

Introduzione all'Intelligenza Artificiale

Alessio Delgadillo

Anno accademico 2020-2021

Indice

1	Introduzione	5
1.1	Intelligenza Artificiale: una doppia sfida	5
1.1.1	Definizioni di IA	6
1.2	Agenti Intelligenti: la visione “moderna”	8
1.3	The AI revolution?	9
1.3.1	Priorità di ricerca	10
1.4	In sintesi	10
2	Agenti Intelligenti	12
2.1	Caratteristiche degli agenti	12
2.1.1	Percezioni e azioni	13
2.1.2	Agenti razionali	14
2.1.3	Ambienti	15
2.1.4	Struttura di un agente	18
3	Agenti risolutori di problemi	22
3.1	Algoritmi di ricerca	23
3.1.1	Itinerario: il problema	24
3.1.2	Aspirapolvere: il problema (toy problem)	25
3.1.3	Il puzzle dell’otto	26
3.1.4	Le otto regine: il problema	26
3.1.5	Dimostrazione di teoremi	28
3.1.6	Problemi reali	28
3.2	Ricerca della soluzione	29
3.2.1	Ricerca ad albero	30
3.2.2	I nodi dell’albero di ricerca	30
3.2.3	Struttura dati per la frontiera	30
3.2.4	Diversi tipi di strategie (di ricerca)	31
3.2.5	Valutazione di una strategia	31
3.2.6	Ricerca in ampiezza (BF - Breadth-first)	31
3.2.7	Ricerca in profondità (DF)	33

3.2.8	Approfondimento iterativo (ID)	35
3.2.9	Direzione della ricerca	35
3.2.10	Ricerca “ad albero” / “a grafo”: cammini ciclici	36
3.2.11	Ricerca di costo uniforme (UC)	37
3.2.12	Confronto delle strategie (albero)	39
3.3	Conclusioni	39
4	Ricerca euristica	41
4.1	Funzioni di valutazione euristica	41
4.1.1	Algoritmo di ricerca Best-First	42
4.1.2	Algoritmo A: definizione	42
4.1.3	Algoritmo A*	44
4.1.4	Euristica consistente o monotona	46
4.1.5	Ottimalità di A*	47
4.1.6	Su f : Due sotto-casi speciali	48
4.2	(Costruire) le euristiche di A*	49
4.2.1	Misura del potere euristico	50
4.2.2	Come si inventa un’euristica?	52
4.3	Algoritmi evoluti basati su A*	54
5	Oltre la ricerca classica	56
5.1	Ricerca locale	56
5.1.1	Algoritmi di ricerca locale	56
5.1.2	Ricerca in salita (Hill climbing) steepest ascent/descent	57
5.1.3	Tempra simulata	60
5.1.4	Ricerca local beam	61
5.1.5	Beam search stocastica	61
5.2	Spazi continui	63
5.3	Assunzioni sull’ambiente da riconsiderare	64
5.3.1	Azioni non deterministiche	64
6	I giochi con avversario	66
6.1	Giochi come problemi di ricerca	67
6.1.1	Algoritmo Min-Max	68
6.1.2	Potatura alfa-beta	72
6.2	Problemi di soddisfacimento di vincoli	74
6.2.1	Strategie per problemi CSP	75
6.2.2	Ricerca in problemi CSP	75

7 Agenti basati su conoscenza	78
7.1 Agente basato su conoscenza	79
7.1.1 Formalismi per la R.C.	80
7.2 Agenti logici: calcolo proposizionale	80
7.2.1 L'algoritmo TT-entails	83
7.2.2 Algoritmi per la soddisfacibilità (SAT)	83
7.2.3 L'algoritmo DPLL per la soddisfacibilità	84
7.2.4 Metodi locali per SAT	86
7.2.5 Inferenza come deduzione	87
8 Agenti logici: la logica del prim'ordine	90
8.1 Linguaggio	91
8.1.1 Semantica dichiarativa	92
8.1.2 Semantica compostionale	93
8.1.3 Semantica “standard” e semantica “database”	94
8.2 Inferenza nella logica del primo ordine	95
8.2.1 Unificazione e Sostituzione	96
9 Introduzione al Machine Learning	99
9.1 Cos’è il ML?	99
9.1.1 Apprendimento supervisionato	103
9.1.2 Apprendimento non supervisionato	103
9.1.3 Modelli e rassegna di concetti utili	103
9.1.4 Algoritmi di apprendimento	104
9.1.5 Generalizzazione	105
10 Concept Learning	106
10.1 Regole congiuntive	107
10.1.1 Rappresentazione delle ipotesi	107
10.1.2 Ipotesi di apprendimento induttivo	107
10.1.3 General to Specific Ordering	108
10.1.4 Algoritmo Find-S	108
10.1.5 Versions Spaces	109
10.1.6 Algortimo Candidate Elimination	110
10.2 Bias Induttivo	110
10.2.1 Unbiased Learner	110
10.2.2 Inductive Bias	111
11 Modelli Lineari	113
11.1 Regressione	113
11.1.1 Gradient Descent	115

11.1.2 Una generalizzazione: LBE	118
11.1.3 Regolarizzazione da Ridge Regression	119
11.1.4 Limitazioni delle Fixed Basis Functions	120
11.2 Classificazione	120
12 Introduzione all'apprendimento degli alberi di decisione	123
12.1 Top-down induction of Decision Trees	123
12.1.1 Bias induttivo - DT	125
12.1.2 Overfittinng	126
12.1.3 Gestire attributi a valore continuo	127
12.1.4 Gestione dei dati di training incompleti	128
12.1.5 Gestione di attributi con costi differenti	128
12.1.6 Conclusioni	129
13 Introduzione alla convalidazione e questioni teoriche	130
13.1 Statistical Learning Theory (SLT)	132
13.1.1 Vapnik-Chervonenkis-dim e SLT	132
14 Introduzione all'uso di SVM	134
14.1 Classificatore a massimo margine	134
14.1.1 Hard Margin SVM	135
14.1.2 Soft Margin	136
14.2 Kernel	137
14.2.1 Evitare interpretazioni errate	139
15 K-NN, apprendimento senza supervisione e altri approcci	140
15.1 K-Nearest Neighbors	140
15.1.1 Limitazioni	141
15.2 Unsupervised learning	142
15.2.1 K-means	143
15.3 Altri approcci per il preprocessing	144
15.4 Altri task	144
15.5 Altri modelli	145

Capitolo 1

Introduzione

1.1 Intelligenza Artificiale: una doppia sfida

L'Intelligenza Artificiale si occupa della

1. comprensione
2. riproduzione

del comportamento *intelligente*.

L'IA come scienza empirica

L'approccio della psicologia cognitiva:

Obiettivo: comprensione dell'intelligenza umana.

Metodo: costruzione di modelli computazionali, verifica sperimentale.

Criterio di successo: risolvere i problemi con gli stessi processi usati dall'uomo.

L'IA come disciplina informatica

L'approccio "costruttivo":

Obiettivo: costruzione di entità dotate di razionalità.

Metodo: codifica del pensiero razionale; comportamento razionale.

Criterio di successo: l'importante è risolvere i problemi che richiedono intelligenza.

1.1.1 Definizioni di IA

	Umanamente	Razionalmente
Pensare	<p><i>“L’automazione delle attività che associamo al pensiero umano, come il processo decisionale, la risoluzione di problemi, l’apprendimento...”</i></p> <p>[Bellman 1978]</p>	<p><i>“Lo studio delle facoltà mentali attraverso l’uso di modelli computazionali”</i></p> <p>[Charniak, McDermott, 1985]</p>
Agire	<p><i>“L’arte di creare macchine che svolgono funzioni che richiedono intelligenza quando svolte da esseri umani”</i></p> <p>[Kurzweil 1990]</p>	<p><i>“Il ramo della scienza dei calcolatori che si occupa dell’automazione del comportamento intelligente”</i></p> <p>[Luger-Stubblefield 1993]</p> <p><i>“L’impresa di costruire artifatti intelligenti”</i></p> <p>[Ginsberg 1993]</p>

Da “*Strategic directions in Artificial Intelligence*”

Il settore dell’IA consiste nell’indagine tecnologica e intellettuale, a lungo termine, che mira al raggiungimento dei seguenti obiettivi scientifici e pratici:

- costruzione di macchine intelligenti, sia che operino come l’uomo che diversamente;
- formalizzazione della conoscenza e meccanizzazione del ragionamento, in tutti i settori di azione dell’uomo;
- comprensione mediante modelli computazionali della psicologia e comportamento di uomini, animali e agenti artificiali;
- rendere il lavoro con il calcolatore altrettanto facile e utile che del lavoro con persone, capaci, cooperative e possibilmente esperte.

Prendiamo per buona questa definizione:

L’arte di creare macchine che svolgono funzioni che richiedono intelligenza quando svolte da esseri umani
 - Kurzweil, 1990

Ma cosa significa “intelligente”? Settanta definizioni di intelligenza, di cui 35 fuori dal settore AI. È una qualità intrinseca o un comportamento? Difficile

arrivare ad una definizione condivisa e completa perché esistono diversi tipi di “intelligenza”.

Che tipo di capacità?

- Capacità di simulare il comportamento umano?
- Capacità di ragionamento logico/matematico?
- Intelligenza come competenza “da esperto”?
- Intelligenza come “buon senso” (common sense)?
- Capacità di interagire efficacemente con un ambiente?
- Capacità sociali, di comunicazione e coordinamento?
- Capacità di comprendere e provare emozioni? (intelligenza emozionale)
- Capacità di “immagazzinare” esperienza? Apprendere?
- Creatività?

Capacità di imitazione dell'uomo? **Il test di Turing** (1950): un tentativo di definizione operativa di intelligenza. Le previsioni di Turing:

“Credo che tra circa 50 anni sarà possibile programmare computer con una memoria di un miliardo di byte in maniera tale che essi giochino il gioco dell’imitazione tanto bene che una persona comune non avrà più del 70% di probabilità di identificarli dopo 5 minuti di interrogazione”

Obiettivo raggiunto? Nel 2014 Eugene Goostman simula un ragazzo ucraino di 13 anni in maniera credibile ed è riuscito ad ingannare più del 30% dei giudici. Molto rumore nella stampa, molte reazioni scettiche.

Flying like a pigeon?

Sarebbe come se gli ingegneri aeronautici definissero il loro obiettivo come *“creare delle macchine in grado di volare come piccioni, in maniera così perfetta da ingannare gli altri piccioni”*.

Definizione di “intelligenza”

“Qualità mentale che consiste nell’abilità di apprendere dall’esperienza, di adattarsi a nuove situazioni, comprendere e gestire concetti astratti. E utilizzare conoscenza per agire sul proprio ambiente.”

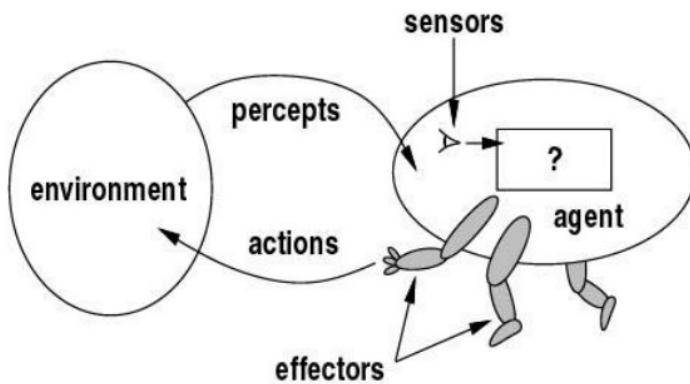
[Enciclopedia britannica]

1.2 Agenti Intelligenti: la visione “moderna”

Gli agenti sono **situati**

- ricevono *percezioni* da un ambiente;
- agiscono sull’ambiente mediante *azioni*.

Gli agenti hanno **abilità sociale**: sono capaci di comunicare, collaborare, difendersi da altri agenti. Gli agenti hanno **opinioni, obiettivi, intenzioni...**.
Gli agenti sono **embodied**: hanno un **corpo** e forse provano “emozioni”.



Agente intelligente

La sfida: RoboCup

La *Robot World Cup Initiative* (RoboCup) è un problema di riferimento per la ricerca in I.A. Si tratta di realizzare agenti in grado di giocare a calcio (entro il 2050!). Un problema difficile, da usare come banco di prova per nuove idee e tecnologie.

Tecnologie da sviluppare e integrare

- agenti autonomi
- collaborazione tra agenti
- acquisizione di strategie
- ragionamento e pianificazione in tempo reale
- robotica
- tecnologie hw e sw per infrastruttura

Capacità di emozioni?

“The question is not whether intelligent machines can have emotions, but whether machines can be intelligent without any emotions.”
[Minsky, The Society of Mind]

Capacità di comprendere e mostrare emozioni → ruolo delle emozioni nel meccanismo di decisione.

1.3 The AI revolution?

- Algoritmi di apprendimento automatico che estraggono modelli statistici predittivi da immense quantità di esempi.
- Tecniche di estrazione di “*significati*” da grande quantità di testi.
- Sistemi in grado di rispondere a domande in linguaggio naturale in un “dominio aperto”.

La disponibilità di grosse quantità di dati

L'enfasi si sposta dagli algoritmi ai dati e agli algoritmi di apprendimento (ML e DM). Esempi dalle tecnologie del linguaggio:

- La traduzione automatica di Google
- L'interazione in linguaggio naturale di SIRI

Più dati, maggiore l'accuratezza, ... apparentemente senza limite (“Unreasonable Effectiveness of Big Data”)

La domanda diventa: l'intelligenza può essere estratta o inferita dai dati?

The *deep learning* tsunami

L'idea di usare reti neurali con molti livelli (deep NN) era stata tentata per diversi anni, ma solo verso il 2011 si riescono ad avere successi evidenti, come riconoscimento di immagini e parlato.

C. Manning: “le reti neurali profonde sono in giro da un po' di anni ma nel 2015 hanno colpito come uno tsunami il settore del NLP”.

I maggiori esperti (Le Cun, Hinton, Benjo) sono concordi nel ritenere che ci saranno sviluppi importanti a breve nella comprensione dei testi, video, traduzione automatica, QA, ...

Apprendimento di Deep Networks diviene praticabile dal punto di vista computazionale:

- Possibilità di usare GPU multiple (Graphic Processing Unit); ottimizzate per il prodotto di matrici.
- Se avessimo usato gli stessi algoritmi 10 anni fa, non avrebbero ancora finito.

1.3.1 Priorità di ricerca

L'intelligenza artificiale... “deve fare solo quello che noi vogliamo che faccia”. Servono ricerche non solo per rendere l'IA più **capace** ma anche fare in modo che sia **robusta** e **benefica** per la società (*Trustworthy AI*)

- Impatto economico sul mercato del lavoro e sulla società
- Responsabilità dei veicoli autonomi, etica delle macchine, armi autonome, privacy
- Verifica (il sistema è “corretto”?), validità (il sistema è “giusto”?)
- Sicurezza (protezione da terzi), controllo (dei sistemi autonomi)

1.4 In sintesi

È difficile dare una definizione univoca di “intelligenza” e quindi di “intelligenza artificiale”. A seconda dei periodi storici gli approcci sono diversi, l'enfasi è diversa, gli obiettivi stessi sono diversi. Parafrasando la definizione di Kurzweil

*A.I è l'arte di creare macchine che svolgono funzioni che tuttora richiedono intelligenza umana per essere svolte
... un “bersaglio mobile”.*

A.I. (ML) come alternativa all'approccio algoritmico tradizionale in presenza di incertezza.

- *Sorgenti di incertezza: sensori imperfetti, dati incompleti, limiti alle capacità di calcolo*
- ...
- *Thrun: AI is the technique of uncertainty management in computer software. AI is the discipline that you apply when you don't know what to do.*

Ma ricordate: l'IA non coincide con il ML; la General AI è ancora lontana e dovrà integrare diverse capacità e in particolare apprendimento e ragionamento.

Capitolo 2

Agenti Intelligenti

IA - “ri”considerazioni

Non si tratta di avere delle collezioni di tecniche per risolvere problemi specifici. L'intelligenza artificiale è il vertice e il fronte del progresso dei metodi/sistemi informatici (e.g. algoritmica, logica, ottimizzazione) per fornire metodologie sistematiche per dotare le macchine di comportamenti intelligenti/*razionali* (cfr. AIMA) in problemi considerati generalmente difficili.

Iniziamo con l'inquadramento degli **agenti**.

Agenti intelligenti

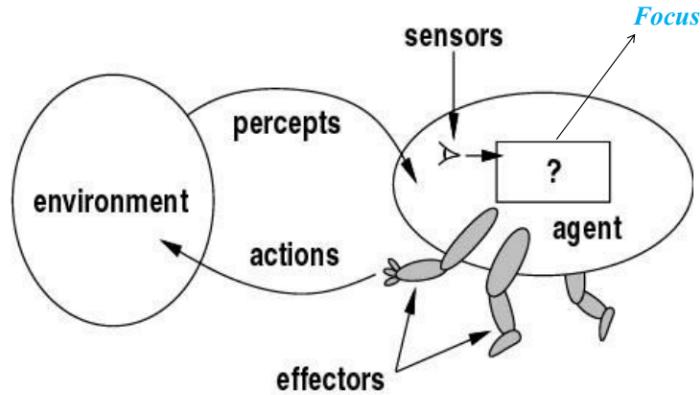
L'approccio “moderno” all'IA è la costruzione di *agenti intelligenti*. La visione ad agenti ci offre un quadro di riferimento e una prospettiva diversa all'analisi dei sistemi software. È “comoda” per trattare sistemi razionali (uniformità) e vedremo schemi di agenti contenitori di funzionalità studiate nel resto del corso. Il primo obiettivo sarà quello di capire gli agenti per risoluzione di problemi vista come **ricerca** in uno **spazio di stati** (*problem solving*).

2.1 Caratteristiche degli agenti

Gli agenti sono **situati**

- ricevono *percezioni* da un ambiente;
- agiscono sull'ambiente mediante *azioni*.

Gli agenti hanno **abilità sociale**: sono capaci di comunicare, collaborare, difendersi da altri agenti. Gli agenti hanno **opinioni**, **obiettivi**, **intenzioni**...
Gli agenti sono **embodied**: hanno un **corpo** e forse provano “**emozioni**”.



Ciclo **percezione- azione**

Agente intelligente

2.1.1 Percezioni e azioni

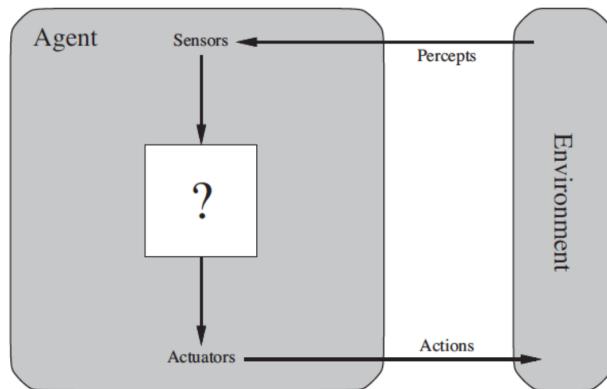
- Percezione: input da sensori
- **Sequenza percettiva:** *storia completa delle percezioni*

La scelta dell'azione è funzione unicamente della sequenza percettiva.

Funzione agente: definisce l'azione da compiere per ogni sequenza percettiva (descrive completamente l'agente).

$$\text{Sequenza percettiva} \rightarrow \text{Azione}$$

Implementata da un **programma agente**. Il compito (IA) è progettare il *programma agente*.



Agente e ambiente: architettura astratta

2.1.2 Agenti razionali

Un **agente razionale** interagisce con il suo ambiente in maniera “efficace” (fa la cosa “giusta”).

Serve un criterio di **valutazione** oggettivo dell’effetto delle azioni dell’agente (della sequenza di stati dell’ambiente).

Valutazione della prestazione

Misura di prestazione

- Esterna (come vogliamo che il mondo evolva?)
- Scelta dal progettista a seconda del problema considerando l’effetto desiderato sull’ambiente
- (possibile) Valutazione su ambienti diversi

Agente razionale: definizione

La razionalità è relativa a (dipende da):

- la misura di prestazioni,
- le conoscenze pregressa dell’ambiente,
- le percezioni presenti e passate (seq. percettiva),
- le capacità dell’agente (azioni possibili).

Agente razionale: per ogni sequenza di percezioni compie l’azione che *massimizza il valore atteso della misura delle prestazioni*, considerando le sue percezioni passate e la sua conoscenza pregressa.

Razionalità non onniscienza: non si pretendono perfezione e conoscenza del “futuro”, ma massimizzare il risultato atteso. La scelta dell’azione è *funzione unicamente* della sequenza percettiva: potrebbero essere necessarie azioni di acquisizione di informazioni o esplorative.

Razionalità non onnipotenza: le capacità dell’agente possono essere limitate.

Razionalità e apprendimento

Raramente tutta la conoscenza sull'ambiente può essere fornita “a priori” dal programmatore, quindi l'agente razionale deve essere in grado di modificare il proprio comportamento con l'esperienza (le percezioni passate): può migliorare esplorando, *apprendendo*, aumentando autonomia per operare in ambienti differenti o mutevoli.

Agenti autonomi

Agente autonomo: un agente è autonomo nella misura in cui il suo comportamento dipende dalla sua esperienza. Un agente il cui comportamento fosse determinato solo dalla sua conoscenza *built-in*, sarebbe non autonomo e poco flessibile (*comportamento stereotipato*).

2.1.3 Ambiente

Definire un problema per un agente significa caratterizzare l'ambiente in cui l'agente opera (ambiente operativo). Agente razionale → soluzione.

Descrizione PEAS dei problemi

- **P**erformance: prestazione
- **E**nvironment: ambiente
- **A**ctuators: attuatori
- **S**ensors: sensori

Prestazione	Ambiente	Attuatori	Sensori
Arrivare alla destinazione, sicuro, veloce, ligio alla legge, viaggio confortevole, minimo consumo di benzina, profitti massimi	Strada, altri veicoli, pedoni, clienti	Sterzo, acceleratore, freni, frecce, clacson, schermo di interfaccia o sintesi vocale	Telecamere, sensori a infrarossi e sonar, tachimetro, GPS, contachilometri, accelerometro, sensori sullo stato del motore, tastiera o microfono

Esempio: agente guidatore di taxi.

Proprietà dell'ambiente-problema

- Completamente/parzialmente osservabile
- Agente singolo/multi-agente
- Deterministico/stocastico/non deterministico
- Episodico/sequenziale (esiste storia?)
- Statico/dinamico
- Discreto/continuo

Osservabilità

Ambiente **completamente osservabile**: l'apparato percettivo è in grado di dare una conoscenza completa dell'ambiente o almeno tutto quello che serve a decidere l'azione. Non c'è bisogno di mantenere uno stato del mondo (esterno).

Ambiente **parzialmente osservabile**: sono presenti limiti o inaccuratezze dell'apparato sensoriale.

Ambiente singolo/multiagente

Distinzione agente/non agente: il mondo può anche cambiare per eventi, non necessariamente per azioni di agenti.

Ambiente **multi-agente competitivo** (scacchi): comportamento randomizzato (è razionale).

Ambiente **multi-agente cooperativo** (o benigno):

- Stesso obiettivo
- Comunicazione

Predicibilità

Deterministico: se lo stato successivo è completamente determinato dallo stato corrente e dall'azione. Esempio: scacchi.

Stocastico: esistono elementi di incertezza con associata probabilità. Esempi: guida, tiro in porta.

Non deterministico: si tiene traccia di più stati possibili risultato dell'azione (ma non in base ad una probabilità).

Episodico/sequenziale

Episodico: l'esperienza dell'agente è divisa in episodi atomici indipendenti.
In ambienti episodici non c'è bisogno di pianificare.

Sequenziale: ogni decisione influenza le successive.

Statico/dinamico

Statico: il mondo non cambia mentre l'agente decide l'azione.

Dinamico: cambia nel tempo, va osservata la contingenza. Tardare equivale a non agire.

Semi-dinamico: l'ambiente non cambia ma la valutazione dell'agente sì.
Esempio: scacchi con timer.

Discreto/continuo

Possono assumere valori discreti o continui

- lo stato: solo un numero finito di stati;
- il tempo;
- le percezioni;
- le azioni.

La guida del taxi è un problema con stato e tempo continuo...
Combinatoriale (nel discreto) versus infinito (nel continuo).

Noto/ignoto

Distinzione riferita allo stato di conoscenza dell'agente sulle leggi fisiche dell'ambiente. L'agente conosce l'ambiente oppure deve compiere azioni esplorative?

Noto è diverso da osservabile (e.g. carte coperte, regole note).

Ambienti reali: parzialmente osservabili, stocastici, sequenziali, dinamici, continui, multi-agente, ignoti.

Simulatore di ambienti

Uno strumento software che si occupa di:

- generare stimoli per gli agenti,
- raccogliere le azioni in risposta,

- aggiornare lo stato dell’ambiente,
- attivare altri processi che influenzano l’ambiente,
- valutare le prestazioni degli agenti.

Esperimenti su classi di ambienti (variando le condizioni) essenziale per valutare capacità di generalizzare. Valutazione prestazione come medie su più istanze.

2.1.4 Struttura di un agente

Agente = Architettura + Programma

Agente: Percezioni → Azioni

Il **programma dell’agente** implementa la funzione *Agente*.

Programma agente

```
function Skeleton-Agent (percept) returns action
    static: memory, the agent’s memory of the world
    memory  $\leftarrow$  UpdateMemory(memory, percept)
    action  $\leftarrow$  Choose-Best-Action(memory)
    memory  $\leftarrow$  UpdateMemory(memory, action)
    return action
```

Agente basato su tabella

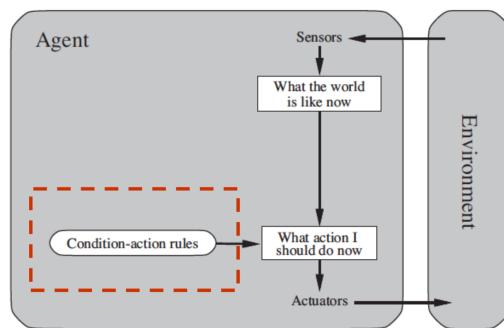
La scelta dell’azione è un accesso a una tabella che associa un’azione ad ogni possibile sequenza di percezioni. Problemi:

1. Dimensione: per giocare a scacchi servirebbe una tabella con un numero di righe $>> 10^{80}$ numero di atomi nell’universo! \rightarrow ingestibile
2. Difficile da costruire
3. Nessuna autonomia
4. Di difficile aggiornamento, apprendimento complesso.

In IA vogliamo realizzare agenti razionali con programma “*compatto*”.

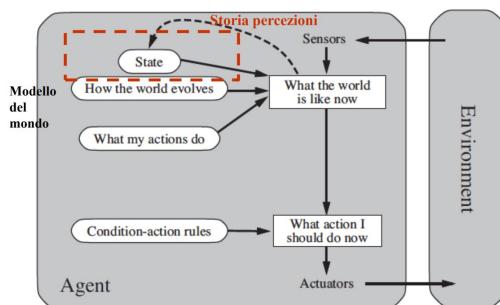
Agenti reattivi semplici

```
function Agente-Reattivo-Semplice (percezione) returns azione
  persistent: regole, un insieme di regole
  condizione-azione (if-then)
  stato  $\leftarrow$  Interpreta-Input(percezione)
  regola  $\leftarrow$  Regola-Corrispondente(stato, regole)
  azione  $\leftarrow$  regola.Azione
  return azione
```



Agenti basati su modello

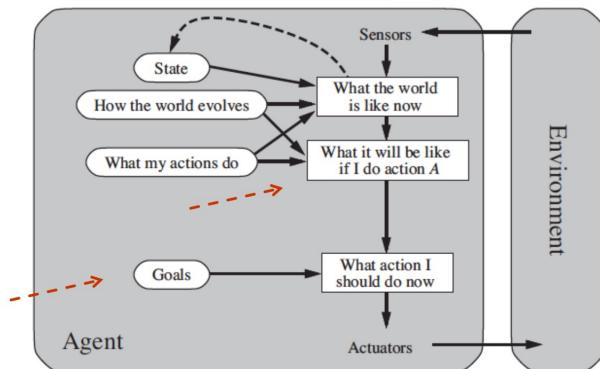
```
function Agente-Basato-su-Modello (percezione) returns azione
  persistent: stato, una descrizione dello stato corrente
  modello, conoscenza del mondo
  regole, un insieme di regole condizione-azione
  azione, l'azione più recente
  stato  $\leftarrow$  Aggiorna-Stato(stato, azione, percez., modello)
  regola  $\leftarrow$  Regola-Corrispondente(stato, regole)
  azione  $\leftarrow$  regola.Azione
  return azione
```



Agenti con obiettivo

Sono guidati da un obiettivo nella scelta dell'azione (è fornito un goal esplicito, per esempio città da raggiungere).

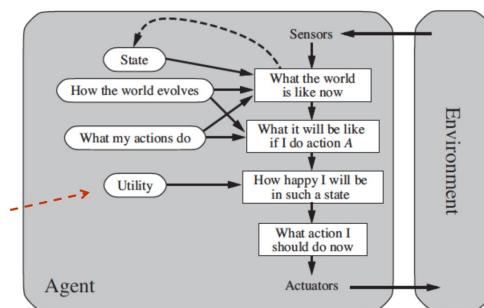
- A volte l'azione migliore dipende da qual è l'obiettivo da raggiungere (es. da che parte devo girare?).
- Devono pianificare una sequenza di azioni per raggiungere l'obiettivo.
- Meno efficienti ma più flessibili di un agente reattivo (obiettivo può cambiare, non è già codificato nelle regole).



Agenti con valutazione di utilità

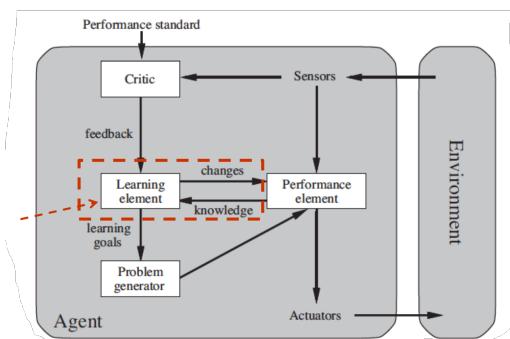
Obiettivi alternativi (o più modi per raggiungerlo), l'agente deve decidere verso quali di questi muoversi. Necessaria una funzione di **utilità** (che associa ad uno stato obiettivo un numero reale).

Obiettivi più facilmente raggiungibili di altri, la funzione di utilità tiene conto anche della probabilità di successo (e/o di ciascun risultato): **utilità attesa** (o in media)



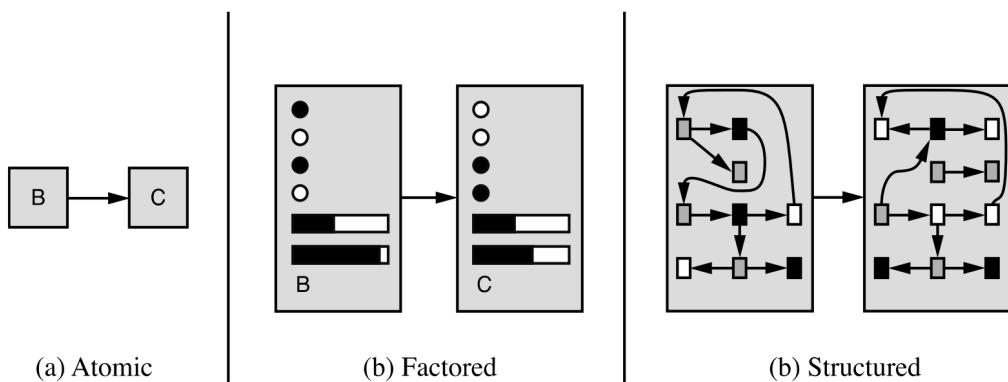
Agenti che apprendono

1. Componente di apprendimento: produce cambiamenti al programma agente e ne migliora prestazioni, adattando i suoi componenti, apprendendo dall'ambiente.
2. Elemento esecutivo: il programma agente
3. Elemento critico: osserva e dà feedback sul comportamento
4. Generatore di problemi: suggerisce nuove situazioni da esplorare



Tipi di rappresentazione

- Rappresentazione atomica (stati)
- Rappresentazione fattorizzata (piu variabili e attributi)
- Rappresentazione strutturata (aggiunge relazioni)



Capitolo 3

Agenti risolutori di problemi

Adottano il paradigma della **risoluzione di problemi come ricerca in uno spazio di stati** (*problem solving*). Sono agenti **con modello** (storia percezioni) che adottano una rappresentazione atomica dello stato. Sono particolari **agenti con obiettivo**, che pianificano l'intera sequenza di mosse prima di agire.

Il processo di risoluzione

Passi da seguire:

1. Determinazione obiettivo (un insieme di stati in cui obiettivo è soddisfatto)
2. Formulazione del problema
 - rappresentazione degli stati
 - rappresentazione delle azioni
3. Determinazione della soluzione mediante **ricerca** (un piano)
4. Esecuzione del piano

Che tipo di assunzioni?

L'ambiente è statico, osservabile (so dove sono), discreto (un insieme finito di azioni possibili) e deterministico (1 azione → 1 risultato). Si assume che l'agente possa eseguire il piano “ad occhi chiusi”, niente può andare storto.

Formulazione del problema

Un problema può essere definito formalmente mediante cinque componenti:

1. Stato iniziale
2. Azioni possibili in s : $\text{Azioni}(s)$
3. Modello di transizione:
 - Risultato: $\text{stato} \times \text{azione} \rightarrow \text{stato}$
 - Risultato(s, a) = s' , uno stato **successore**
4. Test obiettivo:
 - Un insieme di stati obiettivo
 - Goal-Test: $\text{stato} \rightarrow \{\text{true}, \text{false}\}$
5. Costo del cammino
 - somma dei costi delle azioni (costo dei passi)
 - costo di passo: $c(s, a, s')$
 - Il costo di un'azione/passo non è mai negativo

1, 2 e 3 definiscono implicitamente lo spazio degli stati (definirlo esplicitamente può essere molto oneroso, come in quasi tutti i problemi di IA, questo sarà rilevante nel seguito).

3.1 Algoritmi di ricerca

*“Il processo che cerca una sequenza di azioni che raggiunge l’obiettivo è detto **ricerca**”*

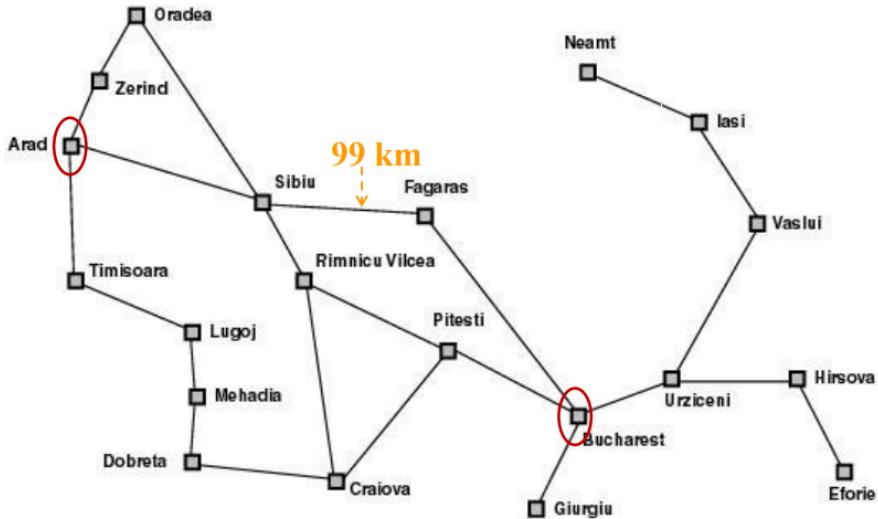
Gli algoritmi di ricerca prendono in input un problema e restituiscono un **cammino soluzione**, i.e. un cammino che porta dallo stato iniziale a uno stato goal.

Misura delle prestazioni: Trova una soluzione? Quanto costa trovarla? Quanto efficiente è la soluzione?

Costo totale = costo della ricerca + costo del cammino soluzione

3.1.1 Itinerario: il problema

Trovare il percorso più breve da una città di partenza a una città di arrivo.



Formulazione

Stati: le città.

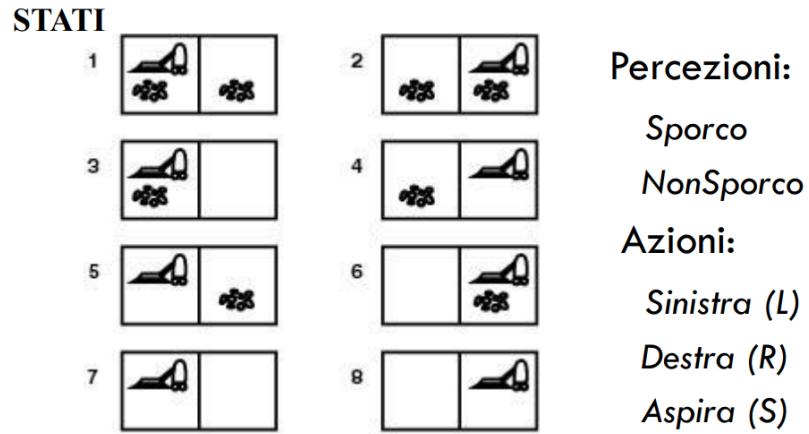
1. *Stato iniziale*: la città da cui si parte. $In(Arad)$
2. *Azioni*: spostarsi su una città vicina collegata.
 $Azioni(In(Arad)) = \{ Go(Sibiu), Go(Zerind) \dots \}$
3. *Modello di transizione*: Risultato($In(Arad), Go(Sibiu)\} = In(Sibiu)$)
4. *Test Obiettivo*: $\{In(Bucarest)\}$
5. *Costo del cammino*: somma delle lunghezze delle strade

Lo **spazio degli stati** coincide con la rete (grafo) di collegamenti tra città i.e. grafo di stati collegati da azioni, rappresentabile in modo esplicito in questo caso semplice, tramite la mappa.

Astrazione dai dettagli: essenziale per “modellare”.

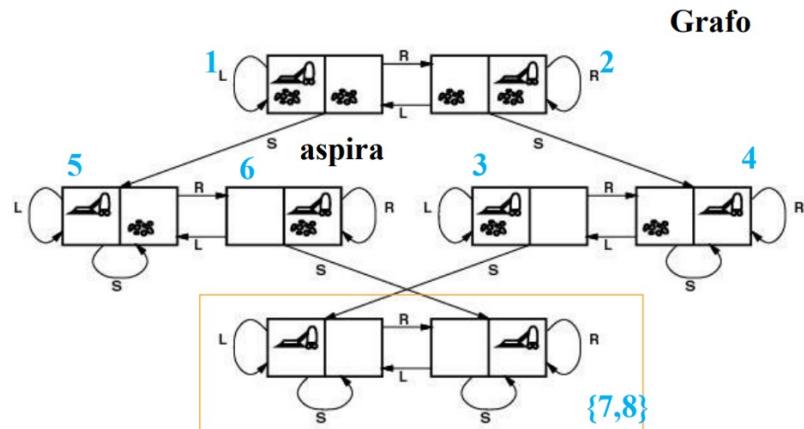
3.1.2 Aspirapolvere: il problema (toy problem)

Versione semplice: solo due locazioni, sporche o pulite, l'agente può essere in una delle due.

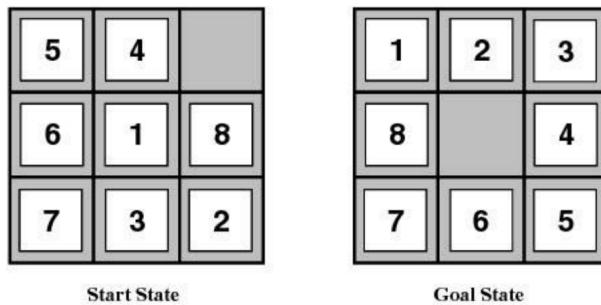


Formulazione

Obiettivo: rimuovere lo sporco {7, 8}. Ogni azione ha costo 1.



3.1.3 Il puzzle dell'otto



Formulazione

- *Stati*: possibili configurazioni della scacchiera.
- *Azioni*: mosse della casella bianca

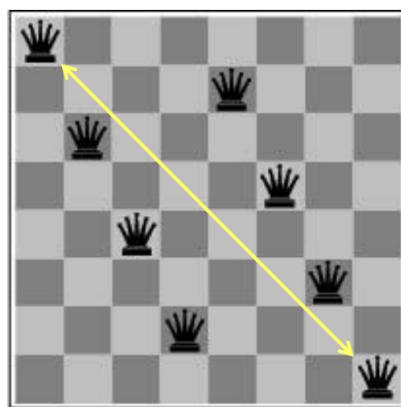
in sù: ↑ in giù: ↓
a destra: → a sinistra: ←

- *Costo cammino*: ogni passo costa 1

Lo spazio degli stati è un grafo con possibili cicli. NP-completo. Per 8 tasselli: $9!/2 = 181K$ stati! Ma risolvibile in poco tempo (ms). Se cresce no!

3.1.4 Le otto regine: il problema

Collocare otto regine sulla scacchiera in modo tale che nessuna regina sia attaccata da altre.



Formulazione incrementale 1

Si aggiungono le regine una alla volta.

- *Stati*: scacchiere con 0-8 regine
- *Goal-Test*: 8 regine sulla scacchiera, nessuna attaccata
- *Costo cammino*: zero (resta 8 per le 8 mosse effettive e non è rilevante, interessa solo lo stato finale)
- *Azioni*: aggiungi una regina
- *Spazio stati*: $64 \times 63 \times \dots \times 57 \sim 1.8 \times 10^{14}$ sequenze possibili da considerare!

La ricerca può essere molto onerosa!

Formulazione incrementale 2

- *Stati*: scacchiere con 0-8 regine, **nessuna minacciata**
- *Goal-Test*: 8 regine sulla scacchiera, nessuna minacciata
- *Costo cammino*: zero
- *Azioni*: aggiungi una regina **nella colonna vuota più a destra ancora libera in modo che non sia minacciata**.

2057 sequenze da considerare.

Formulazione a stato completo

- *Stati*: 8 regine già sulla scacchiera, nessuna minacciata
- *Costo cammino*: zero
- *Stati*: scacchiera con **8 regine, una per colonna**
- *Azioni*: **sposta una regina nella colonna, se minacciata**

3.1.5 Dimostrazione di teoremi

Il problema: dato un insieme di premesse

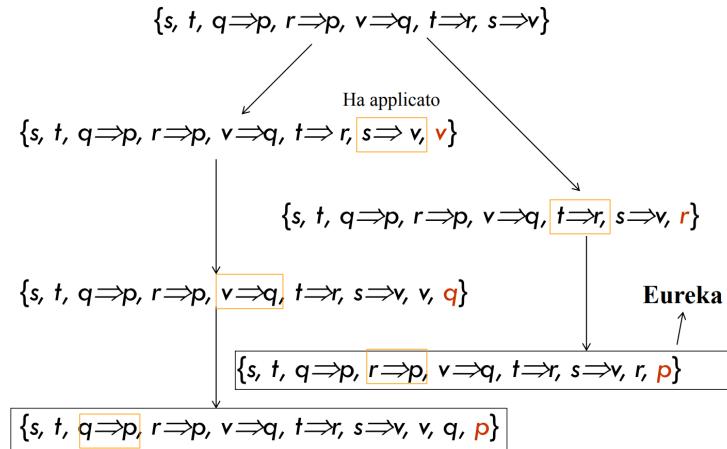
$$\{s, t, q \Rightarrow p, r \Rightarrow p, v \Rightarrow q, t \Rightarrow r, s \Rightarrow v\}$$

dimostrare una proposizione p . Nel calcolo proposizionale un'unica regola di inferenza, il Modus Ponens (MP):

Se p e $p \Rightarrow q$ allora q .

Dimostrazione teoremi: formulazione

- *Stati*: insiemi di proposizioni
- *Stato iniziale*: un insieme di proposizioni (le premesse).
- *Stato obiettivo*: un insieme di proposizioni contenente il teorema da dimostrare.
- *Operatori*: l'applicazione del MP, che aggiunge teoremi.



3.1.6 Problemi reali

Pianificazione di viaggi aerei

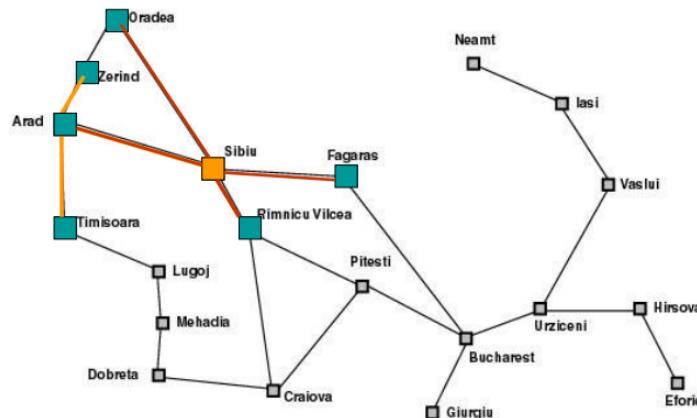
- Problema del commesso viaggiatore
- Configurazione VLSI
- Navigazione di robot (spazio continuo!)

- Montaggio automatico
- Progettazione di proteine
- ...

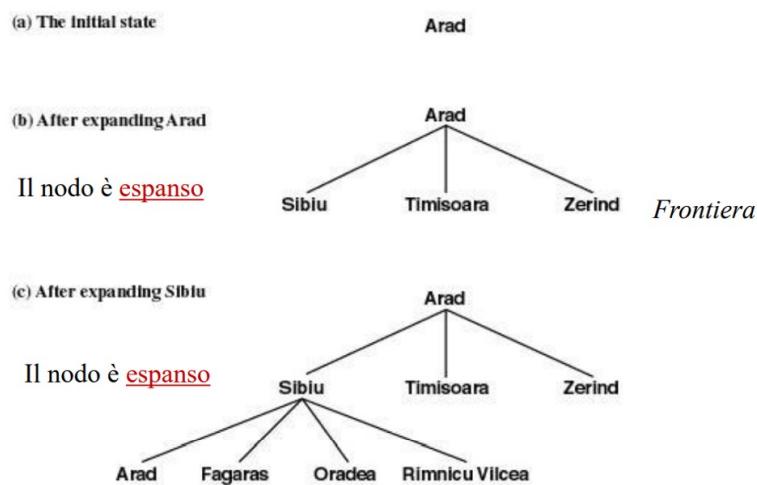
3.2 Ricerca della soluzione

Generazione di un **albero di ricerca** sovrapposto allo **spazio degli stati** (generato da *possibili* sequenze di azioni).

La **ricerca** è approfondire un'opzione, tenendo da parte le altre per poi ripredenderle se non si trova una soluzione.



Generazione di un *albero di ricerca* sovrapposto allo spazio degli stati.



Nota: “nodo” diverso da “stato”, per esempio possono esistere esistono nodi di un albero di ricerca con stesso stato (città).

3.2.1 Ricerca ad albero

Ossia senza controllare se i nodi (stati) siano già stati esplorati. Vedremo la ricerca “a/su grafo” con questi controlli.

```
function Ricerca-Albero (problema) returns soluzione oppure fallimento
    Inizializza la frontiera con stato iniziale del problema
    loop do
        if la frontiera è vuota then return fallimento
        Scegli un nodo foglia da espandere e rimuovilo dalla frontiera
        if il nodo contiene uno stato obiettivo
            then return soluzione corrispondente
        Espandi il nodo e aggiungi i successori alla frontiera
```

3.2.2 I nodi dell’albero di ricerca

Un nodo n è una struttura dati con quattro componenti: uno stato, $n.\text{stato}$, il nodo padre, $n.\text{padre}$, l’azione effettuata per generarlo, $n.\text{azione}$, e il costo del cammino dal nodo iniziale al nodo, $N.\text{costo-cammino}$ indicata come $g(n)$ ($= \text{padre}.\text{costo-cammino} + \text{costo-passo ultimo}$).

3.2.3 Struttura dati per la frontiera

Frontiera: lista dei nodi in attesa di essere espansi (le foglie dell’albero di ricerca). La frontiera è implementata come una coda con operazioni:

- Vuota?(coda)
- POP(coda) estraie il primo elemento
- Inserisci(elemento, coda)
- Diversi tipi di coda hanno diverse funzioni di inserimento e implementano *strategie* diverse

3.2.4 Diversi tipi di strategie (di ricerca)

- FIFO - First In First Out → BF: viene estratto l'elemento più vecchio (in attesa da più tempo); i nuovi nodi sono aggiunti alla fine.
- LIFO - Last In First Out → DF: viene estratto il più recentemente inserito; i nuovi nodi sono inseriti all'inizio.
- Coda con priorità → UC, et altri successivi: viene estratto quello con priorità più alta in base a una funzione di ordinamento; dopo l'inserimento dei nuovi nodi si riordina.

Strategie non informate

- Ricerca in ampiezza (BF)
- Ricerca in profondità (DF)
- Ricerca in profondità limitata (DL)
- Ricerca con approfondimento iterativo (ID)
- Ricerca di costo uniforme (UC)

Tali strategie verranno confrontate con strategie di ricerca euristica (o informata): fanno uso di informazioni riguardo alla distanza stimata dalla soluzione.

3.2.5 Valutazione di una strategia

- *Completezza*: se la soluzione esiste viene trovata
- *Ottimalità* (ammissibilità): trova la soluzione migliore, con costo minore (per il “costo del cammino soluzione”)
- *Complessità in tempo*: tempo richiesto per trovare la soluzione
- *Complessità in spazio*: memoria richiesta

Quest'ultimi due fattori riguardano il “costo della ricerca”.

3.2.6 Ricerca in ampiezza (BF - Breadth-first)

Consiste nell'esplorare il grafo dello spazio degli stati a livelli progressivi di stessa profondità. Implementata con una coda che inserisce alla fine (**FIFO**).

Ricerca in ampiezza ad albero

Senza gestire problemi stati già esplorati.

```
function Ricerca-Aampiezza-A (problema) returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e
    costo-di-cammino=0
    if problema.Test-Obiettivo(nodo.Stato) then return Soluzione(nodo)
    frontiera = una coda FIFO con nodo come unico elemento
    loop do
        if Vuota?(frontiera) then return fallimento
        nodo = POP(frontiera)
        for each azione in problema.Azioni(nodo.Stato) do
            figlio = Nodo-Figlio(problema, nodo, azione)
            if problema.TestObiettivo(figlio.Stato) then return Soluzione(figlio)
            frontiera = Inserisci(figlio, frontiera)
    end
```

Ricerca in ampiezza su grafo

Senza gestire problemi stati già esplorati.

```
function Ricerca-Aampiezza-A (problema) returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e
    costo-di-cammino=0
    if problema.Test-Obiettivo(nodo.Stato) then return Soluzione(nodo)
    frontiera = una coda FIFO con nodo come unico elemento
    esplorati = insieme vuoto
    loop do
        if Vuota?(frontiera) then return fallimento
        nodo = POP(frontiera); aggiungi nodo.Stato a esplorati
        for each azione in problema.Azioni(nodo.Stato) do
            figlio = Nodo-Figlio(problema, nodo, azione)
            if figlio.Stato non è in esplorati e non è in frontiera then
                if problema.TestObiettivo(figlio.Stato) return Soluzione(figlio)
                frontiera = Inserisci(figlio, frontiera)
    end
```

Nota che in entrambe le versioni i *nodo.Stato* sono goal-tested al momento in cui sono generati. → **Più efficiente**, si ferma appena trova goal prima di espandere.

Analisi complessità spazio-temporale (BF)

Assumiamo:

b = fattore di ramificazione (branching) (numero max di successori)

d = profondità del nodo obiettivo più superficiale (depth) (più vicino all'iniziale)

m = lunghezza massima dei cammini nello spazio degli stati (max)

È una strategia *completa* ed è anche *ottimale se gli operatori hanno tutti lo stesso costo k*, cioè $g(n) = k \cdot \text{depth}(n)$, dove $g(n)$ è il costo del cammino per arrivare a n .

Complessità nel tempo (nodi generati):

$$T(b, d) = b + b^2 + \dots + b^d \rightarrow O(b^d)$$

Nota bene: il numero nodi cresce esponenzialmente, non assumiamo di conoscere già il grafo né una notazione di linearità nel numero nodi. Questo è tipico dei problemi in AI (pensate a quelli generati per le configurazioni dei giochi, con rappresentazione implicita dello spazio stati).

Complessità spazio (nodi in memoria): $O(b^d)$ (solo per la frontiera)

3.2.7 Ricerca in profondità (DF)

Implementata da una coda che mette i successori in testa alla lista (LIFO, pila o stack).

Cancelli rami completamente esplorati ma tiene tutti i fratelli del path corrente: memoria solo $b \times m$.

Ricerca in profondità: analisi (versione su albero)

Se m lunghezza massima dei cammini nello spazio degli stati e b è il fattore di diramazione

Tempo: $O(b^m)$, che può essere $> O(b^d)$

Occupazione memoria: bm (frontiera sul cammino)

Strategia *non completa* (loop) e *non ottimale*. Drastico risparmio in memoria:

BF	$d = 16$	10 esabyte
DF	$d = 16$	156 Kbyte

Ricerca in profondità: analisi (versione su grafo)

In caso di DF con visita a grafo si perdono i vantaggi di memoria: la memoria torna da $b \times m$ a tutti i possibili stati (potenzialmente, caso pessimo, esponenziale come BF) (per mantenere la lista dei visitati/esplorati), ma così DF diviene completa in spazi degli stati finiti (tutti i nodi verranno espansi nel caso pessimo). Resta comunque non completa in spazi infiniti.

Sarebbe possibile controllare anche solo i nuovi stati rispetto al cammino radice-nodo corrente senza aggravio di memoria evitando però così solo i cicli finiti in spazi finiti ma non i cammini ridondanti.

Ricerca in profondità (DF) ricorsiva

Ancora più efficiente in occupazione di memoria perché mantiene solo il cammino corrente (solo m nodi nel caso pessimo). Realizzata da un algoritmo ricorsivo “con backtracking” che non necessita di tenere in memoria b nodi per ogni livello, ma salva lo stato su uno stack a cui torna in caso di fallimento per fare altri tentativi.

```
function Ricerca-DF-A (problema) returns soluzione oppure fallimento
    return Ricerca-DF-ricorsiva(CreaNodo(problema.Stato-iniziale),
                                  problema)  
  
function Ricerca-DF-ricorsiva(nodo, problema) returns soluzione oppure
fallimento
    if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
    else
        for each azione in problema.Azioni(nodo.Stato) do
            figlio = Nodo-Figlio(problema, nodo, azione)
            risultato = Ricerca-DF-ricorsiva(figlio, problema)
            if risultato ≠ fallimento then return risultato
    return fallimento
```

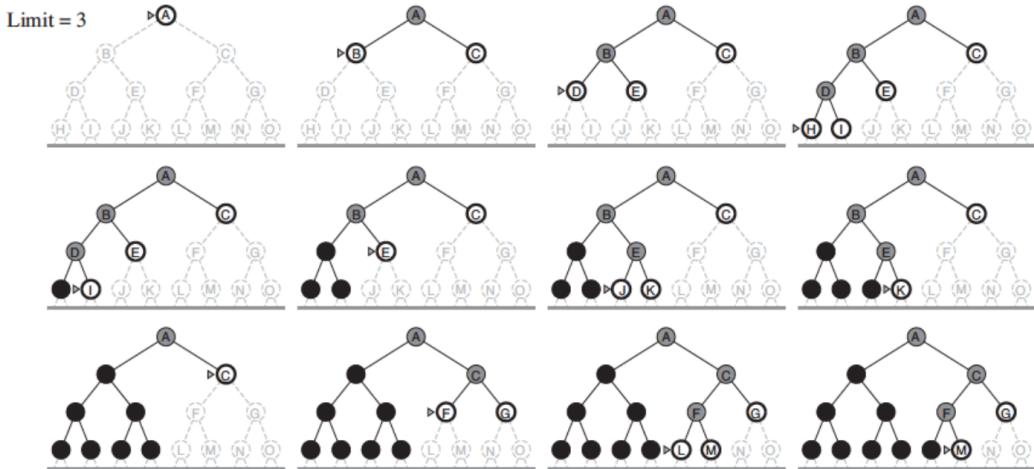
Ricerca in profondità limitata (DL)

Si va in profondità fino ad un certo livello predefinito l . Completa per problemi in cui si conosce un limite superiore per la profondità della soluzione.

È completo se $d < l$, ma non è ottimale.

Complessità tempo: $O(b^l)$
Complessità spazio: $O(bl)$

3.2.8 Approfondimento iterativo (ID)



Miglior compromesso tra BF e DF

$$\text{BF} : b + b^2 + \dots + b^{d-1} + b^d$$

con $b = 10$ e $d = 5$

$$10 + 100 + 1000 + 10.000 + 100.000 = 111.110$$

ID: I nodi dell'ultimo livello generati una volta, quelli del penultimo 2, quelli del terzultimo 3 ... quelli del primo d volte.

$$\begin{aligned} \text{ID: } & (d)b + (d-1)b^2 + \dots + 3b^{d-2} + 2b^{d-1} + 1b^d = \\ & 50 + 400 + 3000 + 20.000 + 100.000 = 123450 \end{aligned}$$

Complessità tempo: $O(b^d)$
 Spazio: $O(bd)$ versus $O(b^d)$ della BF.

Ergo: vantaggi della BF, con tempi analoghi ma costo memoria analogo a quello di DF.

3.2.9 Direzione della ricerca

Un problema ortogonale alla strategia è la *direzione della ricerca*. Nella ricerca **in avanti** o *guidata dai dati* si esplora lo spazio di ricerca dallo stato iniziale allo stato obiettivo; invece nella ricerca **all'indietro** o *guidata*

dall'obiettivo si esplora lo spazio di ricerca a partire da uno stato goal e riconducendosi a sotto-goal fino a trovare uno stato iniziale.

Quale direzione scegliere? Conviene procedere nella direzione in cui il fattore di diramazione è minore. Si preferisce ricerca all'indietro quando:

- l'obiettivo è chiaramente definito (e.g. theorem proving) o si possono formulare una serie limitata di ipotesi;
- i dati del problema non sono noti e la loro acquisizione può essere guidata dall'obiettivo.

Si preferisce ricerca in avanti quando:

- gli obiettivi possibili sono molti (design);
- abbiamo una serie di dati da cui partire.

Nella ricerca bidirezionale si procede nelle due direzioni fino ad incontrarsi.

Ricerca bidirezionale: analisi

Complessità tempo: $O(b^{d/2})$ ($/2$ = radice quadrata!) (assumendo test intersezione in tempo costante, per esempio hash table).

Complessità spazio: $O(b^{d/2})$ (almeno tutti i nodi in una direzione in memoria, per esempio usando BF).

Nota: non sempre applicabile, per esempio nei casi di predecessori non definiti, troppi stati obiettivo...

3.2.10 Ricerca “ad albero” / “a grafo”: cammini ciclici

I cammini ciclici rendono gli alberi di ricerca infiniti con spazi degli stati infiniti. Su spazi di stati a grafo si generano più volte gli stessi nodi (o meglio nodi con stesso stato) nella ricerca, **anche in assenza di cicli**.

Compromesso tra spazio e tempo

Ricordare gli stati già visitati occupa spazio (ad esempio lista *esplorati* in visita a grafo) ma ci consente di evitare di visitarli di nuovo.

Gli algoritmi che dimenticano la propria storia sono destinati a ripeterla!

Tre soluzioni

In ordine crescente di costo e di efficacia:

- Non tornare nello stato da cui si proviene: si elimina il genitore dai nodi successori (non evita i cammini ridondanti).
- Non creare cammini con cicli: si controlla che i successori non siano antenati del nodo corrente (detto per la DF).
- Non generare nodi con stati già visitati/esplorati: ogni nodo visitato deve essere tenuto in memoria per una complessità $O(s)$ dove s è il numero di stati possibili (e.g. hash table per accesso efficiente).

Repetita: il costo può essere alto; in caso di DF la memoria torna da $b \times m$ a tutti gli stati, ma diviene una ricerca completa (per spazi finiti). Ma in molti casi gli stati crescono in modo esponenziale (gioco otto, scacchi, ...).

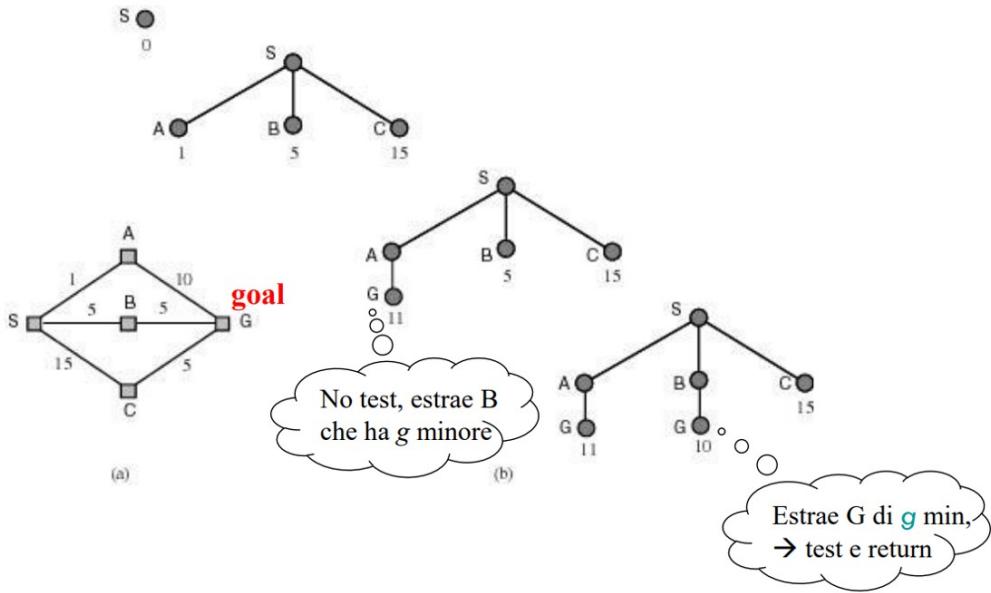
Ricerca “su grafi”

Mantiene una lista dei nodi (stati) visitati/esplorati (anche detta **lista chiusa**). Prima di espandere un nodo si controlla se lo stato era stato già incontrato prima o è già nella frontiera; se questo succede, il nodo appena trovato non viene espanso.

Ottimale solo se abbiamo la garanzia che il costo del nuovo cammino sia maggiore o uguale (cioè il nuovo cammino non conviene).

3.2.11 Ricerca di costo uniforme (UC)

Generalizzazione della ricerca in ampiezza (costi diversi tra passi): *si sceglie il nodo di costo minore sulla frontiera* (si intende il costo $g(n)$ del cammino), si espande sui contorni di **uguale costo** (e.g. in km) invece che sui contorni di uguale profondità (BF).



Implementata da una coda ordinata per costo cammino crescente (in cima i nodi di costo minore).

Ricerca UC (su albero)

```

function Ricerca-UC-A (problema) returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    frontiera = una coda con priorità con nodo come unico elemento
    loop do   if Vuota?(frontiera) then return fallimento
        nodo = POP(frontiera)
        if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
        for each azione in problema.Azioni(nodo.Stato) do
            figlio = Nodo-Figlio(problema, nodo, azione)
            frontiera = Inserisci(figlio, frontiera) /* in coda con priorità
    end
```

Ricerca-grafo UC

```

function Ricerca-UC-A (problema) returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    frontiera = una coda con priorità con nodo come unico elemento
    esplorati = insieme vuoto
    loop do   if Vuota?(frontiera) then return fallimento
        nodo = POP(frontiera)
        if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
```

```

aggiungi nodo.Stato a esplorati
for each azione in problema.Azioni(nodo.Stato) do
    figlio = Nodo-Figlio(problema, nodo, azione)
    if figlio.Stato non è in esplorati e non è in frontiera then
        frontiera = Inserisci(figlio, frontiera) /* in coda con priorità
    else if figlio.Stato è in frontiera con Costo-cammino più alto then
        sostituisci quel nodo frontiera con figlio
end

```

Analisi

Ottimalità e *completezza* garantite purché il costo degli archi sia maggiore di $\varepsilon > 0$.

Assunto C^* come il costo della soluzione ottima $\lfloor C^*/\varepsilon \rfloor$ è il numero di mosse nel caso peggiore, arrotondato per difetto (e.g. attratto ad andare verso tante mosse di costo ε prima di una che parta più alta ma poi abbia un path a costo totale più basso).

Complessità: $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$. Si noti che quando ogni azione ha lo stesso costo UC somiglia a BF, ma ha complessità $O(b^{1+d})$ causata dall'esame e arresto posticipato: solo dopo aver espanso anche l'ultima frontiera, oltre la profondità del goal.

3.2.12 Confronto delle strategie (albero)

Criterio	BF	UC	DF	DL	ID	Bidir
Completa?	sì	sì (\sim)	no	sì (+)	sì	sì
Tempo	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Spazio	$O(b^d)$	$O(b^{1+\lfloor C^*/\varepsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Ottimale?	sì (*)	sì (\sim)	no	no	sì (*)	sì

(*) se gli operatori hanno tutti lo stesso costo.

(\sim) per costi degli archi $\geq \varepsilon > 0$.

(+) per problemi per cui si conosce un limite alla profondità della soluzione (se $l > d$).

3.3 Conclusioni

Un agente per “problem solving” adotta un paradigma generale di risoluzione dei problemi:

- formula il problema (parte non automatica),
- ricerca la soluzione nello spazio degli stati (diventa automatico).

Strategie “non informate” per la ricerca della soluzione.

Capitolo 4

Ricerca euristica

La ricerca esaustiva non è praticabile in problemi di complessità esponenziale (e.g. 10^{120} configurazioni in scacchi). Noi usiamo la conoscenza del problema e l'esperienza per riconoscere i cammini più promettenti: usiamo una stima del costo futuro *evitando di generare gli altri* (pruning)!

La conoscenza euristica (dal greco “eureka”) aiuta a fare scelte “oculate”: non evita la ricerca, ma la riduce e in genere consente di trovare una **buona** soluzione in tempi accettabili. Sotto certe condizioni garantisce completezza e ottimalità.

4.1 Funzioni di valutazione euristica

Conoscenza del problema data tramite una *funzione di valutazione* f , che include h detta funzione di valutazione euristica:

$$h : n \rightarrow R$$

La funzione si applica al nodo ma dipende solo dallo stato ($n.\text{Stato}$). g dipendeva anche dal cammino fino al *nodo*.

$$f(n) = g(n) + h(n)$$

dove $g(n)$ è il costo cammino visto con UC.

Esempi di euristica h

Per procedere preferibilmente verso il percorso migliore, seguendo “problem-specific information”, di stima del costo futuro: la città più vicina (o la città più vicina alla metà in linea d'aria) nel problema dell'itinerario, il numero

delle caselle fuori posto nel gioco dell'otto, il vantaggio in pezzi nella dama o negli scacchi.

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

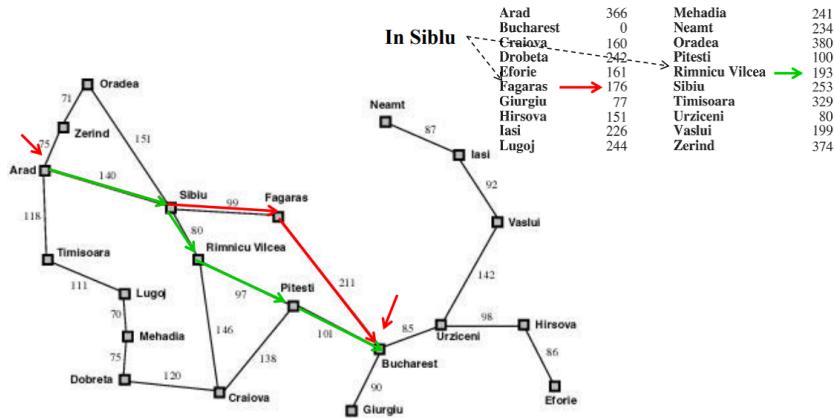
Mappa Romania: distanza in linea d'aria

4.1.1 Algoritmo di ricerca Best-First

Best first - heuristic *con stesso algoritmo di UC* ma con uso di f (stima di costo) per la *coda con priorità*. La scelta di f determina la strategia di ricerca: a ogni passo si sceglie il nodo sulla frontiera per cui il valore della f è migliore (*il nodo più promettente*).

Nota: migliore significa “minore” in caso di un’euristica che stima la distanza della soluzione. Caso speciale: **greedy best-first**, si usa solo h ($f = h$).

Ricerca greedy best-first: esempio $f = h$



Da Arad a Bucarest Greedy best-first: Arad, Sibiu, Fagaras, Bucharest (450); ma non è l’ottimo: Arad, Sibiu, Rimnicu, Pitesti, Bucarest (418).

4.1.2 Algoritmo A: definizione

Si può dire qualcosa di f per avere garanzie di completezza e ottimalità?

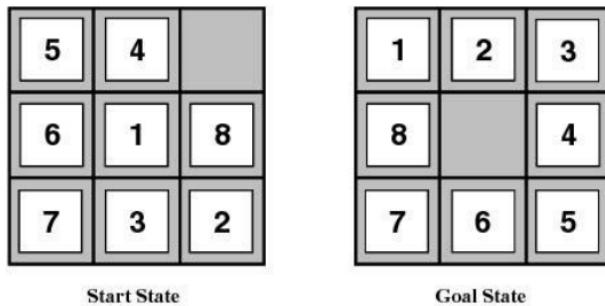
Definizione: un algoritmo A è un algoritmo **Best First** con una funzione di valutazione dello stato del tipo

$$f(n) = g(n) + h(n), \text{ con } h(n) \geq 0 \text{ e } h(goal) = 0$$

- $g(n)$ è il costo del cammino percorso per raggiungere n ;
- $h(n)$ una stima del costo per raggiungere da n un nodo goal.

Vedremo come casi particolari dell'algoritmo A: se $h(n) = 0$ [$f(n) = g(n)$] si ha **Ricerca Uniforme (UC)**, se $g(n) = 0$ [$f(n) = h(n)$] si ha **Greedy Best First**.

Esempio nel gioco dell'otto



$$\begin{aligned} f(n) &= \#\text{mosse fatte} + \#\text{caselle fuori posto} \\ f(\text{Start}) &= 0 + 7 \quad \text{dopo } \leftarrow, \downarrow, \uparrow, \rightarrow \quad f = 4 + 7 \\ f(\text{goalstate}) &=? + 0 \text{ stesso stato, } g \text{ è cambiata.} \end{aligned}$$

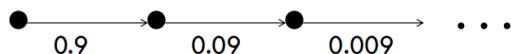
L'algoritmo A è completo

Teorema 4.1.1. *L'algoritmo A con la condizione*

$$g(n) \geq d(n) \cdot \varepsilon$$

dove $\varepsilon > 0$ costo minimo arco e d è la profondità, è completo.

Nota: la condizione ci garantisce che non si verifichino situazioni strane del tipo



e che il costo lungo un cammino non cresca "abbastanza"; se cresce abbastanza possiamo fermare quel path per costo alto di g .

Dimostrazione di completezza. Sia $[n_0 \ n_1 \ n_2 \ \dots \ n' \ \dots \ n_k = goal]$ un cammino soluzione. Sia n' un nodo della frontiera su un cammino soluzione: n' prima o poi sarà espanso, infatti esistono solo un numero finito di nodi x che possono essere aggiunti alla frontiera con $f(x) \leq f(n')$ (è la condizione sulla crescita di g t.c. non esista una catena infinita di archi e nodi che possa aggiungere con costo sempre $\leq f(n')$).

Quindi, se non si trova una soluzione prima, n' verrà espanso e i suoi successori aggiunti alla frontiera, tra questi anche il suo successore sul cammino soluzione. Il ragionamento si può ripetere fino a dimostrare che anche il nodo *goal* sarà selezionato per l'espansione.

□

4.1.3 Algoritmo A*

La stima ideale

Funzione di valutazione ideale (oracolo):

$$f^*(n) = g^*(n) + h^*(n)$$

$g^*(n)$ costo del cammino minimo da radice a n .

$h^*(n)$ costo del cammino minimo da n a *goal*.

$f^*(n)$ costo del cammino minimo da radice a *goal*, attraverso n .

Normalmente: $g(n) \geq g^*(n)$ e $h(n)$ è una stima di $h^*(n)$. Si può andare in sottostima (e.g. linea d'aria) o sovrastima della distanza dalla soluzione.

Definizione 4.1.1. Si dice che un'euristica è ammissibile se

$$\forall n. \ h(n) \leq h^*(n)$$

h è una **sottostima**. Per esempio l'euristica della distanza in linea d'aria.

Definizione 4.1.2. Un **Algoritmo A*** è un algoritmo A in cui h è una funzione euristica ammissibile.

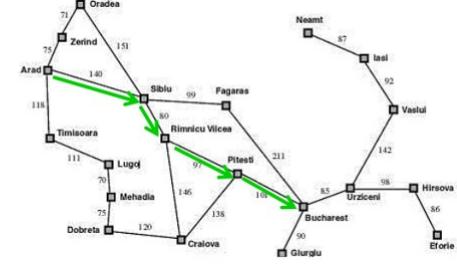
Teorema 4.1.2. *Gli algoritmi A* sono ottimali.*

Corollario 4.1.1. *BF (con passi a costo costante) e UC sono ottimali*

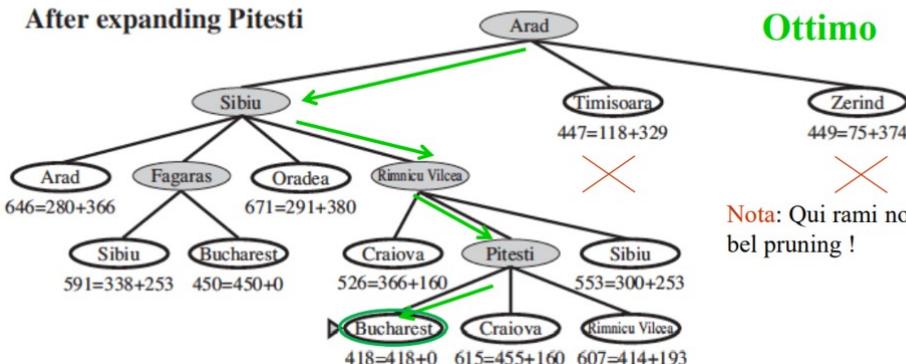
$$h(n) = 0$$

Itinerario con A*

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



After expanding Pitesti



Ottimo

Nota: Qui rami non espansi:
bel pruning !

Osservazioni su A*

1. Rispetto a greedy best-first, la componente g fa sì che si abbandonino cammini che vanno troppo in profondità.
2. h sotto o sovra-stima?
 - Una sottostima (h) può farci compiere del lavoro inutile (tenendo anche candidati non buoni), però non ci fa perdere il cammino migliore (quando prendo nodo goal è il cammino migliore).
 - Una funzione che qualche volta sovrastima può farci perdere la soluzione ottimale (taglio per causa di sovrastima, invece era buona).

Ottimalità di A*

Nel caso di ricerca a/su albero l'uso di un'euristica *ammissibile* è sufficiente a garantire l'ottimalità di A*. Nel caso di ricerca su grafo serve una proprietà più forte, la **consistenza** (detta anche **monotonicità**), per evitare il rischio

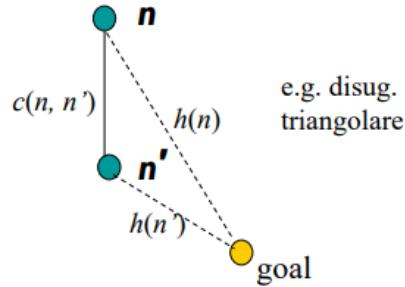
di scartare candidati ottimi (stato già incontrato) a causa dell'uso della lista esplorati che fa "sparire", o meglio, non considera candidati ottimali al momento dell'espansione. Cerchiamo quindi condizioni per garantire che il primo *espanso* sia il migliore.

4.1.4 Euristica consistente o monotona

Definizione: euristica **consistente**

$$[h(goal) = 0] \\ \forall n. h(n) \leq c(n, a, n') + h(n') \quad \text{dove } n' \text{ è un successore di } n.$$

Ne segue che $f(n) \leq f(n')$.



Nota: se h è consistente la f non decresce mai lungo i cammini, da cui il termine **monotona**.

Proprietà

Teorema: Un'euristica monotona è ammissibile.

Esistono euristiche ammissibili che non sono monotone, ma sono rare*.

Le euristiche monotone garantiscono che **la soluzione meno costosa venga trovata per prima** e quindi sono ottimali anche nel caso di ricerca su grafo. Non si devono recuperare tra gli antenati nodi con costo minore: lista degli esplorati, stato già esplorato è sul cammino ottimo → posso evitare di inserire il corrente ripetuto senza perdere l'*ottimalità*.

```
if figlio.Stato non è in esplorati e non è in frontiera then
    frontiera = Inserisci(figlio, frontiera)
```

Per la frontiera, volendo evitare stati ripetuti, resta "if" finale di UC

```
if figlio.Stato è in frontiera con Costo-cammino più alto then
    sostituisci quel nodo frontiera con figlio
```

4.1.5 Ottimalità di A*

Se $h(n)$ è consistente i valori di $f(n)$ lungo un cammino sono non decrescenti:

$$\begin{aligned}
 & \text{Se } h(n) \leq c(n, a, n') + h(n') && \text{def. consistenza} \\
 & g(n) + h(n) \leq g(n) + c(n, a, n') + h(n') && \text{sommendo } g(n) \\
 & \text{ma siccome } g(n) + c(n, a, n') = g(n') \\
 & g(n) + h(n) \leq g(n') + h(n') \\
 & f(n) \leq f(n') && \rightarrow f \text{ monotona}
 \end{aligned}$$

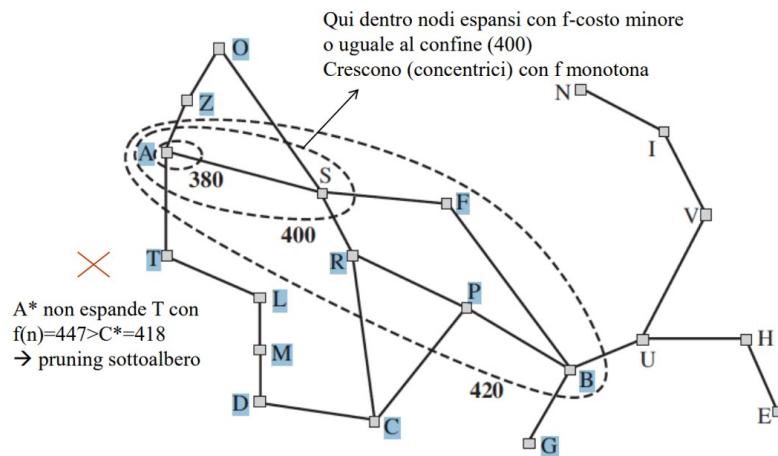
Ogni volta che A* seleziona un nodo (n) per l'*espansione*, il cammino ottimo a tale nodo è stato trovato: se così non fosse, ci sarebbe un altro nodo n' della frontiera sul cammino ottimo (che va a n con un cammino ottimo ancora da trovare) con $f(n')$ minore (per la monotonia e n successore di n'); ma ciò non è possibile perché tale nodo sarebbe già stato espanso (si espande prima un nodo con f minore).

Quando seleziona nodo goal è cammino ottimo ($h = 0, f = C^*$).

Perché A* è vantaggioso?

- A* espande tutti i nodi con $f(n) < C^*$ (C^* = costo ottimo)
- A* espande alcuni nodi con $f(n) = C^*$
- **A* non espande alcun nodo con $f(n) > C^*$**

Quindi alcuni nodi (e suoi sottoalberi) non verranno considerati per l'espansione (ma restiamo ottimali): **pruning** (h opportuna, più alta possibile tra le ammissibili, fa tagliare molto).



Più f è aderente a stima ottimale, più taglio! Ovali più stretti. Cercheremo quindi una h il più alta possibile tra le ammissibili. Se molto bassa molti (sino a tutti i) nodi restano minore di C^* → espando tutti (a cerchi). Il pruning sotto-alberi è il punto focale: non li abbiamo già in memoria e evitiamo di generarli (decisivo per i problemi di AI a spazio stati esponenziali).

Bilancio su A*

- A* è **completo**: discende dalla completezza di A (A* è un algoritmo A particolare).
- A* con euristica monotona è **ottimale**.
- A* è **ottimamente efficiente**: a parità di euristica nessun altro algoritmo espande meno nodi (senza rinunciare a ottimalità).

Problemi: “Quale euristica?” e ancora l’occupazione di memoria ($O(b^{d+1})$).

4.1.6 Su f : Due sotto-casi speciali

Casi particolari dell’algoritmo A

- Se $h(n) = 0$ [$f(n) = g(n)$] si ha Uniform Cost, ossia g non basta.
- Se $g(n) = 0$ [$f(n) = h(n)$] si ha Greedy Best First, ossia h non basta (già visto all’inizio).

Dijkstra

L’algoritmo A* è una generalizzazione dell’algoritmo di Dijkstra che riduce la dimensione del sottografo che deve essere esplorato, se è disponibile un limite inferiore sulla “distanza” dal bersaglio (h).

4.2 (Costruire) le euristiche di A*

Valutazione di funzioni euristiche

A parità di ammissibilità, una euristica può essere più efficiente di un'altra nel trovare il cammino soluzione migliore (visitare meno nodi). Questo dipende da quanto *informata* è l'euristica (dal **grado di informazione posseduto**).

$$\begin{aligned} h(n) &= 0 \quad \text{minimo di informazione (BF o UC)} \\ h^*(n) &\quad \text{massimo di informazione (oracolo)} \end{aligned}$$

In generale, per le euristiche ammissibili:

$$0 \leq h(n) \leq h^*(n)$$

Più informata, più efficiente

Teorema 4.2.1. Se $h_1 \leq h_2$, i nodi espansi da A^* con h_2 sono un sottoinsieme di quelli espansi da A^* con h_1 .

A^* espande tutti i nodi con $f(n) = g(n) + h(n) < C^*$, e sono meno per h maggiore (h maggiore fa andare più nodi oltre C^*). Se $h_1 \leq h_2$, A^* con h_2 è almeno efficiente quanto A^* con h_1 . Un'euristica più informata (accurata) riduce lo spazio di ricerca (è più efficiente), ma è tipicamente più costosa da calcolare.

Confronto di euristiche ammissibili

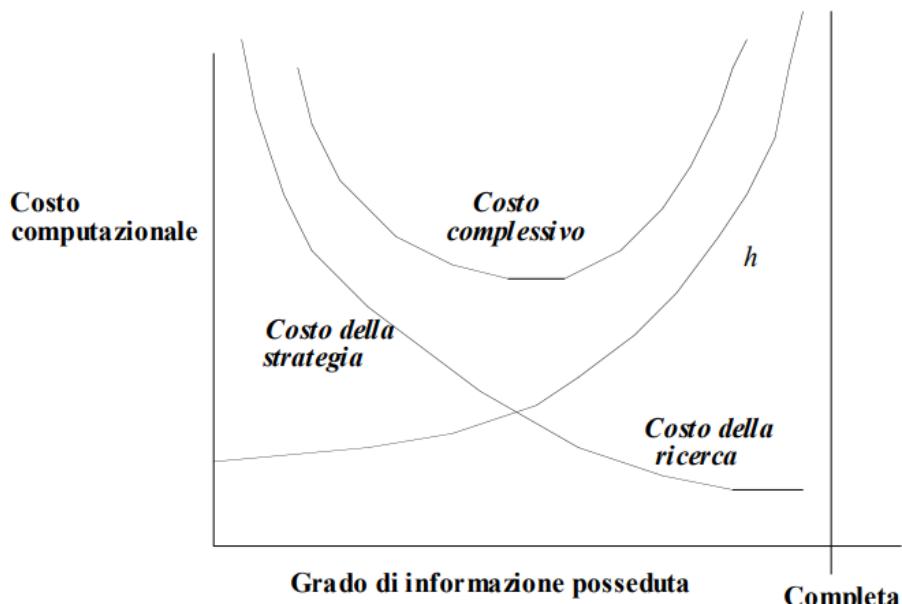
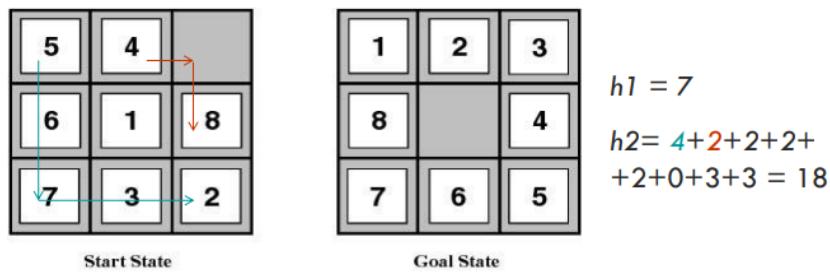
Due euristiche ammissibili per il gioco dell'8

- h_1 : conta il numero di caselle fuori posto
- h_2 : somma delle distanze Manhattan¹ (orizz./vert.) delle caselle fuori posto dalla posizione finale.

h_2 è più informata di h_1 , infatti $\forall n. h_1(n) \leq h_2(n)$.

Definizione: h_2 domina h_1 .

¹ $h((x, y)) = MD((x, y), (x_g, y_g)) = |x - x_g| + |y - y_g|$



Costo ricerca vs costo euristica

4.2.1 Misura del potere euristico

Come valutare gli algoritmi di ricerca euristica...

*Fattore di diramazione effettivo b^**

N : numero di nodi generati

d : profondità della soluzione

b^* è il fattore di diramazione di un albero uniforme con $N+1$ nodi; soluzione dell'equazione

$$N + 1 = b^* + (b^*)^2 + \dots + (b^*)^d$$

Sperimentalmente una buona euristica ha un b^* abbastanza vicino a 1 (< 1.5).

d	ID (appr. it. non inf)	$A^*(h_1)$	$A^*(h_2)$
2	10 (2,43)	6 (1,79)	6 (1,79)
4	112 (2,87)	13 (1,48)	12 (1,45)
6	680 (2,73)	20 (1,34)	18 (1,30)
8	6384 (2,80)	39 (1,33)	25 (1,24)
10	47127 (2,79)	93 (1,38)	39 (1,22)
12	3644035 (2,78)	227 (1,42)	73 (1,24)
14	-	539 (1,44)	113 (1,23)
...

Esempio: dal gioco dell'otto

Sono riportati: nodi generati e fattore di diramazione effettivo (b^*). I dati sono mediati, per ogni d, su 100 istanze del problema.

Capacità di esplorazione

Influenza di b^* :

Con $b = 2$

$$\begin{array}{ll} d=6 & N=100 \\ d=12 & \leftarrow N=10.000 \end{array}$$

con $b = 1.5$

$$\begin{array}{ll} d=12 & N=100 \\ d=24 & \leftarrow N=10.000 \end{array}$$

Migliorando di poco l'euristica si riesce, a parità di nodi espansi, a raggiungere una profondità doppia!

Quindi...

1. Tutti i problemi dell'IA (o quasi) sono di complessità esponenziale... (nel generare nodi, i.e. configurazioni possibili) ma c'è esponenziale e esponenziale!
2. L'euristica può migliorare di molto la capacità di esplorazione dello spazio degli stati rispetto alla ricerca cieca
3. Migliorando anche di poco l'euristica si riesce ad esplorare uno spazio molto più grande (più in profondità).

4.2.2 Come si inventa un'euristica?

Alcune strategie per ottenere euristiche ammissibili:

- Rilassamento del problema
- Massimizzazione di euristiche
- Database di pattern disgiunti
- Combinazione lineare
- Apprendere dall'esperienza

Rilassamento del problema

Nel gioco dell'8 mossa da A a B possibile se

1. B adiacente a A
2. B libera

h_1 e h_2 sono calcoli della distanza esatta della soluzione in versioni semplificate del puzzle:

- h_1 (nessuna restrizione, ne 1 ne 2): sono sempre ammessi scambi a piacimento tra caselle (si muove ovunque) \rightarrow #caselle fuori posto.
- h_2 (solo restrizione 1): sono ammessi spostamenti anche su caselle occupate, purché adiacenti \rightarrow somma delle distanze Manhattan.

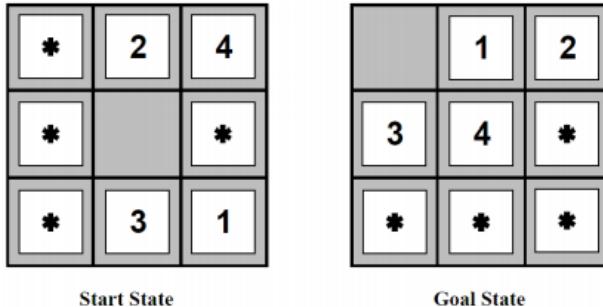
Massimizzazione di euristiche

Se si hanno una serie di euristiche ammissibili h_1, h_2, \dots, h_k **senza che nessuna “domini” un’altra** allora conviene prendere il massimo dei loro valori:

$$h(n) = \max \{(h_1(n), h_2(n), \dots, h_k(n))\}$$

Se le h_i sono ammissibili anche la h lo è. La h domina tutte le altre.

Euristiche da sottoproblemi



Costo della soluzione ottima al sottoproblema (di sistemare 1,2,3,4) è una sottostima del costo per il problema nel suo complesso (rilevatesi più accurata della Manhattan).

Database di pattern: memorizzare ogni istanza del sottoproblema con relativo costo della soluzione. Usare questo database per calcolare h_{DB} (estraendo dal DB la configurazione corrispondente allo stato completo corrente). Potremmo poi fare la stessa cosa per altri sottoproblemi: 5-6-7-8, 2-4-6-8, ... ottenendo altre euristiche ammissibili. Poi prendere il valore massimo: ancora una euristica ammissibile.

Ma potremmo sommarle e ottenere un'euristica ancora più accurata?

In generale no perchè le soluzioni ai sottoproblemi interferiscono (condividono alcune mosse, se sposto 1-2-3-4, sposterò anche 4-5-6-7) e la somma delle euristiche in generale non è ammissibile (potremmo sovrastimare avendo avuto aiuti mutui). Si deve eliminare il costo delle mosse che contribuiscono all'altro sottoproblema.

Database di **pattern disgiunti** consentono di sommare i costi (euristiche additive) [e.g. solo costo mosse su 1-2-3-4]. Sono molto efficaci: gioco del 15 in pochi ms. Difficile scomporre per cubo Rubik.

Apprendere dall'esperienza

Far girare il programma, raccogliere dati: coppie $\langle \text{stato}, h^* \rangle$. Usare i dati per apprendere a predire la h con algoritmi di apprendimento induttivo (da istanze note stimiamo h in generale).

Gli algoritmi di apprendimento si concentrano su caratteristiche salienti dello stato (*feature*, x_i) [e.g. apprendiamo che da numero tasselli fuori posto 5 \rightarrow costo ~ 14 , etc].

Combinazione di euristiche

Quando diverse caratteristiche influenzano la bontà di uno stato, si può usare una combinazione lineare

$$h(n) = c_1 x_1(n) + c_2 x_2(n) + \dots + c_k x_k(n)$$

Gioco dell'8:

$$h(n) = c_1 \#fuori-posto + c_2 \#\text{coppie-scambiate}$$

Scacchi:

$$h(n) = c_1 \text{ vant-pezzi} + c_2 \text{ pezzi-attacc.} + c_3 \text{ regina} + \dots$$

Il peso dei coefficienti può essere aggiustato con l'esperienza, anche qui apprendendo automaticamente da esempi di gioco. $h(goal) = 0$ (e.g. gioco dell'8), ma ammissibilità e consistenza non automatiche.

4.3 Algoritmi evoluti basati su A*

Beam search

Nel Best First viene tenuta tutta la frontiera; se l'occupazione di memoria è eccessiva si può ricorrere ad una variante: la *Beam search*. La **Beam Search** tiene ad ogni passo solo $i k$ nodi più promettenti, dove k è detto *l'ampiezza del raggio* (*beam*).

La *Beam Search* non è completa.

IDA*

IDA* combina A* con ID: ad ogni iterazione si ricerca **in profondità** con un limite (cut-off) dato dal valore della **funzione f** (e non dalla profondità); il limite $f\text{-limit}$ viene aumentato ad ogni iterazione, fino a trovare la soluzione. Punto critico: di quanto viene aumentato $f\text{-limit}$.

Cruciale la scelta dell'incremento per garantire l'ottimalità. Nel caso di costo delle azioni fisso è chiaro: il limite viene incrementato del costo delle azioni. Nel caso che i costi delle azioni siano variabili? Costo minimo oppure si potrebbe ad ogni passo fissare il limite successivo al valore minimo delle f scartate (in quanto superavano il limite) all'iterazione precedente.

IDA* completo e ottimale

- Se le azioni hanno costo costante k (caso tipico 1) e $f\text{-limit}$ viene incrementato di k .

- Se le azioni hanno costo variabile e l'incremento di f -*limit* è $\leq \varepsilon$ (minimo costo degli archi)
- Se il nuovo f -*limit* = min. valore f dei nodi generati ed esclusi all'iterazione precedente.

Occupazione di memoria: $O(bd)$ [dall'algoritmo DF].

Capitolo 5

Oltre la ricerca classica

Risolutori “classici”

Gli agenti risolutori di problemi “classici” assumono:

- Ambienti completamente osservabili
- Ambienti deterministici
- Sono nelle condizioni di produrre offline un piano (una sequenza di azioni) che può essere eseguito senza imprevisti per raggiungere l’obiettivo.

5.1 Ricerca locale

Assunzioni per ricerca locale

Gli algoritmi visti esplorano gli spazi di ricerca alla ricerca di un goal e restituiscono un **cammino soluzione**. Ma a volte lo stato **goal è la soluzione** del problema. Gli algoritmi di ricerca locale sono adatti per problemi in cui:

- la sequenza di azioni non è importante, quello che conta è unicamente lo stato goal;
- tutti gli elementi della soluzione sono nello stato ma alcuni vincoli sono violati, per esempio le regine nella formulazione a stato completo.

5.1.1 Algoritmi di ricerca locale

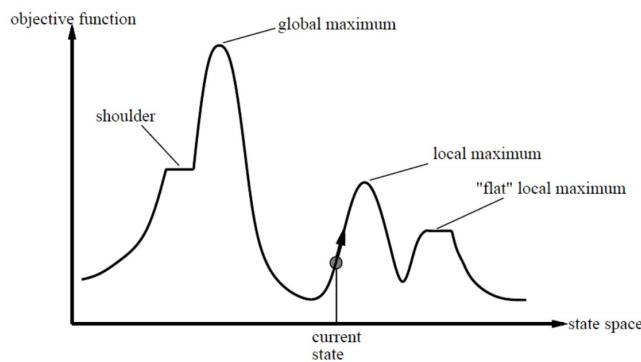
Non sono sistematici. Tengono traccia solo del nodo corrente e si spostano su nodi adiacenti, non tengono traccia dei cammini (non servono in uscita!)

1. Efficienti in occupazione di memoria.

- Possono trovare soluzioni ragionevoli anche in spazi molti grandi e infiniti, come nel caso di spazi continui.

Utili per risolvere problemi di ottimizzazione: lo stato migliore secondo una funzione obiettivo, lo stato di costo minore, ...

Panorama dello spazio degli stati



f euristica di costo della funzione obiettivo (non del cammino)

Uno stato ha una posizione sulla superficie e una altezza che corrisponde al valore della funzione di valutazione (funzione obiettivo). Un passo di un algoritmo provoca movimento sulla superficie.

Trovare l'avvallamento più basso (e.g. min costo) o il picco più alto (e.g. max di un obiettivo).

5.1.2 Ricerca in salita (Hill climbing) steepest ascent/descent

Ricerca locale **greedy**. Vengono generati i successori e valutati; viene scelto un nodo che migliora la valutazione dello stato attuale (non si tiene traccia degli altri [no albero di ricerca in memoria]):

- il migliore → Hill climbing a salita rapida
- uno a caso (tra quelli che salgono) → Hill climbing stocastico (anche dipendendo da pendenza)
- il primo → Hill climbing con prima scelta

Se non ci sono stati successori migliori l'algoritmo termina con fallimento.

L'algoritmo Hill climbing

```

function Hill-climbing (problema)
    returns uno stato che è un massimo locale
    nodo-corrente = CreaNodo(problema.Stato-iniziale)
    loop do
        vicino = il successore di nodo-corrente di valore più alto
        if vicino.Valore  $\leq$  nodo-corrente.Valore then
            return nodo-corrente.Stato // interrompe la ricerca
        nodo-corrente = vicino
        // (altrimenti, se vicino è migliore, continua)
    
```

Nota: si prosegue solo se il vicino (più alto) è migliore dello stato corrente
→ Se tutti i vicini sono peggiori si ferma. Non c'è frontiera a cui ritornare, si tiene un solo stato.

Tempo: numero cicli variabile in base al punto di partenza.

Il problema delle 8 regine (formulazione a stato completo)

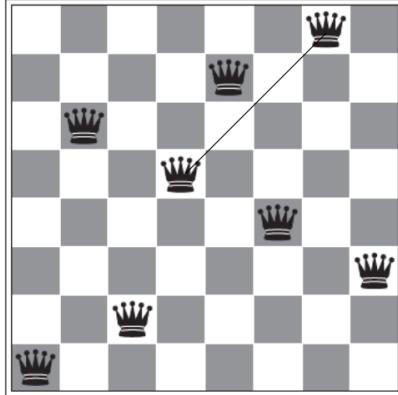
Costo h: numero di coppie di regine che si attaccano a vicenda (valore 17 nella figura sotto). Si cerca il minimo.

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	14	14	13	16	13
14	14	17	15	15	14	16	16
17	14	16	18	15	15	15	15
18	14	14	15	15	14	14	16
14	14	13	17	12	14	12	18

I numeri sono i valori dei successori (7x8) [7 posizioni per ogni regina = su ogni colonna]. Tra i migliori (valore 12) si sceglie a caso. Minimo globale = 0.

Un minimo locale

$h = 1$



Tutti gli stati successori non migliorano la situazione (minimo locale). Per le 8 regine Hill climbing si blocca l'86% delle volte, ma in media solo quattro passi per la soluzione e tre quando si blocca su $8^8 = 17$ milioni di stati.

Problemi con Hill-climbing

Se la funzione è da ottimizzare i picchi sono massimi locali o soluzioni ottimali.

- Massimi locali
- Pianori o spalle
- Crinali (o creste)

Miglioramenti

Consentire un numero limitato di mosse laterali (ossia ci si ferma per $<$ nell'algoritmo invece che per \leq). L'algoritmo sulle 8 regine ha successo nel 94%, ma impiega in media 21 passi.

Hill-climbing stocastico: si sceglie a caso tra le mosse in salita (magari tenendo conto della pendenza). Converge più lentamente ma a volte trova soluzioni migliori.

Hill-climbing con prima scelta: può generare le mosse a caso, una alla volta, fino a trovarne una migliore dello stato corrente (si prende il primo migliore). Come la stocastica ma utile quando i successori sono molti (e.g. migliaia).

Hill-Climbing con riavvio casuale (random restart): ripartire da un punto scelto a caso. Se la probabilità di successo è p saranno necessarie in media $1/p$ ripartenze per trovare la soluzione; per esempio 8 regine, $p =$

0.14, 7 iterazioni: 6 fail, un successo. Hill-climbing con random-restart è tendenzialmente completo, insistendo si generano tutte! Per le regine: caso con 3 milioni di regine in meno di un minuto! Se funziona o no dipende molto dalla forma del panorama degli stati.

5.1.3 Tempra simulata

L'algoritmo di **tempra simulata** (*Simulated annealing*) [Kirkpatrick, Gelett, Vecchi 1983] combina hill-climbing con una scelta stocastica (ma non del tutto casuale, perché poco efficiente...). Analogia con il processo di tempra dei metalli in metallurgia: i metalli vengono portati a temperature molto elevate (alta energia/stocasticità iniziale) e raffreddati gradualmente consentendo di cristallizzare in uno stato a (più) bassa energia.

Ad ogni passo si sceglie un successore n' a caso:

- se migliora lo stato corrente viene espanso,
- altrimenti (caso in cui $\Delta E = f(n') - f(n) < 0$) quel nodo viene scelto con probabilità $p = e^{\Delta E/T}$ $[0 \leq p \leq 1]$.

Si genera un numero casuale tra 0 e 1: se questo è $< p$ il successore viene scelto, altrimenti no. Ossia: p è inversamente proporzionale al peggioramento; infatti se la mossa peggiora molto, ΔE alto negativo, la p si abbassa.

T (temperatura) decresce col progredire dell'algoritmo (quindi anche p) secondo un piano definito. Col progredire rende improbabili le mosse peggiorative.

Tempra simulata: analisi

La probabilità di una mossa in discesa diminuisce col tempo e l'algoritmo si comporta sempre di più come Hill Climbing. Se T viene decrementato abbastanza lentamente con probabilità tendente ad 1 si raggiunge la soluzione ottimale. Analogia col processo di tempra dei metalli:

- T corrisponde alla temperatura
- ΔE alla variazione di energia

Tempra simulata: parametri

Valore iniziale e decremento di T sono parametri. Valori per T determinati sperimentalmente: il valore *iniziale* di T è tale che per valori medi di ΔE , $p = e^{\Delta E/T}$ sia all'incirca 0.5.

5.1.4 Ricerca local beam

La versione locale della *beam search*. Si tengono in memoria k stati, anziché uno solo, e ad ogni passo si generano i successori di tutti i k stati. Se si trova un goal ci si ferma, altrimenti si prosegue con i k migliori tra questi.

Note:

- diverso da K restart (che riparte da zero)
- diverso da beam search: perché?
- $k = 1$ o illimitato a cosa portano?

5.1.5 Beam search stocastica

Si introduce un elemento di casualità come in un processo di **selezione naturale** (diversificare la nuova generazione). Nella variante stocastica della *local beam*, si scelgono k successori, ma con probabilità maggiore per i migliori. La terminologia:

- *organismo* [stato]
- *progenie* [successori]
- *fitness* [il valore della f], idoneità

Algoritmi genetici

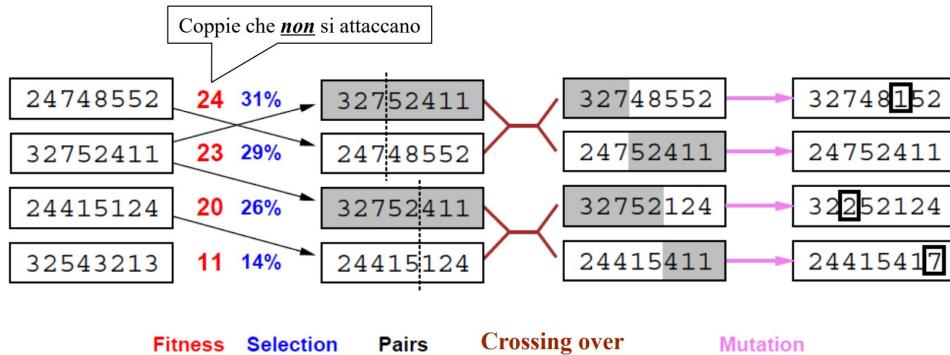
Sono **varianti** della **beam search stocastica** in cui gli stati successori sono ottenuti combinando due stati genitore (anziché per evoluzione). La terminologia:

- **popolazione di individui** [stati]
- **fitness**
- **accoppiamenti + mutazione genetica**
- **generazioni**

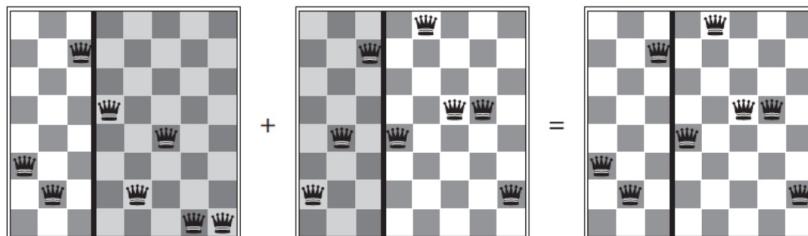
Popolazione iniziale: k stati/**individui** generati casualmente e ogni individuo è rappresentato come una stringa. Gli individui sono valutati da una **funzione di fitness**.

Si **selezionano** gli individui per gli “**accoppiamenti**” con una probabilità proporzionale alla fitness. Le coppie danno vita alla generazione successiva

combinando materiale genetico (crossover) e con un meccanismo aggiuntivo di mutazione genetica (casuale), la popolazione ottenuta dovrebbe essere migliore. La cosa si ripete fino ad ottenere stati abbastanza buoni (stati obiettivo).



Per ogni coppia viene scelto un punto di *crossing over* e vengono generati due figli scambiandosi pezzi (del DNA). Viene infine effettuata una *mutazione* casuale che dà luogo alla prossima generazione.



Le parti chiare sono passate al figlio, le parti grigie si perdono.

Se i genitori sono molto diversi anche i nuovi stati sono diversi. All'inizio spostamenti maggiori che poi si raffinano.

Suggestivi (area del *Natural computing*: e.g. swarm, ...). Usati in molti problemi reali e.g. configurazione di circuiti e scheduling di lavori. Combinano la tendenza a salire della beam search stocastica e l'interscambio di informazioni tra thread paralleli di ricerca (blocchi utili che si combinano).

Funziona meglio se il problema (soluzioni) ha componenti significative rappresentate in sottostringhe → **punto critico**: la rappresentazione del problema in stringhe.

5.2 Spazi continui

Molti casi reali hanno spazi di ricerca continua, e.g. fondamentale per Machine Learning!

Stato descritto da variabili continue, x_1, \dots, x_n , vettore \mathbf{x} . Prendiamo ad esempio movimenti in spazio 3D, con posizione data da $x = (x_1, x_2, x_3)$.

Apparentemente ostico, fattori di ramificazione infiniti con gli approcci precedenti. In realtà ci sono molti strumenti matematici per spazi continui, che portano ad approcci anche molto efficienti...

Gradient

Se la f è continua e differenziabile, e.g. quadratica rispetto ad \mathbf{x} (vettore). Il minimo o massimo si può cercare utilizzando il gradiente, che restituisce la direzione di massima pendenza nel punto.

Data f obiettivo

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right)$$

Hill climbing iterativo:

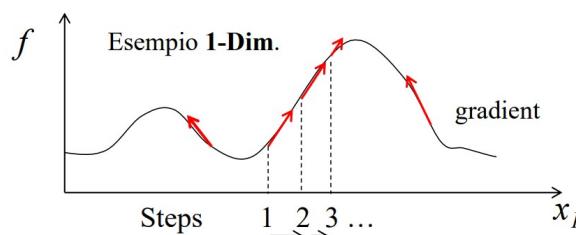
$$x_{new} = x + \eta \nabla f(x)$$

Quantifica lo spostamento, senza cercarlo tra gli infiniti possibili successori!

Uso + per salire (maximization).

Uso - per scendere (minimization).

η (eta): step size.



Spostamenti guidati dal gradiente

Nota generale: non sempre è necessario il min/max assoluto.

5.3 Assunzioni sull'ambiente da riconsiderare

Ambienti più realistici

Gli agenti *risolutori di problemi* “classici” assumono:

- ambienti completamente osservabili;
- azioni/ambienti deterministici;
- il piano generato è una sequenza di azioni che può essere generato *offline* e eseguito senza imprevisti;
- le percezioni non servono se non nello stato iniziale.

Soluzioni più complesse

In un ambiente parzialmente osservabile e non deterministico le **percezioni** sono importanti perché restringono gli stati possibili e informano sull’effetto dell’azione. Più che un piano l’agente può elaborare una “*strategia*”, che tiene conto delle diverse eventualità: un **piano con contingenza**.

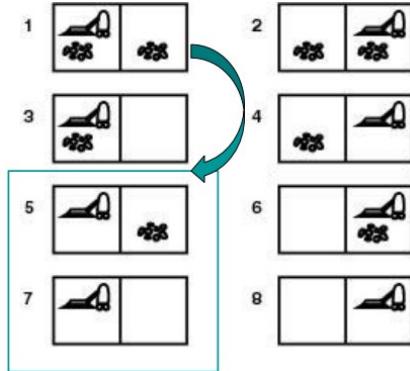
5.3.1 Azioni non deterministiche

L’aspirapolvere imprevedibile

Comportamento: se aspira in una stanza sporca la pulisce, ma talvolta pulisce anche una stanza adiacente; se aspira in una stanza pulita, a volte rilascia sporco.

Variazioni necessarie al modello:

- Il modello di transizione restituisce un insieme di stati: l’agente non sa in quale si troverà.
- Il piano di contingenza sarà un piano condizionale e magari con cicli.



Esempio: $Risultati(\text{Aspira}, 1) = \{5, 7\}$

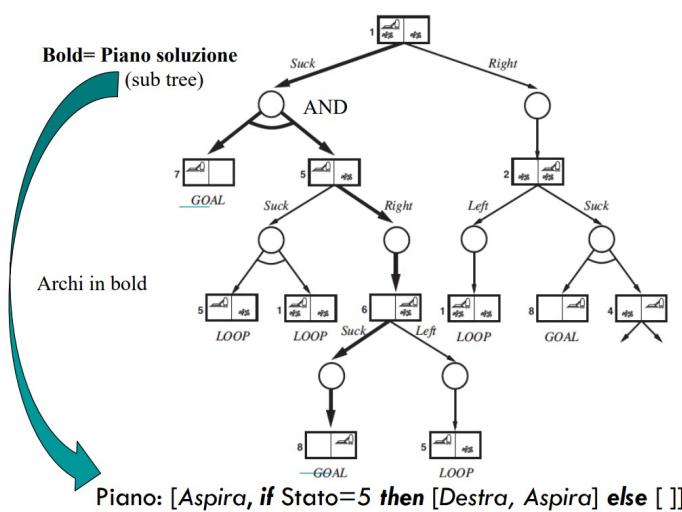
Piano possibile

```
[Aspira,
  if stato=5
then [Destra, Aspira]
  else [ ]
]
```

Come si pianifica: alberi di ricerca AND-OR

Nodi OR le scelte dell'agente (1 sola azione). Nodi AND le diverse contingenze (le scelte dell'ambiente, più stati possibili), da considerare tutte.

Una soluzione a un problema di ricerca AND-OR è un **albero** che ha un nodo obiettivo in ogni foglia: specifica un'unica azione nei nodi OR e include tutti gli archi uscenti da nodi AND (tutte le contingenze).



Capitolo 6

I giochi con avversario

I giochi che potremo esplorare hanno regole semplici e sono formalizzabili, in particolare avranno un ambiente accessibile e deterministico e vi saranno due giocatori con turni alterni, a **somma zero**. Si tratta di giochi **a informazione perfetta**.

Siamo nel contesto di un ambiente multi-agente **competitivo**: la presenza dell'avversario rende l'ambiente **strategico** ⇒ più difficile rispetto ai problemi di ricerca visto finora.

Complessità e vincoli di tempo reale: la mossa migliore nel tempo disponibile.

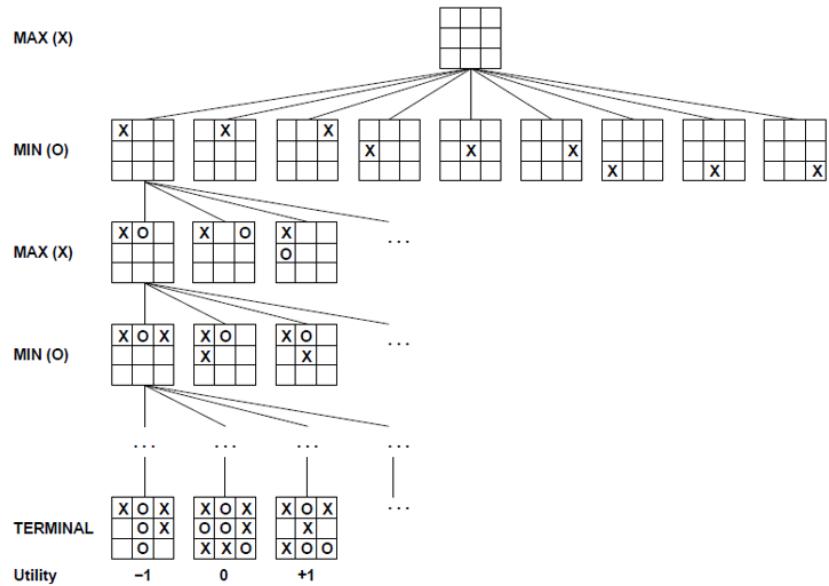
I giochi sono un po' più simili ai problemi reali.

Ciclo pianifica-agisci-percepisci

Caso di due agenti che agiscono “a turno”: si può ancora pianificare considerando le possibili risposte dell'avversario e le possibili risposte a queste risposte...

Una volta decisa la mossa migliore da fare:

- si esegue la mossa,
- si vede cosa fa l'avversario,
- si ri-pianifica la prossima mossa.

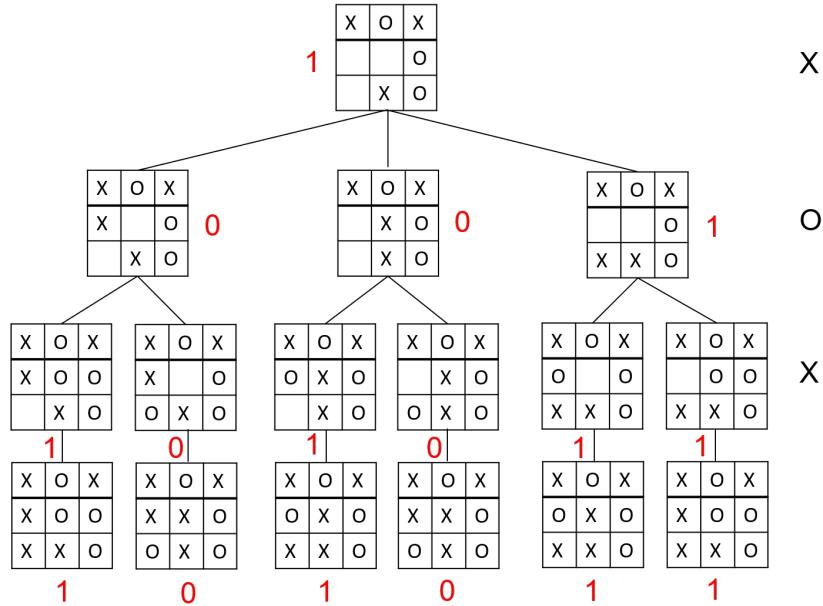


Albero di gioco parziale

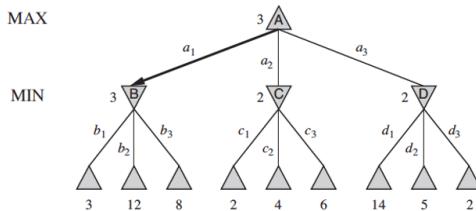
6.1 Giochi come problemi di ricerca

- *Stati*: configurazioni del gioco.
- *Player(s)*: a chi tocca muovere nello stato s .
- *Stato iniziale*: configurazione iniziale del gioco.
- *Actions(s)*: le mosse legali in s .
- *Result(s,a)*: lo stato risultante da una mossa, è il modello di transizione.
- *Terminal-Test(s)*: determina la fine del gioco.
- *Utility(s, p)*: funzione di *utilità* (o *pay-off*), valore numerico che valuta gli stati terminali del gioco per p , per esempio $1| -1| 0$, conteggio punti, ... l'importante è che la somma sia costante.

6.1.1 Algoritmo Min-Max



Esempio di una mossa (e contromossa)



Albero di gioco profondo "una mossa", due livelli o **ply**

Calcolo del valore minimax dei nodi: il triangolo con la punta in su rappresenta un nodo MAX, mentre quello con la punta in giù un nodo MIN. Le foglie sono situazioni terminali di gioco.

Valore Minimax

$\text{Minimax}(s) =$

$$\begin{aligned}
 & \text{Utility}(s, \text{MAX}) && \text{if } \text{Terminal-Test}(s) \\
 & \max_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a)) && \text{if } \text{Player}(s) = \text{MAX} \\
 & \min_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s, a)) && \text{if } \text{Player}(s) = \text{MIN}
 \end{aligned}$$

Ma come conviene esplorare l'albero di gioco?

Algoritmo MIN-MAX ricorsivo

```

function MINIMAX-DECISION(state) returns an action
  return  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(s, a))$ 

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow \infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
  return v
```

Costo

Tempo: come DF, $O(b^m)$; spazio: $O(m)$.

Scacchi: 35^{100} (35 mosse in media; 50 mosse per player); grafo degli scacchi 10^{40} nodi $\rightarrow 10^{22}$ secoli!

Improprio un'esplorazione sistematica del grafo degli stati, se non per giochi veramente semplici (come il filetto). È necessario fare uso di **euristiche** per stimare la bontà di uno stato del gioco.

Min-Max euristico (con orizzonte)

In casi più complessi occorre una **funzione di valutazione euristica** dello stato, $Eval(s)$. Strategia: guardare avanti d livelli

- si espande l'albero di ricerca un certo numero di livelli d (compatibile col tempo e lo spazio disponibili),
- si valutano gli stati ottenuti e si propaga indietro il risultato con la regola del MAX e MIN.

Algoritmo MIN-MAX come prima ma...

if $Terminal - Test(s)$ **then return** $Utility(s)$ diventa
if $Cutoff-Test(s, d)$ **then return** $Eval(s)$

Cutoff-Test riconosce stati terminali.

Valore H-Minimax

Se d è il limite alla profondità consentita . . .

$H\text{-Minimax}(s, d) =$

$$\begin{aligned} & Eval(s) && \text{if } Cutoff-Test(s, d) \\ & \max_{a \in Actions(s)} H\text{-Minimax}(Result(s, a), d + 1) && \text{if } Player(s) = \mathbf{MAX} \\ & \min_{a \in Actions(s)} H\text{-Minimax}(Result(s, a), d + 1) && \text{if } Player(s) = \mathbf{MIN} \end{aligned}$$

Il filetto

$$Eval(s) = X(s) - O(s)$$

- $X(s)$: righe aperte per X
- $O(s)$: righe aperte per O

Una configurazione vincente per X viene stimata $+\infty$, una vincente per O $-\infty$.

La funzione di valutazione

La funzione di valutazione $Eval$ è una **stima della utilità attesa** a partire da una certa posizione. La qualità della *funzione* è determinante:

- deve essere consistente con l'utilità se applicata a stati terminali del gioco (indurre lo stesso ordinamento);
- deve essere efficiente da calcolare;
- deve riflettere le probabilità effettive di vittoria (A valutato meglio di B se da A ci sono più probabilità di vittoria che da B);
- valore “atteso” che combina probabilità con utilità dello stato terminale, può essere appreso con l'esperienza.

Per gli scacchi si potrebbe pensare di valutare caratteristiche diverse dello stato:

- Valore del materiale (pedone 1, cavallo o alfiere 3, torre 5, regina 9, ...)
- Buona disposizione dei pedoni, protezione del re

Una funzione lineare pesata potrebbe essere

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

f_i : caratteristica (es. numero dei pezzi)

w_i : peso relativo (es. valore dei pezzi)

È possibile avere anche combinazioni non lineari di caratteristiche: alfiere vale più nei finali di partita, 2 alfieri valgono più del doppio di 1.

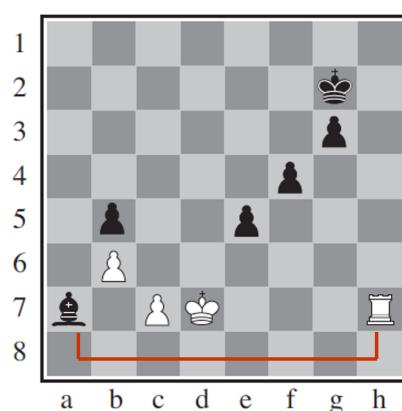
Problemi con MIN-MAX: Stati non quiescenti

Stati non quiescenti: l'esplorazione fino ad un livello può mostrare una situazione molto vantaggiosa, poi alla mossa successiva la regina nera viene catturata.

Soluzione: applicare la valutazione a stati *quiescenti*, stati in cui la funzione di valutazione non è soggetta a mutamenti repentini (ricerca di quiete).

Problemi con MIN-MAX: Effetto orizzonte

Effetto orizzonte: può succedere che vengano privilegiate mosse divisorie che hanno il solo effetto di spingere il problema oltre l'orizzonte.



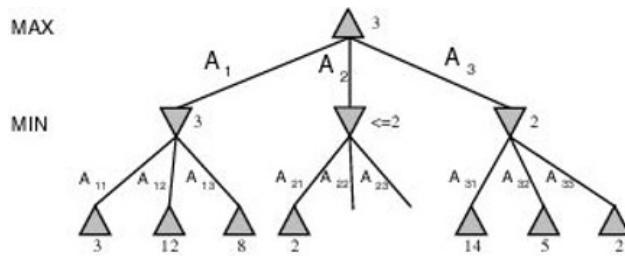
L'alfiere in A7, catturabile in 3 mosse dalla torre, è spacciato. Mettere il re bianco sotto scacco con il pedone in E5 e poi con quello in F4... evita il problema temporaneamente, ma è un sacrificio inutile di pedoni.

Ottimizzazione

Ma dobbiamo necessariamente esplorare ogni cammino? No, esiste un modo di dimezzare la ricerca pur mantenendo la decisione minimax corretta: Potatura alfa-beta (McCarthy 1956, per scacchi).

6.1.2 Potatura alfa-beta

Tecnica di *potatura* per ridurre l'esplorazione dello spazio di ricerca in algoritmi MIN-MAX.



$$\begin{aligned}
 \text{MINMAX}(\text{radice}) &= \max(\min(3, 12, 8), \min(2, x, y), \min(14, 5, 2)) \\
 &= \max(3, \min(2, x, y), 2) \\
 &= \max(3, z, 2) = 3 \quad \text{con } z \leq 2
 \end{aligned}$$

Nota: il risultato è lo stesso ma la ricerca è più efficiente; sperimentalmente si è visto che si può raddoppiare il numero di nodi esplorati a parità di risorse spazio-temporali.

Più in generale, consideriamo un valore v trovato a una certa profondità. Se c'è una scelta migliore, α , a profondità minore, quel v non sarà mai raggiunto: MAX passerà da α piuttosto che finire a v .

Implementazione

Si va avanti in profondità fino al livello desiderato e propagando indietro i valori si decide se si può abbandonare l'esplorazione nel sotto-albero.

MaxValue e MinValue vengono invocate con due valori di riferimento: α (inizialmente $-\infty$) e β (inizialmente $+\infty$) che rappresentano rispettivamente la migliore alternativa per MAX e per MIN fino a quel momento. I tagli avvengono quando nel propagare indietro:

- $v \geq \beta$ per i nodi MAX (taglio β)
- $v \leq \alpha$ per i nodi MIN (taglio α)

L'algoritmo Alfa-Beta

```

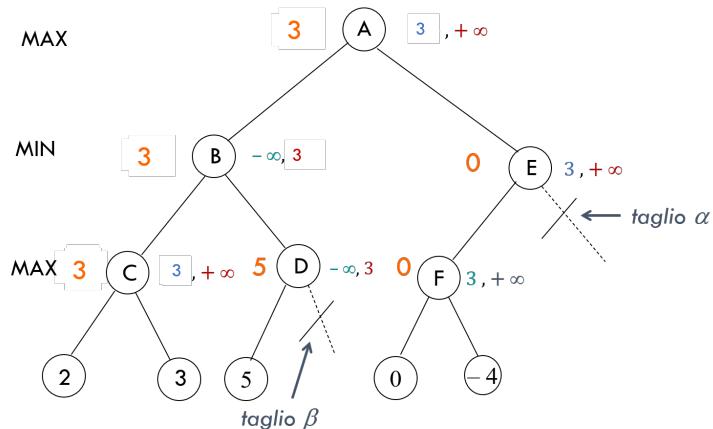
function ALPHA-BETA-SEARCH(state) returns an action
    v  $\leftarrow$  MAX-VALUE(state,  $-\infty$ ,  $+\infty$ )
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow -\infty$ 
    for each a in ACTIONS(state) do
        v  $\leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s,a), \alpha, \beta))$ 
        if v  $\geq \beta$  then return v    $\leftarrow$  taglio  $\beta$ 
         $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
    return v

function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v  $\leftarrow +\infty$ 
    for each a in ACTIONS(state) do
        v  $\leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s,a), \alpha, \beta))$ 
        if v  $\leq \alpha$  then return v    $\leftarrow$  taglio  $\alpha$ 
         $\beta \leftarrow \text{MIN}(\beta, v)$ 
    return v

```

Alfa-beta in azione



Nota bene: i nodi sotto vengono visitati (esamina alcuni discendenti per raccogliere le info) come che D vale 5 e F 0 e -4... Il taglio è cioè sotto la linea indicata! Ma è qua il senso importante: si taglano sottolaberi come quello radicato in G a destra perché α e β risalgono.

Ordinamento delle mosse

La potatura ottimale si ottiene quando ad ogni livello sono generate prima le mosse migliori per chi gioca: per nodi MAX sono generate prima le mosse con valore più alto, mentre per i nodi MIN sono generate prima le mosse con valore più basso (migliori per MIN).

Complessità: $O(b^{m/2})$ anziché $O(b^m)$.

Alfa-Beta può arrivare a profondità doppia rispetto a Min-Max! Ma come avvicinarsi all'ordinamento ottimale? Ordinamento dinamico:

- Usando approfondimento iterativo si possono scoprire ad una iterazione informazioni utili per l'ordinamento delle mosse, da usare in una successiva iterazione (mosse killer).
- Tabella delle trasposizioni: per ogni stato incontrato si memorizza la sua valutazione. Situazione tipica: $[a_1, b_1, a_2, b_2]$ e $[a_1, b_2, a_2, b_1]$ portano nello stesso stato.

Altri miglioramenti

Potatura in avanti: esplorare solo alcune mosse ritenute promettenti e tagliare le altre.

- Beam search;
- Tagli probabilistici (basati su esperienza). Miglioramenti notevoli in Logistello.

Database di mosse di apertura e chiusura:

- Nelle prime fasi ci sono poche mosse sensate e ben studiate, inutile esplorarle tutte.
- Per le fasi finali il computer può esplorare off-line in maniera esaustiva e ricordarsi le migliori chiusure (già esplorate tutte le chiusure con 5 e 6 pezzi...).

6.2 Problemi di soddisfacimento di vincoli

Sono problemi con una struttura particolare, che si prestano ad algoritmi di ricerca specializzati. È un esempio di rappresentazione **fattorizzata**, in cui si comincia a dire qualcosa sulla struttura dello stato. Esistono euristiche generali che si applicano e che consentono la risoluzione di problemi di dimensioni significative per questa classe. La classe di problemi formulabili in questo modo è piuttosto ampia: layout di circuiti, scheduling, ...

Formulazione di problemi CSP

Problema: descritto da tre componenti

1. X un insieme di variabili.
2. D un insieme di domini, dove D_i è l'insieme dei valori possibili per X_i .
3. C un insieme di vincoli (relazioni tra le variabili).

Stato: un assegnamento [**parziale|completo**] di valori a variabili

$$\{X_i = v_i, X_j = v_j, \dots\}$$

Stato iniziale: { }

Azioni: assegnamento di un valore a una variabile (tra quelli leciti)

Soluzione (goal test): un assegnamento **completo** (le variabili hanno tutte un valore) e **consistente** (i vincoli sono tutti soddisfatti).

6.2.1 Strategie per problemi CSP

Finora potevamo solo ricercare la soluzione nel grafo degli stati (guidati da una metrica definita sullo stato). Adesso possiamo

- usare delle euristiche specifiche per questa classe di problemi;
- fare delle inferenze che ci portano a restringere i domini e quindi a limitare la ricerca: **propagazione di vincoli**;
- fare backtracking **intelligente**.

Tipicamente un mixto di queste strategie.

6.2.2 Ricerca in problemi CSP

Ad ogni passo si assegna una variabile. La massima profondità della ricerca è fissata dal numero di variabili n . Versione “ingenua”:

- l'ampiezza dello spazio di ricerca è $|D_1| \times |D_2| \times \dots \times |D_n|$ dove $|D_i|$ è la cardinalità del dominio di X_i ,
- il fattore di diramazione è pari a nd al primo passo; $(n - 1)d$ al secondo... le foglie sarebbero $n! \cdot d^n$.

Riduzione drastica dello spazio di ricerca dovuta al fatto che il *goal-test* è commutativo, non è importante l'ordine con cui scelgo le variabili quindi ad ogni livello posso scegliere una e una sola variabile per un fattore di diramazione pari a d . In realtà il fattore di diramazione è pari alla dimensione dei domini d (d^n foglie).

Strategie di ricerca

Ricerca con *backtracking* (BT) a profondità limitata. **Controllo anticipato** della violazione dei vincoli: è inutile andare avanti fino alla fine e poi controllare; si può fare *backtracking* non appena si scopre che un vincolo è stato violato.

La ricerca è limitata naturalmente in profondità dal numero di variabili quindi il metodo è completo.

Backtracking ricorsivo per CSP

```

function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment                      trovata soluzione
  var  $\leftarrow$  SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then                         controllo anticipato
      add {var = value} to assignment
      inferences  $\leftarrow$  INFERENCE(csp, var, value)                           riduce i domini
      if inferences  $\neq$  failure then                                         nessun dominio è vuoto
        add inferences to assignment
        result  $\leftarrow$  BACKTRACK(assignment, csp)
        if result  $\neq$  failure then
          return result
        remove {var = value} and inferences from assignment           si disfa lo stato
      return failure

```

SelectUnassignedVariable: Quale variabile scegliere?

OrderDomainValues: Quali valori scegliere?

Inference: Qual è l'influenza di un assegnamento sulle altre variabili? Come restringe i domini? *Propagazione di vincoli, riduzione problema.*

BackTrack: Come evitare di ripetere i fallimenti? *Backtracking intelligente.*

Scelta delle variabili

1. MRV (*Minimum Remaining Values*): scegliere la variabile che ha meno valori legali [residui], la variabile **più vincolata**. Si scoprano prima i fallimenti (*fail first*).
2. Euristica *del grado*: scegliere la variabile coinvolta in più vincoli con le altre variabili (*la variabile più vincolante o di grado maggiore*). Da usare a parità di MRV.

Scelta dei valori

Una volta scelta la variabile come scegliere il valore da assegnare?

Valore ***meno vincolante***: quello che esclude meno valori per le altre variabili direttamente collegate con la variabile scelta, meglio valutare prima un assegnamento che ha più probabilità di successo. Se volessimo tutte le soluzioni l'ordine non sarebbe importante.

Propagazione di vincoli

1. Verifica in avanti (*Forward Checking* o *FC*): assegnato un valore ad una variabile si possono eliminare i valori incompatibili per le altre variabili ***direttamente collegate*** da vincoli (non si itera).
2. Consistenza di nodo e d'arco: si restringono il valori dei domini delle variabili tenendo conto dei vincoli unari e binari su tutto il grafo (si itera finché tutti i nodi ed archi sono consistenti).

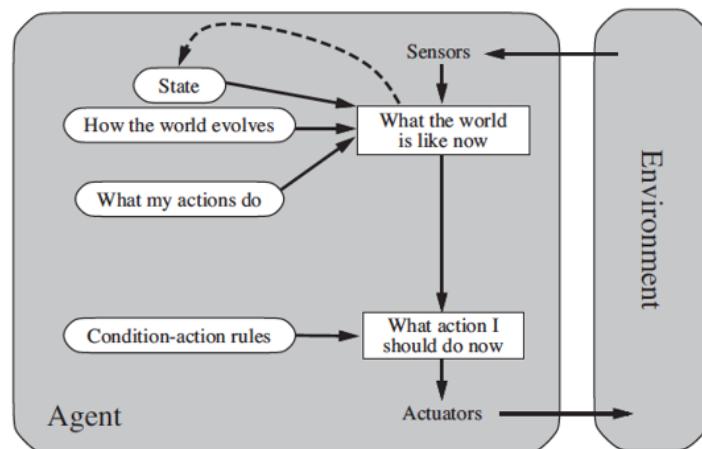
Capitolo 7

Agenti basati su conoscenza

Abbiamo trattato:

- Agenti con stato e con obiettivo in mondi osservabili con stati atomici e azioni descrivibili in maniera semplice; enfasi sul processo di ricerca.
- Descrizioni “fattorizzate” (come nei CSP) che consentono di iniziare a “guardare dentro” lo stato, descritto come un insieme di caratteristiche rilevanti (o *feature*).

Vogliamo adesso migliorare le **capacità di ragionamento** dei nostri agenti dotandoli di rappresentazioni di mondi più complessi e **astratti**, non descrivibili semplicemente: agenti **basati su conoscenza**, dotati di una KB (*Knowledge Base*) con conoscenza espressa in maniera esplicita e dichiarativa.



Agenti basati su modello

Agenti “Knowledge Based”

La maggior parte dei problemi di I.A. sono “*knowledge intensive*”. Il mondo è tipicamente complesso: ci serve una rappresentazione **parziale** e **incompleta** (un’astrazione) del mondo utile agli scopi dell’agente.

Per ambienti parzialmente osservabili e complessi ci servono linguaggi di rappresentazione della conoscenza più espressivi e capacità inferenziali. La conoscenza può essere codificata a mano ma anche estratta dai testi o appresa dall’esperienza.

Approccio dichiarativo vs procedurale

La KB racchiude tutta la conoscenza necessaria a decidere l’azione da compiere in forma **dichiarativa**. L’alternativa (*approccio procedurale*) è scrivere un programma che implementa il processo decisionale, una volta per tutte.

Un agente KB è più flessibile: più semplice acquisire conoscenza incrementalmente e modificare il comportamento con l’esperienza.

7.1 Agente basato su conoscenza

Un agente basato su conoscenza mantiene una **base di conoscenza** (KB): un insieme di enunciati espressi in un linguaggio di rappresentazione. Interagisce con la KB mediante una interfaccia funzionale *Tell-Ask*:

- *Tell*: per aggiungere nuovi enunciati a KB
- *Ask*: per interrogare la KB
- *Retract*: per eliminare enunciati

Gli enunciati nella KB rappresentano le **opinioni/credenze dell’agente**. Le risposte α devono essere tali che α è una conseguenza (discende necessariamente) della KB.

Il problema: data una base di conoscenza KB, contenente una rappresentazione dei fatti che si **ritengono veri**, vorrei sapere se un certo fatto α è vero di conseguenza

$$\text{KB} \models \alpha \quad (\text{conseguenza logica})$$

Base di conoscenza vs base di dati

Base di conoscenza: una rappresentazione esplicita, parziale e compatta, in un linguaggio simbolico, che contiene fatti di tipo specifico e fatti di tipo generale, o regole.

Base di dati: solo fatti specifici, solo recupero.

Quello che caratterizza una KB è la **capacità inferenziale**, cioè la capacità di derivare nuovi fatti da quelli memorizzati esplicitamente.

Il trade-off fondamentale della R.C.

Sfortunatamente più il linguaggio è *espressivo*, meno *efficiente* è il meccanismo inferenziale. Il problema “fondamentale” nella rappresentazione della conoscenza (R.C.) è trovare il giusto compromesso tra: espressività del linguaggio di rappresentazione e complessità del meccanismo inferenziale. Questi due obiettivi sono in contrasto e si tratta di mediare tra queste due esigenze.

7.1.1 Formalismi per la R.C.

Un formalismo per la rappresentazione della conoscenza ha tre componenti:

1. Una **sintassi**: un linguaggio composto da un vocabolario e regole per la formazione delle frasi (*enunciati*).
2. Una **semantica** che stabilisce una corrispondenza tra gli enunciati e fatti del mondo; se un agente ha un enunciato α nella sua KB, crede che il fatto corrispondente sia vero nel mondo.
3. Un **meccanismo inferenziale** (codificato, o meno, tramite regole di inferenza come nella logica) che ci consente di inferire nuovi fatti.

Logica come linguaggio per la R.C.

Qual è la complessità computazionale del problema $\text{KB} \models \alpha$ nei vari linguaggi logici? Quali sono gli algoritmi di decisione e le strategie di ottimizzazione? I linguaggi logici come calcolo proposizionale (PROP) e logica dei predicati (FOL) sono adatti per la rappresentazione della conoscenza?

7.2 Agenti logici: calcolo proposizionale

Sintassi

La sintassi definisce quali sono le frasi legittime (**ben formate**) del linguaggio:

<i>formula</i>	\rightarrow	<i>formulaAtomica</i> <i>formulaComplessa</i>
<i>formulaAtomica</i>	\rightarrow	True False <i>simbolo</i>
<i>simbolo</i>	\rightarrow	P Q R ...
<i>formulaComplessa</i>	\rightarrow	\neg <i>formula</i> (<i>formula</i> \wedge <i>formula</i>) (<i>formula</i> \vee <i>formula</i>) (<i>formula</i> \Rightarrow <i>formula</i>) (<i>formula</i> \Leftrightarrow <i>formula</i>)

Esempio: $((A \wedge B) \Rightarrow C)$

Possiamo omettere le parentesi assumendo questa **precedenza** tra gli operatori:

$\neg > \wedge > \vee > \Rightarrow > \Leftrightarrow$

Semantica e mondi possibili (modelli)

La semantica ha a che fare col significato delle frasi: definisce se un enunciato è vero o falso rispetto ad una **interpretazione** (mondo possibile). Un'interpretazione definisce un valore di verità per tutti i simboli proposizionali.

Per esempio: $\{P_{1,1} \text{ vero}, P_{1,2} \text{ falso}, W_{2,3} \text{ vero}\}$

Il valore di una formula complessa è fissato di conseguenza:

$P_{1,1} \Rightarrow W_{2,3} \vee P_{1,2}$ è vera in questa interpretazione.

Un **modello** è un'interpretazione che *rende vera* una formula o un insieme di formule.

Semantica compositiva

Il significato di una frase è determinato dal significato dei suoi componenti, a partire dalle frasi atomiche (i simboli proposizionali).

- *True* sempre vero; *False* sempre falso
- $P \wedge Q$, vero se P e Q sono veri
- $P \vee Q$, vero se P oppure Q , o entrambi, sono veri
- $\neg P$ vero se P è falso
- $P \Rightarrow Q$, vero se P è falso oppure Q è vero
- $P \Leftrightarrow Q$, vero se entrambi veri o entrambi falsi

Conseguenza logica

Una formula α è **conseguenza logica** di un insieme di formule KB se e solo se in ogni modello di KB, anche α è vera ($\text{KB} \models \alpha$). Indicando con $M(\alpha)$ l'insieme delle interpretazioni che rendono α vera (i **modelli** di α) e con $M(\text{KB})$ i modelli dell'insieme di formule in KB...

$$\text{KB} \models \alpha \text{ sse } M(\text{KB}) \subseteq M(\alpha)$$

Equivalenza logica: leggi

Equivalenza logica: $A \equiv B$ se e solo se $A \models B$ e $B \models A$.

$$\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \quad \text{commutatività di } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) \quad \text{commutatività di } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associatività di } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associatività di } \vee \\
\neg(\neg\alpha) &\equiv \alpha \quad \text{doppia negazione} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contrapposizione} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) \quad \text{eliminazione dell'implicazione} \\
(\alpha \Leftrightarrow \beta) &\equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha) \quad \text{eliminazione del sse} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) \quad \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) \quad \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributività di } \wedge \text{ su } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributività di } \vee \text{ su } \wedge
\end{aligned}$$

Validità, soddisfabilità

A è **valida** sse è vera in tutte le interpretazioni (anche detta tautologia).
A è **soddisfacibile** sse esiste un'interpretazione in cui A è vera.

Ne discende che

$$A \text{ è valida sse } \neg A \text{ è insoddisfacibile}$$

Inferenza per calcolo proposizionale

- *Model checking*: una forma di inferenza che fa riferimento alla definizione di conseguenza logica (si enumerano i possibili modelli), per esempio usando la tecnica delle tabelle di verità.
- Algoritmi per la *soddisfabilità* (SAT): $\text{KB} \models A$ sse $(\text{KB} \wedge \neg A)$ è insoddisfacibile. Un problema può essere ricondotto all'altro.

7.2.1 L'algoritmo TT-entails

$$\text{KB} \models \alpha?$$

Enumera tutte le possibili interpretazioni di KB (k simboli, 2^k possibili interpretazioni). Per ciascuna interpretazione

- Se non soddisfa KB, OK.
- Se soddisfa KB, si controlla che soddisfi anche α .

Se si trova anche solo un'interpretazione che soddisfa KB e non α la risposta sarà NO.

```

function TT-ENTAILS?(KB,  $\alpha$ ) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
             $\alpha$ , the query, a sentence in propositional logic
  symbols  $\leftarrow$  a list of the proposition symbols in KB and  $\alpha$ 
  return TT-CHECK-ALL(KB,  $\alpha$ , symbols, { })

```

```

function TT-CHECK-ALL(KB,  $\alpha$ , symbols, model) returns true or false
  if EMPTY?(symbols) then
    if PL-TRUE?(KB, model) then return PL-TRUE?( $\alpha$ , model)
    else return true // when KB is false, always return true
  else do
    P  $\leftarrow$  FIRST(symbols)
    rest  $\leftarrow$  REST(symbols)
    return (TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = true})
           and
           TT-CHECK-ALL(KB,  $\alpha$ , rest, model  $\cup$  {P = false }))

```

7.2.2 Algoritmi per la soddisfacibilità (SAT)

Usano KB in **forma a clausole** (insiemi di letterali)

$$\{A, B\} \{\neg B, C, D\} \{\neg A, F\}$$

La forma a clausole è la forma normale congiuntiva (CNF): una congiunzione di disgiunzioni di letterali

$$(A \vee B) \wedge (\neg B \vee C \vee D) \wedge (\neg A \vee F)$$

Non è restrittiva: è sempre possibile ottenerla con trasformazioni che preservano l'equivalenza logica.

Trasformazione in forma a clausole

I passi sono:

1. Eliminazione del \Leftrightarrow : $(A \Leftrightarrow B) \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$
2. Eliminazione dell' \Rightarrow : $(A \Rightarrow B) \equiv (\neg A \vee B)$
3. Negazioni all'interno:

$$\begin{aligned}\neg(A \vee B) &\equiv (\neg A \wedge \neg B) \quad (\text{De Morgan}) \\ \neg(A \wedge B) &\equiv (\neg A \vee \neg B)\end{aligned}$$

4. Distribuzione di \vee su \wedge : $(A \vee (B \wedge C)) \equiv (A \vee B) \wedge (A \vee C)$

7.2.3 L'algoritmo DPLL per la soddisfabilità

DPLL: Davis, Putman, e poi Lovemann, Loveland

Parte da una KB in **forma a clausole**. È un'enumerazione *in profondità* di tutte le possibili interpretazioni alla ricerca di un modello. Tre miglioramenti rispetto a TTEntails:

1. terminazione anticipata,
2. euristica dei simboli (o letterali) puri,
3. euristica delle clausole unitarie.

Terminazione anticipata

Si può decidere sulla verità di una clausola anche con interpretazioni parziali: basta che un *letterale* sia vero. Per esempio se A è vero, lo sono anche $\{A\}$, $\{B\}$ e $\{A, C\}$ indipendentemente dai valori di B e C .

Se anche una sola clausola è falsa l'interpretazione non può essere un modello dell'insieme di clausole.

Simboli puri

Simbolo puro: un simbolo che appare con lo stesso segno in tutte le clausole, per esempio

$$\{A, \neg B\} \ \{\neg B, \neg C\} \ \{C, A\} \quad A \text{ puro, } B \text{ anche}$$

Nel determinare se un simbolo è puro se ne possono trascurare le occorrenze in clausole già rese vere.

I simboli puri possono essere assegnati a *True* se il letterale è positivo, *False* se negativo. Non si eliminano modelli utili: se le clausole hanno un modello continuano ad averlo dopo questo assegnamento. L'assegnamento è obbligato.

Clausole unitarie

Clausola unitaria: una clausola con un solo letterale **non assegnato**. Per esempio $\{B\}$ è unitaria ma anche $\{B, \neg C\}$ è unitaria quando $C = \text{True}$.

Conviene assegnare prima valori ai letterali in clausole unitarie. L'assegnamento è univoco (*True* se positivo, *False* se negativo).

```

function DPLL-SATISFIABLE?(s) returns true or false
  inputs: s, a sentence in propositional logic
  clauses  $\leftarrow$  the set of clauses in the CNF representation of s
  symbols  $\leftarrow$  a list of the proposition symbols in s
  return DPLL(clauses, symbols, { })

```

```

function DPLL(clauses, symbols, model) returns true or false
  if every clause in clauses is true in model then return true
  if some clause in clauses is false in model then return false
  P, value  $\leftarrow$  FIND-PURE-SYMBOL(symbols, clauses, model)
  if P is non-null then return DPLL(clauses, symbols - P, model  $\cup$  {P=valueP, value  $\leftarrow$  FIND-UNIT-CLAUSE(clauses, model)
  if P is non-null then return DPLL(clauses, symbols - P, model  $\cup$  {P=valueP  $\leftarrow$  FIRST(symbols); rest  $\leftarrow$  REST(symbols)
  return DPLL(clauses, rest, model  $\cup$  {P=true}) or
         DPLL(clauses, rest, model  $\cup$  {P=false}))

```

Miglioramenti di DPLL

DPLL è completo e termina sempre. Alcuni miglioramenti:

- Analisi di componenti (sotto-problemi indipendenti): se le variabili possono essere suddivise in sotto-insiemi disgiunti (senza simboli in comune).
- Ordinamento di variabili e valori: scegliere la variabile che compare in più clausole.
- Backtracking intelligente e altre ottimizzazioni...

7.2.4 Metodi locali per SAT

Gli stati sono assegnamenti completi, l'obiettivo è un assegnamento che soddisfi tutte le clausole (un modello). Si parte da un assegnamento casuale e ad ogni passo si cambia il valore di un simbolo proposizionale (*flip*). Gli stati sono valutati contando il numero di clausole **non soddisfatte** (meno sono meglio è) [o soddisfatte].

Ci sono molti minimi locali per sfuggire ai quali serve introdurre perturbazioni casuali

- Hill climbing con riavvio casuale
- Simulated Annealing

Molta sperimentazione per trovare il miglior compromesso tra il grado di “avidità” e casualità. WALK-SAT è uno degli algoritmi più semplici ed efficaci.

WalkSAT

WalkSAT ad ogni passo sceglie a caso una clausola non ancora soddisfatta, sceglie un simbolo da modificare (*flip*) con probabilità p (di solito 0,5) tra una delle due:

- sceglie un simbolo a caso (**passo casuale**),
- sceglie quello che rende più clausole soddisfatte (**passo di ottimizzazione**).

Il passo casuale corrisponde a un **random walk** locale, quello di ottimizzazione a un tentativo di andare in salita.

Si arrende dopo un certo numero di *flip* predefinito.

```
function WALKSAT(clauses, p, max_flips) returns a satisfying model or failure
  inputs: clauses, a set of clauses in propositional logic
          p, the probability of choosing to do a “random walk” move, typically around 0.5
          max_flips, number of flips allowed before giving up

  model ← a random assignment of true/false to the symbols in clauses
  for i = 1 to max_flips do
    if model satisfies clauses then return model
    clause ← a randomly selected clause from clauses that is false in model
    with probability p flip the value in model of a randomly selected symbol from clause
    else flip whichever symbol in clause maximizes the number of satisfied clauses
  return failure
```

Analisi di WalkSAT

Se $\text{max-flips} = \infty$ e l'insieme di clausole è soddisfacibile, prima o poi termina. Va bene per cercare un modello, sapendo che c'è, ma se è insoddisfacibile non termina quindi non può essere usato per verificare l'insoddisfacibilità.

Il problema è decidibile ma l'algoritmo non è completo.

Problemi SAT difficili

Se un problema ha molte soluzioni (problema sotto-vincolato) è più probabile che WalkSAT ne trovi una in tempi brevi. Per esempio se ho 16 soluzioni su 32; un assegnamento ha il 50% di probabilità di essere giusto: 2 passi in media!

Esempio: Istanza di 3SAT

$$(\neg D \vee \neg B \vee C) \wedge (B \vee \neg A \vee \neg C) \wedge (\neg C \vee \neg B \vee E) \wedge (E \vee \neg D \vee B) \wedge (B \vee E \vee \neg C)$$

Quello che conta è il rapporto m/n , dove m è il numero di clausole (vincoli) e n il numero di simboli. In questo caso, $5/5 = 1$. Più grande il rapporto, più vincolato è il problema.

Le regine sono facili perché il problema è sotto-vincolato.

7.2.5 Inferenza come deduzione

Un altro modo per decidere se $\text{KB} \models A$ è usare un **sistema di deduzione**; si scrive $\text{KB} \vdash A$ (A è deducibile da KB). La deduzione avviene specificando delle **regole di inferenza**.

In un sistema di inferenza le regole

- dovrebbero derivare *solo* formule che sono conseguenza logica,
- dovrebbero derivare *tutte* le formule che sono conseguenza logica.

Correttezza e completezza

Correttezza: Se $\text{KB} \vdash A$ allora $\text{KB} \models A$

Tutto ciò che è derivabile è conseguenza logica. Le regole preservano la verità.

Completezza: Se $\text{KB} \models A$ allora $\text{KB} \vdash A$

Tutto ciò che è conseguenza logica è ottenibile tramite il sistema deduttivo.

Dimostrazione come ricerca

Problema: come decidere ad ogni passo qual è la regola di inferenza da applicare? ... e a quali premesse? Come evitare l'esplosione combinatoria?

Si tratta di un problema di esplorazione di uno spazio di stati. Una *procedura di dimostrazione* definisce:

- la direzione della ricerca,
- la strategia di ricerca.

Direzione della ricerca

Nella dimostrazione di teoremi conviene procedere all'indietro. Con una applicazione *in avanti* delle regole di inferenza non controllata: da A, B posso derivare $A \wedge B$, $A \wedge (A \wedge B)$, ..., $A \wedge (A \wedge (A \wedge B))$.

Meglio *all'indietro*:

- se si vuole dimostrare $A \wedge B$, si cerchi di dimostrare A e poi B
- se si vuole dimostrare $A \Rightarrow B$, si assuma A e si cerchi di dimostrare B

Strategia di ricerca

Completezza

- Le regole della deduzione naturale sono un insieme di regole di inferenza completo (2 per ogni connettivo)
- Se l'algoritmo di ricerca è completo siamo a posto

Efficienza → la complessità è alta: è un problema decidibile ma NP-completo.

Regola di risoluzione per PROP

Meno regole ci sono e meglio è, senza rinunciare alla completezza. Un'unica regola: la regola di risoluzione (presuppone forma a clausole).

$$\frac{\{P, Q\} \quad \{\neg, R\}}{\{Q, R\}} \qquad \frac{P \vee Q \quad \neg P \vee R}{Q \vee R}$$

È corretta? Basta pensare ai modelli. Il motivo per cui viene preferita la notazione insiemistica è che gli eventuali duplicati si eliminano.

La regola di risoluzione generale

$$\frac{\{l_1, l_2, \dots, l_i, \dots, l_k\} \quad \{m_1, m_2, \dots, m_j, \dots, m_n\}}{\{l_1, l_2, \dots, l_{i-1}, l_{i+1}, \dots, l_k, m_1, m_2, \dots, m_{j-1}, m_{j+1}, \dots, m_n\}}$$

Gli l e m sono **letterali**, simboli di proposizione positivi o negativi; l_i e m_j sono uguali e di segno opposto.

Caso particolare: clausola vuota \rightarrow *contraddizione*.

$$\frac{\{P\} \quad \{\neg P\}}{\{ \}}$$

La regola è sufficiente? È sicuro che applicando la regola in tutti i modi possibili si riesca a dedurre A quando è conseguenza logica?

Completezza: se $\text{KB} \models \alpha$ allora $\text{KB} \vdash_{\text{res}} \alpha$? Non sempre, per esempio $\text{KB} \models \{A, \neg A\}$, ma non è vero che $\text{KB} \vdash_{\text{res}} \{A, \neg A\}$.

Nella versione proposizionale è di aiuto il teorema di risoluzione [ground]:

$$\text{KB insoddisfacibile } sse \text{ KB} \vdash_{\text{res}} \{ \}$$

in qualche modo si tratta di una garanzia di *completezza*. Il teorema di refutazione offre un modo alternativo:

$$\text{KB} \models \alpha \text{ sse } (\text{KB} \cup \{\neg \alpha\}) \text{ è insoddisfacibile}$$

Nell'esempio: $\text{KB} \cup \text{FC}(\neg(A \vee \neg A))$ è insoddisfacibile? Sì, perché... $\text{KB} \cup \{A\} \cup \{\neg A\} \vdash_{\text{res}} \{ \}$ in un passo. Quindi $\text{KB} \models \{A, \neg A\}$.

Conclusioni

- Abbiamo visto come gli agenti KB che usano PROP come linguaggio di rappresentazione possono decidere se $\text{KB} \models \alpha$.
- Il problema è decidibile, ma intrattabile (NP) nel caso peggiore.
- Esistono algoritmi efficienti e completi che consentono di affrontare problemi di grosse dimensioni.
- I metodi locali sono particolarmente efficienti ma non completi.

Capitolo 8

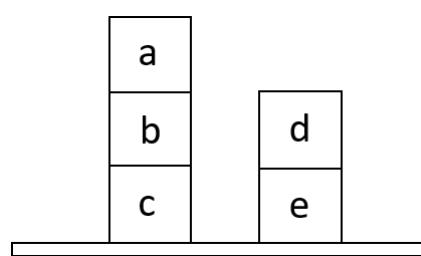
Agenti logici: la logica del prim'ordine

Nella logica dei predicati abbiamo assunzioni ontologiche più ricche: gli *oggetti*, le *proprietà* e le *relazioni*. Si inizia con una **concettualizzazione**: si tratta di decidere quali sono le cose di cui si vuole parlare.

- Gli *oggetti*: un libro, un evento, una persona, un istante di tempo, un insieme, una funzione, un unicorno... Gli oggetti possono essere identificati con simboli o relativamente ad altri oggetti, mediante *funzioni* (“*la madre di Pietro*”); l’insieme degli oggetti rilevanti costituiscono il **dominio del discorso**.
- Le proprietà: “*la madre di Pietro è simpatica*”.
- Le relazioni tra gli oggetti: “*Pietro è amico di Paolo*”.

Esempio: il mondo dei blocchi

Ci interessano i blocchi e alcune loro relazioni spaziali.



Dominio: {a, b, c, d, e} ← blocchi veri!

Le **funzioni**: si individuano le funzioni rilevanti che servono anch'esse per identificare oggetti. Per esempio *Hat* è la funzione unaria che dato un blocco identifica il blocco che ci sta sopra: $\text{Hat}(b) = a$.

Le **relazioni**: si individuano le relazioni interessanti. Per esempio:

- $On = \{\langle a, b \rangle, \langle b, c \rangle, \langle d, e \rangle\}$
- $Clear = \{a, d\}$
- $Table = \{c, e\}$
- $Block = \{a, b, c, d, e\}$

$$\langle \{a, b, c, d, e\}, \{\text{Hat}\}, \{On, Clear, Table, Block\} \rangle$$

Le concettualizzazioni possibili sono infinite: un aspetto importante è il livello di astrazione *giusto* per gli scopi della rappresentazione.

8.1 Linguaggio

Vocabolario

- *Connettivo* $\rightarrow \wedge \mid \vee \mid \neg \mid \Rightarrow \mid \Leftarrow \mid \Leftrightarrow$
- *Quantificatore* $\rightarrow \forall \mid \exists$
- *Variabile* $\rightarrow x \mid y \mid \dots \mid$ (lettere minuscole)
- *Costante* $\rightarrow A \mid B \mid \dots \mid$ (lettere maiuscole)
- *Funzione* $\rightarrow \text{Hat} \mid \text{Padre-di} \mid + \mid - \mid \dots$ (con arità ≥ 1)
- *Predicato* $\rightarrow On \mid Clear \mid \geq \mid \leq \mid \dots$ (con arità ≥ 0)

Nota: l'ordine dei quantificatori è importante

$$\begin{aligned} \forall x (\exists y \text{Ama}(x, y)) & \quad \text{Tutti amano qualcuno} \\ \exists x (\forall y \text{Ama}(x, y)) & \quad \text{Esiste qualcuno amato da tutti} \end{aligned}$$

I termini

La sintassi dei termini:

$$\begin{aligned} \text{Termine} \rightarrow & \quad \text{Costante} \mid \text{Variabile} \mid \text{Funzione}(\text{Termine}, \dots) \\ & \quad (\text{un numero di termini pari alla arit\`a della funzione}) \end{aligned}$$

Le formule

La sintassi delle formule:

$$\begin{aligned}
 \text{Formula-atomica} &\rightarrow \text{True} \mid \text{False} \mid \\
 &\quad \text{Termine} = \text{Termine} \mid \\
 &\quad \text{Predicato}(\text{Termine}, \dots) \\
 &\quad (\text{un numero di termini pari alla arità del predicato}) \\
 \text{Formula} &\rightarrow \text{Formula-atomica} \mid \\
 &\quad \text{Formula Connettivo Formula} \mid \\
 &\quad \text{Quantificatore Variabile Formula} \mid \\
 &\quad \neg \text{Formula} \mid (\text{Formula})
 \end{aligned}$$

Di solito le variabili sono usate nell'ambito di quantificatori. In tal caso le occorrenze si dicono **legate**. Se non sono legate allora sono **libere**.

$$\begin{aligned}
 Mela(x) \Rightarrow Rossa(x) &\quad x \text{ è libera in entrambe le occorrenze} \\
 \forall x \ Mela(x) \Rightarrow Rossa(x) &\quad x \text{ è legata} \\
 Mela(x) \Rightarrow \exists x \ Rossa(x) &\quad \text{la prima è libera, la seconda legata}
 \end{aligned}$$

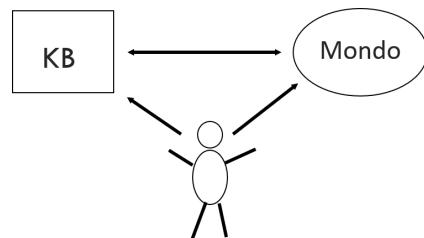
Definizione 8.1.1. Una formula si dice **chiusa** se non contiene occorrenze di variabili libere. Altrimenti è detta **aperta**.

Definizione 8.1.2. Una formula si dice **ground** se non contiene variabili.

Precedenza tra gli operatori logici

$$= > \neg > \wedge > \vee > \Rightarrow, \Leftrightarrow > \forall, \exists$$

8.1.1 Semantica dichiarativa



Consiste nello stabilire una corrispondenza tra:

- i termini del linguaggio e gli oggetti del mondo,
- le formule chiuse e i valori di verità.

Interpretazione

Una interpretazione \mathcal{I} stabilisce una corrispondenza precisa tra elementi atomici del linguaggio ed elementi della concettualizzazione. \mathcal{I} interpreta:

- i simboli di costante come elementi del dominio;
- i simboli di funzione come funzioni da *n-uple* di D in D;
- i simboli di predicato come insiemi di *n-uple*.

8.1.2 Semantica compositiva

Il significato di un termine o di una formula composta è determinato in funzione del significato dei suoi componenti:

- La formula $A \wedge B$ è vera in una certa interpretazione se entrambe A e B sono vere
- $\neg A$ è vera se A è falsa
- $A \vee B$ è vera se A è vera oppure B è vera (o entrambe)
- $A \Rightarrow B$ è vera se A è falsa oppure B è vera (come $\neg A \Rightarrow B$)

Semantica (\forall)

$\forall x A(x)$ è vera se per ciascun elemento del dominio A è vera. Se il dominio è finito equivale a un grosso \wedge .

Tipicamente \forall si usa quasi sempre insieme a \Rightarrow : difficilmente una proprietà è universale, le condizioni nell'antecedente restringono la portata dell'asserzione e la qualificano.

Semantica (\exists)

$\exists x A(x)$ è vera se esiste almeno un elemento del dominio per cui A è vera. Se il dominio è finito equivale a un grosso \vee .

Tipicamente \exists si usa con \wedge .

Relazione tra \forall e \exists

Da qui discendono delle proprietà che mettono in relazione \forall e \exists .

$$\begin{aligned}\forall x \neg P(x) &\equiv \neg \exists x P(x) & \neg P \wedge \neg Q &\equiv \neg(P \vee Q) \\ \neg \forall x P(x) &\equiv \exists x \neg P(x) & \neg(P \wedge Q) &\equiv \neg P \vee \neg Q \\ \forall x P(x) &\equiv \neg \exists x \neg P(x) & P \wedge Q &\equiv \neg(\neg P \vee \neg Q) \\ \neg \forall x \neg P(x) &\equiv \exists x P(x) & P \vee Q &\equiv \neg(\neg P \wedge \neg Q)\end{aligned}$$

8.1.3 Semantica “standard” e semantica “database”

Riccardo ha solo due fratelli: Giovanni e Goffredo. In logica classica:

$$\begin{aligned}&\text{Fratello(Riccardo, Giovanni)} \wedge \text{Fratello(Riccardo, Goffredo)} \\ &\quad \wedge \text{Giovanni} \neq \text{Goffredo} \wedge \\ &\quad \forall x \text{Fratello(Riccardo, }x\text{)} \Rightarrow (x = \text{Giovanni}) \vee (x = \text{Goffredo}).\end{aligned}$$

Semantica dei database (e di alcuni linguaggi per la RC)

- Ipotesi dei nomi unici: simboli distinti, oggetti distinti.
- Ipotesi del mondo chiuso: tutto ciò di cui non si sa che è vero è falso.
- Chiusura del dominio: esistono solo gli oggetti di cui si parla.

Interazione con la KB in FOL

Asserzioni

$\text{TELL(KB, King(John))}, \text{TELL(KB, King(George))},$
 $\text{TELL(KB, } \forall x \text{King}(x) \Rightarrow \text{Person}(x)\text{)}$

Conseguenze logiche

ASK(KB, Person(John))	Sì, se $\text{KB} \models \text{Person(John)}$
ASK(KB, $\exists x \text{Person}(x)$)	“Sì” sarebbe riduttivo: la risposta è una lista di sostituzioni o legami.

8.2 Inferenza nella logica del primo ordine

Istanziamento dell'Universale (\forall -eliminazione)

$$\frac{\forall x \ A[x]}{A[g]}$$

dove g è un termine *ground* e $A[g]$ è il risultato della sostituzione di g per x in A .

Istanziamento dell'esistenziale (\exists -eliminazione)

$$\frac{\exists x \ A[x]}{A[k]}$$

Se \exists non compare nell'ambito \forall , k è una costante nuova (*costante di Skolem*), altrimenti va introdotta una funzione (di Skolem) nelle variabili quantificate universalmente.

Riduzione a inferenza proposizionale

Proposizionalizzazione (*Grounding*)

- Creare tante istanze delle formule quantificate universalmente quanti sono gli oggetti menzionati.
- Eliminare i quantificatori esistenziali skolemizzando.

A questo punto possiamo trattare la KB come proposizionale e applicare gli algoritmi visti. Problemi? Le costanti sono in numero finito... ma se ci sono funzioni, il numero di istanze da creare è infinito.

Teorema di Herbrand

Se $\text{KB} \models A$ allora c'è una dimostrazione che coinvolge solo un sotto-insieme finito della KB proposizionalizzata. Si può procedere incrementalmente:

1. creare le istanze con le costanti,
2. creare le istanze con un solo livello di annidamento,
3. creare quelle con due livelli di annidamento,
4. ...

Se $\text{KB} \not\models A$ il processo non termina. Il problema è *semidecidibile*.

Forma a clausole

Costanti, funzioni, predici sono come definiti, ma escludiamo nel seguito formule atomiche del tipo ($t_1 = t_2$). Una clausola è un insieme di **letterali** che rappresenta la loro disgiunzione. Una KB è un insieme di clausole.

Trasformazione in forma a clausole

Teorema 8.2.1. *Per ogni formula chiusa del FOL è possibile trovare in maniera **effettiva** un insieme di clausole $FC()$ che è soddisfacibile sse α lo era [insoddisfacibile sse α lo era].*

Trasformazione:

1. Eliminazione delle implicazioni (\Rightarrow e \Leftrightarrow).
2. Negazioni all'interno.
3. Standardizzazione delle variabili: facciamo in modo che ogni quantificatore usi una variabile diversa.
4. Skolemizzazione: eliminazione dei quantificatori esistenziali.
5. Eliminazione quantificatori universali.
6. Forma normale congiuntiva (congiunzione di disgiunzioni di letterali).
7. Notazione a clausole.
8. Separazione delle variabili: clausole diverse, variabili diverse.

8.2.1 Unificazione e Sostituzione

Unificazione: operazione mediante la quale si determina se due espressioni possono essere rese **identiche** mediante una **sostituzione** di termini a variabili. Il risultato è la **sostituzione** che rende le due espressioni identiche, detta **unificatore**, o FAIL, se le espressioni non sono unificabili.

Sostituzione: un insieme finito di associazioni tra variabili e termini, in cui ogni variabile compare una sola volta sulla sinistra.

Nota: sulla sinistra solle variabili, sulla destra costanti variabili, funzioni... con la restrizione che una variabile sulla sinistra non può comparire anche sulla destra.

$$\begin{array}{ll} \{x/f(x)\} & \text{NO, sostituzione circolare.} \\ \{x/g(y), y/z\} & \text{NO, non normalizzata.} \end{array}$$

Applicazione di sostituzione

Sia σ una sostituzione, A un'espressione: $A\sigma$ istanza generata dalla sostituzione (delle variabili con le corrispondenti espressioni), in AIMA $\text{SUBST}(\sigma, A)$. **Nota:** le variabili vengono sostituite **simultaneamente** e si esegue solo un passo di sostituzione.

Espressioni unificabili

Espressioni unificabili: se esiste una sostituzione che le rende **identiche** (*unificatore*) oppure FAIL. Per esempio $P(A,y,z)$ e $P(x,B,z)$ sono unificabili con $\tau = \{x/A, y/B, z/C\}$; τ è un unificatore, ma non l'unico... un altro è $\sigma = \{x/A, y/B\}$. σ è *più generale* di τ (istanzia “meno”). Vorremmo l'*unificatore più generale* di tutti (MGU) e normalizzato.

Teorema 8.2.2. *L'unificatore più generale è unico, a parte i nomi delle variabili (l'ordine non conta).*

Algoritmo di unificazione

L'algoritmo di unificazione prende in input due espressioni p, q e restituisce un MGU Θ se esiste

- $\text{UNIFY}(p,q) = \Theta$ tale che $\text{SUBST}(\Theta,p) = \text{SUBST}(\Theta,q)$
- altrimenti FAIL

L'algoritmo esplora in parallelo le due espressioni e costruisce l'unificatore strada facendo; appena trova espressioni non unificabili fallisce. Una causa di fallimento sono le sostituzioni del tipo $x=f(x)$; questo controllo si chiama *occur check*.

```
function UNIFY(x, y, Θ) returns a substitution
    Θ, the substitution built up so far (optional, defaults to empty)
    if Θ = failure then return failure
    else if x = y then return Θ
    else if VARIABLE?(x) then return UNIFY-VAR(x, y, Θ)
    else if VARIABLE?(y) then return UNIFY-VAR(y, x, Θ)
    else if COMPOUND?(x) and COMPOUND?(y) then
        return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, Θ))
    else if LIST?(x) and LIST?(y) then
        return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST, Θ))
    else return failure

function UNIFY-VAR(var, x, Θ) returns a substitution
    if {var/val} ∈ Θ then return UNIFY(val, x, Θ)
    else if {x/val} ∈ Θ then return UNIFY(var, val, Θ)
    else if OCCUR-CHECK?(var, x) then return failure
    else return EXTEND({var/x}, Θ)
```

OCCURR-CHECK controlla se `var` occorre all'interno dell'espressione `x`, in tal caso fallisce. È un controllo di complessità quadratica. **Attenzione:** EXTEND non aggiunge semplicemente ma applica la sostituzione in Θ .

Problema dei fattori

Se un sotto insieme dei letterali di una clausola può essere unificato allora la clausola ottenuta dopo tale unificazione si dice **fattore** della clausola originaria. Il metodo di risoluzione va applicato ai *fattori* delle clausole. La deduzione per risoluzione è **corretta**

Correttezza: Se $\Gamma \vdash_{\text{RES}} A$ allora $\Gamma \models A$

La deduzione per risoluzione *non è completa*:

può essere $\Gamma \models A$ e non $\Gamma \vdash_{\text{RES}} A$

Risoluzione per refutazione

Il *teorema di refutazione* ci suggerisce un metodo alternativo **completo**.

Teorema 8.2.3 (Refutazione). $\Gamma \cup \{\neg A\}$ è *insoddisfacibile* sse $\Gamma \models A$.

Teorema 8.2.4. Γ è *insoddisfacibile* sse $\Gamma \vdash_{\text{RES}} \{\}$.

Abbiamo un metodo **meccanizzabile**, **corretto** e **completo**: basta aggiungere il negato della formula da dimostrare e provare a generare la clausola vuota.

Capitolo 9

Introduzione al Machine Learning

Apprendimento: principi universali per esseri viventi, società, macchine.

*The problem of **learning** is arguably at the very core of the problem of **intelligence**, both biological and artificial.*

- Poggio, Shelton, *AI Magazine* 1999

L'apprendimento è una sfida importante e un modo strategico per fornire *intelligenza* ai sistemi.

9.1 Cos'è il ML?

Apprendimento: un obiettivo complesso, un campo di ricerca in continua crescita. In Informatica, campo teorico e applicativo denominato Machine Learning. Il Machine Learning è emerso come un'area di ricerca che combina gli obiettivi di creare *computer in grado di apprendere* (IA) e nuovi *potenti strumenti adattivi/statistici* con basi rigorose nella scienza computazionale. Macchine che *imparano* da sole. Perché? Lusso o necessità?

- Crescita della disponibilità e della necessità di analisi di dati empirici → *Ruolo centrale/metodologico* dovuto al cambio di paradigma nella scienza: *data-driven*
- Difficile fornire capacità di adattamento/intelligenza programmando (vedi Turing) → *imparare* come unica scelta...

Gli obiettivi includono:

- Come metodologia AI → **Costruire sistemi intelligenti adattivi**, dal motore di ricerca alla robotica...
- Come apprendimento statistico → **Costruire un potente sistema predittivo per l'analisi intelligente dei dati**, strumenti per “data scientist”
- Come metodo informatico per aree di applicazione innovative → **Utilizzo dei modelli come strumento per problemi complessi (interdisciplinari)**, dall'analisi dei dati biologici alla comprensione delle immagini...

Semplici esempi concreti

Apprendimento automatizzato dal sistema dell'esperienza (serie di esempi) per affrontare un compito computazionale: casi in cui non c'è nessuna (o scarsa) conoscenza/regole precedenti per la soluzione ed è (più) facile avere una fonte di *esperienza di formazione* (dati con risultati noti).

Modelli utilizzati nei sistemi del mondo reale (pervasivi) e in una nuova area interdisciplinare, comprendenti: Pattern Recognition, Robotics, Computer Vision, Natural Language Processing, Information Retrieval, Web search engine, Complex Analyzes of Data (Med, Bio, Web), Data mining, previsioni finanziarie, sistemi e filtri adattivi, reti di sensori intelligenti (Smart IoT), componenti personalizzati, ...

Un'istanza su un risultato “recente”

Riconoscimento facciale che combina reti neurali (profonde) e altri approcci ML. A partire da quattro milioni di immagini del viso appartenenti a più di 4.000 identità. Alla domanda se due foto mostrano la stessa persona, DeepFace risponde correttamente il 97,25% delle volte, solo un passo indietro gli umani (97,53%).

(Automatic) Machine Translation

I giganti della tecnologia Google, Microsoft e Facebook stanno tutti applicando le lezioni del machine learning alla traduzione. Per esempio, *Google Neural Machine Translation* per lo strumento Google Translate dal 2016, ma anche le piccole imprese stanno facendo grandi progressi: DeepL dall'agosto 2017 è un sistema completamente basato sulla **rete neurale**.

Musica

“AIVA prima ha composto un brano per solo piano, [...] , poi un intero album, Genesi, per piano e orchestra; infine la musica per la festa nazionale del Lussemburgo; e qualche mese fa, la colonna sonora per uno dei videogame più popolari del mondo, Battle Royale di Fortnite.”

“Come ci riesce è presto detto: al software sono stati fatti conoscere, diciamo così, gli spartiti delle composizioni dei più grandi autori della storia, da Mozart a Beethoven fino a Bach. Ha studiato dai migliori, insomma, con una tecnica che si chiama ‘deep learning’. Da qui AIVA ha ricavato gli schemi ricorrenti di una composizione musical, e a quanto pare è in grado di replicarli adattandosi alla richiesta che viene fatta”

Turing award 2018

Il 27 marzo 2019, l’Association for Computing Machinery, la più grande società al mondo di professionisti informatici, ha annunciato che Drs. Hinton, LeCun e Bengio avevano vinto il Premio Turing di quell’anno per il loro lavoro sulle reti neurali. Il Premio Turing, introdotto nel 1966, è spesso chiamato il Premio Nobel per l’informatica e include un premio da 1 milione di dollari, che i tre scienziati condivideranno.

“Per innovazioni concettuali e ingegneristiche che hanno reso le *deep neural network* una componente fondamentale del computing.”

2020 news!

“Coronavirus, a Roma si usa l’intelligenza artificiale per abbattere i tempi delle diagnosi: da 24h a 30 minuti.”

Imaging Center del Policlinico Universitario. Il sistema è stato *addestrato* a riconoscere dalle immagini delle tomografie computerizzate del torace le aree di alterazione dei polmoni tipiche del Covid-19. In questo modo si passa dalle 24 ore necessarie per le diagnosi con i tamponi a circa 30 minuti. L’uso dell’IA, per diagnosticare l’infezione da Coronavirus è stato fatto per la prima volta in Cina, dove ha dimostrato un’attendibilità del 98,5%! “Della macchina non ci si può fidare al 100%. Ma ha già *imparato* tanto e *continuerà a farlo* grazie ai casi italiani.”

Machine learning quando? (riassumendo)

Opportunità (se utile) e *consapevolezza* (esigenze e limiti). Casi di utilità dei modelli di apprendimento predittivo:

- nessuna (o scarsa) teoria (o conoscenza per spiegare il fenomeno o difficile da formalizzare);
- dati incerti, rumorosi o incompleti (che ostacolano la formalizzazione delle soluzioni);
- ambienti dinamici, non conosciuti in anticipo (ad esempio, adattarsi al comportamento personalizzato in base alle preferenze dinamiche dell'utente).

Richieste: fonte di esperienza di formazione (dati rappresentativi) e tolleranza sulla precisione dei risultati.

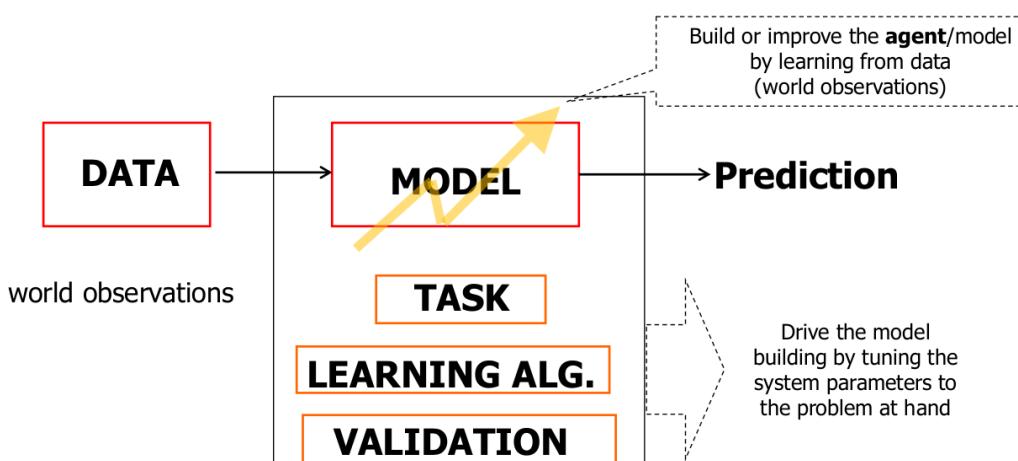
ML: perché?

Un'opportunità per conoscere *nuovi paradigmi informatici* con un approccio diverso. Approcci IA algoritmici/classici, ad esempio trattamento dell'incertezza, tolleranza dell'imprecisione, ...

Tipico dell'area del *soft computing/intelligenza computazionale*. Trovare *soluzioni approssimative* per problemi difficili da formalizzare con algoritmi "fatti a mano". Costruire nuove soluzioni robuste e *sistemi intelligenti* ampiamente applicabili.

Ma **NON** è una metodologia approssimativa!

È un approccio rigoroso per trovare una *funzione approssimativa* per affrontare problemi complessi.



Overview of a ML System

9.1.1 Apprendimento supervisionato

Dato un esempio di addestramento come $\langle \text{input}, \text{output} \rangle = \langle x, d \rangle$ (esempi etichettati) per una funzione sconosciuta f (nota solo nei punti dati dell'esempio), trova una *buona* approssimazione di f (un'*ipotesi* h che può essere usata per previsione su dati invisibili x').

Valore target: il valore desiderato d o t o y è dato dall'insegnante secondo $f(x)$; può essere un'etichetta categoriale o numerica.

- **Classificazione:** $f(x)$ restituisce la classe corretta (presunta) per x . $f(x)$ è una *funzione a valori discreti* $\in \{1, 2, \dots, k\}$ classi.
- **Regressione:** approssimare una funzione target a valori reali (in \mathbb{R} o \mathbb{R}^K)

Entrambi come compito di approssimazione di funzioni.

9.1.2 Apprendimento non supervisionato

Apprendimento senza supervisione: nessun insegnante! TR (Training Set) = set di dati senza etichetta $\langle x \rangle$. Per esempio per trovare *raggruppamenti naturali* in un insieme di dati

- Raggruppamento
- Riduzione/visualizzazione/pre-elaborazione della dimensionalità
- Modellazione della densità dei dati

Clustering: partizione dei dati in cluster (sottoinsiemi di dati “simili”). Centroidi: centro dei cluster.

9.1.3 Modelli e rassegna di concetti utili

Modello: definisce la classe di funzioni che la macchina di apprendimento può implementare (spazio delle ipotesi). Obiettivo: acquisire/descrivere le relazioni tra i dati (sulla base del compito), per esempio insieme di funzioni $h(x, w)$, dove w è il parametro (astratto).

Esempio di allenamento (superv.) della forma $(x, f(x) + rumore)$ x è solitamente un vettore di caratteristiche, $(d \text{ o } t \text{ o }) y = f(x) + rumore$ è chiamato valore target.

Funzione target: la vera funzione f

Ipotesi: una funzione proposta h ritenuta simile a f . Un'espressione in un determinato linguaggio che descrive le relazioni tra i dati

Spazio ipotesi: lo spazio di tutte le ipotesi (modelli specifici) che possono, in linea di principio, essere emesse dall'algoritmo di apprendimento.

Esempi di modelli

Giusto per avere un'anteprima della diversa rappresentazione delle ipotesi:

- Modelli lineari (la rappresentazione di H definisce uno spazio continuamente parametrizzato di ipotesi potenziali); ogni assegnazione di w è un'ipotesi diversa, ad esempio:

$$h_w(x) = w_1x + w_0$$

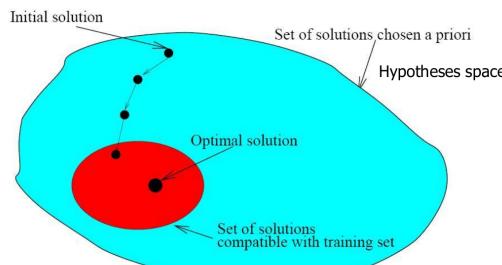
- Regole simboliche: lo spazio delle ipotesi è basato su rappresentazioni discrete; sono possibili regole diverse, ad esempio:

$$\begin{aligned} \text{if } (x_1 = 0) \text{ and } (x_2 = 1) \text{ then } h(x) = 1 \\ \text{else } h(x) = 0 \end{aligned}$$

- Modelli probabilistici: stima $p(x, y)$.
- Approcci basati sull'istanza: prevedere il valore medio y dei vicini più vicini (basato sulla memoria).

9.1.4 Algoritmi di apprendimento

Basandosi su dati, task e modello, l'apprendimento come *ricerca* (euristica) attraverso lo spazio delle ipotesi H dell'**ipotesi migliore** cioè la migliore approssimazione alla funzione target (sconosciuta).



Ricerca locale

In genere la ricerca della h con il minimo “errore”. H potrebbe non coincidere con l’insieme di tutte le possibili funzioni e la ricerca non può essere esaustiva: bisogna fare delle ipotesi → vedremo il ruolo del bias induttivo.

9.1.5 Generalizzazione

Apprendimento: ricerca di una ***buona funzione*** in uno spazio funzionale da dati noti (in genere riducendo al minimo un errore/perdita). **Buona** rispetto all’errore di generalizzazione: misura la precisione con cui il modello prevede su nuovi campioni di dati (errore/perdita misurata su nuovi dati, basso errore, alta precisione e viceversa). La generalizzazione è il punto cruciale del ML! Strumenti ML facili da usare rispetto a un uso corretto del ML.

Fase di apprendimento (formazione, adattamento): costruire il modello da dati conosciuti, di addestramento (e bias).

Fase predittiva (test): applicare al nuovo esempio (prendiamo l’input x , calcoliamo la risposta dal modello, confrontiamo con il suo obiettivo che il modello non ha mai visto); valutazione dell’ipotesi predittiva, ovvero della **capacità di generalizzazione**.

Nota: quando parliamo di *prestazioni* in ML ci riferiamo all’*accuratezza predittiva* stimata dall’errore calcolato sul set di test.

Capitolo 10

Concept Learning

Concept Learning: inferire una *funzione booleana* da esempi di addestramento positivi e negativi $C : X \rightarrow \{t, f\}$ or $\{+, -\}$ or $\{0, 1\}$, dove X è detto *spazio d'istanza*.

Definizione 10.0.1. $h : X \rightarrow \{0, 1\}$ soddisfa x se $h(x) = 1$.

Definizione 10.0.2. Un “esempio” è una coppia $\langle x, c(x) \rangle$ in D (o *TR set*).

Definizione 10.0.3. Un'ipotesi h è **consistente** con

- un esempio $\langle x, c(x) \rangle$, x in X , se $h(x) = c(x)$;
- D , se $h(x) = c(x)$ per ogni esempio di addestramento $\langle x, c(x) \rangle$ in D .

Esempio

Abbiamo una funzione sconosciuta con quattro input, una funzione di uscita e una tabella di verità: conosciamo alcuni dei valori di uscita delle combinazioni degli input; l'obiettivo è trovare una funzione h che approssimi la funzione sconosciuta. Si tratta quindi di un **problema mal posto** (inverso): possiamo violare l'esistenza, l'unicità, la stabilità della soluzione (o delle soluzioni). Ci sono $2^{16} = 2^{2^4} = 65536$ possibili funzioni booleane su quattro funzioni di input: non possiamo capire quale sia corretta finché non abbiamo visto ogni possibile coppia input-output; dopo 7 esempi abbiamo ancora 2^9 possibilità. Nel caso generale:

$$|H| = 2^{\# \text{istanze}} = 2^{2^n}$$

per ingressi/uscite binari, $n = \text{dimensione ingresso}$.

Lavoreremo con uno spazio di ipotesi ristretto: iniziamo scegliendo uno spazio di ipotesi H che è notevolmente più piccolo dello spazio di tutte le possibili funzioni (*language bias*). Vedremo:

- **Regole congiuntive** (semplici) (in una H discreta e finita)
- **Funzioni lineari** (in un H continuo e infinito)

Vedremo come organizzare efficientemente la ricerca in questo spazio discreto: alcuni algoritmi per un insieme molto ristretto di ipotesi; non solo regole congiuntive, ma assumeremo che non ci siano **rumori** nei dati.

10.1 Regole congiuntive

Quante regole congiuntive ci sono? Quante ipotesi h è possibile fare? È possibile avere una regola congiuntiva che è sempre vera, quattro singole, sei coppie, quattro triple o una quadrupla, per un totale di 16. Nel caso generale:

- Letterali positivi, per esempio: $h_1 = l_1$, $h_2 = (l_1 \text{ and } l_2)$, $h_3 = \text{true}$, ...
Regole congiuntive semplici: $|H| = 2^n$
- Letterali (anche $\text{not}(l_i)$): $|H| = 3^n + 1$

10.1.1 Rappresentazione delle ipotesi

Un'ipotesi h è una congiunzione di vincoli sugli attributi. Ogni vincolo può essere un valore specifico, un valore “non importante” oppure un valore non permesso (ipotesi nulla).

10.1.2 Ipotesi di apprendimento induttivo

Assumiamo che:

“Qualsiasi ipotesi trovata per approssimare la funzione obiettivo ben oltre gli esempi di addestramento, approssimerà bene anche la funzione obiettivo rispetto agli esempi non osservati”.

Si ricerca quindi una funzione $h(x) = c(x) \forall x \in D$, cioè *consistente* con D , ma non è dato sapere se $h(x) = c(x) \forall x \in X$.

Tuttavia, le dimensioni da esplorare nello spazio delle ipotesi sono molto grandi, anche infinite: la strutturazione dello spazio di ricerca è un aspetto strategico per rendere efficiente tale ricerca.

10.1.3 General to Specific Ordering

Definizione 10.1.1. Siano h_j e h_k funzioni a valori booleani definite su X . Allora h_j è **più generale o uguale** a h_k (scritto $h_j \geq h_k$) sse

$$\forall x \in X : [(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

Esempi su binario l_i : $l_1 \geq (l_1 \wedge l_2)$, l_1 versus l_2 non confrontabile.

La relazione \geq impone un *ordine parziale* (p.o.) sullo spazio delle ipotesi H che viene utilizzato da molti metodi di apprendimento concettuale. Possiamo trarre vantaggio da questo p.o. per organizzare efficacemente la ricerca in H .

10.1.4 Algoritmo Find-S

Sfrutta questo ordine parziale per cercare in h efficientemente (*senza enumerare esplicitamente* ogni $h \in H$).

1. Inizializza h all'ipotesi più specifica in H .

2. For each **positive** training instance x

 For each attribute a_i in h

 If the a_i in h is satisfied by x then do nothing

 else replace a_i in h by the next more general constraint that is

 satisfied by x (e.g. remove from h literals not satisfying x)

3. Output hypothesis h .

Proprietà di Find-S

Spazio delle ipotesi descritto da congiunzioni di attributi: limite rigido!

L'algoritmo produrrà l'*ipotesi più specifica* all'interno di H che è coerente con gli esempi di allenamento **positivi**. L'ipotesi di output h sarà consistente anche con gli esempi negativi a condizione che il concetto di destinazione sia contenuto in H , poiché $c \geq h$.

Non si sa se il learner è convergente al concept target, nel senso che non è in grado di determinare se ha trovato l'unica ipotesi coerente con gli esempi formativi. Non è possibile stabilire se i dati di addestramento sono incoerenti, poiché ignora gli esempi di addestramento negativi: *nessuna tolleranza al rumore!*

Perché preferire l'ipotesi più specifica? E se ci sono più ipotesi massimamente specifiche?

10.1.5 Versions Spaces

Idea chiave: uscire una descrizione con tutto l'insieme delle ipotesi consistenti con D. Possiamo farlo senza enumerarli tutti.

$$\text{Consistente}(h, D) := \forall \langle x, c(x) \rangle \in D \ h(x) = c(x)$$

Il **version space**, $VS_{H,D}$, rispetto allo spazio delle ipotesi H , e l'insieme di addestramento D , è il sottoinsieme di ipotesi da H consistente con tutti gli esempi di addestramento:

$$VS_{H,D} = \{h \in H \mid \text{Consistente}(h, D)\}$$

Algoritmo List-Then Eliminate

Un'alternativa sarebbe il List-Then Eliminate

1. *Version Space* \leftarrow una lista contenente ogni ipotesi in H
2. Per ogni esempio di addestramento $\langle x, c(x) \rangle$ rimuovi da *Version Space* qualsiasi ipotesi che non sia coerente con l'esempio di addestramento $h(x) \neq c(x)$
3. Visualizza l'elenco di ipotesi in *Version Space*.

Irrealistico: enumerazione esaustiva di tutte le h in H .

Rappresentare i Version Spaces

Il **confine generale**, G , dello spazio delle versioni $VS_{H,D}$ è l'insieme dei membri massimamente generali (di H consistente con D).

Il **confine specifico**, S , dello spazio delle versioni $VS_{H,D}$ è l'insieme dei membri massimamente specifici (di H consistente con D).

Teorema 10.1.1. *Ogni membro dello spazio delle versioni si trova tra questi confini*

$$VS_{H,D} = \{h \in H \mid (\exists s \in S) (\exists g \in G) (g \geq h \geq s)\}$$

dove $x \geq y$ significa x è più generale o uguale di y .

Possiamo fare una *ricerca completa* di ipotesi coerenti usando i due confini S e G per VS, cioè non limitati a S come per Find-S.

10.1.6 Algoritmo Candidate Elimination

$G \leftarrow$ ipotesi massimamente generali in H .

$S \leftarrow$ ipotesi massimamente specifiche in H .

Per ogni esempio di addestramento $d = \langle x, c(x) \rangle$ se d è un esempio **positivo** rimuove da G qualsiasi ipotesi che sia inconsistente con d (definizione di VS) e per ogni ipotesi s in S che non è consistente con d (**generalize S**) rimuove s da S e aggiunge a S tutte le generalizzazioni minime h di s tali che

- h coerente con d ,
- alcuni membri di G sono più generali di h ,

ed infine rimuove da S qualsiasi ipotesi più generale di un'altra ipotesi in S .

Se d è un esempio **negativo** rimuove da S qualsiasi ipotesi che è incoerente con d e per ogni ipotesi g in G che non è coerente con d (**specialize G**) rimuove g da G e aggiunge a G tutte le specializzazioni minime h di g tali che

- h coerente con d ,
- alcuni membri di S sono più specifici di h ,

ed infine rimuove da G qualsiasi ipotesi meno generale di un'altra ipotesi in G .

10.2 Bias Induttivo

Lo spazio delle nostre ipotesi non è in grado di rappresentare un semplice concetto di obiettivo disgiuntivo. **Bias**: supponiamo che l'ipotesi spazio H contenga il concept target c . In altre parole, c può essere descritto da una *congiunzione* (dall'operatore AND) di letterali.

10.2.1 Unbiased Learner

Idea: scegliamo una H che esprime ogni concetto insegnabile, questo significa che H è l'insieme di tutti i possibili sottoinsiemi di X , il powerset $P(X)$.

$$|X| = 96, |P(X)| = 2^{96} \sim 10^{28} \text{ concetti distinti}$$

$H =$ disgiunzioni, congiunzioni, negazioni. H contiene sicuramente il concept target.

Cosa sono S e G in questo caso? Si assumono esempi positivi (x_1, x_2, x_3) ed esempi negativi (x_4, x_5)

$$S : \{(x_1 \vee x_2 \vee x_3)\} \quad G : \{\neg(x_4 \vee x_5)\}$$

Gli unici esempi classificati in modo univoco sono gli esempi di formazione stessi (H può rappresentarli). In altre parole, per apprendere il target concept bisognerebbe presentare ogni singola istanza in X come esempio di formazione.

Proprietà: un unbiased learner *non è in grado di generalizzare*.

Prova: ogni istanza non osservata sarà classificata come positiva esattamente per metà dell'ipotesi in VS e negativa per l'altra metà (*rifiuto*). Infatti

$$\begin{aligned} \forall h \text{ consistente con } x_i, (\exists h'. h' \equiv h \wedge h'(x_i) \neq h(x_i)) \\ h \in VS \rightarrow h' \in VS \end{aligned}$$

Sono identici su D.

Futilità del Bias-Free Learning

Un learner che non fa ipotesi preliminari sull'identità del concept target non ha basi razionali per classificare istanze invisibili. (Restrizione, preferenza) Bias non solo assunto per l'efficienza, ma *necessario per la generalizzazione*. Tuttavia, non ci dice (non quantifica) quale sia la migliore soluzione per la generalizzazione.

10.2.2 Inductive Bias

Considera:

- Algoritmo di apprendimento concettuale L.
- Istanze X, target concept c .
- Esempi di addestramento $D_c = \{\langle x, c(x) \rangle\}$.
- Sia $L(x_i, D_c)$ la classificazione assegnata all'istanza x_i da L dopo l'addestramento su D_c .

Definizione 10.2.1. Il bias induttivo di L è un qualsiasi insieme minimo di asserzioni B tale che per ogni concetto di obiettivo c e corrispondenti dati di addestramento D_c

$$(\forall x_i \in X)[B \wedge D_c \wedge x_i] \vdash L(x_i, D_c)$$

dove $A \vdash B$ significa che A implica logicamente B (segue deduttivamente da).

Tre Learners con Bias differenti

Rote learner (*lookup table*): memorizza esempi, classifica x se e solo se corrisponde a un esempio osservato in precedenza. Nessun bias induttivo → nessuna generalizzazione.

Version space e candidate elimination algorithm → Bias: lo spazio delle ipotesi contiene il concetto di destinazione (insieme agli attributi).

Find-S-Bias: lo spazio delle ipotesi contiene il concetto di destinazione e tutte le istanze sono negative a meno che l'opposto non sia implicato dalla sua altra conoscenza (visto come esempi positivi). In altre parole abbiamo un *bias linguistico* a causa dell'AND sui letterali più il *bias di ricerca* a causa della preferenza dell'ipotesi più specifica.

Capitolo 11

Modelli Lineari

11.1 Regressione

La *regessione* è quel processo di stima di una funzione a valori reali sulla base di un insieme finito di campioni rumorosi.

Considereremo il caso semplificato della regressione lineare a una variabile: iniziamo con una variabile di input x , una variabile di output y . Assumiamo un modello $h_w(x)$ espresso come $out = w_1x + w_0$, dove w_i è il coefficiente/parametro libero (peso) a valori reali. **Adattamento** dei dati con una “linea retta”.

Esempio

Trovare h (modello lineare) che si *adatta* meglio ai dati (partendo da i dati osservati di valore x e y). Supponendo che la variabile data (y) sia (linearmente) correlata a un’altra variabile (x) o variabili per $y = w_1x + w_0 + rumore$, dove w è il parametro libero e il *rumore* è l’errore nella misurazione dei target (con distribuzione normale), costruiamo un modello (trovando valori w) per prevedere/stimare il prezzo (y) di punti per altri valori x (non osservati).

Spazio ipotesi infinito (valori w continui), ma abbiamo una bella soluzione dalla matematica classica (risalendo a Gauss/Legendre ~1795!). Sorprendentemente possiamo “imparare” da questo strumento di base; sebbene semplice, include molti concetti rilevanti della moderna ML ed è una base di metodi evoluti nel campo. Definiamo una funzione di perdita/errore e utilizziamo l’approccio Least Mean Square (LMS).

Dato un insieme di training $(x_p, y_p) \quad p = 1 \dots l$

Trovare $h_w(x)$ nella forma $w_1x + w_0$ tale che si minimizzi la perdita attesa sui dati di training.

Per la perdita si usa l'errore quadratico.

Least (Mean) Square: trovare w che minimizzi la somma dei quadrati degli errori

$$Loss(h_w) = E(w) = \sum_{p=1}^l (y_p - h_w(x_p))^2 = \sum_{p=1}^l (y_p - (w_1 x + w_0))^2$$

dove x_p è il p -esimo input, y_p l'output per p , w i parametri liberi e l il numero di esempi.

Per ottenere la media dividere per l .

Il metodo Least Mean Square è un approccio standard alla soluzione approssimativa di sistemi sovradeterminati, cioè insiemi di equazioni in cui ci sono più equazioni che incognite.

Si ricordi che il minimo locale corrisponde a un punto stazionario: il *gradiente* è nullo

$$\frac{\partial E(w)}{\partial w_i} = 0, \quad i = 1, \dots, \text{dim_input} + 1$$

Regole di base:

$$\frac{\partial}{\partial w} k = 0, \quad \frac{\partial}{\partial w} w = 1, \quad \frac{\partial}{\partial w} w^2 = 2w, \quad \frac{\partial(f(w))^2}{\partial w} = 2f(w) \frac{\partial(f(w))}{\partial w}$$

quindi

$$\begin{aligned} \frac{\partial E(w)}{\partial w_i} &= \frac{\partial(y - h_w(x))^2}{\partial w_i} = 2(y - h_w(x)) \frac{\partial(y - h_w(x))}{\partial w_i} \\ &= 2(y - h_w(x)) \frac{\partial(y - (w_1 x + w_0))}{\partial w_i} \end{aligned}$$

allora

$$\frac{\partial E(w)}{\partial w_0} = -2(y - h_w(x)) \quad e \quad \frac{\partial E(w)}{\partial w_1} = -2(y - h_w(x)) \cdot x$$

da cui si ottiene

$$\Rightarrow w_1 = \frac{\sum x_p y_p - \frac{1}{l} \sum x_p \sum y_p}{\sum x_p^2 - \frac{1}{l} (\sum x_p)^2} = \frac{Cov[x, y]}{Var[x]}, \quad p = 1 \dots l$$

$$\Rightarrow w_0 = \bar{y} - w_1 \bar{x}, \quad \text{con } \bar{y} = \frac{1}{l} \sum_{p=1}^l y_p, \quad \bar{x} = \frac{1}{l} \sum_{p=1}^l x_p$$

La derivazione precedente suggerisce la linea per costruire un algoritmo iterativo sulla base di $\frac{\partial E(w)}{\partial w_i}$. Gradiente = direzione di salita: possiamo muoverci verso il minimo con una discesa del gradiente ($\Delta w = -\text{gradiente di } E(w)$).

Ricerca locale: inizia con il vettore di peso iniziale e lo diminuisce iterativamente fino a ridurre al minimo la funzione di errore (discesa più ripida).

$$w_{new} = w + \eta \cdot \Delta w$$

dove *eta* (η) è una costante (dimensione del passo), chiamata **velocità di apprendimento**.

11.1.1 Gradient Descent

Prima abbiamo ottenuto

$$\Delta w_0 = -\frac{\partial E(w)}{w_0} = 2(y - h_w(x)) \quad \Delta w_1 = -\frac{\partial E(w)}{w_1} = 2(y - h_w(x)) \cdot x$$

Questa è una regola di “correzione di errore” (chiamata **regola delta**) che modifica w proporzionalmente all’errore (*target – output*):

- (*target* $y - output$) = $err = 0 \rightarrow$ nessuna correzione
- *output* $> target \rightarrow (y - h) < 0$ l’output è troppo alto: Δw_0 negativo \rightarrow si riduce w_0 e se (input $x > 0$) Δw_1 negativo \rightarrow riduci w_1 , altrimenti aumenta w_1
- *output* $< target \rightarrow (y - h) > 0$ l’output è troppo basso

L’approccio di discesa a gradiente è un approccio di ricerca locale semplice ed efficace alla soluzione LMS e ci permette di cercare attraverso uno spazio di ipotesi infinito! Può essere facilmente applicato sempre per H continua e perdite differenziabili: non solo per modelli lineari, ad esempio reti neurali e modelli di deep learning. È efficiente? Sono possibili molti miglioramenti, ad esempio metodi di Newton, gradiente coniugato, ...

Estensione a l dati

$$\Delta w_0 = -\frac{\partial E(w)}{w_0} = 2 \sum_{p=1}^l (y_p - h_w(x_p))$$

$$\Delta w_1 = -\frac{\partial E(w)}{w_1} = 2 \sum_{p=1}^l (y_p - h_w(x_p)) \cdot x_p$$

dove x_p è il p -esimo input, y_p l'output per p , w i parametri liberi e l il numero di esempi.

Possiamo aggiornare w dopo un'“epoca” di l dati di addestramento, *algoritmo batch* (come usato sopra), oppure possiamo aggiornare w dopo ogni pattern p , *algoritmo on-line* (discesa del gradiente stocastico): può essere il più veloce, ma richiede un passo più piccolo.

Estensione a input di vettori

Questo è il caso standard, utilizzando da due a centinaia di variabili/features di input $x = [x_1, x_2, \dots, x_n]$. Il pattern di input è un vettore: tante tante possibilità in una vasta gamma di campi applicativi.

Notazione

Pattern	x_1	x_2	x_i	x_n
Pat 1	$x_{1,1}$	$x_{1,2}$	$x_{1,i}$	$x_{1,n}$
...				
Pat p	$x_{p,1}$	$x_{p,2}$	$x_{p,i}$	$x_{p,n}$
...				

X è una matrice $l \times n$: l righe, n colonne (features/variabili). Supponiamo il vettore colonna per x e w .

l : numero di dati

n : dimensione del vettore di input

y_p , $p = 1 \dots l$: target

$$w^T x + w_0 = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n = w_0 + \sum_{i=1}^n w_i x_i$$

Notare che spesso, come prima, la notazione di trasposizione T in w^T viene omessa.

w_0 è l'*intercept, threshold, bias, offset* . . .

Spesso è conveniente includere la costante $x_0 = 1$ in modo da poterla scrivere come: $w^T x = x^T w$, $x^T = [1, x_1, x_2, \dots, x_n]$

Quindi

$$h(x_p) = x_p^T w = \sum_{i=0}^n x_{p,i} w_i$$

Riepilogo

Dato un insieme di training (x_p, y_p) con $p = 1 \dots l$, **trovare** $h_w(x)$ nella forma $w_1 x + w_0$ tale che si minimizzi la perdita attesa sui dati di training.

$$E(w) = \sum_{p=1}^l (y_p - x_p^T w)^2 = \|y - Xw\|^2$$

$$\Delta w_i = -\frac{\partial E(w)}{\partial w_i} = 2 \sum_{p=1}^l (y_p - h_w(x_p)) \cdot x_{p,i} = 2 \sum_{p=1}^l (y_p - x_p^T w) \cdot x_{p,i}$$

Un algoritmo semplice:

1. Inizia con il vettore peso w iniziale (piccolo), fissa *eta* ($0 < \eta < 1$).
2. Calcola $\Delta w = -\text{gradient of } E(w) = -\frac{\partial E(w)}{\partial w}$.
3. Calcola $w_{new} = w + \eta \cdot \Delta w$ (cioè per ogni w_i), dove η è il parametro “dimensione del passo” (velocità di apprendimento).
4. Ripeti (2) fino a quando la convergenza o $E(w)$ è “sufficientemente piccolo”.

Vantaggi dei modelli lineari

Se funziona bene è un modello “meraviglioso”

- Molto semplice
- Tutte le informazioni sui dati sono in w
- Facile da interpretare: pratica quotidiana in medicina, biologia, chimica, economia, . . .

- Sono consentiti dati rumorosi

Gli statistici sono felici (belle proprietà). Fenomeni lineari: un sogno per la scienza, ideale per fare una “legge naturale”. Una linea di base per l’apprendimento, viene utilizzato/incluso nei modelli più complessi.

Limitazioni

Nota che $h_w(x) = w_1x + w_0$ e $h_w(x) = w^T \cdot x$, come i modelli parametrici statistici, sono detti “**lineari**”: tale termine non si riferisce al fatto che generano una retta, ma piuttosto al modo in cui i coefficienti di regressione w si verificano nell’equazione di regressione. Quindi, possiamo usare anche input trasformati, come x, x_2, x_3, x_4, \dots con input e output di relazione **non lineare**, tenendo il macchinario di apprendimento (soluzione del Least Square) sviluppato finora:

$$h_w(x) = w_0 + w_1x + w_2x^2 + \cdots + w_Mx^M = \sum_{j=0}^M w_jx^j$$

11.1.2 Una generalizzazione: LBE

Linear basis expansion:

$$h_w(x) = \sum_{k=0}^K w_k \phi_k(x)$$

Aumenta il vettore di input con variabili aggiuntive che sono trasformazioni di x secondo una funzione phi ($\phi_k : \mathbb{R}^n \rightarrow \mathbb{R}$).

Il modello è lineare nei parametri (anche in phi, non in x): possiamo usare lo stesso algoritmo di apprendimento di prima!

Critica

Quale phi (ϕ)? Verso i cosiddetti approcci “**dizionario**”.

PRO: è più espressivo, può modellare relazioni più complicate (rispetto a quelle lineari).

CONTRO: con un’ampia base di funzioni, richiediamo metodi per controllare la *complessità* del modello.

Un modello troppo semplice non si adatta bene ai dati, soluzione parziale: **underfitting**; un modello troppo complesso è altamente sensibile a lievi perturbazioni dei dati, soluzione ad alta varianza (“*Bias-variance*”): **overfitting**.

Si vuole scegliere una regolarizzazione per bilanciare bias e varianza attraverso il controllo della complessità del modello (vista come flessibilità).

La regolarizzazione può controllare l’overfitting penalizzando le funzioni “complesse” (pesi elevati), cioè riducendo i valori dei coefficienti w , pur mantenendo la flessibilità dello spazio delle ipotesi.

Rasoio di Ockham [1300]:

“*La spiegazione più semplice è più probabilmente quella corretta*”

oppure

“*Preferisci l’ipotesi più semplice che si adatta ai dati*”

Questo concetto fondamentale in ML verrà ritrovato e alla fine lo “razionalizzeremo”, per quantificarlo. Considereremo la complessità non come costo computazionale ma come misura della flessibilità del modello per adattarsi ai dati.

11.1.3 Regolarizzazione da Ridge Regression

Ridge regression/**regolarizzazione di Tikhonov**, *smoothed models*: è possibile aggiungere vincoli alla somma del valore di $|w_j|$ favorendo modelli “sparsi”, ad esempio con meno termini a causa dei pesi $w_j = 0$ (o $w_j \rightarrow 0$) (significa un modello meno complesso).

Termine dati errore + **Termine regolarizzazione/penalità**

$$Loss(h_w) = \sum_{p=1}^l (y_p - h_w(x_p))^2 + \lambda \|w\|^2$$

dove $\|w\| = \sum_i w_i^2$ e *lambda* (λ) è chiamato coefficiente di regolarizzazione (un parametro costante o iperparametro).

Effetto: **decadimento del peso**, in pratica aggiunge $2\lambda w$ al gradiente della perdita

$$w_{new} = w + \eta \cdot \Delta w - 2\lambda w$$

Per esempio, con gradiente zero diminuisce il valore di ogni w con una frazione del vecchio w .

Applicabilità generale (non solo per polinomi), per esempio possiamo controllare la complessità del modello semplicemente usando lambda, senza conoscere M per i polinomi, o anche quando la complessità del modello non è nota. Notare il bilanciamento (trade-off) tra i due termini: il termine dei dati di piccolo errore (minimizza solo l'errore di addestramento) non è sufficiente per noi, vogliamo anche controllare la complessità del modello per controllare l'***overfitting*** e quindi introduciamo il secondo termine nella minimizzazione. D'altra parte possiamo sforare perché un peso eccessivo al secondo termine (valore lambda alto) porta a focalizzare la minimizzazione solo o prevalentemente sulla regolarizzazione, quindi l'errore dei dati (primo termine) potrebbe crescere troppo, ovvero passare all'***underfitting***. Il compromesso è regolato dal valore di *lambda* (λ).

11.1.4 Limitazioni delle Fixed Basis Functions

Avendo una funzione di base lungo ogni dimensione D, lo spazio di input richiede un numero combinatorio di funzioni: *la maledizione della dimensionalità*. Ad esempio i polinomi generali di ordine 3 utilizzano tutte le combinazioni di variabili di input dovute ai prodotti $x_1x_2, x_2x_3, \dots, x_1x_2x_3, \dots \rightarrow D^3$ (approssimazione all'aumentare di D). I Phi vengono *fissati* prima di osservare i dati di allenamento. In altri modelli, vedremo come possiamo farla franca con meno funzioni di base, scegliendole utilizzando i dati di addestramento: phi (in uno strato computazionale nascosto) dipende da w e il modello non è lineare nei parametri, per esempio le ***reti neurali***. In altri modelli ancora, il calcolo del nuovo spazio di incorporamento (trasformazione dell'input) viene effettuato implicitamente attraverso le funzioni del kernel e controllando la complessità del modello, per esempio ***SVM***.

11.2 Classificazione

Gli stessi modelli (utilizzati per la regressione) possono essere utilizzati per la **classificazione**: *target categorici*, ad es. 0/1 o -1/+1. In questo caso utilizziamo un iperpiano (wx) che assume valori negativi o positivi: sfruttiamo tali modelli per decidere se un punto x appartiene alla zona positiva o negativa dell'iperpiano (per classificarlo), quindi vogliamo impostare w (imparando) in modo tale da ottenere una buona precisione di classificazione.

Definizione 11.2.1. Il **decision boundery** è il luogo geometrico dei punti dove $wx = 0$

Se si desidera una funzione che restituisce valori $\in \{0, 1\}$ si usa

$$h(x) = \begin{cases} 1 & \text{se } wx + w_0 \geq 0 \\ 0 & \text{altrimenti} \end{cases}$$

altrimenti per valori $\in \{-1, +1\}$

$$h(x) = sign(wx + w_0)$$

$$h(x_p) = sign(x_p^T w) = sign \left(\sum_{i=0}^n x_{p,i} w_i \right)$$

Si noti che, dato il bias w_0 , nella LTU dire $h_x = w^T x + w_0 \geq 0$ è equivalente a dire $h_x = w^T x \geq -w_0$ con $-w_0$ come valore di “soglia”. Le due forme identificano la stessa zona positiva del classificatore: la seconda enfatizza il ruolo del bias come valore di soglia per “attivare” l’uscita $+1$ del classificatore.

Riepilogo

I modelli sono **sottoposti a training** (tramite il training set) secondo LMS (*Least Medium Square*) su wx attraverso la semplice discesa di gradiente usato per la regressione lineare. È possibile applicare anche la *linear basis expansion* e *Tikhonov* per regolarizzare.

In seguito i modelli possono essere **usati** per costruire classificazioni applicando la funzione di soglia $h(x) = sign(wx)$. L’errore può essere calcolato come errore di classificazione o numero di modelli classificati erroneamente (non solo dall’errore quadratico medio):

$$L(h(x_p, d_p)) = \begin{cases} 0 & \text{se } h(x_p) = d_p \\ 1 & \text{altrimenti} \end{cases} \quad \text{Errore_medio} = \frac{1}{l} \sum_{p=1}^l L(h(x_p, d_p))$$

Definizione 11.2.2. L’**accuracy** (accuratezza del modello) è la media dei correttamente classificati, ossia $(l - \#err)/l$.

Nota: l’algoritmo converge asintoticamente al minimo di LS, tuttavia per il classificatore non ottiene il numero minimo di errori di classificazione (proprio perché la *Loss* non teneva conto della classificazione).

Limitazioni

In geometria, due insieme di punti in uno spazio bidimensionale sono **linearmente separabili** quando i due insieme possono essere completamente

separati da una linea; più in generale, due gruppi sono *linearmente separabili* in uno spazio *n-dimensionale* se possono essere separati da un iperpiano (*n-1*)-dimensionale.

Il decision boundary lineare fornisce soluzioni esatte solo per insiemi di punti linearmente separabili.

Capitolo 12

Introduzione all'apprendimento degli alberi di decisione

Gli alberi decisionali rappresentano una disgiunzione di congiunzioni di vincoli sul valore degli attributi, o regole *if-then-else*. In particolare, lo spazio delle ipotesi di un albero di decisione è capace di esprimere qualsiasi funzione finita a valori discreti.

12.1 Top-down induction of Decision Trees

ID3 (Quinlan, 1986) è un algoritmo di base per l'apprendimento di DT. Dato un insieme di esempi di training, l'algoritmo per la creazione di DT esegue la ricerca nello spazio degli alberi decisionali. La costruzione dell'albero è *dall'alto verso il basso*. L'algoritmo esegue una ricerca *greedy*. Seleziona l'**attributo migliore**, quindi crea un nodo discendente per ogni possibile valore di questo attributo e gli esempi sono partizionati in base a questo valore. Il processo viene ripetuto per ogni nodo successore fino a quando tutti gli esempi non vengono classificati correttamente o non sono rimasti attributi.

Scegliere l'attributo migliore

Usiamo la nozione di **entropia**, comunemente usata nella teoria dell'informazione. L'entropia misura l'*impurità* di una raccolta di esempi; dipende dalla distribuzione della variabile casuale p .

- S è una raccolta di esempi di formazione.
- p_+ la proporzione di esempi positivi in S .
- p_- la proporzione di esempi negativi in S .

$$Entropy(S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_-$$

con $0 \leq p \leq 1$ e $0 \leq entropy \leq 1$ e si assume che $0 \log_2 0 = 0$.

Definizione 12.1.1. L'information gain è la riduzione attesa dell'entropia causata dalla partizione degli esempi su un attributo.

Riduzione attesa dell'entropia conoscendo A:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$Values(A)$ possibili valori per A. S_v sottoinsieme di esempi S per cui A come valore v (*somma ponderata*).

Maggiore è l'information gain, più efficace è l'attributo nella classificazione dei dati di addestramento (maggior variazione nell'omogeneità della distribuzione delle classi sui sottoinsiemi, massima separazione delle classi). L'**omogeneità** ci consente una classificazione "chiara". Poiché l'entropia misura l'omogeneità, l'impurità, della classe del sottoinsieme di esempi, si sceglie A in modo tale da massimizzare $Gain(S, A)$. Lo scopo è separare gli esempi sulla base del target, trovando l'attributo che discrimina gli esempi che appartengono a classi target differenti.

Il problema dell'information gain è che si favoriscono gli attributi con più valori possibili. Come evitare la generazione di tanti sottoinsiemi troppo piccoli?

Viene introdotto un correttivo:

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

dove

$$SplitInformation(S, A) \equiv - \sum_{i=1}^c \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

S_i sono gli insiemi ottenuti partizionando sul valore i di A, fino ai valori di una classe c .

$SplitInformation$ misura l'entropia di S rispetto ai valori di A. Più i dati sono dispersi in modo uniforme, più sono alti.

$GainRatio$ penalizza gli attributi che dividono gli esempi in molte piccole classi.

Problema: $SplitInformation(S, A)$ può essere zero o molto piccolo quando $|S_i| \approx |S|$ per qualche valore i [$\log 1 = 0$]. Un caso estremo è $|S_1| = 0$ $|S_2| = n$.

Per mitigare questo effetto, è stata utilizzata la seguente euristica:

1. calcolare il guadagno per ogni attributo,
2. applicare $GainRatio$ solo agli attributi con $Gain$ superiore alla media.

Ricerca nello spazio delle ipotesi - DT

ID3 esegue una ricerca (*hill-climbing*) nello spazio del possibile DT dal più semplice al sempre più complesso.

- Lo spazio delle ipotesi è completo (rappresenta tutte le funzioni a valori discreti).
- La ricerca mantiene un'unica ipotesi corrente.
- Nessun backtracking, nessuna garanzia di ottimalità (optima locale).
- Utilizza tutti gli esempi disponibili (non incrementali).
- Può terminare prima, accettando classi rumorose.

12.1.1 Bias induttivo - DT

Qual è il bias induttivo dell'apprendimento DT? *Gli alberi più corti sono preferiti a quelli più lunghi* a causa di una ricerca dal semplice al complesso. Questo non basta: questo bias sarebbe esibito anche da un semplice algoritmo breadth first che genera tutti i DT e seleziona quello consistente più breve. Si preferiscono alberi che posizionano vicino alla radice gli attributi con *InformationGain* più alto.

Nota: i DT non si limitano a rappresentare tutte le funzioni possibili. Le restrizioni non riguardano lo spazio delle ipotesi, ma la strategia di ricerca.

Due tipi di bias

Preferenze o **bias di ricerca** (dovuti alla strategia di ricerca): ID3 ricerca uno spazio *completo* delle ipotesi, ma la strategia di ricerca è *incompleta*. Restrizione o **bias linguistici** (dovuti all'insieme di ipotesi esprimibili o considerate): *Candidate-Elimination* ricerca uno spazio di ipotesi *incompleto*, ma la strategia di ricerca è *completa* (tutte le ipotesi coerenti).

Nell'apprendimento di modelli lineari si usa una combinazione di bias.

Perché il bias di ricerca può essere preferito al bias linguistico? In ML si utilizzano tipicamente *approcci flessibili* (capacità universale dei modelli), senza escludere a priori la funzione target sconosciuta; naturalmente, flessibile → problema di overfitting!

12.1.2 Overfittinng

La costruzione di alberi che “si adattano troppo” agli esempi di training può portare a un “overfitting”. Si consideri l’errore dell’ipotesi h sui dati di training, $\text{error}_D(h)$ (errore empirico), e sull’intera distribuzione dei dati X , $\text{error}_X(h)$ (errore previsto o vero).

Definizione 12.1.2. Un’ipotesi h fa *overfitting* sui dati di training se esiste un’ipotesi alternativa $h' \in H$ tale che

$$\text{error}_D(h) < \text{error}_D(h') \quad \text{e} \quad \text{error}_X(h') < \text{error}_X(h)$$

cioè h' si comporta peggio con i dati di training, ma meglio con i dati invisibili.

Approcci flessibili possono facilmente incontrare overfitting.

Evitare l’overfitting nei DT

“Evitare” non è corretto: non esiste un modo per evitarlo, ma esistono strategie per gestirlo in modo consapevole. Due strategie:

1. Fermare la crescita dell’albero prima di avere una classificazione perfetta.
2. Consentire all’albero di fare *overfitting* di dati e quindi *post-potare* l’albero.

Come valutare l’effetto? *Training and validation set*: dividere il training set in due parti (training e validazione) e utilizzare il set di validazione per valutare l’utilità del post-potatura. Altri approcci:

- Utilizzare un test statistico per stimare l’effetto dell’espansione o della potatura.
- *Minimum description length principle*: utilizza una misura della complessità della codifica del DT e degli esempi e interrompe la crescita dell’albero quando questa dimensione di codifica è minima.

Reduced-error pruning

Ogni nodo è un candidato per l'eliminazione. La potatura consiste nel rimuovere un sottoalbero radicato in un nodo: il nodo diventa una foglia e viene assegnata la classificazione più comune. I nodi vengono rimossi solo se l'albero risultante non ha prestazioni peggiori sul **validation set**: ad ogni iterazione viene eliminato il nodo la cui rimozione aumenta maggiormente la precisione sul validation set. La potatura si interrompe quando nessuna potatura aumenta la precisione.

Regola post-pruning

1. Creare l'albero decisionale dal training set.
2. Convertire l'albero in un insieme equivalente di regole:
 - ogni percorso corrisponde a una regola;
 - ogni nodo lungo un percorso corrisponde a una pre-condizione;
 - ogni classificazione foglia alla post-condizione.
3. Potare (generalizzare) ogni regola rimuovendo quelle precondizioni la cui rimozione migliora la precisione sul *validation set* e sul training con una misura pessimistica, statisticamente ispirata (euristica).
4. Ordinare le regole in base all'ordine di accuratezza stimato e considerarle in sequenza durante la classificazione di nuove istanze.

Ogni percorso distinto produce una regola diversa: la rimozione di una condizione può essere basata su un criterio locale (contestuale). L'eliminazione delle precondizioni è **specifico per le regole**, l'eliminazione dei nodi è globale e influisce su tutte le regole. La conversione in regole migliora la leggibilità per gli esseri umani.

12.1.3 Gestire attributi a valore continuo

Finora sono stati considerati valori discreti sia per gli attributi che per il risultato.

Dato un attributo a valore continuo A , creare dinamicamente un nuovo attributo A_c

$$A_c = \text{True} \text{ se } A < c, \text{ False altrimenti}$$

Come determinare il valore di soglia c ?

Determinare le soglie dei candidati calcolando la media dei valori consecutivi in caso di modifica della classificazione. Valutare le soglie dei candidati (attributi) in base al guadagno di informazioni. Il nuovo attributo compete con gli altri.

12.1.4 Gestione dei dati di training incompleti

Come affrontare la **mancanza** del valore di qualche attributo? **Imputation**: utilizzare altri esempi per indovinare l'attributo (all'interno dei dati di addestramento in un dato nodo).

1. *Più comune*: assegnare il valore più comune tra tutti gli esempi di addestramento al nodo/a quelli della stessa classe.
2. Assegnare una probabilità a ciascun valore, in base alle frequenze, e assegnare i valori all'attributo mancante secondo questa distribuzione di probabilità (aggiungendo più esempi ponderati per probabilità).

I valori mancanti nelle nuove istanze da classificare vengono trattati di conseguenza (ponderazione) e viene scelta la classificazione più probabile.

12.1.5 Gestione di attributi con costi differenti

Gli attributi dell'istanza possono avere un costo associato: si preferirebbero alberi decisionali che utilizzano attributi a basso costo. ID3 può essere modificato per tenere conto dei costi:

1. Tan e Schlimmer (1990)

$$\frac{Gain^2(S, A)}{Cost(S, A)}$$

2. Nunez (1988)

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}, \quad \text{dove } w \in [0, 1]$$

Una visione geometrica

Confini decisionali che possono essere prodotti da un DT: gli alberi decisionali dividono lo spazio di input in rettangoli paralleli all'asse e etichettano ogni rettangolo con una delle classi K (foglia dell'albero).

12.1.6 Conclusioni

DT è un approccio popolare per la **classificazione** in un numero discreto di classi.

- Spazio delle ipotesi espressive nell'area degli approcci proposizionali basati su regole.
- Dati da robusti a rumorosi. Gestisce i valori degli attributi mancanti.
- Facile da capire (regole esplicite *if-then-else*, a meno che non siano enormi).
- Molte estensioni allo schema di base... .

Bias linguistici e di ricerca: ID3 ricerca uno spazio di ipotesi completo, con una strategia *greedy* e incompleta. Approccio flessibile: l'*overfitting* è una questione importante, affrontata dalla post-potatura e dalla generalizzazione delle regole indotte.

Capitolo 13

Introduzione alla convalidazione e questioni teoriche

La validazione ha due importanti scopi. Il primo è il **model selection**: stima le prestazioni (*errore di generalizzazione*) di diversi modelli di apprendimento al fine di scegliere quello migliore (per generalizzare), questo include la ricerca dei migliori *iper-parametri* del modello (cioè parametri che non vengono appresi direttamente, che non vengono modificati dalla formazione; ad es. l'ordine del polinomio, lambda di ridge regression, ...), e restituisce un modello. Il secondo è il **model assessment**: sceglie un modello finale, stimando/valutando il suo errore/rischio di previsione (*errore di generalizzazione*) sui nuovi dati di test (misura della qualità/prestazione del modello finale scelto), e restituisce una stima.

Hold out

Se la dimensione del set di dati è sufficiente, ad esempio 50% TR, 25% VL, 25% TS (**insiemi disgiunti**):

- TR: il *training set* viene utilizzato per adattarsi [**training**].
- VL: il *validation set* (o *selection set*) può essere utilizzato per selezionare i modelli migliori (ad es. ottimizzazione degli iperparametri) [**model selection**].
- TS: il *test set* viene utilizzato per la stima dell'errore di generalizzazione (del modello finale) [**valutazione del modello**].

TR + VL a volte sono chiamati solo set di sviluppo/design, ovvero utilizzati per costruire il modello finale.

Nota: la stima effettuata per il model selection (sul validation set) [è a scopo di selezione del modello] non è una buona stima per la fase di valutazione/test di rischio e il test set non può essere utilizzato per il model selection (come validation set).

Cosa succede se il test set viene utilizzato in un ciclo di progettazione (ripetuto)?

Si effettuerebbe un *model selection* e una *valutazione* non affidabile (stima dell'errore di generalizzazione previsto) e non si sarebbe in grado di farlo su esempi futuri: concetto di *blind test*. In tal caso, l'errore del test set utilizzato fornisce una valutazione eccessivamente ottimistica del vero errore di test.

Regola d'oro: mantenere la separazione tra gli obiettivi e utilizzare set separati (TR per il training, VL per il model selection, TS per la stima del rischio).

Un meta-algoritmo semplice

Set separati TR (training), VL (convalida) e TS (test).

Si cerca la **migliore** $h_{w,\lambda}()$ cambiando gli iperparametri del modello λ [es. l'ordine del polinomio, lambda per la ridge regression] e per ogni valore diverso di λ (griglia di ricerca) si cercano i migliori $h_{w,\lambda}()$ che minimizzano l'errore/la perdita empirica (adattando l'insieme TR) trovando i **migliori** parametri w , dove migliore = errore minimo sull'insieme TR [$\operatorname{argmin}_w \text{Loss}(w)$ in L_2]. Si seleziona il migliore $h_{w,\lambda}()$, migliore = errore minimo sull'insieme VL. (Opzionale: ora è anche possibile adattare $h_{w,\lambda}(x)$ su TR + VL con il miglior modello λ .)

Infine si valuta $h_{w,\lambda}(x)$ finale su TS.

Hold out e K-fold cross validation

Si divide il data set D in k sottoinsiemi mutualmente esclusivi D_1, D_2, \dots, D_k , si addestra l'algoritmo di apprendimento su $D \setminus D_i$ e si testa su D_i . Infine si riassume il tutto calcolando la media di tutti i risultati di D_i .

Utilizza tutti i dati per l'addestramento, la convalida o il test.

Nota: questa tecnica può essere utilizzata sia per il validation set che per il test set.

13.1 Statistical Learning Theory (SLT)

Si vuole approssimare $f(x)$ con un certo valore desiderato d che è il target ($d = \text{true } f + \text{noise}$), riducendo al minimo la *funzione di rischio*:

$$R = \int L(d, h(x))dP(x, d)$$

Dato

- un valore dall'insegnante (d) e una distribuzione di probabilità $P(x, d)$;
- una funzione di loss (o costo), ad es. $L(h(x), d) = (d - h(x))^2$;

si cerca $h \in H$ che minimizzi R avendo solo l'insieme di dati finito $TR = (x_p, d_p)$, $p = 1 \dots l$.

Quindi si cerca h che minimizzi il rischio empirico (**errore di training E**), trovando i valori migliori per i parametri liberi del modello

$$R_{emp} = \frac{1}{l} \sum_{p=1}^l (d_p - h(x_p))^2$$

Principio induttivo di minimizzazione del rischio empirico (ERM).

Possiamo usare R_{emp} per approssimare R ?

Dato *VC-dim (VC)*, una misura di *complessità* di H (*flessibilità per adattare i dati*), ad esempio il numero di parametri per modelli lineari/polinomi.

13.1.1 Vapnik-Chervonenkis-dim e SLT

VC-bounds nella forma: con probabilità $1 - \delta$ vale che

$$R \leq R_{emp} + \varepsilon \left(\frac{1}{l}, VC, \frac{1}{\delta} \right)$$

Prima spiegazione (di base):

- ε è una funzione direttamente proporzionale a *VC* (*VC-dim*), inversamente proporzionale a l e δ .
- Sappiamo che R_{emp} diminuisce utilizzando modelli complessi (con *VC-dim* elevato).

- δ è la confidenza, regola la probabilità che il limite sia valido (es. δ basso 0,01, vale con probabilità 0,99).

Ora possiamo vedere come si può “spiegare” l’*underfitting* e l’*overfitting* e gli aspetti che li controllano:

- Maggiore l (dati) \rightarrow *VC-confidence* (ε) diminuisce e il limite tende a R .
- Un modello troppo semplice (VC-dim basso) può non essere sufficiente a causa dell’alto R_{emp} (*underfitting*).
- VC-dim più alto (fix l) $\rightarrow R_{emp}$ inferiore ma *VC-confidence*, e quindi R , può aumentare (*overfitting*).

Lo Statistical Learning Theory (SLT) permette l’inquadramento formale del problema della generalizzazione e dell’underfitting/overfitting, fornendo limitazioni superiori analitiche e quantitative al rischio R di predizione su tutti i dati, indipendentemente dal tipo di learning algorithm o dai dettagli del modello.

Il ML è ben fondato: il rischio del learning (e l’errore di generalizzazione) può essere analiticamente limitato e solo pochi concetti sono fondamentali. Si può trovare un buona approssimazione della f target da esempi, pur di avere un buon numero di dati e un’adeguata complessità del modello (misurabile formalmente con la VC-dim).

L’SLT porta a nuovi modelli (SVM) (e altri metodi che direttamente considerino il controllo della complessità nella costruzione del modello) e fonda uno dei principi induttivi sul controllo della complessità.

Capitolo 14

Introduzione all'uso di SVM

Un classificatore derivato dalla *statistical learning theory* di Vapnik e altri. Dopo anni di sviluppi teorici, SVM è diventato famoso quando, utilizzando immagini come input, ha fornito una precisione paragonabile alle reti neurali allo stato dell'arte (negli anni '90), con funzionalità progettate a mano in un'attività di riconoscimento della grafia.

Attualmente, SVM è ampiamente utilizzato in tutti i campi di applicazione dell'apprendimento supervisionato; viene utilizzato anche per la regressione.

Gli obiettivi di questa introduzione SVM in IIA sono:

1. Controllo della complessità del modello tramite un approccio di ottimizzazione per approssimare direttamente la minimizzazione del rischio strutturale: **classificatore a massimo margine**.
2. Utilizzare un'espansione di base lineare in modo efficiente tramite il kernel, così da ottenere un altro approccio flessibile per l'apprendimento supervisionato *non lineare*.
3. Evitare interpretazioni errate tipiche nell'uso di SVM, tali sono convinzioni eccessivamente ottimistiche sull'overfitting e sull'evitamento del corso di dimensionamento.

14.1 Classificatore a massimo margine

Problema di classificazione binaria. Cominciamo assumendo un problema linearmente separabile e assumendo anche nessun dato di rumore.

Non tutti gli iperpiani che risolvono il compito sono uguali... Variando l'iperpiano di separazione, il margine varia altrettanto.

Definizione 14.1.1. Il **margine** è il doppio della distanza tra l'iperpiano e il punto più vicino.

Intuitivamente si cerca la distanza massima dai punti dati per massimizzare la zona “safe”.

Definizione 14.1.2.

$$\text{Support Vectors : } \mathbf{x}_p \text{ tale che } |\mathbf{w}^T \mathbf{x}_p + b| = 1 (b = w_0)$$

Tutti i punti sono correttamente classificati se $\forall i (w^T x_i + b) y_i \geq 1$

Consideriamo il problema di imparare un *modello lineare* per la classificazione binaria, cioè una funzione $h : \mathbb{R}^n \rightarrow \{-1, 1\}$

$$h(x) = \text{sign}(wx + b)$$

basata su esempi (x_p, y_p)

Training problem: trovare (w, b) in modo tale che tutti i punti siano classificati correttamente e il *margine sia massimizzato*.

Due fatti utili

$$\text{Margine} = \frac{2}{|w|} \quad [|w|^2 = (w^T w)]$$

$$\Rightarrow \text{massimizza il margine} \Leftrightarrow \text{minimizza } |w| \Leftrightarrow \text{minimizza } \frac{|w|^2}{2}$$

VC-dim dell'SVM è inverso al margine \rightarrow controllo della complessità del modello dal margine.

L'*iperpiano ottimale* è l'iperpiano che massimizza il margine (e risolve il training problem).

14.1.1 Hard Margin SVM

Training problem (forma primale):

$$\min \left[\frac{|w|^2}{2} \right], \text{ (i.e. } w^T w \text{) tale che } (wx_p + b)y_p \geq 1 \quad p = 1 \dots l$$

Nota: minimizzazione diretta della complessità del modello (funzione di ottimizzazione), tenendo la soluzione (0 errori) nei *vincoli*. La funzione obiettivo è convessa in w .

Ovviamente, se esiste una forma primale, esiste anche un forma duale. Supponendo di aver trovato la soluzione del problema duale, possiamo calcolare (w, b) :

$$w = \sum_{p=1}^l \alpha_p y_p x_p \quad b = y_k - w^T x_k, \quad \forall \alpha_k > 0$$

$$h(x) = \text{sign}(w^T x + b) = \text{sign} \left(\sum_{p=1}^l \alpha_p y_p x_p^T x + b \right) = \text{sign} \left(\sum_{p \in SV} \alpha_p y_p x_p^T x + b \right)$$

1. $\alpha_p \neq 0 \leftrightarrow$ vettori di supporto ($\alpha_p \neq 0 \rightarrow x_p$ è un vettore di supporto). La soluzione è sparsa e formulata solo in termini di SV. *L'iperpiano dipende solo dai vettori di supporto.*
2. Una forma speciale della soluzione: non è nemmeno necessario calcolare (w, b) esplicitamente per classificare i punti.

Nota: i dati vengono inseriti sotto forma di prodotti puntiformi di coppie di punti.

14.1.2 Soft Margin

L'hard margin per tutti i punti può essere troppo restrittivo; alcuni errori possono essere consentiti per la *tolleranza al rumore* e per *fornire un margine più ampio*.

Soluzione: consentire errori che introducono **slack-variables**.

Le *slack-variables* ξ_i possono essere aggiunte per consentire una classificazione errata di punti dati difficili o rumorosi.

Training problem (forma primale):

$$\min \left[\frac{|w|^2}{2} + C \cdot \sum_i \xi_i \right] \text{ tale che } (wx_p + b)y_p \geq 1 - \xi_p, \quad \forall p \quad \xi_p \geq 0$$

ξ_p positivo indica un errore o un margine troppo piccolo, $C > 0$ guida il numero di errori consentiti (gli indici di ξ_p calcolati dal risolutore).

C è un iperparametro definito dall'utente. Se C è piccolo allora sono consentiti molti errori di training, possibile underfitting. Se C è grande allora non sono ammessi errori di training (piccolo margine), possibile overfitting.

14.2 Kernel

Mapping to a High-Dimensional Space

Mappare i punti dati nello spazio di input su uno spazio delle features ad alta dimensione (cosiddetto in SVM), dove possono essere separabili linearmente.

Si può usare la *Linear Basic Expansion* $\Phi(x)$ invece di x ; tuttavia, l'uso di spazi di features ad alta dimensione (espansione di funzioni di base di grandi dimensioni) può essere computazionalmente irrealizzabile e, cosa più importante, può **facilmente** portare all'*overfitting* senza controllare la dimensione dello spazio delle features e la *complessità* del classificatore: la complessità è qui correlata alla dimensionalità dell'input, numero di parametri liberi sull'equazione

$$h_w(x) = \text{sign} \left(\sum_k w_k \phi_k(x) \right)$$

Si propone l'approccio Kernel per gestire (*implicitamente*) lo spazio delle features nel contesto del modellamento regolarizzato (dove la complessità dipende dal margine); pertanto, grazie alla regolarizzazione automatizzata di SVM, *la complessità del classificatore può essere mantenuta piccola* indipendentemente dalla dimensionalità nel nuovo spazio delle features.

In SVM non è necessario calcolare w e i dati inseriti sotto forma di prodotti puntiformi di coppie di punti.

$$h_w(x) = \text{sign} \left(\sum_{p \in SV} \alpha_p y_p x_p^T x + b \right)$$

$$h_w(x) = \text{sign} \left(\sum_k w_k \phi_k(x) \right)$$

$$h_w(x) = \text{sign} \left(\sum_{p \in SV} \alpha_p y_p \phi^T(x_p) \phi(x) \right)$$

e non è nemmeno necessario calcolare direttamente ϕ

$$h(x) = \text{sign} \left(\sum_{p \in SV} \alpha_p y_p K(x_p, x) \right)$$

cioè possiamo *gestire implicitamente* lo spazio delle features tramite una **funzione del kernel**.

Definizione 14.2.1. Un **kernel** $k : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ è una funzione tale che esistono uno spazio di Hilbert X (possibilmente ad alta dimensione) e una funzione $\Phi : \mathbb{R}^n \rightarrow X$ con

$$K(x_i, x_j) = \Phi^T(x_i)\Phi(x_j)$$

cioè un kernel è una potenziale scorciatoia per calcolare il prodotto scalare in modo efficiente anche in spazi ad alta dimensione.

Kernel notevoli

Lineare:

$$K(x_i, x_j) = x_i^T x_j$$

Mapping $\Phi : x \rightarrow \phi(x)$, dove $\phi(x)$ è x stessa.

Polinomiale di potenza p :

$$K(x_i, x_j) = (1 + x_i^T x_j)^p$$

Mapping $\Phi : x \rightarrow \phi(x)$, dove $\phi(x)$ ha dimensione esponenziale in p .

RBF (funzione a base radiale) Gaussiana:

$$K(x_i, x_j) = e^{\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)}$$

Mapping $\Phi : x \rightarrow \phi(x)$, dove $\phi(x)$ è *infinita-dimensionale*.

RBF è una scelta molto popolare; si noti che ha un iperparametro (σ). Può essere molto potente sulla discriminazione TR ma ovviamente incline all'**overfitting**.

La progettazione di un nuovo kernel per tipi speciali di dati è un argomento di ricerca corrente.

- Si sceglie il parametro di trade-off C e la funzione kernel K (e i suoi parametri).
- Si risolve il problema di ottimizzazione per trovare α :
 - Il costo computazionale scala con l (numero di dati) invece di n (dimensione dello spazio delle features).
 - Modularità: cambia solo il kernel (con lo stesso risolutore).
- Il modello finale

$$h(x) = \text{sign} \left(\sum_{p \in SV} \alpha_p y_p K(x_p, x) \right)$$

14.2.1 Evitare interpretazioni errate

- L'overfitting può verificarsi senza un'attenta selezione degli iperparametri SVM: C , kernel, parametri del kernel, ...
- Il trattamento隐式的 dello spazio High dim è nel *feature space* e non nell'input space (supponendo che vi proiettiamo input significativi).
- Tecnica di convalida vista finora per la selezione del modello (es. C , kernel e gli iperparametri del kernel) e anche la valutazione del modello dovrebbe essere usata rigorosamente.

Conclusioni

SVM è uno strumento avanzato, molto utile e molto noto nel Machine Learning; generalmente le prestazioni sono molto buone, ma *non necessariamente* le migliori in confronto ad altri metodi (come le reti neurali e altri).

Provenire dalla teoria (SRM) è un buon modo per fornire nuovi approcci ML. Combinare in modo efficiente l'*espansione lineare* di base tramite kernel all'interno di un approccio **max margin** consente di combinare modelli flessibili e il controllo della complessità.

La modularità del kernel apre nuove possibilità.

Capitolo 15

K-NN, apprendimento senza supervisione e altri approcci

15.1 K-Nearest Neighbors

Si tratta di un modello supervisionato. Memorizza semplicemente i dati di allenamento $\langle x_p, y_p \rangle$ $p = 1 \dots l$. Dato un input x (con dimensione n) trova l'esempio di training più vicino x_i , cioè trova i t.c. abbiamo

$$\min d(x, x_i) \rightarrow i(x) = \arg \min_p d(x, x_p)$$

Per esempio la distanza euclidea:

$$d(x, x_p) = \sqrt{\sum_{t=1}^n (x_t - x_{p,t})^2} = \|x - x_p\|$$

Quindi ritorna y_i .

Un modo naturale per classificare un nuovo punto è dare un'occhiata ai suoi vicini e fare una media:

$$\text{avg}_k(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

dove $N_k(x)$ è un intorno di x che contiene esattamente k vicini (pattern più vicini secondo d): k-nearest neighborhood (**K-nn**).

Se c'è una chiara dominanza di una delle classi nelle vicinanze di un'osservazione x , allora è probabile che anche l'osservazione stessa appartenga a quella classe. Quindi la regola di classificazione è il **voto a maggioranza**

tra i membri di $N_k(x)$. Come prima,

$$h(x) = \begin{cases} 1 & \text{se } avg_k(x) > 0.5 \\ 0 & \text{altrimenti} \end{cases} \quad \text{per } y_i = \{0, 1\} \text{ (targets)}$$

Per l'attività di regressione utilizzare direttamente *avg*: media su K-nn.

Ancora una volta c'è un compromesso tra underfitting e overfitting dai possibili valori di K. Classica curva a forma di U dell'errore di test che si sposta tra a

- caso estremamente flessibile ($K = 1$)
- fino a un modello molto rigido ($K = l$): 1 media per tutti i dati

Nearest Neighbor non fa nessuna ipotesi globale (per tutte le istanze), non c'è nessun modello che va a fare un fitting: non c'è un modello parametrico, è in contrasto al modello lineare che era parametrico con un insieme fissato di parametri w . Fa delle stime locali (mediante funzioni costanti locali), rispetto a un'approssimazione/stima lineare globale della funzione target (nello spazio dell'istanza).

È un metodo pigro, basato sulla memoria sull'istanza e sulla **distanza**.

15.1.1 Limitazioni

Costo computazionale

Si noti che K-NN effettua l'approssimazione locale alla funzione target per ogni nuovo esempio da prevedere: il costo computazionale è rimandato alla **fase di previsione!** Inoltre c'è un alto costo di recupero:

- Computazionalmente intensivo per ogni nuovo input: calcolo delle distanze dal campione di prova a tutti i vettori memorizzati e il tempo è proporzionale al numero di modelli memorizzati, anche se possono essere usati algoritmi di ricerca di prossimità “ad-hoc” da ottimizzare.
- Costo in spazio (tutti i dati vengono memorizzati).

Curse of dimensionality

Quando abbiamo molte variabili di input (n grande), i metodi K-NN spesso falliscono a causa della **curse of dimensionality**: quando la dimensionalità d aumenta, il volume dello spazio aumenta così velocemente che i dati

disponibili diventano scarsi, ovvero la quantità di dati necessari a supportare il risultato spesso cresce *esponenzialmente* con la dimensionalità.

Features irrilevanti (**Curse of Noisy**): se il target dipende solo da poche delle molte features in x , si potrebbe recuperare un “modello simile” con la somiglianza dominata dal gran numero di features irrilevanti.

Metodi basati sulla distanza: considerazione e bias induttivo

Approcci basati sulla **distanza** o sulla **metrica**: linea comune tra ad es. K-means, K-NN e approcci basati sul kernel; cosa c’è di **simile**? Misura quando un paio di pattern sono *simili*.

Dare una metrica (ad esempio la metrica euclidea è stata assunta fino ad ora) pone un **bias** rilevante sulla soluzione (molto più dei dettagli dell’algoritmo). La metrica dipende in genere dal dominio (per esempio due stringhe in una lingua, in biologia, . . .), è basata sulla conoscenza preliminare dell’esperto del dominio e in qualche modo è la stessa cosa dell’*ingegnerizzazione delle features* per rappresentare il problema.

15.2 Unsupervised learning

Clustering

Partizione dei dati in cluster (sottoinsiemi di dati “simili”): i modelli all’interno di un cluster valido sono più simili tra loro di quanto non lo siano a un modello appartenente a un cluster diverso. Inoltre è possibile individuare dei punti particolari chiamati centroidi (o prototipi), ossia i “rappresentanti”.

Spazio delle ipotesi per il clustering

Obiettivo: partizionamento ottimale della distribuzione sconosciuta nello spazio x in regioni (cluster) approssimate da un centro o *prototipo* del cluster. $H : x \rightarrow c(x)$, dove x è un insieme di vettori di quantizzazione e $c(x)$ è il vettore che rappresenta il cluster.

$$\text{spazio continuo} \rightarrow \text{spazio discreto}$$

Un esempio di funzione di loss è una funzione che misura l’ottimalità del vettore di quantizzazione: una **funzione di loss comune sarebbe la distorsione dell’errore al quadrato**:

$$L(h(x_p)) = \|x_p - c(x_p)\|^2$$

Il valore medio sulla distribuzione degli input è l’errore medio di distorsione o ricostruzione o quantizzazione.

15.2.1 K-means

Il k -means è l'algoritmo più semplice e più comunemente usato che impiega un **criterio di errore quadrato**: è popolare perché è facile da implementare e, in generale, efficiente; tuttavia, ha diversi inconvenienti e **limitazioni** che vedremo dopo la presentazione dell'algoritmo.

1. Sceglie k centri del cluster in modo che coincidano con k pattern scelti a caso o k punti definiti in modo casuale all'interno dell'ipervolume contenente l'insieme di pattern.
2. Assegna ogni modello al centro del cluster più vicino (il vincitore).
3. Ricalcola i centri del cluster (centroide geometrico, cioè la media) utilizzando le attuali appartenenze al cluster.
4. Se un criterio di convergenza non è soddisfatto, si ritorna al punto 2.

Dati i centri del cluster c_1, \dots, c_k , per ogni x il vincitore è (il prototipo più vicino):

$$i^*(x) = \arg \min_i \|x - c_i\|^2$$

Adesso x appartiene al cluster i^* . Per ogni cluster i la nuova media (centroide) è:

$$c_i = \frac{1}{|cluster_i|} \sum_{j:x_j \in cluster_i} x_j$$

Limitazioni

- Il numero dei cluster deve essere fornito.
- I minimi locali della loss $L(x)$ rendono il metodo dipendente dall'inizializzazione: è necessario eseguire più volte da diverse inizializzazioni casuali oppure inizializzare con un'euristica.
- Può funzionare bene per cluster compatti e ipersferici.
- Nessuna proprietà di visualizzazione: k -means non consente di proiettare i dati in uno spazio più debole.

15.3 Altri approcci per il preprocessing

Tra la moltitudine di approcci che meritano di essere menzionati in questo corso:

- **Riduzione della dimensionalità** (nel learning non supervisionato)

$$\langle x_1, x_2, \dots, x_n \rangle \rightarrow \langle x'_1, x'_2, \dots, x'_n \rangle \quad n > n'$$

dove le nuove features possono essere una combinazione di quelle originali.

- **Selezione delle features**: scegliamo un sottoinsieme di tutte le funzionalità in base alla conoscenza del dominio (*features engineering*) oppure *automaticamente* in base alla loro ridondanza o rilevanza nell'attività (la più informativa). Molti approcci: il problema è difficile come quello dell'apprendimento.
- **Rilevamento dei valori anomali**: trova valori di dati insoliti che non sono coerenti con la maggior parte delle osservazioni (ad esempio a causa di errori di misurazione anormali).

15.4 Altri task

- **Reinforcement Learning** (apprendimento con critica positiva/negativa), viene usato nell'adattamento dei sistemi autonomi (soprattutto nella robotica): “l'algoritmo apprende una politica di come agire data un'osservazione del mondo; ogni **azione** ha un certo impatto sull'ambiente e l'ambiente fornisce un feedback che guida l'algoritmo di apprendimento”. Invece della supervisione per ogni passaggio, abbiamo informazioni su vittorie/sconfitte (premi o punizioni) per lo stato finale; le azioni devono massimizzare la quantità di ricompense ricevute. L'apprendimento decide quali azioni sono state maggiormente responsabili di vittorie/sconfitte
- **Apprendimento semi-supervisionato**: combina esempi etichettati e non etichettati (in genere in numero maggiore) per generare una funzione o un classificatore appropriato.
- **Learn to rank**: (ad es. per i motori di ricerca) quando l'input è un insieme di oggetti e l'output desiderato è un ranking di quegli oggetti.
- **On-line learning**: nuovi esempi vengono appresi nel tempo.

- **Apprendimento del dominio strutturato e apprendimento relazionale:** il dominio di input e/o output può essere strutturato sotto forma di sequenze (segnali, serie temporali, ...) o anche qualcosa di più complesso come alberi, grafi e reti.

15.5 Altri modelli

Reti neurali: per l'apprendimento sia supervisionato che non supervisionato. Sono vicine alla nostra visione di Linear Threshold Unit (LTU), in effetti la LTU è vicina all'unità di base di una rete neurale artificiale, il *perceptron* e una rete neurale è una *rete* di tali unità *non lineari* con capacità di approssimazione universale.

Approccio alla discesa in gradiente per l'apprendimento (backpropagation).

.Lo strato interno (nascosto) con unità non lineari fornisce a NN la capacità di estrarre apprendendo una nuova rappresentazione dei dati (learning abstract features); le nuove rappresentazioni semplificano l'attività di classificazione all'ultimo strato. È una **basis expansion** non lineare in w adattativa in cui **vengono appresi** i ϕ (dipendono da w), ma sfortunatamente porta anche a un problema di ottimizzazione non lineare. Usa una rappresentazione distribuita (contro simbolica): la somiglianza può essere trattata più facilmente da vettori di valori reali; approccio molto flessibile per attività non lineari, sia per attività di classificazione che di regressione (capacità di approssimazione universale).

Aspetti comuni del **Deep Learning** (DL) tra diversi approcci:

- *Più strati* di unità di elaborazione non lineari
- Apprendimento supervisionato o non supervisionato delle *rappresentazioni delle features* in ogni livello, con i livelli che formano una gerarchia dalle caratteristiche/rappresentazioni di livello basso a quelle di alto livello (*diversi livelli di astrazione*).