

+

×

-

÷

Overview programma del corso

Intelligenza
Artificiale
~ Tommaso Pellegrini

Intelligenza artificiale : - definizione McCarthy - 4 punti di vista - turing test
- neural networks - teoria dei giochi - giocatori reali

Reti neurali : - neuroni - unità di McCulloch - Pitts - tipi di rete neurale
- tipi di training - feed forward networks - single-layer perceptrons
- multi-layer perceptrons - back/forward propagation - regole di apprendimento

constraint satisfaction problems: - CSPs - soluzioni e assegnamenti
- tipi di CSPs - grafici - generate and test - standard backtracking
- euristica - propagazione vincoli - forward checking - arco-consistenza
- CSP parametrici - minizinc

- Introduzione all'intelligenza artificiale e gli agenti intellettuali
- Cenni sulle reti neurali
- Problemi di soddisfacimento vincoli
- Logica e agenti razionali
- Soluzioni di problemi mediante ricerca informata e non informata.

Riassunto slides:

Intelligenza artificiale: pg

Scopo: macchine intelligenti → capaci di interagire con il mondo reale.
+ risolvere problemi complessi, comportamento relazionale, interazioni con mondi complessi e dinamici.

Definizione di McCarthy: c'è la scienza ed ingegneria del creare macchine intelligenti.
→ **intelligenza?** È la parte computazionale della capacità di raggiungere obiettivi nel mondo.

Ai
forte: intelligenza non distinguibile da quella umana
ragionare non è nient'altro che calcolo
debile: non sono mai in grado di uguagliare la mente umana
non sono mai così complesso

pensare come una persona	②	pensare razionalmente
agire come una persona	①	agire razionalmente

①

Test di Turing: se una persona non si rende conto di stare interrogando con una macchina, allora quella macchina è intelligente tanto quanto le persone che sta imitando.

Definisce il termine "comportamento intelligente", iniziere quello dell'uomo.

② Imitare il comportamento dell'uomo → simulare il cervello
↳ creare cervello elettronico

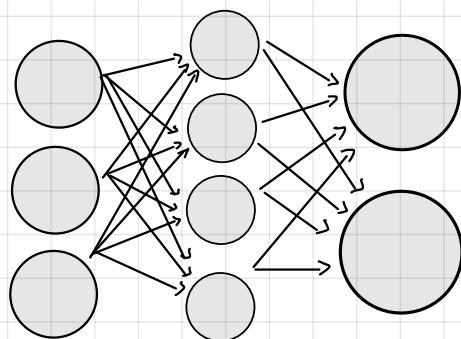
Rete neurale:

unità simula 1 neurone

↳ collegate in rete

riceve input da sensori

produce output su attuatori



La rete viene addestrata:

è troppo difficile da programmare, apprende per rinforzo.

Esempio: un riconoscitore di testo manoscritto è un esempio di rete convoluzionale profonda (CNN) che apprende tramite deep learning.

- 3 Pensiero dell'uomo: non sempre razionale (condizionato).
 Reasoning razionale → solo quello delle logiche.
 Logica → consente di compiere deduzioni.
- Sistemi esperti: macchine con regole di inferenza + conoscenza base.
 - ↳ fatti ritenuti veri
 - ↳ deduce nuovi fatti dalle regole e impone lavorando.

Esempio:

inferenze:

- IF (diarrea AND temp > 37.5)
 THEN colera
- IF colera
 THEN ospedalizza - paziente

base di conoscenza:

- Ø
- colera
- diarrea
- temp
- ospedalizza - paziente

- 4 Approccio moderno → sistemi che si comportano razionalmente: agenti intelligenti: non importa come, basta che si comporti tale.

La teoria dei giochi descrive il comportamento razionale senza cercare di capire come questo venga generato → massimizza l'utilità.

Reti Neurali (artificiali)

p36

! Una rete neurale artificiale è un approssimatore di funzioni multi-variabile

base
 solo unità semplici (neuroni)
 neuroni collegati tramite una rete
 la topologia della rete determina la capacità di approssimazione

rete neurale
 non eseguono programmi, rispondono reattivamente agli input
 vengono addestrate, rispondono a input mai incontrati (*)
 incapaci di giustificare i risultati

* capacità di generalizzazione

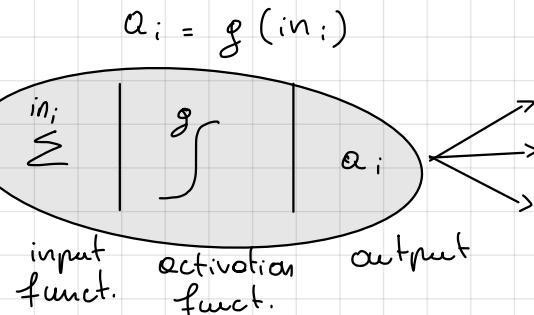
o Unità di McCullagh - Pitts

bias weight

$$Q_0 = -1 \quad w_{0,i}$$

$$Q_j \quad w_{j,i} \quad >$$

input links



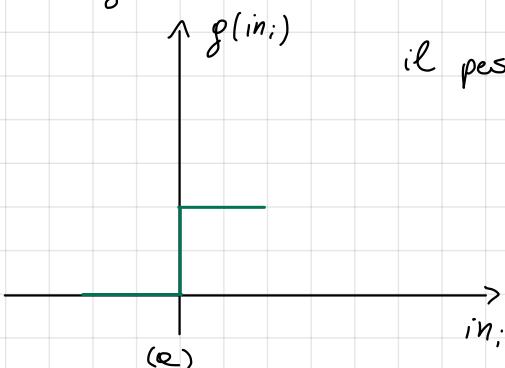
$$a_i \leftarrow g(in_i) = g(\sum_j w_{j,i} a_j)$$

output links

E' un modello semplice (poco realistico) dei neuroni \rightarrow potenza nella rete
 parametri:
 - pesi su ogni arco di input $w_{j,i}$
 - funzione di attivazione $g(\cdot)$

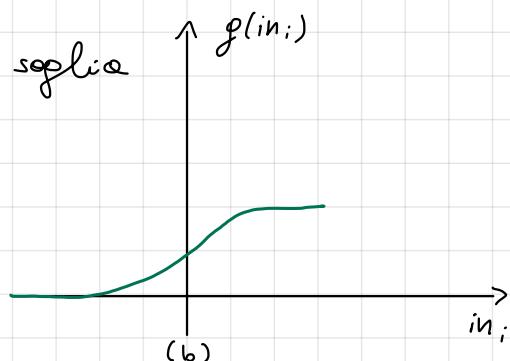
o Funzioni di attivazione

- gradino unitario $U(x)$



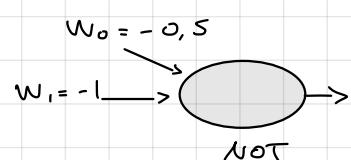
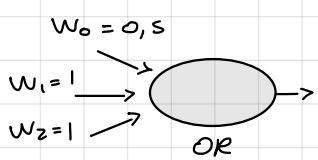
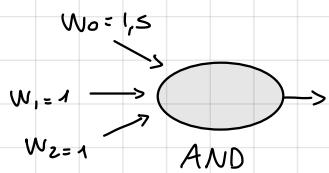
il peso $w_{0,i}$ sposta la soglia

- sigmoida $1/(1 + e^{-x})$



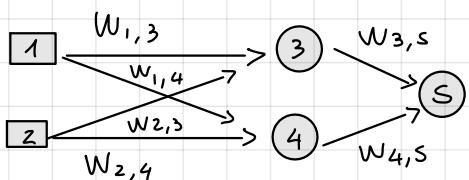
o Neuroni e dispositivi logici

Input e output in $B = \{0, 1\}$, funzione di attivazione e gradino unitario.



o tipi di rete

- Reti in avanti (feed forward) {
 - topologie e grafo orientato aciclico
 - strutturate su vari strati
 - non ci sono archi che collegano out con input
- Reti ricorrenti {
 - topologie e grafo orientato
 - la rete memorizza su uno stoto



Esempio di rete feed forward

Sempre feed forward:

RFF → f. att. fissate e priori
 realizza f non lineari in più variabili → non linearità in f. att.
 proiettore RFF: identificare i pesi che permettono di associare ogni input all'output desiderato.

apprendimento: identificazione dei pesi sulla base di esempi forniti durante l'addestramento.

o tipi di apprendimento:

- Fase di addestramento: tramite set di coppie $\langle x, y \rangle$ corrette training set. Training set rappresentato per più epochhe di apprendimento.
- Test (validazione): fornito un insieme di coppie $\langle x, y \rangle$ non usato nella fase di addestramento. Vengono valutate le performance.
- Addestramento supervisionato: per ogni coppia $\langle x, y \rangle$ si valuta l'errore commesso. I pesi vengono modificati a ridurre l'errore.
 - ↳ modifica A coppia $\langle x, y \rangle$
 - ↳ modifica A epoca, raccolgendo info sull'errore complessivo.

- **Addestramento non supervisionato**: stabilite a priori una regola di modifica dei pesi. Nel training, per ogni coppia $\langle x, y \rangle$ si applica la regola solo per x . La validazione viene fatta sul test usando input e output.
↳ usato spesso da reti ricorrenti

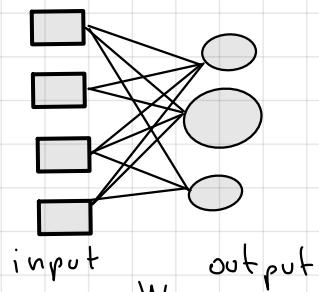
• Single layer perceptron

La rete più semplice: un solo insieme di pesi separa l'input dall'output.

Perceptron:

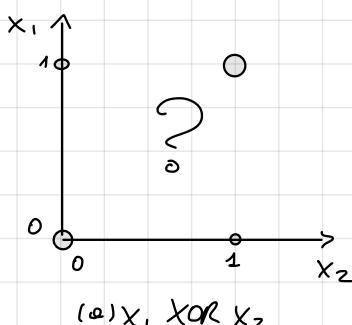
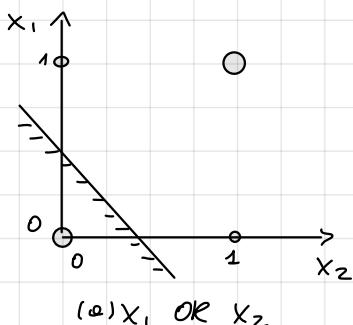
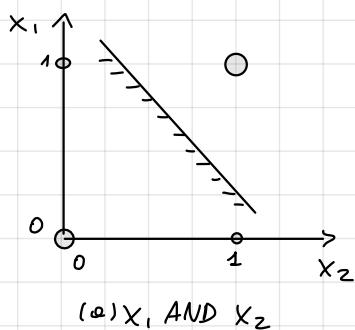
- classificatore basato applicate solo nello strato di uscite.
sulle unità di McCullagh
 - per ogni input si attiva un solo output \rightarrow corrisponde alla classe di appartenenza dell'input.

- Si sceglie come f. il gradino unitario
 - Ha potenze di calcolo limitate (no xor)



fuzioni da separatore lineare

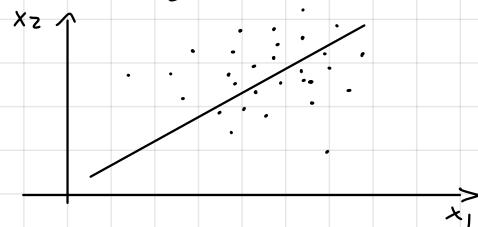
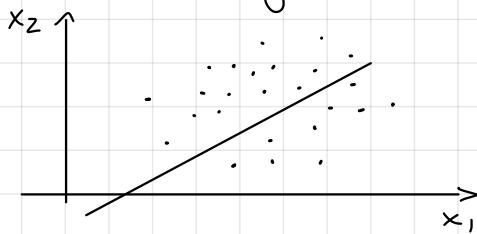
$$\sum_j w_j x_j > 0 \quad \text{OR} \quad w \cdot x > 0$$



• Classificatore binario

Gli input vengono classificati in due categorie (classi)

$$\sum_{i=0}^n w_i x_j = 0$$



Parametri di qualità di un classificatore binario:

- TP = veri positivi
 - TN = veri negativi
 - FP = falsi positivi
 - FN = falsi negativi

sensitivity, specificity :

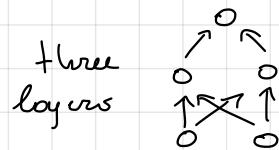
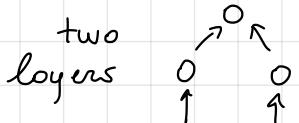
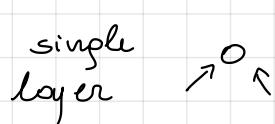
$$- SE = TP / (TP + FN), \quad SP = TN / (TN + FP)$$

precision, recall:

$$- PR = TP / (TP + FP), \quad RE = SE = TP / (TP + FN)$$

Le caratteristiche dello spazio degli input impongono le scelte della rete!

↓
Non sono (spesso) conosciute a priori { approccio incrementale
parte da rete semplice
si arriva a rete richiesta } ↓



o Regole di addestramento:

reti → approssimatori di funzioni

errore nelle approssimazioni → cambia il valore dei pesi

errore → inteso come una funzione dei pesi: $E(W)$

$E(W) \rightarrow$ avrà punti in cui il suo valore è minimo

↳ ricerche dei minimi di E

→ Apprendimento supervisionato { $E = \frac{1}{2} Err^2 = \frac{1}{2} (y - h_W(x))^2$
cerca di minimizzare l'errore quadratico }

tecniche di minimizzazione: discesa del gradiente

1 - insieme di pesi casuali

1 - valutiamo E e modifichiamo i pesi nella direzione
in cui punta $-\nabla E$

$$\text{Siccome vale } \frac{\partial E}{\partial w_j} = Err \times \frac{\partial Err}{\partial w_j} = Err \times \frac{\partial}{\partial w_j} (y - g(\sum_{j=0}^n w_j \times j)) = -Err \times g'(in) \times x_j$$

Possiamo usare le seguenti regole per coeff. di apprendimento a fisso:

$$w_j \leftarrow w_j + \alpha \times Err \times g'(in) \times x_j$$

- Il metodo potrebbe fermarsi in un punto di minimo locale

- Valori piccoli di α fanno sì che le ricerche più accurate

o Overfit e generalizzazione

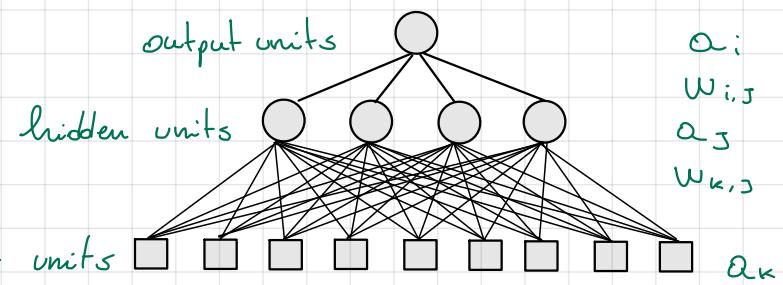
Una rete deve approssimare bene i valori su cui $\xrightarrow{(1)} e$ e $\xrightarrow{(2)}$ non e state addestrate

1 → impone addestramento accurato (non troppo)
↳ overfit

2 → impone addestramento su un training set rappresentativo
↳ deve saper generalizzare

o Multi-Layer Perceptron

- ha uno o più strati nascosti
- le f. di att. (sigmoide) fissate e priori.



! Supera i limiti del single layer perceptron:

- 1 strato nascosto \rightarrow approssima tutte le funzioni continue con
- 2 strati nascosti \rightarrow approssima tutte le funzioni

o Procedura di addestramento:

back propagation:

inizializza pesi con val piccoli in $(-1, 1)$

do

init e a 0 l'errore complessivo su training set

for each pattern nel set do

Forward Propagate

Back Propagate

end

while (n iterazioni < n_{max}) and (errore > min richiesto)

Forward Propagate

for each strato rete do

for each nodo nello strato scelto do

1. calcola somma input pesati

2. aggiungi soglia

3. calcola valore della f. di attivazione

end

end

Back Propagate

for each nodo nello strato out do

calcola l'errore

end

for each strato nascosto do

for each nodo nello strato nascosto do

1. calcola errore del nodo

2. aggiorna i pesi del nodo

end

end

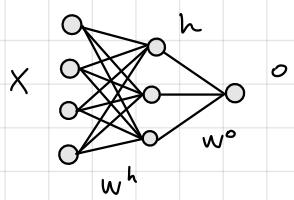
o Aggiornamento dei pesi:

- Back propagation → su strato output si applica le regole viste per il single layer perceptron

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times \alpha_j \times \Delta_i \quad \Delta_i = E_{out,i} \times g'(in_i)$$

- Si retropropaga l'errore su strati nascosti $\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \quad W_{k,j} \leftarrow W_{k,j} + \alpha \times \alpha_k \times \Delta_j$

- Nel caso di un solo strato nascosto e f. di att. sigmoidale:



$$\Delta w_j^o = \alpha \delta^o h_j$$

$$\delta^o = (y - o) \cdot o \cdot (1 - o)$$

$$\Delta w_{ij}^h = \alpha \delta_j^h x_i$$

$$\delta_j^h = \delta^o \cdot w_j^o \cdot h_j \cdot (1 - h_j)$$

o Algoritmo di discesa del gradiente:

Si usa per trovare il max o il min di una funzione in più variabili.

1. Calcolare le derivate parziali delle f rispetto a ciascuna variabile.
es: $f(x,y) = 3x^3 - 2y^2 + 7 \rightarrow$ le derivate parziali sono:
 $\frac{\partial}{\partial x} = 9x^2$
 $\frac{\partial}{\partial y} = -4y$
2. Scegliere un punto di partenza (x,y) dentro all'intervallo di interesse.
3. Scegliere un valore piccolo per $\alpha \rightarrow$ tasso di apprendimento.
4. Iterare fino a quando x e y non cambiano più o limite prestabilito.
 $x = x + \alpha \cdot \frac{\partial}{\partial x}$
 $y = y + \alpha \cdot \frac{\partial}{\partial y}$
5. I valori finali di x e y rappresentano il massimo e il minimo.

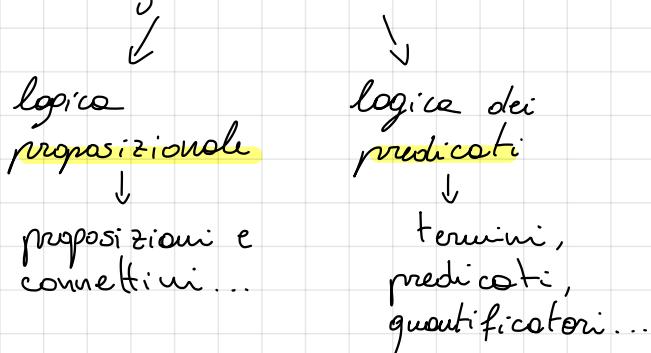
Agenti che pensano in modo razionale

- Linguaggi logici → sintassi: insieme regole ben formate (L)

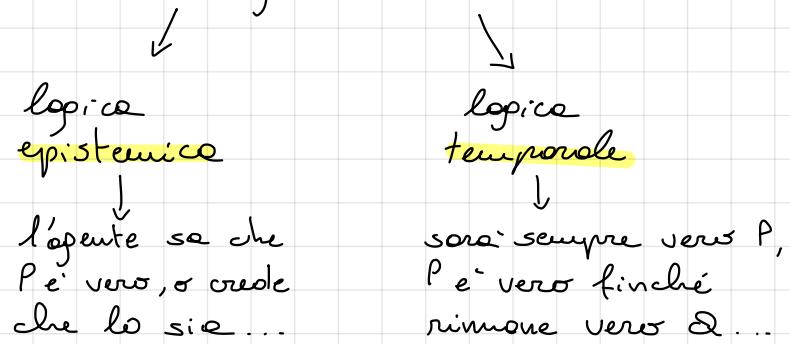
→ semantica: interpretazione regole ben formate

Linguaggio logico: linguaggio formale con un sistema di inferenza.
 ↳ serve a dedurre verità nuove.

L logici classici



L logici moderni



• Logica proposizionale

- Elemt. base: simboli proposizionali → T o F
 ↳ atomo non strutturato

¬, ∨, ∨, →, ≡
 ↑

- Proposizioni → ottenute componendo simboli proposizionali con connettivi.

NB: un letterale è un atomo P o la sua negazione $\neg P$

Sintassi: dato un insieme di simboli $P \neq \emptyset$ di s. prop., l'insieme $P[P]$ delle formule proposizionali su P è definito da:

- ogni simbolo prop. è una formula
- T vero e I falso sono formule
- Se A è formula, anche $\neg A$ lo è
- Se A e B sono formule, anche $(A \vee B)$ lo è
 \uparrow
 + altri

Semantica. stabilisce il significato dei connettivi logici

- ragionamento corretto: premesse vere → conclusione vera
- simboli prop. → il loro significato può variare
- interpretazione di un insieme di simboli prop. P : $I: P \rightarrow B$ $B = \{F, T\}$
 è la funzione che determina come interpreteatore singolarmente i simboli prop.

Def: **interpretazione** \rightarrow data una int. I di P si può definire l'interpretaz. G_I di una formula in $P[P]$

$$G_I(A) = I(A)$$

$$G_I(T) = T, \quad G_I(\perp) = F$$

$$G_I(\neg A) = F \text{ se } G_I(A) = T \in T \text{ altrimenti}$$

$$G_I(A \wedge B) = T \text{ se } G_I(A) = T \text{ e } G_I(B) = T \in F \text{ altrimenti}$$

$$G_I(A \vee B) = F \text{ se } G_I(A) = F \text{ e } G_I(B) = F \in T \text{ altrimenti}$$

$$G_I(A \rightarrow B) = T \text{ sse } G_I(A) = F \text{ o } G_I(B) = T$$

$$G_I(A \equiv B) = T \text{ sse } G_I(A) = G_I(B) = T \text{ o } G_I(A) = G_I(B) = F$$

[Data una interpretazione I e una formula $A: A \models I$ (A è vera in I) sse $G_I(A) = T$]

(soddisfa)

[Una interpretazione I è **modello** di A sse $I \models A$]

A è **soddisfacibile** sse esiste un modello di A

Tautologia: A è logicamente valido sse \forall interpretazione $I, I \models A$

Contraddizione: A è insoddisfacibile sse non esistono modelli di A

$\left\{ \begin{array}{l} A \text{ è tautologia sse } \neg A \text{ è una contraddizione} \\ A \text{ è una contraddizione sse } \neg A \text{ è una tautologia} \\ A \text{ è una contraddizione sse } A \text{ è non è soddisfacibile} \end{array} \right.$

Equivalenze logiche: A e B sono logicamente equivalenti $(A \leftrightarrow B)$ sse $\models (A \equiv B)$
 $\hookrightarrow \forall$ interpr. I di $A \equiv B: I \models A$ sse $I \models B$

NB: **assorbimento** : $A \vee (A \wedge B) \leftrightarrow A, A \wedge B \leftrightarrow A$

definibilità di \equiv : $A \equiv B \leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$

Modello: M è un modello di formula S sse è modello di ciascuna formula in S

A è una conseguenza logica di un insieme di formule $S (S \models A)$, sse ogni modello di S è un modello di A

- **Ragionamento corretto**: delle ipotesi S conclude ad A è corretto sse $S \models A$

Per verificare se $\{A_1, \dots, A_n\} \models B$ si può:

scrivere la tabella
delle verità e vedere
se è tautologia

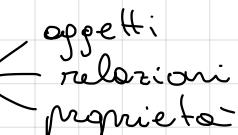
creare una interpretazione
che non sia un modello

dimostrare B e partire
da A_1, \dots, A_n usando
un insieme di regole
di ragionamento corretto

Se si trova una interpretazione I tc $I \models A_1, \dots, I \models A_n$ e I non è modello di B , non
è vero che delle ipotesi $\{A_1, \dots, A_n\}$ si può concludere B .

• Logica dei predicati

Describe il mondo



N.B.:
Ha elevata potenza espressiva,
e volte troppo, conviene ridurne
la potenza.

- Sintassi

- Simboli logici:

- variabili (x, y, z)
- connettivi proposizionali \top, \perp
- quantificatori \forall, \exists
- simboli separatori

- Simboli non logici:

- insieme numerabile $\neq \emptyset$ es $=^2, P^n, Q^m$
- costanti (a, b, c)
- simboli di funzione f^n, g^m

+ Ogni variabile è un termine

+ Se t_1, \dots, t_n sono termini e f^n simbolo di funz. $\rightarrow f(t_1, \dots, t_n)$ è termine
+ $\text{vars}(t)$ è l'insieme delle variabili che occorrono in un termine t .
+ t è chiuso (ground) se $\text{vars}(t) \neq \emptyset$

Le formule ben formate vengono costruite così:

- P predicato a n argomenti e t_1, \dots, t_n termini $\rightarrow P(t_1, \dots, t_n)$ è formula
- \top e \perp sono formule
- Allora $\neg A$, vale con tutti i connettivi $\equiv, \wedge, \vee, \rightarrow \Rightarrow$ sono formule
- Se A formula e x variabile $\forall x.A, \exists x.A$ sono formule.
- Nient'altro lo è. NIENTEEHH!!!

Variabili libere e vincolate:

- Quantificatore \rightarrow ha campo d'azione (scope) \rightarrow la sua variabile è legata solo alla formula seguente
- Occorrono vincolate di var $\rightarrow x$ è la var di un quantific. oppure occorre nello scope del quantific.
- Occorrenza libera di $x \rightarrow$ non è vincolata
 $\Leftrightarrow x$ è libera in A sse ha almeno una occorrenza libera in A .
- Formula chiusa \rightarrow formula senza var. libere

Sostituzione di variabili:

Se A è formula
 x variabile
 t termine } - $A[t/x]$ è la formula che si ottiene sostituendo
ogni occorrenza libera di x con t .
- $A[t_1/x_1, \dots, t_n/x_n]$ è la formula che si ottiene sostituendo simultaneamente tutte le variabili

- Semantica:

Sia L linguaggio con C costanti e F simboli di funz e P simboli pred.

Interpretazione di L :

Insieme non vuoto D (dominio)

Funzione di interpretazione che associa:

ad ogni $c \in C$ un elemento $I(c) \in D$

ad ogni $f^n \in F$ una funzione $I(f^n) : D^n \rightarrow D$

ad ogni $p^n \in P$ una relazione n -aria su D , $I(p) \subseteq D^n$

$I(=) = \{ \langle d, d \rangle \mid d \in D \}$

Interpretazione di una formula A : interpretazione di qualsiasi linguaggio che contiene tutti i simboli non logici di A .

→ interpretazioni dei termini \langle chiusi con var in \times p 98 + scord 99

- + $M \models A$ sse \forall assegnazione s su M , $(M, s) \models A$
 - + A è falsa ($M \not\models A$) in una interpretazione M sse nessuna assegnazione lo soddisfa
 - + Una formula può non essere né vera né falsa in una interpretazione M
- NB: vengono anche le conseguenze logiche
- soddisfribilità, validità, equivalenti logici p 101, 103, 104

• Deduzione automatica

→ **Forward chaining**: data una base di conoscenza KB , quali affermazioni (rule driven procedure) seguono? Operazione tell, usata quando viene aggiunto un nuovo fatto a KB .

→ **Backward chaining**: data una base di conoscenza KB e un A è vero che ($goal$ driven procedure) $KB \models A$? Si usa per rispondere a domande.

refutazione: $S \models A$ sse $S \cup \{\neg A\}$ insoddisfacibile

• Metodo di risoluzione:

Sistema di inferenza con una sola regola: quella di **risoluzione**
Si applica su \vdash logica delle proposizioni
logica dei predicati

Lavorare sulle **clausole**, che sono disgiunzioni di letterali.

• Risoluzione proposizionale

Si applica a clausole $L_1 \vee L_2 \vee \dots \vee L_n$ dove $L_i = p \vee \neg p$
rappresentate dai suoi letterali $L_1 \cup L_2 \cup \dots \cup L_n \Rightarrow \{L_1, \dots, L_n\}$

Date due clausole la regola di risoluzione ottiene una terza clausola:
 $k_1 \in k_2$ con $k_1 = C_1 \cup \{p\}$ e $k_2 = C_2 \cup \{\neg p\}$ allora la regola di risoluzione
genera $k_3 = C_1 \cup C_2$

$$\frac{C_1 \cup \{p\} \quad C_2 \cup \{\neg p\}}{C_1 \cup C_2}$$

Teo:

Se C è un risolvente di C_1 e C_2 allora $\{C_1, C_2\} \models C$

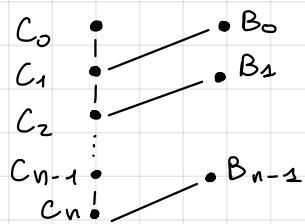
Il sistema di ris. prop. è corretto $S \vdash_R C \Rightarrow S \models C$

• Dimostrazione per refutazione

$S \models A$ se $S \cup \{\neg A\}$ è insoddisfacibile \rightarrow dim. $S \models A$ si refuta $S \cup \{\neg A\}$ cioè
si dimostra che $S \cup \{\neg A\} \vdash \perp$

\vdash per dimostrare una affermazione basta negarla e cercare di ottenere \perp
mediante l'applicazione delle regole di risoluzione.

• Risoluzione lineare



C_0 clausola iniziale

C_i clausole centrali

B_i clausole laterali

$\forall i = 0, \dots, n-1$

$B_i \in S$ oppure $B_i = C_j$
per qualche $j < i$

La risoluzione lineare evita la ridondanza dovuta alla
risoluzione di conclusioni intermedie con altre
conclusioni intermedie.

NB: Se $S = S' \cup \{C\}$ con S' soddisfacibile allora S è insoddisfacibile se
 \exists una refutazione lineare di S con C come clausola iniziale

• Clausole unitarie

clausola \rightarrow unitaria se contiene un unico letterale

risoluzione \rightarrow con clausole unitarie se ogni risolvente è unitario

• PROLOG

- Formato da un insieme di clausole di Horn o definite
- Un goal si dimostra per refutazione mediante SLD \rightarrow selective linear definite clause

A
A $\vee \neg B_1 \vee \dots \vee \neg B_n$ A :- B₁, ..., B_n } clausole definite

Il prolog è un linguaggio per supportare la risoluzione dei problemi mediante ricerche ma c'è anche: dichiarativo, strumento per dimostrare teoremi in automatico, generatore di modelli ecc.

Programma composto da:

fatti \rightarrow affermazioni sempre vere

regole \rightarrow

gli oggetti nei predicati possono essere \swarrow atomici
 \searrow strutturati

Query: caricato un programma si possono porre domande (goal) interattivamente

Per rispondere, Prolog:

- cerca un fatto che unifica il goal (alto \rightarrow basso) e se trovato, soddisfatto
 - \hookrightarrow sost vor che rende uguali fatto e goal
- cerca una regola le cui teste (parte sx) unificano il goal. Se le trova, cerca di soddisfare i sotto-goal di cui è composta la regola

NB: una domanda può avere più risposte, un goal soddisfatto in più modi

• Le liste

Possono essere usate nei predicati:

[] lista vuota

[a, b, c] una lista

che equivale a .(a, .(b, .(c, [])))

[a | T] è una lista il cui primo elem. (testo) è a e il resto è T

NB: possono essere implementate con liste concatenate!

127 131
zebra Barone

• cut,

Prolog permette di intervenire sulle ricerche: trovata una soluzione parziale si può dire di cercare la soluzione estendendo (mai riducendola).

Un cut (!) in una regola impedisce di riprovare i predicati prima del

cut delle regole. Inoltre impedisce che una regola con lo stesso predicato nella testa venga provata come alternativa alla regola corrente

green cut: le soluzioni non cambiano, solo le performance

red cut: le soluzioni cambiano, dipendono dalle presenze del cut



Per approfondimenti sul Prolog vedi → Prolog tutorial

↳ codice prolog collezionato

o Problemi di soddisfacimento di vincoli

- Problema di ricerca → lo stato è opaco

1 problemi di soddi. di vincoli (CSP) possono essere visti come specifici problemi di ricerca.

Un CSP è definito per:

- un insieme finito di variabili $V = \{x_i\}$
- un insieme finito di domini $\text{dom}(x_i)$, uno per ogni variabile
- un insieme finito di vincoli, relazioni definite fra variabili

unari → una var

Un CSP può avere zero, una o più soluzioni

binari → due var

globali → più di due var

Una soluzione di un CSP è una associazione tra ogni var del CSP e un valore nel rispettivo dominio tc tutti i vincoli siano soddisfatti

Non importa tanto le soluzioni, ma opportune restrizioni dei domini che contengono almeno una soluzione.

es:

$$V = \{x_1, x_2\}$$

$$\text{dom}(x_1) = [1 \dots 100], \text{dom}(x_2) = \text{dom}(x_1)$$

$$x_1 \leq 5, x_1 + x_2 = 10$$

$$1^{\circ} \text{ vincolo} \rightarrow \text{dom}(x_1) = \{1, 2, 3, 4\}$$

$$2^{\circ} \text{ vincolo} \rightarrow x_2 = 10 - x_1$$

$$\text{Quindi } 3^{\circ} \text{ vincolo } \text{dom}(x_2) = \{6, 7, 8, 9\}$$

Il CSP ammette 16 soluzioni

$$x_1 = 1, x_2 = 6$$

$$x_1 = 2, x_2 = 6$$

$$x_1 = 1, x_2 = 7$$

$$x_1 = 2, x_2 = 7$$

$$x_1 = 1, x_2 = 8$$

$$x_1 = 2, x_2 = 8$$

$$x_1 = 1, x_2 = 9$$

$$x_1 = 2, x_2 = 9$$

Una soluzione di un CSP è  completa: associa ogni var ad un valore
consistente: rispetta tutti i vincoli

Una assegnazione è una associazione tra alcune var del CSP e un valore nel rispettivo dominio tc alcuni vincoli siano soddisfatti

Soluzione  parziale: se coinvolge solo alcune variabili
inconsistente: se non rispetta tutti i vincoli

• Tipi di CSP:



• Grafo dei vincoli:

CSP

```

    CSP --> unario -> coinvolge solo vincoli unari
    CSP --> binario -> coinvolge solo vincoli binari
    binario --> possibile definire un grafonon orientato dei vincoli:
      - nodi -> variabili
      - archi -> vincoli
  
```

es:

Vars: TWO FOUR

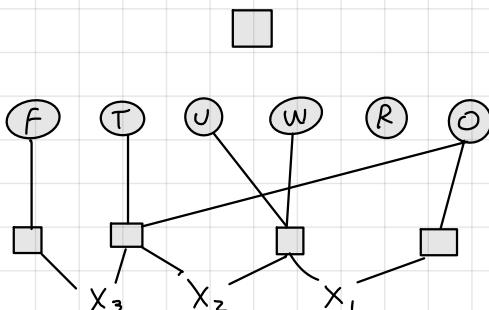
X_1, X_2, X_3

$$\begin{array}{c} \text{TWO} \\ + \quad \text{TWO} \\ \hline \text{FOUR} \end{array}$$

Domini: tutti uguali a $[0 \dots 9]$

Vincoli:

- $O + O = R + 10X_1$
- $X_1 + W + W = U + 10X_2$
- $X_2 + T + T = O + 10X_3$
- $X_3 = F, T \neq O, F \neq O$
- all different (F, T, U, W, R, O)



Questo è un caso semplice. Un caso reale sarebbe l'applicazione di risoluzione per un sudoku.

• CSP nei casi reali

- Problemi di allocazione risorse:

planificazione attività personale, delle produzioni, delle visite dei corrieri ecc.

- Realizzazione negozi virtuali

- Progettazione circuiti integrati digitali

Risoluzione mediante ricerca:

Per risolvere un CSP si può impostare un problema di ricerca:

- generate and test

- ① Gli stati di ricerca sono assegnamenti parziali, partendo da uno vuoto.
- ② La funz. successore estende l'assegnamento dando un valore a una var libera (non ancora assegnata)
- ③ Goal test → assegnamento completo e consistente?

es:

```
void risolviGT (int indice)
if (indice == n) {
    if (vincoliNonViolati())
        soluzioneTrovata ();
} else {
    if (vor [indice] == vuota) {
        for (int i=0; i < d; i++) {
            vor [indice] = dom [i];
            risolviGT (indice + 1);
        }
        vor [indice] = vuota;
    } else
        risolviGT (indice + 1);
}
```

- standard backtracking

- ① Gli stati della ricerca sono gli assegnamenti parziali, partendo da un nullo.
- ② La funz. successore assegna un valore ad una var non ancora assegnata così da estendere l'assegnamento in modo consistente
- ③ Goal test → assegnamento consistente?

es: void risolviSBT (int index) {
 if (index == n)
 soltrrovaato ();
 else {
 if (vor [index] == empty)
 for (int i=0; i < d; i++) {
 vor [index] = dom [i];
 if (vincoliNonViolati ())
 risolviSBT (index + 1);
 }
 }
}

NB: Se il CSP ha n variabili le soluzioni si trovano a profondità n .

→ ricerca in profondità +

Per ogni nodo dello albero si opera un solo assegnamento

```

vor[index] = empty;
} else risolu BST(index+1);
}

```

Lo standard backtracking può essere migliorato usando euristiche generali.

- scelta della prossima variabile da assegnare
- scelta del prossimo valore da assegnare
- identificazione anticipate di fallimenti
- scelta di quali vor de liberare in caso di backtracking

• Propagazione dei vincoli

Per identificare i casi di fallimento ASAP \rightarrow tenere traccia dei valori ammissibili per ogni variabile con ammissibile = elenzi su dom. non ancora eliminati.
 \downarrow

assegnamento = vincolo unario aggiuntivo ($x = \alpha$) si parla di propagaz. dei vincoli

- **Forward checking**: si tiene traccia dei valori ammissibili e si opera un backtracking quando almeno una variabile ha un insieme dei valori ammissibili vuoto.

Propaga l'informazione delle variabili assegnate e quelle non assegnate, ma non identifica tutti i fallimenti.

• Consistenza di orco:

orientati! $x \rightarrow y$

Dato un CSP binario \rightarrow archi forzati a essere orco-consistenti: cioè sse \forall valore x nell'insieme dei valori ammissibili di X c'è almeno un valore ammiss. y nell'insieme dei valori ammiss. di Y .

Un orco non orientato $X - Y$ è consistente sse $X \rightarrow Y$ e $Y \rightarrow X$ sono consistenti.

N.B.: ogni volta che cambia l'insieme dei val. ammissibili di una vor x è necessario ricontrollare tutte le vor collegate.

- **Algoritmo MAC (monitoring orco consistency)**: voriente SBT. forza la orco-consistenza prima di ogni assegnamento. È parametrico rispetto al SBT e può essere combinato con le euristiche di ricerca.

N.B.: la orco-consistenza non è in grado di identificare tutte le inconsistenze.

o CSP parametrici

- Sono CSP che possono essere estesi a piacere in base ad un unico parametro
- consentono di svincolarsi dal caso singolo e poter stimare come si comporterebbero varie tecniche.

o MiniZinc :

È un linguaggio per la descrizione dei CSP
+ zinc : livello più alto
floatzinc : minivale × CSP; liv più basso

svincolato dal risolutore
standard di descrizione
CSP su int, float, bool e insiem

o Programmazione logica con vincoli: (CLP)

Prog. logico → estesa ai vincoli applicati a variabili

CLP → parametrica rispetto al dominio delle variabili
↪ CLP(D) se vor hanno dom in D es: D=B, D=Z, D=R

CLP(FD) → se le variabili hanno domini limitati, tipo numeri interi.

o Posting e Labeling

Date una o più vor → possiamo aggiungere un vincolo → op. di post

possiamo richiedere di assegnare un val. ammesso
a ogni vor. → op. di labeling delle variabili

o Agenti che risolvono problemi mediante ricerche

o Agenti e problemi

agente → ha dei goal → è un insieme di caratteristiche del mondo che l'agente vorrebbe fossero vere

comple azioni sul mondo per raggiungere i suoi goal

decidere quale azione compiere viene risolto per ricerca

modo generale di risoluzione (per quali azioni compiere)

- parametrico → rispetto alle descrizioni del problema
- che consenta di risolvere in modo ottimale, se possibile
- che trovi una buona soluzione se quella ottimale non c'è

Uno delle più usate è il GPS (general problem solver)

L'agente → lavora cercando nello spazio delle sol. possibili
(opportunistamente estese portano a soluzioni complete)

→ esplora lo spazio delle sol. parziali fino alla soluzione
(può scartarne una × trovarne una migliore)

Use le sol trovate × generare la seq. di azioni che portano il mondo da stato iniziale a stato di goal

Problema: descritto da quadruplo $P \langle S_i, S_o, M, F(c) \rangle$

- stato iniziale
- funzione successore $S(a)$ dato una azione a , calcola lo stato successivo
- un modo per verificare se uno stato è un goal
- una funz. di costo $c(x, a, y) \geq 0$ A stato corrente x , azione a e stato futuro y

A seq. di azioni viene definita una f. di costo come somma dei costi delle azioni

o Alberi di ricerca

Albero → descrive le tecniche con cui si ricerca una soluzione

↳ non serve che il problema sia una ricerca in un albero esistente

↳ A sol. parziale si genera un nodo

Uno stato rappresenta una configurazione del mondo

In nodo è una punte dell'albero di ricerca che include

stato
 padre
 azione
 profondità
 costo percorso

Una strategia viene definita dall'ordine in cui vengono espansi i nodi

Parametri della strategia

- completezza: una strategia è completa se trova sempre una soluzione, se esiste
- complessità spaziale: misurata in termini di numero massimo di nodi tenuti in memoria
- complessità temporale: misurata in termini di nodi generati
- ottimalità: nei termini della funzione di costo del percorso

Complessità misurate in termini di

- b: branching factor max dell'albero
- d: min. profondità di sol. e costo minimo
- m: max. profondità dell'albero (∞)

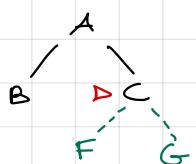
NB: una ricerca non informata usa solo info disponibili nella descrizione del problema.

Ricerca

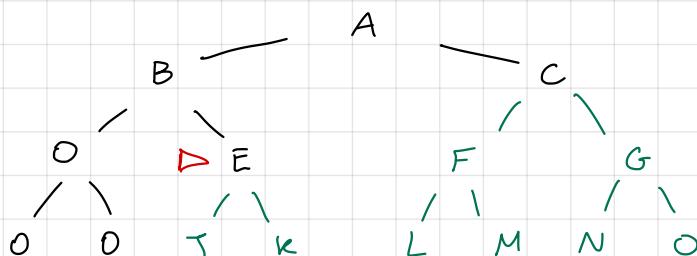
- in ampiezza
- in profondità
- e profondità limitata ad approfondimento

1
2
3
4

① Viene espanso un nodo non ancora espanso a profondità min fra quei mantenute in coda FIFO



② Viene espanso un nodo non ancora espanso a profondità max fra quei mantenute in coda LIFO



(3) vedi slide n. 200

(4) Si prova a risolvere usando una profondità fissata; se non si trova la soluzione, si riporta usando una profondità maggiore + slide 201

	①	③	②	④
Confronto:	Breadth First	Depth First	Depth Limited	Iterative Deepening
Completo?	Yes	No	No	Yes
Time comp.	$O(b^d+1)$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space comp.	$O(b^d+1)$	$O(bm)$	$O(bl)$	$O(bd)$
Optimale?	Yes	No	No	Yes

• Ricerca informata

Estende la descrizione del problema con informazioni che aiutano a guidare le ricerche

best first → il nodo funz. di valutazione $f(n)$ → stima "bontà" del nodo
→ espansi prima i nodi con $f(n)$ più basso

greedy best first → funz. euristica $f(n) = h(n)$

↓
→ stima il costo del percorso dal nodo n più vicino al goal
↳ viene espanso il nodo che sembra migliore
(val $f(n)$ più basso)

Non è completa

$O(b^m)$ comp. temp.

$O(b^m)$ comp. spaziale

Non è ottimale

A^* → non conviene espandere i nodi a costo elevato $f(n) = g(n) + h(n)$

con $g(n) = \text{costo} \times \text{raggiungere il nodo } n$

$h(n) = \text{stima costo da } n \text{ a goal più vicino}$

$f(n) = \text{stima costo percorso che posso per } n$

A^* espande i nodi in ordine crescente di $f(n)$

il contorno i contiene tutti i nodi con $f = f_i$ e $f_i < f_{i+1}$
aggiunge contorni gradualmente

teorema (ottimalità di A^*) \rightarrow se $h(n)$ è ammessa, allora le procedure
TREE-SEARCH con strategia A^* è ottima

$A^* \rightarrow$ è completa, è ottimale, c. temp esponenziale $O(b^m)$
c. spaz esponenziale $O(b^m)$

o Euristica ammessa

Euristica \rightarrow amm. se \forall nodo $n \rightarrow h(n) \leq h^*(n)$ con $h^*(n)$ costo per
raggiungere da nodo n uno tra i goal più vicini.

NB: una euristica ammessa non sovrastima mai il costo \times goal

o Ricerca e pianificazione

problem solving mediante ricerca \rightarrow base STRIPS (stanford res. inst. prob. solv.)

sistema con specifico
linguaggio di descrizione
dell'ambiente

→ date una descrizione dell'
ambiente attuale e in
grado di trovare una seq.
di azioni che portano a goal

Planning \rightarrow caso particolare di problem solving mediante ricerca

Il pianificatore \rightarrow può iniziare la ricerca da uno stato qualsiasi
 \rightarrow assume che il mondo sia diviso in varie parti fra loro indipendenti
 \downarrow
gli stati sono rappresentati in STRIPS tramite proposizioni \leftrightarrow fatti

NB: closed world assumption \rightarrow fatti non elencati nella rappresentazione di
uno stato \rightarrow assunti come falsi

Goal \rightarrow insieme di proposizioni

Stato \rightarrow soddisfa goal se contiene tutte le sue proposizioni

o Azioni: rappresentate in STRIPS mediante quadruple

- nome che le identifica
- insieme di precondizioni: fatti che devono far parte dello stato x azione applicabile
- add list: fatti che saranno veri dopo che l'azione e' stata eseguita
- delete list: fatti che non saranno più veri dopo che l'azione e' stata eseguita

Gli unici effetti delle azioni sono dati da add e delete list
C'è rischio in situazioni reali → non so l'effetto delle azioni

↓
Ai frame problem → ciò che non e' menzionato nella descrizione delle azioni e' supposto invertito

- proposizioni tipo $on(A, B)$ sono $\begin{cases} \text{predicati (on)} \\ \text{oggetti: } (A, B) \end{cases}$]>pianificatori proposizionali

- per semplificare la descrizione si usano schemi di azioni

- putDown(x, y)
| - precondizioni { holding(x), clear(y) }
|
| - addlist { on(x, y), handEmpty, clear(x) }
|
| - delete list { holding(x), clear(y) }

o Problemi di pianificazione → descritto in STRIPS come:

- stato iniziale s
- goal G
- insieme di azioni che portano al goal

STRIPS cerca una sequenza di azioni che porta al goal

↓
pianificatori lineari → portano dallo stato iniziale e cercano di raggiungere lo stato di goal → pianificatori forward chaining

le ricerche di goal e stato iniziale riduce il tempo di ricerca

→ pianificatori backward chaining

• Means-end analysis:

È una tecnica di decomposizione dei problemi che, partendo dal risultato che si vuole ottenere, cerca le azioni che lo producono.

↳ backward chaining è un esempio

Vengono selezionate solo azioni che possono produrre il goal corrente e vengono generati sotto goal corrispondenti e stati che verificano le precondizioni

↓
Si applica l'algoritmo ai singoli sotto goal → alg. termina se una precondizione è falsa o se è vera o se è impossibile renderla vera

o Ricerca con avversari

Studiare i giochi → studiare perché offrono criteri formalizzati, problemi complessi e modi per studiare ambienti multi-ogente
↳ è correlato alla teoria dei giochi

giochi vs problemi di ricerca

1. gli avversari rendono l'ambiente impredicibile
2. i giochi hanno vincoli di tempo legati a mosse non stocastico
3. per i giochi è necessario trovare strategie approssimate

Ridurre i giochi a problemi di ricerca approssimate è un buon modo per affrontarli
giochi perfetti: 2 players, turni alterni, somma zero, info a tutti i players, no casualità, mosse finite, solo un risultato

ogente che gioca: considerare tutte le mosse possibili a ogni turno → porta ad una Strategia ottima.

I problemi sono: rappresentare lo stato di gioco
generare la lista delle mosse possibili
assegnare un valore di qualità ad ogni mossa

albero di gioco: usato per rappresentare gli stati futuri

↓
la radice è lo stato attuale

ogni mossa è un arco che collega lo stato attuale ad uno nuovo
le foglie corrispondono agli stati finali del gioco

→ serve una funz. di valutazione che assegna un valore di qualità a nodo

Mio turno: radice tag min o max → ogni livello ha nodi tag tutti: max o min
↳ livelli diversi sono tag tutti: max o min a turno $i+1$

L'albero può essere completo → contiene tutti gli stati fino a tutte le foglie
incompleto → si ferma ad una certa profondità
(prefissata)

o Algoritmo Minimax:

E' una ottima strategia per i giochi perfetti. L'idea e' di determinare le mosse di MAX che massimizzano il suo punteggio garantito.

- punteggio ottenuto nel caso di partita vs giocatore ottimo \rightarrow caso peggiore
- punteggio indipendente dalle mosse di MIN
- il valore di ogni nodo e' determinato sulla base del valore dei suoi figli

Il valore di un nodo

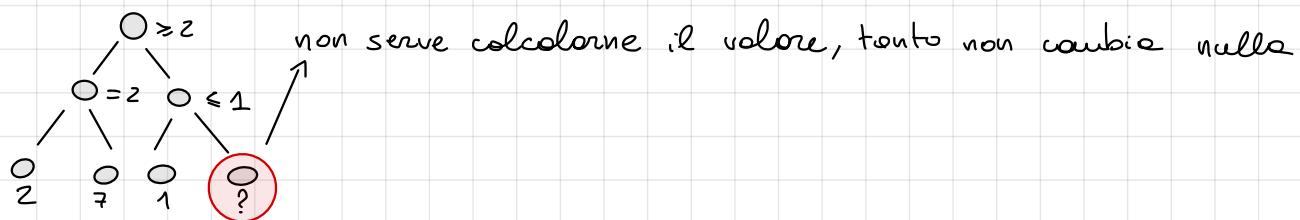
per un nodo MAX e' ottenuto dal val massimo dei suoi figli

per un nodo MIN e' ottenuto dal val minimo dei suoi figli

1. Crea nodo radice top MAX con stato corrente della partita
2. Espandi fino alle foglie o profondita' max prefissata (come look-ahead)
3. Applica funzione di valutazione ad ogni nodo foglie
4. Calcola il valore di ogni nodo intermedio portando dalle foglie
5. Scegli una delle mosse che consentono di massimizzare il val sulla radice

o $\alpha - \beta$ Pruning

Per migliorare le prestazioni di minimax senza appesantire



\rightarrow si attraversa l'albero di gioco in profondita'

- o A ogni nodo Max n: $\alpha(n) = \max$ val trovato
 - inizia con $-\infty$ e incrementa soltanto
 - incrementa se un figlio di n genera $\text{val} >$ del corrente $\alpha(n)$
 - e' un lower bound del val finale
- o A ogni nodo Min n: $\beta(n) = \min$ val trovato
 - inizia con $+\infty$ e decremente soltanto
 - decrementa se un figlio di n genera $\text{val} <$ del corrente $\beta(n)$
 - e' un upper bound del val finale

In più: $\beta(n) \times$ un nodo MAX n e' il più piccolo β dei nodi MIN che lo precedono

$\alpha(n) \times$ un nodo MIN n e' il più grande α dei nodi MAX che lo precedono

Si portano α e β in basso durante la ricerca

α cambia solo a nodi MAX

β cambia solo a nodi MIN

Il pruning si fa quando $\alpha \geq \beta$:

- **α cutoff**: dato un nodo MAX se $\alpha \geq \beta$ non si generano ulteriori figli

- **β cutoff**: dato un nodo MIN se $\alpha \geq \beta$ non si generano ulteriori figli

Proprietà del α - β pruning:

- non introduce approssimazioni

- l'ordine delle valutazioni influenza le performance \rightarrow ottimo $O(b^{m/2})$
↳ caso ottimo: ai nodi max il figlio con val maggiore è generato x prius
ai nodi min il figlio con val minore è generato x prius

Funzione di valutazione:

Serve per interrompere la ricerca (es. vincoli di tempo)

→ quantificare la bontà di uno stato del gioco

- non è euristico, introduce approssimazioni

- se il gioco è a somma zero ne basta una

$f(n) > 0$ stato buono per MAX

$f(n) < 0$ stato buono per MIN

$f(n) \approx 0$ stato neutro

$f(n) \gg 0$ MAX ha praticamente vinto

$f(n) \ll 0$ MIN ha praticamente vinto

es: per il tris si basa sulle triplette - riga / colonna / diagonale complete

interessa il # di triplette possibili:

$$f(n) = (\# \text{ trip. poss. MAX}) - (\# \text{ trip. poss. MIN})$$

Di solito le f. di val. si esprimono come somma pesata di feature

$$f(n) = w_1 \text{feature}_1(n) + \dots + w_k \text{feature}_k(n)$$

o I CSP nel dettaglio

CSP \rightarrow triple $\langle V, D, C \rangle$ con $V \neq \emptyset$ variabili $n = |V|$
 $D \neq \emptyset$ domini delle variabili $|D| \leq n$
 C vincoli

dom: $V \rightarrow D$ funz. suriettiva totale che associa un dom ad ogni variabile.
Ogni var ha il suo dominio. Vor sempre in ordinamento crescente.

dato un CSP $P \rightsquigarrow \text{dom}(P) = \prod_{x \in V} \text{dom}(x)$ V insieme vor.

Ogni vincolo $c \in C$ è una coppia $\langle V_c, \Delta_c \rangle$ con $V_c \subseteq V$ insieme vars CSP.
e $\Delta_c \subseteq \prod_{x \in V_c} \text{dom}(x)$

Δ_c viene detto insieme degli assegnamenti consistenti del vincolo c .

Se $|V_c|=1 \rightarrow$ allora c è unario \rightarrow binario, negli altri così c è globale
 $|V_c|=2 \rightarrow$

$\text{vars}(c) = V_c$ è riferimento alle variabili di un vincolo

$\text{dom}(c) = \prod_{x \in V_c} \text{dom}(x)$ è il dominio del vincolo c .

imp: $C \rightarrow \Delta$ funz. suriettiva totale che associa ad ogni vincolo l'estensione
a Δ dell'insieme degli assegnamenti consistenti, $\text{imp}(C)$ è
l'insieme degli assegnamenti totali consistenti.

esempio 1

CSP con 3 vars: x, y, z t.c. $\text{dom}(x) = \text{dom}(y) = \text{dom}(z) = [1 \dots 6]$
insieme variabili $V = \{x, y, z\}$ dom CSP = $\Delta = [1 \dots 6]^3$
vincolo $c \rightarrow x \neq z$ ed entrambi pari

$c = \langle V_c, \Delta_c \rangle$ dove $V_c = \{x, z\} \subset \Delta_c$ espresso in forma estensionale
 $\Delta_c = \{(2,4), (2,6), (4,6), (4,2), (6,2), (6,4)\}$

$\Delta_c = \{(x, z) : x \in 1 \wedge z \in 1 \wedge x \neq z \wedge x \bmod 2 = 0 \wedge z \bmod 2 = 0\}$

$\text{imp}(c) = \{(x, y, z) : x \in 1 \wedge y \in 1 \wedge z \in 1 \wedge x \neq z \wedge x \bmod 2 = 0 \wedge z \bmod 2 = 0\}$

$\text{sol}(P) = \bigcap_{c \in C} \text{imp}(c)$ e Prisolubile se $\text{imp}(C) \neq \emptyset$

$P_1 = P_2$ equivalenti se $\text{sol}(P_1) = \text{sol}(P_2)$

L'idea è di generare trasformazioni di CSP equivalenti ma più utili per la ricerca di soluzioni.

• Logica dei predicati

Escuipio:

Iniziamo con un esempio: "I numeri primi maggiori di due sono dispari".

- aspetti: numeri, due.
 - relazioni e proprietà: essere un numero primo, dispori, maggiore di (rel. a due argomenti).

Per ogni oggetto x , se x è un numero primo ed è maggiore di due, allora x è dispari.

\forall -> quantificatore universale \exists -> quantificatore esistenziale
 \rightarrow simbolo di implicazione
 \neg -> simbolo di negazione
 \wedge -> simbolo di connessione logica
 \vee -> simbolo di connessione logica
 \rightarrow simbolo di implicazione
 \leftrightarrow -> simbolo di connessione logica
 \exists -> quantificatore esistenziale
 \forall -> quantificatore universale

$\forall x (\text{primo}(x) \wedge \text{maggior(x, 2)} \rightarrow \text{dispon}(x))$

Sintassi:

1. Simboli logici (comuni x tutti i ling. del primo ordine)
 - Variabili x, y, z
 - Connettivi proposizionali \perp, T
 - Quantificatori \forall, \exists
 - Simboli separatori $(, ,$
 2. Simboli non logici (specifici del linguaggio)
 - Simboli di predicato (con orite $p^n, q^m\dots$)
 - Costanti (a, b, c, \dots)
 - Simboli funzionali (con orite $f_1^{n_1}, f_2^{n_2}, \dots$)

Termini:

Una formula atomica rappresenta il fatto che una delle relazioni vale fra alcuni oggetti: \rightarrow maggiore (z, o)

Def. induit. \rightarrow ogni variabile, costante e funzione.

Se t_1, \dots, t_n sono termini e f^n è un simbolo funzionale allora $f(t_1, \dots, t_n)$ è un termine.

Termini del linguaggio \rightarrow classi di espressioni con le quali si possono denotare gli oggetti.
I termini chiusi sono termini senza variabili.

Formule:

1. Se P è un simbolo di predicato a n argomenti e t_1, \dots, t_n sono termini, allora $P(t_1, \dots, t_n)$ è una formula.
2. L e T sono formule atomiche.
3. Se A è una formula, anche $\neg A$ lo è.
4. Se A e B sono formule, anche $A \wedge B$, $A \vee B$, $A \rightarrow B$, $A \equiv B$ lo sono.
5. Se A è formula e x variabile, allora $\forall x A$, $\exists x A$ lo sono.
6. Nient' altro è una formula

Esempi:

fratello (carlo, obele), \neg fratello (gesù, giude)

$\forall x (\text{fratello}(x, \text{obele}) \rightarrow \text{assassino}(x))$

$\exists x (\text{fratello}(x, \text{obele}) \wedge \text{assassino}(x))$

$\forall x \forall y (\text{fratello}(x, y) \equiv \exists z (\text{padre}(z, x) \wedge (\text{padre}(z, y)))$

Variabili libere e vincolate:

Scope di un quantificatore: di $\forall x$ in $\forall x A : A$

di $\exists x$ in $\exists x A : A$

$$p(c) \wedge \forall x (q(x, c) \rightarrow \exists y (p(y) \vee r(x, y, z))) \equiv \exists x \forall y q(x, y)$$

scope $\forall x$ scope $\exists y$ scope $\forall y$

vincolate di var: x è la var. di un quantific. o del suo scope
occorrenze <
libere di x : occorrenze non vincolate

una formula è chiusa se è senza variabili libere

NB: var libere \approx var globali
var vincolate \approx var locali

Sostituzioni di variabili con termini:

Se A è una formula, t un termine e x una variabile : $A[t/x]$ denota la formula che si ottiene da A sostituendo ogni occorrenza LIBERA della variabile x con il termine t .

Sostituzione simultanea : $A[t_1/x_1, \dots, t_n/x_n]$

$$p(x, y)[f(y)/x, f(x)/y] = p(f(y), f(x)).$$

$$p(x, y)[f(y)/x][f(x)/y] = p(f(y), y)[f(x)/y] = p(f(f(x)), f(x))$$

$$p(x, y)[f(x)/y][f(y)/x] = p(x, f(x))[f(y)/x] = p(f(y), f(f(y)))$$

Solo le occorrenze libere vengono sostituite

$$\begin{aligned} p(x, c) \wedge \forall x (q(x, y) \rightarrow \exists y r(y)) [t_1/x, t_2/x] \\ = p(t_1, c) \wedge \forall x (q(x, t_2) \rightarrow \exists y r(y)) \end{aligned}$$

Semantica :

$\exists x p(x) \vee \exists y q(y) \rightarrow \exists z (p(z) \wedge q(z))$ è vero o falso?

Dove? dominio $\rightarrow D = \{1, 2\}$

Interpretazione p e q : $\begin{array}{ll} p(x) \text{ pari} & \text{estensione } p \in \{2\} \\ q(x) \geq 1 & \text{estensione } q \in \{1, 2\} \end{array}$

$\exists x p(x)$ è vero se $\exists d \in \{1, 2\} t.c. d \in \{2\}$

$\exists x q(x)$ è vero se $\exists d \in \{1, 2\} t.c. d \in \{1, 2\}$

$\exists z (p(z) \wedge q(z))$ è vero se $\exists d \in \{1, 2\} t.c. d \in \{2\} e d \in \{1, 2\}$

Logica proposizionale classica

Semantica:

Le espressioni delle logiche proposizionali sono formule. Il linguaggio proposizionale è costruito sulle basi di un determinato insieme P di variabili proposizionali.

$$P = \{ p, q, r \}$$

$$P' = \{ p_0, p_1, p_2, \dots \}$$

$$P'' = \{ \text{cervo-fratello-di-bole}, \text{porco-dio} \dots \}$$

$$\text{Alfabeto} \rightarrow P \cup \{ \neg, \wedge, \vee, \rightarrow, \equiv, (,) \}$$

Semantica:

Linguaggio \rightarrow dominio

Prop [P] \rightarrow Bool = {T, F}

Un ragionamento è corretto se ogni volta che le premesse sono vere, è vera anche la conclusione.

La semantica della lp stabilisce il significato dei connettivi logici.

Interpretazioni:

M : assegnaz. di un bool ad ogni var in P $M: P \rightarrow \text{bool}$

Esempio: se $P = \{ p, q, r \}$ ci sono 8 possibili interpretazioni di P .

	p	q	r
M_1	T	T	T
M_2	T	T	F
M_3	T	F	T
M_4	T	F	F

	p	q	r
M_5	F	T	T
M_6	F	T	F
M_7	F	F	T
M_8	F	F	F

Interpretazione di una formula F :
interpretaz. del linguaggio di F

• Bonus: domande per esame

- **Bound consistent:** i domini hanno come estremi il valore minimo ed il valore massimo che la variabile può avere, per cui tutti i valori possibili sono compresi dentro quel range.
- **Arc consistent:** deve avere un CSP binario (tutti i vincoli binari). Per entrambe le variabili coinvolte nel vincolo, a tutti i valori del loro dominio corrisponde un valore nel dominio dell'altra variabile, e questo deve valere in entrambi i sensi.
- **Classificatore binario:** è una rete neurale in grado di operare una separazione lineare, ovvero c'è una rete che: dato un vettore di input restituisce in output la classe a cui appartiene quell'input. covid test \rightarrow SLE