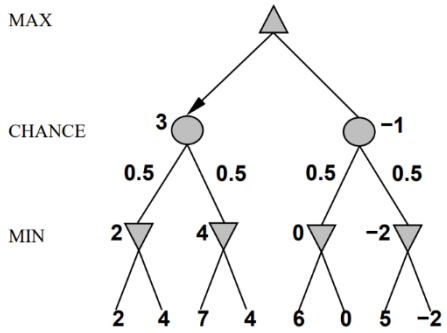
Go: i campioni umani si rifiutano di competere contro i computer, che sono troppo cattivi. In go, b > 300, quindi la maggior parte dei programmi utilizza basi di conoscenza dei modelli per suggerire mosse plausibili.

Giochi non deterministici: backgammon



Giochi non deterministici in generale

Nei giochi non deterministici, probabilità introdotta da dadi, mescolamento di carte. Esempio semplificato con lancio di monete:



Algoritmo per giochi non deterministici

Expectiminimax offre un gioco perfetto: proprio come Minimax, tranne per il fatto che dobbiamo gestire anche i nodi casuali:

```
...
if state is a MAX node then
        return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
        return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
        return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
...
```

Giochi non deterministici in pratica

Aumento dei tiri di dado b: 21 possibili lanci con 2 dadi.

Backgammon ≈ 20 mosse legali (possono essere 6.000 con 1-1 tiro).

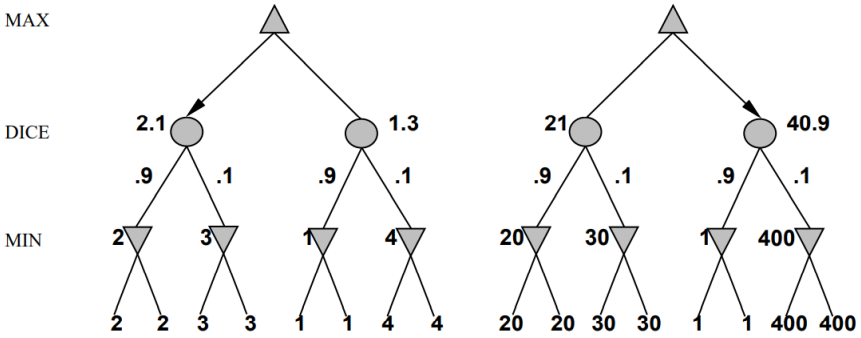
Profondità 4 = $20 \times (21 \times 20)^3 \approx 1{,}2 \times 10^9$: all'aumentare della profondità, la probabilità di raggiungere un dato nodo si riduce (il valore del look-ahead diminuisce)

La potatura α–β è molto meno efficace.

TDGammon utilizza la ricerca in profondità-2 + un'ottima Eval ≈ livello di campione del mondo

I valori esatti contano: il comportamento è preservato solo dalla trasformazione lineare positiva di Eval (Eval dovrebbe essere proporzionale al payof atteso)

Giochi di informazione imperfetta

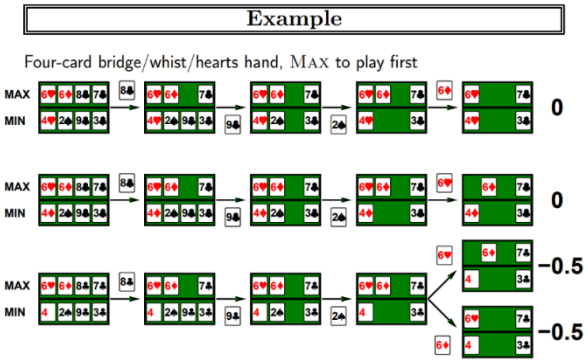


Ad esempio, giochi di carte, in cui le carte iniziali dell'avversario sono sconosciute. Tipicamente possiamo calcolare una probabilità per ogni possibile affare. Sembra proprio come avere un grande lancio di dadi all'inizio del gioco.

Idea: calcola il valore minimax di ogni azione in ogni affare; quindi, scegli l'azione con il valore atteso più alto su tutte le offerte*****.

Caso speciale: se un'azione è ottimale per tutte le operazioni, è ottimale*****.

GIB, l'attuale miglior programma di bridge, si avvicina a questa idea di **generare** cento offerte coerenti con le informazioni sulle offerte e **scegliere** l'azione che in media vince più prese



Esempio di buon senso: la strada A conduce a un piccolo mucchio di monete d'oro, mentre la strada B porta a un bivio

- prendi il bivio a sinistra e troverai un mucchio di gioielli;
- prendi il bivio a destra e sarai investito da un autobus.

Oppure il contrario:

- prendete il bivio a sinistra e sarete travolti da un autobus;
- prendi il bivio a destra e troverai un mucchio di gioielli.

Analisi corretta

L'intuizione che il valore di un'azione è la media dei suoi valori in tutti gli stati effettivi è **SBAGLIATA**

Osservabilità parziale: il valore di un'azione dipende dallo **stato di informazione** o dallo **stato di convinzione** in cui si trova l'agente. Può generare e cercare un albero di stati di informazione e porta a comportamenti razionali come agire per ottenere informazioni, segnalare al proprio partner o agire in modo casuale per ridurre al minimo la divulgazione di informazioni.

Riepilogo: punti importanti sull'AI

- La perfezione è irraggiungibile, deve approssimarsi
- Buona idea per pensare a cosa pensare
- L'incertezza vincola l'assegnazione dei valori agli stati
- Le decisioni ottimali dipendono dallo stato dell'informazione, non dallo stato reale

Vincoli (Logica) Linguaggi di programmazione

Esempio 1: un problema combinatorio

Problema: disponi tre 1, tre 2, … tre 9 in sequenza in modo che per tutti i ∈ [1,9] ci siano esattamente i numeri tra le occorrenze successive di i

Una soluzione: 1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7

Un programma Prolog che risolve il problema precedente

```
sequence([_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_]).
% Sequence(X) -> X is a lit of 27 variables


question(S):-
    sequence(S),
    sublist([9,_,_,_,_,_,_,_,_,9,_,_,_,_,_,_,_,_,9],S),
    sublist([8,_,_,_,_,_,_,_,8,_,_,_,_,_,_,_,8] ,S),
    sublist([7,_,_,_,_,_,_,7,_,_,_,_,_,_,7] ,S),
    sublist([6,_,_,_,_,_,6,_,_,_,_,_,6] ,S),
    sublist([5,_,_,_,_,5,_,_,_,_,5] ,S),
    sublist([4,_,_,_,4,_,_,_,4] ,S),
    sublist([3,_,_,3,_,_,3] ,S),
    sublist([2,_,2,_,2] ,S),
    sublist([1,_,1,_,1] ,S).
% S is a solution
```

Esempio 2: un problema di ottimizzazione

Problema delle torte: un problema di pianificazione della produzione semplificato. Dobbiamo preparare delle torte per una festa nella nostra scuola locale e abbiamo 4kg di farina auto-lievitante, 6 banane, 2kg di zucchero, 500g di burro e 500g di cacao. Sappiamo fare due tipi di torte:

1. Una torta alla banana che richiede 250 g di farina auto-lievitante, 2 banane schiacciate, 75 g di zucchero e 100 g di burro (può essere venduta a $4,00).

2. Una torta al cioccolato che richiede 200 g di farina auto-lievitante, 75 g di cacao, 150 g di zucchero e 150 g di burro (può essere venduta a $4,50).

Quante torte di ogni tipo dovremmo cuocere per la festa al fine di massimizzare il profitto?

Un programma MiniZinc che risolve il problema delle torte

```
CAKES ≡
  % Baking cakes for the school fete

  var 0..100: b; % no. of banana cakes
  var 0..100: c; % no. of chocolate cakes

  % flour
  constraint 250*b + 200*c <= 4000;
  % bananas
  constraint 2*b  <= 6;
  % sugar
  constraint 75*b + 150*c <= 2000;
  % butter
  constraint 100*b + 150*c <= 500;
  % cocoa
  constraint 75*c <= 500;

  % maximize our profit
  solve maximize 400*b + 450*c;

  output ["no. of banana cakes = \(b)\n",
          "no. of chocolate cakes = \(c)\n"];
```

Programmazione logica

Bob Kowalski: **"Algoritmo = Logica + Controllo"**

Nella programmazione tradizionale: il programmatore si occupa di entrambi gli aspetti

Nella programmazione dichiarativa: il programmatore si occupa solo di Logica e l'interprete del linguaggio si occupa di Controllo

Programmazione dichiarativa: tre paradigmi

1. Programmazione logica (Prolog, …)
2. Programmazione funzionale (ML, Haskel, OCAML, …)
3. **Programmazione con vincoli** (MinZinc, CLP, ILOG, …): rappresenta uno degli approcci più vicini che l'informatica abbia mai fatto al Santo Graal della programmazione (l'utente afferma il problema, il computer lo risolve. E.C. Freuder)

Programmazione dichiarativa

Il compito del programmatore è specificare il problema da risolvere: dobbiamo "dichiarare" **cosa** vogliamo ottenere e non **come** lo otteniamo.

Ragionamento vincolante

- Esempio: Lucchetto a combinazione (0123456789) maggiore o uguale 5 e numero primo.
- Rappresentazione dichiarativa del problema per variabili e vincoli: $x \in \{0,1, \dots ,9\} \wedge x \geq 5 \wedge prime(x)$
- La propagazione e la semplificazione dei vincoli riducono lo spazio di ricerca: $x \in \{0,1, \dots ,9\} \wedge x \geq 5 \wedge x \in \{5,6,7,8,9\}$

Programmazione dei vincoli: software robusto, flessibile e mantenibile più veloce.

- Modellazione dichiarativa per vincoli:
  - Descrizione di proprietà e relazioni tra oggetti parzialmente conosciuti.
  - Corretta gestione di informazioni precise e imprecise, finite e infinite, parziali e complete.
- Ragionamento automatico dei vincoli:
  - **Propagazione** degli effetti di nuove informazioni (come vincoli).
  - **La semplificazione** rende esplicite le informazioni implicite.
- Risolvere problemi combinatori in modo efficiente:
  - **Facile combinazione** di risoluzione dei vincoli con ricerca e ottimizzazione.

Terminologia: il linguaggio è logica del primo ordine con uguaglianza.

- Vincolo: congiunzione di vincoli atomici (predicati). Ad esempio, 4X + 3Y = 10 ∧ 2X – Y = 0
- Problema di vincolo (interrogazione): un vincolo iniziale dato
- Soluzione dei vincoli (risposta): una valutazione per le variabili in un dato problema di vincoli che soddisfa tutti i vincoli del problema. Ad esempio, X = 1 ∧ Y = 2

In generale, una forma normale/risolta, ad esempio, del problema 4X + 3Y + Z = 10 ∧ 2X – Y = 0 si semplifica in Y + Z = 10 ∧ 2X – Y = 0

Ragionamento dei vincoli e programmazione dei vincoli

Un quadro generico per

- Modellazione con informazioni parziali o con infinite informazioni
- Ragionamento con nuove informazioni
- Risolvere problemi combinatori

Due classi principali di problemi

1. **Problemi di soddisfazione dei vincoli (CSP):** definito da un insieme finito di variabili $\{X_1, \dots, X_n\}$, un insieme di valori (Dominio) per ogni variabile D(X_1),... D(X_n) e un insieme di vincoli $\{C_1, \dots, C_m\}$
   - Una soluzione a un CSP è un'assegnazione completa a tutte le variabili che soddisfa i vincoli.
   - Per esempio: $X \in \{1,2\} \wedge Y \in \{1,2\} \wedge Z \in \{1,2\} \wedge X = Y \wedge X \neq Z \wedge Y > Z$

2. **Problemi di ottimizzazione dei vincoli (COP):** un COP è un CSP definito sulle variabili $\{X_1, ..., X_n\}$ e sui domini D(X_1),... D(X_n), insieme ad una funzione obiettivo f : D(X_1),... D(X_n) -> D. Una soluzione a un COP è una soluzione del CSP che ottimizza il valore di f.

Due famiglie principali di linguaggi di vincolo

Programmazione logica dei vincoli (CLP): aggiunge vincoli e relativi solutori alla programmazione logica

- Due paradigmi dichiarativi insieme: risoluzione dei vincoli e programmazione logica
- Più espressivo, flessibile e in generale **più efficiente** dei programmi logici
- Per esempio $X - Y = 3 \wedge X + Y = 7$ porta a $X = 5 \wedge Y = 2$
- Per esempio, X < Y ∧ Y < X fallisce senza la necessità di conoscere i valori
- Le moderne implementazioni di Prolog sono CLP

Linguaggi imperativi con vincoli: integra i vincoli e i relativi solutori ai linguaggi imperativi

- Distribuzioni e applicazioni commerciali
- Un esempio (gratuito): MinZinc

Risoluzione dei vincoli

Il risolutore è una parte essenziale dei linguaggi dei vincoli: risolve i vincoli adattando e combinando algoritmi efficienti esistenti da matematica (ricerche operative, teoria dei grafi, algebra), informatica (automi finiti, dimostrazione automatica), economia e linguistica. Esistono molti domini di vincolo diversi, con diversi solutori.

Domini dell'applicazione

- Modellazione
- Specifiche eseguibili
- Risolvere problemi combinatori: programmazione, pianificazione, calendario; configurazione, layout, posizionamento, design; analisi: simulazione, verifica, diagnosi di software, hardware e processi industriali.
- Intelligenza artificiale: visione artificiale, comprensione del linguaggio naturale, ragionamento temporale e spaziale, dimostrazione di teoremi, ragionamento qualitativo, robotica, agenti, bioinformatica

Applicazioni nella ricerca

- Informatica: analisi dei programmi, robotica, agenti
- Biologia Molecolare, Biochimica, Bioinformatica: ripiegamento delle proteine, sequenziamento genomico
- Economia: programmazione
- Linguistica: analisi
- Medicina: supporto alla diagnosi
- Fisica: modellazione del sistema
- Geografia: Geo-Informazione-Sistemi

Prime applicazioni commerciali

- Lufthansa: pianificazione del personale a breve termine.
- Hong Kong Container Harbour: pianificazione delle risorse.
- Renault: pianificazione della produzione a breve termine.
- Nokia: configurazione software per telefoni cellulari.
- Airbus: disposizione della cabina.
- Siemens: Verifica del circuito.
- Caisse d'epargne: gestione del portafoglio.

Nei sistemi di supporto alle decisioni per la pianificazione, la configurazione, la progettazione, l'analisi.

□

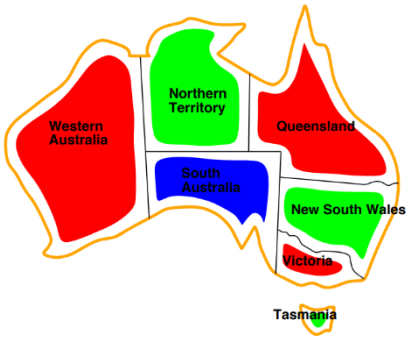
Problemi di soddisfazione dei vincoli (CSP)

Problema di ricerca standard: lo **stato** è una "scatola nera" (qualsiasi vecchia struttura di dati che supporta test obiettivo, valutazione, successore).

CSP: lo **stato** è definito dalle **variabili** X_i con **valori** dal **dominio** D_i. Il **test obiettivo** è un insieme di **vincoli** che specificano combinazioni consentite di valori per sottoinsiemi di variabili

Semplice esempio di **linguaggio di rappresentazione formale:** consente utili algoritmi di **uso generale** con più potenza rispetto agli algoritmi di ricerca standard

Esempio: Mappa-Colorazione



Variabili WA, NT, Q, NSW, V, SA, T

Domini D_i = {rosso, verde, blu}

Vincoli: le regioni adiacenti devono avere colori diversi

Esempio: **WA ≠ NT** (se la lingua lo consente), o **(WA, NT) ∈ {(rosso, verde), (rosso, blu), (verde, rosso), (verde, blu), . . .}**

Le **soluzioni** sono assegnazioni che soddisfano tutti i vincoli, ad esempio: **{WA = rosso, NT = verde, Q = rosso, NSW = verde, V = rosso, SA = blu, T = verde}**

Grafico dei vincoli

CSP binario: ogni vincolo riguarda al massimo due variabili

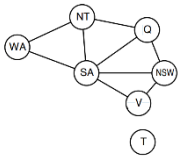


Grafico dei vincoli: i nodi sono variabili, gli archi mostrano dei vincoli

Gli algoritmi CSP generici utilizzano la struttura del grafico per accelerare la ricerca. Ad esempio, la Tasmania è un sotto-problema indipendente!

Varietà di CSP

1. **Variabili discrete**
   - **Domini finiti:** taglia **d** ⇒ **O(d^n)** assegnazioni complete. Ad esempio, CSP booleani, includono Soddisfabilità booleana (NP-completo)
   - **Domini infiniti:** interi, stringhe, ecc. Ad esempio, pianificazione del lavoro, le variabili sono i giorni di inizio/fine per ogni lavoro. **Linguaggio di vincoli**, ad esempio $\mathbf{StartJob_1} + 5 \leq \mathbf{StartJob_3}$. Vincoli **lineari** risolvibili, **non lineari** indecidibili
2. **Variabili continue:** vincoli lineari risolvibili in tempo poli dai metodi LP. Ad esempio, orari di inizio/fine per le osservazioni del telescopio Hubble

Varietà di vincoli

- Vincoli **unari:** coinvolgono una singola variabile, ad esempio **SA ≠ verde**
- Vincoli **binari:** coinvolgono coppie di variabili, ad esempio **SA ≠ WA**
- Vincoli di **ordine superiore:** coinvolgono 3 o più variabili, ad esempio, vincoli di colonna cripto-aritmetici

- Vincoli soft (**preferenze**): ad esempio, il **rosso** è meglio del **verde** spesso rappresentabile da un costo per ogni assegnazione variabile → problemi di ottimizzazione vincolata

Esempio: Cripto-aritmetica

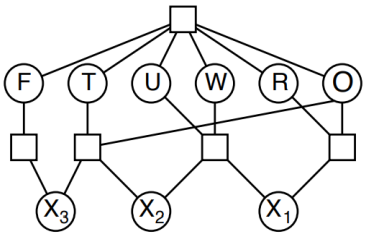
Variabili: F T U W R O $\mathbf{X_1}$ $\mathbf{X_2}$ $\mathbf{X_3}$

Domini: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Vincoli

- **alldiff(F, T, U, W, R, O)**
- **O + O = R + 10 ·** $\mathbf{X_1}$, ecc.

```
  T W O
+ T W O
-------
F O U R
```



CSP del mondo reale

- Problemi di assegnazione (ad esempio, chi insegna in quale classe)
- Problemi di orario (ad esempio, quale classe viene offerta quando e dove?)
- Configurazione hardware
- Fogli di calcolo
- Programmazione del trasporto
- Programmazione di fabbrica
- Pianificazione
- Nota che molti problemi del mondo reale coinvolgono variabili a valori reali

Formulazione di ricerca standard (incrementale)

Iniziamo con l'approccio semplice e stupido, quindi risolviamolo

Gli stati sono definiti dai valori assegnati finora

- **Stato iniziale:** l'assegnazione vuota, **{}**
- **Funzione successore:** assegnare un valore a una variabile non assegnata che non sia in conflitto con l'assegnazione corrente.
- **Test obiettivo:** l'assegnazione corrente è completata
  1) Questo è lo stesso per tutti i CSP!
  2) Ogni soluzione appare a profondità **n** con **n** variabili (usa la ricerca in profondità)
  3) Il percorso è irrilevante, quindi è possibile utilizzare anche la formulazione dello stato completo
  4) **b = (n − ℓ)d** alla profondità **ℓ,** quindi **n!** $\boldsymbol{d^n}$ foglie!

Ricerca a ritroso

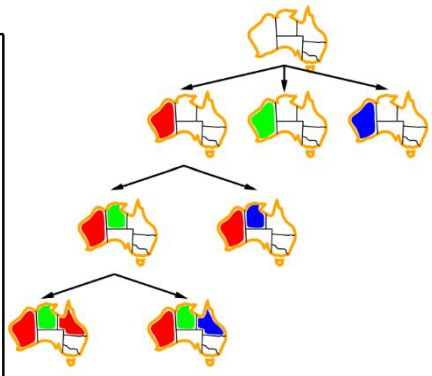
Le assegnazioni di variabili sono **commutative**, cioè [**WA = rosso** poi **NT = verde**] come [**NT = verde** poi **WA = rosso**]

È necessario considerare solo le assegnazioni a una singola variabile in ogni nodo ⇒ **b = d** e ci sono $\boldsymbol{d^n}$ foglie

Ricerca **backtracking:** ricerca in profondità per CSP con assegnazioni a variabile singola. La ricerca di backtracking è l'algoritmo disinformato di base per i CSP. Può risolvere **n**-regine per **n ≈ 25**

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
          add {var = value} to assignment
          result ← RECURSIVE-BACKTRACKING(assignment, csp)
          if result ≠ failure then return result
          remove {var = value} from assignment
  return failure
```



Miglioramento dell'efficienza del backtracking

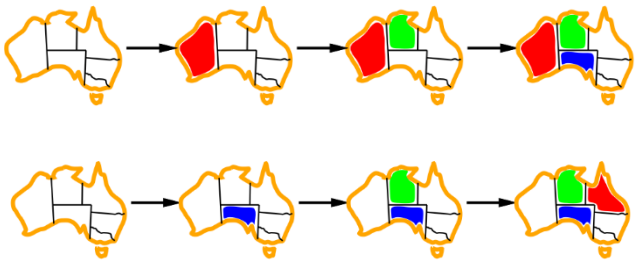
I metodi **generici** possono dare enormi vantaggi in termini di velocità:

1. Quale variabile dovrebbe essere assegnata successivamente?
2. In quale ordine dovrebbero essere provati i suoi valori?
3. Possiamo rilevare in anticipo l'inevitabile fallimento?
4. Possiamo sfruttare la struttura del problema?

La combinazione di queste euristiche rende fattibili 1000 regine:

Valori minimi rimanenti (MRV): scegli la variabile con il minor numero di valori legali

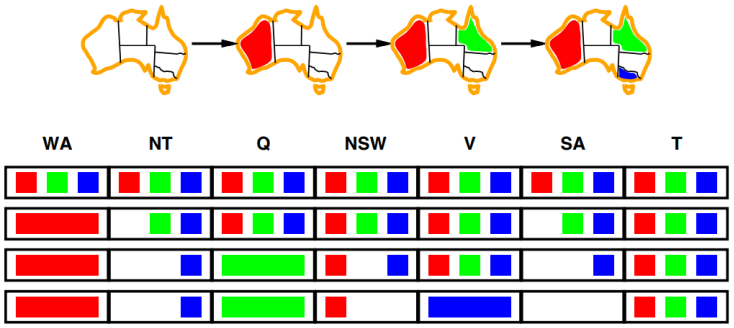
Grado euristico (tie-breaker tra le variabili MRV)**:** scegli la variabile con più vincoli sulle variabili rimanenti



Valore meno vincolante: data una variabile, scegli il valore meno vincolante, cioè quello che esclude il minor numero di valori nelle restanti variabili

Controllo in avanti

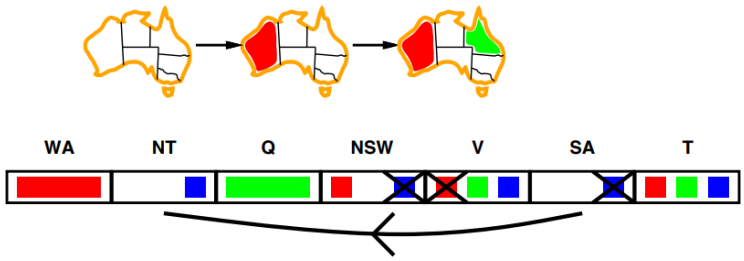
Idea: tenere traccia dei valori legali rimanenti per le variabili non assegnate. Termina la ricerca quando una variabile non ha valori legali



Propagazione dei vincoli: il controllo in avanti propaga le informazioni dalle variabili assegnate alle variabili non assegnate, ma non fornisce il rilevamento precoce di tutti i guasti

- **NT** e **SA** non possono essere entrambi blu!
- La **propagazione dei vincoli** impone ripetutamente i vincoli a livello locale

Coerenza dell'arco, la forma più semplice di propagazione rende ogni **arco coerente: X → Y** è coerente se per **ogni** valore **x** di **X** ce n'è **uno** consentito **y**



- Se **X** perde un valore, i vicini di **X** devono essere ricontrollati
- La coerenza dell'arco rileva l'errore prima del controllo in avanti
- Può essere eseguito come preprocessore o dopo ogni assegnazione

Algoritmo di coerenza dell'arco

$O(n^2 d^3)$, può essere ridotto a $O(n^2 d^2)$, ma rilevare **tutto** è NP-difficile

```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
      for each X_k in NEIGHBORS[X_i] do
        add (X_k, X_i) to queue

---

function REMOVE-INCONSISTENT-VALUES( X_i, X_j) returns true iff succeeds
  removed ← false
  for each x in DOMAIN[X_i] do
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint X_i ↔ X_j
      then delete x from DOMAIN[X_i]; removed ← true
  return removed
```

Struttura del problema

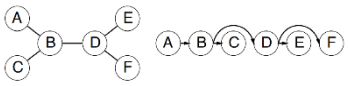
- La Tasmania e la terraferma sono **problemi secondari indipendenti**
- Identificabili come **componenti connessi** del grafico dei vincoli
- Supponiamo che ogni sotto-problema abbia **c** variabili su **n** totale

- Il costo della soluzione nel caso peggiore è **n/c** · d^c, **lineare** in **n**
- Ad esempio, **n = 80, d = 2, c = 20**
  - 2^{80} = 4 miliardi di anni a 10 milioni di nodi/sec
  - 4 · 2^{20} = 0,4 secondi a 10 milioni di nodi/sec

CSP strutturati ad albero

- **Teorema:** se il grafo dei vincoli non ha loop, il CSP può essere risolto in tempo **O(n**d^2**)**
- Confronta con CSP generali, dove il tempo nel caso peggiore è **O(**d^n**)**
- Questa proprietà si applica anche al ragionamento logico e probabilistico: un esempio importante della relazione tra restrizioni sintattiche e complessità del ragionamento.

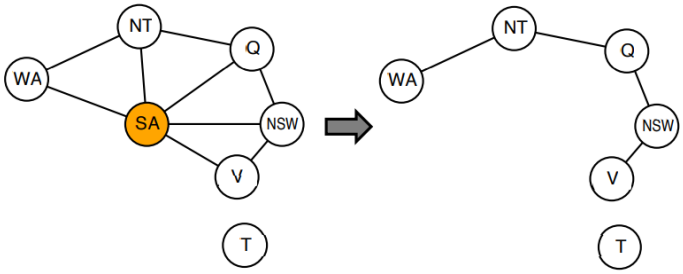
Algoritmo per CSP strutturati ad albero



1. Scegli una variabile come radice, ordina le variabili da radice a foglie in modo tale che il genitore di ogni nodo lo preceda nell'ordinamento
2. Per **j** da **n** fino a **2**, applica RemoveInconsistent**(Parent(**X_j**),** X_j**)**
3. Per **j** da **1** a **n**, assegna X_j coerentemente con **Parent(**X_j**)**

CSP quasi strutturati ad albero

Condizionamento: istanziare una variabile, eliminare i domini dei suoi vicini



Condizionamento del set di taglio: istanziare (in tutti i modi) un insieme di variabili in modo tale che il grafico dei vincoli rimanenti sia un albero

Dimensione cutset **c** ⇒ runtime **O(**d^c **· (n − c)** d^c**)**, molto veloce per **c** piccoli

Algoritmi iterativi per CSP

In salita, la ricottura simulata in genere funziona con stati "completi", ovvero tutte le variabili assegnate

Per candidarsi ai CSP:

- consentire stati con vincoli non soddisfatti
- gli operatori **riassegnano** i valori delle variabili

Selezione variabile: seleziona casualmente qualsiasi variabile in conflitto

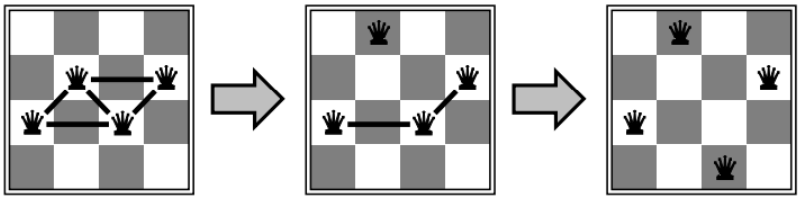
Selezione del valore per euristica dei **conflitti minimi**:

- scegli il valore che viola il minor numero di vincoli
- cioè, cronoscalata con **h(n)** = numero totale di vincoli violati

Esempio: 4-regine

- **Stati:** 4 regine in 4 colonne (4^4 **= 256** stati)
- **Operatori:** sposta la regina in colonna

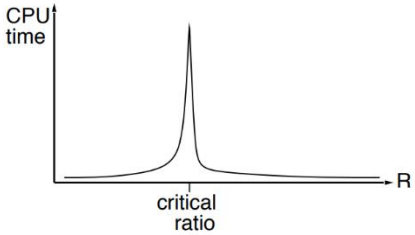
- **Test del gol:** nessun attacco
- **Valutazione: h(n)** = numero di attacchi



h = 5 **h = 2** **h = 0**

Prestazioni di mini-conflitti: dato lo stato iniziale casuale, può risolvere **n**-regine in tempo quasi costante per **n** arbitrario con alta probabilità (ad esempio, **n** = 10.000.000). Lo stesso sembra essere vero per qualsiasi CSP generato casualmente **tranne** che in un intervallo ristretto del rapporto

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Riepilogo

- I CSP sono un tipo speciale di problema:
  - stati definiti da valori di un insieme fisso di variabili
  - test obiettivo definito da vincoli sui valori delle variabili
- Backtracking = ricerca in profondità con una variabile assegnata per nodo
- L'ordinamento delle variabili e l'euristica di selezione del valore aiutano in modo significativo
- Il controllo in avanti impedisce le assegnazioni che garantiscono il fallimento successivo
- La propagazione dei vincoli (ad esempio consistenza dell'arco) svolge un lavoro aggiuntivo per vincolare i valori e rilevare le incongruenze
- La rappresentazione CSP consente l'analisi della struttura del problema
- I CSP strutturati ad albero possono essere risolti in tempo lineare
- I mini-conflitti iterativi sono generalmente efficaci nella pratica

Introduzione a MinZinc

CP - Programmazione Dichiarativa

Procedurale (Python, C, Java)**:** devi specificare i passaggi esatti per ottenere il risultato.

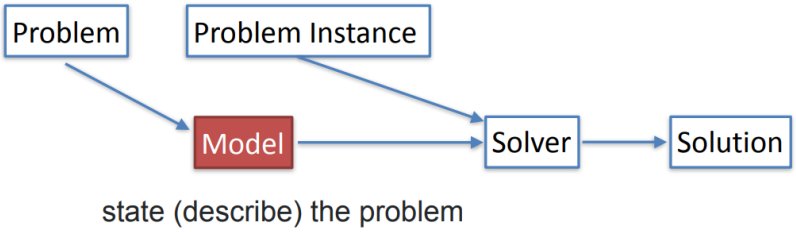
1. Vai in cucina
2. Prendi foglia di tè e acqua
3. Mescolare e scaldare sul fuoco fino a bollore
4. Mettilo in una tazza e portamelo

Dichiarativo (MiniZinc)**:** descrivi ciò che vuoi senza dover dire come farlo.

1. Il tè è composto da foglie di tè con acqua calda
2. Portami una tazza di tè.

CP – Metodologia

Model Problem - Solve Model



state (describe) the problem

CP - Come descrivere un problema al Computer?

- Entità (**Variabili**) nel problema e chi sono? (numero, stringa, insieme di numero/stringa; il dominio delle Entità)
- Relazione (**Vincoli**) tra le Entità e come dovrebbe apparire l'entità
- Il mio **obiettivo**

Per esempio. pianificare un tour

- Entità: città, distanza tra le città...
- Relazione: non visiterò una città due volte, parto dalla città x…
- Obiettivo: voglio risparmiare tempo
- => un percorso che soddisfi i vincoli

=> Formalmente, lo descriviamo come CSP (e COP)

CSP - Problema di soddisfazione dei vincoli

Un CSP è definito da

- Un insieme finito di variabili **{X1, …, Xn}**
- Un insieme di valori (dominio) per ogni variabile **dom(X1),… dom(Xn)**
- Un insieme di vincoli **{C1, …, Cm}**

Una **soluzione** a un CSP è un'assegnazione completa a tutte le variabili che soddisfa i vincoli.

COP - Problema di ottimizzazione dei vincoli

Un COP è un CSP definito sulle variabili x1, ..., xn, insieme a una funzione obiettivo f: D(x1) × ⋯ × D(xn) → Q che assegna un valore a ciascuna variabile. **Soluzione ottimale** per un COP**:** una soluzione d che ottimizzi il valore di f(d).

Modello CSP con MiniZinc

Alcune caratteristiche di Minizinc (un noto linguaggio di modellazione dei vincoli che sta diventando lo standard)

- Ogni espressione termina con ';'
- Dominio variabile, dominio indice array devono essere tutti specificati: un dominio potrebbe essere 1..10 (Sì, '..', non '…') o int o un array
- La parola chiave "vincolo" indica le regole che una soluzione dovrebbe soddisfare
- L'indice inizia da "1" e non da "0"
- '%' per i commenti
- Importa la libreria dei vincoli globali per utilizzare 'all_different()'
- Parametro (non variabile), il suo valore sarà dato dall'istanza del problema
- Variabile (nota anche come variabile decisionale), i suoi valori di dominio verranno controllati per trovare una soluzione

- I vincoli specificano i tuoi requisiti (per esempio, inizio e finisco il mio viaggio in città specifiche)

Minizinc: rappresentazione dei dati

- Parametri: i valori vengono passati dall'istanza del problema
  - **[domain]** : [nome parametro]
  - **int:** n = 10;
- Variabili: i valori dipendono dalla soluzione
  - **var [domain]** : [nome variabile]
  - **var int:** distanza_totale; % distanza percorsa in un tour
- Array: può essere un insieme di parametri o variabili
  - **array**[index_domain] of **[domain]** %parameter array
  - **array**[index_domain] of **var [domain]** %variable array
  - **array**[1..n] **of var** 1..n: città; % l'ordine della città visito da 1 a n
  - **array**[1..n,1..n] **of int:** distanza; % distanza tra una coppia di città

```
include "globals.mzn";

int: n;
array[1..n,1..n] of int: dist;
int: start_city;
int: end_city;
array[1..n] of var 1..n: city;
array[1..n] of string: city_name;

constraint  city[1] = start_city;
constraint  city[n] = end_city;
constraint  all_different(city);
```

Minizinc: alcune funzioni di aggregazione

- **sum, product, min, max**
  - sum(array_x), % somma di tutti gli elementi in array_x
  - sum(i in 1..3)(array_x[i]), % somma degli elementi da 1 a 3 di array_x
- **forall (**counter(s) in domain**) (**vincoli**)**,
- **forall (**counter(s) in domain **where** condition**) (**vincoli**)**
  - **forall** (i,j **in** 1..3 **where** i < j) (array_x[i] != array_x[j]) % i primi 3 elementi in array_x sono diversi
- **exists (**counter(s) in domain**) (**vincoli**)**

Minizinc: vincoli (regole che una soluzione deve rispettare)

- **constraints** [espressione]
  - **constraint** city[1] = start_ city
  - **constraint** all_different(city);

Minizinc: linguaggio dei vincoli

- **Operatori aritmetici:** +, -, *, /, ^, =, != (per esempio x+y^2 = z)
- **Operatori logici** (per esempio x=1 -> y != 5% if x equals to 1 then y must not be 5)
  - \/, or
  - /\, and, vincolo A /\ B; significa anche vincolo A; vincolo B;
  - ->, implica
  - ! negazione
- **Vincoli globali:** notazioni facilmente riconoscibili dai risolutori CP in cui verranno applicate tecniche di risoluzione efficienti
  - all_different(): si occupa di un numero arbitrario di variabili
    - Vincoli globali composti anche da vincoli di base: all_different(array x) = forall (i, j in index(x) where i < j) (x[i] != x[j])
  - all_equal() …

Due tipi di soluzioni:

1) Problemi di soddisfacibilità: chiamiamo semplicemente il solutore per trovare una soluzione. Scriviamo nel codice risolvere soddisfare

2) Problemi di ottimizzazione: vogliamo calcolare la migliore soluzione che ottimizzi una data funzione

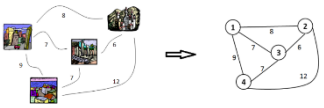
Minizinc: Obiettivo

- Definire l'obiettivo

- **var int:** total_distance = **sum**(i **in** 2..n) (dist [city[i-1], city[i]]);
- Definire lo scopo della soluzione
  - **solve minimize** total_distance;

Capire Minizinc con problemi

Problema del commesso viaggiatore: visito ogni città una volta e voglio risparmiare tempo



Codice completo - Problema del commesso viaggiatore

```
include "globals.mzn";

int: n;
array[1..n,1..n] of int: dist;
int: start_city;
int: end_city;

array[1..n] of var 1..n: city;
array[1..n] of string: city_name;

constraint  city[1] = start_city;
constraint  city[n] = end_city;
constraint  all_different(city);

var int: total_distance = sum(i in 2..n)(dist[city[i-1],city[i]]);
solve minimize total_distance;

output [city_name[fix(city[i])] ++ " -> " | i in 1..n ] ++
        ["\nTotal hours travelled: ", show(total_distance) ];

```

Problemi risolti con Minizinc

- programmazione di fabbrica (JSSP)
- percorso del veicolo (VRP)
- problemi di imballaggio (NumPart e BinPack)
- orari (esami, lezioni, treni)
- configurazione e design (hardware)
- gestione della forza lavoro (call center, ecc.)
- car sequencing (programmazione catena di montaggio)
- costruzione di super alberi (bioinformatica)
- progetto di rete (problema di telecomunicazioni)
- arrivo al gate (negli aeroporti)
- logistica (Desert Storm un esempio)
- programmi di manutenzione degli aeromobili
- pianificazione dell'equipaggio di aeromobili (compagnie aeree commerciali)
- copertura aerea per flotta navale

Un problema di investimento: 225K disponibili, 28 persone div., max 9 progetti, alcuni progetti non possono essere selezionati con altri e alcuni progetti richiedono altri progetti

Project name	Value	Budget	Personnel	Not With	Require
Ischia	6000	35	5	10	-
Speltra	4000	34	3	-	-
Hotel	1000	26	4	-	15
Restaurant	1500	12	2	-	15
ContoA	800	10	2	6	-
ContoB	1200	18	2	5	-
Scuola	2000	32	4	-	-
Dago	2200	11	1	-	7
Lago	900	10	1	-	-
small	3800	22	5	1	-
Iper	2900	27	3	15	-
Bivio	1300	28	2	-	-
Tela	800	16	2	-	2
Idro	2700	29	4	-	2
Batment	2800	22	3	11	-

Esercizi

Un primo programma MinZinc % Trova 4 numeri tali che la loro somma sia 711 e il loro prodotto sia 711*10*10*10,

- **var 1..711:** elemento1;
- **var 1..711:** elemento2;
- **var 1..711:** elemento3;

- **var 1..711:** elemento4;
- **constraint** elemento1 + elemento2 + elemento3 + elemento4 == 711; **constraint** elemento1 * elemento2 * elemento3 * elemento4 == 711 * 100 * 100 * 100;
- **constraint** 0 < elemento1 /\ elemento1 <= elemento2 /\ elemento2 <= elemento3 /\ elemento3 <= elemento4;
- **solve satisfy;**
- **output** ["{", show(elemento1), ",", show(elemento2), ",", show(elemento3), ",", show(elemento4),"}\n"];

Problema con le torte: dobbiamo preparare delle torte per una festa nella nostra scuola locale. Abbiamo 4kg di farina auto-lievitante, 6 banane, 2kg di zucchero, 500g di burro e 500g di cacao. Sappiamo fare due tipi di torte.

1. Una torta di banane che richiede 250 g di farina auto-lievitante, 2 banane schiacciate, 75 g di zucchero e 100 g di burro (venduta per $4,00)
2. Una torta al cioccolato che richiede 200 g di farina auto-lievitante, 75 g di cacao, 150 g di zucchero e 150 g di burro (venduta per $4,50)

Quante torte di ogni tipo si possono cuocere per la festa al fine di massimizzare il profitto?

Esempio 1 da Minizinc Tutorial

CAKES ≡ [DOWNLOAD]

```
% Baking cakes for the school fete

var 0..100: b; % no. of banana cakes
var 0..100: c; % no. of chocolate cakes

% flour
constraint 250*b + 200*c <= 4000;
% bananas
constraint 2*b  <= 6;
% sugar
constraint 75*b + 150*c <= 2000;
% butter
constraint 100*b + 150*c <= 500;
% cocoa
constraint 75*c <= 500;

% maximize our profit
solve maximize 400*b + 450*c;

output ["no. of banana cakes = \(b)\n",
        "no. of chocolate cakes = \(c)\n"];
```

Esempio 2 da Minizinc Tutorial

Un'altra potente funzionalità di modellazione in MiniZinc è che le variabili decisionali possono essere utilizzate per l'accesso agli array. Ad esempio, si consideri il (vecchio) problema del matrimonio stabile. Abbiamo n (etero) donne e n (etero) uomini. Ogni uomo ha una graduatoria di donne e viceversa. Vogliamo trovare un marito/moglie per ogni donna/uomo in modo che tutti i matrimoni siano stabili nel senso che:

- ogni volta che m preferisce un'altra donna o a sua moglie w, o preferisce suo marito a me, e
- ogni volta che w preferisce un altro uomo o a suo marito m, o preferisce sua moglie a w

I primi tre elementi del modello dichiarano il numero di uomini/donne e l'insieme di uomini e donne. Qui introduciamo l'uso di tipi enumerati anonimi. Sia Uomini che Donne sono insiemi di taglia n, ma non vogliamo confonderli, quindi usiamo un tipo enumerato anonimo. Ciò consente a MiniZinc di rilevare errori di modellazione laddove utilizziamo Men for Women o viceversa.

```
STABLE-MARRIAGE ≡
  int: n;

  enum Men = anon_enum(n);
  enum Women = anon_enum(n);

  array[Women, Men] of int: rankWomen;
  array[Men, Women] of int: rankMen;

  array[Men] of var Women: wife;
  array[Women] of var Men: husband;

  ▶ ASSIGNMENT
  ▶ RANKING
  solve satisfy;

  output ["wives= \(wife)\nhusbands= \(husband)\n"];
```

Le matrici rankWomen e rankMen, rispettivamente, danno la classifica femminile degli uomini e la classifica maschile delle donne. Pertanto, la voce rankWomen[w,m] fornisce la classifica per donna w di uomo m. Più basso è il numero in graduatoria, più si preferisce l'uomo o la donna.

Ci sono due array di variabili decisionali: moglie e marito. Questi, rispettivamente, contengono la moglie di ogni uomo e il marito di ogni donna.

```
ASSIGNMENT ≡
  constraint forall (m in Men) (husband[wife[m]]=m);
  constraint forall (w in Women) (wife[husband[w]]=w);
```

Assicurarsi che l'assegnazione di mariti e mogli sia coerente: w è la moglie di m implica m è il marito di w e viceversa. Notare come in marito[moglie[m]] l'espressione indice moglie[m] sia una variabile decisionale, non un parametro.

I prossimi due vincoli sono una codifica diretta della condizione di stabilità:

```
RANKING ≡
  constraint forall (m in Men, o in Women) (
       rankMen[m,o] < rankMen[m,wife[m]] ->
           rankWomen[o,husband[o]] < rankWomen[o,m] );

  constraint forall (w in Women, o in Men) (
       rankWomen[w,o] < rankWomen[w,husband[w]] ->
           rankMen[o,wife[o]] < rankMen[o,w] );
```

Linguaggi di programmazione logica

Preliminari sui sistemi di transizione

La lingua: una famiglia di linguaggi di programmazione basati su FOL, dove i programmi sono formule (più precisamente, clausole). Una query per un programma (obiettivo) è anche una formula (clausola).

Semantica: poiché i programmi e le query sono un insieme di formule, si possono considerare come una teoria FOL e fornire una lettura logica.

- **Teoria dei modelli** (chiamata anche semantica dichiarativa in LP)
- **Teoria della dimostrazione in termini di risoluzione** (chiamata anche semantica operazionale in LP)
- **Risultati di solidità e completezza**

Calcolo come deduzione: la novità per i linguaggi di programmazione logica è che il calcolo è esattamente lo stesso della deduzione logica (risoluzione). Ciò consente di dimostrare le proprietà dei programmi (ad esempio la correttezza) in un modo molto più semplice

Semantica operativa: definiamo un sistema di transizione che descrive il funzionamento dell'interprete del linguaggio. La semantica operazionale corrisponde alla risoluzione, ma fornisce una definizione più chiara del calcolo, e può essere compresa anche senza considerare affatto la logica.

Lingue dichiarative: i linguaggi di programmazione logica sono linguaggi dichiarativi, nel senso che il programmatore specifica il "cosa", mentre la macchina si occupa del "come".

Preliminari

Sintassi dei linguaggi di programmazione logica (EBNF) definito come segue:

La forma estesa Backus-Naur (EBNF) è una notazione costituita da regole di produzione della forma: "Nome: G, H ::= A | B, Condizione" dove le lettere maiuscole sono entità sintattiche, i simboli G e H sono definiti dalla regola con nome Nome e | se la Condizione vale, allora G e H possono essere della forma A o B.

Semantica dei linguaggi di programmazione logica (sistemi di transizione di stato, chiamati anche calcolo raffinato):

- I sistemi di transizione di stato forniscono la semantica **operativa** (detta anche procedurale): questa definisce come si evolve il calcolo.
- Il calcolo è deduzione (risoluzione): useremo sistemi di transizione di stato per descrivere i passaggi di risoluzione
- **Si noti** che la semantica operazionale, come definita da un sistema di transizione di stato, può essere compresa anche **senza** alcun riferimento alla logica! (è possibile anche una semantica dichiarativa, ma non la considereremo)

Sistema di transizione di stato (semplice)

- Un Simple State **T**ransition **S**ystem (TS) è una coppia (S, ->) dove
  - ➔ S insieme di stati
  - ➔ -> relazione binaria sugli stati: relazione di transizione
  - ➔ (stato) transizione da S_1 a S_2 possibile se (qualche data) condizione vale, cioè S_1 e S_2 sono in relazione ->, scritto S_1 -> S_2
- distinti sottoinsiemi di S: stati iniziale e finale
- S_1 -> S_2 -> ... -> Derivazione S_n (calcolo)
- ->* chiusura riflessiva-transitiva di ->
- Riduzione è sinonimo di transizione

Regole di transizione

La relazione -> è definita utilizzando regole di transizione della forma: "IF Condizione THEN S -> S'"

- (stato) transizione (riduzione, passo di derivazione, calcolo) da S a S' possibile se la condizione Condizione è valida
- semplice concretizzare un calcolo ("⊢") in un sistema di transizione di stato ("->")

Calcolo raffinato

Un **Calcolo Raffinato** è un linguaggio del primo ordine, dotato di una **congruenza** (utilizzata per fattorizzare formule equivalenti e per ridurre il numero di regole) e di un **sistema di transizione** che definisce le regole di calcolo (semantica operativa); è fornito da un Triplo (Σ, ≡,T) dove:

- Σ è una firma per un linguaggio logico del primo ordine

- ≡ è una congruenza (relazione di equivalenza) sugli stati
- T = (S, ->) è un sistema di transizione, dove uno stato s ∈ S rappresenta un'espressione logica sulla firma Σ

Congruenza: utilizzata per identificare gli stati considerati equivalenti ai fini del calcolo (*congruenza* invece di modellare con regole di transizione aggiuntive). Formalmente la congruenza è una **relazione di equivalenza compatibile con la struttura:**

(Riflessività) A ≡ A

(Simmetria) If A ≡ B then B ≡ A

(Transitività) If A ≡ B and B ≡ C then A ≡ C

Esempio derivato da tautologie: (X = 3) ≡ (3 = X)

Commutatività: $G_1 \wedge G_2 \equiv G_2 \wedge G_1$

Associatività: $G_1 \wedge (G_2 \wedge G_3) \equiv (G_1 \wedge G_2) \wedge G_3$

Identità: G ∧ T ≡ G

Assorbimento: G ∧ ⊥ ≡ ⊥

Esempio: sistema di transizione per il calcolo della risoluzione

Gli **Stati** di un sistema di transizione per il calcolo della risoluzione sono definiti come segue

- Uno *stato* è un insieme di clausole
- Lo stato enfitiziale è un insieme di clausole che rappresentano la teoria e la conseguenza negata
- Lo stato *finale* contiene la clausola vuota

Le **regole di transizione** per la logica proposizionale sono le seguenti:

- **Resolvent**

  IF $R \vee A \in S$ and $R' \vee \neg A \in S$ for some atom A
  THEN $S \mapsto S \cup \{F\}$ where $F = R \vee R'$ is called Resolvent.

- **Factor**

  IF $R \vee L \vee L \in S$ for some literla L
  THEN $S \mapsto S \cup \{F\}$ where $F = R \vee L$ is called Factor

Le **regole di transizione** per FOL sono le seguenti

- **Resolvent**

IF $R \vee A \in S$ and $R' \vee \neg A \in S$ and
σ is a most general unifier for the atoms A and A',
THEN $S \mapsto S \cup \{F\}$ where $F = (R \vee R')\sigma$ is called Resolvent.

- **Factor**

IF $R \vee L \vee L' \in S$ and
σ is a most general unifier for the literals L and L',
THEN $S \mapsto S \cup \{F\}$ where $F = (R \vee L)\sigma$ is called Factor

Programmazione logica

Sintassi e semantica

Programma: definisce un insieme di conseguenze, che è il suo significato.

Programma logico: insieme di assiomi o regole che definiscono le relazioni tra gli oggetti. L'arte della programmazione logica consiste nel costruire programmi concisi ed eleganti che abbiano il significato desiderato.

Calcolo del programma logico**:** deduzione delle conseguenze del programma

Calcolo LP (Logic Programming): Sintassi

- **Goal** (Query)**:** empty goal T (in alto) o ⊥ (in basso), o **atom**, o congiunzione di obiettivi
- **Clause** (Horn)**:** A <- G
  - **head A:** atomo
  - **body G:** goal
- Convenzioni di denominazione
  - **fact:** clausola di forma A <- T
  - **rule:** tutti gli altri
- Simbolo predicato definito: si verifica all'inizio di una clausola
- **Program** (logico)**:** insieme finito di clausole di Horn (una clausola di Horn è una clausola con al massimo un letterale positivo)

Atom:	A, B	$::=$	$p(t_1, \ldots, t_n),\ n \geq 0$
Goal:	G, H	$::=$	$\top \mid \bot \mid A \mid G \wedge H$
Clause:	K	$::=$	$A \leftarrow G$
Program:	P	$::=$	$K_1 \ldots K_m,\ m \geq 0$

Calcolo LP: Sistema di transizione di stato

Definiamo ora la semantica di LP in termini di un sistema di transizione:

- Uno stato è una coppia $\langle G, \theta \rangle$ dove G è un goal e θ è una sostituzione
- Uno stato *iniziale* ha la forma $\langle G, \epsilon \rangle$

- Uno *stato finale di successo* ha la forma $\langle \mathrm{T}, \theta \rangle$
- Uno *stato finale fallito* ha la forma $\langle \bot, \epsilon \rangle$

Derivazione:

- di *successo* se il suo stato finale ha successo
- *fallita* se il suo stato finale è fallito
- *infinita* se esiste una sequenza infinita di stati e transizioni $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \cdots$

Goal G è

- di *successo* se ha una derivazione riuscita che inizia con $\langle G, \epsilon \rangle$
- *finitamente fallita* se ha fallito solo derivazioni che iniziano con $\langle G, \epsilon \rangle$

Sostituzione della risposta del computer

Dato un obiettivo iniziale G (o query) e un programma P, se esiste una derivazione riuscita (in P):

$$\langle G, \epsilon \rangle \rightarrow * \langle \mathrm{T}, \theta \rangle$$

Quindi la sostituzione θ si chiama Computed Answer Aubstitution (C.A.S.) di G (in P). Il C.A.S. è il risultato del calcolo: i valori associati dal C.A.S. alle variabili nell'obiettivo forniscono i risultati.

Regole di transizione LP

Unfold	
If	$(B \leftarrow H)$ is a fresh variant of a clause in P
and	β is the most general unifier of B and $A\theta$
then	$\langle A \wedge G, \theta \rangle \mapsto \langle H \wedge G, \theta\beta \rangle$
Failure	
If	there is no clause $(B \leftarrow H)$ in P
with	a unifier of B and $A\theta$
then	$\langle A \wedge G, \theta \rangle \mapsto \langle \bot, \epsilon \rangle$

Non determinismo

La transizione **Unfold** mostra due tipi di non determinismo.

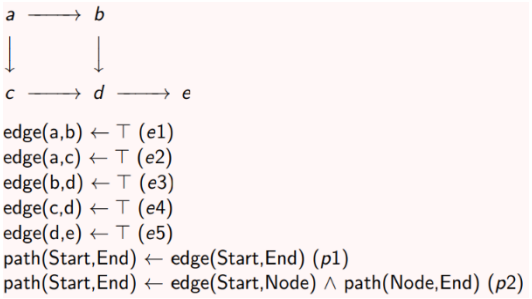
1. **non importa non determinismo:** influenza la lunghezza di derivazione (infinitamente nel peggiore dei casi). Qualsiasi atomo in A ∧ G può essere scelto come atomo A secondo la congruenza definita sugli stati
2. **non so non determinismo:** determina la risposta calcolata di derivazione. Può essere scelta qualsiasi clausola (B <- H) in P per la quale B e A θ sono unificabili

Risoluzione SLD: regola di selezione Risoluzione lineare guidata per clausole definite. L'idea è l'uso di una funzione di selezione per selezionare l'atomo nell'obiettivo per l'applicazione della regola di spiegamento (che corrisponde alla selezione di un letterale per l'applicazione della regola di risoluzione). Questo elimina il non importa non determinismo (il non so rimane). Utilizzando tutte le possibili clausole con una specifica regola di selezione si ottiene un **albero di ricerca** chiamato albero **SLD**. Occorre definire una strategia di ricerca per eliminare anche il non so non determinismo, ovvero visitare l'albero SLD per trovare una derivazione di successo.

- **Implementazione del Prolog:** Prolog utilizza una regola di selezione che seleziona l'atomo più a sinistra. Prolog utilizza per la strategia di ricerca l'ordine testuale delle clausole con backtracking (cronologico). Ciò fornisce un'esplorazione approfondita dell'albero SLD da sinistra a destra. Si può ottenere un'implementazione efficiente utilizzando un approccio basato su stack, ma si può rimanere intrappolati in infinite derivazioni (la ricerca in ampiezza è troppo inefficiente).

Esempio: accessibilità in DAG

e1 en e p1, p2 sono nomi di regole nel metalinguaggio, fanno parte della sintassi LP



$$
\begin{array}{ll}
 & \langle \mathtt{path(b,Y)}, \varepsilon \rangle \\
\mapsto_{(p1)} & \langle \mathtt{edge(S,E)}, \{\mathtt{S} \mapsto \mathtt{b}, \mathtt{E} \mapsto \mathtt{Y}\} \rangle \\
\mapsto_{(e3)} & \langle \top, \{\mathtt{S} \mapsto \mathtt{b}, \mathtt{E} \mapsto \mathtt{d}, \mathtt{Y} \mapsto \mathtt{d}\} \rangle
\end{array}
$$

With the second rule *p2* for path selected:

$$
\begin{array}{ll}
 & \langle \mathtt{path(b,Y)}, \varepsilon \rangle \\
\mapsto_{(p2)} & \langle \mathtt{edge(S,N)} \wedge \mathtt{path(N,E)}, \{\mathtt{S} \mapsto \mathtt{b}, \mathtt{E} \mapsto \mathtt{Y}\} \rangle \\
\mapsto_{(e3)} & \langle \mathtt{path(N,E)}, \{\mathtt{S} \mapsto \mathtt{b}, \mathtt{E} \mapsto \mathtt{Y}, \mathtt{N} \mapsto \mathtt{d}\} \rangle \\
\mapsto_{(p1)} & \langle \mathtt{edge(N,E)}, \{\mathtt{S} \mapsto \mathtt{b}, \mathtt{E} \mapsto \mathtt{Y}, \mathtt{N} \mapsto \mathtt{d}\} \rangle \\
\mapsto_{(e5)} & \langle \top, \{\mathtt{S} \mapsto \mathtt{b}, \mathtt{E} \mapsto \mathtt{e}, \mathtt{N} \mapsto \mathtt{d}, \mathtt{Y} \mapsto \mathtt{e}\} \rangle
\end{array}
$$

Albero di ricerca parziale

With the first rule:

$$
\begin{array}{ll}
 & \langle \mathtt{path(f,g)}, \varepsilon \rangle \\
\mapsto_{(p1)} & \langle \mathtt{edge(S,E)}, \{\mathtt{S} \mapsto \mathtt{f}, \mathtt{E} \mapsto \mathtt{g}\} \rangle \\
\mapsto & \langle \bot, \varepsilon \rangle
\end{array}
$$

With the second rule (and special selection) we get an infinite derivation:

$$
\begin{array}{ll}
 & \langle \mathtt{path(f,g)}, \varepsilon \rangle \\
\mapsto_{(p2)} & \langle \mathtt{path(N,E)} \wedge \mathtt{edge(S,N)}, \{\mathtt{S} \mapsto \mathtt{f}, \mathtt{E} \mapsto \mathtt{g}\} \rangle \\
\mapsto_{(p2)} & \langle \mathtt{edge(S,N)} \wedge \mathtt{edge(N,N1)} \wedge \mathtt{path(N1,E)}, \{\mathtt{S} \mapsto \mathtt{f}, \mathtt{E} \mapsto \mathtt{g}\} \rangle
\end{array}
$$

Semantica Dichiarativa: l'implicazione (G -> A) è una clausola (Horn), dove tutte le variabili sono implicitamente considerate universalmente quantificate. La lettura logica di un programma P si ottiene considerando la congiunzione delle clausole di P. Si noti che questa lettura logica è incompleta, poiché si possono ricavare solo informazioni positive. Per trattare anche le informazioni negative dovremmo prendere il completamento di un programma, cioè trasformare le implicazioni in doppie implicazioni (aggiungere condizioni sufficienti).

- Esempio: in DAG con nodi a, b, c, d, e: P = path(f,g) tuttavia non possiamo derivare dal programma P = ¬path(f,g) e neanche P = path(f,g)

Solidità e completezza della risoluzione SLD: dato programma logico P, se un C.A.S. σ di G si ottiene utilizzando una regola di selezione r si ottiene anche utilizzando una diversa regola di selezione r'. Inoltre, goal G e θ sostituzione:

- **Solidità:** se θ è una risposta calcolata di G, allora P |= G θ.
- **Completezza:** se P |= G θ, allora esiste una sostituzione di risposta calcolata σ di G, tale che G θ = G $\sigma\beta$

Prolog

Liste, aritmetica, strutture dati

Sintassi del Prolog

- Variabili: **X, Y** (Iniziale maiuscola)
- Costanti: **alice, a** (Iniziale minuscola)
- Funzioni: **padre(X)** (Iniziale minuscola)
- Predicati: **fratello(X, Y)** (Iniziale minuscola)
- **Liste**
- Fatti: head.

- Regole: head :- body.

Semantica del Prolog: risoluzione SLD

1. **Partiamo dal goal** e cerchiamo di dimostrarlo (in realtà, negazione del goal originale e cerchiamo di ottenere la clausola vuota)
2. **Regola di selezione**: gli atomi più a sinistra sono selezionati negli obiettivi
3. **Ordine delle clausole**: ordine sintattico delle clausole nel programma (dall'alto al basso del programma)

LISTE

In Prolog una lista è rappresentata da un termine così definito:

- [ ] è l'elenco vuoto
- [t1|t2] è la lista il cui primo elemento (head) è t1 e il resto della lista (tail) è t2.

Abbreviazioni:

- [a, b] significa [a | [b | []]]
- [a, b | X] significa [a | [b | X]]

Esempio: length(L, N): N è la lunghezza della lista L

- length([], 0).
- length([_ |L],s(N)) :- length(L, N).
  - "_" rappresenta la variabile anonima
- ?- length([X|[Y|[a|[]]]],R).
- ?- length([a,b,Z],R).
- ?- length([a|x],R).

Esempio: member(X, L) "X è un membro della lista L"

- member (X, [X | _]).
- member (X, [_ | L]) :- member (X, L).
- ?- member (a, [c | [b | [a | []]]]).
- ?- member (a, [a | [b | [a | []]]]).
- ?- member (a, [c | [Z | [a | []]]]).
- ?- member (a, [c | Z]).

Esempio: append(L1, L2, L) L è la concatenazione di L1 e L2

- append ([], L, L).
- append ([X | L1], L2, [X | L]) :- append (L1, L2, L).
- ?- append ([a, b], [c], Z).
- ?- append ([X, a, Y], [b, c], Z).
- ?- append ([a, b], [c | Y], Z).
- ?- append ([X], [c | Y], [a, c, e, f]).

Tipi (Prolog **non** è digitato)

- ?- member (a, [a | I_am_not_a_list]).
  - succeeds!
- append ([], I_am_not_a_list, L).
  - succeeds with C.A.S. {L / I_am_not_a_list}

La lista dei predicati: possiamo controllare il tipo di lista introducendo il predicato list/1

- list ([]).
- list ([_ | L]) :- list (L).

Esempio: delete(X, LB, LS) lista LS si ottiene da LB cancellando un elemento che si unifica con X.

- delete(X, LB, LS)
  - append (F, [X | R], LB),
  - append (F, R, LS).
- delete(X, [X | R], R).
- delete(X, [Y | L], [Y | R]): - delete (X, L, R).

Esercizio: descrivi il comportamento dell'interprete Prolog per ciascuno dei seguenti obiettivi.

- ?- delete (a, [c, d, a, b, a], Z).
- ?- delete (a, [X | Z], [a, b, a, c]).
- ?- delete (X, [a, f, a, c], Z).
- ?- delete (a, [b, X, c], Y).
- ?- delete (a, [b | X], Y).

Aritmetica

- Interi, float, numeri per una data base x' sono costanti predefinite:
  - integer: 0, 1, 9977, -79393
  - float: 4.5e3, 1.0, -0.5e+7
  - base x': 2'101, 8'174
- Anche diverse funzioni numeriche (+, _, *, abs, max, …) sono predefinite

Aritmetica incorporata: ogni variabile che appare in un'espressione **deve essere dichiarata** quando valutata

- Z **is** X il valore di X è unificato con Z
  - p (X) :- X is 3 + 5.
    - ?- p (X).
    - X = 8
  - p (X) :- Y is 3 + 5, X is Y.
    - ?- p (X).
    - X = 8
  - p (X) :- X is Y, Y is 3 + 5.
    - ?- p (X).
    - error
- X =:= Y il valore di X e Y sono uguali
- X=\= Y il valore di X e Y sono diversi
- X > Y il valore di X è maggiore del valore di Y

"length" con l'aritmetica: length(L, N) "N è la lunghezza della lista L"

- length ([], 0).
- length ([_ | L], N1) :- length (L, N), N1 is N + 1.

Esempio: merge(L1, L2, L) L è la fusione dell'ordinato lists L1 e L2.

- merge ([], Y, Y).
- merge (Y, [], Y).
- merge ([X | L1], [Y | L2], [X | L]) :- X ≤ Y, merge (L1, [Y | L2], L).
- merge ([X | L1], [Y | L2], [Y | L]) :- X> Y, merge ([X | L1], L2, L).

Esempio: split(L, L1, L2) L = L1.L2 e length (L1) = length (L2)

- split ([X | L], [X | L1], L2) :- split (L, L2, L1).
- split ([], [], []).
- split ([a, b, c], L1, L2)

- = {L1 / [a | L1 ']} => split ([b, c], L2, L1')
- = {L2 / [b | L2 ']} => split ([c], L1', L2 ')
- = {L1 '/ [c | L1 "]} => split ([], L2', L1")
- = {L2 '/ [], L1 "/ []} =>'

Esempio: merge_sort(L,LSort) LSort è la versione ordinata di L

- merge_sort([], []).
- merge_sort([X], [X]).
- merge_sort([X, Y | R], Lsort) :- split([X, Y | R], L1, L2),

  merge_sort(L1, L1Sort),

  merge_sort(L2, L2Sort),

  merge(L1Sort, L2Sort, LSort).

Se lungh(L) ≥ 2 allora, dopo lo split, lungh(L1) < lungh(L) e lungh(L2) < lungh(L)

Strutture dati

In Prolog le strutture dati sono rappresentate mediante l'uso di termini. Dobbiamo decidere la rappresentazione, i costruttori ei predicati.

Esempio: albero binario

- **tree/3** (funzione che costruisce un albero non vuoto): **albero (Radice, Sinistra, Destra).**
- **empty(T) test per albero vuoto:** l'albero vuoto è rappresentato dalla costante "void"
  - empty(void).
- **build_tree(X, T1, T2, T): albero costruito usando il Root X, il sottoalbero sinistro T1 e il sottoalbero destro T2**
  - build_tree(X, T1, T2, albero(X,T1,T2)).
- **albero_sinistro(T,L): L è il sottoalbero sinistro di T**
  - left_tree(tree(Root,Left,Right), Left).

Preordina l'attraversamento dell'albero

pre_visit(T, L) L è l'elenco delle chiavi di nodi di T ottenuti da un attraversamento di pre-ordine

- pre_visit (void, []).
- pre_visit (tree (Root, Left, Right), [Root | L]) :-

  pre_visit (Left, L1),

  pre_visit (Right, L2),

  append (L1, L2, L).

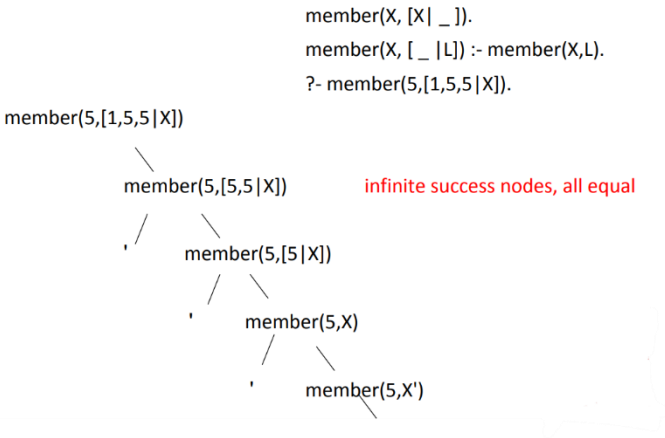
Esempio: rotate_left(T, Tr) Tr si ottiene da T con una rotazione a sinistra

rotate_left (tree (R1, Alfa, tree (R2, Beta, Delta)), tree (R2, tree (R1, Alfa, Beta), Delta)).

Il predicato ! (taglio)

Utilità di cut (!): l'albero SLD che viene cercato dall'interprete Prolog per un dato obiettivo G e il programma P può contenere diversi percorsi ripetuti e percorsi falliti (fonte di inefficienza). Il ! ci permette di potare parte dei sentieri inutili.

Esempio 1

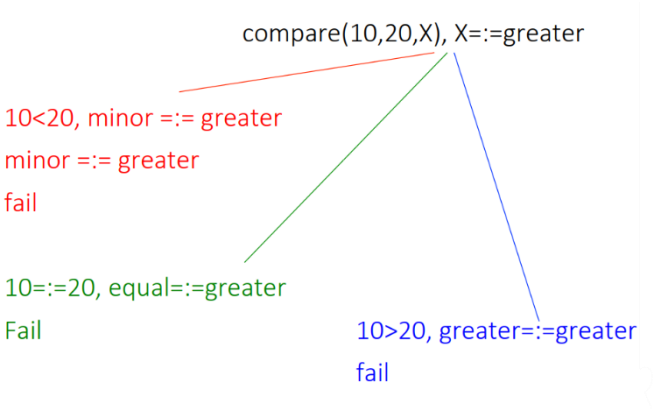


Esempio 2

- compare(X, Y, minor): - X <Y.
- compare(X, Y, equal): - X =: = Y.
- compare(X, Y, greater): - X> Y.
- ?- compare (10,20, X), X = greater.

Notiamo che l'obiettivo fallisce: le tre regole si escludono a vicenda, si può evitare di continuare la ricerca dopo il primo tentativo (cioè il primo fallimento).

Esempio 3



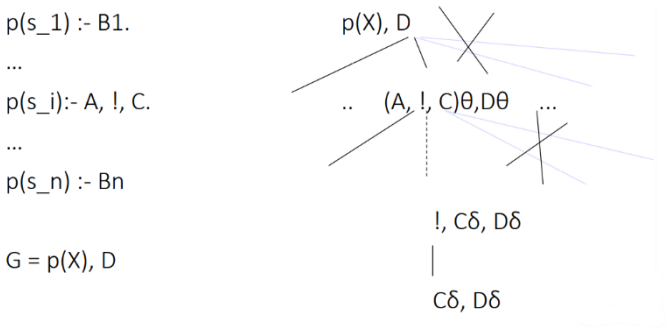
Il taglio: il problema con gli esempi precedenti deriva dalla strategia di ricerca dell'interprete prologo che cerca sempre di ispezionare **l'intero albero SLD**. Per modificare questa strategia il programmatore può utilizzare il predicato **cut (!)** che può essere inserito nel corpo di qualsiasi clausola. L'esecuzione del taglio predicato ha l'effetto di potare l'albero SLD, tagliando rami ridondanti o morti secondo le regole seguenti:

Significato operativo di !: l'esecuzione del predicato **!** rende definitiva ogni scelta fatta nel calcolo fatto per risolvere l'atomo unificato con H, cioè rende definitiva la **scelta della clausola c** e le **scelte fatte per risolvere A**

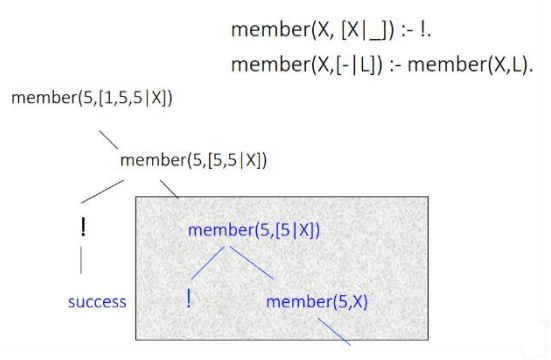
Let **! , G1:** è un obiettivo che ha tagliato (!) come atomo più a sinistra e lascia

c: H :- A, ! , B. (la clausola che ha introdotto quell'occorrenza di taglio)

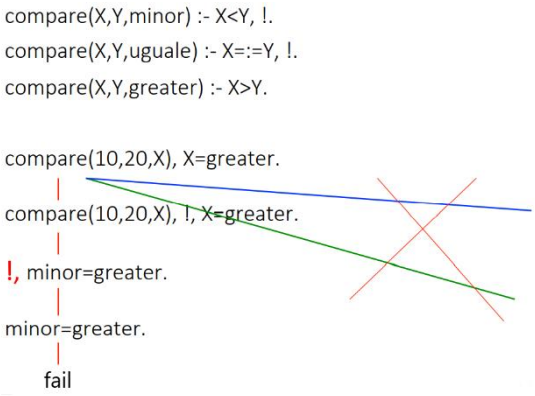
Esempio 1: la parte dell'albero cancellata non verrà più analizzata **(anche nel caso in cui il goal Cδ, Dδ fallisca)**



Esempio 2: la parte grigia è tagliata (non cercata)



Esempio 3



Semantica del taglio

Negli esempi precedenti l'introduzione del taglio non ha alterato la corrispondenza tra semantica operazionale e semantica dichiarativa: si ottiene lo stesso insieme di soluzioni (C.A.S.) con e senza l'esecuzione del taglio. Questo non è sempre vero: l'introduzione del taglio può alterare il significato del programma modificando l'insieme delle soluzioni.

Il predicato fail: è un altro predicato predefinito che quando eseguito genera un errore.

Esempio 1: quando X si unifica con la testa della lista [X|-] prima viene valutato il taglio e questo taglia tutte le scelte successive, poi il predicato fail che interrompe il calcolo (effetto desiderato). È un modo per ottenere la negazione!

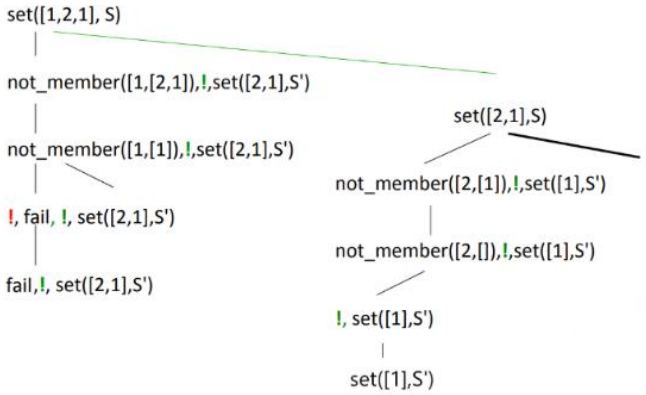
- not_member (X, []).
- not_member (X, [X | _]) :- !, fail.
- not_member (X, [- | L]) :- not_member (X, L).

Esempio 2: set(L,S) S si ottiene dall'insieme L rimuovendo i duplicati

- set([],[]).
- set([X|L],[X|S]) :- not_member(X, L), !, set(L, S).
- set([X|L], S) :- set(L, S).

In un albero SLD possono essere necessari più tagli. Ogni taglio, se valutato, può generare una potatura. È importante un'attenta analisi degli effetti dei diversi tagli.

Esempio 3



Esempio 4: definisci flatten(L, L') dove L è un elenco di liste e L' è la versione flatten di L, ad es. L=[[a,b],[c],[[d,e],f]] L=[a,b,c,d,e]

- flatten ([X | L], F) :- flatten (X, F1), flatten (L, F2), append (F1, F2, F).
- flatten (X, [X]) :- constant (X), X = ! = [].
- flatten ([], []).

flatten_dl: sostituiamo ogni lista di risultati con una lista di differenze

- flatten_dl([X | L], F-Z) :- flatten_dl (X, F1-Y), flatten_dl (L, F2-W), append_dl(F1-Y, F2-W, F-Z).
- flatten_dl(X, [X | Z] -Z) :- costant (X), X =! = [].
- flatten_dl([], Z-Z).

Risolvendo aggiungi nella prima clausola Y = F2, W = Z e F1 = F:

- flatten_dl ([X | L], F-Z) :- flatten_dl (X, F-F2), flatten_dl (L, F2-Z).
- flatten_dl (X, [X | Z] -Z) :- costant (X), X =! = [].
- flatten_dl ([], Z-Z).

Esercizio: definire il tipo di dati coda con le operazioni in coda e dequeue.

CLP

Sintassi CLP

- firma aumentata con *simboli di vincolo*
- teoria dei vincoli del primo ordine (CT) coerente
- almeno i simboli di vincolo T e ⊥
- uguaglianza sintattica ˙= come vincoli (includendo CET in CT)
- vincoli gestiti (risolti) da un risolutore di vincoli predefinito e dato
- **Atom:** espressione p(t_1,..., t_n), con simbolo predicato p/n
- **Constraint: vincolo atomico** (espressione c(t_1,..., t_n), con simbolo di vincolo n-ario c/n) o congiunzione di vincoli

- **Goal:** T (in alto), o ⊥ (in basso), o atomo, o un vincolo atomico, o congiunzione di obiettivi
- **CL Clause:** A ← G, con atomo A e obiettivo G
- **CL Program:** insieme finito di clausole CL

Atom:	A, B	::=	$p(t_1, \ldots, t_n),\ n \geq 0$
Constraint:	C, D	::=	$c(t_1, \ldots, t_n) \mid C \wedge D,\ n \geq 0$
Goal:	G, H	::=	$\top \mid \bot \mid A \mid C \mid G \wedge H$
CL Clause:	K	::=	$A \leftarrow G$
CL Program:	P	::=	$K_1 \ldots K_m,\ m \geq 0$

Semantica CLP tramite sistema di transizione

- stato $\langle G, C \rangle$: G obiettivo (negozio), C vincolo (store)
- stato iniziale: $\langle G, T \rangle$
- stato finale di successo: $\langle T, C \rangle$ i e C è diverso da ⊥
- stato finale fallito: $\langle G, \bot \rangle$
- derivazioni e obiettivi riusciti e falliti: come nel calcolo LP

Semantica operativa CLP

Unfold	
If	$(B \leftarrow H)$ is a fresh variant of a clause in P
and	$CT \models \exists ((B \doteq A) \wedge C)$
then	$\langle A \wedge G, C \rangle \mapsto \langle H \wedge G, (B \doteq A) \wedge C \rangle$
Failure	
If	there is no clause $(B \leftarrow H)$ in P
with	$CT \models \exists ((B \doteq A) \wedge C)$
then	$\langle A \wedge G, C \rangle \mapsto \langle \bot, \bot \rangle$
Solve	
If	$CT \models \forall ((C \wedge D_1) \leftrightarrow D_2)$
then	$\langle C \wedge G, D_1 \rangle \mapsto \langle G, D_2 \rangle$

CLP Unfold: confronto con LP

- generalizzazione di **LP**
- unificatore più generale in LP
- vincolo di uguaglianza tra B e A nel contesto del negozio di vincoli C, aggiungi il vincolo di uguaglianza al negozio C
- (B ˙=A) è una scorciatoia per eguagliare gli argomenti di B e A a coppie

CLP Solve

- forma di semplificazione dipende dal sistema di vincoli e dal suo risolutore di vincoli
- cercando di semplificare i vincoli incoerenti a ⊥
- uno stato finale fallito può essere raggiunto usando **Solve**

Sistema di transizione di stato CLP (rispetto a LP)

- Come in LP, due gradi di non determinismo nel calcolo:
    - selezionando l'obiettivo
    - selezionando la clausola
- Come in LP, alberi di ricerca: principalmente risoluzione SLD
- LP accumula e compone **sostituzioni**: $\{X_1 \rightarrow t_1, \ldots, X_n \rightarrow t_n\}$
- CLP accumula e semplifica i **vincoli** di uguaglianza: $X_1 ^\circ = t_1 \wedge \ldots \wedge X_n ^\circ = t_n$
    - Come le sostituzioni, i vincoli non vengono mai rimossi dall'archivio dei vincoli (l'informazione aumenta in modo monotono durante le derivazioni)
    - I vincoli riassumono diversi (anche infiniti) risposte LP in una risposta (intenzionale), ad esempio: "$X + Y \geq 3 \wedge X + Y \leq 3$ semplificato in X + Y °= 3" (le variabili non hanno bisogno di avere un valore)

- Vincoli con significato usuale:
  - $\leq$ ordine totale
  - = uguaglianza sintattica

Esempio CLP: Min

- min(X, Y, Z) $\leftarrow X \leq Y \wedge X = Z (c1)$
- min(X, Y, Z) $\leftarrow X \leq Y \wedge X = Z (c2)$

Esempio CLP: albero di ricerca per min(1, 2, C)

L'uso di (c2) porta a memorizzare vincoli incoerenti $2 \leq 1 \wedge 2 = C$ – errore di derivazione

Goal `min(1,2,C)`:

$\langle \texttt{min}(1,2,C),\ \texttt{true} \rangle$
$\mapsto_{\textbf{Unfold}\ (c1)}$ $\langle X \leq Y \wedge X \doteq Z,\quad 1 \doteq X \wedge 2 \doteq Y \wedge C \doteq Z \rangle$
$\mapsto_{\textbf{Solve}}$ $\langle \top,\ C \doteq 1 \rangle$

Esempio CLP – Min (più derivazioni)

min(X,Y,Z) $\leftarrow$ X$\leq$Y $\wedge$ X$\doteq$Z ($c1$)
min(X,Y,Z) $\leftarrow$ Y$\leq$X $\wedge$ Y$\doteq$Z ($c2$)

- Goal `min(A,2,1)`:

  $\langle \texttt{min}(A,2,1),\ \texttt{true} \rangle$
  $\mapsto_{\textbf{Unfold}\ (c1)}$ $\langle X \leq Y \wedge X \doteq Z,\quad A \doteq X \wedge 2 \doteq Y \wedge 1 \doteq Z \rangle$
  $\mapsto_{\textbf{Solve}}$ $\langle \top,\ A \doteq 1 \rangle$

  but fails with ($c2$).
- `min(A,2,2)` has answer A$\doteq$2 for ($c1$), and $2 \leq A$ for ($c2$)
- `min(A,2,3)` fails

(In Prolog, these transitions would lead to error messages.)

Generate-and-test in LP: impraticabili, fatti usati solo in modo passivo

Constraint-and-generate in CLP: usa i fatti in modo attivo per ridurre lo spazio di ricerca (vincoli)

Combinazione di ricerca con la risoluzione di vincoli particolarmente utile

- **Linguaggi LP:** dichiarativi, per predicati arbitrari, non deterministici
- **Risolutori di vincoli:** dichiarativi, efficienti per predicati speciali, deterministici

Esempio LP Generate/Test: Send More Money (puzzle cripto-aritmetico)

Sostituisci lettere distinte con cifre distinte, i numeri non hanno zeri iniziali

```
    S E N D
+   M O R E
= M O N E Y
```

```
[S,E,N,D,M,O,R,Y] = [9,5,6,7,1,0,8,2]

    S E N D
    9 5 6 7
    M O R E
+   1 0 8 5
  M O N E Y
= 1 0 6 5 2
```

Esempio CLP: Send More Money (Constraint/Generate)

```
:- use_module(library(clpfd)).

send([S,E,N,D,M,O,R,Y]) :-
     gen_domains([S,E,N,D,M,O,R,Y],0..9),
     S #\= 0, M #\= 0,
     all_distinct([S,E,N,D,M,O,R,Y]),
                  1000*S + 100*E + 10*N + D
     +            1000*M + 100*O + 10*R + E
     #= 10000*M + 1000*O + 100*N + 10*E + Y,
     labeling([],[S,E,N,D,M,O,R,Y]).

gen_domains([],_).
gen_domains([H|T],D) :- H in D, gen_domains(T,D).
```

send **Without labeling**

```
:- send([9,4,N,D,M,O,R,Y]).
no
```

Propagation determines N = 5, R = 8, but fails as D has no possible value. But

```
:- send([9,5,N,D,M,O,R,Y]).
D = 7, M = 1, N = 6,
O = 0, R = 8, Y = 2
yes
```

already computes solution.

```
    S E N D
+   M O R E
= M O N E Y
```

2

send **Without labeling**

```
:- send([S,E,N,D,M,O,R,Y]).
M = 1, O = 0, S = 9,
E in 4..7,
N in 5..8,
D in 2..8,
R in 2..8,
Y in 2..8 ?
```

```
    S E N D
+   M O R E
= M O N E Y
```

3

4

```
send([S,E,N,D,M,O,R,Y]) :-
     gen_domains([S,E,N,D,M,O,R,Y],0..9),
     labeling([],[S,E,N,D,M,O,R,Y]),
     S #\= 0, M #\= 0,
     all_distinct([S,E,N,D,M,O,R,Y]),
                  1000*S + 100*E + 10*N + D
     +            1000*M + 100*O + 10*R + E
     #= 10000*M + 1000*O + 100*N + 10*E + Y.
```

95,671,082 choices to find the solution

Case puzzle logico: il folklore attribuisce questo puzzle a Einstein

Cinque case colorate in fila, ognuna con un colore diverso, un proprietario con nazionalità diversa, un animale domestico diverso, sigarette di marca diversa e un drink diverso.

Case puzzle logico: vincoli

1. L'inglese abita nella casa rossa.
2. Lo spagnolo ha un cane.
3. Bevono caffè nella serra.
4. L'ucraino beve il tè.
5. La casa verde è accanto alla casa bianca.
6. Il fumatore di Winston ha un serpente.
7. Nella casa gialla fumano Kool.
8. Nella casa di mezzo bevono il latte.
9. Il norvegese abita nella prima casa da sinistra.
10. Il fumatore di Chesterfield vive vicino all'uomo con la volpe.
11. Nella casa vicino alla casa con il cavallo fumano Kool. Il fumatore Lucky Strike beve il succo. Il giapponese fuma il Kent.
12. Il norvegese abita vicino alla casa blu.

Chi possiede la zebra e chi beve l'acqua?

Una soluzione Prolog

```
houses(Hs) :-
% each house in the list Hs of houses is represented as:
%  h(Nationality, Pet, Cigarette, Drink, Color)}
       length(Hs, 5), % 1
       member(h(english,_,_,_,red), Hs),  % 2
       member(h(spanish,dog,_,_,_), Hs), % 3
       member(h(_,_,_,coffee,green), Hs), % 4
       member(h(ukrainian,_,_,tea,_), Hs), % 5
       next(h(_,_,_,_,green), h(_,_,_,_,white), Hs), % 6
       member(h(_,snake,winston,_,_), Hs),  % 7
       member(h(_,_,kool,_,yellow), Hs),  % 8
       Hs = [_,_,h(_,_,_,milk,_),_,_],  % 9
       Hs = [h(norwegian,_,_,_,_)|_],  %10
       next(h(_,fox,_,_,_), h(_,_,chesterfield,_,_), Hs), %
       next(h(_,_,kool,_,_), h(_,horse,_,_,_), Hs),  %12
       member(h(_,_,lucky,juice,_), Hs),  %13
       member(h(japanese,_,kent,_,_), Hs),  %14
       next(h(norwegian,_,_,_,_), h(_,_,_,_,blue), Hs), %15
       member(h(_,_,_,water,_), Hs), %one of them drinks
                                          %water
       member(h(_,zebra,_,_,_), Hs). %one of them owns
                                          % a zebra
```

```
zebra_owner(Owner) :-
         houses(Hs),
         member(h(Owner,zebra,_,_,_), Hs).

water_drinker(Drinker) :-
         houses(Hs),
         member(h(Drinker,_,_,water,_), Hs).

next(A, B, Ls) :- append(_, [A,B|_], Ls).
next(A, B, Ls) :- append(_, [B,A|_], Ls).
```

Examples of goals (queries:)

```
?- zebra_owner(Owner).
?- water_drinker(Drinker).
?- houses(Houses).
```

Una soluzione CLP(FD)

1. 25 Variabili:
   - Nazionalità: inglese, spagnola, giapponese, italiana, norvegese,
   - Animale domestico: cane, lumache, volpe, cavallo, zebra,
   - Professione: pittore, scultore, diplomatico, violinista, medico,
   - Drink: the, caffè, latte, succhi, acqua,
   - Colore della casa: rosso, verde, bianco, giallo, blu.
2. Dominio: [1...5].
3. Vincoli:
   alldifferent(red, green, white, yellow, blue),
   alldifferent(english, spaniard, japanese, italian, norwegian),
   alldifferent(dog, snails, fox, horse, zebra),
   alldifferent(painter, sculptor, diplomat, violinist, doctor),
   alldifferent(tea, coffee, milk, juice, water).

```
% The Englishman lives in the red house:
english = red,
% The Spaniard has a dog:
spaniard = dog,
% The Japanese is a painter:
japanese = painter,
% The Italian drinks tea:
italian = tea,
% The Norwegian lives in the first house on
the left:
norwegian = 1,
& The owner of the green house drinks coffee:
green = coffee,
% The green house is on the right of the white
house:
green = white + 1,
% The sculptor breeds snails:
sculptor = snails,
%  The diplomat lives in the yellow house:
diplomat = yellow,
%  They drink milk in the middle house:
milk = 3,
%  The Norwegian lives next door to the blue house:
| norwegian ? blue | = 1,
% The violinist drinks fruit juice:
violinist = juice,
% The fox is in the house next to the doctor's:
|fox ? doctor| = 1,
% The horse is in the house next to the diplomat's:
| horse ? diplomat | = 1.
```

Esercizio: completa il programma utilizzando questi vincoli.

Esempio di mutuo

- P: importo del prestito, debito, capitale
- T: Durata del prestito in mesi
- I: Tasso di interesse al mese
- B: Saldo del debito dopo T mesi
- R: Tasso dei pagamenti al mese

```
mg(P,T,I,B,R):-
    { T = 1,
      R + B = P * (1 + I)
    }.
mg(P,T,I,B,R):-
    { T > 1,
      P1 = P * (1 + I) - B,
      T1 = T - 1
    },
    mg(P1, T1, I, B, R).
```

Possiamo usare il programma precedente per calcolare il saldo se otteniamo un interesse del 5% su un periodo di 30 anni:

- ?- mg(1000, 30, 5/100, B, 0)

Possiamo però anche calcolare la relazione lineare tra il saldo iniziale, il saldo finale e il prelievo:

- ?- mg(B0, 30, 5/100, B, MP)

Logica del Primo Ordine (FOL, First Order Logic)

- **Sintassi: lingua** (alfabeto, espressioni ben formate)
- **Semantica: significato** (interpretazione, conseguenza logica, modelli di marca)
- **Calcoli: derivazione** (regola di inferenza confutazione)
    - Forma normale della clausola per FOL
    - Sostituzioni
    - Unificazione

La logica proposizionale è priva di struttura: in logica proposizionale Q non è una conseguenza logica di P e S, ma vorremmo esprimere questa relazione

- P = Ogni uomo è mortale
- S = Socrate è un uomo
- Q = Socrate è mortale

Sintassi della logica del primo ordine

Firma Σ = (P, F) **di una lingua del primo ordine**

- P: insieme finito di simboli predicati, ciascuno con arità n ∈ N
- F: insieme finito di simboli di funzione, ciascuno con arità n ∈ N

Convenzioni di denominazione: nullary, unario, binario, ternario per arieta 0, 1, 2, 3

- **Costanti:** simboli di funzione nullary
- **Proposizioni:** simboli predicati nulli

Alfabeto

- P: simboli predicati: p, q, r, ...
- F: simboli di funzione: a, b, c, ..., f, g, h, ...
- V: insieme numerabile infinito di variabili: X, Y, Z, ...
- **Simboli logici:**
    - **Simboli di verità:** ⊥ (falso), T (vero)
    - **Connettivi logici:** ¬, ∧, ∨, →
    - **Quantori:** ∀, ∃
    - **Simboli sintattici:** “(”, “)”, “,”

Espressioni ben formate

Insieme di **termini** T(Σ, V): una variabile di V o un termine di funzione f($\bar{t}$), dove f è un simbolo di funzione n-ario da Σ e gli argomenti $\bar{t}$ sono termini (n ≥ 0).

- **Termini:** P = {mortale/1}, F = {socrate/0, padre/1}, V = {X, ...}
    - X, Socrate, padre(socrate),
    - padre(padre(socrate)),

- ma non: padre(X, socrate)

Formula ben formata

Insieme di **formule** F(Σ, V) = {A, B, C, ..., F, G, ...}: una **formula atomica** (atomo) p(t), dove p è un simbolo predicato n-ario da e gli argomenti t sono termini, o ⊥, o ⊤, o $t_1 = t_2$ per i termini t_1 e t_2 = (t_1, t_2), o la **negazione** ¬F di una formula F, o la **congiunzione** (F ∧ F'), la **disgiunzione** (F ∨ F'), o **l'implicazione** (F → F') tra due formule F e F', o una **formula quantificata universalmente** ∀XF, o una **formula quantificata esistenzialmente** ∃XF, dove X è una variabile e F è una formula.

- **Formule atomiche:**
  - mortale(X), mortale(socrate),
  - mortale(padre(socrate))
  - ma non: mortale(mortale(socrate))
- **Formule non atomiche:**
  - mortale(socrate) ∧ mortale(padre(socrate))
  - ∀X.mortal(X) → mortale(padre(X))
  - ∃X.X = socrate

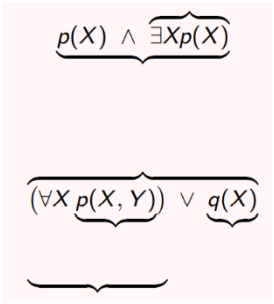
Variabili libere

$$
\begin{aligned}
Fv(t_1 \doteq t_2) &:= vars(t_1) \cup vars(t_2) \\
Fv(p(\overline{t})) &:= \cup vars(\overline{t}) \\
Fv(\top) := Fv(\bot) &:= \emptyset \\
Fv(\neg F) &:= Fv(F) \text{ for a formula } F \\
Fv(F * F') &:= Fv(F) \cup Fv(F') \text{ for formulae } F \text{ and } F' \\
&\qquad \text{and } * \in \{\wedge, \vee, \rightarrow\} \\
Fv(\forall XF) := Fv(\exists XF) &:= Fv(F) \setminus \{X\} \text{ for a variable } X \\
&\qquad \text{and a formula } F
\end{aligned}
$$

- la formula quantificata ∀XF o ∃XF lega la variabile X all'interno dell'ambito F
- imposta F_v(F) di variabili libere (non limitate) di una formula F

Esempio – Variabili libere

Fornisci l'insieme di variabili libere per ogni parte controventata.



$X_1, X_2, ..., X_n$ sono tutte variabili libere di F:

- **Chiusura universale** ∀F **di F:** $\forall X_1 \forall X_1 \ldots X_n$F
- **Chiusura esistenziale** ∃F **di F:** $\exists X_1 \exists X_2 \ldots X_n$F

Convenzioni di denominazione:

- Formula o frase chiusa: non contiene variabili libere
- Teoria: insieme di frasi
- Termine fondamentale o formula: non contiene alcuna variabile

Semantica di FOL: significato

Semantica della logica del primo ordine

Significato di una formula scritta in **FOL**: la presenza di variabili rende il compito più difficile rispetto al caso della Logica Proposizionale, id scriviamo X ≥ 1, la verità di questa formula dipende dal valore di X e dal significato che diamo al predicato ≥ e alla funzione (costante) 1

- Dobbiamo correggere:
  - Un universo, cioè un dominio da cui prendiamo i valori (numeri, grafici, mele, animali, ...)
  - Un'interpretazione dei simboli di funzione e predicato (sul dato Universo)
  - Una valutazione per variabili.

Interpretazione I di Σ

Consiste di:

- Un universo U, cioè un insieme non vuoto
- Una funzione I(f): $U^n \rightarrow$ U per ogni simbolo di funzione n-ario f di Σ
- Una relazione I(p) ⊆ U^n per ogni simbolo predicato n-ario p di Σ

Valutazione variabile per V w.r.t. I (Σ firma di un linguaggio del primo ordine, V insieme di variabili)

- η : V → U (per ogni **variabile** X di **V** nell'universo U di **I**)

Interpretazione dei termini

Data **firma Σ**, **I interpretazione** con l'**universo U**, **η** : V → U **variabile valutazione**, la funzione $\eta^I : T(\Sigma, V) \rightarrow U$ definita come segue prevede l'interoperabilità dei termini:

$\eta^I(X) := \eta(X)$ per una variabile X

$\eta^I(f(t_1, \ldots, t_n)) := I(f)(\eta^I(t_1), \ldots, \eta^I(t_n))$

per una funzione n-aria simbolo f e termini t_1, ..., t_n

- signature $\Sigma = (\emptyset, \{1/0, */2, +/2\})$
- universe $U = \mathbb{N}$
- interpretation I
  $I(1) := 1$
  $I(A+B) := A + B$, $I(A*B) := AB$ for $A, B \in U$
- variable valuation η
  $\eta(X) := 3$, $\eta(Y) := 5$

$$\begin{aligned}\eta^I(X*(Y+1)) &= I(*)(\eta^I(X), \eta^I(Y+1)) \\ &= I(*)(3, I(+)(\eta^I(Y), \eta^I(1)) \\ &= \cdots = 3 \cdot (5+1) = 18\end{aligned}$$

Interpretazione delle formule: preliminari

Per una funzione g, la funzione g[Y → a] è

$$g[Y \mapsto a](X) := \begin{cases} g(X) & \text{if } X \neq Y, \\ a & \text{if } X = Y, \end{cases}$$

dove X e Y sono variabili.

Data un'interpretazione di I di Σ, una valutazione variabile η, dobbiamo definire quando I e η rendono vera o soddisfano una formula F scritta I, η |= F.

Interpretazione delle formule

Data l'interpretazione I di Σ, η valutazione variabile, I, η |= F (I, η soddisfa F) è definita come segue:

$I, \eta \models \top$ and $I, \eta \not\models \bot$
$I, \eta \models s \doteq t$ iff $\eta^I(s) = \eta^I(t)$
$I, \eta \models p(t_1, \ldots, t_n)$ iff $(\eta^I(t_1), \ldots, \eta^I(t_n)) \in I(p)$
$I, \eta \models \neg F$ iff $I, \eta \not\models F$
$I, \eta \models F \wedge F'$ iff $I, \eta \models F$ and $I, \eta \models F'$
$I, \eta \models F \vee F'$ iff $I, \eta \models F$ or $I, \eta \models F'$
$I, \eta \models F \rightarrow F'$ iff $I, \eta \not\models F$ or $I, \eta \models F'$
$I, \eta \models \forall X F$ iff $I, \eta[X \mapsto u] \models F$ for all $u \in U$
$I, \eta \models \exists X F$ iff $I, \eta[X \mapsto u] \models F$ for some $u \in U$

Esempio: interpretazione

$\forall X.p(X,a,b) \rightarrow q(b,X)$

.	$I_1(.)$	$I_2(.)$
U	real things	natural numbers
a	"food"	5
b	"Fitz the cat"	10
p	"_ gives _ _"	_ + _ > _
q	"_ loves _"	_ < _

- l_1: "Fritz il gatto ama tutti quelli che gli danno da mangiare".
- l_2: "10 è minore di qualsiasi X se X + 5 > 10."
- (controesempio: X = 6)

Modello di F, Validità

Data una firma Σ, un'interpretazione I e una formula F:

- Diciamo che I modello di F o I soddisfa F, scritto I |= F quando: I, η |= F per ogni variabile di valutazione η
- I un modello della teoria T quando I è un modello di ogni formula in T

La frase S è

- **Valida:** soddisfatta da ogni interpretazione, cioè I |= S per ogni I
- **Soddisfacibile:** soddisfatta da qualche interpretazione, cioè I |= S per qualche I
- **Falsificabile:** non soddisfatta da qualche interpretazione, cioè I |≠ S per qualche I
- **Insoddisfacente:** non soddisfatto da alcuna interpretazione, cioè I |≠ S per ogni I

Esempio: interpretazione e modello

$$I(\texttt{pair}) := \left\{ \begin{array}{lll} (Mon, Tue), & (Mon, Wed), & \ldots, (Mon, Sun), \\ & (Tue, Wed), & \ldots, (Tue, Sun), \\ & \ddots & \vdots \\ & & (Sun, Sun) \end{array} \right\}$$

$I(\texttt{next}) : U \rightarrow U$, next day fu
$Mon \mapsto Tue, \ldots, Sun \mapsto Mon$

- Firma Σ = ({pair/2}, {next/1})
- Universo U = {lun, mar, mer, gio, ven, sab, dom}
- Interpretazione I

(one possible) valuation η
$\eta(X) := Sun$, $\eta(Y) := Tue$
- $\eta^I(\texttt{next}(X)) = Mon$
- $\eta^I(\texttt{next}(\texttt{next}(Y))) = Thu$
- $I, \eta \models \texttt{pair}(\texttt{next}(X), Y)$

model relationship ("for all variable valuations")
- $I \not\models \forall X \forall Y.\texttt{pair}(\texttt{next}(X), Y)$
- $I \models \forall X.\texttt{pair}(\texttt{next}(X), Sun)$
- $I \models \forall X \exists Y.\texttt{pair}(X, Y)$

Conseguenza logica

Un enunciato/teoria T_1 è una conseguenza logica di un enunciato/teoria T_2, scritto T_2 |= T_1, se ogni modello di T_2 è anche un modello di T_1, cioè I |= T_2 implica I |= T_1.

Due frasi o teorie sono equivalenti (⇔) se sono conseguenze logiche l'una dell'altra.

Teorema: è indecidibile se una formula logica del primo ordine F sia vera sotto tutte le possibili interpretazioni, cioè se |= F. [Church]

- Esempio: ¬(A ∧ B) ⇔ ¬A ∨ ¬B (de Morgan)

Esempio: leggi di tautologia

Le leggi duali valgono per ∧ e ∨ scambiate.

- A ⇔ ¬¬A (doppia negazione)
- ¬(A ∧ B) ⇔ ¬A ∨ ¬B (de Morgan)
- A ∧ A ⇔ A (idempotenza)
- A ∧ (A ∨ B) ⇔ A (assorbimento)
- A ∧ B ⇔ B ∧ A (commutatività)
- A ∧ (B ∧ C) ⇔ (A ∧ B) ∧ C (associatività)
- A ∧ (B ∨ C) ⇔ (A ∧ B) ∨ (A ∧ B) (distributività)
- A → B ⇔ ¬A ∨ B (implicazione)
- A → B ⇔ ¬B → ¬A (contrapposizione)
- (A → (B → C)) ⇔ (A ∧ B) → C
- ¬∀XA ⇔ ∃X¬A
- ¬∃XA ⇔ ∀X¬A
- ∀X(A ∧ B) ⇔ ∀XA ∧ ∀XB
- ∃X(A ∨ B) ⇔ ∃XA ∨ ∃XB
- ∀XB ⇔ B ⇔ ∃XB (con X non libero in B)

Esempio: conseguenza logica, validità, insoddisfazione

Le seguenti dichiarazioni sono equivalenti (in generale, F |≠ G non implica F |= ¬G):

$F_1, \ldots, F_k \models G$
(G is a logical consequence of $F_1, \ldots, F_k$)
$\left(\bigwedge_{i=1}^{k} F_i\right) \rightarrow G$ is valid.
$\left(\bigwedge_{i=1}^{k} F_i\right) \wedge \neg G$ is unsatisfiable.

Interpretazione di Herbrand

Motivazione: la formula A è valida in I, I |= A, se I, η |= A per ogni valutazione η. Ciò richiede di correggere un universo U poiché sia I che η usano U.

Jacques Herbrand scoprì che esiste un dominio universale insieme a un'interpretazione universale, s.t. che qualsiasi formula universalmente valida è valida in qualsiasi interpretazione. Pertanto, solo le interpretazioni nell'universo di Herbrand devono essere verificate, a condizione che l'universo di Herbrand sia infinito.

- **Universo di Herbrand:** insieme T(Σ, ∅) di termini ground
- Per ogni simbolo di funzione n-ario f di Σ, la funzione assegnata I(f) mappa una tupla ($\bar{t}$) di termini ground al termine ground "f($\bar{t}$)".
- **Base di Herbrand per la firma Σ:** insieme di atomi macinati in F(Σ, ∅), cioè {p($\bar{t}$) | p è un simbolo predicato n-ario di Σ e $\bar{t} \in$ T (Σ, ∅)}.
- **Modello di frase/teoria di Herbrand:** interpretazione di Herbrand che soddisfa frase/teoria.

Esempio: interpretazione di Herbrand

Per la formula F ≡ ∃X∃Y.p(X, a) ∧ ¬p(Y, a) l'universo di Herbrand è {a} e F è insoddisfacente nell'universo di Herbrand come p(a, a) ∧ ¬p(a, a) è falso, cioè non esiste un modello Herbrand.

Tuttavia, se aggiungiamo (un altro elemento) b abbiamo p(a, a) ∧ ¬p(b, a) quindi F è valido per qualsiasi interpretazione la cui cardinalità dell'universo sia maggiore di 1.

Per la formula ∀X∀Y.p(X, a) ∧ q(X, f (Y)) l'(infinito, poiché esiste una costante e un simbolo di funzione) Il dominio di Herbrand è {a, f(a), f(f(a)), f(f(f(a))), ...}.

Teorema di Herbrand (versione semplice): sia P un insieme di enunciati universali. Sono equivalenti:

1. P ha un modello Herbrand
2. P ha un modello
3. ground(P) è soddisfacibile

Dimostrazione del teorema di Herbrand

- (1) ⇒ (2) Ovvio
- (2) ⇒ (3) Ogni enunciato in ground(P) è una conseguenza logica di P. Quindi ogni modello di P è un modello di ground(P).
- (3) ⇒ (1) Se ground(P) è soddisfacibile allora ground(P) ha un modello di Herbrand **A**. Se fatto sia M un modello di ground(P). Allora possiamo definire **A** nel solito modo per i simboli di funzione, mentre per le formule atomiche possiamo definire **A** |= p(t_1, ..., t_n) se **M** |= p(t_1, ..., t_n) e quindi induttivamente per formule arbitrarie (dettagli per l'esercizio).
- Ora abbiamo che **A** è anche un modello di P. Infatti, supponiamo che A |= ground(∀∅) dove ∅ sia libero da quantificatore e Var(∅) = {x_1, ..., x_n}. Sia t_1, ..., t_n be n be n termini ground arbitrari e definiamo θ = {x_1/t_1, ..., x_n/t_n}. Poiché A |= ground(∀∅) si ha che A |= ∅θ. Poiché i termini ti sono elementi generici del dominio di **A** ciò significa che A |= ∀∅ e conclude la dimostrazione.

Calcolo per FOL: risoluzione

Calcolo

- **Assiomi:** date formule, tautologie elementari e contraddizioni non deducibili all'interno del calcolo
- **Regole di inferenza:** consentono di derivare nuove formule da formule date
- **Derivazione ∅ ⊢ ψ:** una sequenza di applicazioni di regole di inferenza che iniziano con la formula ∅ e terminano con la formula ψ

⊨ e ⊢ dovrebbero coincidere

- **Solidità:** ∅ ⊢ ρ implica ∅ ⊨ ρ
- **Completezza:** ∅ ⊨ ρ implica ∅ ⊢ ρ

Possiamo definire un calcolo corretto e completo per FOL? E possiamo usare la risoluzione?

Calcolo della risoluzione per FOL

L'idea è la stessa della logica proposizionale: la teoria unita alla conseguenza negata deve essere insoddisfacente. Tuttavia, dobbiamo prima trasformare le formule FOL in clausole, che è più complicato. Quindi dobbiamo definire la regola di inferenza della risoluzione per FOL: ciò richiede sostituzioni e unificazioni

Forme normali

Negazione Forma Normale della Formula F

F in forma normale negativa F_{neg}:

- nessuna sotto-formula della forma F → F'
- in ogni sotto-formula della forma ¬F' la formula F' è atomica

Forme normali e clausole per FOL

Negazione Forma Normale: calcolo

Per ogni enunciato F, esiste un enunciato equivalente F_{neg} in forma normale negata (applicare "tautologie" da sinistra a destra):

Negation

$$\neg\bot \Leftrightarrow \top \qquad \neg\top \Leftrightarrow \bot$$

$$\neg\neg F \Leftrightarrow F \quad F \text{ is atomic}$$

$$\neg(F \wedge F') \Leftrightarrow \neg F \vee \neg F' \qquad \neg(F \vee F') \Leftrightarrow \neg F \wedge \neg F'$$

$$\neg\forall XF \Leftrightarrow \exists X\neg F \qquad \neg\exists XF \Leftrightarrow \forall X\neg F \qquad \neg(F \rightarrow F') \Leftrightarrow F \wedge \neg F'$$

Implication

$$F \rightarrow F' \Leftrightarrow \neg F \vee F'$$

Skolemizzazione della Formula F

- F = F_{neg} ∈ F(Σ, V) in forma normale negativa
- occorrenza della sotto-formula ∃XG con variabili libere $\bar{V}$
- simbolo di funzione f/n non presente in Σ

Calcola F' sostituendo ∃XG con G[X → f ($\bar{V}$)] in F (tutte le occorrenze di X in G sono sostituite da f ($\bar{V}$)).

Convenzioni di denominazione:

- Forma skolemizzata di F: F'
- Funzione Skolem: f /n

Esempio: skolemization

$$\forall z \exists x \forall y\, (p(x,z) \wedge q(g(x,y),x,z)) \\ \Leftrightarrow \forall z \forall y\, (p(f(z),z) \wedge q(g(f(z),y),f(z),z))$$

Equivalenza della forma Skolemized

- Una formula F è soddisfacibile se la sua forma skolemizzata è soddisfacibile
- Una formula F e la sua forma skolemizzata non sono logicamente equivalenti

Consideriamo F = ∀X.∃Y.p(a, Y, a, b) e consideriamo un'interpretazione I dove f(a) = b (funzione f skolem) e solo p(a, a, a, b) vale.

I è un modello di F, ma non è un modello della forma skolemizzata di F

Prenex Forma di Formula F

F = $Q_1 X_1$ ... $Q_n X_n$G

- Quantificatori di Q_i
- Variabili X_i
- Formula G senza quantificatori
- Prefisso quantificatore: $Q_1 X_1$ ... $Q_n X_n$
- Matrice: G
- Per ogni enunciato F esiste un enunciato equivalente in forma prenex ed è possibile calcolare tale enunciato da F applicando leggi di tautologia per spingere i quantificatori verso l'esterno

Esempio: forma normale Prenex

$$\neg \exists x p(x) \vee \forall x r(x) \Leftrightarrow \forall x \neg p(x) \vee \forall x r(x) \\ \Leftrightarrow \forall x \neg p(x) \vee \forall y r(y) \\ \Leftrightarrow \forall x\, (\neg p(x) \vee \forall y r(y)) \\ \Leftrightarrow \forall x \forall y\, (\neg p(x) \vee r(y))$$

Clausole e letterali

- Letterale: atomo (letterale positivo) o negazione dell'atomo (letterale negativo)
- Letterali complementari: letterale positivo L e sua negazione ¬L
- Clausola (in forma normale disgiuntiva): formula della forma $\bigvee_{i=1}^{n} L_i$ dove L_i sono letterali.
- Proposizione vuota (disgiunzione vuota): n = 0: ⊥

Forma di implicazione della clausola:

$$F = \underbrace{\bigwedge_{j=1}^{n} B_j}_{body} \rightarrow \underbrace{\bigvee_{k=1}^{m} H_k}_{head}$$

per

$$F = \bigvee_{i=1}^{n+m} L_i \text{ with } L_i = \begin{cases} \neg B_i & \text{for } i = 1, \ldots, n \\ H_{i-n} & \text{for } i = n+1, \ldots, n+m \end{cases}$$

per atomi B_j e H_k

- Clausola chiusa: frase $\forall \bar{x}$C con clausola C
- Forma claustrale della teoria: consiste di clausole chiuse

Fasi di normalizzazione: una teoria arbitraria T può essere trasformata in forma claustrale come segue

1. Converti ogni formula della teoria in una formula equivalente in forma normale negativa.
2. Eseguire Skolemization per eliminare tutti i quantificatori esistenziali.
3. Converti la teoria risultante, che è ancora in forma normale negativa, in una teoria equivalente in forma claustrale: sposta verso l'esterno le congiunzioni ei quantificatori universali.

Sostituzioni e unificazione

Sostituzione

- Una sostituzione σ è una mappatura σ : V → T (Σ, V') che modifica un numero finito di variabili, scritte come $\{X_1 \rightarrow t_1, ..., X_n \rightarrow t_n\}$ dove X_i sono variabili distinte e t_i sono termini.
- La sostituzione di identità è indicata da $\epsilon = \emptyset$
- Spesso scritti come operatori suffissi, applicazione da sinistra a destra nella composizione
- Sui termini una sostituzione σ : T (Σ, V) → T (Σ, V') è l'estensione omomorfa implicita, cioè f($\bar{t}$)σ := f (t_1σ, ..., t_nσ).

Esempio: σ = {X → 2, Y → 5}: (X ∗ (Y + 1))σ = 2 ∗ (5 + 1)

Sostituzione applicata ad una Formula

Homomorphic extension

- $p(\bar{t})\sigma := p(t_1\sigma, \ldots, t_n\sigma)$
- $(s \doteq t)\sigma := (s\sigma) \doteq (t\sigma)$
- $\bot\sigma := \bot$ and $\top\sigma := \top$
- $(\neg F)\sigma := \neg(F\sigma)$
- $(F * F')\sigma := (F\sigma) * (F'\sigma)$ for $* \in \{\wedge, \vee, \rightarrow\}$

Except

- $(\forall X F)\sigma := \forall X'(F\sigma[X \mapsto X'])$
- $(\exists X F)\sigma := \exists X'(F\sigma[X \mapsto X'])$

where X' is a fresh variable.

Esempio: applicazione di sostituzione

Espressione logica su V: termine con variabili in V, formula con variabili libere in V, sostituzione da un insieme arbitrario di variabili in T (Σ, V), o tupla di espressioni logiche su V. Un'espressione logica è un'espressione semplice se non contiene quantificatori. Esempio: V = {X}: f (X), ∀Y.p(X) ∧ q(Y), {X → a}, $\langle p(X), \sigma \rangle$

Istanza, Rinomina variabile, Varianti

- e istanza di e': e = e'σ
- e' più generale di e: e è istanza di e'
- rinomina variabile per e: sostituzione σ
  - σ iniettivo

$\sigma = \{X \mapsto Y, Z \mapsto 5\}$: $(X * (Z + 1))\sigma = Y * (5 + 1)$

$\sigma = \{X \mapsto Y, Y \mapsto Z\}$: $p(X)\sigma = p(Y) \neq p(X)\sigma\sigma = p(Z)$

$\sigma = \{X \mapsto Y\}, \tau = \{Y \mapsto 2\}$
- $(X * (Y + 1))\sigma\tau = (Y * (Y + 1))\tau = (2 * (2 + 1))$
- $(X * (Y + 1))\tau\sigma = (X * (2 + 1))\sigma = (Y * (2 + 1))$

$\sigma = \{X \mapsto Y\}$: $(\forall X p(3))\sigma = \forall X' p(3)$

$\sigma = \{X \mapsto Y\}$: $(\forall X p(X))\sigma = \forall X' p(X')$, $(\forall X p(Y))\sigma = \forall X' p(Y)$

$\sigma = \{Y \mapsto X\}$: $(\forall X p(X))\sigma = \forall X' p(X')$, $(\forall X p(Y))\sigma = \forall X' p(X)$

  - Xσ ∈ V per tutti X ∈ V
  - Xσ non si verifica in e per variabili libere X di e
  - varianti e ed e' (identica ridenominazione variabile modulo): e = e'σ ed e' = eτ

Esempio (e, e' espressioni logiche, σ, τ sostituzioni): rinomina variabile (∀Xp(X_1)){$X_1 \to X_2$} = ∀X'p(X_2) ma non (∀Xp(X_1) ∧ q(X_2)){ $X_1 \to X_2$} = ∀X'.p(X_2) ∧ q(X_2)

Unificatore e MGU (e_1, ..., e_n espressioni semplici, sostituzioni σ, τ, ρ, σ_i)

- σ è un unificatore per e_1, ..., e_n se $e_1\sigma = \cdots = e_n\sigma$
- e_1, ..., e_n unificabile se esiste unificatore
- σ è un **Most General Unifier** (MGU) per e_1, ..., e_n se ogni unificatore τ per $\bar{e}$ è istanza di σ, cioè, τ = σρ per qualche ρ

Esempio: unificatore più generale

$$f(X, a) \doteq f(g(U), Y) \doteq Z$$

MGU:

$$\sigma = \{X \mapsto g(U), Y \mapsto a, Z \mapsto f(g(U), a)\}$$

Proof: $f(X, a)\sigma = f(g(U), Y)\sigma = Z\sigma = f(g(U), a)$ one element.

Unifier, but not MGU:

$$\sigma' = \{X \mapsto g(h(b)), U \mapsto h(b), Y \mapsto a, Z \mapsto f(g(h(b)), a)\}$$

Proof: $\sigma' = \sigma\{U \mapsto h(b)\}$.

Unificatore più generale a mano

1. variabile svincolata: non c'è sostituzione per essa
2. Inizia con ϵ
3. scansiona i termini contemporaneamente da sinistra a destra in base alla loro struttura
4. controlla l'equivalenza sintattica dei simboli incontrati ripeti
   - diversi simboli di funzione: arresto con guasto
   - identico: continua

- una è variabile non legata e l'altro termine:
  - variabile si verifica in un altro termine: arresto con guasto
  - applica la nuova sostituzione alle espressioni logiche

  aggiungi la sostituzione corrispondente
- variabile non è svincolata: sostituiscila applicando la sostituzione

Esempio – Unificatore più generale

s	t	
f	g	failure
a	a	ϵ
X	a	$\{X \mapsto a\}$
X	Y	$\{X \mapsto Y\}$, but also $\{Y \mapsto X\}$
$f(a, X)$	$f(Y, b)$	$\{Y \mapsto a, X \mapsto b\}$
$f(g(a, X), Y)$	$f(c, X)$	failure
$f(g(a, X), h(c))$	$f(g(a, b), Y)$	$\{X \mapsto b, Y \mapsto h(c)\}$
$f(g(a, X), h(Y))$	$f(g(a, b), Y)$	failure

to unify	current substitution, remarks
$p(X, f(a)) \doteq p(a, f(X))$	ϵ, start
$X \doteq a$	$\{X \mapsto a\}$, substitution added
$f(a) \doteq f(X)$	continue
$a \doteq X$	$\{X \mapsto a\}$, variable is not unbound
$a \doteq a$	continue
MGU is $\{X \mapsto a\}$	
What about $p(X, f(b)) = p(a, f(X))$?	

Regola di inferenza della risoluzione per FOL

Calcolo della risoluzione: regole di inferenza

Opere per assurdo: la teoria unita alla conseguenza negata deve essere insoddisfacente (“deriva clausola vuota”).

R ∨ A e R' ∨ ¬A' devono avere variabili diverse: rinominare le variabili a parte

Axiom
empty clause (i.e. the elementary contradiction)
Resolution

$$\frac{R \vee A \qquad R' \vee \neg A'}{(R \vee R')\sigma}$$

σ is a most general unifier for the atoms A and A'

Factoring

$$\frac{R \vee L \vee L'}{(R \vee L)\sigma}$$

σ is a most general unifier for the literals L and L'

Risoluzione: osservazioni

Regola di **risoluzione**:

$$\frac{p(a, X) \vee q(X) \qquad \neg p(a, b) \vee r(X)}{(q(X) \vee r(X))\{X \mapsto b\}}$$

- due clausole C e C' istanziate s.t. letterale da C e letterale da C' complementare
- due clausole istanziate sono combinate in una nuova clausola
- risolvente aggiunto

Regola di **factoring**:

$$\frac{p(X) \vee p(b)}{p(X)\{X \mapsto b\}}$$

- clausola C istanziata, s.t. due letterali diventano uguali
- rimuovere un letterale
- fattore aggiunto

Confutazione completezza della risoluzione

Teorema: assumiamo che ⊢ denoti la risoluzione e che F sia un insieme di clausole. Allora F ⊢ ⊥ se F ⊨ ⊥

Completezza di FOL (nella dimostrazione originale ⊢ si riferisce al sistema di dimostrazione di Hilbert-Ackermann)

- **Teorema di completezza di Gödel:** se ⊨ F allora ⊢ F
- **Forma più generale:** Γ ⊨ F IFF Γ ⊢ F

Indecidibilità di FOL

Teorema: è indecidibile se una formula logica del primo ordine è dimostrabile (o vera sotto tutte le possibili interpretazioni).

Aritmetica FO (First-Order): l'aritmetica del primo ordine è un linguaggio di termini e formule. I termini o polinomi (positivi) sono costruiti dalle variabili X, Y, Z, ..., le costanti 0 e 1 e gli operatori + e x di addizione e moltiplicazione. L'operatore di moltiplicazione è normalmente soppresso per iscritto. Le formule più semplici sono le equazioni, ottenute scrivendo un = tra due termini, ad esempio X + Y^2 = 2Z^3 che è l'abbreviazione di X + YY = (1 + 1)ZZZ. Formule più complicate possono essere costruite da equazioni utilizzando i soliti connettivi e quantificatori FOL. L'aritmetica è interpretata in termini di numeri naturali. Ogni formula è vera o falsa (se ci sono variabili libere una formula è considerata equivalente alla sua chiusura universale).

Teorema: è **indecidibile** se una formula aritmetica è vera

Teorema: l'insieme delle vere formule aritmetiche non è nemmeno semi-decidibile.

Teorema di incompletezza di Gödel: se un sistema di dimostrazione per l'aritmetica è valido (solo le formule vere sono dimostrabili), allora deve esserci una formula vera che non è dimostrabile.

Machine Learning

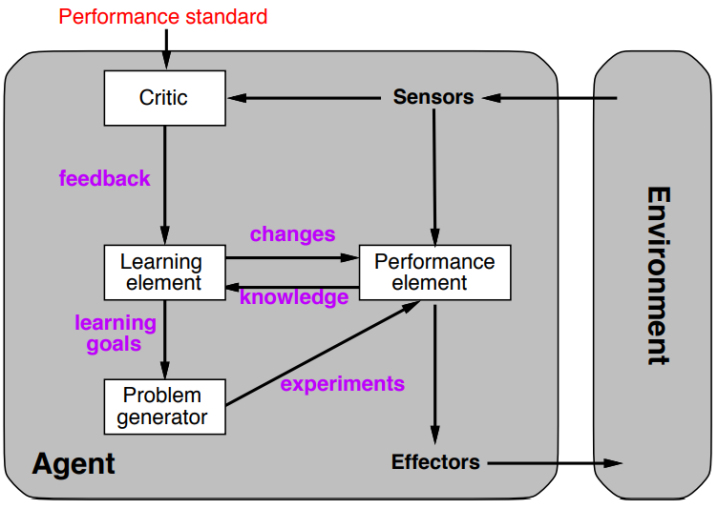
Apprendimento

- L'apprendimento è essenziale per ambienti sconosciuti, cioè quando il designer manca di onniscienza
- L'apprendimento è utile come metodo di costruzione del sistema, ovvero esporre l'agente alla realtà piuttosto che cercare di scriverlo
- L'apprendimento modifica i meccanismi decisionali dell'agente per migliorare le prestazioni

Apprendimento automatico: migliorare le prestazioni dopo aver osservato il mondo. Tre principali applicazioni in AI:

1. **Estrazione dei dati:** decisioni che utilizzano dati storici per migliorare le decisioni (es. cartelle cliniche -> conoscenze mediche), dimensione della sfida attuale dei dati (grandi) da Web, sensori (IoT), Biologia …
2. **Applicazioni software che non possiamo programmare a mano:** guida autonoma, riconoscimento vocale, accensione, PNL, visione artificiale, applicazioni Fintech, ...
3. **Programmi auto personalizzabili:** impara i comportamenti degli utenti per fornire servizi migliori: Newsreader NeUlix, Amazon ...

Agenti di apprendimento



Elemento di apprendimento

Il design dell'elemento di apprendimento è dettato da

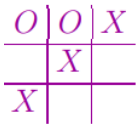
- che tipo di elemento di prestazione viene utilizzato
- quale componente funzionale deve essere appreso e come si rappresenta
- che tipo di feedback è disponibile

Scenari di esempio:

Elemento di prestazione	Componente	Rappresentazione	Feedback
Ricerca alfa−beta	Eval. fn.	Funzione lineare ponderata	Vittoria/Sconfitta
Agente logico	Modello di transizione	Assiomi dello stato successore	Risultato
Agente basato sull'utilità	Modello di transizione	Rete di Bayes dinameica	Risultato
Agente riflesso semplice	Percetto-azione fn	Rete neurale	Azione corretta

Apprendimento induttivo (alias Scienza)

Forma più semplice: imparare una funzione dagli esempi (**tabula rasa**)



f è la **funzione target**

Un **esempio** è una coppia **x, f(x)**

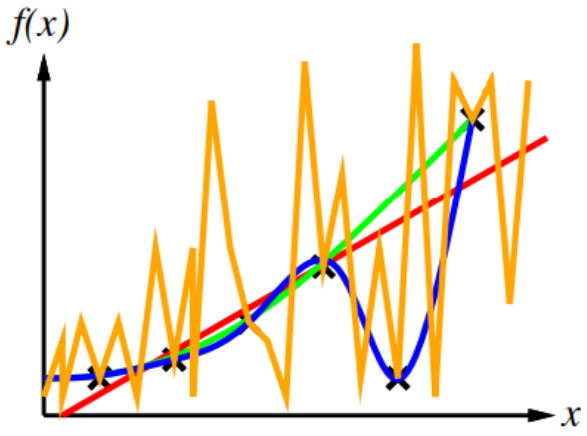
Problema: trovare una(n) **ipotesi h** tale che **h ≈ f** data una **serie** di esempi di **addestramento**

Questo è un modello altamente semplificato di apprendimento reale:

- Ignora le conoscenze pregresse
- Assume un "ambiente" deterministico e osservabile
- Presuppone che vengano forniti degli esempi
- Presuppone che l'agente voglia imparare f—perché?

Metodo di apprendimento induttivo

Costruisci/aggiusta **h** per concordare con **f** sul training set (**h** è **coerente** se concorda con **f** su tutti gli esempi). Ad esempio, adattamento della curva:

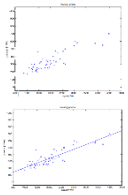


Il rasoio di Ockham: massimizza una combinazione di consistenza e semplicità

Machine Learning: tante tecniche diverse

Apprendimento supervisionato (risposte corrette per ogni istanza)**:** l'agente osserva un training set contenente un insieme di coppie (I, O) e apprende una funzione sulla mappatura Input to Output. Input fornito in termini di **caratteristiche** che rappresentano i dati. Dati **etichettati**. Esempio: prezzo di una casa in una zona

- **Metodi di apprendimento supervisionato**
  - **Classificazione:** l'uscita ha valori discreti
  - **Regressione lineare:** l'output ha un valore continuo



    - Dati alcuni dati di I/O (prezzi della casa), prova a imparare la funzione di ipotesi h: "$h_0(x) = \theta_0 + \theta_1 x$" riducendo al minimo la funzione di costo: "$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_0(x^{(i)}) - y^{(i)})^2$", usando la discesa del gradiente si ottiene
    - Osserviamo sia un insieme di caratteristiche X1, X2, ..., Xp per ogni oggetto, sia una variabile di risposta o risultato Y. L'obiettivo è quindi prevedere Y utilizzando X1, X2, ..., Xp.

Apprendimento non supervisionato: l'agente apprende il modello nell'input senza alcun feedback dall'output. Esempio: notizie di Google

- Osserviamo solo le caratteristiche X1, X2, ..., Xp. Non siamo interessati alla previsione, perché non abbiamo una variabile di risposta associata Y.
- **Obiettivi:** scoprire cose interessanti sulle misurazioni. Esiste un modo informativo per visualizzare i dati? Possiamo scoprire sottogruppi tra le variabili o tra le osservazioni? Due metodi principali:
  1. **Analisi dei componenti principali:** strumento utilizzato per la visualizzazione o la preelaborazione dei dati prima dell'applicazione delle tecniche supervisionate. Produce una rappresentazione a bassa dimensione di un set di dati. Trova una sequenza di combinazioni lineari delle variabili che hanno varianza massima e sono reciprocamente non correlate
  2. **Clustering:** un'ampia classe di metodi per scoprire sottogruppi sconosciuti nei dati. Cerchiamo una parità dei dati in gruppi distinti in modo che le osservazioni all'interno di ciascun gruppo siano abbastanza simili tra loro
     a. **K-means clustering**: parliamo dei dati in k cluster.
     b. **Clustering gerarchico**: non sappiamo quanti cluster vogliamo.
- **Vantaggi dell'apprendimento senza supervisione:** le tecniche per l'apprendimento non supervisionato sono utilizzate in molti campi (sottogruppi di pazienti con cancro al seno raggruppati in base alle misurazioni dell'espressione genica, gruppi di acquirenti caratterizzati dalla loro cronologia di navigazione e di acquisto, film raggruppati in base ai livelli assegnati dagli spettatori dei film) ed è spesso più facile ottenere dati non etichettati, da uno strumento di laboratorio o da un computer, piuttosto che dati etichettati, che possono richiedere l'intervento umano.

Apprendimento rafforzativo (ricompense occasionali)**:** l'agente impara dalle ricompense o dalle punizioni.

Apprendimento semi-supervisionato: alcuni esempi etichettati noti in un insieme più ampio di dati non etichettati. Esempio: riconoscimento dell'età delle persone dalle immagini.

Indurre l'apprendimento nella logica: trovare un'ipotesi, espressa in termini logici, che classifichi bene gli esempi e generalizzi bene a nuovi esempi. Processo di eliminazione graduale delle ipotesi incoerenti con gli esempi, restringendo le possibilità.

- **Programmazione logica induttiva**
  - scoperta automatizzata di nuove regole per il ripiegamento proteico
  - nuove conoscenze sulla genomica funzionale del lievito (robot che esegue esperimenti biologici)

- estrarre informazioni complesse dal testo

Rappresentazioni basate sugli attributi

Esempi descritti dai **valori degli attributi** (booleano, discreto, continuo, ecc.)

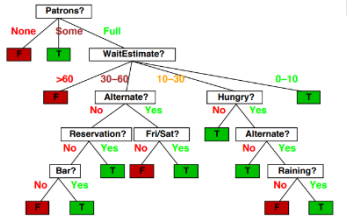
Ad esempio, situazioni in cui aspetterò o non aspetterò un tavolo:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$$$	F	T	Burger	30–60	T

La **classificazione** degli esempi è **positiva** (T) o **negativa** (F)

Decision Tree (DT)

Una possibile rappresentazione per ipotesi. Ad esempio, ecco l'albero "vero" per decidere se aspettare:



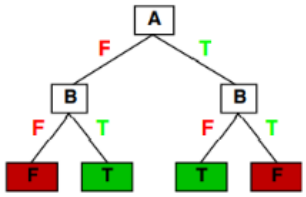
Un DT rappresenta una funzione che prende come input un vettore di valori di attributo e restituisce una “decisione”

Ogni nodo interno in un DT corrisponde a un tes su uno degli attributi

Espressività

Gli alberi decisionali possono esprimere qualsiasi funzione degli attributi di input. Ad esempio, per le funzioni booleane, riga della tabella di verità → percorso alla foglia:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Banalmente, esiste un albero decisionale coerente per qualsiasi set di allenamento con un percorso da sfogliare per ogni esempio (a meno che **f** non deterministico in **x**) ma probabilmente non generalizzerà a nuovi esempi. Preferisco trovare alberi decisionali più **compatti**

Spazi di ipotesi

Quanti alberi decisionali distinti con n attributi booleani? Ad esempio, con 6 attributi booleani, ci sono 18.446.744.073.709.551.616 alberi

- = numero di funzioni booleane
- = numero di tabelle di verità distinte con 2^n righe = 2^{2^n}

Quante ipotesi puramente congiuntive (es. Hungry ∧ ¬Rain)? Ogni attributo può essere in (positivo), in (negativo) o out ⇒ 3^n ipotesi congiuntive distinte

Spazio ipotesi più espressivo: aumenta la possibilità che la funzione target possa essere espressa e aumenta il numero di ipotesi coerenti con il set di allenamento (⇒ può peggiorare le previsioni)

Apprendimento dell'albero decisionale

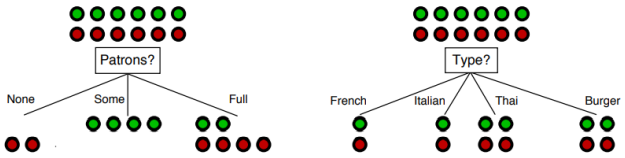
Obiettivo: trovare un piccolo albero coerente con gli esempi di formazione

Idea: scegli ricorsivamente l'attributo "più significativo" come radice del sottoalbero

```
function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
      best ← CHOOSE-ATTRIBUTE(attributes, examples)
      tree ← a new decision tree with root test best
      for each value v_i of best do
          examples_i ← {elements of examples with best = v_i}
          subtree ← DTL(examples_i, attributes − best, MODE(examples))
          add a branch to tree with label v_i and subtree subtree
      return tree
```

Scegliere un attributo

Idea: un buon attributo divide gli esempi in sottoinsiemi che sono idealmente "tutti positivi" o "tutti negativi"



Patroni? è una scelta migliore: fornisce **informazioni** sulla classificazione

Informazioni

Le informazioni rispondono alle domande. Più sono all'oscuro della risposta inizialmente, più informazioni sono contenute nella risposta

- Scala: 1 bit = risposta a domanda booleana con precedente **⟨0,5, 0,5⟩**

Informazioni in una risposta quando prima è $\langle P_1, ..., P_n \rangle$ è $H(\langle P_1, ..., P_n \rangle) = \sum_{i=1}^{n} -P_i \log_2 P_i$ (detta anche **entropia** del priore)

Supponiamo di avere **p** esempi positivi e **n** negativi alla radice ⇒ **H(⟨p/(p+n), n/(p+n)⟩)** bit necessari per classificare un nuovo esempio

Ad esempio, per 12 esempi di ristoranti, **p = n = 6** quindi abbiamo bisogno di 1 bit

Un attributo suddivide gli esempi **E** in sottoinsiemi E_i, ognuno dei quali (speriamo) necessita di meno informazioni per completare la classificazione

Lascia che E_i abbia p_i positivi e n_i negativi esempi

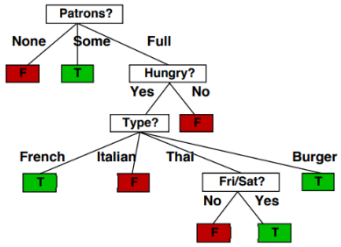
⇒ **H(⟨p_i/(p_i+n_i), n_i/(p_i+n_i)⟩)** bit necessari per classificare un nuovo esempio

⇒ il numero **previsto** di bit, per esempio, su tutti i rami è $\sum_i \frac{p_i+n_i}{p+n} H(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \rangle)$

Per **Patrons?**, questo è 0.459 bit, per **Type** questo è (ancora) 1 bit

⇒ scegli l'attributo che riduce al minimo le informazioni rimanenti necessarie

Albero decisionale appreso dai 12 esempi:



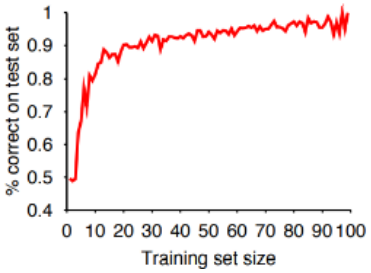
Sostanzialmente più semplice dell'albero "vero": un'ipotesi più complessa non è giustificata da una piccola quantità di dati

Valutazione della prestazione

Come facciamo a sapere che **h ≈ f**? (Il **problema dell'induzione** di Hume)

1) Usa i teoremi della teoria dell'apprendimento computazionale/statistica

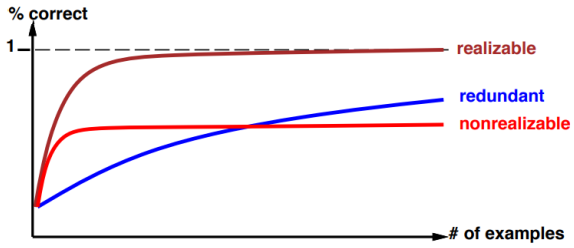
2) Prova **h** su un nuovo **set di test** di esempi (usa la **stessa distribuzione sullo spazio di esempio** come set di addestramento)



Curva di apprendimento = % corretta sul set di prova in funzione della dimensione del set di allenamento

La curva di apprendimento dipende da

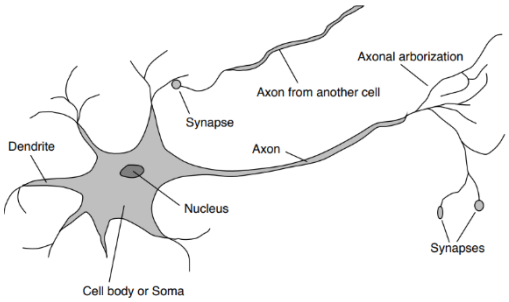
- **Realizzabile:** può esprimere la funzione target)
- **Non realizzabile:** può essere dovuta ad attributi mancanti o classi di ipotesi ristrette (ad esempio, funzione lineare con soglia)
- **Espressività ridondante** (ad esempio, carichi di attributi irrilevanti)



Reti neurali

Cervello

- 10^{11} neuroni di **> 20** tipi, 10^{14} sinapsi, tempo di ciclo 1ms–10ms
- I **segnali sono rumorosi "treni di picchi" di potenziale elettrico**



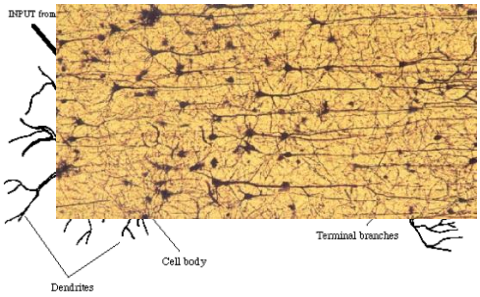
Reti neurali

Ispirato al modello del cervello. Una rete con:

- Molte unità computazionali (neuroni) con bassa potenza di calcolo
- Molte connessioni (ponderate)
- Controllo distribuito altamente parallelo

Approccio completamente diverso da quello simbolico:

- Conoscenza non esplicita
- Conoscenza incorporata nella struttura della rete e nei pesi delle connessioni.

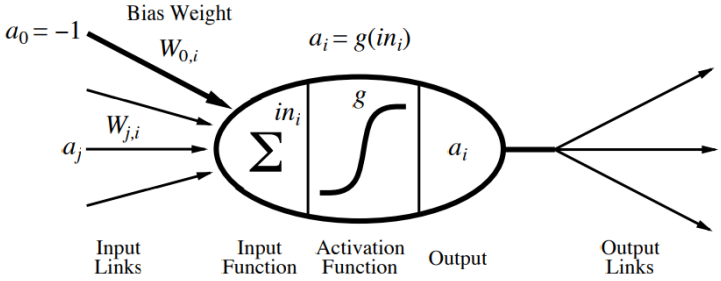


Capacità di apprendimento

- Imparare dalle differenze tra obiettivo noto e fatti osservati (I/O)
- Apprendimento delle funzioni matematiche associate al calcolo dei nodi

McCulloch-Pitts "unità"

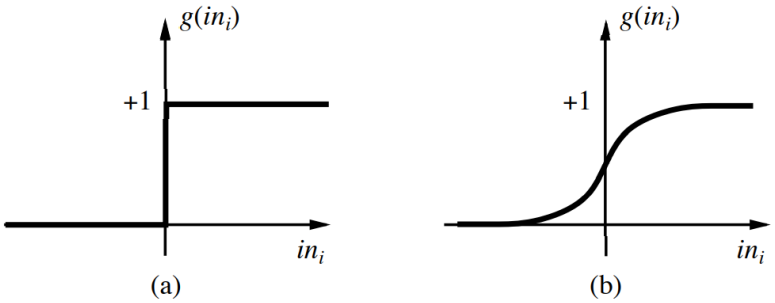
L'uscita è una funzione lineare "schiacciata" degli ingressi: $\boldsymbol{a_i \leftarrow g(in_i) = g(\sum_j W_{j,i} a_i)}$



Una grossolana semplificazione dei neuroni reali, ma il suo scopo è sviluppare la comprensione di cosa possono fare le reti di unità semplici

Funzioni di attivazione: modificando il peso di polarizzazione $\boldsymbol{W_{0,i}}$ sposta la posizione della soglia

- (a) è una **funzione a gradino** o una **funzione di soglia**



- (b) è una funzione **sigmoide 1/(1 + $\boldsymbol{e^{-x}}$)**

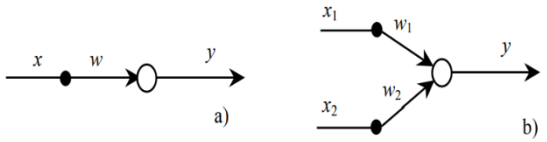
Un esempio: rappresentazione di funzioni booleane

Consideriamo le funzioni booleane elementari NOT, OR, AND, XOR.

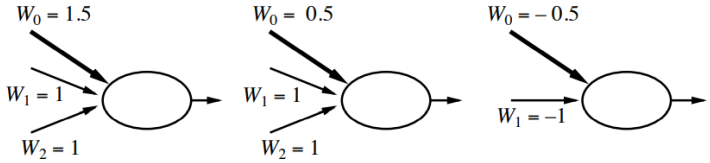
Elementary Boolean functions

x_1	x_2	$\overline{x_1}$	$x_1 + x_2$	$x_1 \cdot x_2$	$x_1 \oplus x_2$
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	1	0	1	1	0

La domanda fondamentale è se un neurone è sufficiente per rappresentare ciascuna di queste funzioni booleane.



Implementazione di funzioni logiche



McCulloch e Pitts: ogni funzione booleana può essere implementata

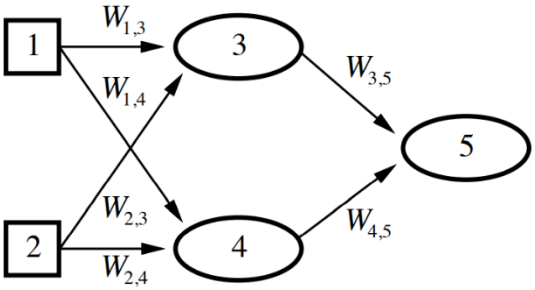
Strutture di rete

Reti feed-forward: percettroni a strato singolo o multistrato. Le reti feed-forward implementano funzioni, non hanno uno stato interno

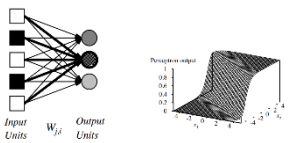
Reti ricorrenti:

- Le **reti Hopfield** hanno pesi simmetrici **($W_{i,j} = W_{j,i}$): g(x) = sign(x)**, **a_i = ± 1**; **memoria associativa olografica**
- Le **macchine Boltzmann** utilizzano funzioni di attivazione stocastica: ≈ MCMC nelle reti di Bayes
- Le reti neurali ricorrenti hanno cicli diretti con ritardi: avere uno stato interno (come gli infradito), può oscillare ecc.

Esempio di feed-forward network = una famiglia parametrizzata di funzioni non lineari: $a_5 = g(W_{3,5} * a_3 + W_{4,5} * a_4) = g(W_{3,5} * g(W_{1,3} * a_1 + W_{2,3} * a_2) + W_{4,5} * g(W_{1,4} * a_1 + W_{2,4} * a_2))$. La regolazione dei pesi cambia la funzione: impara così!



Perceptron a strato singolo

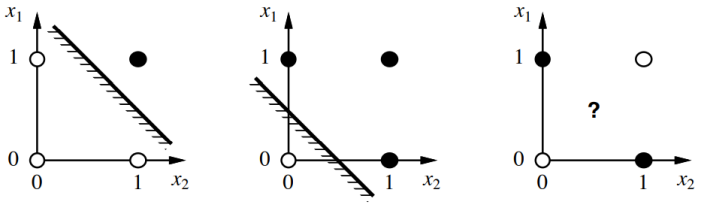


Le unità di output funzionano tutte separatamente, nessun peso condiviso. La regolazione dei pesi sposta la posizione, l'orientamento e la pendenza della scogliera

Espressività dei perceptron

- Consideriamo un perceptron con **g** = funzione a gradino (Rosenblatt, 1960)
- Può rappresentare AND, OR, NOT, maggioranza, ecc., ma non XOR

- Rappresenta un **separatore lineare** nello spazio di input: $\sum_j \boldsymbol{W_j x_i > 0}$ **o** $\boldsymbol{W * x > 0}$
- Minsky e Papert (1969) hanno perforato il palloncino della rete neurale



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Percezione Regola di apprendimento

h_w(x) = soglia (W * x) (o equivalente = soglia ($\sum_i W_i x_i$)

Errore = y - h_w(x)

Apprendimento della percezione: aggiorna i pesi per ridurre al minimo l'errore.

Non possiamo usare la discesa del gradiente perché la soglia non è differenziabile in 0. Usiamo la seguente **regola:** $\boldsymbol{W_i \leftarrow W_i + \alpha\big(y - h_w(x)\big) * x_i}$

Questa è essenzialmente una regressione lineare, anche se qui abbiamo un problema di classificazione (l'output è 0 o 1). Intuizione:

1. Y = 1 e h_w(x) = 0: $\boldsymbol{W_i}$ **aumenta se** $\boldsymbol{x_i}$ **è positivo** e W_i **diminuisce se** $\boldsymbol{x_i}$ **è negativo**. Vogliamo **aumentare l'output** h_w(x)
2. Y = 0 e h_w(x) = i: $\boldsymbol{W_i}$ **aumenta se** $\boldsymbol{x_i}$ **è negativo** e W_i **diminuisce se** $\boldsymbol{x_i}$ **è positivo**. Vogliamo **diminuire l'output** h_w(x)

Regressione lineare univariata

Funzione lineare univariata (= retta): **y =** $\boldsymbol{W_i}$ **x +** $\boldsymbol{W_o}$

Usando la notazione **w** per il vettore [$\boldsymbol{W_0}$, $\boldsymbol{W_1}$] possiamo scrivere l'equazione precedente come segue: $\boldsymbol{h_w}$**(x) =** $\boldsymbol{W_1}$ **x +** $\boldsymbol{W_o}$

Dato un training set di n punti (x,y) nel piano, la **regressione lineare** consiste nel trovare la migliore h che si adatta ai dati.

Possiamo farlo minimizzando la soglia di errore (W * x) (o equivalentemente = soglia ($\sum_i W_i x_i$)

Error($\boldsymbol{h_w}$**) =** $\boldsymbol{\sum_j (y_j - h_w(x_j))^2 = \sum_j (y_j - W_1 x + W_o)^2}$

Questa funzione può essere minimizzata considerando i valori di $\boldsymbol{W_0}$ e $\boldsymbol{W_1}$ per i quali le derivate parziali w.r.t. $\boldsymbol{W_0}$ e $\boldsymbol{W_1}$ sono 0.

Le equazioni **δ Error(**$\boldsymbol{h_w}$**)/δ** $\boldsymbol{W_1}$ **= 0 e δ Error(**$\boldsymbol{h_w}$**)/δ** $\boldsymbol{W_1}$ hanno un'unica soluzione per ogni problema di regressione lineare con la suddetta funzione di errore. Questo conclude la storia per i modelli lineari.

Discesa gradiente

Per andare oltre i modelli lineari possiamo usare la discesa del gradiente: partiamo da un punto nello spazio dei pesi (qui il piano ($\boldsymbol{W_0}$, $\boldsymbol{W_1}$) e ci spostiamo in un punto vicino che è "in discesa" rispetto alla funzione Errore, ripetendo fino a convergere

W ← qualsiasi punto nello spazio

Loop fino alla convergenza do

For each w_i in w do

$w_i \leftarrow w_1$ – α * δ Error(h_w)/δ w_i

Apprendimento perceptron

Impara regolando i pesi per ridurre **l'errore** sul set di allenamento.

L'**errore al quadrato** per un esempio con input x e output vero y è

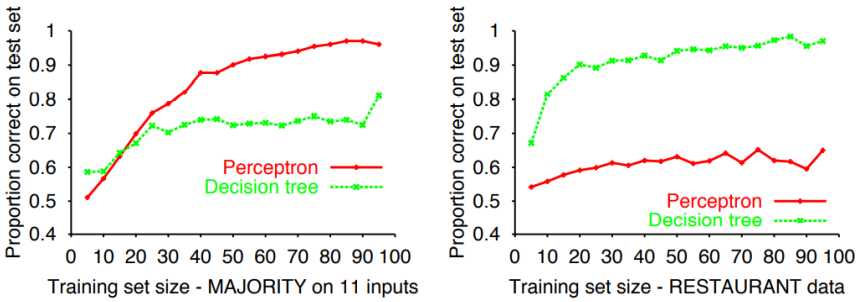
$$E = \frac{1}{2} Err^2 = \frac{1}{2}(y - h_w(x))^2$$

Eseguire la ricerca di ottimizzazione per discesa del gradiente: $\frac{\partial E}{\partial W_j} = Err * \frac{\partial Err}{\partial W_j} = Err * \frac{\partial}{\partial W_j}\left(y - g\left(\sum_{j=0}^{n} W_j x_j\right)\right) = -Err * g'(in) * x_j$

Semplice regola di aggiornamento del peso: $W_j \leftarrow W_j + \alpha * Err * g'(in) * x_j$

Ad esempio, +ve errore → aumenta l'output di rete → aumenta i pesi sugli ingressi +ve, diminuisce sugli ingressi -ve

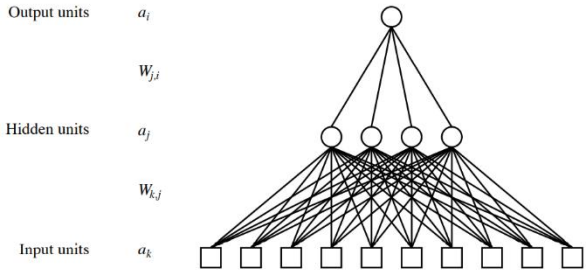
La regola di apprendimento di Perceptron converge a una funzione coerente per qualsiasi set di dati linearmente separabile



Perceptron impara facilmente la funzione di maggioranza, DTL è senza speranza. DTL apprende facilmente la funzione del ristorante, perceptron non può rappresentarla

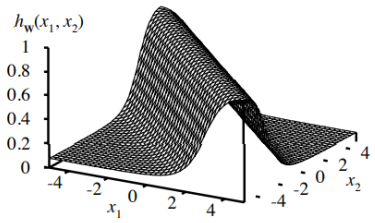
Perceptron multistrato

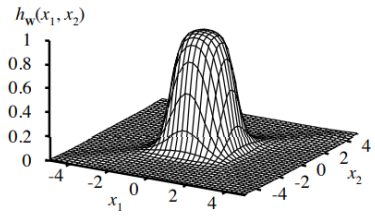
I livelli sono generalmente completamente collegati; numeri di **unità nascoste** tipicamente scelti a mano



Espressività delle MLP

Tutte le funzioni continue con 2 strati, tutte le funzioni con 3 strati





- Combina due funzioni di soglia opposte per creare una cresta
- Combina due creste perpendicolari per creare una protuberanza
- Aggiungi dossi di varie dimensioni e posizioni per adattarsi a qualsiasi superficie
- La prova richiede esponenzialmente molte unità nascoste (vedi prova DTL)

Apprendimento con retro-propagazione

La maggior parte dei neuroscienziati nega che la retro-propagazione avvenga nel cervello

- Strato di output: lo stesso del perceptron a strato singolo, $\boldsymbol{W_{j,i} \leftarrow W_{j,i} + \alpha * a_j * \Delta_i}$ dove $\boldsymbol{\Delta_i = Err_i \times g'(in_i)}$
- Livello nascosto: **retro-propaga** l'errore dal livello di output:

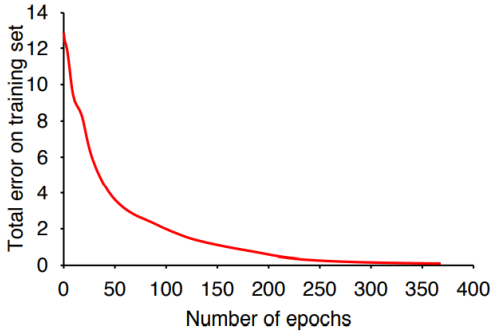
$$\boldsymbol{\Delta_i = g'(in_i) \sum_i W_{j,i} \Delta_i}$$

- Regola di aggiornamento per i pesi nel livello nascosto:

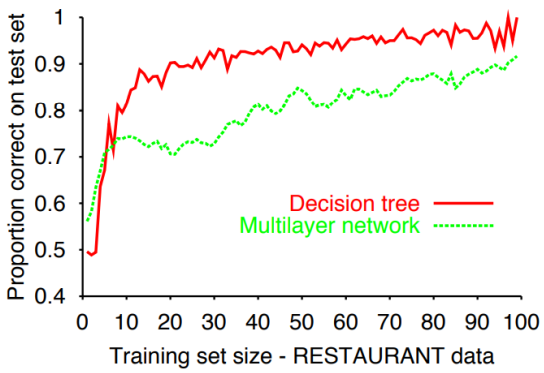
$$\boldsymbol{W_{k,j} \leftarrow W_{k,j} + \alpha * a_k * \Delta_j}$$

- Ad ogni **epoca**, somma gli aggiornamenti del gradiente per tutti gli esempi e applica

Curva di allenamento per 100 esempi di ristoranti: trova la misura esatta. Problemi tipici: convergenza lenta, minimi locali



Curva di apprendimento per MLP con 4 unità nascoste: gli MLP sono abbastanza buoni per compiti di riconoscimento di modelli complessi, ma le ipotesi risultanti non possono essere comprese facilmente

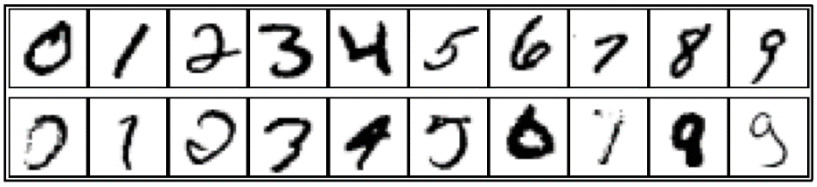


Derivazione della retro-propagazione: l'errore al quadrato su un singolo esempio è definito come $\boldsymbol{E = \frac{1}{2}\sum_i (y_i - a_i)^2}$, dove la somma è sui nodi nel livello di output.

$$\frac{\partial E}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}}$$
$$= -(y_i - a_i)g'(in_i)\frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right) = -(y_i - a_i)g'(in_i)a_j$$
$$= -a_j\Delta_i$$

$$\frac{\partial E}{\partial W_{j,i}} = -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial W_{j,i}} = -\sum_i (y_i - a_i)\frac{\partial g(in_i)}{\partial W_{j,i}}$$
$$= -\sum_i (y_i - a_i)g'(in_i)\frac{\partial in_i}{\partial W_{j,i}} = -\sum_i \Delta_i \frac{\partial}{\partial W_{j,i}}\left(\sum_j W_{j,i}a_j\right)$$
$$= -\sum_i \Delta_i W_{j,i}\frac{\partial a_j}{\partial W_{j,i}} = -\sum_i \Delta_i W_{j,i}\frac{\partial g(in_j)}{\partial W_{j,i}}$$
$$= -\sum_i \Delta_i W_{j,i}g'(in_j)\frac{\partial in_j}{\partial W_{j,i}}$$
$$= -\sum_i \Delta_i W_{j,i}g'(in_j)\frac{\partial}{\partial W_{j,i}}\left(\sum_k W_{k,j}a_k\right) = -\sum_i \Delta_i W_{j,i}g'(in_j)a_k$$
$$= -a_k\Delta_j$$

Riconoscimento delle cifre scritte a mano



3-prossimo-vicino = errore del 2,4%

400-300-10 unità MLP = errore 1,6%

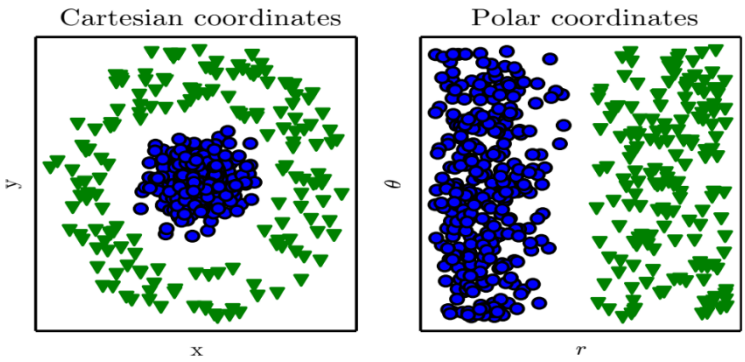
LeNet: 768–192–30–10 unità MLP = errore 0,9%

Migliore attuale (macchine kernel, algoritmi di visione) ≈ errore dello 0,6%

Apprendimento Profondo

La rappresentazione dei dati è importante

Le coordinate polari consentono di separare i dati utilizzando una semplice linea verticale, questo è impossibile con le coordinate cartesiane. Non tutti i dettagli dei dati sono davvero importanti, dobbiamo estrarre alcune funzionalità



Caratteristica: una proprietà misurabile individuale o una caratteristica di un fenomeno osservato. In altre parole, un'informazione inclusa nella rappresentazione dei dati. Ad esempio, quando viene utilizzata una tecnica di ML (regressione logistica) per raccomandare il parto cesareo, il sistema di IA non esamina direttamente il paziente, ma diverse informazioni rilevanti, come la presenza o l'assenza di una cicatrice uterina. La selezione delle caratteristiche giuste è difficile e cruciale.

Selezione delle caratteristiche

Consideriamo il compito di rilevare un'auto in una fotografia: sicuramente hanno le ruote; quindi, potremmo pensare di utilizzare la presenza di una ruota come caratteristica. Tuttavia, è difficile descrivere una ruota in termini di pixel (la semplice forma geometrica può essere complicata da ombre, sole, parafango dell'auto, ecc.).

Una possibile soluzione è **l'apprendimento della rappresentazione**: utilizzare ML non solo per scoprire l'output ma anche per scoprire la rappresentazione dei dati (ovvero le caratteristiche). Esempio, Auto-encoder, ovvero **encoder** (dati originali H -> rappresentazione) + **decoder** (rappresentazione H -> dati originali con proprietà migliori)

Fattori di variazione: spiegano i dati osservati e influenzano la nostra comprensione dei dati.

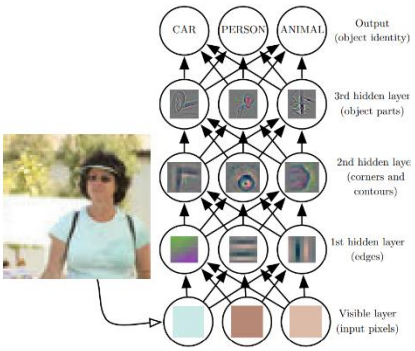
- Spesso non si tratta di quantità osservate direttamente ma di concetti e astrazioni.
- Possono anche essere dovuti a costrutti nella mente umana.
- Ad esempio, quando si considera un discorso, i Fattori di Variazione sono l'età, il sesso, l'accento dell'oratore, ... Quando si considera l'immagine di un'auto, i Fattori di Variazione sono, posizione, colore, angolo rispetto al Sole.

La **difficoltà** principale **è districare i Fattori di Variazione** e scartare quelli che non sono utili. Alcuni Fattori di Variazione influenzano ogni dato che osserviamo. E alcuni di essi sono molto difficili da estrarre (ad esempio, l'accento probabilmente non è importante ma è difficile da estrarre dai dati)

Apprendimento profondo

Districare i fattori di variazione e scartare quelli che non sono utili. Il deep learning risolve questo problema centrale nell'apprendimento delle rappresentazioni introducendo rappresentazioni che sono espresse in termini di altre rappresentazioni più semplici. Il deep learning consente al computer di costruire concetti complessi a partire da concetti più semplici.

1. L'**input** è presentato al **livello visibile** che contiene le variabili che siamo in grado di osservare.
2. Una serie di **livelli "nascosti"** estrae dall'immagine caratteristiche sempre più astratte, i valori di questi livelli non sono riportati nei dati. Le immagini qui sono visualizzazioni del tipo di caratteristica rappresentata da ciascuna unità nascosta.
    I. Dati i pixel, il primo livello può facilmente identificare i **bordi**, confrontando la luminosità dei pixel vicini.



II. Il secondo livello nascosto può facilmente cercare **angoli** e **contorni** estesi, che sono riconoscibili come raccolte di bordi.
III. Il terzo livello nascosto può rilevare intere **parti di oggetti** specifici, trovando raccolte specifiche di contorni e angoli.

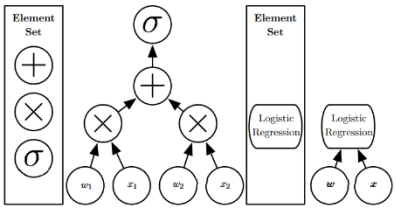
3. Infine, questa descrizione dell'immagine in termini di parti dell'oggetto può essere utilizzata per riconoscere gli oggetti presenti nell'immagine.

Un semplice modello DL è Multilayer Perceptron (MP) che è solo una funzione che mappa un insieme di valori di input a valori di output. La funzione è formata componendo molte funzioni più semplici.

1. Possiamo pensare che ogni applicazione di una funzione diversa fornisca una nuova rappresentazione dell'input, quindi deep learning = apprendimento della corretta rappresentazione dei dati.
2. Possiamo considerare la profondità come ciò che consente al computer di apprendere un programma per computer in più fasi. Ogni strato della rappresentazione è lo stato della memoria del computer dopo aver eseguito un'altra serie di istruzioni in parallelo. Le reti con maggiore profondità possono eseguire più istruzioni in sequenza.

Cos'è la profondità e quando un modello è profondo?

1. **Profondità:** numero di istruzioni sequenziali che devono essere eseguite nel modello (cioè numero di nodi in un percorso nel grafo computazionale. In questo caso le profondità dipendono dal linguaggio, vedi sotto)
2. Profondità del grafico che descrive come i concetti sono correlati tra loro (di solito più piccolo del precedente)
3. Nessun consenso su un numero per "profondo"



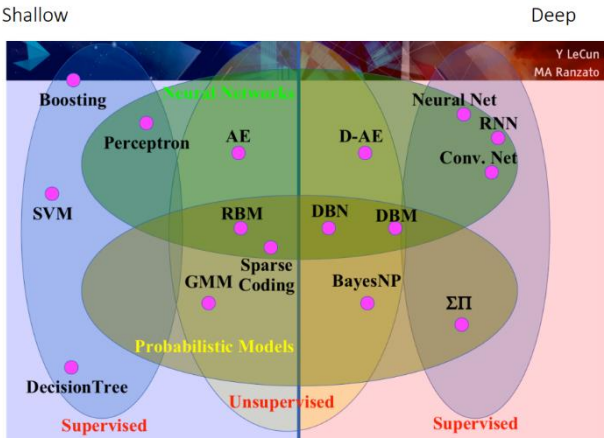
Perché molti livelli: possiamo approssimare qualsiasi funzione il più vicino possibile con un'architettura superficiale. Perché avremmo bisogno di quelli profondi? $y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \quad y = F(W^1 . F(W^0 . X))$

Le macchine profonde sono più efficienti per rappresentare determinate classi di funzioni, in particolare quelle coinvolte nel riconoscimento visivo. Può rappresentare funzioni più complesse con meno "hardware". Abbiamo bisogno di una parametrizzazione efficiente delle funzioni che sono utili per le attività "AI" (visione, audizione, NLP, ...) $y = F(W^K . F(W^{K-1} . F(\dots F(W^0 . X) \dots)))$

Rivoluzione causata dall'apprendimento delle funzionalità

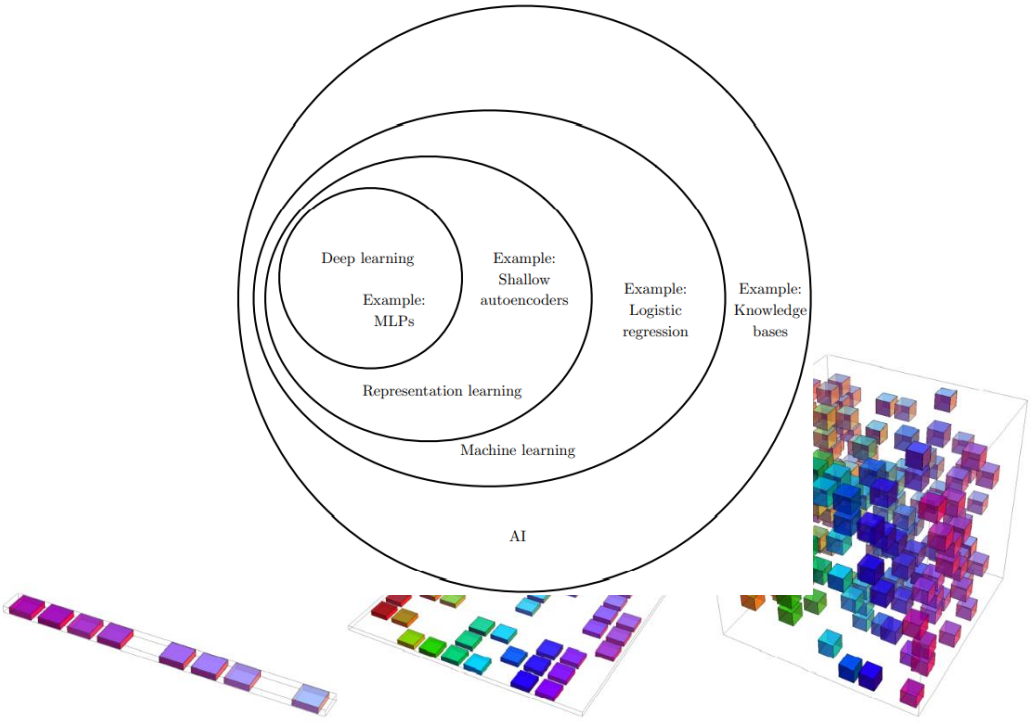
- Riconoscimento vocale I (fine anni '80)
- Funzionalità di livello medio addestrate con miscele gaussiane (classificatore a 2 strati)
- Riconoscimento della scrittura a mano e OCR (dagli anni '80 a '90)
- Reti convoluzionali supervisionate operanti su pixel
- Rilevamento di volti e persone (dagli anni '90 alla metà degli anni 2000)
- Reti convoluzionali supervisionate operanti su pixel (Garcia 2004)
- Haar dispone di generazione/selezione (ViolaHJones 2001)
- Riconoscimento degli oggetti I (da metà a fine anni 2000)
- Funzionalità di livello intermedio addestrabili (medie K o codifica sparsa)
- Riconoscimento di oggetti a bassa risoluzione: segnali stradali, numeri civici (inizio 2010)
- Rete convoluzionale supervisionata operante su pixel
- Riconoscimento vocale II (circa 2011)
- Reti neurali profonde per la modellazione acustica
- Riconoscimento degli oggetti III, Etichettatura semantica (2012)
- Reti convoluzionali supervisionate operanti su pixel

Modelli di apprendimento



Maledizione della dimensionalità

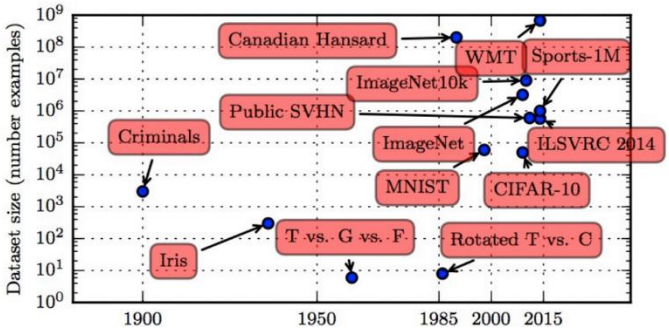
Machine learning e AI



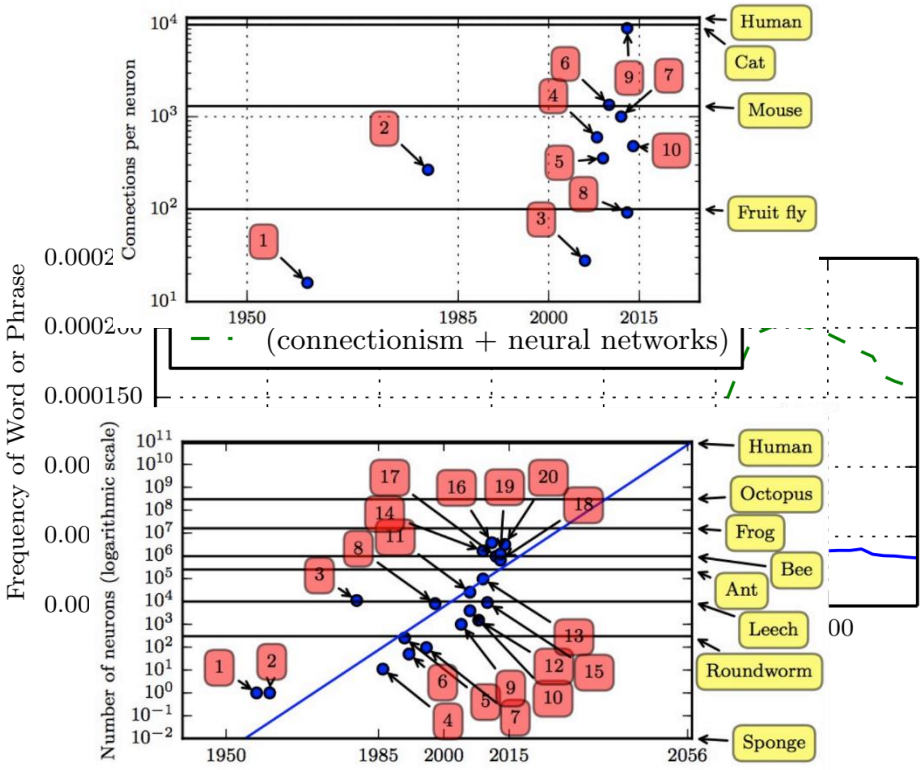
Apprendimento profondo rispetto ad altri sistemi di intelligenza artificiale

Tendenza

Dimensione del set di dati

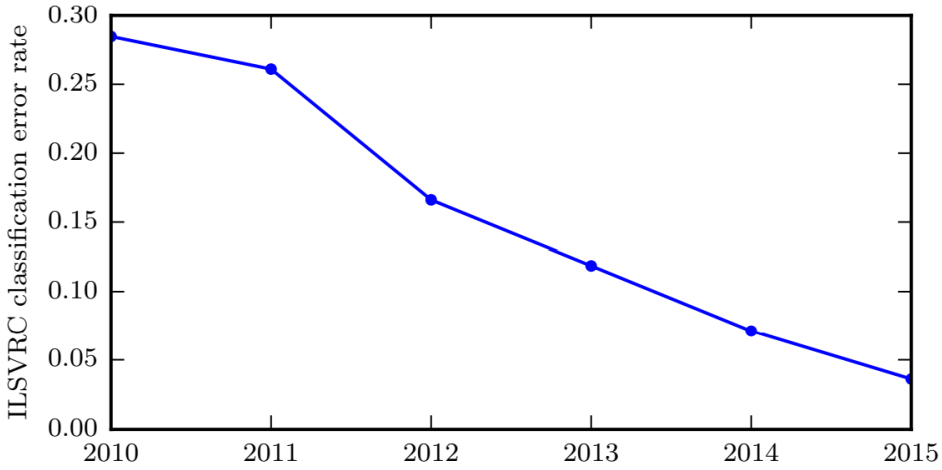


Connessioni per neurone



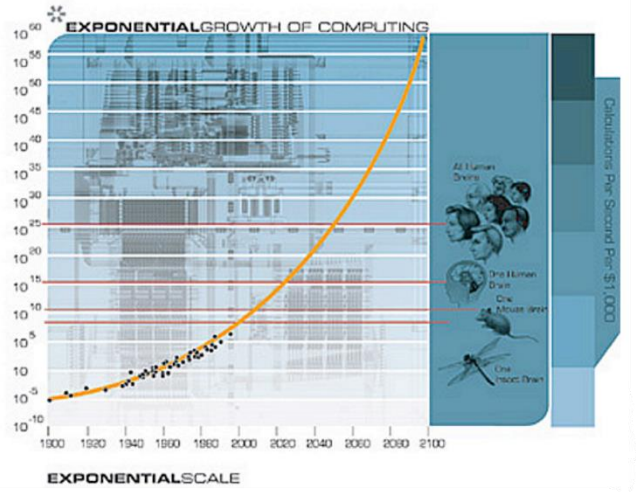
Numero di neuroni

Riconoscimento di oggetti



Considerazioni economiche ed etiche

Perché ora? Crescita esponenziale della potenza di calcolo



1996	2006 (9 years later!)
ASCI Red, the world faster supercomputer	Sony PlayStation 3
$55 million	$500
1,600 square feet	1/10 of a square feet
1.8 teraflops (1.8 trillion, i.e. 10^{12} operations per second)	1.8 teraflops
800,000 watts per hour	200 watts per hour

	Supercomputer	Personal Computer	Human Brain
Computational units	10^4 CPUs, 10^{12} transistors	4 CPUs, 10^9 transistors	10^{11} neurons
Storage units	10^{14} bits RAM	10^{11} bits RAM	10^{11} neurons
	10^{15} bits disk	10^{13} bits disk	10^{14} synapses
Cycle time	10^{-9} sec	10^{-9} sec	10^{-3} sec
Operations/sec	10^{15}	10^{10}	10^{17}
Memory updates/sec	10^{14}	10^{10}	10^{14}

Figure 1.3 A crude comparison of the raw computational resources available to the IBM BLUE GENE supercomputer, a typical personal computer of 2008, and the human brain. The brain's numbers are essentially fixed, whereas the supercomputer's numbers have been increasing by a factor of 10 every 5 years or so, allowing it to achieve rough parity with the brain. The personal computer lags behind on all metrics except cycle time.

Quanto sarebbe costato un iPhone nel 1991? 32 GB di memoria flash ($1,44 milioni) + 1 GB nel 1991 ($45.000, ora costa $0,55) + processore ($620.000) + connettività ($1,5 milioni) = $3,56 milioni + fotocamera, rilevamento del movimento, sistema operativo, display, ecc.

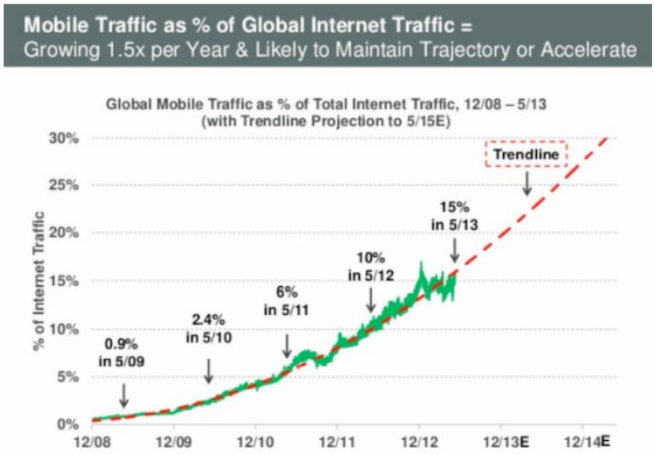
Tanta più potenza di elaborazione ora: il nostro telefono ha la stessa potenza di tutta la NASA nel 1969, quando mandarono un uomo sulla luna. Il chip nei nostri biglietti d'auguri è più potente di tutte le forze alleate nel 1945!

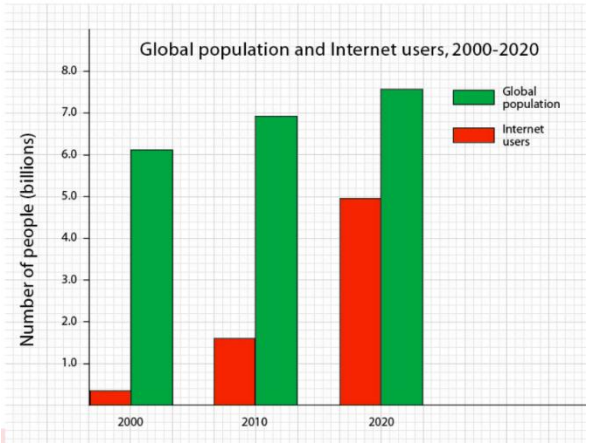
Google ha 1.800.000 server, 43 petaflop (1 petaflop: 1.000 trilioni = 1015 opere al secondo)

Perché ora?

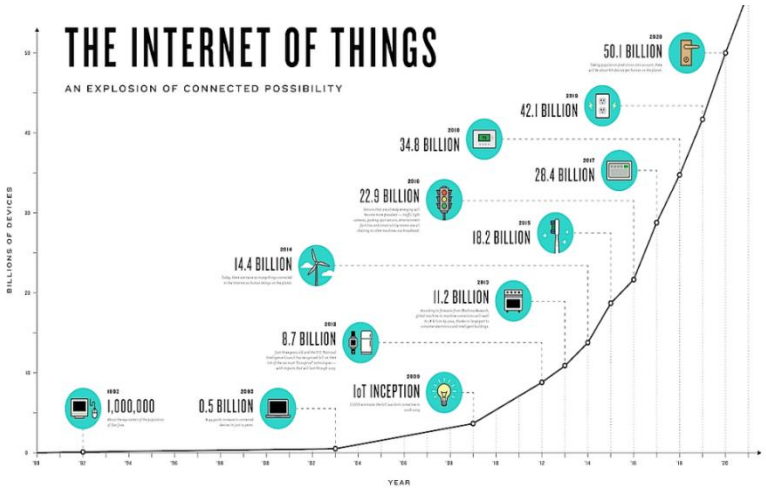
- Immense quantità di dati disponibili sul mondo e sul comportamento umano che possono essere usati come esempi, insegnando alle IA ad essere più intelligenti
- Apprendimento profondo, buon software open source riutilizzabile (algoritmi e modelli migliori), calcolo parallelo economico

Sempre più persone si connettono ogni giorno





Non solo persone, ma anche cose



Tutti possono essere innovatori: più di 6 miliardi di telefoni cellulari nel 2012 e tutte queste persone possono cercare in rete, leggere Wikipedia, seguire i corsi online, condividere opinioni su blog, Twitter, ecc., eseguire l'analisi dei dati utilizzando i servizi cloud

Impatto sulle imprese e sul mercato del lavoro

	cognitive	manual
routine	Processing payments, Bank tellers, cashiers, mail clerks, translation, accounting, driving, Secretary, real estate	Machine operators, cement masons, janitors, house cleaning
Non-routine	Handling customers' questions, Financial analysis	Hair-dressing

Cambiamenti tecnologici e posti di lavoro

- Kodak 1984: 45.000 persone; 2012: bancarotta!
- Instagram 2012: 13 persone, vendute a FB per 1 miliardo di dollari
- Foxconn (produzione di componenti elettronici): $100 miliardi, 1,2 milioni di persone, sta creando un esercito di 1 milione di robot!
- Lo stesso in Canon e molti altri

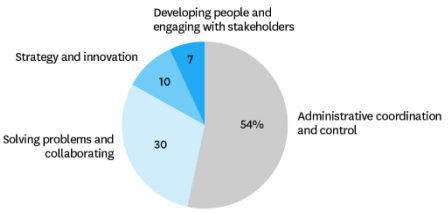
Il futuro del lavoro

2013: il **47% dei posti di lavoro negli Stati Uniti sarà sostituito in 20 anni** dall'automazione.

1. **Persone sostituite in settori vulnerabili:** produzioni, trasporti/logistica, supporto amministrativo
2. **Rallentamento della sostituzione** a causa del collo di bottiglia dell'ingegneria: intelligenza creativa e sociale, percezione e manipolazione
3. **L'IA consentirà di sostituire i posti di lavoro** nei settori della **gestione**, della scienza, dell'ingegneria e delle arti

Studio più recente (2016): a rischio il 77% dei posti di lavoro in Cina e il 69% dei posti di lavoro in India. La divergenza nell'adozione della tecnologia può rappresentare l'82% dell'aumento del divario di reddito in tutto il mondo negli ultimi 180 anni. Nel 1820, i redditi nei paesi occidentali erano di 1,9 Ames quelli nei paesi non occidentali. Nel 2000, 7.2 Ames!

D'altra parte, l'**innovazione** è importante per la crescita e l'**intelligenza artificiale** è importante per l'innovazione



Cinque pratiche che i manager di successo dovranno padroneggiare

1) Lasciare l'amministrazione ad AI: la società di analisi dei dati Tableau e la società NPL Narrative Science hanno sviluppato un software che crea automaticamente spiegazioni scritte per la grafica di Tableau. L'86% dei manager intervistati apprezza il supporto dell'IA per il monitoraggio e il reporting.

2) Concentrarsi sul lavoro di giudizio: molte decisioni richiedono conoscenza della storia e della cultura organizzativa, empatia, riflessione etica. L'IA fornisce supporto per la decisione, non per la sostituzione.

3) Trattare le macchine AI come "colleghi" non concorrenti: l'intelligenza artificiale può fornire supporto decisionale, simulazioni basate sui dati, attività di ricerca e scoperta. Il 78% crede che si fiderà dei consigli dell'AI nel prendere decisioni aziendali. Il sistema Kensho Technologies consente ai gestori degli investimenti di porre domande in un inglese semplice, come "Quali settori e industrie ottengono i migliori risultati tre mesi prima e dopo un aumento dei tassi?".

4) Lavora come un designer (capacità di sfruttare la creatività degli altri)**:** il 33% dei manager ha identificato il pensiero creativo e la sperimentazione come un'area di abilità chiave di cui hanno bisogno per imparare per avere successo.

5) Sviluppare abilità e reti sociali: i manager hanno sottovalutato le abilità sociali fondamentali per il networking, il coaching e la collaborazione che li aiuteranno in un mondo in cui l'IA svolge molti dei compiti amministrativi e analitici che svolgono oggi.

Suggerimenti

a) **Esplora presto l'IA:** il disturbo sta arrivando
b) **Adottare nuovi indicatori chiave di prestazione:** l'intelligenza artificiale porterà nuovi criteri per il successo (capacità di collaborazione, condivisione delle informazioni, efficacia dell'apprendimento e del processo decisionale e capacità di andare oltre l'organizzazione per ottenere approfondimenti).
c) **Sviluppare strategie di formazione e reclutamento** per la creatività, la collaborazione, l'empatia e le capacità di giudizio. I leader dovrebbero sviluppare una forza lavoro diversificata

Questioni filosofiche ed etiche: dipende molto dal significato di "intelligenza". I ricercatori di IA danno per scontata l'IA debole (e non si preoccupano troppo dell'ipotesi di IA forte)

- **IA debole:** possiamo costruire macchine che possano agire come se fossero intelligenti?
- **IA forte:** possiamo costruire macchine che siano realmente intelligenti?

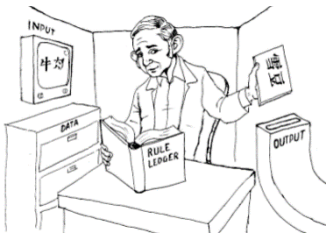
Obiezione di coscienza all'IA

Le macchine possono pensare? Turing l'ha riconosciuta come una domanda mal posta -> **convenzione educata che tutti pensano**

1. **L'obiezione teologica:** solo gli esseri creati da Dio possono pensare
2. **L'obiezione matematica:** qualsiasi teoria formale forte come l'aritmetica di Peano contiene affermazioni vere che non hanno prove all'interno della teoria stessa.
3. **Varie disabilità:** non possono essere gentili, intraprendenti, belli, amichevoli, avere iniziativa, avere senso dell'umorismo, innamorarsi, gustare fragole e panna
4. **L'obiezione di Lady Lovelace:** la macchina analitica non ha alcuna pretesa di dare origine a qualcosa. Può fare tutto ciò che sappiamo come ordinargli di eseguire.
5. **Informalità del comportamento:** impossibilità di fornire regole che descrivano come comportarsi in ogni possibile situazione

Due visioni opposte

1. Naturalismo biologico: gli stati mentali sono caratteristiche emergenti di alto livello causate da processi fisici di basso livello nei neuroni, e sono le proprietà non specificate dei neuroni che contano.



- **Stanza cinese:** mano che parla inglese monolingue traccia un programma di comprensione del linguaggio naturale per il cinese seguendo le istruzioni scritte in inglese. Dall'esterno vediamo un sistema che risponde in cinese senza capirlo

2. Funzionalismo: uno stato mentale è qualsiasi condizione causale intermedia tra input e output. Qualsiasi due sistemi con processi causali isomorfi avrebbero gli stessi stati mentali. Pertanto, un programma per computer potrebbe avere gli stessi stati mentali di una persona. Il presupposto è che ci sia un certo livello di astrazione al di sotto del quale l'implementazione specifica non ha importanza.

- **Esperimento di sostituzione del cervello (1988):** sostituzione frammentaria dei neuroni con dispositivi elettronici funzionalmente equivalenti. Il comportamento esterno rimane lo stesso**.** Per i funzionalisti (es. Moravec) il comportamento interno (cioè la coscienza) rimarrebbe lo stesso. Per i naturalisti biologici (Searle) la coscienza svanirebbe. Un problema più generale di Mente Corpo.

Problemi etici

Etica della macchina: assunzioni computazionali e filosofiche per macchine che possono prendere decisioni morali autonome

Veicoli autonomi: chi dovrebbe essere ucciso in caso di incidente?

Robot medici: devono sempre dire la verità ai pazienti?

Come possiamo garantire che le macchine non prendano decisioni “immorali”?

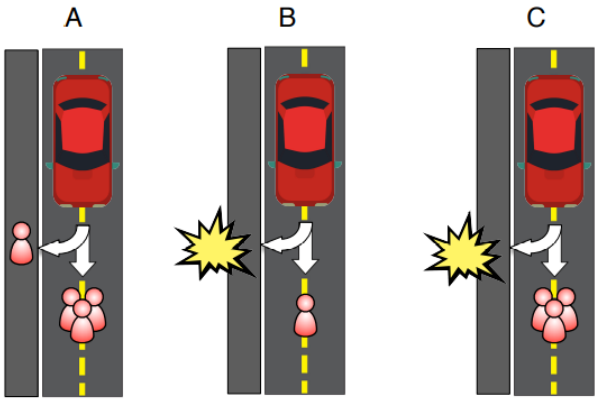
1. Regole semplici (es. Asimov tre leggi sulla robotica) non bastano, viste le complesse informazioni contestuali
2. La simulazione della decisione morale e delle azioni degli umani non è sufficiente, poiché gli umani prendono anche cattive decisioni morali: **le macchine devono essere "sante"**

Avremmo bisogno di

1. Un'etica normativa che risolva tutti i dilemmi morali esistenti e che è accettata dalla maggior parte degli esseri umani
2. Una traduzione di tale etica in termini computazionali
3. La capacità di incorporare il ragionamento di buon senso nelle macchine

Problema del carrello

I comportamenti umani e delle macchine sono regolati da leggi diverse, ovvero si applicano diverse responsabilità legali



1. Auto guidata dall'uomo

a) La scelta di rimanere sulla rotta e lasciare che diversi pedoni vengano uccisi, piuttosto che sterzare e uccidere un passante, può essere giustificata sulla posizione morale-legale che **condanna la causa intenzionale della morte** (distinta dal risultato della morte condotto dalla propria omissione).
b) La scelta di mantenere la rotta può essere giustificata invocando lo **stato di necessità**, poiché tale scelta è necessaria per salvare la vita del conducente.
c) La stessa giustificazione vale anche qui, anche se in questo caso la scelta del conducente di salvare la propria vita porta alla morte di diverse altre persone.

2. Veicolo autonomo pre-programmato

a) **È dubitativo se il programmatore sarebbe giustificato** quando sceglie di programmare un AV in modo che rimanga sulla rotta e uccida diversi pedoni piuttosto che sterzare e uccidere un solo passante. Infatti, la distinzione tra omettere di intervenire (condurre l'auto a seguire il suo percorso) e agire in modo determinato (scegliere di sterzare); distinzione che nel caso di un'auto presidiata può giustificare la scelta umana di consentire all'auto di andando dritto, non sembra applicarsi al programmatore, dal momento che quest'ultimo sceglierebbe deliberatamente di sacrificare un numero maggiore di vite.
b) Quando l'autore non è direttamente in pericolo e non agisce per autoconservazione (o conservazione dei parenti), l'applicabilità della difesa generale dello stato di necessità è controversa. Ad esempio, Santoni de Sio (2017) sostiene che la legge generalmente non consente l'uccisione di una persona innocente per aver salvato la vita di altre persone. Su questa base rifiuta la pre-programmazione utilitaristica degli AV. **Se la giurisdizione legale consentisse tale caso particolare di stato di necessità, il programmatore non sarebbe punibile per nessuna delle due scelte. Diversamente**, se questo non è accettato dalla giurisdizione, allora **è molto dubitativo se pre-programmare l'auto sia per andare dritto (uccisione di un pedone) sia per sterzare (uccisione del passeggero) sarebbe legalmente accettabile**: in entrambi i casi il programmatore sceglierebbe arbitrariamente tra due vite.
c) Sembra che **la pre-programmazione dell'auto per continuare la sua traiettoria**, causando la morte di un numero maggiore di persone, **non potrebbe essere moralmente e legalmente giustificata in nessuna giurisdizione**: equivarrebbe a una scelta arbitraria uccidere molti piuttosto di uno.

Conclusioni: sviluppare una società di menti e macchine…

- Economica, affidabile, intelligenza digitale dietro a tutto, quasi invisibile
- Man mano che le macchine sostituiranno e aumenteranno gli umani in un numero sempre maggiore di compiti, capiremo cosa ci rende umani e cosa significa intelligenza
- Ame più libero, meno bisogno di lavorare
- Amplificazione delle capacità umane e collettive



...ma dovremmo davvero farlo?

- Si perderanno molti posti di lavoro: occorre ridefinire politiche ed economie
- Le persone potrebbero perdere il senso di essere uniche: l'umanità è sopravvissuta ad altre battute d'arresto (Copernico, Darwin…)
- I sistemi di intelligenza artificiale potrebbero essere utilizzati per fini indesiderabili: l'esercito americano ha schierato oltre 17.000 veicoli autonomi in Iraq
- L'uso di sistemi di intelligenza artificiale potrebbe comportare una perdita di responsabilità: responsabilità per diagnosi/decisioni sbagliate? Salute, finanza, automobili...
- Il successo dell'IA potrebbe significare la fine della razza umana
    - La stima dello stato del sistema di intelligenza artificiale potrebbe essere errata
    - Funzione di utilità giusta per un sistema di intelligenza artificiale da massimizzare (sofferenza umana...)
    - **L'apprendimento permette di sviluppare comportamenti non intenzionali: singolarità tecnologica**