Appunti del corso

Elementi di Intelligenza Artificiale

1. L'Intelligenza Artificiale: concetti di base

Vogliamo che i computer svolgano attività che richiedono intelligenza umana

Calcoli complessi, che richiedono intelligenza a un essere umano, sono svolti rapidamente

Ci sono compiti che vengono svolti senza accorgersene (es. riconoscere un volto) che sono difficilissimi da automatizzare

L'Intelligenza artificiale (AI) si occupa di queste attività che richiedono complessi ragionamenti

Perché automatizzare l'intelligenza umana ?

a) capire come funziona l'intelligenza umana
b) avere programmi e macchine più intelligenti

Creare un robot human-like è un obiettivo forse possibile, forse non desiderabile

Per il momento AI cerca di costruire programmi più intelligenti e giungere a una migliore comprensione di come ragioniamo

AI ha delle naturali intersezioni con molte discipline quali, ad esempio, psicologia, filosofia, linguistica

- Tali discipline ci danno informazioni su come noi agiamo e comunichiamo intelligentemente

- grazie a queste informazioni costruiamo sistemi che svolgono compiti che richiedono intelligenza

- in tal modo facciamo luce sull'intelligenza umana fornendo nuove informazioni alle discipline

Problemi AI tipici

Andare a fare acquisti in centro:

- decidere cosa comprare
- come arrivare in centro
- quali negozi visitare
- muoversi nei negozi affollati
- interpretare ciò che vediamo
- comunicare con le persone che incontriamo

Questo è un esempio di problema *quotidiano*: sono problemi che risolviamo quasi senza accorgercene ma che richiedono ragionamenti molto complessi da automatizzare.

I problemi quotidiani corrispondono alle seguenti aree AI:

- *pianificazione*: decidere la sequenza corretta di azioni per raggiungere un obiettivo
- *visione*: la capacità di dare un senso a ciò che viene visto
- *robotica*: la capacità di muoversi e agire nel mondo, rispondendo a nuove percezioni
- *linguaggio naturale*: la capacità di comunicare con altri utilizzando il linguaggio naturale (es. italiano, inglese)

Vi è poi un'altra classe di problemi che richiedono abilità specialistiche e sono detti problemi *esperti*, quali, ad esempio:

- diagnosi mediche
- riparazione di attrezzature
- configurazione di computer
- pianificazione finanziaria

Automatizzare queste attività può essere molto utile, in quanto si risparmia l'intervento di un esperto umano. I cosiddetti *sistemi esperti* si occupano di automatizzare questa classe di problemi.

AI tenta di automatizzare entrambe le classi di problemi: viene fuori che i problemi quotidiani sono molto più complessi di quelli esperti.

Tecniche AI

Le tecniche AI si occupano di come noi rappresentiamo, manipoliamo e ragioniamo sulla conoscenza al fine di risolvere problemi.

Rappresentazione della conoscenza

Per ragionare con la conoscenza occorre innanzitutto rappresentarla in modo formale.

Risolvere problemi apparentemente semplici come capire il significato di una frase richiede una conoscenza approfondita sia del linguaggio sia del contesto.

La conoscenza va rappresentata in modo *efficiente* e *significativo*.

L'efficienza è importante in quanto è praticamente impossibile rappresentare esplicitamente ogni fatto di cui si può aver bisogno. Bisogna essere in grado di desumere, quando necessario, nuovi fatti a partire dalla conoscenza disponibile.

Per rappresentare la conoscenza in modo significativo è importante poter correlare fatti in uno schema di rappresentazione formale con fatti nel mondo reale.

La rappresentazione formale sarà manipolata dal sistema e nuovi fatti verranno desunti, per cui è essenziale che tali conclusioni rappresentate formalmente possano essere interpretate in termini del nostro problema di partenza.

La *semantica* di un linguaggio di rappresentazione fornisce un modo per far corrispondere il mondo reale con espressioni in un linguaggio formale.

Tecniche di ricerca

Un'altra tecnica essenziale quando si scrivono programmi AI è la *ricerca*. Spesso non vi è nessun modo diretto per risolvere un problema però si sa come generare possibili soluzioni.

Ad esempio, se voglio andare rapidamente in una certa località, partendo dalla conoscenza di quali sono le strade e i mezzi di trasporto, mi metto alla ricerca del percorso migliore fra i tanti possibili.

Essendo spesso enorme il numero di possibilità è essenziale sviluppare algoritmi che consentano, per quanto possibile, di individuare una buona soluzione rapidamente.

Tecniche basate sulla *forza bruta*, in cui si genera ogni possibile soluzione e la si verifica possono funzionare ma sono spesso estremamente inefficienti (complessità esponenziale) in quanto vi sono troppe possibili soluzioni da verificare.

Le cosiddette tecniche *euristiche* sono spesso migliori, in quanto la ricerca viene limitata alle sole opzioni, scelte in base a un certo criterio, che vengono ritenute più adatte per giungere alla soluzione.

Prospettive dell'AI

Trenta anni fa c'erano molte aspettative sui possibili successi dell'AI. La ricerca in quest'area ha raffreddato molti entusiasmi ma nel contempo ha rivelato nella sua pienezza la complessità dei problemi da affrontare.

Attualmente gli obiettivi che ci si pone sono più limitati e i risultati soddisfacenti: ad esempio, i sistemi esperti vengono utilizzati con successo, sistemi "intelligenti" di controllo vengono utilizzati nella vita quotidiana (ad esempio, fotocamere, lavatrici), si parla diffusamente di *agenti intelligenti*, sistemi per la comprensione limitata della voce sono abbastanza diffusi, il robot ASIMO della Honda sale le scale, algoritmi genetici e reti neurali hanno trovato molte applicazioni.

Aspetti fondazionali

La ricerca AI assume che l'intelligenza umana può essere ridotta alla manipolazione sofisticata di simboli e che non interessa qual è il mezzo utilizzato per manipolare tali simboli.

Il test di Turing

Turing si chiese come decidere se una macchina è intelligente o meno. La sua opinione è che solo un test poteva dare una risposta.

Ipotizzò un test in cui un essere umano comunica a distanza con un altro essere umano e con un computer. Il primo umano può formulare qualsiasi domanda al secondo umano e al computer. Se al termine delle domande egli non è in grado di dire quale dei due è umano, allora il computer è intelligente.

La "stanza cinese" di Searle

Searle obiettò che *comportarsi* intelligentemente non è sufficiente e suggerì un ipotetico esperimento detto "Chinese room".

Immaginiamo che una persona ignori completamente il cinese ma abbia a disposizione un enorme libro di regole che permette di cercare frasi in cinese con le corrispondenti risposte in cinese. Avendo questo libro immenso ci si può comportare in modo apparentemente intelligente. Infatti la persona riceve una domanda scritta, cerca nel libro la domanda, la trova, a fianco c'è la corrispondente risposta, la dà in output, riceve un'altra domanda e si va avanti così.

Finora alcuni programmi hanno superato una versione ristretta del test di Turing con domande limitate a una piccola area.

Il sistema ELIZA

Weizenbaum sviluppò il programma ELIZA che imitava il comportamento di uno psicanalista.

Il programma usava vari trucchi per formulare frasi ragionevoli. Ad esempio:

Paziente: I am tired "I am X" → "how long have you been X ?"

ELIZA: How long have you been tired ?

P: Six months ELIZA non trova appigli e pertanto dice:

E: Please go on

P: I feel sad "I feel X" → "Do you often feel that way ?"

E: Do you often feel that way ?

P: Always, when I'm with my mother ELIZA riconosce una parola "importante" e costruisce una domanda con questa parola

E: Tell me more about your mother

P: You won't trust me
"You X me" → "What makes you think I X you ?

E: What makes you think I won't trust you ?

P: Nobody ever does ELIZA non avendo meglio da fare tira fuori la mamma

E: Earlier you talked of your mother

2. Rappresentazione della conoscenza e regole di inferenza

Una delle assunzioni dell'AI è che un comportamento intelligente può essere ottenuto attraverso la manipolazione di *strutture simboliche* rappresentanti elementi di conoscenza.

Ad esempio, potremmo usare il simbolo red per denotare un particolare colore, il simbolo alisons-car per denotare un'auto, e la struttura simbolica red(alisons-car) per denotare che l'auto di Alison è rossa.

Nel computer possiamo rappresentare fatti usando strutture dati e scrivere programmi per ragionare con essi.

I *linguaggi per la rappresentazione della conoscenza* (KRL = Knowledge Representation Languages) sono stati creati per rendere più facile questa attività. Invece di usare le strutture dati di un linguaggio come il C++ e il Pascal noi utilizziamo formalismi ad alto livello specifici.

I KRL possono a loro volta essere realizzati mediante linguaggi procedurali usuali, per cui, ad esempio, un fatto come red(alisons-car) può essere rappresentato mediante strutture dati convenzionali, il fatto importante è che grazie ai KRL il programmatore AI può utilizzare direttamente formalismi più adatti alla sua attività.

inferenza: s.f. operazione mentale per cui si passa da un giudizio o da un fatto noto ad un altro giudizio.

Requisiti per i KRL

Un KRL dovrebbe permettere di rappresentare *fatti complessi* in un modo chiaro, preciso e naturale e in modo da poter *dedurre nuovi fatti* a partire dalla conoscenza esistente.

La capacità di rappresentare adeguatamente fatti complessi è denotata come l'*adeguatezza rappresentativa* di un KRL. Alcuni fatti sono difficili da rappresentare in modo tale da poterci ragionare sopra. Ad esempio, un semplice fatto come "John believes no-one likes cauliflowers" può essere rappresentato molto chiaramente usando la lingua inglese.

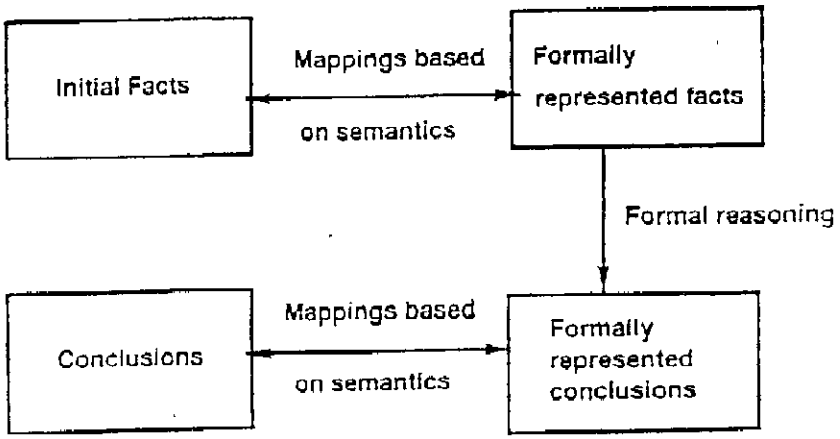
Il problema è che questa rappresentazione non si presta ad effettuare deduzioni così da poter ottenere, ad esempio, il nuovo fatto "John believes Mary doesn't like cauliflowers".

Il requisito di rappresentare la conoscenza in modo chiaro e preciso implica che dobbiamo avere una *sintassi e una semantica ben definite.* In sostanza dobbiamo sapere quali sono le espressioni ammissibili nel linguaggio e qual è il loro significato.

Supponiamo che abbiamo definito a cosa si riferisce ogni simbolo (ad esempio, red si riferisce alla proprietà di essere di colore rosso, alisons-car si riferisce all'auto di Alison. La *sintassi* del linguaggio definisce le strutture ammissibili del linguaggio (e.g. red(alisons-car) va bene, alisons-car (grey & green) no).

La *semantica* del linguaggio associa un significato a una particolare struttura (e.g. red(alisons-car) significa che l'auto di Alison è rossa, e non che l'auto va dipinta di rosso).

Per risolvere un problema AI prima dobbiamo rappresentare la conoscenza del mondo reale utilizzando un KRL. Poi il programma AI lavora deducendo nuovi fatti e giungendo a delle conclusioni. E' chiaramente essenziale che noi possiamo interpretare cosa significano queste conclusioni in termini del nostro problema del mondo reale. Se il sistema conclude interest(alison, high) dobbiamo sapere che questo si riferisce alla personalità di Alison e non al suo mutuo.



Un altro requisito del KRL è che esso sia ragionevolmente *naturale*, catturando la struttura della conoscenza in modo facilmente comprensibile.

Inoltre, è anche importante scegliere *nomi* significativi per i simboli. Potremmo rappresentare il fatto "if someone has a headache they should take aspirin" con if(x, h, a) ma un modo più leggibile e manipolabile è qualcosa del tipo:

if symptom (X, headache) then medication (X, aspirin)

La capacità di un KRL di dedurre nuovi fatti dalla conoscenza esistente è denotata come *adeguatezza inferenziale.*

Non è ragionevole rappresentare esplicitamente qualsiasi fatto di cui il sistema potrebbe avere bisogno, alcune cose devono essere lasciate implicite, consentendo al sistema di dedurle se e quando ne ha bisogno.

Ad esempio, supponiamo che un sistema ha bisogno della conoscenza su cento studenti. Supponiamo che tutti seguono le lezioni e sostengono esami. Sarebbe antieconomico memorizzare questi fatti uno a uno per i cento studenti.

E' molto meglio memorizzare che "tutti gli studenti seguono le lezioni e sostengono esami". Nel momento in cui il sistema ha bisogno di sapere se lo studente Fred frequenta le lezioni, il sistema può dedurlo dalla frase generale e dal fatto che Fred è uno studente.

Osserviamo che più le deduzioni sono complesse, maggiore è il tempo necessario. Bisogna trovare un compromesso tra l'*adeguatezza inferenziale* (ciò che possiamo dedurre) e l'*efficienza inferenziale* (la velocità delle inferenze).

In sostanza, i requisiti di un buon KRL sono:

Adeguatezza rappresentativa: il linguaggio deve essere in grado di rappresentare tutta la conoscenza di cui abbiamo bisogno per lavorare

Adeguatezza inferenziale: nuova conoscenza può essere dedotta a partire da quella disponibile

Efficienza inferenziale: le inferenze devono essere ragionevolmente rapide

Sintassi e semantica chiare: dobbiamo sapere quali sono le espressioni ammesse e il loro significato

Naturalezza: il linguaggio deve essere naturale e facile da utilizzare

Nessun KRL soddisfa pienamente questi requisiti. La scelta del linguaggio dipende dal tipo di deduzioni richieste, analogamente a quanto accade per i linguaggi di programmazione in cui la scelta dipende da problema.

I principali KRL

Logica

Vi sono tre approcci principali al problema della rappresentazione della conoscenza in AI. Il più importante è l'utilizzo della *logica* per rappresentare oggetti. Per esempio, nella logica dei predicati la frase "tutti gli uccelli volano" viene rappresentata con:

$\forall$ X (bird (X) $\rightarrow$ flies (X))

Una logica ha *sintassi* e *semantica* ben definite e si preoccupa di effettuare inferenze corrette, pertanto è un buon metodo per rappresentare la conoscenza e manipolarla.

Però l'utilizzo della logica per rappresentare cose fa nascere problemi. Innanzitutto la logica può risultare poco efficiente e inoltre è difficile rappresentare cose per noi abituali. Infatti rappresentare e ragionare con cose che involvono* il tempo, le credenze o l'incertezza è molto complicato.

Frames e reti semantiche

Un'alternativa è utilizzare schemi di rappresentazione più naturali, specificando gli algoritmi per manipolare la conoscenza, senza formalizzare pienamente la semantica del linguaggio.

I *frames* e le *reti semantiche* forniscono uno strumento naturale per rappresentare la conoscenza fattuale riguardante classi di oggetti e loro proprietà.

La conoscenza è rappresentata mediante una collezione di oggetti e relazioni, le relazioni di base essendo la *sottoclasse* e la *instance*.

La relazione sottoclasse dice che una classe è sottoclasse di un'altra, la relazione instance dice che un certo elemento appartiene a una specifica classe.

*involve trascinare con sé.

Pertanto, ad esempio, lo studente Fred Bloggs è un instance della classe rappresentante gli studenti AI, mentre la classe degli studenti AI è una sottoclasse della classe studenti.

Si può quindi definire la proprietà di *ereditarietà* per cui Fred *eredita* gli attributi tipici degli studenti AI, e gli studenti AI ereditano gli attributi tipici degli studenti.

Regole

Un altro importante metodo per rappresentare la conoscenza è l'uso delle *regole IF-THEN* (anche dette *regole condizione-azione)* all'interno di un *sistema basato su regole.*

Una regola condizione-azione specifica cosa fare in una certa situazione. Ad esempio, si potrebbe avere la regola: IF fire THEN shout-help

Un linguaggio basato su regole fornisce algoritmi per ragionare con queste regole, in modo da ricavare nuove conclusioni.

Le regole appaiono simili alle implicazioni logiche (ad esempio, fire -> shout-help), va però notato che nei KRL basati su regole l'enfasi è più su ciò che si può fare con le regole e meno sul loro significato. Formalmente si dice che gli aspetti *procedurali* sono enfatizzati rispetto a quelli *dichiarativi.*

Le regole condizione-azione sono largamente usate nei sistemi esperti, fornendo uno strumento flessibile per rappresentare la conoscenza dell'esperto e tecniche efficienti per ragionare con tale conoscenza.

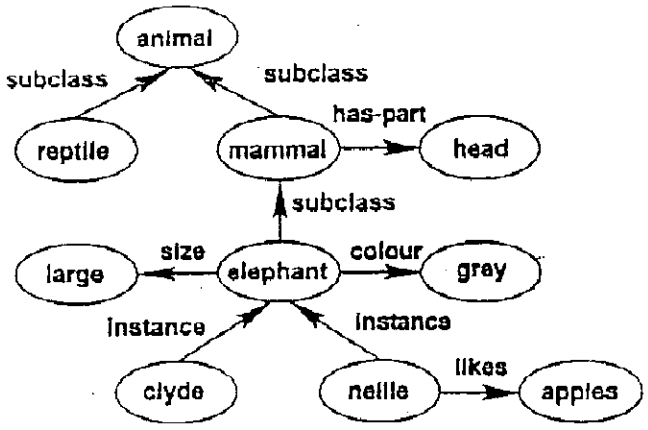
Reti semantiche e frames

Le reti semantiche e i frames offrono un modo semplice e intuitivo di rappresentare fatti circa gli oggetti. Entrambi gli schemi permettono di rappresentare classi di oggetti e relazioni tra oggetti, ed effettuare semplici inferenze basate su questa conoscenza. Vi è poca differenza tra le reti e i frames.

Reti semantiche

In una rete semantica la conoscenza è rappresentata mediante un *grafo*, dove i *nodi* rappresentano *concetti* e i *rami* rappresentano *relazioni* tra concetti.

Le relazioni più importanti tra concetti sono la relazione *sottoclasse* e la relazione *instance* tra specifici esempi di oggetti e la loro classe padre. Qualsiasi altra relazione è consentita, ad esempio has-part, colour, permettendo di rappresentare le proprietà degli oggetti.



Le relazioni di sottoclasse definiscono una *gerarchia di classi*. Le relazioni sottoclasse e instance possono essere usate per derivare nuova informazione che non è esplicitamente rappresentata. Clyde e Nellie *ereditano* proprietà dalle loro classi padre.

Quando le reti semantiche diventarono popolari negli anni 70 si discusse a lungo sul significato di nodi e relazioni.

Ad esempio, il nodo *elefante* può essere usato per rappresentare la classe di tutti gli elefanti o soltanto un elefante. Dicendo che un elefante *has-part* testa può significare che ogni elefante ha una particolare testa, oppure che qualche elefante ha una testa di qualche tipo, oppure che uno specifico elefante ha una testa particolare, e così via.

Ne consegue che se una relazione può significare più cose la rappresentazione può avere vari significati e di conseguenza le inferenze non sono chiare.

Oggigiorno si riconosce l'importanza di definire precisamente la *semantica* di un linguaggio, in modo che il significato delle espressioni e delle inferenze sia chiaro.

Un modo semplice per descrivere precisamente il significato di nodi e link è servirsi della teoria degli insiemi. Un nodo viene a denotare un insieme di oggetti, pertanto il nodo *elefante* denota l'insieme di tutti gli elefanti, Clyde e Nellie sono specifici elefanti.

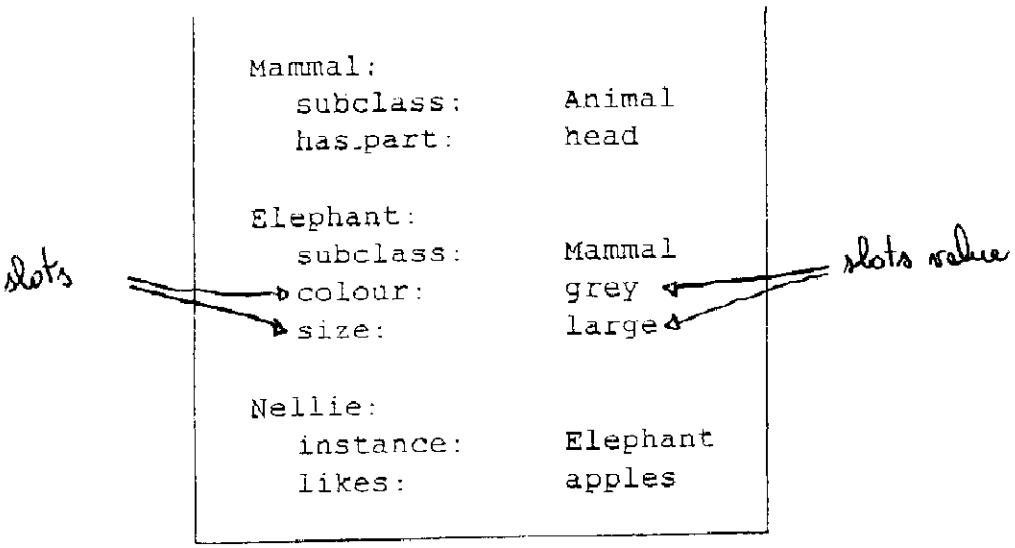
La relazione *instance* può essere definita in termini di appartenenza a un insieme (Nellie è un elemento dell'insieme di tutti gli elefanti), mentre la relazione *sottoclasse* viene definita in termini di sottoinsiemi, l'insieme di tutti gli elefanti è un sottoinsieme dell'insieme di tutti i mammiferi.

Le reti semantiche ci permettono di rappresentare la conoscenza su oggetti e relazioni tra oggetti in modo semplice e intuitivo. Le inferenze possibili si riducono all'ereditarietà delle proprietà per cui da un lato le inferenze sono effettuate molto semplicemente, dall'altro la notazione non è adatta per rappresentazioni complesse della conoscenza.

Frames

I *frames* sono una variante delle reti semantiche, e sono largamente utilizzati per rappresentare i fatti nei sistemi esperti. Tutta l'informazione relativa a un particolare concetto viene memorizzata in una singola unità complessa (frame). A prima vista i frames somigliano moltissimo alle strutture dati tipo record.

Tre semplici frames sono riportati nella seguente figura:



Mammal, Elephant e *Nellie* sono detti *oggetti.* Le proprietà, quali *colour* e *size,* sono dette *slots* e *grey, large,* etc. sono detti *slot values.* Possiamo dedurre, usando l'ereditarietà, che Nellie è grande, di colore grigio, ha una testa e le piacciono le mele.

E' facile passare dalla rappresentazione a frame a quella con le reti semantiche e viceversa. I nodi nelle reti diventano oggetti nei frames, i links diventano slots, e il nodo all'estremo del link diventa lo slot value.

Faremo nel seguito degli esempi basati sui frames, ma si potrebbero anche utilizzare le reti.

Valori di difetto ed ereditarietà multipla

Finora abbiamo visto oggetti (ad esempio, Nellie) che ereditano tutte le proprietà della classe padre. Per cui Nellie deve essere grande e grigia.

E' utile però descrivere proprietà *tipiche* di una classe e permettere a un instance di quella classe di essere un'eccezione alla regola. Il valore di una proprietà che è tipica di una classe è detto valore di difetto (*default),* e può essere annullato dando un differente valore a un instance o a una sottoclasse.

Nella seguente figura gli slots preceduti da un asterisco contengono valori default che possono essere annullati. Ora gli oggetti e le classi ereditano le proprietà di default delle loro classi padre a meno che essi abbiano un valore individuale differente da quello ereditato.

```
Mammal:
   subclass:        Animal
   warm_blooded:    yes
   * furry:         yes

Elephant:
   subclass:        Mammal
   has_trunk:       yes
   * colour:        grey
   * size:          large
   * furry:         no

Clyde:
   instance:        Elephant
   colour:          pink
   owner:           Fred

Nellie:
   instance:        Elephant
   size:            small
```

L'ereditarietà è semplice quando ogni oggetto e classe ha una sola classe padre. Spesso però abbiamo a che fare con ereditarietà *multipla* cioè più classi padre sono presenti, e questo complica le cose:

```
Elephant:
   subclass:      Mammal
   has_trunk:     yes
   * colour:      grey
   * size:        large
   * habitat:     jungle

Circus-Animal:
   subclass:      Animal
   habitat:       tent
   skills:        balancing-on-ball

Clyde:
   instance:      Circus-Animal Elephant
   colour:        pink
   owner:         Fred
```

Cosa dire circa l'habitat di Clyde ? Essendo un elefante dovrebbe essere la giungla, ma come animale da circo dovrebbe essere una tenda.

Un frame system deve avere dei meccanismi per decidere quale valore ereditare in presenza di conflitti simili. Una possibile soluzione è che l'autore del frame system specifichi un ordine di precedenza per le classi padre. Ad esempio, mettendo per prima *Circus-Animal* in caso di conflitto i valori vanno presi da *Circus-Animal* e non da *Elephant.*

Questa strategia può però non essere sufficiente. Supponiamo che ci sia uno slot che dice che gli animali da circo sono tipicamente piccoli. In tal caso dovremmo ereditare la proprietà size da *Elephant* e la proprietà habitat da *Circus-Animal.* Per gestire la situazione dovremmo introdurre una nuova classe *Circus-Elephant* con i suoi tipici valori.

Slots e procedure

In generale, sia gli slots che gli slot values possono a loro volta essere dei frames. Ad esempio, il padrone di Clyde è Fred. Fred è il valore dello slot owner, ma potrebbe essere rappresentato da un altro frame che descrive le proprietà di Fred.

Permettendo agli slots di essere dei frames possiamo specificare vari attributi di uno slot. Ad esempio, potremmo specificare che lo slot owner può assumere solo valori della classe *person*, ha uno slot inverso *owns*, e può assumere *multiple-values* in quanto più persone possono possedere lo stesso oggetto.

Inoltre gli slots possono includere delle *procedure*. Un programma può essere inserito in uno slot ed essere eseguito quando viene richiesto un valore per quello slot.

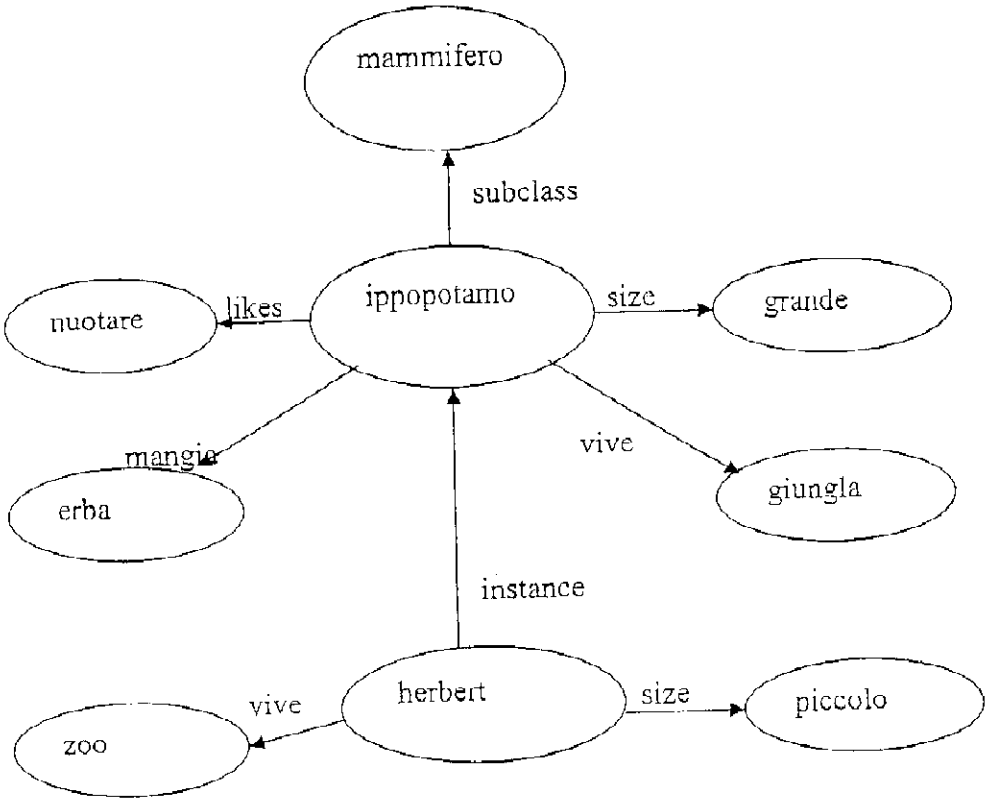
Da quanto detto si vede che è difficile, semplicemente guardando l'insieme dei frames, predire esattamente cosa sarà dedotto riguardo a uno specifico oggetto.

Esempio

Rappresentiamo i seguenti fatti a) mediante una rete semantica, b) mediante frames

Herbert è un piccolo ippopotamo che vive nello zoo. Come tutti gli ippopotami mangia erba e gli piace nuotare.

Rete semantica:



Frames:
Ippopotamo
Subclass: mammiferi
* Size: grande
* Vive: giungla
Mangia: erba
Likes: nuotare

Herbert
Instance: ippopotamo
Size: piccolo
Vive: zoo

Realizzare un semplice frame system

Un frame system elementare, che consente valori default ma non l'ereditarietà multipla, può essere facilmente realizzato in qualsiasi linguaggio di programmazione.

I frames stessi possono essere rappresentati utilizzando istruzioni tipo struct del C, noi assumiamo solo che una funzione *slot-value* è stata definita la quale, dati un oggetto e un attributo, restituisce un valore se esiste un valore, altrimenti una segnalazione di non esistenza.

Definiamo ora una funzione che determina ciò che può essere dedotto per ereditarietà:

Find-value (O, A)

- IF slot-value (O, A) restituisce un valore V, return V

- Else IF slot-value (O, subclass) oppure slot-value (O, instance) restituisce un valore C, Find-value (C, A)

- Else fallimento

Consideriamo l'esempio di Nellie e supponiamo di voler sapere il suo colore, cioè value (Nellie, colour).

L'algoritmo prima prova slot-value (Nellie, colour) e poiché non c'è uno specifico valore per il colore di Nellie, ciò viene segnalato da slot-value.

Pertanto si prova senza esito slot-value (Nellie, subclass) e quindi slot-value (Nellie, instance) che fornisce Elephant.

Si chiama allora ricorsivamente la funzione find alla ricerca di value (Elephant, colour).

Ora slot-value (Elephant, colour) restituisce grey, per cui questo valore è associato a value (Elephant, colour) e quindi a value (Nellie, colour).

Esempio

Utilizziamo la funzione ricorsiva Find-value, che determina ciò che può essere dedotto per ereditarietà in un frame system, per stabilire se una Smart ha i fari:

autoveicolo:
    ruote: sì
    motore: sì
    fari: sì

autovettura:
    subclass: autoveicolo
    ruote: 4
    * posti: 2-7
    * lunghezza: 4-5 metri

smart:
    subclass: autovettura
    posti: 2
    lunghezza: 2.5 metri

Nel nostro caso cerchiamo value (smart, fari). Nel frame smart non c'è lo slot fari per cui vediamo se c'è uno slot per subclass o instance. Lo slot per subclass è presente per cui chiamiamo ricorsivamente Find-value (autovettura, fari).

Lo slot per fari non c'è nel frame autovettura, ma è presente lo slot per subclass per cui chiamiamo ancora Find-value (autoveicolo, fari). Stavolta lo slot fari è presente per cui
value (smart, fari) = sì.

Adeguatezza rappresentativa

Le reti semantiche e i frames forniscono un modo semplice e chiaro per rappresentare proprietà di oggetti e categorie di oggetti. Vi sono però molte cose che non possono essere facilmente rappresentate.

Ad esempio, è difficile esprimere la *negazione* (cioè che qualcosa non è vero), la *disgiunzione* (cioè che una cosa OR un'altra sono vere), certi tipi di *quantificazione* (cioè che qualcosa è vero per tutti o per alcuni elementi di un insieme di oggetti).

Se esprimere queste cose è necessario conviene usare una *logica.* Comunque reti e frames conservano la loro utilità quando semplici tipi di conoscenza vanno rappresentati.

Logica

Il più importante KRL è senza dubbio la *logica dei predicati* la quale ci permette di rappresentare fatti complessi e di derivare nuovi fatti in modo tale che, se i fatti di partenza sono veri, anche le conclusioni lo sono. Si tratta di un linguaggio formale che ha sintassi, semantica e regole di inferenza ben definite.

Una *logica* è un sistema formale che può essere descritto in termini della sua *sintassi* (quali sono le espressioni ammissibili), la sua *semantica* (cosa significano le espressioni) e la sua *teoria della dimostrazione (proof theory)* (come possiamo dedurre nuove conclusioni date alcune premesse).

Questi tre aspetti li consideriamo prima riguardo alla *logica proposizionale* (molto più semplice) e poi riguardo alla logica dei predicati.

Logica proposizionale

Sintassi

Nella logica proposizionale i simboli vengono usati per rappresentare fatti. Per esempio, il fatto "Alison likes cakes" potrebbe essere rappresentato dal simbolo P. Fatti semplici come questo vengono detti *proposizioni atomiche.*

E' possibile costruire proposizioni più complesse, dette *sentenze,* combinando proposizioni atomiche mediante i *connettivi logici* $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (implicazione), $\leftrightarrow$ (equivalenza).

Ad esempio, se la proposizione Q rappresenta il fatto "Alison eats cakes", noi possiamo avere le sentenze:

$P \vee Q$: "Alison likes cakes or Alison eats cakes"
$P \wedge Q$: "Alison likes cakes and Alison eats cakes"
$\neg Q$: "Alison doesn't eat cakes"
$P \rightarrow Q$: "If Alison likes cakes then Alison eats cakes"
$P \leftrightarrow Q$: "If Alison likes cakes then Alison eats cakes, and vice versa"

In generale, se X e Y sono sentenze, lo sono anche $X \wedge Y$, $X \vee Y$, $\neg X$, $X \rightarrow Y$, $X \leftrightarrow Y$.

In tal modo definiamo la *sintassi* della logica. Le seguenti sono tutte sentenze valide in logica proposizionale:

$P \vee \neg Q$
$P \wedge (P \rightarrow Q)$
$(Q \vee \neg R) \rightarrow P$

Semantica

La *semantica* della logica proposizionale permette di stabilire il significato delle sentenze. Ad esempio, sapendo che P, Q e R sono sentenze vere, la semantica ci consente di sapere se la sentenza $(P \vee Q) \wedge R$ è vera.

Possiamo determinare la verità o la falsità (il *valore di verità)* delle sentenze utilizzando *tavole di verità* che definiscono i valori di sentenze contenenti connettivi in funzione dei valori delle sentenze componenti. Le tavole di verità forniscono una semplice *semantica* per i connettivi logici:

OR

X	Y	X ∨ Y
T	T	T
T	F	T
F	T	T
F	F	F

AND

X	Y	X ∧ Y
T	T	T
T	F	F
F	T	F
F	F	F

NOT

X	¬X
T	F
F	T

→

X	Y	X → Y
T	T	T
T	F	T
F	T	F
F	F	T

$X \vee (\neg Y)$

Viceversa, supponiamo, ad esempio, di sapere che la sentenza *raining* ∨ *sunny* è vera. Guardando la tavola di verità per l'OR, possiamo desumere che uno dei tre casi seguenti si verifica: 1) piove, 2) c'è il sole, 3) piove e c'è il sole. Pertanto, partendo dal valore di verità della sentenza e conoscendo le tabelle, possiamo risalire ai possibili valori di verità delle proposizioni componenti la sentenza.

Proof theory

Per dedurre nuovi fatti dobbiamo applicare *regole di inferenza.* La semantica della logica definisce quali regole di inferenza sono *valide.* Ciò ci fornisce la proof theory della logica.

Un'importante regola è il *modus ponens:*

$$\frac{A,\ A \rightarrow B}{B}$$

Questa regola dice che se A è vera e $A \rightarrow B$ è anch'essa vera, allora B è necessariamente vera.
Vi sono molte altre regole di inferenza. Una molto importante è la regola di *risoluzione:*

$$\frac{A \vee B,\ \neg B \vee C}{A \vee C}$$

Questa regola esprime il fatto che, siccome B non può essere sia vera che falsa, uno degli altri or deve essere vero in almeno una delle proposizioni componenti (A, C).

Ad esempio, supponiamo che le seguenti sentenze siano vere:

sunny $\vee$ raining
$\neg$ raining $\vee$ carryumbrella

Utilizzando la risoluzione concludiamo sunny $\vee$ carryumbrella

Se vogliamo dimostrare la verità o la falsità di una certa proposizione, date alcune sentenze che sappiamo essere vere, la regola di risoluzione è sufficiente.

Prima dobbiamo mettere le sentenze vere in una forma standardizzata (detta *forma normale)* che consente l'applicazione della regola.

Quindi vi è una procedura detta *dimostrazione per refutazione* che può essere applicata. Assumiamo che la proposizione sia falsa e vediamo se questo ci porta a una contraddizione.

Logica dei predicati

Sintassi

Il problema che nasce con la logica proposizionale è che non è possibile gestire frasi del tipo "Alison eats everything that she likes". Dovremmo avere moltissime regole, una per ogni cosa che piace ad Alison. La logica dei predicati invece ammette sentenze generali di questo tipo.

Le sentenze nella logica dei predicati sono costruite a partire da *sentenze atomiche*. La logica esprime i fatti di base in termini di un predicato e alcuni argomenti. Ad esempio, nel caso di "Alison likes chocolate" possiamo avere un nome di predicato *likes* e argomenti *alison* e *chocolate* in modo da avere la sentenza *likes (alison, chocolate)*.

Questo modo di procedere risulta molto più flessibile rispetto ad avere una proposizione indivisibile P che rappresenta l'intera frase, in quanto possiamo gestire le entità *alison* e *chocolate*.

In generale gli argomenti in una sentenza atomica sono i cosiddetti *termini*. I termini possono essere:

Simboli costanti tipo *alison*

Simboli variabili tipo X

Espressioni funzionali tipo *father (alison)*. Un'espressione funzionale consiste di un funtore seguito da i suoi argomenti.

Pertanto, ad esempio, abbiamo le seguenti sentenze atomiche:

friends (alison, richard)
friends (father (fred), father (joe))
likes (X, richard)

Le sentenze, analogamente a quanto avviene nella logica proposizionale, sono costruite combinando sentenze atomiche mediante connettivi. Ad esempio:

friends (alison, richard) $\rightarrow$ *likes (alison, richard)*

likes (alison, richard) $\vee$ *likes (alison, chocolate)*

(likes (alison, richard) $\vee$ *likes (alison, chocolate))* $\wedge$
$\neg$ *likes(alison, chocolate)*

Le sentenze possono anche essere costruite usando i *quantificatori* i quali indicano come vanno trattate le variabili nella sentenza. I due quantificatori utilizzati nella logica dei predicati sono $\forall$ e $\exists$.

$\forall$ viene letto "per tutti" e viene usato per affermare che qualcosa è vero per tutti gli oggetti, $\exists$ si legge "esiste" e viene usato per affermare che qualcosa è vero per almeno un oggetto. Ad esempio, abbiamo:

$\forall$ *X (likes (alison, X)* $\rightarrow$ *eats (alison, X))* cioè Alison mangia tutto ciò che le piace

$\exists$ X (*bird* (X) $\wedge$ $\neg$ *flies* (X)) cioè esiste qualche uccello che non vola

$\forall$ X (*person* (X) $\rightarrow$ $\exists$ Y *loves* (X, Y)) cioè ogni persona ha qualcosa da amare

Una sentenza deve avere tutte le sue variabili quantificate. Pertanto una espressione come $\forall$ X *loves* (X, Y) pur essendo una *formula ben formata* del calcolo dei predicati, non è una sentenza in quanto la variabile Y non è quantificata. Le sentenze sono anche dette *formule chiuse*.

Semantica

La semantica della logica dei predicati è definita, come nella logica proposizionale, in termini dei valori di verità delle sentenze. Possiamo usare le tavole di verità per trovare il valore di verità delle sentenze contenenti connettivi logici a partire dai valori delle parti componenti la sentenza.

Dobbiamo però ora gestire predicati, argomenti e quantificatori. Formalmente il significato di un predicato come *likes* può essere definito, fissato un *dominio* di interesse, in termini dell'insieme di tutte le coppie (X,Y) per cui Y piace a X. Noi comunque assumiamo che in qualche modo un valore di verità può essere assegnato a un frase come *likes (alison, chocolate)*.

Il significato di $\forall$ può essere definito esaminando se la sentenza è vera per tutti gli oggetti nel dominio di interesse. Per esempio, supponendo che il nostro dominio include solo Fred, Jim e Joe, noi possiamo decidere se la sentenza $\forall$ X *likes (X, chocolate)* è vera o meno controllando se le sentenze *likes (Fred, chocolate), likes (Jim, chocolate)* e *likes (Joe, chocolate)* sono tutte vere.

Il significato di $\exists$ può essere definito in modo analogo esaminando se la sentenza è vera per almeno uno degli oggetti nel dominio. Ad esempio, se a Fred piace il cioccolato e agli altri due non piace la sentenza $\exists$ X *likes (X, chocolate)* è vera, mentre $\forall$ X *likes (X, chocolate)* è falsa.

Quanto detto serve solo a dare un cenno su come possiamo associare una semantica alle espressioni nella logica dei predicati. La cosa importante è che ogni cosa è definita con precisione.

Proof theory

Le regole di inferenza e le procedure dimostrative nella logica dei predicati sono simili a quelle presenti nella logica proposizionale. Modus Ponens e Risoluzione sono ancora valide ma vanno modificate per gestire espressioni che includono variabili e quantificatori.

Per esempio, partendo dai fatti $\forall$ X *(man (X)* $\rightarrow$ *mortal (X))* e *man (socrates),* vorremmo essere in grado di concludere *mortal (socrates).* A tal fine ci occorre il Modus Ponens modificato che consente di abbinare sentenze con altre sentenze. Ad esempio, *man (X)* può essere abbinato con *man (socrates)* con X=*socrates*.

Anche la regola di risoluzione può essere opportunamente modificata. La dimostrazione per refutazione può essere applicata e così abbiamo una procedura di dimostrazione appropriata *(sound)* che ci garantisce che le conclusioni sono valide.

Rappresentazione della conoscenza utilizzando la logica dei predicati

Rappresentazione dei fatti nella logica

Vediamo altri esempi di come possiamo passare da una frase del linguaggio naturale a una sentenza della logica.

La frase "Mary is tall and beautiful" diventa:
tall (mary) $\wedge$ *beautiful (mary)*

La frase "Alison likes chocolate and cream" diventa:
likes (alison, chocolate) $\wedge$ *likes (alison, cream)*

Frasi della forma "se X allora Y" possono essere tradotte con $X \rightarrow Y$.
Ad esempio, la frase "if Alison is hungry then she eats chocolate" diventa:
hungry (alison) $\rightarrow$ *eats (alison, chocolate)*

La frase "Alison eats chocolate or biscuits" diventa:
eats (alison, chocolate) $\vee$ *eats (alison, biscuits)*

La frase "Alison doesn't like cauliflowers" diventa :
$\neg$ *likes (alison, cauliflowers)*

La frase "all students study" diventa:
$\forall$ X *student (X)* $\rightarrow$ *study (X)*

La frase "Someone strange likes cauliflowers" diventa:
$\exists$ X *strange (X)* $\wedge$ *likes (X, cauliflowers)*

vi è almeno una tavola che non ha quattro gambe
$\exists X$ *(table (X)* $\wedge$ $\neg$ *numlegs (X, 4)*

tutti gli elefanti sono grigi
$\forall$ *X (elephant (X)* $\rightarrow$ *grey(X))*

tutti gli abitanti di Glasgow tifano o per i Rangers o per il Celtic
$\forall X$ *(glaswegian (X)* $\rightarrow$ *(supports (X, rangers)* $\vee$ *supports (X, celtic)))*

vi è qualcosa piccolo e appiccicoso sulla tavola
$\exists X\ (small\ (X) \wedge slimy\ (X) \wedge on\ (X, table))$

non vi è nessun cavolfiore che sia di buon sapore
$\neg\ \exists X\ (cauliflower\ (X) \wedge tasty\ (X))$

ogni mela è verde o gialla
$\forall X\ \ mela\ (x) \rightarrow verde\ (X) \vee gialla\ (X)$

nessuna mela è blu
$\neg\ \exists X\ (mela\ (X) \wedge blu\ (X))$

se una mela è verde allora è saporita
$\forall X\ ((mela\ (X) \wedge verde\ (X)) \rightarrow saporita\ (X))$

a ogni uomo piace una mela saporita
$\forall X, Y\ ((uomo\ (X) \wedge mela\ (Y) \wedge\ saporita\ (Y)) \rightarrow\ likes\ (X,\ Y)$

se X è genitore di Y allora Y è figlio di X
$\forall X, Y\ \ (genitore\ (X,Y)\ \rightarrow figlio\ (Y,X))$

Herbert è un piccolo ippopotamo che vive nello zoo. Come tutti gli ippopotami mangia erba e gli piace nuotare.

$\exists X\ (\ ippopotamo\ (X) \wedge nome\ (X,\ Herbert) \wedge vive\ (X,\ zoo))$
$\forall Y\ (\ ippopotamo\ (Y) \rightarrow\ mangia\ (Y,\ erba) \wedge\ likes\ (Y,\ nuotare))$

Ogni comune ha un solo sindaco
$\forall X\ \ comune\ (X) \rightarrow (\exists Y\ \ sindaco\ (Y,\ X) \wedge\ \forall Z\ \ sindaco\ (Z,\ X) \rightarrow Y = Z)$

Utilizzo della logica in AI

La logica dei predicati fornisce un potente strumento per rappresentare e ragionare con la conoscenza. Alcune cose che non possono essere facilmente rappresentate usando i frames, quali la negazione, la disgiunzione e la quantificazione, sono agevolmente rappresentate con la logica dei predicati. Le regole di inferenza e le procedure di dimostrazione disponibili consentono di effettuare un'ampia varietà di inferenze oltre la semplice inferenza basata sull'ereditarietà consentita nei frame systems.

Anche nella logica dei predicati ci sono però cose difficili da rappresentare, ad esempio fatti che contengono incertezza (probabilmente pioverà domani), default (di solito piove a Glasgow), opinioni (John pensa che pioverà, io no) e cambiamenti temporali (diventerà sempre più piovoso man mano che ti avvicini a Glasgow).

Sono stati sviluppati metodi per gestire questi tipi di fatti e logiche più complesse sono state introdotte. Al crescere della complessità della logica, però le inferenze diventano meno efficienti. Per problematiche complesse il programmatore AI deve a volte scegliere tra un approccio rigoroso basato sulla logica, con semantica chiara e inferenze sound, e un approccio ad hoc che offre una maggiore efficienza.

La logica non viene solo usata come KRL ma anche come la base di linguaggi di programmazione. L'esempio più noto è il Prolog, basato sulla logica dei predicati. Ogni istruzione di un programma Prolog corrisponde a una sentenza nella logica dei predicati. Ad esempio, vediamo un frammento di programma Prolog e le corrispondenti sentenze nella logica:

father(jim, fred).
father (joe, jim).
grandfather (X, Y) :- father (X, Z), father (Z, Y).

father (jim, fred)
father (joe, jim)
$\forall$ X Y Z *((father (X, Z)* $\wedge$ *father (Z, Y)* $\rightarrow$ *grandfather (X, Y))*

Dato il precedente programma possiamo chiedere al Prolog di dimostrare che *grandfather (joe, fred)* è una sentenza vera. Prolog dimostra le sentenze usando una variante della procedura di risoluzione e risponde affermativamente.

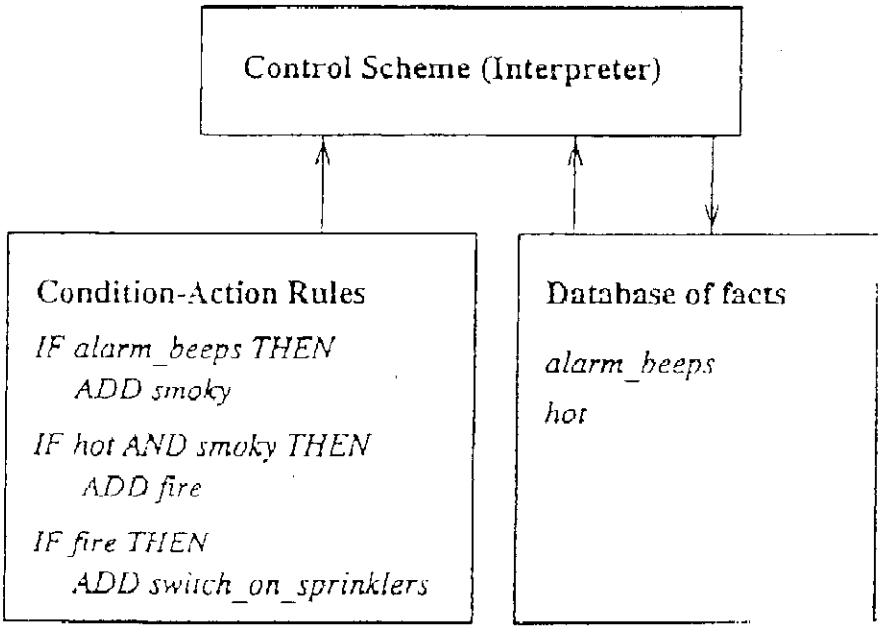
La logica dei predicati è un KRL più potente delle reti e dei frames, però alcuni fatti possono essere meno chiari utilizzando la logica. Inoltre le procedure di dimostrazione sono più efficienti con i frames anche se l'adeguatezza rappresentativa dei frames è inferiore a quella della logica dei predicati.

In generale, al crescere dell'adeguatezza rappresentativa, cioè la capacità espressiva del sistema, diminuisce l'efficienza inferenziale.

Sistemi basati su regole

Invece di rappresentare la conoscenza in modo dichiarativo e statico (un insieme di sentenze che sono vere), i sistemi basati su regole si servono di un insieme di regole che dicono cosa fare o cosa concludere a seconda delle situazioni che si presentano.

Un sistema basato su regole consiste di un insieme di regole IF-THEN, un insieme di *fatti* che rappresentano cose considerate vere e un *interprete* che controlla, dati i fatti, l'applicazione delle regole.



Le regole IF-THEN sono trattate differentemente dai costrutti similari presenti nei linguaggi procedurali tipo il C. Infatti mentre il C tratta i costrutti IF-THEN come parte di una *sequenza* di istruzioni che vengono eseguite una di seguito all'altra, un sistema basato su regole tratta ciascuna regola come un pezzo indipendente di conoscenza, da utilizzare quando è necessario sotto il controllo dell'interprete.

Vi sono due tipi principali di interprete: *forward chaining* (FC) e *backward chaining* (BC). In un sistema FC si parte da alcuni fatti iniziali e si usano le regole per arrivare a delle conclusioni. In un sistema BC si parte da un'ipotesi (goal) che si vorrebbe dimostrare e si cercano le regole appropriate per arrivare al goal, anche individuando dei subgoal nel corso del procedimento. Si dice che i sistemi FC sono *data-driven* mentre i sistemi BC sono *goal-driven*.

Sistemi forward chaining

In un sistema FC i fatti sono contenuti in una *memoria di lavoro* che è continuamente aggiornata man mano che le regole vengono utilizzate.

Le regole rappresentano azioni possibili da compiere quando dei fatti si verificano nella memoria di lavoro. Le regole vengono anche dette regole *condizione-azione.* Le azioni in genere comportano l'aggiunta e la cancellazione di elementi presenti nella memoria di lavoro, ma l'azione può anche consistere, ad esempio, nella stampa di un messaggio o nell'esecuzione di una specifica procedura.

L'interprete controlla l'applicazione delle regole e la sua attività si basa su un ciclo detto *recognize-act.* Il sistema innanzitutto trova tutte le regole le cui parti condizione sono vere, dato lo stato attuale della memoria di lavoro. Quindi seleziona una delle regole ed esegue le azioni specificate nella parte azione. Si dice che avviene il *firing* (sparo) della regola.

La regola viene selezionata basandosi su strategie fissate dette *strategie per la risoluzione dei conflitti.* Le azioni eseguite modificano la memoria di lavoro e il ciclo inizia di nuovo. Il ciclo viene ripetuto finchè o nessuna regola spara o un simbolo di halt viene aggiunto alla memoria di lavoro.

L'algoritmo di base per il FC può essere schematizzato come segue:

Ripeti:

- trova tutte le regole che hanno condizioni (parte IF) verificate
- seleziona una regola, usando le strategie per la risoluzione dei conflitti
- esegui le azioni previste nella parte THEN, modificando eventualmente la memoria di lavoro

finchè nessuna regola può sparare oppure halt è presente nella memoria di lavoro.

Esempio

Consideriamo le regole e i fatti del nostro possibile incendio:

R1: IF hot AND smoky THEN ADD fire

R2: IF alarm-beeps THEN ADD smoky

R3: IF fire THEN ADD switch-on-sprinklers

F1: alarm-beeps

F2: hot

Inizialmente il sistema controlla se ci sono regole per le quali valgono le condizioni, dati i due fatti F1 e F2. L'unica regola attivabile è R2, pertanto la regola viene selezionata e la parte azione eseguita. Per cui il nuovo fatto *smoky* viene aggiunto alla memoria di lavoro:

F3: *smoky*

Riparte il ciclo, le condizioni della prima regola sono verificate e uno nuovo fatto viene aggiunto alla memoria di lavoro:

F4: *fire*

Il ciclo inizia di nuovo, la regola R3 spara, il fatto *switch-on-sprinklers* viene aggiunto alla memoria e sperabilmente l'impianto antincendio sprinklers viene attivato.

Questo semplicissimo esempio chiarisce bene la radicale differenza con gli IF-THEN dei linguaggi procedurali: l'ordine in cui sparano le regole dipende dal contenuto della memoria di lavoro e non dall'ordine in cui sono fornite le regole stesse.

Cosa accade se più di una regola ha la parte condizione soddisfatta ? Estendiamo l'esempio aggiungendo due regole e un fatto:

R1: IF hot AND smoky THEN ADD fire

R2: IF alarm-beeps THEN ADD smoky

R3: IF fire THEN ADD switch-on-sprinklers

R4: IF dry THEN ADD switch-on-humidifier

R5: IF sprinklers-on THEN delete-dry

F1: alarm-beeps

F2: hot

F3: dry

Nel primo ciclo vi sono due regole che possono essere applicate, R2 e R4. Se viene scelta R4, l'umidificatore viene acceso, il ciclo riparte, viene selezionata R2 essendo R4 già stata applicata, e tutto prosegue come nell'esempio precedente.

Se invece nel primo ciclo viene scelta R2, supponendo che le scelte successive siano R1, R3 e R5, *dry* viene cancellato dalla memoria e l'umidificatore non avrà modo di essere attivato.

Ovviamente nel secondo, terzo e quarto ciclo poteva anche essere attivata R4, per cui vanno individuati dei metodi di selezione.

Risoluzione di conflitti

Chiaramente è importante che in presenza di più regole attivabili venga selezionata la più appropriata in quanto come visto nell'esempio la conclusione può essere molto diversa in funzione della scelta fatta.

Un sistema FC avrà al suo interno delle *strategie per la risoluzione di conflitti* che verranno attivate per decidere quale regola selezionare. Strategie usuali sono:

a) scegliere regole che coinvolgono fatti aggiunti recentemente alla memoria di lavoro. In tal caso, se inizialmente R2 spara, la regola successivamente scelta sarà R1 e non R4 in quanto *smoky* è stato aggiunto per ultimo.

b) scegliere regole aventi condizioni più specifiche. Ad esempio, se avessimo la regola R6: *IF hot THEN ADD summer* e fatti *hot* e *smoky* , R1 sarebbe scelta e non R6 in quanto R1 ha condizioni più dettagliate. La regola R1 può essere vista come una eccezione alla regola più generale R6.

c) permettere all'utente di assegnare priorità alle regole. Per esempio, R4 potrebbe avere una priorità bassissima, in modo da essere attivata solo se non c'è nessuna altra regola attivabile.

d) attivare insieme tutte le regole applicabili. Questa è una strategia opposta alla prima delineata e porta alla esplorazione di tutte le possibili catene di ragionamento. Le strategie a) e d) sono simili alle strategie di ricerca che vedremo in seguito dette *depth first search* e *breadth first search.*

e) eliminare automaticamente i fatti le cui *giustificazioni* non sono più valide per effetto di cambiamenti nella memoria di lavoro. Ad esempio, *switch-on-humidifier* è stato aggiunto perché ci sono i fatti *hot* e *dry*. Ma quando *dry* viene eliminato dalla memoria, *switch-on-humidifier* non ha più una giustificazione valida e può anch'esso essere eliminato.

Pattern matching

Abbiamo esaminato regole e fatti semplicissimi. Nella realtà ci saranno fatti più complessi, ad esempio *temperature (kitchen, hot)* invece di *hot*, e le regole potranno avere *patterns* che sono abbinati (*matched)* con i fatti presenti nella memoria di lavoro. Ad esempio, supponiamo di avere:

R7: *IF temperature (R, hot) AND environment (R, smoky)*
THEN ADD fire-in (R)

F6: *temperature (kitchen, hot)*

F7: *environment (kitchen, smoky)*

La regola contiene una variabile R che può assumere qualsiasi valore. Pertanto *temperature (R, hot)* si abbina con *temperature (kitchen, hot)* dove R=*kitchen*. Inoltre *environment (R, smoky)*, con R=*kitchen*, si abbina con F7 e pertanto il fatto *fire-in (kitchen)* viene aggiunto alla memoria di lavoro.

Nel backward chaining il pattern matching viene usato per abbinare goals con parti destre delle regole. Supponendo che stiamo cercando di provare *fire-in (library)*, questo goal matches la conclusione della R7 con R=*library*.

Sistemi backward chaining

Finora abbiamo visto come i sistemi basati su regole possono essere usati per ottenere nuove conclusioni dai dati esistenti, aggiungendo queste conclusioni alla memoria di lavoro. Questo approccio è appropriato allorchè tutti i fatti iniziali sono noti e non sappiamo quali sono le probabili conclusioni.

Se invece sappiamo quale potrebbe essere la conclusione, o abbiamo una specifica ipotesi da verificare, i sistemi FC possono essere inefficienti. Infatti si può procedere finchè nessuna regola risulta applicabile o l'ipotesi è stata aggiunta alla memoria di lavoro. Ma nel corso del processo è molto probabile che il sistema abbia fatto molto lavoro inutile, aggiungendo conclusioni irrilevanti alla memoria di lavoro.

Ad esempio, supponiamo di avere le seguenti regole:

R1: IF smoky AND hot THEN ADD fire

R2: IF alarm-beeps THEN ADD smoky

R3: IF alarm-beeps THEN ADD ear-plugs

R4: IF fire THEN ADD switch-on-sprinklers

R5: IF smoky THEN ADD poor-visibility

F1: alarm-beeps

F2: hot

Se siamo interessati solo all'attivazione o meno degli sprinklers, concludere che la visibilità è scarsa o che vanno messe le cuffie è chiaramente di nessun interesse.

Con il backward chaining evitiamo di effettuare queste inferenze corrette ma irrilevanti e ci concentriamo sull'ipotesi significativa, cioè se vanno attivati gli sprinklers.

Iniziamo con il goal da dimostrare e cioè *switch-on-sprinklers*. Il sistema controlla se il goal si abbina con uno dei fatti presenti. Se ciò accade il goal è dimostrato altrimenti il sistema cerca regole le cui conclusioni concordano con il goal.

Se nessuna regola si abbina con il goal si ha fallimento altrimenti una regola viene selezionata e il sistema cerca di provare i fatti presenti nella parte condizione della regola (detti *precondizioni)* che diventano i nuovi goals da provare.

Notiamo che un sistema BC non ha bisogno di aggiornare la memoria di lavoro, però deve conservare traccia di quali goals deve dimostrare per arrivare a provare l'ipotesi di partenza.

Tralasciando per il momento il problema di cosa fare se più di una regola può essere applicata per arrivare al goal, l'algoritmo di base è il seguente:

Per dimostrare il goal G:

- se G è presente nei fatti iniziali, G è provato

- altrimenti, trova una regola che può essere usata per concludere G, e tenta di provare ciascuna precondizione della regola.
  G è dimostrato vero se tutte le precondizioni sono provate vere.

Esempio

Supponiamo di avere le regole precedenti e il seguente goal iniziale:

G1: *switch-on-sprinklers*

Controlliamo se G1 è presente tra i fatti, non lo è e quindi cerchiamo di abbinarlo con le parti destre delle regole. Il match c'è solo con la R4 per cui la precondizione di questa regola diventa il nuovo goal da provare:

G2: *fire*

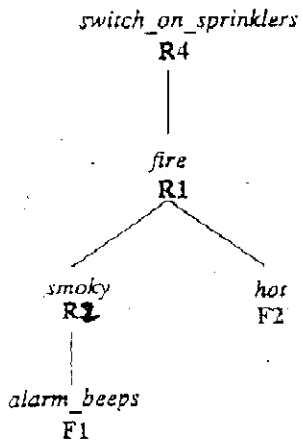
G2 non è presente tra i fatti ma si abbina con la conclusione di R1, pertanto le sue precondizioni diventano i nuovi goals da provare:

G3: *smoky*
G4: *hot*

G3 concorda con la conclusione di R2 per cui *alarm-beeps* diventa un nuovo goal. I goals da provare sono:

G5: *alarm-beeps*
G4: *hot*

Entrambi i goals sono presenti nella memoria di lavoro, per cui sono veri. Poiché tutti i goals sono dimostrati, il goal iniziale deve essere vero e pertanto gli sprinklers vanno attivati. La dimostrazione effettuata per la nostra ipotesi *switch-on-sprinklers* può essere così rappresentata:



Esempio

Supponiamo che un sistema estremamente semplificato e non realistico abbia le seguenti regole:

R1: IF NOT risparmi-adeguati THEN ADD investire-in-risparmi
R2: IF risparmi-adeguati AND reddito-adeguato THEN
ADD investire-in-azioni
R3: IF NOT avere-figli THEN ADD risparmi-adeguati
R4: IF avere-coniuge AND coniuge-lavora THEN ADD reddito-adeguato

e che i fatti iniziali siano i seguenti: "NOT avere-figli", "avere-coniuge", "coniuge-lavora".

a) Applichiamo l'algoritmo di base per il forward chaining
b) Utilizziamo l'algoritmo di base per il backward chaining per dimostrare l'ipotesi G1: "investire-in-azioni".

Forward chaining

Le regole R3 e R4 possono essere attivate, supponiamo che venga per prima attivata la R3, pertanto il fatto "risparmi-adeguati" viene aggiunto alla memoria di lavoro, viene quindi attivata la R4 e viene aggiunto "reddito-adeguato". Ora può essere attivata la R2 e viene aggiunto "investire-in-azioni".

Backward chaining

G1 non è presente nei fatti iniziali, però c'è la regola R2 che mi permette di concludere G1, per cui tento di provare le precondizioni di R2. "risparmi-adeguati" è provato in quanto è la conclusione di R3 e la sua precondizione è un fatto iniziale. Lo stesso accade per "reddito-adeguato" per cui G1 è provato.

Implementazione

Un sistema backward chaining può essere realizzato utilizzando uno stack per memorizzare i goals che vanno dimostrati. Si preleva un goal dallo stack e si cerca di provarlo. Se è nella memoria di lavoro il goal è provato. Altrimenti, se si abbina con la conclusione di una regola allora le sue precondizioni vengono inserite nello stack, se non c'è nessun match possibile il goal fallisce. Il goal iniziale ha successo se tutti i subgoals sono man mano prelevati dallo stack e nessuno fallisce.

Cosa fare se vi è più di una regola che ha la stessa conclusione e può essere usata per provare un particolare goal ? In questo caso il sistema deve provare tutte le regole e controllare se il goal può essere provato usando una di esse.

Le *tecniche di ricerca* che esamineremo in seguito possono essere utilizzate per provare tutte le possibilità. Per esempio, utilizzando l'algoritmo *depth first search,* se fallisce la dimostrazione di un goal usando una regola, l'algoritmo "torna indietro" all'ultimo passo in cui c'era una possibilità di scelta, e seleziona un'altra regola.

Confronto FC – BC

La scelta fra FC e BC per risolvere un problema dipende dalle regole e dai fatti iniziali e da quante ipotesi vanno considerate. Se vi è una particolare ipotesi da verificare il BC, come accennato, può essere più efficiente in quanto FC ricaverebbe conclusioni da fatti irrilevanti. A volte però anche BC può essere molto dispersivo, ciò accade se vi sono molte ipotesi e per ognuna di esse vi sono molti modi per cercare di provarla.

Ad esempio, consideriamo i due sistemi esperti MYCIN e XCON. Il primo sistema è stato utilizzato per effettuare diagnosi in caso di infezioni batteriche, il secondo è servito per configurare i computers VAX.

Nella diagnosi medica in un'area ristretta vi è un numero limitato di possibili agenti che causano la malattia per cui operare in modo BC è appropriato e MYCIN opera in tal modo cercando di provare ciascuna delle possibili ipotesi.

Nella configurazione di computer, il numero di possibili configurazioni è enorme e non avrebbe senso esaminarle tutte fino a incontrare quella che soddisfa le esigenze dell'acquirente. Per cui è ragionevole operare in modo FC, partendo dalle specifiche del cliente e costruendo man mano nella memoria di lavoro la configurazione appropriata. Il sistema XCON infatti opera in modo FC.

Confronto fra KRL

Frames e reti sono utili per rappresentare informazione dichiarativa riguardo a insiemi di oggetti e concetti correlati, e in particolare quando sussiste una chiara *gerarchia di classi,* cosicchè possiamo utilizzare l'*ereditarietà* per dedurre gli attributi di oggetti presenti nelle sottoclassi a partire dagli attributi degli oggetti presenti nelle classi padre.

Frames e reti non sono appropriati quando si vogliono effettuare inferenze diverse da quelle basate sull'ereditarietà. A tal fine occorre utilizzare la potenza della logica dei predicati.

Gli approcci basati sulla logica permettono di rappresentare oggetti complessi, che comportano quantificazione, negazione e disgiunzione, e hanno una sintassi, una semantica e una proof theory ben definite.

Questi approcci possono però rivelarsi poco flessibili: ogni regola di inferenza deve essere validata da una precisa semantica per le espressioni nel linguaggio. Ciò comporta che inferenze e conclusioni quotidianamente utilizzate non sono permesse e il ragionamento può essere inefficiente.

I sistemi basati su regole forniscono maggiore flessibilità, a volte le regole includono procedure da eseguire. Mentre la logica è usata in modo dichiarativo, dicendo cosa è vero nel mondo di riferimento, i sistemi basati su regole sono più rivolti alla conoscenza procedurale, cosa fare quando si verifica una certa condizione.

Una caratteristica comune ai tre approcci è che le regole e i fatti sono rappresentati separatamente dalle procedure per il problem-solving e per l'effettuazione delle inferenze. Possiamo, ad esempio, aggiungere nuove regole, frames o sentenze, riutilizzando i fatti e i metodi esistenti.

Un KRL ci mette a disposizione una notazione per la rappresentazione e metodi per l'inferenza della cui soundness siamo sicuri. Utilizzare un KRL per risolvere un problema AI presenta perciò notevoli vantaggi rispetto, ad esempio, allo scrivere ex novo un programma C++ in cui il programmatore deve inventarsi notazioni appropriate e affrontare il difficile compito di verificare la correttezza dei metodi di inferenza utilizzati.

Sommario

I KRL forniscono formalismi ad alto livello per rappresentare la conoscenza richiesta per la soluzione di problemi AI. Un buon linguaggio dovrebbe essere naturale, chiaro e preciso, permettere di rappresentare agevolmente ciò che è necessario, e consentire la deduzione appropriata di nuovi fatti.

Frames e *reti semantiche* rappresentano la conoscenza come un insieme di oggetti aventi attributi e organizzati gerarchicamente. Se un oggetto è una sottoclasse di un altro può ereditare i suoi attributi. Questi linguaggi hanno un potere rappresentativo e inferenziale limitato, ma forniscono una rappresentazione naturale ed efficiente.

La *logica dei predicati* è un linguaggio preciso e formalizzato in grado di rappresentare un'ampia varietà di oggetti. La logica può anche essere usata come base per linguaggi di programmazione (ad esempio, il Prolog).

I *sistemi basati su regole* fanno sì che la conoscenza sia rappresentata come un insieme di regole IF-THEN che dicono quale azione attivare date certe condizioni. Il ragionamento può essere controllato utilizzando un interprete *forward chaining* o *backward chaining*.

3. Sistemi esperti

Abbiamo esaminato come possiamo rappresentare la conoscenza, ora vediamo come possiamo utilizzarla per risolvere problemi reali. In particolare esamineremo i *sistemi esperti*, sistemi che forniscono, come un esperto umano, consigli e diagnosi per la soluzione di vari problemi.

Gli esperti umani possono essere pochi, costosi, non disponibili immediatamente. Un sistema esperto invece può essere sempre a disposizione senza problemi. Un esempio può essere la diagnosi di malattie rare. Un medico generico non ha la competenza necessaria, specialisti non sono subito disponibili, una terapia va adottata con urgenza.

Per costruire un sistema esperto innanzitutto bisogna ricavare la conoscenza appropriata dall'esperto umano. Tale conoscenza è spesso di natura *euristica,* basata sull'esperienza e su regole non formalizzate.

Estrarre tale conoscenza dall'esperto è un lavoro difficile, che richiede a sua volta una persona esperta in tale compito. Un *ingegnere della conoscenza* ha il compito di ricavare la conoscenza al fine di costruire la *base di conoscenza* del sistema esperto.

La costruzione di un sistema esperto pienamente operativo procede per raffinamenti successivi. Infatti l'esperto in genere trova molto difficile esprimere esattamente quali conoscenze e regole vengono da lui usate per risolvere il problema. Molte conoscenze dell'esperto risiedono a livello quasi subconscio o gli sembrano così ovvie da non menzionarle nemmeno.

L'*acquisizione della conoscenza* è una parte fondamentale nella progettazione di un sistema esperto. Si effettuano interviste con l'esperto e gli si chiede di risolvere una varietà di problemi tipici. Un prototipo iniziale viene approntato e mostrato all'esperto il quale verifica le prestazioni del sistema e fornisce indicazioni sulle modifiche da apportare.

Questo processo verifica-modifica procede finchè l'esperto è soddisfatto del comportamento del sistema. Ovviamente, oltre all'esperto umano, il sistema deve essere gradito anche al futuro utente del sistema.

Perché questo processo di sviluppo iterativo sia fattibile è importante che il sistema sia scritto in modo tale che possa essere facilmente modificato. Inoltre il sistema deve essere in grado di spiegare il ragionamento adottato per giungere alla conclusione.

Un approccio ampiamente usato per rappresentare la conoscenza in un sistema esperto è l'uso di regole IF-THEN. I sistemi basati su regole sono flessibili, facilmente modificabili e possono giustificare abbastanza il ragionamento effettuato.

Spesso le regole non indicano conclusioni certe, vi sarà solo uno specifico grado di certezza che la conclusione sia valida se valgono le condizioni. Pertanto occorre definire metodi che determinino il grado di certezza della conclusione finale a partire dal grado di certezza delle regole utilizzate.

I sistemi esperti sono stati usati per risolvere un ampio spettro di problemi in aree quali la medicina, la matematica, l'ingegneria, la geologia, l'informatica, la finanza, l'educazione.

All'interno di ciascuna area, essi sono stati usati per risolvere tipi diversi di problemi. Ad esempio, in elettronica un tipo di problema riguarda la *diagnosi* di guasti nei circuiti, un altro la *progettazione* di un circuito per uno specifico scopo.

Dato un problema, i metodi di problem-solving appropriati dipendono più dal tipo di problema che dal campo di applicazione. Ad esempio, diagnosticare un guasto in un circuito è più vicino alla diagnosi di una malattia che alla progettazione di un circuito, mentre progettare un circuito è più vicino, ad esempio, alla configurazione di una cucina.

E' chiaro che sia il progetto che la diagnosi del circuito hanno in comune moltissima informazione sui componenti del circuito ma rappresentare la conoscenza è un problema relativamente semplice, il lavoro difficile è sviluppare un metodo valido per risolvere il problema.

Vi sono moltissimi metodi per il problem-solving, sviluppati per gestire i vari compiti di un sistema esperto e le prestazioni del sistema dipenderanno in gran parte dalla scelta del metodo più appropriato.

Progettazione di un sistema esperto

I sistemi esperti sono molto diversi sia per i problemi gestiti sia per i metodi di problem-solving utilizzati. Vi sono però degli aspetti comuni a tutti i sistemi, e cioè il tipo di problemi appropriati per un sistema esperto, il modo in cui il sistema viene costruito e l'architettura che viene generalmente usata.

Scelta del problema

La costruzione di un sistema esperto che operi efficacemente nel mondo reale richiede un notevole dispendio di tempo e di denaro. Al fine di evitare fallimenti costosi ed imbarazzanti conviene seguire delle linee guida per stabilire se un problema è adatto per una soluzione tramite sistema esperto.

Innanzitutto, come per ogni progetto software, è importante essere realistici nella valutazione del rapporto costo/benefici. E' facile sottostimare lo sforzo e il costo per lo sviluppo di un sistema esperto pienamente operativo.

Pur essendo oggigiorno disponibili sofisticati strumenti software per lo sviluppo di sistemi esperti, esso è ancora un processo complesso, innanzitutto per le difficoltà di acquisizione della conoscenza dell'esperto umano con i necessari raffinamenti man mano da apportare.

Inoltre ci dobbiamo assicurare che le tecniche tipiche dei sistemi esperti siano appropriate. Problemi che richiedono destrezza manuale o capacità fisiche non sono appropriati, al pari di problemi che richiedono molto senso comune in quanto esso è molto difficile da rappresentare.

Generalmente aree molto tecniche si prestano meglio in quanto coinvolgono piccole quantità di conoscenza ben formalizzata, spesso contenuta in documenti scritti (ad esempio, testi di medicina).

Per alcuni problemi utilizzare tecniche AI è francamente esagerato. Spesso è possibile disegnare un semplice flow-chart che indica cosa fare a seconda delle situazioni (ad esempio, come investire i risparmi) e poi ricavare un semplice programma. Questo programma può essere definito un sistema esperto in quanto fornisce consigli ma i metodi usati sono assolutamente quelli tradizionali della programmazione procedurale. Se metodi semplici funzionano, non ha senso complicarsi la vita.

Infine, una volta stabilito che un sistema esperto è necessario e che le tecniche AI sono appropriate, è importante verificare che l'ambiente sia propizio per la costruzione del sistema. In particolare, è essenziale che vi sia la collaborazione attiva e partecipe di esperti umani che non temano che l'arrivo del sistema esperto metta a rischio il loro posto di lavoro. Inoltre i futuri utenti devono essere coinvolti ed avere un atteggiamento positivo verso il progetto.

Solo un piccolo spettro di problemi è adatto per una soluzione sistemi esperti. Se il problema appartiene a questa classe i benefici apportati dal sistema possono essere notevoli. Ad esempio, sono attualmente in uso con piena soddisfazione sistemi che analizzano campioni di petrolio raccolti durante un'esplorazione e sistemi che configurano computers.

Bisogna anche dire che molti sistemi esperti, pur avendo buone prestazioni, non sono usciti dalla fase di sperimentazione e non vengono praticamente utilizzati. Per avere successo un sistema deve integrarsi facilmente nel lavoro quotidiano, avere un'interfaccia gradevole, essere di agevole utilizzo e offrire consistenti e tangibili benefici all'utente.

Ingegneria della conoscenza

Una volta verificato che il problema è appropriato per una soluzione AI, bisogna estrarre la conoscenza dall'esperto e rappresentarla utilizzando un KRL. Questo è il compito dell'*ingegnere della conoscenza*, e richiede la stretta collaborazione degli *esperti* e degli *utenti finali*.

L'ingegnere della conoscenza ha piena familiarità con le tecniche di problem-solving e con i KRL, inoltre sa come mettere in forma utilizzabile le conoscenze dell'esperto. L'ingegnere della conoscenza può inizialmente non avere alcuna conoscenza del dominio del sistema. Ovviamente, per svolgere bene il suo compito è opportuno che acquisti una certa familiarità con l'area oggetto del sistema in modo da poter interagire al meglio con l'esperto umano.

Generalmente all'esperto vengono sottoposti dei problemi tipici ed egli spiega passo passo il processo di soluzione adottato. L'ingegnere ricava delle regole generali da queste spiegazioni e le verifica con l'esperto.

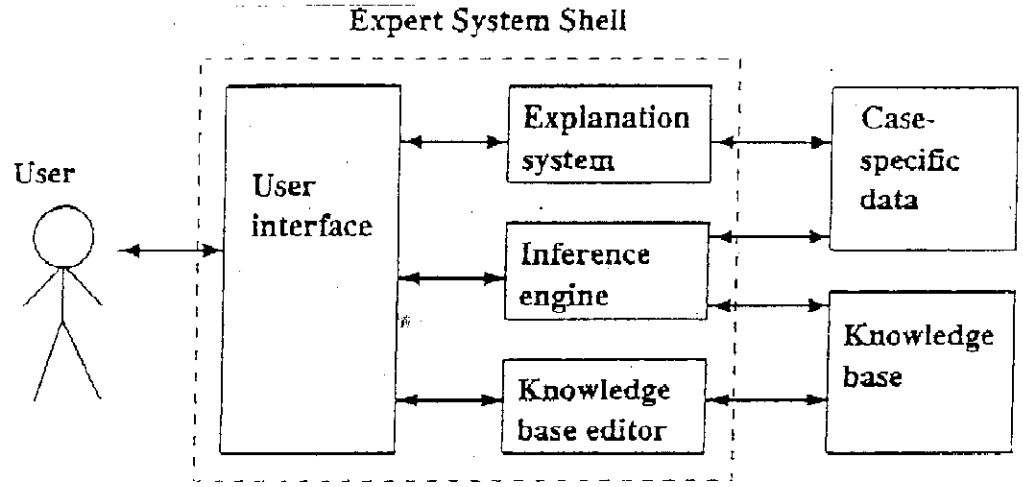
Come succede in molte applicazioni informatiche, il sistema sviluppato sarà pressocchè inutilizzato se l'utente finale non lo accetta di buon grado, per cui è importante che i futuri utenti vengano coinvolti nel processo di sviluppo.

Il ciclo di sviluppo del sistema parte da un prototipo iniziale seguito da verifiche e modifiche iterative da effettuare sia con gli esperti sia con i futuri utenti per assicurarsi che il sistema renda loro più facile e non più difficile la vita.

Nello sviluppo del prototipo iniziale l'ingegnere può prendere decisioni non definitive circa la rappresentazione della conoscenza e i metodi di inferenza. Al fine di verificare la validità di queste decisioni il prototipo può gestire solo una piccola parte del problema complessivo. Se le soluzioni adottate si rivelano appropriate si può arricchire il prototipo con tutto il resto della problematica da gestire.

Architettura di un sistema esperto

Mentre i dettagli di un sistema esperto possono differire molto tra loro, generalmente i sistemi hanno un'architettura complessiva simile:



L'utente interagisce con il sistema tramite un'*interfaccia* utente (ad esempio, menu o linguaggio naturale). Il *motore di inferenza* contiene i metodi di problem-solving ed effettua ragionamenti utilizzando la conoscenza presente nella *base di conoscenza* e i dati specifici del particolare problema da risolvere. Questi *dati specifici del caso* includono i dati forniti dall'utente e le conclusioni parziali basate su questi dati.

Ad esempio, consideriamo un sistema semplicissimo per la diagnosi del raffreddore. Potremmo usare un motore di inferenza di tipo backward chaining e regole del tipo

IF symptom (person, runny-nose) THEN disease (person, cold)

I dati specifici del caso potrebbero includere il fatto
symptom (fred, runny-nose).

Generalmente i sistemi esperti hanno anche un *sottosistema spiegazione* al fine di spiegare all'utente il ragionamento svolto. Alcuni sistemi hanno anche un *knowledge base editor* per facilitare gli interventi sulla base di conoscenza.

Una importante caratteristica dei sistemi esperti è il modo in cui generalmente *separano* la conoscenza specifica del dominio dalle tecniche di rappresentazione e di inferenza.

La parte general-purpose, tratteggiata in figura, può spesso essere riutilizzata per diversi problemi, e viene chiamata *expert system shell* o *toolkit*. Lo shell include il motore di inferenza, l'interfaccia utente, il sistema di spiegazione e a volte un editor.

Dato un nuovo tipo di problema da risolvere, spesso è possibile reperire uno shell che fornisce il supporto appropriato per quel problema, c'è solo da fornire la base di conoscenza.

Sono disponibili commercialmente numerosi shell, ciascuno appropriato per una classe di problemi. Utilizzando gli shell per costruire i sistemi esperti riduce tempi e costi di sviluppo.

Metodi di problem-solving

Un semplice sistema esperto può includere fatti del tipo

symptom (patient1, cough)

e regole del tipo *IF symptom (P, cough) THEN ADD disease (P, cold)*

Le regole di inferenza stabiliscono cosa può essere dedotto a partire dai fatti attuali. Un metodo di problem-solving è un modo di controllare le inferenze in modo che vengano effettuate solo quelle appropriate per il problema in esame. Se le inferenze venissero fatte a caso si farebbe molto lavoro inutile.

Questo è particolarmente importante quando il sistema pone domande all'utente per ricavare più informazioni specifiche (ad esempio, dopo aver trovato che il paziente ha la tosse, "Il paziente ha la gola infiammata ?"). Se le inferenze non fossero opportunamente controllate il sistema potrebbe chiedere "Il paziente ha il piede infiammato ?".

Sistemi esperti basati su regole e con backward chaining

I metodi FC e BC prima visti forniscono metodi di validità generale per il controllo delle inferenze. Vediamo ora come può essere costruito un semplicissimo sistema esperto basato su regole IF-THEN e avente un interprete di tipo BC.

Noi forniamo al sistema un'ipotesi e il sistema cerca di stabilire se essa è vera. All'utente vengono poste solo quelle domande che sono pertinenti all'ipotesi in esame. Abbiamo un insieme di malattie che il paziente potrebbe avere, il sistema considera una alla volta ciascuna ipotesi (ad esempio, *disease (fred, cold))* come goals da provare e cerca di stabilire se l'ipotesi è vera.

A volte il sistema non ha dati sufficienti per procedere nel ragionamento e pertanto può formulare delle domande (ad esempio, "Hai mal di testa ?") appartenenti a un insieme di domande *ammissibili* cioè gestibili dal sistema. Utilizzando i dati iniziali più le risposte alle domande il sistema cerca di individuare quale delle ipotesi è quella giusta.

L'algoritmo di tipo BC che segue viene ripetuto per ciascuna ipotesi G:

Per provare G:

- se G è presente nella memoria di lavoro è provata

- se G è ammissibile formula la domanda all'utente, registra la risposta come nuovo fatto, successo o fallimento a seconda della risposta

- altrimenti, cerca una regola che può essere usata per concludere G, e cerca di provare ogni precondizione della regola

- altrimenti, G fallisce

Esempio

Supponiamo che abbiamo le seguenti regole estremamente semplificate per diagnosticare le emergenze domestiche:

R1: If coughing THEN ADD smoky

R2: IF wet and NOT raining THEN ADD burst-pipe

R3: IF NOT coughing AND alarm-rings THEN ADD burglar

R4: IF smoky AND hot THEN ADD fire

Supponiamo di avere la sensazione che qualcosa non va e che le possibilità sono *fire, burst-pipe* e *burglar*. Queste sono le ipotesi fornite al sistema esperto.

Assumiamo che il sistema non ha fatti iniziali, per cui deve formulare domande per acquisire ulteriori informazioni. Assumiamo che le domande ammissibili sono le seguenti: *hot, coughing, wet, raining, alarm-rings*.

Innanzitutto il sistema cerca di provare *fire*. La R4 è potenzialmente utile per cui il sistema cerca di provare *smoky* e *hot*: *smoky* può essere dedotto mediante la R1 per cui cerchiamo di provare *coughing*. Possiamo pertanto porre la domanda:

Stai tossendo ?

Supponiamo che l'utente risponda negativamente. Questa risposta viene memorizzata, in caso sia utile in seguito. Allora l'ipotesi *fire* fallisce e il sistema passa all'ipotesi successiva: *burst-pipe*.
Usando la R2 si pone la domanda:

C'è una perdita d'acqua ?

Se l'utente risponde "no" anche l'ipotesi *burst-pipe* fallisce e il sistema passa a *burglar*.

Usando la R3 il sistema individua i subgoals *NOT coughing* e *alarm-rings*. L'utente ha già risposto che non ha tosse per cui la domanda non va formulata di nuovo. Il sistema chiede:

Sta suonando l'allarme ?

Se l'utente risponde affermativamente il sistema deduce che c'è stato uno scassinatore.

La sequenza di domande-risposte è la seguente:

Sistema: Stai tossendo ?
Utente: No

Sistema: C'è una perdita d'acqua ?
Utente: No

Sistema: Sta suonando l'allarme ?
Utente: Si

Sistema: Concludo che c'è stato un furto con scasso

In questo esempio c'è solo una regola che può essere usata in ciascuna deduzione. In generale le regole possono essere molte e dimostrare un'ipotesi comporta effettuare una *ricerca* di tutti i possibili modi che potrebbero condurre a provare l'ipotesi stessa. Le tecniche di ricerca saranno esaminate più avanti. Un sistema esperto realistico contiene centinaia di regole, ciascuna contenente variabili.

Esempio

Supponiamo di avere un sistema esperto per la consulenza finanziaria il quale abbia le regole semplificate già prima viste:

R1: IF NOT risparmi-adeguati THEN ADD investire-in-risparmi
R2: IF risparmi-adeguati AND reddito-adeguato THEN ADD
investire-in-azioni
R3: IF NOT avere-figli THEN ADD risparmi-adeguati
R4: IF avere-coniuge AND coniuge-lavora THEN ADD reddito-adeguato

Vediamo come un interprete di tipo backward chaining può essere usato per verificare le ipotesi "investire-in-risparmi" o "investire-in-azioni". Non vi sono fatti iniziali per cui il sistema deve porre domande.

Per provare l'ipotesi "investire-in-risparmi" la regola R1 è utile, però poiché R1 dipende da R3 il sistema cerca di provare "NOT avere-figli". La domanda è: "Hai figli ?". Se la risposta è SI' allora i risparmi non sono adeguati e per la R1 il sistema conclude che l'ipotesi "investire-in-risparmi" è provata, se la risposta è NO allora i risparmi sono adeguati per cui l'ipotesi "investire-in-risparmi" fallisce e si passa all'ipotesi "investire-in-azioni".

La regola R2 è utile, la prima precondizione (risparmi adeguati) è già stata provata per cui il sistema cerca di provare la seconda. La R4 lo induce a chiedere "Hai il coniuge ?": se la risposta è NO l'ipotesi fallisce, altrimenti il sistema chiede "Il coniuge lavora ?". Se la risposta è NO l'ipotesi "investire-in-azioni" fallisce, altrimenti l'ipotesi è provata.

Possibilità di spiegazioni

I sistemi esperti spesso permettono all'utente di chiedere *perché* il sistema ha chiesto un'informazione e *come* il sistema è giunto a una conclusione. Queste domande l'utente le formula riferendosi ai goals del sistema (ad esempio, provare *burst-pipe)*, alle regole applicate, a qualsiasi dato del problema. Per esemplificare il tipo di spiegazioni possibili vediamo un tipico dialogo sistema utente:

Sistema: Stai tossendo ?
Utente: No

Sistema: C'è una perdita d'acqua ?
Utente: No

Sistema: Sta suonando l'allarme ?
Utente: Perchè ?

Sistema: Sto cercando di provare che c'è stato un furto con scasso. E' stato stabilito che non stai tossendo. Pertanto, se l'allarme suona posso concludere che c'è stato un furto con scasso.
Sta suonando l'allarme ?
Utente: Si

Sistema: Concludo che c'è stato un furto con scasso
Utente: Come ?

Sistema: Questo si deduce dalla regola R3:
R3: IF NOT coughing AND alarm-rings THEN ADD burglar
Mi hai detto che non stai tossendo
Mi hai detto che sta suonando l'allarme

Per fornire spiegazioni occorre che il sistema memorizzi le regole applicate per effettuare deduzioni, al fine di comporre spiegazioni del tipo illustrato ora. Dare queste spiegazioni non è molto difficile però esse sono spesso giudicate insufficienti dagli utenti.

Una ragione è che le spiegazioni si riferiscono alla conoscenza *di superficie* presente nelle regole e non alla conoscenza *profonda* che ha motivato le regole ma che di solito non è rappresentata.

Ad esempio, il sistema dirà che ha concluso X perché c'è la regola 23, ma non spiegherà il motivo per cui è presente la regola 23. Ad esempio, in un sistema per diagnosticare i guasti nelle automobili le motivazioni delle regole sono i principi di funzionamento delle auto, a meno che questa conoscenza sia accessibile all'explanation system, qualsiasi spiegazione non potrà essere pienamente convincente.

Un altro motivo per cui l'explanation system spesso è inadeguato è che se l'utente non è convinto della spiegazione il sistema non è in grado di formulargliela in altro modo. La generazione di spiegazioni è un'area di ricerca importante che fa parte dell'area più ampia che investiga come giungere a una comunicazione efficiente sistema-utente. Accenneremo in seguito ai problemi che riguardano l'elaborazione del linguaggio naturale.

Sommario

I *sistemi esperti* sono utilizzati per fornire prestazioni simili a quelle di esperti umani.

Problemi appropriati sono quelli in cui i benefici superano i costi, collaborazione attiva di esperti umani è disponibile, gli utenti sono pronti ad accettare la nuova tecnologia, e il problema stesso è altamente specializzato.

La costruzione di un sistema esperto, cioè *l'ingegneria della conoscenza*, richiede l'estrazione della conoscenza degli esperti umani, la individuazione di appropriati metodi per la rappresentazione della conoscenza e per il problem-solving, la creazione di un prototipo e una fase di testing e affinamento delle prestazioni del sistema.

Un tipico sistema esperto include i seguenti moduli: *base di conoscenza, motore di inferenza, dati specifici del caso, sistema per le spiegazioni, interfaccia utente, editor della base di conoscenza.*

Un semplice sistema esperto può essere sviluppato utilizzando regole e un motore di tipo backward chaining. Il sistema cerca di provare ciascuna delle ipotesi formulando domande all'utente.

4. La gestione della conoscenza incerta

Finora nei casi esaminati abbiamo sempre assunto che tutta la conoscenza fosse certa. Per esempio, abbiamo detto che se fa caldo e c'è fumo allora c'è un incendio. Però nella maggior parte delle applicazioni reali le cose tendono ad essere vaghe. Se fa caldo e c'è fumo potrebbe esserci un incendio, ma potrebbe anche trattarsi di una festa con molti fumatori. Se tossisci e ti brucia la gola potresti avere il raffreddore, ma potresti anche essere un ospite di quella festa.

Pertanto la maggior parte dei sistemi esperti deve avere il modo di esprimere che qualcosa è probabilmente, ma non necessariamente, vero. O anche che alcuni dati (ad esempio, sintomi) sono usualmente, ma non sempre, associati ad una certa causa (malattia).

Nel seguito vedremo, a titolo di esempio, un'applicazione, nel campo della diagnosi medica, basata sul teorema di Bayes. Questo modo di procedere, basato su risultati ben noti del calcolo delle probabilità, può essere utilizzato per costruire semplici sistemi esperti.

Richiami di calcolo delle probabilità

Probabilità a priori

Vi sono diverse interpretazioni per il concetto di probabilità. Noi qui ci limiteremo a dire che supponiamo di poter assegnare a un certo evento A il valore della *probabilità a priori* che si verifichi, P(A).

Negli esempi che faremo, quando l'esempio riguarda eventi quali il lancio di una moneta o di un dado oppure la scelta a caso di un elemento di un insieme, la probabilità a priori la consideriamo data dal rapporto fra il numero di casi favorevoli al verificarsi dell'evento diviso il numero di casi possibili.

Invece quando consideriamo, ad esempio, la probabilità di una malattia, il valore della probabilità viene derivato da dati statistici o, più in generale, dal grado di credenza che si ha nel verificarsi dell'evento.

Probabilità condizionale

La *probabilità condizionale* di un evento A dato l'evento H è definita come segue:

$$P(A/H) = \frac{P(A \wedge H)}{P(H)}$$

Nel caso in cui gli eventi possibili sono equiprobabili, abbiamo:

$$P(A/H) = \frac{\text{n. casi favorevoli per } A \wedge H}{\text{n. casi possibili}} \;/\; \frac{\text{n. casi favorevoli per H}}{\text{n. casi possibili}} =$$

$$= \frac{\text{n. casi favorevoli per } A \wedge H}{\text{n. casi possibili}} * \frac{\text{n. casi possibili}}{\text{n. casi favorevoli per H}} =$$

$$= \frac{\text{n. casi favorevoli per } A \wedge H}{\text{n. casi favorevoli per H}}$$

Esempio 1

Supponiamo di avere un insieme costituito da 100 persone di cui 40 uomini e 60 donne. Le probabilità a priori, scegliendo a caso, sono:

$P(M) = 40/100 = 0.4$
$P(F) = 60/100 = 0.6$

Supponiamo di sapere che 10 fra le 100 persone sono daltoniche (8 uomini e 2 donne), per cui:

$P(D) = 10/100 = 0.1$

La probabilità di scegliere un uomo daltonico è data da:
$P(M \wedge D) = 8/100 = 0.08$

La probabilità di una donna daltonica è:
$P(F \wedge D) = 2/100 = 0.02$

Vogliamo ora calcolare quanto vale la probabilità condizionale di scegliere una persona daltonica dato che venga scelto un maschio:

$$P(D/M) = \frac{P(D \wedge M)}{P(M)} = \frac{8/100}{40/100} = \frac{8}{100}\,\frac{100}{40} = \frac{8}{40} = \frac{2}{10} = 0.2$$

Questo valore può anche essere ricavato pensando che, una volta che ci restringiamo agli uomini, i casi possibili sono 40 e i casi favorevoli sono 8.

Come si vede la P(D/M) è più del doppio della $P(D \wedge M)$ in quanto ora il numero di casi possibili è sceso da 100 a 40.

Analogamente possiamo calcolare la probabilità condizionale:

$$P(D/F) = \frac{P(D \wedge F)}{P(F)} = \frac{2/100}{60/100} = \frac{2}{100}\,\frac{100}{60} = \frac{2}{60} = 0.0333\ldots$$

Anche il valore 2/60 può essere facilmente ricavato pensando che, limitandoci alle donne, 60 sono i casi possibili e solo 2 i favorevoli.

Esempio 2

Viene lanciata una moneta tre volte. Vogliamo calcolare la probabilità che esca testa tutte e tre le volte sapendo:

a) niente

b) che la prima volta è uscito testa

c) che in almeno uno dei lanci è uscito testa

Denotando con A l'evento "tre volte testa" abbiamo:

a) la probabilità a priori $P(A) = 1/8$

in quanto i casi possibili sono [TTT, TTC, TCT, TCC, CTT, CTC, CCT, CCC] mentre abbiamo il solo caso favorevole [TTT]

b) se E è l'evento "la prima volta testa" allora

$$P(A/E) = \frac{P(A \wedge E)}{P(E)} = \frac{1/8}{1/2} = 1/4$$

infatti per quanto riguarda $P(A \wedge E)$ l'evento $A \wedge E$ denota il verificarsi sia di A che di E: i casi favorevoli per A si riducono alla sequenza TTT, i casi favorevoli per E sono TTT, TTC, TCT, TCC, l'intersezione $A \wedge E$ si riduce alla sola sequenza TTT, per cui casi favorevoli 1, casi possibili 8. Invece per P(E) caso favorevole 1, casi possibili 2.

Anche in questo esempio al valore 1/4 di P(A/E) si può arrivare pensando che i casi possibili sono quattro (TTT, TTC, TCT, TCC) e il caso favorevole uno solo. In altri termini, se mi sono assicurato testa al primo lancio, per realizzare l'evento A è sufficiente che esca testa agli altri due e questo evento ha probabilità 1/4.

c) se H è l'evento "almeno uno dei lanci è testa" allora

$$P(A/H) = \frac{P(A \wedge H)}{P(H)} = \frac{1/8}{7/8} = 1/7$$

Infatti i casi favorevoli per l'evento H sono [TTT, TTC, TCT, TCC, CTT, CTC, CCT] per cui i favorevoli sono 7.

I casi favorevoli per l'evento $A \wedge H$ si ottengono intersecando il favorevole ad A (TTT) con i sette favorevoli ad H e si ottiene il solo TTT.

Poiché l'evento H appare molto simile ad E, sembra strano che P(A/H) sia quasi la metà di P(A/E): in effetti supponendo che si sia verificato E i casi possibili per A sono solo 4, invece il verificarsi di H porta solo ad escludere il caso CCC e i casi possibili per A scendono da 8 a 7.

Questo esempio mostra come il calcolo delle probabilità a volte produca risultati che a prima vista vanno contro il senso comune.

Esempio 3

Supponiamo di lanciare una coppia di dadi. Assumiamo che la somma dei dadi sia 6, calcoliamo la probabilità che almeno uno dei dadi valga 2.

In altri termini, avendo l'evento

E = [la somma è 6] = [(1,5),(2,4),(3,3),(4,2),(5,1)]

e l'evento A = [un 2 si presenta su almeno un dado]

vogliamo calcolare $P(A/E) = P(A \wedge E)/P(E)$

I casi possibili sono le 36 coppie :
[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(2,1),(2,2),(2,3),.....]

I casi favorevoli per l'evento E sono 5 per cui
P(E) = 5/36

L'evento A consta di 11 coppie e cioè:
[(2,1),(2,2),(2,3),(2,4),(2,5),(2,6),(1,2),(3,2),(4,2),(5,2),(6,2)]

Intersecando A con E si ottiene
$A \wedge E = [(2,4),(4,2)]$

per cui $P(A \wedge E) = 2/36$ e infine

$$P(A/E) = P(A \wedge E) / P(E) = \frac{2}{36} / \frac{5}{36} = \frac{2}{36} * \frac{36}{5} = \frac{2}{5}$$

Vogliamo ora calcolare $P(\neg A/E)$. Ricordando che per ogni evento X, $P(X) + P(\neg X) = 1$ potremmo subito dedurre che:

$P(\neg A/E) = 1 - P(A/E) = 1 - 2/5 = 3/5$

Possiamo anche calcolare esplicitamente la probabilità:
L'evento $\neg A$ è [non si presenta un 2 su almeno un dado]

e consta delle 25 coppie [(1,1),(1,3),(1,4),(1,5),(1,6),(3,1),(3,3),(3,4),....]

Intersecando $\neg A$ con E otteniamo le tre coppie [(1,5),(3,3),(5,1)]

$P(\neg A \wedge E) = 3/36$ e quindi:

$P(\neg A/E) = P(\neg A \wedge E) / P(E) = (3/36) / (5/36) = 3/5$

Infine ci possiamo chiedere quanto vale $P(A/ \neg E) = P(A \wedge \neg E) / P(\neg E)$

$\neg E$ è l'evento "la somma non vale 6" ed include le 31 coppie [(1,1),(1,2),(1,3),(1,4),(1,6),(2,1),(2,2),.....]

per cui $P(\neg E) = 31/36$

D'altra parte l'evento $A \wedge \neg E$ è
[esce almeno un 2 AND la somma non vale 6]

per cui intersecando si hanno le 9 coppie:
[(2,1),(2,2),(2,3),(2,5),(2,6),(1,2),(3,2),(5,2),(6,2)] e abbiamo:

$P(A \wedge \neg E) = 9/36$ e infine:

$P(A/ \neg E) = P(A \wedge \neg E) / P(\neg E) = (9/36) / (31/36) = 9/31$

Il teorema di Bayes

Ricordiamo che

$P(A/B) = P(A \wedge B)/ P(B)$

$P(B/A) = P(B \wedge A) / P(A) = P(A \wedge B)/ P(A)$

Da cui:

$P(A \wedge B) = P(A/B)\ P(B)$

$P(B \wedge A) = P(B/A)\ P(A)$

Uguagliando le parti destre e dividendo per P(A) otteniamo:

$$P(B/A) = \frac{P(A/B)\ P(B)}{P(A)}$$

Questo è il *teorema di Bayes* ed è alla base di tutti i moderni sistemi di AI per l'inferenza probabilistica.

A prima vista la regola di Bayes non sembra molto utile. Ha bisogno di tre termini, una probabilità condizionale e due a priori, per calcolare una sola probabilità condizionale. Invece la regola si rivela utile in quanto vi sono molti casi in cui possiamo stimare agevolmente queste tre quantità.

Ad esempio, nella diagnosi medica abbiamo spesso disponibili le probabilità di vari sintomi in relazione alle malattie e vorremmo derivare una diagnosi.

Eventi incompatibili e necessari

Due eventi sono detti *incompatibili* o *mutuamente esclusivi* se la probabilità che si verifichino entrambi è uguale a zero.

Ad esempio, l'evento H_1 = [lanciando un dado esce un numero pari] e l'evento H_2 = [lanciando un dado esce un numero dispari] sono incompatibili e la $P(H_1 \wedge H_2) = 0$.

Se abbiamo due eventi incompatibili, la probabilità che si verifichi l'evento unione è la somma delle singole probabilità.

Ad esempio, $P(H_1 \vee H_2) = P(H_1) + P(H_2) = 1/2 + 1/2 = 1$

Un insieme di eventi è detto *necessario* o *esaustivo* se la probabilità dell'evento unione è uguale a 1.

Ad esempio, $P(H_1 \vee H_2) = 1$

Pertanto H_1 e H_2 sono sia mutuamente esclusivi sia necessari.

Se invece H_1 denota [esce 1 o 2] e H_2 denota [esce 3 o 4] gli eventi sono mutuamente esclusivi ma non esaustivi, infatti:

$P(H_1 \vee H_2) = P(H_1) + P(H_2) = 1/3 + 1/3 = 2/3$

Possiamo ora ottenere una semplice formula di grande utilità. Supponiamo di avere l'insieme necessario di eventi incompatibili $H_1, H_2, \ldots, H_n$.

Pertanto un qualsiasi evento X può accadere solo in congiunzione con un H_i, cioè:

$$X = (X \wedge H_1) \vee (X \wedge H_2) \vee (X \wedge H_3) \vee \ldots \vee (X \wedge H_n)$$

Poichè gli eventi $X \wedge H_i$ sono incompatibili le loro probabilità si sommano per cui

$$P(X) = P[(X \wedge H_1) \vee (X \wedge H_2) \vee (X \wedge H_3) \vee \ldots \vee (X \wedge H_n)] =$$

$$= P(X \wedge H_1) + P(X \wedge H_2) + P(X \wedge H_3) + \ldots + P(X \wedge H_n)$$

Ricordando che $P(X \wedge H) = P(X/H)\, P(H)$ abbiamo:

$$P(X) = P(X/H_1)\, P(H_1) + P(X/H_2)\, P(H_2) + \ldots + P(X/H_n)\, P(H_n)$$

Esempio

Sia A l'evento "lanciando un dado esce 3". Chiaramente:

$P(A) = 1/6$

Consideriamo ora gli eventi necessari e incompatibili:

H_1 = [esce un numero pari]

H_2 = [esce un numero dispari]

abbiamo che:

$P(H_1) = 1/2$

$P(H_2) = 1/2$

$P(A/ H_1) = 0$ in quanto se è uscito un pari il numero non può essere il 3

$P(A/ H_2) = 1/3$ in quanto se è uscito un dispari i casi possibili sono tre (1, 3, 5) di cui uno solo favorevole.

Pertanto, applicando la formula abbiamo:

$P(A) = P(A/H_1)\, P(H_1) + P(A/H_2)\, P(H_2) = 0 * 1/2 + 1/3 * 1/2 = 1/6$

Cosa fare in presenza di incertezza.

Supponiamo che un sistema esperto abbia il compito di consigliarci il comportamento da adottare dovendo prendere un aereo. Supponiamo che il piano migliore sia il piano A_{90} che comporta lasciare casa 90 minuti prima della partenza del volo guidando a una velocità ragionevole.

Anche se l'aeroporto è a soli 20 km di distanza, il sistema non potrà arrivare a una conclusione certa del tipo "Il piano A_{90} ti farà arrivare in tempo all'aeroporto" ma solo alla conclusione "Il piano A_{90} ti farà arrivare in tempo all'aeroporto a meno di imprevisti più o meno catastrofici".

Il problema è decidere se la cosa migliore da fare è adottare il piano A_{90} oppure optare, ad esempio, per il piano A_{120} che offre maggiori garanzie di arrivare in tempo all'aeroporto però con un probabile maggior tempo di attesa.

Pertanto quando si parla di scelta migliore si intende che di tutti i possibili piani che potrebbero essere eseguiti, ci si aspetta che A_{90} massimizzi la misura della prestazione richiesta che comprende non solo arrivare in tempo all'aeroporto, ma anche evitare una lunga e inutile attesa, evitare multe per eccesso di velocità nel tragitto, e così via.

L'informazione a disposizione non può garantire nessuna di queste cose per il piano A_{90} ma può fornire una *probabilità* che verranno ottenute. La cosa giusta da fare, la *decisione razionale,* dipende dunque sia dall'importanza relativa dei vari obiettivi, dalla loro probabilità e dal grado al quale verranno raggiunti.

Esempio.

Al fine di chiarire la natura dei problemi da affrontare facciamo un semplice esempio di diagnosi. La diagnosi, sia che si tratti di medicina, di riparazioni di apparecchiature, o di altro tipo, è un compito che quasi sempre comporta incertezza. Se tentassimo di costruire un sistema di diagnosi dentale usando la logica del primo ordine, potremmo proporre regole del tipo

$\forall$ X sintomo (X, maldidenti) $\rightarrow$ *malattia (X, carie)*

Il problema è che questa regola è sbagliata. Non tutti i pazienti con mal di denti hanno carie, pertanto la regola andrebbe modificata come segue:

$\forall$ X sintomo (X, maldidenti) $\rightarrow$ malattia (X, carie) $\vee$ malattia (X, gengivite) $\vee$ malattia (X, piorrea) $\vee$ .....

Bisognerebbe però introdurre un lista lunghissima di possibili malattie. Potremmo tentare di trasformare la regola:

$\forall$ X malattia (X, carie) $\rightarrow$ sintomo (X, maldidenti)

Ma anche questa regola è sbagliata in quanto non tutte le carie sono causa di dolore. L'unico modo di aggiustare questa regola è di estendere il lato sinistro affinché copra tutte le possibili malattie che causano il mal di denti.

Il tentativo di usare la logica del primo ordine per gestire il dominio della diagnosi medica fallisce per tre motivi principali:

- Pigrizia: l'elencazione dell'insieme di tutti gli antecedenti o conseguenti necessari per garantire una regola senza eccezioni è troppo laboriosa e, la regola risultante, troppo difficile da utilizzare

- Ignoranza teorica: la scienza medica non ha una teoria completa per il dominio

- Ignoranza pratica: anche se conoscessimo tutte le regole, potremmo essere incerti a proposito di un paziente particolare perché tutti i test necessari non sono stati fatti o non è stato possibile farli.

La connessione tra il mal di denti e la carie non è semplicemente una implicazione logica nè in un verso nè nell'altro. Questo è tipico dei domini medici, ma anche di quei domini che coinvolgono il giudizio: legge, affari, progettazione, riparazioni, etc.

Si può ricavare solo un *grado di credenza* nella formula rilevante. Il nostro strumento principale per trattare i gradi di credenza sarà la teoria della probabilità, che assegna alle formule un valore numerico compreso tra zero e uno.

La probabilità fornisce un modo per riassumere l'incertezza che deriva dalla pigrizia e dall'ignoranza.

I valori delle probabilità condizionali possono essere ricavati o dagli esperti o dai dati disponibili. Ad esempio, supponiamo di voler calcolare la probabilità di un infarto in presenza di un dolore al braccio. Possiamo chiedere a un medico in base alla sua esperienza oppure raccogliere dati su pazienti che hanno avuto un dolore al braccio e vedere quanti di essi hanno ricevuto una diagnosi di infarto.

Nei problemi di diagnosi medica noi spesso abbiamo disponibili dati sulle probabilità di vari sintomi in rapporto alle malattie, ma non abbiamo invece le probabilità delle malattie dati i sintomi.

Ad esempio, se, su cento infartuati, 70 hanno avuti dolori al braccio possiamo dire che, approssimativamente, P(dolore/infarto) = 7/10. Sarebbe però utile sapere quanto vale P(infarto/dolore) e questo è più complicato in quanto è più facile ottenere dati sui sintomi delle persone che hanno avuto un infarto; anziché sulle malattie presenti in persone che hanno avuto dolori al braccio.

Applicazione del teorema di Bayes

Supponiamo che un dottore sappia che la meningite induce nel malato, nel 50% dei casi, un irrigidimento del collo (IC). Il dottore sa anche che la probabilità a priori che un paziente abbia la meningite è 1/50,000 mentre la probabilità a priori che un paziente abbia IC è 1/20. Sia C la proposizione che dice che il paziente ha IC e M la proposizione che il paziente ha la meningite. Abbiamo:

$P(C/M) = 1/2$

$P(M) = 1/50{,}000$

$P(C) = 1/20$

$$P(M/C) = \frac{P(C/M)\,P(M)}{P(C)} = \frac{1/2 \quad 1/50.000}{1/20} = 1/5000 = 0.0002$$

Pertanto ci si aspetta che solo un paziente su 5000 con IC abbia la meningite. Si noti che nonostante che un IC è fortemente suggerito dalla meningite (probabilità 0.5) la probabilità di meningite rimane molto piccola. Questo perché la probabilità a priori di IC è molto più alta di quella di meningite.

Una domanda ovvia da porsi è perché si dovrebbe disporre di una probabilità condizionata, ma non dell'altra. Nel caso della meningite forse il dottore sa che un paziente su 5000 con un IC ha la meningite, quindi non ha bisogno della regola di Bayes. Sfortunatamente la conoscenza diagnostica è spesso molto più vaga di quella causale.

Se vi è un'improvvisa epidemia di meningite, la probabilità incondizionata della meningite, P(M), crescerà. Il dottore che derivasse P(M/C) da osservazioni statistiche dei pazienti prima dell'epidemia non saprà come aggiornare il valore, ma il dottore che calcoli P(M/C) dagli altri tre valori vedrà che P(M/C) crescerà proporzionalmente con P(M). In altri termini, l'informazione causale P(C/M) *non* è affetta dall'epidemia, poiché riflette semplicemente il modo di funzionare della meningite.

Verosimiglianza relativa

Consideriamo di nuovo la regola di Bayes per la meningite e IC:

$$P(M/C) = \frac{P(C/M)\,P(M)}{P(C)}$$

Supponiamo che siamo anche interessati alla possibilità che il paziente abbia un IC dovuto a un colpo di frusta (F):

$$P(F/C) = \frac{P(C/F)\,P(F)}{P(C)}$$

Confrontando queste equazioni possiamo calcolare la *verosimiglianza relativa* della meningite e del colpo di frusta in presenza di IC.

La verosimiglianza relativa è data dal rapporto

$$\frac{P(M/C)}{P(F/C)}$$

e pertanto, supponendo che P(C/F)=8/10 e P(F)=1/1000, vale:

$$\frac{P(M/C)}{P(F/C)} = \frac{P(C/M)\,P(M)}{P(C)} \;/\; \frac{P(C/F)\,P(F)}{P(C)} =$$

$$= \frac{P(C/M)\,P(M)}{P(C)} * \frac{P(C)}{P(C/F)\,P(F)} = \frac{P(C/M)\,P(M)}{P(C/F)\,P(F)} = \frac{1/2 * 1/50{,}000}{8/10 * 1/1000} =$$

$$= \frac{10^{-5}}{8\ 10^{-4}} = \frac{10^{-1}}{8} = 1/80$$

Pertanto il colpo di frusta è 80 volte più probabile della meningite, in presenza di IC. E' importante notare che per calcolare la verosimiglianza relativa non abbiamo bisogno della probabilità a priori dell'IC.

Normalizzazione.

La verosimiglianza relativa ci dice solo quanto è più o meno probabile una malattia rispetto ad un'altra in presenza di un certo sintomo. In alcuni casi la verosimiglianza relativa è sufficiente per prendere decisioni, ma quando, come nell'esempio ora visto, le due possibili malattie (meningite e colpo di frusta) portano a terapie completamente diverse, si ha bisogno del valore preciso di P(M/C) per poter prendere decisioni razionali.

Osserviamo che è ancora possibile non valutare direttamente la probabilità a priori dei "sintomi" (nel nostro esempio, P(C)), considerando un insieme di eventi mutuamente esclusivi ed esaustivi, cioè tali che la probabilità dell'evento unione sia uguale a 1. In tal caso, infatti, abbiamo visto che vale la formula:

$$P(X) = P(X/H_1)\,P(H_1) + P(X/H_2)\,P(H_2) + \ldots + P(X/H_n)\,P(H_n)$$

Nel caso in cui siamo in grado di ottenere il valore di $P(C/\neg M)$, poiché M e $\neg$ M sono eventi esaustivi e incompatibili, possiamo applicare la formula ora vista e otteniamo:

$$P(M/C) = \frac{P(C/M)\,P(M)}{P(C)} =$$

$$= \frac{P(C/M)\,P(M)}{P(C/M)\,P(M) + P(C/\neg M)\,P(\neg M)}$$

Possiamo arrivare a questo risultato anche senza utilizzare esplicitamente la formula per P(X). Infatti, sempre nell'ipotesi che siamo in grado di ottenere il valore di $P(C/\neg M)$, applichiamo la regola di Bayes per M e $\neg$ M e otteniamo:

$$P(M/C) = \frac{P(C/M)\,P(M)}{P(C)} \qquad (*)$$

$$P(\neg M/C) = \frac{P(C/\neg M)\,P(\neg M)}{P(C)}$$

sommando ed eguagliando parti sinistre e destre otteniamo:

$$P(M/C) + P(\neg M/C) = \frac{P(C/M)\,P(M)}{P(C)} + \frac{P(C/\neg M)\,P(\neg M)}{P(C)}$$

ricordando che:

$$P(M/C) + P(\neg M/C) = 1$$

abbiamo:

$$1 = \frac{P(C/M)\,P(M) + P(C/\neg M)\,P(\neg M)}{P(C)}$$

da cui:

$$P(C) = P(C/M)\,P(M) + P(C/\neg M)\,P(\neg M)$$

Sostituiamo questo valore di P(C) nella (*) e otteniamo:

$$P(M/C) = \frac{P(C/M)\,P(M)}{P(C/M)\,P(M) + P(C/\neg M)\,P(\neg M)}$$

Questo procedimento si chiama *normalizzazione* poichè tratta 1/P(C) come costante di normalizzazione che permette ai termini al numeratore di dare 1 come somma.

Pertanto, se conosciamo il valore di $P(C/\neg M)$, possiamo evitare di valutare P(C) ed ottenere comunque i valori di P(M/C).

Esempio

Torniamo al caso della meningite e dell'IC. Al fine di semplificare i calcoli facciamo l'assunzione irrealistica che P(M) = 1/50 e P(C) = 1/10. Con queste ipotesi ricaviamo, nello stesso modo di prima:

P(C/M) = 1/2

P(M) = 1/50

P(C) =1/10

$$P(M/C) = \frac{P(C/M)\,P(M)}{P(C)} = \frac{1/2 * 1/50}{1/10} = \frac{1/100}{1/10} = 1/10$$

Supponiamo ora di non conoscere P(C) ma di sapere che P(C/ ¬M) = 1/12
Inoltre possiamo subito ottenere P(¬M) = 1 – P(M) = 1 – 1/50 = 49/50

Allora:

$$P(M/C) = \frac{P(C/M)\,P(M)}{P(C)} = \frac{1/2 * 1/50}{P(C)} = \frac{1/100}{P(C)} \qquad (**)$$

$$P(\neg M/C) = \frac{P(C/\neg M)\,P(\neg M)}{P(C)} = \frac{1/12 * 49/50}{P(C)} = \frac{49/600}{P(C)}$$

Ma abbiamo visto prima che:

P(C) = P(C/M) P(M) + P(C/¬ M) P(¬ M) per cui abbiamo:

P(C) = 1/2 * 1/50 + 1/12 * 49/50 = 1/100 + 49/600 = 55/600

Sostituiamo questo valore di P(C) nella (**) e otteniamo:

$$P(M/C) = \frac{P(C/M)\,P(M)}{P(C/M)\,P(M) + P(C/\neg M)\,P(\neg M)} = \frac{1/100}{55/600} = 6/55$$

Ovviamente, $P(\neg M/C) = 1 - P(M/C) = 1 - 6/55 = 49/55$

Possiamo anche calcolare esplicitamente questo valore e abbiamo:

$$P(\neg M/C) = \frac{P(C/\neg M)\,P(\neg M)}{P(C/M)\,P(M) + P(C/\neg M)\,P(\neg M)} = \frac{49/600}{55/600} = 49/55$$

Osserviamo che il valore calcolato per $P(C) = 55/600$ è minore del valore che abbiamo attribuito inizialmente a $P(C) = 1/10$.

In effetti, in questo esempio, a differenza del lancio di monete o dadi, abbiamo supposto valori di probabilità attribuiti statisticamente o empiricamente e pertanto i valori delle probabilità possono essere fra loro leggermente diversi.

Sommario

Nella maggior parte delle applicazioni reali un sistema esperto deve essere in grado di rappresentare e gestire la *conoscenza incerta* in quanto spesso le informazioni sono vaghe e incomplete.

Una varietà di tecniche sono state sviluppate, molte di esse sono basate su tecniche probabilistiche. Il concetto di *probabilità condizionale* risulta utile per ricavare il *teorema di Bayes,* ampiamente utilizzato per effettuare inferenze probabilistiche.

Un sistema esperto per la diagnosi medica può utilizzare il teorema di Bayes per ricavare, a partire dalle probabilità dei sintomi, la probabilità di una malattia.

E' anche possibile calcolare la *verosimiglianza relativa* di due malattie in presenza di un sintomo. Inoltre la tecnica della *normalizzazione* consente di avere valori più precisi per la probabilità di una malattia in relazione a un sintomo.

5. Tecniche di ricerca per il problem-solving

Vediamo ora come possiamo usare tecniche di *ricerca* per trovare la soluzione di un problema. Ad esempio, supponiamo di voler andare da un punto a un altro in una città. Possiamo cercare tutti i possibili percorsi e poi magari scegliere il più breve.

Ancora, supponiamo di voler ricostruire un puzzle, sappiamo che una soluzione esiste ed è unica. Sappiamo anche che le mosse necessarie (mettere i tasselli) sono semplici e fattibili, non sappiamo però dove mettere ciascun tassello. Possiamo allora dire: prova tutte le possibili configurazioni dei tasselli finchè non trovi quella giusta che ti ricostruisce il puzzle.

L'idea generale è che se conosci le azioni di base da intraprendere per risolvere il problema, ma non sai quali specificamente vanno applicate per arrivare alla soluzione, puoi ispezionare tutte le possibilità per trovare quella che ti fornisce la soluzione.

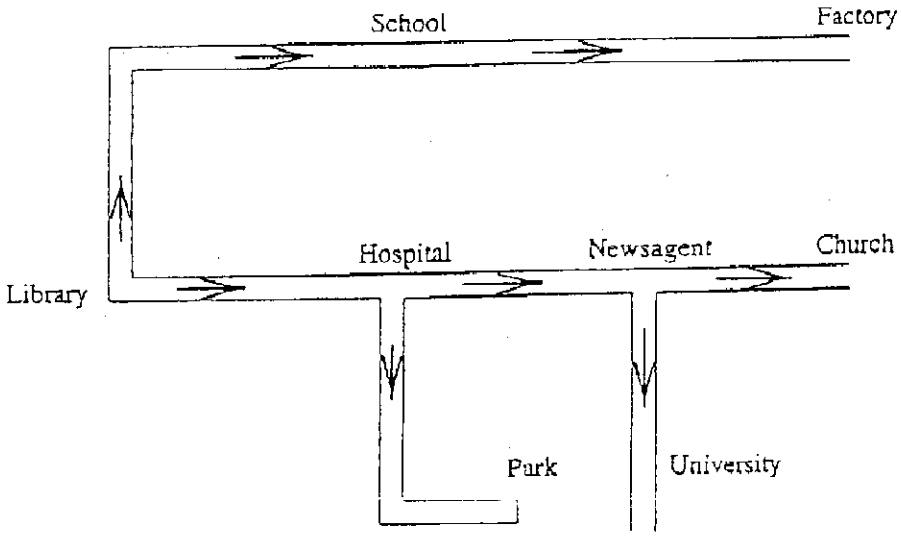
La nozione di ricerca si applica a moltissimi tipi di problemi. Quando abbiamo parlato di backward chaining abbiamo accennato alla necessità di una strategia di ricerca che analizzi le possibili scelte quando c'è da provare un goal. Questo richiede la ricerca di una sequenza di passi di inferenza che costituirà la prova di un certo goal.

Un altro esempio può essere la individuazione delle azioni necessarie per effettuare la spesa settimanale. Ancora, possiamo chiedere quale sequenza di mosse ci porta a vincere giocando a dama o scacchi (qui le cose sono più complicate perché abbiamo un avversario che gioca contro).

In genere questi problemi possono essere espressi in questi termini: partendo da uno *stato iniziale* (punti della città, puzzle da ricostruire, goal da provare, frigorifero vuoto, situazione di gioco) come arrivare a uno *stato finale (target state)* e cioè percorso individuato, puzzle ricostruito, goal provato, frigo pieno, vincita al gioco. Noi vogliamo effettuare una ricerca attraverso tutte le azioni possibili al fine di individuare una sequenza che ci porta dallo stato iniziale allo stato finale.

Tecniche di ricerca

Supponiamo che, in auto, stiamo cercando la strada giusta in una piccola città con molti sensi unici:



Questa città ha la spiacevole caratteristica che una volta lasciato un posto non puoi tornare indietro e inoltre se arrivi nel parco ci resti per sempre.

Supponiamo che vogliamo individuare un possibile percorso dalla biblioteca (stato iniziale) all'università (stato finale). Si vede subito che l'unico percorso passa per l'ospedale e l'edicola. Se abbiamo una situazione reale molto più complessa, come possiamo effettuare la ricerca del percorso ?

Nel nostro esempio è semplice cercare sistematicamente ed esaustivamente tutti i percorsi. Ovvero noi controlliamo ogni stato raggiungibile a partire dallo stato iniziale per vedere se è finale. L'insieme di tutti gli stati è lo *spazio di ricerca.*

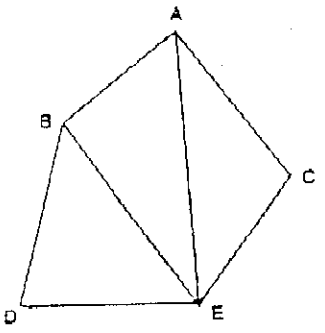
Se lo spazio di ricerca è piccolo, semplici tecniche che provano ogni possibile percorso in modo sistematico sono appropriate. Tali tecniche sono dette *forza bruta* o *cieche* e includono la *breadth first search* e la *depth first search* che vedremo tra poco.

Quando si ha a che fare con problemi complessi lo spazio di ricerca può diventare enorme e non è possibile esplorare tutti gli stati in un tempo ragionevole. Per gestire questi problemi occorrono tecniche *euristiche*, basate sul buon senso, per individuare quali percorsi hanno più probabilità di condurci alla soluzione.

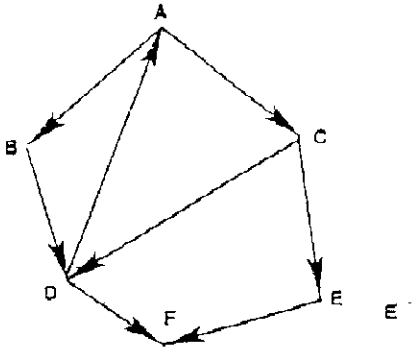
Grafi e alberi

Gli algoritmi di ricerca che andiamo a descrivere sono molto generali e si applicano a tutti i tipi di problemi. Pertanto è opportuno rappresentare i problemi e gli algoritmi in modo astratto in modo da non dover riformulare l'algoritmo per ogni tipo di problema. A tal fine ci serviremo dei grafi.

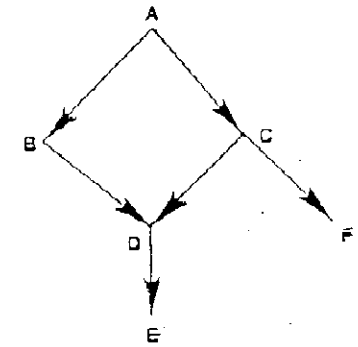
Un *grafo* è un insieme di *nodi* con *archi* che li collegano. Gli archi possono essere direzionati o non direzionati. Dato un nodo, il termine *successore* è riferito a un nodo adiacente collegato con un arco. Dati due nodi, un *percorso* è una sequenza di nodi che portano dall'uno all'altro nodo via archi. Un grafo *aciclico* non ha cicli, cioè percorsi che uniscono un nodo con se stesso.



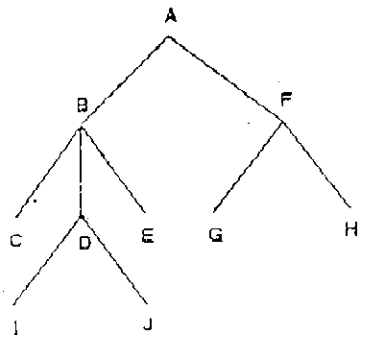
A (labelled) graph.



A directed graph.



An acyclic directed graph.



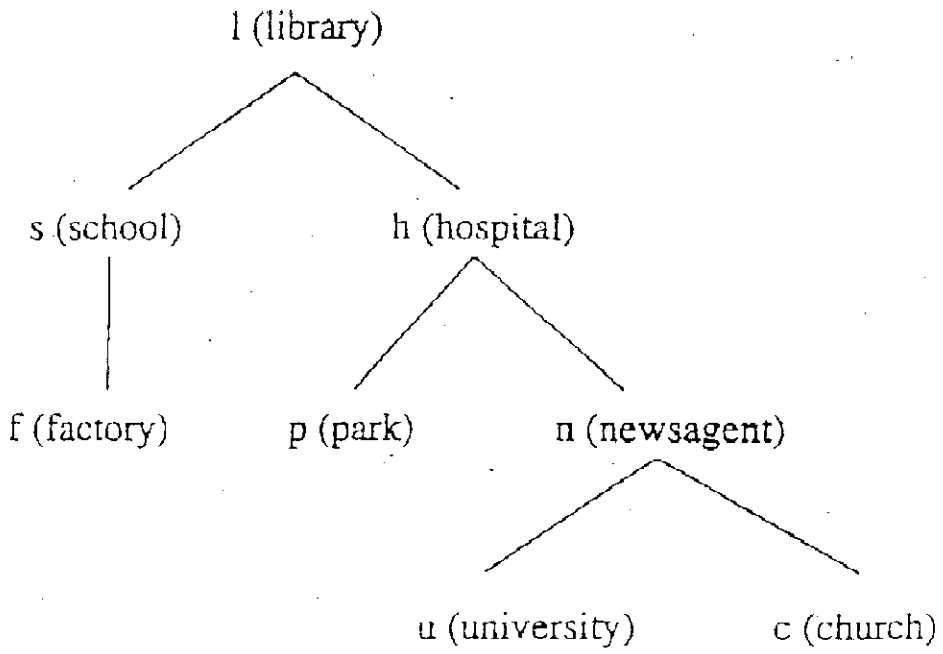
A (rooted) tree.

Il numero medio di successori per un nodo nello spazio di ricerca viene detto *fattore di diramazione.*

Un *albero* è un tipo particolare di grafo con un nodo speciale detto *radice* e un unico percorso tra coppie di nodi. A è detto il *padre* di B e F, B e F sono *figli* di A. B è *fratello* di F. A è *antenato* di tutti i nodi nell'albero i quali sono detti *discendenti* di A. Nodi senza figli sono detti *foglie*.

Usando questa terminologia astratta è possibile descrivere nello stesso modo molti tipi di problemi e pertanto utilizzare lo stesso algoritmo per risolverli.

La mappa della cittadina può essere rappresentata in astratto mediante l'*albero di ricerca* seguente:



In tal modo, invece di parlare di un percorso dalla biblioteca all'università possiamo dire che siamo interessati a trovare un percorso dal nodo iniziale l al nodo finale u.

Semplici tecniche di ricerca: breadth first e depth first

Le tecniche di ricerca più semplici sono due, *breadth first search* (BFS) e *depth first search* (DFS). Gli algoritmi BFS e DFS valgono per alberi e grafi di ricerca. Tranne casi particolari esistono molte possibili visite in quanto gli algoritmi non indicano in quale ordine vanno inseriti nello stack o nella coda i successori di un nodo.

"Visitare un nodo" è un termine generale che indica il momento in cui il nodo viene preso in considerazione: può indicare la semplice stampa del nodo o l'effettuazione di un'operazione particolare. Nel caso degli alberi tutti i nodi vengono visitati al momento della cancellazione dallo stack o dalla coda, nel caso dei grafi se un nodo cancellato è già stato visitato non viene visitato di nuovo.

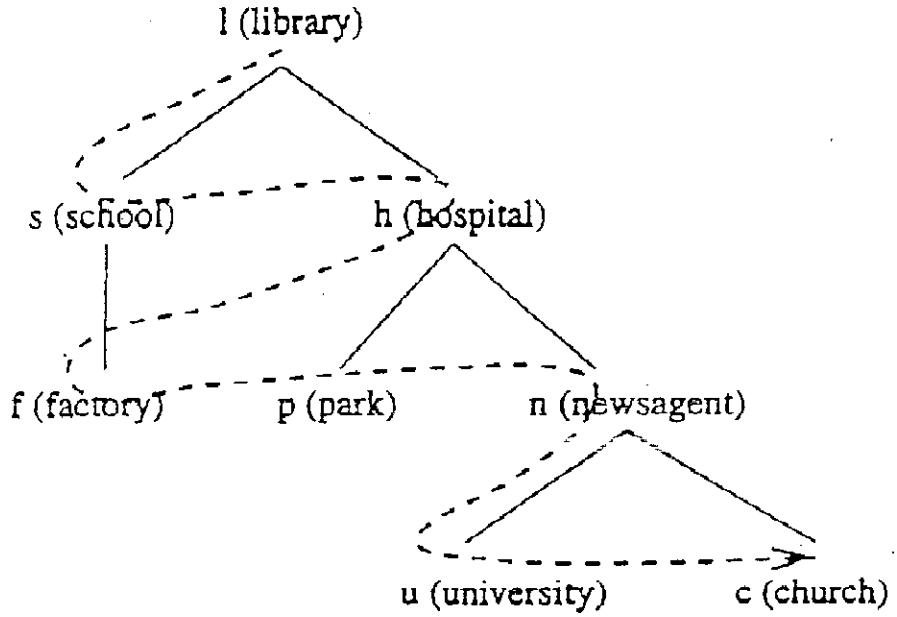
Esaminiamo le tecniche BFS e DFS prima in riferimento a un albero e poi a un grafo. Nel caso degli alberi le due tecniche sono essenzialmente algoritmi di attraversamento di un albero al pari degli algoritmi ben noti inorder, preorder, postorder visti per gli alberi binari. In più vi è il controllo per vedere se si è giunti a uno stato finale.

I due algoritmi attraversano l'albero in modo diverso, ma comunque tutti i nodi dell'albero possono essere visitati. Gli algoritmi si servono di una lista di nodi che sono stati visitati ma che vanno conservati in quanto i loro successori non sono stati ancora visitati. Nel caso del BFS la lista è una coda, nel caso del DFS è uno stack.

Gli algoritmi di attraversamento sono immediatamente trasformati in algoritmi di ricerca inserendo un controllo sul nodo dell'albero raggiunto in quel momento. In particolare, la ricerca BFS è simile all'attraversamento *per livelli* di un albero, mentre la ricerca DFS è analoga all'attraversamento *preorder*.

Breadth first search

Con l'algoritmo di attraversamento breadth first i nodi vengono visitati come indicato nella seguente figura e cioè in quest'ordine: l, s, h, f, p, n, u, c, questo nell'ipotesi che i successori di un nodo vengano inseriti nella coda da sinistra a destra. L'algoritmo di ricerca breadth first si ferma nel momento in cui si visita il nodo finale u.



La ricerca avviene in "ampiezza" considerando prima i nodi più vicini al nodo di partenza, poi quelli più vicini a quelli prima raggiunti, e così via. L'algoritmo breadth first per funzionare ha bisogno di una struttura dati coda, per cui nodi vengono inseriti in coda e prelevati dalla testa.

Osserviamo che pur essendo unico, in un albero, il percorso che va dalla radice a un nodo, in generale i nodi visitati durante la ricerca sono diversi a seconda dell'ordine in cui i successori di un nodo (cioè i suoi figli) vengono inseriti nella coda. Ad esempio, nel caso in cui i figli di un nodo venissero inseriti da destra a sinistra la ricerca BFS porterebbe alla visita dei seguenti nodi: l, h, s, n, p, f, c, u.

Inizialmente la coda contiene lo nodo iniziale l. Estraiamo l dalla coda, troviamo i suoi successori e li inseriamo nella coda. Quindi estraiamo il primo elemento della coda e inseriamo i suoi successori. Questo processo viene iterato finchè o il nodo estratto è il nodo finale (successo) o la coda è vuota (fallimento).

Nel nostro esempio inizialmente la coda è: [l]

estraiamo l dalla coda e inseriamo i suoi successori: [s, h]

s è estratto dalla coda, il suo successore f viene inserito: [h, f]

preleviamo h e inseriamo p e n:

[f, p, n]

f, senza successori, viene prelevato e lo stesso accade a p. Ora la coda è:

[n]

n viene estratto e i successori vengono inseriti:

[u, c]

u viene prelevato e poiché è lo stato finale la ricerca ha successo.

L'algoritmo può così essere descritto:

Inizialmente coda = [nodo iniziale] e *found* = FALSE

While coda non vuota AND not found:

a) cancella il nodo N dalla coda
b) visita il nodo N
c) se N è lo stato finale allora *found* = TRUE
d) trova tutti i successori di N e inseriscili nella coda

L'algoritmo si limita a mettere a TRUE il valore di *found*. Vedremo tra poco che l'algoritmo può essere facilmente modificato per fornire il percorso fino allo stato finale.

Individuare il percorso

L'algoritmo ora visto si limita a indicare se c'è o meno un percorso che dallo stato iniziale arriva allo stato finale. Volendo sapere, in caso di successo, qual è il percorso possiamo man mano memorizzare nella coda o nello stack il percorso invece che il semplice nodo, controllando ogni volta l'ultimo elemento del percorso per vedere se la ricerca ha avuto successo.

Allora, nel nostro esempio, la coda inizialmente contiene il percorso costituito dal solo elemento [l]:

[[l]]

estraiamo [l] dalla coda e inseriamo i due percorsi che si hanno considerando i successori di l:

[[l, s], [l, h]]

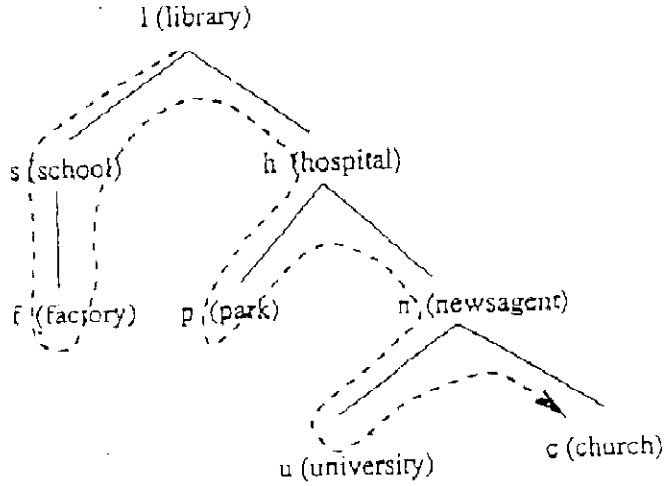
Estraiamo [l, s] dalla coda, controlliamo se l'ultimo elemento del percorso (s) è lo stato finale, il successore di s è f, inseriamo il percorso [l, s, f] nella coda:

[[l, h], [l, s, f]]

e così via finchè quando estraiamo dalla coda [l, s, h, f, p, n, u] ci accorgiamo che l'ultimo elemento è lo stato finale e la ricerca termina con successo.

Depth first search

Nell'attraversamento DFS si parte dallo stato iniziale e si cerca di spingersi in "profondità" finchè si arriva allo stato finale. Nel caso che si arrivi a una foglia che non è lo stato finale, si fa un passo indietro e si cerca un nodo alternativo (backtracking). L'attraversamento DFS nell'esempio porta alla sequenza l, s, f, h, p, n, u, c nell'ipotesi che i successori di un nodo vengano inseriti nello stack da destra a sinistra:



Se i figli vengono inseriti da sinistra a destra i nodi vengono visitati nell'ordine: l, h, n, c, u. L'algoritmo DFS ha bisogno di uno stack per lavorare: push del nodo iniziale, pop, controllo se stato finale, push dei successori, pop, controllo e così via. Se non c'è un successore, pop e push dei successori del nodo estratto, se non ce ne sono pop di nuovo finchè o si giunge allo stato finale o lo stack si svuota.

L'algoritmo può essere anche formulato ricorsivamente con gestione implicita dello stack, la versione con stack esplicito evidenzia che l'unica differenza è che BFS usa una coda, mentre DFS usa uno stack:

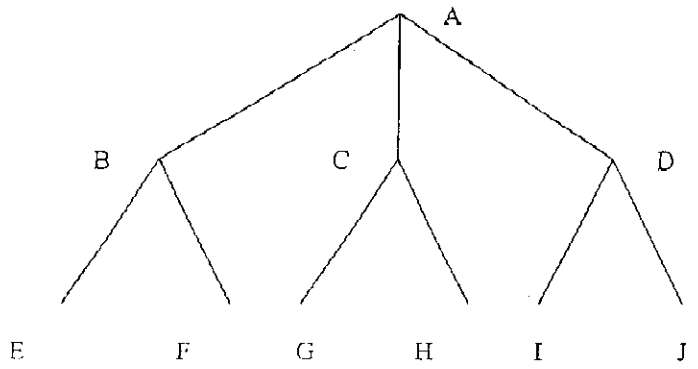
Inizialmente stack = [nodo iniziale] e *found* = FALSE

While stack non vuoto AND not found:

a) cancella il nodo N dallo stack
b) visita N
c) se N è lo stato finale allora *found* = TRUE
d) push nello stack di tutti i successori di N

Esempio

Dato il seguente albero di ricerca, determiniamo in quale ordine vengono visitati i nodi applicando gli algoritmi BFS e DFS. Nel caso del BFS i nodi vengono inseriti nella coda da sinistra a destra, nel caso del DFS i nodi vengono inseriti nello stack da destra a sinistra.

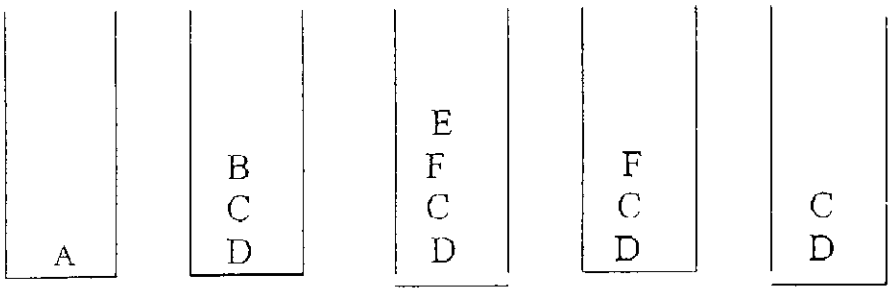


BFS:

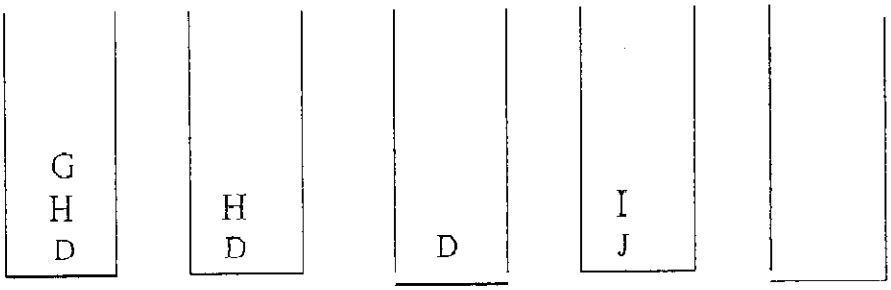
coda	nodo cancellato e visitato
A	A
B C D	B
C D E F	C
D E F G H	D
E F G H I J	E
F G H I J	F
G H I J	G
H I J	H
I J	I

e found viene messo a true

DFS:



Nodo canc. e visitato	A	B	E	F	C



Nodo canc. e visitato	G	H	D	I

e found viene messo a true

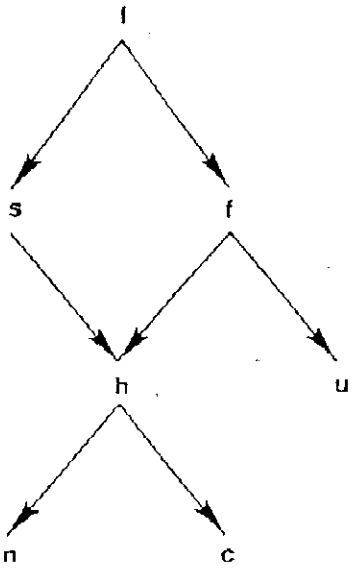
Ricerca in un grafo

Finora abbiamo effettuato le ricerche in un albero. In un grafo, in generale, è possibile, a differenza di un albero, giungere a un nodo con più percorsi diversi e possiamo anche, mediante un ciclo, tornare a un nodo.

La maggior parte dei problemi reali richiede ricerche su grafi e non su alberi. Basti pensare a una mappa in cui ci sono più percorsi per andare da una località a un'altra e nulla impedisce di tornare al punto di partenza.

Se applichiamo i nostri due algoritmi ad un grafo abbiamo due problemi: il primo è che potranno andare in loop riattraversando sempre gli stessi nodi. Il secondo problema è che gli algoritmi diventano meno efficienti in quanto possono rifare un lavoro già fatto.

Riguardo al primo problema, potremmo modificare leggermente gli algoritmi, in modo che evitino di andare in loop, non aggiungendo un nodo a un percorso se il nodo è già presente nel percorso. Questo però non ci risolve il problema dell'efficienza. Infatti consideriamo il grafo che nasce da una piccola modifica della mappa della nostra cittadina:



Non ci sono cicli, ma ci sono due percorsi per l'ospedale (h). Se usiamo l'algoritmo depth first search noi da l andiamo in s, h, n, back up e andiamo in c, back up torniamo ad h, back up torniamo in s, back up torniamo in l, andiamo in f, da f ad h n c esplorando di nuovo una parte di grafo già attraversata.

Il problema è che non abbiamo nessun segnale che ci dice che h n c sono stati già visitati. Possiamo allora introdurre una lista *visited* al fine di contenere i nodi già visitati. Gli algoritmi possono perciò essere semplicemente modificati in modo che i nodi che sono già nella lista visited non vengono esplorati di nuovo.

L'algoritmo BFS modificato per gestire i grafi è il seguente:

Inizialmente coda = [nodo iniziale] e *found* = FALSE

While coda non vuota AND not found:

a) cancella N dalla coda

b) se N non è nella lista visited:

1) visita N
2) aggiungi N a visited
3) se N è lo stato finale allora *found* = TRUE
4) inserisci nella coda tutti i successori di N

Analogamente a quanto visto per gli alberi, l'algoritmo DFS per i grafi differisce dal BFS solo nel fatto che la struttura dati utilizzata è uno stack e non una coda:

Inizialmente stack = [nodo iniziale] e *found* = FALSE

While stack non vuoto AND not found:

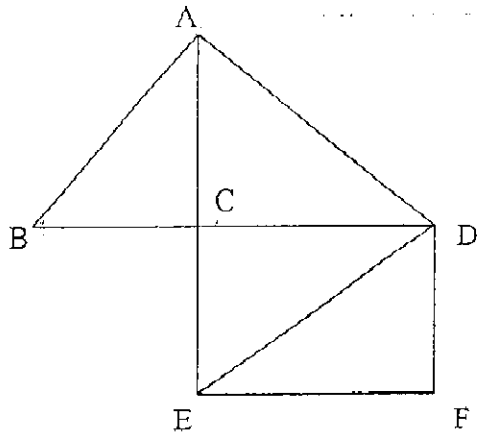
c) pop N dallo stack

d) se N non è nella lista visited:

1) visita N
2) aggiungi N a visited
3) se N è lo stato finale allora *found* = TRUE
4) push nello stack di tutti i successori di N

Esempio

Determiniamo in quale ordine vengono visitati i nodi applicando gli algoritmi BFS e DFS (stato iniziale A, stato finale F). Nel caso del BFS i nodi vengono inseriti nella coda in senso orario a partire dal basso, nel caso del DFS entrano nello stack in senso orario a partire dall'alto.



BFS:

coda	nodo cancellato	nodo visitato	lista visited
A	A	A	A
C B D	C	C	A C
B D E B A D	B	B	A C B
D E B A D A C	D	D	A C B D
E B A D A C F E C A	E	E	A C B D E
B A D A C F E C A C D F	B		A C B D E
A D A C F E C A C D F	A		A C B D E
D A C F E C A C D F	D		A C B D E
A C F E C A A C D F	A		A C B D E
C F E C A A C D F	C		A C B D E
F E C A A C D F	F	F	A C B D E F

F è lo stato finale per cui la visita termina.

DFS:

stack	nodo cancellato	nodo visitato	lista visited
A	A	A	A
D C B	B	B	A B
D C A C	C	C	A B C
D C A A D E B	B		A B C
D C A A D E	E	E	A B C E
D C A A D C D F	F	F	A B C E F

F è lo stato finale per cui la visita termina.

Ricerca euristica

Gli algoritmi di ricerca vengono utilizzati in AI per trovare una sequenza di passi che ci portino da uno stato iniziale a uno o più stati finali. Abbiamo visto i due algoritmi più semplici che effettuano una ricerca sistematica ed esaustiva.

Ci sono però situazioni in cui lo spazio di ricerca è troppo grande e bisogna introdurre una funzione *euristica* che ci selezioni i nodi più promettenti per la soluzione. Questi nodi vengono esplorati per primi e solo in caso di insuccesso si esplorano i nodi rimanenti. Queste tecniche prendono il nome di *tecniche di ricerca euristica.*

L'idea di base della ricerca euristica è che, invece di provare tutti i possibili percorsi, conviene concentrarsi sui percorsi che sembrano più appropriati per giungere allo stato finale. In generale non possiamo essere sicuri che il percorso ci condurrà allo stato finale, il percorso giusto potrebbe essere molto complicato e contorto, ma spesso il metodo euristico funziona.

Nella ricerca euristica noi utilizziamo una *funzione di valutazione* che assegna un punteggio a ogni nodo in funzione di quanto sembra essere vicino allo stato finale. Ad esempio, nella ricerca di un percorso tra due città una possibile funzione di valutazione potrebbe essere la distanza in linea d'aria fra le due città. Pertanto i percorsi che sembra che ti portino più vicino al punto di arrivo sono esplorati prima di quelli che sembra che ti allontanino.

Questa strategia però non funziona, ad esempio, se la strada buona per arrivare a destinazione è un'autostrada e per prenderla dobbiamo prima allontanarci dal punto di arrivo. Comunque la strategia spesso ci permette di fare rapidamente delle previsioni e di trovare buone soluzioni.

Vi sono molti algoritmi per la ricerca euristica, noi ne vedremo tre: hill climbing, best first search e A*. Gli algoritmi valgono per grafi, però noi considereremo per semplicità solo alberi di ricerca.

Hill climbing

L'idea alla base dell'algoritmo è di spostarsi ad ogni passo verso il miglior nodo successore, questo solo se il nodo successore è migliore del nodo attuale. Ad esempio, riferendosi alla mappa della città, se siamo all'ospedale e voglio arrivare alla chiesa, posso andare all'edicola o al parco, ma scelgo di andare all'edicola in quanto questo nodo è più vicino alla chiesa sia del parco che dell'ospedale.

L'algoritmo è il seguente:

1. Parti con *current-state* = [nodo iniziale]

2. While *current-state* NOT = goal-state AND vi è cambiamento in *current-state*:

   a) Ricava i successori di *current-state* e usa la funzione di valutazione per assegnare un punteggio a ciascun successore.

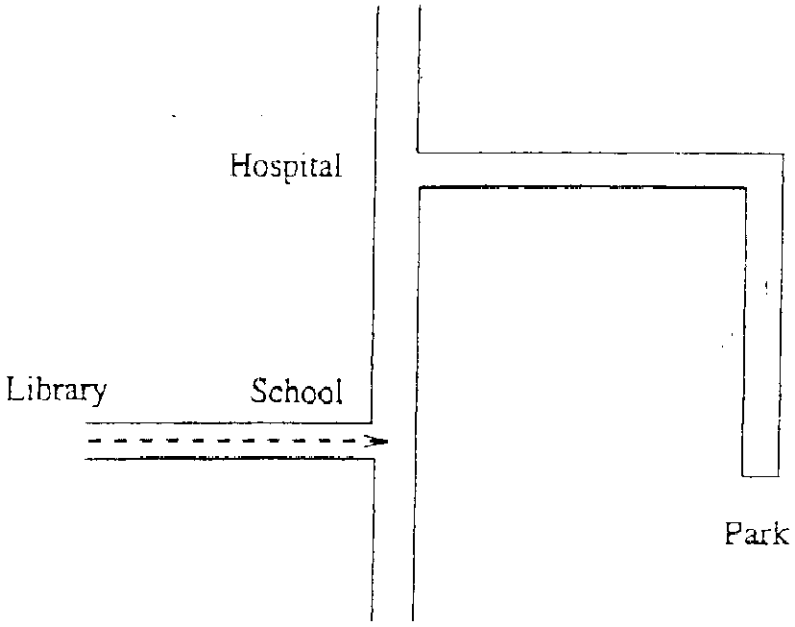
   b) Se uno dei successori ha un punteggio migliore di *current-state*, il nuovo *current-state* diventa il successore con il miglior punteggio.

Osserviamo che l'algoritmo non cerca di provare esaustivamente ogni nodo e percorso, pertanto non c'è una lista di nodi da esplorare che va conservata, solo lo stato attuale.

L'algoritmo termina quando giungiamo allo stato finale oppure non vi sono successori dello stato corrente che siano migliori di quest'ultimo. Questo però può creare un problema.

Ad esempio, supponiamo di cercare un percorso dalla biblioteca all'università, usando la distanza in linea d'aria come funzione di valutazione. Inizialmente lo stato corrente è la biblioteca, poi diventa l'ospedale e quindi il parco (in quanto il parco è più vicino, in linea d'aria, dell'edicola all'università). Ma questo è un vicolo cieco e non possiamo uscirne in quanto l'ospedale ha un punteggio inferiore al parco. Perciò l'algoritmo si ferma e segnala insuccesso.

Un problema analogo si presenta nel caso della seguente mappa:



Se vogliamo andare dalla biblioteca al parco, ci spostiamo alla scuola e ci fermiamo in quanto non c'è un altro nodo più vicino al parco (il nodo ospedale è più distante).

Si può ovviare in una certa misura al problema modificando l'algoritmo, consentendogli ad esempio di fare un limitato backtracking o di controllare se un successore è seguito da un nodo migliore del current-state.

Nessuna soluzione ci garantisce in ogni caso il successo a differenza di DFS e BFS i quali se esiste un percorso lo trovano ma dispongono di stack e coda non presenti nell'hill climbing.

Pertanto l'algoritmo hill climbing è valido solo per un limitata classe di problemi in cui abbiamo una funzione di valutazione che accuratamente predice l'effettiva distanza che ci separa dalla soluzione.

Best first search

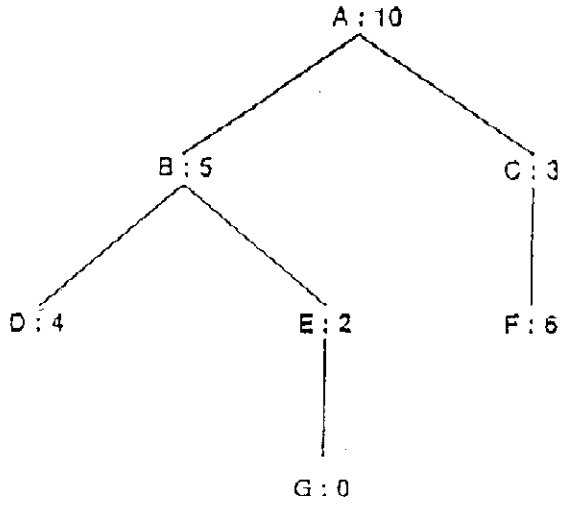
L'algoritmo best first search somiglia all'hill climbing in quanto usa una funzione di valutazione e sempre sceglie il nodo con il miglior punteggio. Però è un algoritmo esaustivo in quanto prova, se necessario, tutti i possibili percorsi.

Utilizza una lista di nodi da esplorare, come avviene in DFS e BFS, ma a differenza di questi, che prelevano dallo stack o dalla coda, l'algoritmo ha bisogno di una *coda a priorità*, cioè una coda in cui gli inserimenti avvengono normalmente in fondo alla coda, ma viene prelevato invece il nodo con il miglior punteggio (nel nostro caso il punteggio minimo). Pertanto l'ordine in cui i nodi vengono inseriti nella coda non influenza la visita in quanto viene sempre cancellato il nodo migliore.

L'algoritmo è il seguente

1. Parti con coda a priorità = [nodo iniziale]
2. While la coda a priorità non è vuota:
    a) preleva il nodo migliore dalla coda a priorità
    b) Se è il goal node esci con successo altrimenti trova i suoi successori
    c) Assegna un punteggio ai successori usando la funzione di valutazione e inserisci i nodi nella coda a priorità.

Supponiamo di avere l'albero di ricerca seguente:



L'etichetta dei nodi indica il costo stimato per giungere alla soluzione, per cui un valore più basso è migliore. Supponiamo che il goal state sia G.

Se usassimo il BFS (inserimento nella coda da sinistra a destra) i nodi verrebbero visitati in quest'ordine: A, B, C, D, E, F, G. Usando il DFS (inserimento nello stack da destra a sinistra) l'ordine sarebbe:A,B, D, E, G.

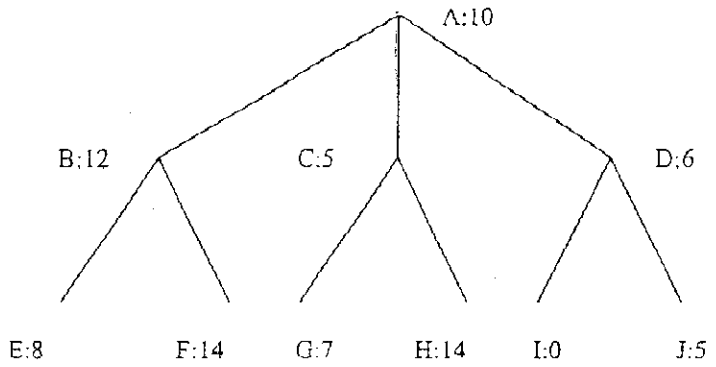
Ovviamente questi algoritmi ignorano il punteggio presente nei nodi. Usando hill climbing una soluzione non verrebbe mai trovata in quanto, una volta giunti in C, ci fermeremmo segnalando insuccesso.

Applicando invece l'algoritmo best first search l'ordine è: A, C, B, E, G. Infatti nella coda a priorità inserisco B e C e prelevo C che ha miglior punteggio, inserisco poi F e prelevo B (costo inferiore a F), inserisco poi D e E e prelevo E, infine inserisco G, prelevo G è giungo alla soluzione.

Se si dispone di una buona funzione di valutazione l'algoritmo best first search può drasticamente ridurre il tempo necessario per trovare una soluzione. La prima soluzione trovata può non essere la migliore, ma se una soluzione esiste viene trovata in genere rapidamente. Se invece la funzione non è appropriata tanto vale usare algoritmi più semplici quali il DFS o il BFS. Va inoltre considerato che se la funzione di valutazione è complessa, cioè richiede del tempo per calcolare il punteggio, i benefici ottenuti dall'algoritmo possono essere vanificati.

Esempio

Consideriamo di nuovo l'albero di ricerca al quale abbiamo applicato gli algoritmi BFS e DFS. Questa volta vicino a ogni nodo c'è il costo stimato per giungere al nodo finale I. Determiniamo in quale ordine vengono visitati i nodi applicando gli algoritmi Hill climbing e Best first.



HILL CLIMBING:

Inizialmente current-state = A, tra i successori di A scelgo C che ha il punteggio più basso (5), i successori di C hanno punteggio maggiore di 5 per cui l'algoritmo si ferma senza giungere allo stato finale I.

BEST FIRST:

coda a priorità	nodo cancellato e visitato
A	A
B C D	C
B D G H	D
B G H I J	I

e sono giunto al goal node.

L'algoritmo A*

L'algoritmo best first search è utile ma non tiene conto, quando deve scegliere il prossimo nodo, del costo del percorso fatto fino a quel momento. Di conseguenza una soluzione viene trovata, se esiste, ma può non essere una buona soluzione.

Vi è una variante del best first search detta A* che tenta di trovare una soluzione che minimizzi il costo totale del percorso (costo va inteso in senso ampio, per problemi di tipo geografico potrebbe essere la distanza).

Nell'algoritmo A* la funzione di valutazione consiste di due parti. La prima parte è basata sul costo associato al percorso per giungere dallo stato iniziale al nodo corrente.

La seconda parte è una *stima* del costo per giungere dal nodo corrente al nodo finale.

Il punteggio totale è la somma (eventualmente pesata) di queste due parti, e fornisce una stima del costo totale dal nodo iniziale al finale passando per il nodo corrente.

Pertanto, se g(nodo) è il costo per andare dal nodo iniziale al nodo corrente, e h(nodo) è il costo stimato per andare dal nodo corrente al nodo finale, il punteggio totale f(nodo) è:

f(nodo) = g(nodo) + h(nodo)

L'algoritmo A* è essenzialmente uguale al best first search, viene solo usata questa funzione più complessa. Il nodo migliore è quello che ha il punteggio totale più basso.

L'algoritmo è il seguente:

1. Parti con coda a priorità = [nodo iniziale]

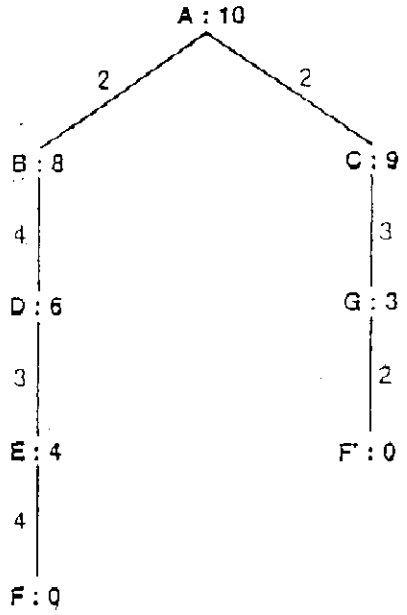
2. While la coda a priorità non è vuota:

   d) preleva il nodo con punteggio totale minimo dalla coda

   e) Se è il goal node esci con successo altrimenti trova i suoi successori

   f) Calcola il punteggio totale per i successori e inserisci i nodi nella coda a priorità

Consideriamo il seguente albero di ricerca:



dove i rami sono etichettati con il costo per andare da un nodo all'altro e le etichette dei nodi sono una stima del costo per arrivare al nodo finale.

Se usiamo il BFS i nodi vengono visitati in quest'ordine: A, B, D, E, F. Poiché F è uno stato finale, abbiamo individuato una soluzione che però ha un costo 2+4+3+4=13.

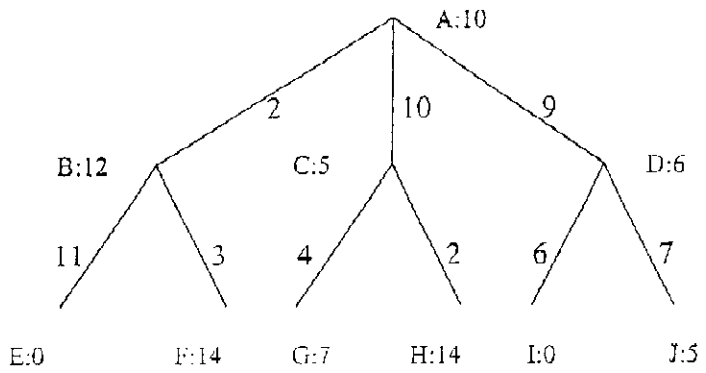
Se applichiamo l'algoritmo A* allora i nodi vengono visitati in quest'ordine: A, B, C, G, F' in quanto il costo di B (2+8=10) è inferiore a quello di C (2+9=11), il costo di C (11) è inferiore a quello di D (2+4+6=12), e così via.

Il costo del percorso da A a F' è 7 e in effetti l'algoritmo A* trova il percorso di minimo costo. Bisogna però assicurarsi che h(nodo) stimi correttamente il costo per arrivare alla soluzione, se ciò non accade l'algoritmo non funziona bene.

Confrontiamo A* con BFS e best first. Utilizzando BFS, se il costo per andare da un nodo a un altro è lo stesso per tutte le coppie di nodi, la prima soluzione trovata avrà il costo minimo, però non è detto che sia trovata rapidamente. Applicando best first search una soluzione viene trovata in genere rapidamente ma non è detto che il suo costo sia basso. Se ci serviamo di A* in genere troviamo rapidamente una buona soluzione.

Esempio

Dato il seguente albero di ricerca, determiniamo in quale ordine vengono visitati i nodi applicando gli algoritmi best first e A*. Vicino a ogni nodo c'è il costo stimato per giungere ai nodi finali E ed I. I rami sono etichettati con il costo per andare da un nodo all'altro.



BEST FIRST:

coda a priorità	nodo cancellato e visitato
A	A
B C D	C
B D G H	D
B G H I J	I

e ho trovato il goal node I che ha un costo 9 + 6 = 15.

A*:

coda a priorità	nodo cancellato e visitato	
A	A	
B C D	B	in quanto il costo di B (2+12) è minore di C (10+5) e D (9+6)
C D E F	E	in quanto il costo di E (2+11) è minore di C (15), D (15), F 19)

E è uno stato finale per cui ci fermiamo e abbiamo trovato un percorso che ha un costo (13) inferiore a quello individuato da best first (15).

Problemi di complessità

Quando applichiamo le tecniche di ricerca a problemi reali spesso dobbiamo fare i conti con la complessità degli algoritmi.

Se cerchiamo in un grafo il cammino più breve (inteso come minimo numero di nodi nel percorso) è preferibile applicare la BFS in quanto ad ogni passo ci si allontana il minimo indispensabile dal nodo di partenza. Il problema è che tutti i successori di ciascun nodo vanno conservati nella coda e questo può portare ad eccedere lo spazio di memoria disponibile. Ad esempio, se il fattore di diramazione è n e lo stato finale si trova a profondità d, allora con la BFS possiamo dover cercare n^d nodi.

L'algoritmo DFS richiede molto meno spazio di memoria in quanto, in un dato istante, nello stack è memorizzato il percorso fino al nodo più tutti i successori dei *soli* nodi facenti parte del percorso. Lo svantaggio della DFS è che può esplorare senza esito un cammino lunghissimo mentre magari esiste un percorso adiacente vincente. In generale, il DFS è più veloce se vi sono molti percorsi che portano alla soluzione, tutti abbastanza lunghi. Il BFS è più veloce se vi è un percorso corto all'interno di un grande spazio di ricerca.

Le tecniche di ricerca su grafo, che evitano di riesaminare nodi già esplorati, possono essere utili ma comportano l'onere aggiuntivo di gestire le liste di nodi già esaminati.

I metodi di ricerca euristica possono essere adatti ma solo quando le funzioni di valutazione sono appropriate.

In generale, perchè un problema sia gestibile occorre che il fattore di diramazione sia abbastanza piccolo. Il fattore spesso può essere ridotto mediante opportune considerazioni che evitino di fare esplorazioni che non portano a nulla. Se non si riesce a ridurre il fattore di diramazione si ha l'*esplosione combinatoriale*. Vi sono troppe combinazioni da provare e quanto più andiamo in profondità tanto più cresce lo spazio di ricerca.

Il problema dell'esplosione rende spesso le tecniche general-purpose viste inadeguate per la soluzione di problemi del mondo reale e vanno arricchite mediante l'utilizzo di tecniche specifiche del dominio in esame.

La ricerca nel problem solving

Abbiamo visto cinque algoritmi di ricerca, ma non come essi sono utilizzati per la soluzione di problemi. In generale, le tecniche di ricerca servono per trovare una sequenza di azioni che ci portano da uno stato iniziale a uno stato finale. Le azioni possono essere fisiche (ad esempio, spostarsi dalla città A alla città B) o astratte (ad esempio, cercare una sequenza di passi che ci permettano di provare X a partire dall'insieme di fatti S).

Vi sono molti differenti modi in cui le tecniche di ricerca possono essere usate per il problem solving. Noi ora, ad esempio, mostreremo come una semplicissima rappresentazione delle possibili azioni possa essere utile per risolvere problemi di pianificazione (planning problems) applicando le idee di base sulle tecniche di ricerca.

Un nodo nello spazio di ricerca rappresenterà una descrizione formale dello stato attuale del problema. I nodi successore saranno i possibili nuovi stati che possono essere raggiunti dallo stato attuale applicando un'azione specifica. Il search space viene anche detto state space in quanto racchiude tutti i possibili stati in cui ci si può trovare.

Tecniche di planning

Il planning, in AI, è il problema di trovare una sequenza di azioni di base per raggiungere un certo obiettivo. Questa sequenza di azioni sarà il piano del sistema che potrà poi essere eseguito.

Il planning compare spesso nell'ambito della robotica, dove c'è un robot fisico che eseguirà il piano. Il planning è importante anche in altri ambiti dell'AI, per esempio nella comprensione del linguaggio naturale dove si ragiona sugli obiettivi delle persone allo scopo di comprendere il vero significato delle loro parole.

Gli algoritmi di ricerca prima visti possono essere visti come semplici tecniche di planning, però, in genere, questo termine è riservato a problematiche più complesse. Noi descriviamo un semplice metodo di planning basato sull'approccio *means-ends analysis* (MEA) al problem solving.

MEA è un approccio al problem solving che, piuttosto che cercare alla cieca utilizzando tutte le possibili azioni, si concentra su azioni che riducono la *differenza* fra lo stato attuale e lo stato finale.

Un'altra caratteristica del metodo è che se si individua un'azione che riduce la differenza, essa viene considerata anche se non è applicabile allo stato attuale. In tal caso ci si pone il problema di arrivare a uno stato in cui questa azione utile possa essere applicata.

Il metodo MEA può essere applicato a molti problemi, noi ci limitiamo a considerare un semplice caso di planning per un robot.

La tecnica utilizza i concetti di base della ricerca nello state-space. Essenzialmente si cerca un percorso dallo stato iniziale a quello finale esaminando le possibili differenti azioni che possono essere intraprese al fine di individuare la sequenza di azioni che ci porta al successo.

Esempio

Lo stato iniziale del problema può essere rappresentato come una lista di fatti che sono veri. Supponiamo che lo stato iniziale sia il seguente:

[at (robot, living-room), at (beer, kitchen), at (fred, living-room), door-closed (kitchen, living-room]

Questa lista rappresenta lo stato in cui Fred è in salotto con il suo robot, ma la birra è in cucina e la porta della cucina è chiusa.

Data questa rappresentazione dello stato iniziale, dobbiamo ora rappresentare le azioni mediante operatori che danno le precondizioni dell'azione e i suoi effetti sullo stato del problema. Un effetto può aggiungere un nuovo fatto allo stato oppure cancellare un fatto.

Pertanto, se assumiamo che le uniche azioni ammesse nel nostro esempio siano "il robot apre/chiude la porta", "il robot si sposta da una stanza a un'altra" e "il robot porta un oggetto da una stanza a un'altra", abbiamo i seguenti operatori che descrivono le possibili azioni:

Operator	Preconditions	Add	Delete
open(R1, R2)	at(robot, R1) door_closed(R1, R2)	door_open(R1, R2)	door_closed(R1, R2)
close(R1, R2)	at(robot, R1) door_open (R1, R2)	door_closed(R1, R2)	door_open(R1, R2)
move(R1, R2)	at(robot, R1) door_open(R1, R2)	at(robot, R2)	at(robot, R1)
carry(R1, R2, O)	door_open(R1, R2) at(robot, R1) at(O, R1)	at(robot, R2) at(O, R2)	at(robot, R1) at(O, R1)

Supponiamo che il nostro stato finale sia Fred con la birra nel salotto, il robot a fianco, la porta della cucina chiusa:

[at (beer, living-room), at (fred, living-room), at (robot, living-room), door-closed (kitchen, living-room]

La tecnica Means-ends analysis cerca di trovare un operatore i cui effetti riducano la differenza tra gli stati iniziale e finale.

Notiamo che l'unica differenza fra gli stati è la birra che deve spostarsi dalla cucina al salotto. L'unico operatore che può spostare la birra è il "carry". Se lo potessimo applicare alla birra arriveremmo allo stato finale. Il problema è che il robot si trova in salotto e la porta è chiusa, per cui l'operatore non lo possiamo applicare.

Il fatto che l'operatore non sia al momento applicabile non ci deve fermare. Noi consideriamo le precondizioni non soddisfatte dell'azione come il nuovo obiettivo da raggiungere. In tal modo ricorsivamente chiamiamo l'algoritmo means-ends con le precondizioni dell'azione come stato finale. Cioè cerchiamo di passare dallo stato iniziale:

[at (robot, living-room), at (beer, kitchen), at (fred, living-room),
door-closed (kitchen, living-room]

allo stato finale intermedio:

[open-door (kitchen, living-room), at (robot, kitchen)]

Dobbiamo ora trovare una sequenza di azioni che realizzi questo stato finale intermedio. E' chiaro che ci servono le azioni:

open (kitchen, living-room) e move (living-room, kitchen).

Ora possiamo applicare l'azione carry (kitchen, living-room, beer).

Pertanto, partendo dallo stato iniziale, le tre azioni ci portano allo stato:

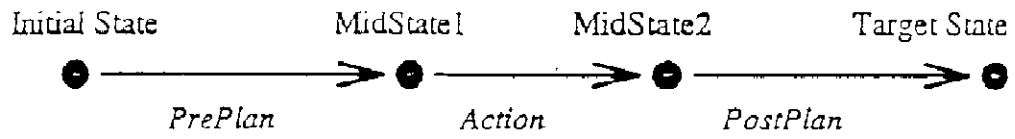
[at (robot, living-room), at (beer, living-room), at (fred, living-room), door-open (kitchen, living-room]

Siamo quasi giunti allo stato finale, ma la porta è aperta. La quarta azione ci porta a completare la sequenza:

open (kitchen, living-room)
move (living-room, kitchen)
carry (kitchen, living-room, beer)
close (kitchen, living-room)

Come si vede abbiamo prima trovato un'azione da compiere nel mezzo della sequenza: carry (kitchen, living-room, beer), poi abbiamo aggiunto la parte iniziale e finale del piano.

Ciò può essere così illustrato:



La versione base dell'algoritmo MEA può così essere descritta:

Find-plan (Initial-state, Target-state)

- se tutti i fatti nel Target-state sono veri in Initial-state allora esci con successo

- altrimenti:

    1. seleziona un Obiettivo non raggiunto (fatto non vero) presente in Target-state

    2. trova un'Azione che aggiunge l'Obiettivo allo stato attuale

    3. trova un piano (Preplan) che realizza le precondizioni dell'Azione trovata, chiama cioè Find-plan (Initial-state, Preconditions). Sia Midstate1 lo stato ottenuto applicando il Preplan allo stato iniziale

    4. applica l'Azione a Midstate1 per ottenere Midstate2

    5. trova un piano (Postplan) per passare da Midstate2 a Target-state, chiama cioè Find-plan (Midstate2, Target-state)

    6. restituisci un piano costituito da Preplan, Action e Postplan

Alberi di gioco

Come esempio finale di come le tecniche di ricerca possono essere usate nel problem solving, consideriamo gli *alberi di gioco*. Supponiamo di avere due giocatori (nel seguito indicati con P1 e P2) i quali sono perfettamente a conoscenza dello stato del gioco. Esempi tipici sono gli scacchi e la dama.

I problemi posti da questi giochi sono diversi da quelli esaminati finora in quanto ora abbiamo due avversari i quali cercano ciascuno di vincere e quindi di far fallire la strategia dell'avversario.

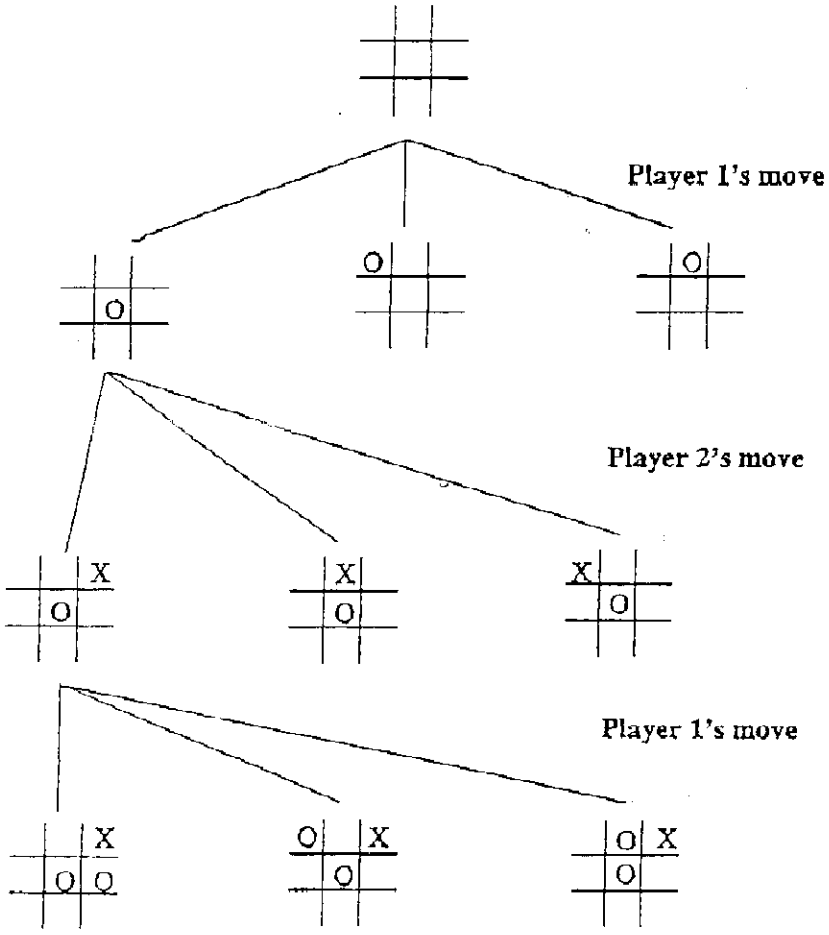
In altri termini, pur essendo possibile che un giocatore individui una sequenza di azioni che lo conduca alla vittoria, è molto probabile che una mossa dell'avversario gli impedisca di portare a termine il piano vincente.

Bisogna cercare di individuare buone mosse che hanno un'alta probabilità di condurre alla vittoria quali che siano le mosse dell'altro giocatore.

Gli alberi di ricerca sono ancora presenti parlando di giochi, prendono ora il nome di *alberi di gioco*.

Il gioco del tris

Consideriamo il gioco del tris, in cui due giocatori, a turno, cercano di allineare tre "x" o tre "o". Il gioco non è molto impegnativo, se entrambi i giocatori fanno sempre la mossa migliore, la partita finisce in parità. Un piccolissimo frammento dell'albero di gioco è il seguente:

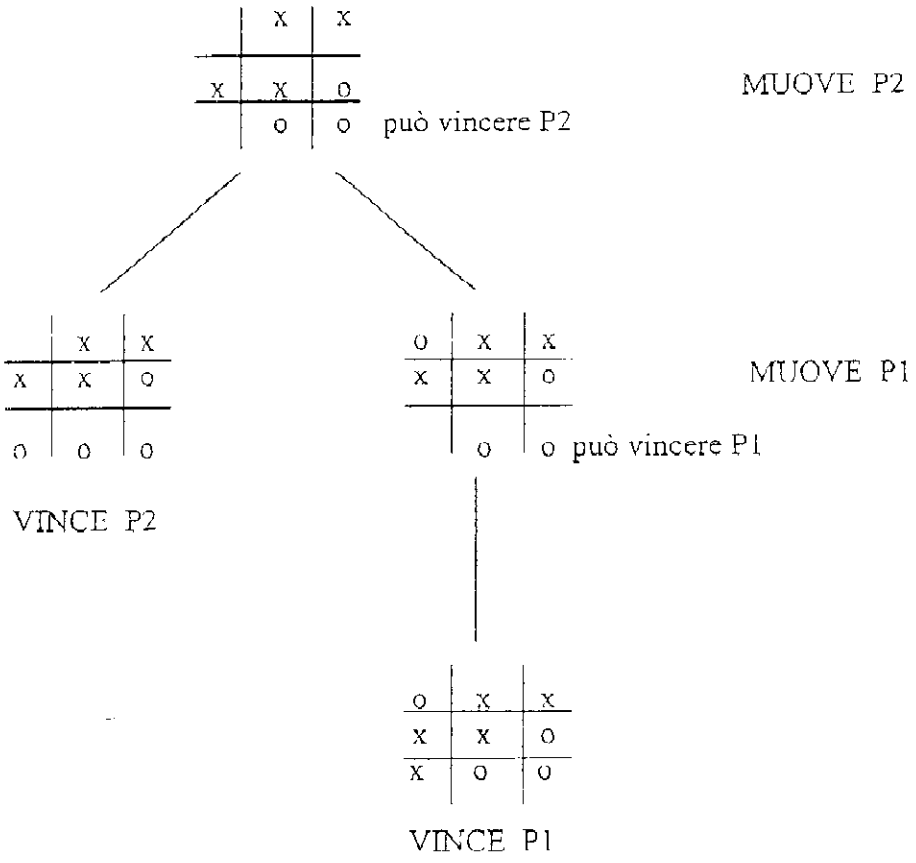


Ovviamente dalla radice dell'albero si dipartono nove rami (le nove possibili mosse di P1) e non solo i tre disegnati. Analogamente, da ciascuno dei nove figli della radice escono otto rami (le otto possibili mosse di P2), e così via.

Quando un gioco viene rappresentato mediante un albero che evidenzia tutte la sequenza di mosse fatte, a turno, dai giocatori si dice che è in *forma estesa*. Esiste anche un altro modo di rappresentare un gioco, la *forma normale*, che è utile quando i giocatori fanno le loro mosse simultaneamente. Dei giochi in forma normale parleremo più avanti.

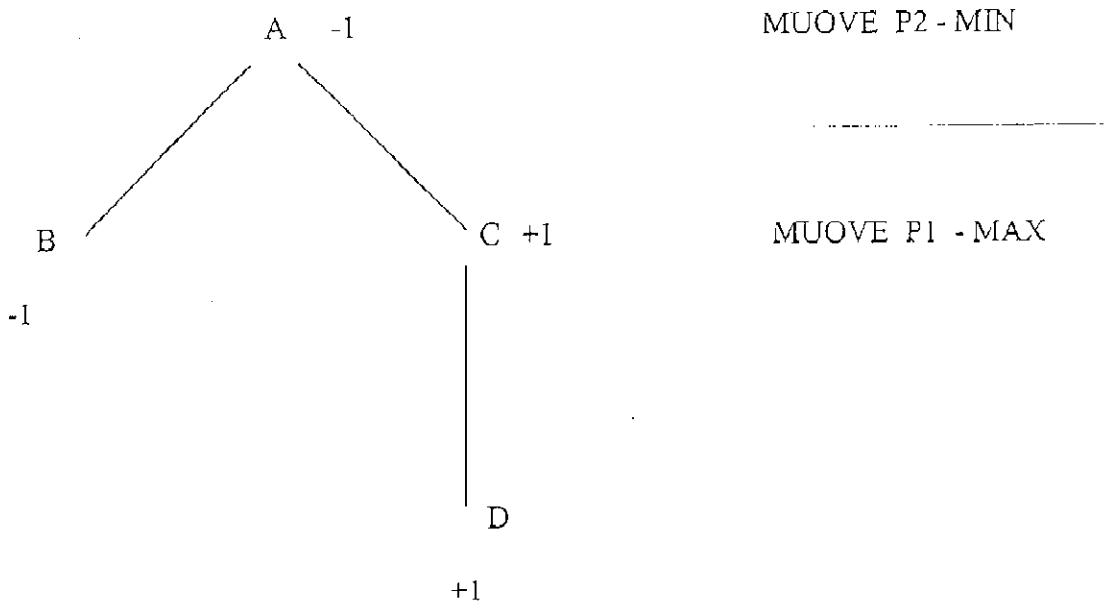
Nel disegno sono presenti solo 10 nodi. Quanti nodi ha l'albero completo per il tris ? P1 ha a disposizione 9 caselle possibili, P2 ne ha 8, P1 quindi ne ha solo 7, P2 a sua volta 6, e così via finchè, alla nona mossa (se non finisce prima il gioco), P1 ha a disposizione solo una casella. Pertanto il numero totale di nodi viene ad essere nell'ordine di $9! < 10^6$ nodi.

Disegnare l'intero albero di gioco per il tris sarebbe impossibile. Possiamo però disegnare l'albero relativo a una fase finale di gioco, in cui tocca a P2 fare la mossa (P1 mette la "x", P2 la "o"):



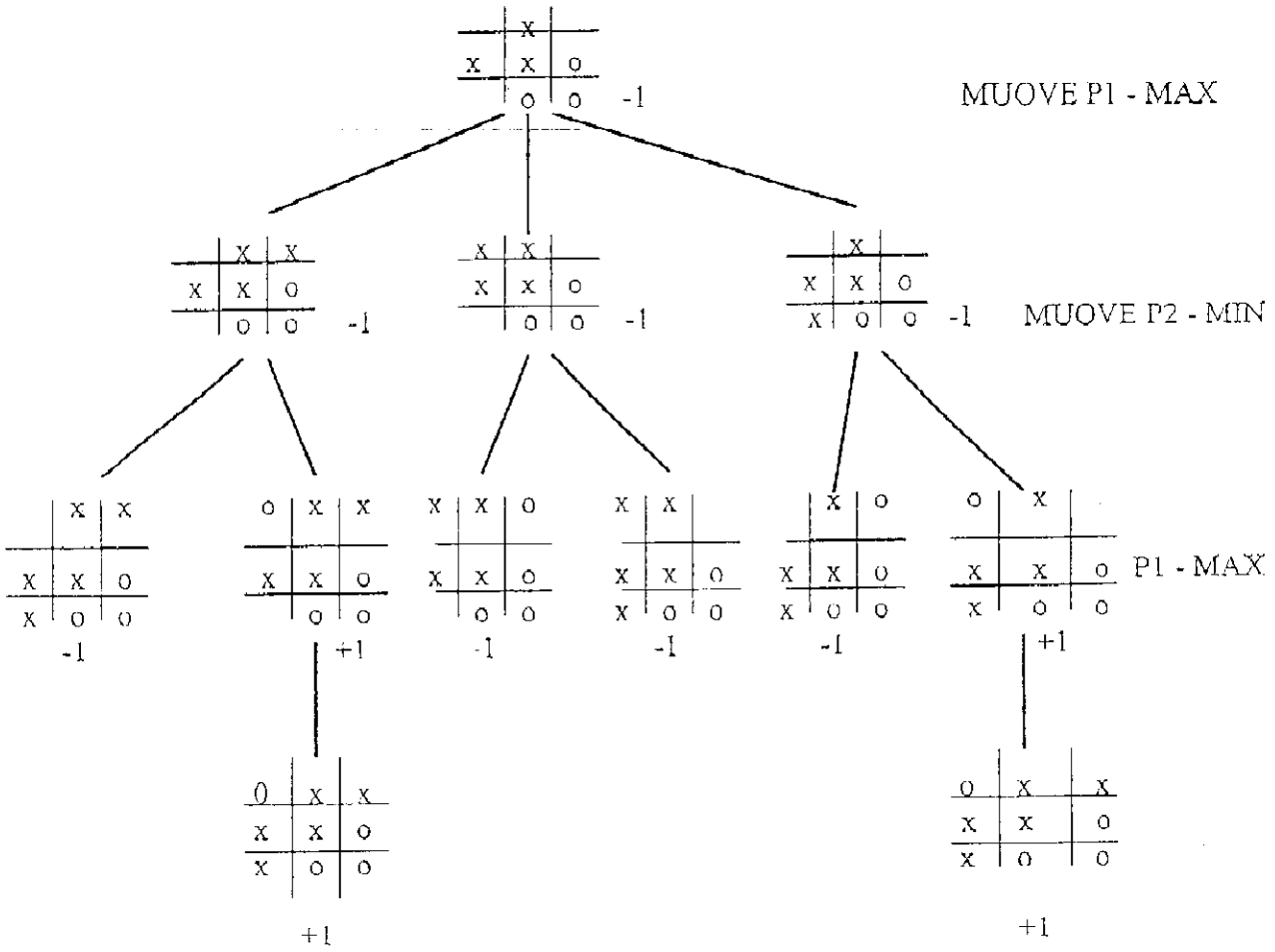
E' chiaro che a P2 conviene mettere il cerchietto in basso a sinistra e vincere. Se P2 sbaglia e lo mette in alto a sinistra, P1 mette la "x" in basso a sinistra e vince. Il fatto importante è che io posso etichettare non solo le foglie ma tutti i nodi dell'albero con il nome del possibile vincitore. La frase "può vincere P2" significa che P2 ha una mossa che lo porta alla vittoria, il "può" è giustificato dal fatto che se P2 fa la mossa sbagliata, perde (a rigore il "più" in "può vincere P1" non è appropriato essendoci solo una casella libera).

Possiamo rappresentare l'albero in modo sintetico ed equivalente come segue:



In questo disegno le lettere A, B, C, D denotano le configurazioni prima viste. Il valore +1 denota la vincita di P1, il valore -1 la vincita di P2. Il -1 vicino al nodo A indica che P1 è condannato a perdere, a meno che P2 sbagli mossa. Il significato delle etichette MAX e MIN sarà chiaro fra poco.

Un finale di gioco leggermente più complesso è il seguente:

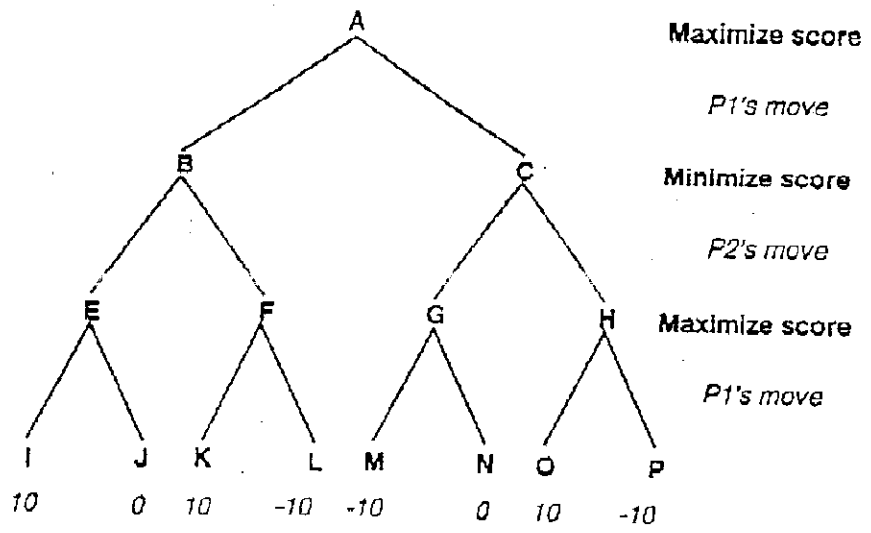


Analogamente a prima, il -1 vicino alla radice dell'albero indica che P2 può vincere se fa le mosse giuste.

L'algoritmo minimax

L'algoritmo minimax che ora vedremo ci serve proprio ad assegnare i valori a tutti i nodi dell'albero. Per capire il suo funzionamento è utile fare un altro esempio.

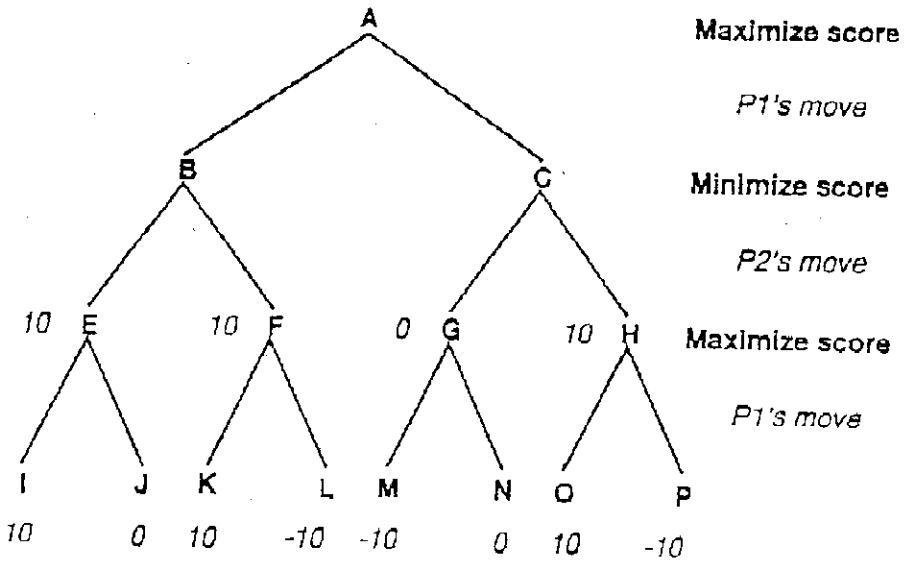
Supponiamo di avere costruito, in corrispondenza di un certo gioco non meglio precisato, il seguente albero:



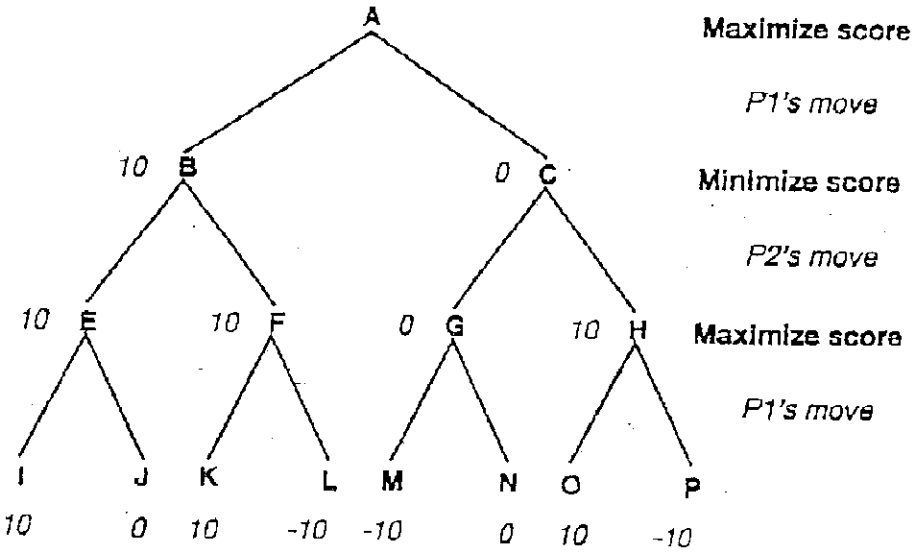
Le foglie dell'albero indicano vincita, perdita o pareggio: supponiamo che la vincita di P1 la indichiamo con il valore +10, la perdita di P1 (e quindi la vincita di P2) con il valore -10, il pareggio con 0.

Osserviamo che i nodi E, F, G, H denotano stati in cui tocca a P1 fare la mossa. Quale mossa gli conviene fare ? Se fa la mossa che lo porta in J o N pareggia, se va in L, M o P perde, se va in I, K, O vince. Perciò a P1 conviene fare una mossa che lo porti in uno stato che ha il massimo punteggio possibile.

Quindi per etichettare i nodi E, F, G, H basta prendere il massimo tra i punteggi dei loro figli. Ad esempio, il valore 0 del nodo G significa che P1 ha a disposizione una mossa che lo porta a pareggiare, il meglio che può ottenere visto che l'altra mossa lo porta a perdere. L'albero di gioco ora è:



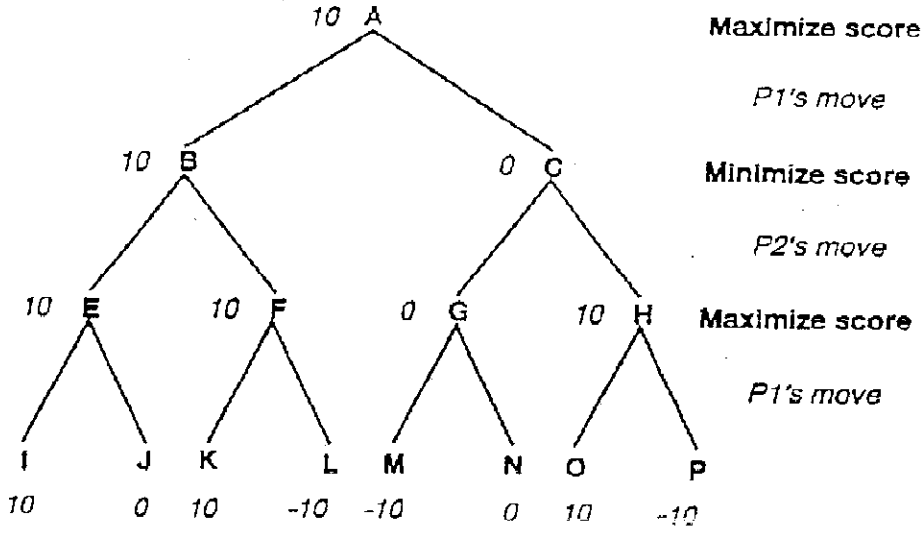
I nodi B e C sono stati in cui tocca a P2 muovere, gli conviene *minimizzare* il punteggio ottenibile da P1, e pertanto il punteggio dei nodi B e C è il minimo tra i loro figli:



Il valore 0 del nodo C significa che, a partire dallo stato denotato con il nodo C, P2 può fare una mossa che lo porterà al pareggio: infatti da C passa in G e a questo punto P1 non può fare di meglio che andare in N pareggiando.

Il valore 10 del nodo B invece denota che P2 è condannato a perdere, ovviamente nell'ipotesi che P1 faccia la mossa giusta.

Il nodo A denota lo stato iniziale in cui P1 muove, gli conviene fare la mossa che massimizza il punteggio, pertanto il valore associato al nodo A è 10 (massimo tra 10 e 0) e significa che P1 può, andando in B, vincere la partita. L'albero completo di punteggi è il seguente:



I nodi E, F, G, H, A sono detti *nodi max* in quanto si prende il massimo valore tra i loro figli, mentre i nodi B e C sono detti *nodi min* poichè si prende il minimo.

In modo informale, l'algoritmo minimax, può essere descritto come segue:

Per ogni nodo dell'albero:

Se il nodo è una foglia, restituisci il punteggio associato alla foglia
else
Se è un nodo max restituisci il massimo tra i punteggi dei suoi figli
else
Se è un nodo min restituisci il minimo tra i punteggi dei suoi figli

Pertanto il giocatore P1 (MAX) sceglie la mossa che lo porta al figlio che ha il massimo punteggio. P2 (MIN), a sua volta, effettua la mossa che lo porta al figlio con il minimo punteggio.

I punteggi sono calcolati a partire dalle foglie dell'albero e tutti i nodi dell'albero vengono esaminati.

In modo più formale, la procedura minimax viene descritta dalla seguente procedura ricorsiva:

```
function MINIMAX (N)
begin
    if N è una foglia then
        return (valore della foglia)
    else
        siano N_1, N_2, .... , N_k i figli di N
        se N è un nodo MAX then
                return max {MINIMAX (N_1), ... , MINIMAX (N_k)}
        else
                return min {MINIMAX (N_1), ... , MINIMAX (N_k)}
end MINIMAX
```

Ad esempio, applicando, per semplicità, la procedura al nodo C dell'esempio precedente, effettuando cioè la chiamata:

call MINIMAX (C)

abbiamo che C assume il valore zero. Infatti (MM sta per MINIMAX):

Livello ric.	Nodo	Call	Return	Max	Min
1	C	min {MM (G), MM (H)}			
2	G	max {MM (M), MM (N)}			
3	M		-10		
2	G				
3	N		0		
2	G			0	
1	C				
2	H	max {MM (O), MM (P)}			
3	O		10		
2	H				
3	P		-10		
2	H			10	
1	C				0

Potatura alfa-beta

La complessità della procedura minimax è esponenziale e il tempo di calcolo aumenta enormemente al crescere dell'albero di gioco. La procedura esamina tutti i nodi dell'albero. Si può migliorare l'efficienza della procedura riducendo il numero di nodi da esaminare. Lo si ottiene con un semplice accorgimento basato sulla considerazione generale che se so che un certo calcolo non influenza il risultato posso evitare di effettuarlo.

Ad esempio, se ho un'istruzione C del tipo:

if (A > 5 || B < 0)

allora, essendoci l'operatore OR, se la prima condizione è verificata non c'è bisogno di verificare la seconda. Analogamente, se ho l'istruzione:

if (A > 5 && B < 0)

se la prima condizione non è verificata, essendoci l'operatore AND, posso tralasciare di verificare la seconda.

Per quanto riguarda l'algoritmo minimax, riferiamoci all'albero precedente e supponiamo che abbiamo effettuato la call:

call MINIMAX (A)

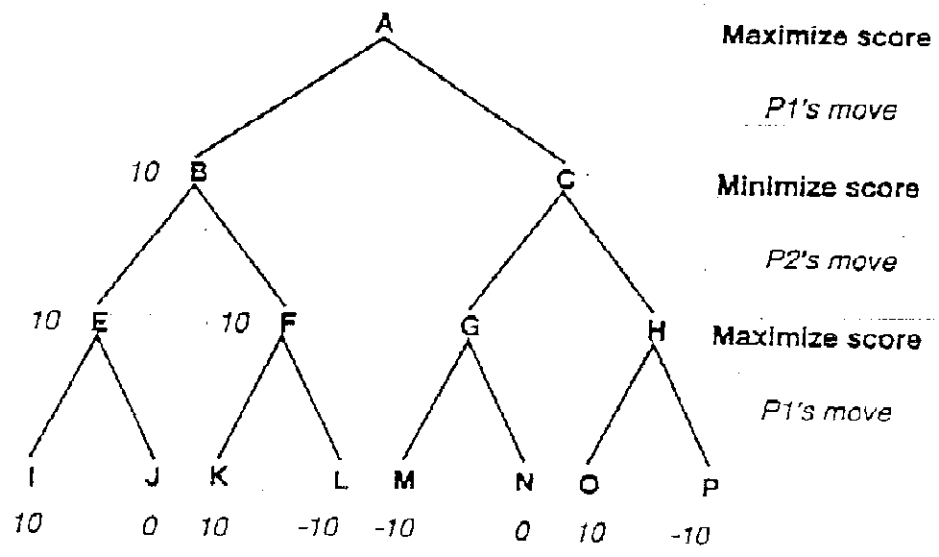
per effetto della call va in esecuzione l'istruzione:

return max {MINIMAX (B), MINIMAX (C)}

e viene fatta la call MINIMAX (B)

Altre call vengono effettuate e alla fine il punteggio 10 viene assegnato al nodo B.

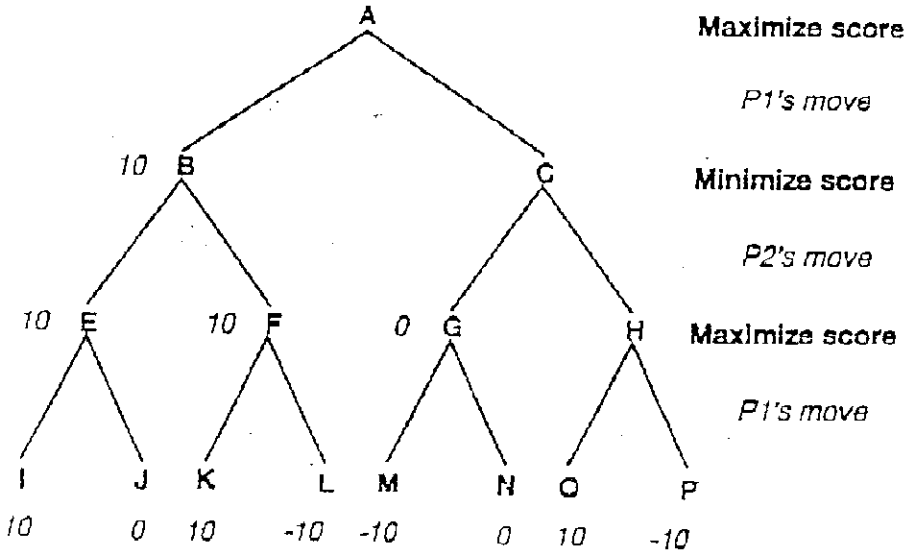
In questo momento l'albero di gioco si presenta così:



Possiamo ora dire qualcosa sul punteggio del nodo A ? Poichè P1 cerca di massimizzare il suo punteggio, possiamo affermare, senza esaminare il nodo C, che il punteggio del nodo A sarà *almeno* 10.

Ora viene fatta la call MINIMAX (C) la quale attiva la call MINIMAX (G)

In tal modo viene assegnato il punteggio 0 al nodo G e l'albero si presenta in questo modo:



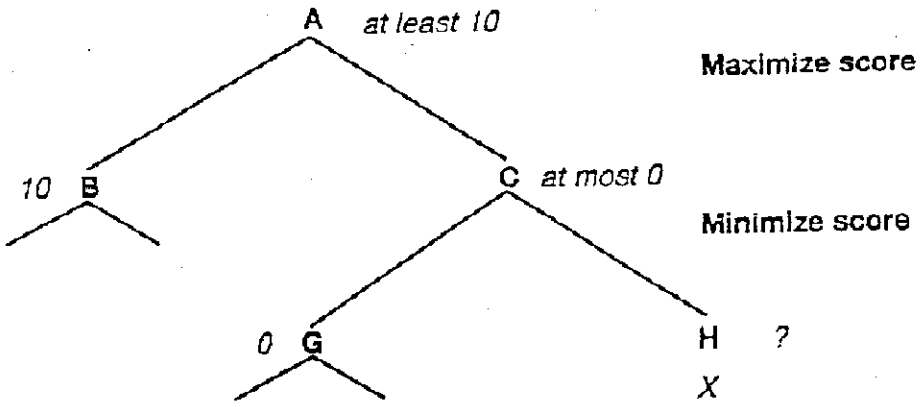
Che possiamo dire sul punteggio del nodo C ?

P2 vuole minimizzare il punteggio, quindi possiamo dire che il punteggio del nodo C sarà *al più* 0.

Queste due informazioni sui punteggi dei nodi A e C ci portano a concludere che è inutile ricavare il punteggio del nodo H, in quanto tale punteggio non può influire in alcun modo sul punteggio del nodo A.

Infatti, se il punteggio di H fosse negativo, il punteggio di C, nodo min, scenderebbe, ma A continuerebbe a valere 10 essendo A un nodo max. Se invece il punteggio di H fosse positivo, il valore di C resterebbe a 0 e il nodo A continuerebbe a valere 10.

La figura seguente mostra solo la parte di albero che ci interessa:



In conclusione, il sottoalbero avente per radice H può essere completamente ignorato. In questo esempio l'albero con radice H è piccolissimo, è chiaro che se non fosse così il risparmio di tempo di elaborazione diventerebbe significativo.

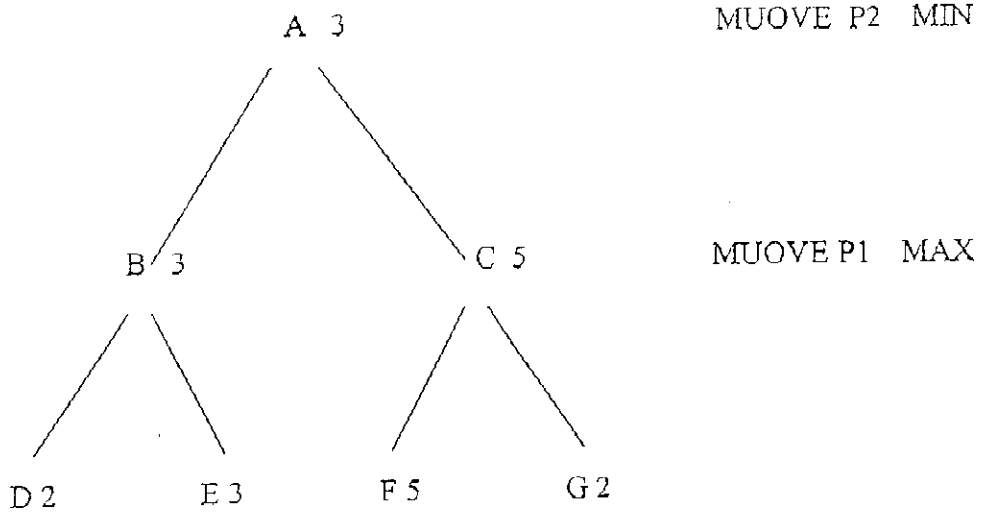
La tecnica di *potatura alfa-beta* utilizza i valori *al più* e *almeno* per individuare le parti dell'albero che è inutile esaminare. In media, usando la potatura il tempo di ricerca si dimezza.

La tecnica si basa sul fatto che un nodo max ha punteggio almeno pari al miglior punteggio dei figli finora esaminati, tale punteggio viene memorizzato in un parametro α.

Analogamente, un nodo min ha punteggio al più pari al peggior punteggio dei figli finora esaminati, tale punteggio viene memorizzato in un parametro β.

Se, come nell'esempio visto, il valore β di un nodo min (C al più 0) è minore del valore α del nodo padre (A almeno 10), allora i calcoli sui rimanenti figli (H) del nodo min possono essere tralasciati. Si dice che l'albero di gioco viene *potato*.

Simmetricamente, se il valore α di un nodo max è maggiore del valore β del nodo padre, i calcoli sui rimanenti figli del nodo max è inutile effettuarli. Ad esempio, consideriamo il seguente albero:



Supponiamo che il punteggio di B sia stato calcolato (3). Essendo A un nodo min possiamo dire che il suo punteggio sarà al più 3. Se ora il punteggio del nodo F è anch'esso calcolato (5) posso essere sicuro che C (nodo max) avrà un valore almeno 5.

Siamo nel caso in cui il valore α di un nodo max (C almeno 5) è maggiore del valore β del nodo padre min (A almeno 3) per cui i punteggi degli altri figli di C (il nodo G) possono essere trascurati.

Nel nostro esempio il nodo G è una foglia, ma, in presenza di un gioco diverso, potrebbe essere la radice di un grande sottoalbero.

Ebbene, il punteggio di G è inutile che lo ricavi in quanto se è minore di 5 (come nel nostro esempio) non influenza il valore di C e quindi di A. Se invece il punteggio di G fosse maggiore di 5, il punteggio di C cambierebbe ma quello di A rimarrebbe inalterato.

La tecnica che abbiamo ora delineato può essere incorporata nella procedura MINIMAX la cui complessità resta esponenziale ma si ha comunque un notevole miglioramento nel tempo di calcolo.

Il metodo di ricerca minimax per alberi di gioco può in teoria servire per determinare la strategia ottimale per un gioco come la dama o gli scacchi. Diciamo "in teoria" perchè occorrerebbe avere a disposizione memoria illimitata e tempo illimitato.

Il problema è che memoria e tempo non sono illimitati. Il numero di nodi da esplorare cresce esponenzialmente col numero di livelli dell'albero ed è approssimativamente pari al fattore di diramazione elevato al numero di livelli.

Nel caso degli scacchi, in media, il fattore di diramazione vale 35 e mediamente una partita consiste di 50 mosse per giocatore, abbiamo quindi un albero con circa 35^{100} nodi. Questo numero vale circa 10^{154} in quanto:

$$\log_{10} 35^{100} = 100 \log_{10} 35 \approx 100 \times 1.54 = 154 = 154 \log_{10} 10 = \log_{10} 10^{154}$$

Pertanto, un computer in grado di esaminare cento miliardi (10^{11}) di nodi al secondo, essendoci in un anno un numero di secondi inferiore a 10^{8}, potrebbe in un anno esaminare meno di 10^{19} nodi e impiegherebbe più di 10^{135} anni per esaminare l'intero albero.

Pertanto bisogna limitare la ricerca alle parti dell'albero che sono più promettenti. Nel caso degli scacchi occorre individuare le posizioni che non meritano di essere prese in considerazione. La tecnica di potatura alfa-beta ha proprio questo scopo e insieme ad altre tecniche più raffinate consente di arrivare a risultati soddisfacenti.

Ricordiamo che nel 1997 il programma Deep Blue, installato su un computer parallelo con centinaia di processori, sconfisse Kasparov, campione del mondo di scacchi. Due partite vinte da Deep Blue, una da Kasparov e tre pareggi. Questo successo, che fece aumentare il valore delle azioni IBM di 18 miliardi di dollari, fu dovuto in pari misura alla potenza dell'hardware e all'utilizzo di sofisticate tecniche software.

Più recentemente, nel 2002, il programma Fritz, montato su un PC, ha disputato un incontro (otto partite) contro il campione del mondo Kramnik ottenendo alla fine un pareggio.

Sommario

La risoluzione di molti problemi di AI richiede l'utilizzo di tecniche di *ricerca*. Ad esempio, partendo da uno stato iniziale, vogliamo individuare quale tra le possibili sequenze di azioni ci porta a uno stato finale. Uno stato specifico viene detto *nodo* nello *spazio di ricerca*, e la sequenza di stati è il *cammino* dallo stato iniziale al nodo.

Lo spazio di ricerca può essere un *albero* o un *grafo*. Esistono vari algoritmi che attraversano lo spazio di ricerca in modo diverso.

I metodi più semplici sono il *breadth first search* e il *depth first search* i quali attraversano lo spazio di ricerca in modo esaustivo. Nel BFS i cammini più brevi sono analizzati prima di quelli più lunghi. Nel DFS un cammino viene analizzato sino alla fine prima di fare backtracking e provare cammini alternativi.

I metodi di ricerca *euristica* (*hill climbing, best first, A**) fanno uso dell'informazione su quanto dista uno stato dallo stato finale per scegliere i cammini da esplorare per primi. In tal modo si cerca di evitare l'esplorazione di tutto lo spazio di ricerca al fine di giungere più rapidamente alla soluzione.

I sistemi AI di *pianificazione* usano una rappresentazione delle azioni più complessa rispetto agli algoritmi basati su semplici tecniche di ricerca. Il sistema MEA cerca azioni che riducono la *differenza* tra lo stato attuale e quello finale e, nel caso che manchi un'azione da applicare allo stato attuale, si pone nuovi sottoproblemi da risolvere.

Le tecniche di ricerca possono anche essere applicate a vari *giochi*, ora però si deve tener conto anche delle mosse dell'avversario. La *procedura minimax* permette di trovare la mossa migliore, tenendo in conto che l'avversario farà del suo meglio per vincere a sua volta.

6. Teoria dei giochi

Abbiamo visto in precedenza che il calcolo delle probabilità risulta utile per gestire situazioni in condizioni di incertezza. Ci siamo limitati a considerare il problema della diagnosi medica affrontato utilizzando il teorema di Bayes.

Abbiamo appena visto come le tecniche di ricerca possono essere utili per formalizzare giochi in cui due avversari, a turno, fanno delle mosse. I giochi considerati sono ad *informazione perfetta*, cioè ogni giocatore ha un quadro completo della situazione e deve decidere qual'è la mossa migliore. L'algoritmo minimax è la risposta al problema, non c'è nulla di probabilistico, vi sono però le limitazioni pratiche cui abbiamo accennato.

Vi sono però giochi per la cui soluzione il calcolo delle probabilità assume un ruolo determinante. Vedremo nel seguito alcuni elementi di *teoria dei giochi* limitandoci a giochi che prevedono *una sola* mossa e in cui due giocatori effettuano *contemporaneamente* le loro mosse.

Il fatto che si parli di giochi che si esauriscono in una sola mossa porta a pensare che tali giochi possano essere utili solo come passatempo. Nulla di più sbagliato: alcuni dei giochi che considereremo, pur essendo molto più semplici rispetto, ad esempio, agli scacchi, hanno un'importanza molto maggiore per le applicazioni al mondo che ci circonda.

In altri termini, mentre studiare gli scacchi non ha conseguenze sui tanti problemi del mondo reale, la teoria dei giochi ha forti implicazioni in aree diversissime tra loro, quali l'economia, l'informatica, la politica, la sociologia, la finanza, le relazioni internazionali, i sistemi elettorali, le strategie militari, l'ambiente, l'evoluzione, la psicologia, lo sport, la biologia, e così via.

Questo accade in quanto la teoria dei giochi, al di là delle apparenze, ha un obiettivo estremamente ambizioso: *studiare il comportamento umano.*

Più precisamente, essa ha obiettivi molto più ampi della formalizzazione dei giochi tra persone, il suo scopo vero è di *razionalizzare il comportamento* delle persone e dei gruppi in situazioni complesse.

I "giochi" sono una metafora scientifica per un vasto insieme di interazioni umane in cui i risultati dipendono dalle strategie poste in atto da due o più persone che cercano di ottimizzare la loro "vincita".

La teoria dei giochi considera sia giochi *a somma zero* sia giochi *non a somma zero*. Inizialmente esamineremo giochi del primo tipo allo scopo di familiarizzarci con i concetti di base della teoria.

Le implicazioni sul mondo reale di tali giochi sono molto limitate, però ricaveremo interessanti informazioni sulle proprietà di tali giochi. Successivamente, considereremo i giochi a somma diversa da zero e mostreremo come molti di essi rivestono un concreto interesse applicativo.

Giochi a somma zero

Esempio 1

Consideriamo la morra a due dita: vi sono due giocatori, R e C, che simultaneamente mostrano un dito o due dita. Se la somma delle dita è pari (2 o 4), R riceve da C 2 o 4 euro, se la somma è dispari (3) C riceve da R 3 euro.

Questo gioco, molto semplice, si presta a essere formalizzato in modo preciso, ricavando quindi informazioni interessanti.

Innanzitutto, il gioco può essere completamente descritto mediante la seguente matrice (il segno - indica che C riceve da R):

	uno	due
uno	2	-3
due	-3	4

Invece di mostrare le dita, i giocatori scelgono, contemporaneamente, R una riga della matrice e C una colonna. Ad esempio, se R sceglie la riga 2 e C la colonna 2 allora R riceve da C 4 euro, se R sceglie la 1 e C la colonna 2, C riceve da R 3 euro. E' chiaro che scegliere righe e colonne della matrice è assolutamente equivalente a mostrare le dita. Quando il gioco viene rappresentato mediante una matrice si dice che è in *forma normale.*

In generale, ogni matrice può definire un gioco come segue:

1) vi sono due giocatori, R e C

2) una partita consiste nella scelta da parte di R di una riga della matrice e, simultaneamente, da parte di C di una colonna

3) dopo ogni partita, R riceve da C l'ammontare indicato dall'elemento all'intersezione riga-colonna, ammontare negativo significa che C riceve da R

Nel linguaggio usuale il termine "vincita" denota un guadagno e non si dice "Tizio ha vinto -20 euro" per indicare che Tizio ha perso 20 euro. Nella teoria dei giochi, invece, i termini "vincere" e "vincita" possono denotare anche una perdita.

Nel seguito, per chiarezza, useremo i termini "vincere" e "vincita" per indicare sia guadagni che perdite. Non c'è nessuna ambiguità in quanto dire "C vince -4 euro" significa che C ha perso 4 euro. Invece useremo i termini "guadagno" e "perdita" nel senso usuale.

I giochi che considereremo sono detti *"giochi con due giocatori a somma zero"* in quanto sono coinvolti due giocatori e ad ogni partita ciò che guadagna un giocatore viene perso dall'altro.

A questo punto ci possiamo porre alcune domande interessanti:

1) esistono mosse migliori da fare sempre e mosse peggiori da evitare sempre ?

Obiettivo di R è guadagnare il massimo possibile. Pertanto sembra giusto guardare la matrice, individuare la riga che contiene il massimo valore, e scegliere sempre questa riga. Nel nostro esempio il massimo valore (4) è contenuto nella seconda riga. Il problema è che R vince 4 euro solo se C sceglie la seconda colonna. E' chiaro che C si accorge che R sceglie sempre la seconda riga e quindi sceglie la prima colonna, in tal modo R perde 3 euro ogni volta. A questo punto R cambia mossa a sua volta e così via. Questo criterio pertanto non è valido.

Anche scegliere sempre la riga 1 non è opportuno per R, in quanto C sceglierebbe la colonna 2 guadagnando ogni volta 3 euro. Un discorso analogo si può fare per C (può scegliere colonna 1 o colonna 2) e la conclusione è che se un giocatore sceglie di fare sempre la stessa mossa, l'altro giocatore può trarre vantaggio da tale scelta.

Ad esempio, la mossa peggiore per C sembra essere la scelta della seconda colonna, in quanto l'elemento 4 indica la vincita di 4 euro da parte di R. Per evitare di perdere 4 euro basta che C eviti di scegliere la seconda colonna, ma allora non può che scegliere la prima e a questo punto R sceglie la prima riga e vince 2 euro.

Lo stesso accadrebbe se uno dei giocatori usasse una regola per scegliere la mossa da fare: ad esempio, se R usasse la strategia "tre volte la riga 1 e poi cinque volte la riga 2", C alla lunga se ne accorgerebbe e ne trarrebbe vantaggio.

Occorre pertanto che le mosse vengano scelte in modo casuale, ad esempio lanciando una moneta: testa=riga 1, croce=riga 2. Il giocatore C, a sua volta, potrebbe, ad esempio, lanciare un dado: numero pari=colonna 1, numero dispari=colonna 2.

Osserviamo che questo gioco può essere visto come la versione semplificata di un *gioco di ispezione*. Un ispettore sceglie un giorno per visitare un luogo (ad esempio, ristorante, impianto di armi biologiche, supermercato) e la persona responsabile del luogo sceglie un giorno per nascondere tutto ciò che non è in regola. L'ispettore vince se i giorni sono diversi, il responsabile vince se coincidono.

Esempio 2

Le cose sarebbero diverse se la matrice del gioco fosse questa:

	uno	due
uno	-2	4
due	2	3

In tal caso al giocatore R conviene scegliere sempre la seconda riga che gli garantisce di guadagnare ogni volta 2 o 3 euro. E' vero che il 4 alla prima riga rappresenta un guadagno maggiore per R, ma questa vincita si verificherebbe se C scegliesse la seconda colonna, cosa che certamente non conviene a C.

Il giocatore C perde sicuramente a ogni partita, ma perde meno se ogni volta sceglie la prima colonna. Per cui la mossa migliore per R è scegliere la riga 2 per C la colonna 1. Chiaramente il gioco descritto da questa matrice non è equo e nessuno prenderebbe il posto di C.

Strategie di gioco

In generale, data un matrice di gioco di dimensione k x m, una *strategia di gioco* per R consiste nello scegliere le righe con una certa distribuzione di probabilità, cioè scegliere la riga 1 con probabilità p_1, la riga 2 con probabilità p_2, ..., la riga k con probabilità p_k.

Pertanto, la strategia di gioco di R è individuata dal vettore $p = (p_1, p_2, \ldots, p_k)$. Ad esempio, se la matrice ha due righe e R lancia una moneta per scegliere la riga, la sua strategia è individuata dal vettore $p = (1/2, 1/2)$.

Analogamente, la strategia di gioco di C, in presenza di m colonne, è descritta dal vettore di probabilità $q = (q_1, q_2, \ldots, q_m)$.

Una strategia che contiene nel vettore tutti 0 e un solo 1, in cui cioè R o C decidono di scegliere sempre la stessa riga o colonna è detta una *strategia pura*. In caso contrario diciamo che è una *strategia mista*.

Valore atteso

Data la *variabile aleatoria* X che assume i valori $x_1, x_2, \ldots, x_n$ con probabilità $p_1, p_2, \ldots, p_n$, il *valore atteso* di X è:

$$E(X) = \sum_{i=1}^{n} x_i p_i$$

Ad esempio, se X denota il risultati del lancio di un dado:

$$E(X) = 1 \frac{1}{6} + 2 \frac{1}{6} + 3 \frac{1}{6} + 4 \frac{1}{6} + 5 \frac{1}{6} + 6 \frac{1}{6} = \frac{21}{6} = 3.5$$

Supponiamo di avere una matrice di gioco k x m, la strategia del giocatore R sia descritta dal vettore $p = (p_1, p_2, \ldots, p_k)$ e quella del giocatore C dal vettore $q = (q_1, q_2, \ldots, q_m)$.

Allora la probabilità che venga scelta da R la riga 1 e da C la colonna 1 è data da $p_1 q_1$. Infatti gli eventi "R sceglie la riga 1" e "C sceglie la colonna 1" sono eventi indipendenti e quindi la probabilità che si verifichino entrambi è data dal loro prodotto.

Pertanto, il valore atteso della variabile aleatoria X che assume come valori gli elementi x_{ij} della matrice è dato da:

$$E(X) = \sum_{i=1}^{k} \sum_{j=1}^{m} x_{ij} p_i q_j$$

Tale valore rappresenta la vincita attesa di R e di C ed è anche detto *vincita media*. Se il valore è positivo si tratta di un guadagno per R e una perdita per C, se è negativo una perdita per R e un guadagno per C.

Possiamo anche ottenere i valori attesi per le singole righe e colonne. Ad esempio, fissata la riga i, il valore atteso per la riga i è dato da:

$$E(\text{riga i}) = \sum_{j=1}^{m} x_{ij} p_i q_j$$

Analogamente, fissata la colonna j, il valore atteso per tale colonna è:

$$E(\text{colonna } j) = \sum_{i=1}^{k} x_{ij}\ p_i\ q_j$$

Ad esempio, consideriamo la matrice dell'Esempio 2:

	uno	due
uno	-2	4
due	2	3

Abbiamo visto che a R conviene scegliere la riga 1 con probabilità zero e la riga 2 con probabilità 1. C, a sua volta, sceglie la colonna 1 con probabilità 1 e la colonna 2 con probabilità zero. R così vince sempre 2 euro e infatti il valore atteso della variabile X è dato da:

$$E(X) = -2 \times 0 \times 1 + 4 \times 0 \times 0 + 2 \times 1 \times 1 + 3 \times 1 \times 0 = 2$$

Supponiamo che R scelga le righe con probabilità (1/2, 1/2) e C scelga le colonne con probabilità (1/2, 1/2), allora la vincita media di R associata alla scelta della riga 2 è:

$$E(\text{riga } 2) = \sum_{j=1}^{2} x_{2j}\ p_2\ q_j = 2 \times 1/2 \times 1/2 + 3 \times 1/2 \times 1/2 = 5/4 = 1.25$$

Supponiamo ora che R abbia scelto la riga 2, allora $p_2 = 1$ e la vincita media di R dipende solo dalla strategia di C. Se C sceglie le colonne con probabilità (1/2, 1/2) allora il valore atteso è:

$$E(\text{riga } 2) = 2 \times 1 \times 1/2 + 3 \times 1 \times 1/2 = 5/2 = 2.5$$

Simmetricamente, una volta che C abbia scelto la colonna 1, allora $q_2 = 1$ e la sua vincita media dipende solo dalla strategia di R. Se R sceglie le righe con probabilità (1/2, 1/2) allora il valore atteso è:

$$E(\text{colonna } 1) = \sum_{i=1}^{2} x_{i1}\ p_i\ q_2 = -2 \times 1/2 \times 1 + 2 \times 1/2 \times 1 = 0$$

Ritorniamo al gioco delle dita (Esempio 11) e poniamoci la seconda domanda:

2) Una volta assodato che nel gioco delle dita non esiste una mossa migliore da fare sempre possiamo dire che il gioco è equo ?

A prima vista il gioco appare equo, in quanto osserviamo che i due casi favorevoli a R gli fanno guadagnare 2+4=6 euro, mentre i due favorevoli a C gli portano in cassa 3+3=6 euro.

Supponendo ora che i due giocatori si servano di una moneta o di un dado per scegliere la mossa, le mosse hanno ciascuna probabilità = 1/2, l'evento riga-colonna ha probabilità pari al prodotto delle due mosse, essendo esse eventi indipendenti, per cui la probabilità è 1/2 x 1/2 = 1/4, cioè i quattro casi possibili (R1,C1), (R1,C2), (R2,C1), (R2,C2) hanno ciascuno probabilità = 1/4.

Pertanto, per la legge dei grandi numeri, dopo un grande numero di partite i quattro casi possibili si saranno verificati un numero x di volte approssimativamente uguale: R avrà guadagnato $2x + 4x = 6x$ euro, C avrà incassato $3x + 3x = 6x$ euro, il gioco appare equo.

Come in precedenza, calcoliamo il valore atteso della variabile X i cui valori sono gli elementi della matrice e abbiamo:

$$E(X) = 2\,\frac{1}{4} + (-3)\,\frac{1}{4} + (-3)\,\frac{1}{4} + 4\,\frac{1}{4} = \frac{(2 - 3 - 3 + 4)}{4} = 0$$

il che significa che i due giocatori, in media, non vincono e non perdono.

La formalizzazione che esamineremo ci mostrerà i seguenti fatti sorprendenti:

a) il gioco *non* è equo in quanto favorisce C

b) C è in grado di guadagnare a patto che adotti una strategia di gioco diversa da quella indicata, scegliendo cioè la colonna 1 con probabilità 7/12 e la colonna 2 con probabilità 5/12.

Per adottare tale strategia C potrebbe, ad esempio, mischiare ogni volta 12 carte, 7 di cuori e 5 di quadri, e prenderne a caso una: se la carta è cuori sceglie la colonna 1, se è quadri la colonna 2.

c) R è condannato a perdere ma può minimizzare la perdita se sceglie, anche lui, la riga 1 con probabilità 7/12 e la riga 2 con probabilità 5/12.

d) adottando la strategia indicata, il valore atteso di X è -1/12 il che significa che C vincerà, in media, dopo n partite, n/12 euro e, se n è grande, la cifra non è disprezzabile.

Infatti, il valore atteso della variabile X è:

$$E(X) = 2\,\frac{7}{12}\,\frac{7}{12} + (-3)\,\frac{7}{12}\,\frac{5}{12} + (-3)\,\frac{5}{12}\,\frac{7}{12} + 4\,\frac{5}{12}\,\frac{5}{12} =$$

$$= \frac{98}{144} - \frac{105}{144} - \frac{105}{144} + \frac{100}{144} = \frac{-12}{144} = \frac{-1}{12}$$

Il valore del gioco

Consideriamo la matrice dell'Esempio 2:

	uno	due
uno	-2	4
due	2	3

Abbiamo visto che il gioco favorisce R il quale, scegliendo sempre la seconda riga, si garantisce un guadagno di 2 euro ogni partita.

Vogliamo introdurre il concetto di *valore del gioco*. Tale concetto non può essere legato alla particolare scelta dei giocatori, ad esempio, è vero che R può vincere 4 euro scegliendo la riga 1, ma tale vincita è legata alla scelta da parte di C della colonna 2.

E' più ragionevole considerare valore del gioco l'esito della partita nel momento in cui entrambi i giocatori fanno la scelta più razionale.

Abbiamo visto che la scelta migliore per R e C sono riga 2 e colonna 1 per cui il valore del gioco viene a essere 2. Il fatto che sia un valore positivo significa che il gioco favorisce R il quale incassa, ad ogni partita, 2 euro.

Abbiamo visto in precedenza che il valore atteso della variabile X è 2, per cui il valore del gioco coincide con il valore atteso di X.

Nell'Esempio 1 la cui matrice è:

	uno	due
uno	2	-3
due	-3	4

le cose vanno diversamente, non esistono mosse migliori da fare sempre, righe e colonne vanno scelte con una certa probabilità, se tale probabilità è 1/2 il valore atteso della variabile X è zero.

Appare ragionevole considerare il valore atteso di X come il valore del gioco, il fatto che E(X) valga zero significa che ciascun giocatore, in media, dopo n partite, ha vinto 0 euro per cui è il segnale che il gioco è equo, cioè non favorisce nè R nè C.

E' giusto prendere il valore $E(X) = 0$ come valore del gioco ?

A prima vista, sì, però le cose non stanno così. Infatti il valore di $E(X)$ è legato ai valori assunti da X e alle loro probabilità. Il fatto che $E(X) = 0$ dipende dalla scelta della strategia $p = (1/2, 1/2)$.

Se cambia la strategia, cioè scegliamo righe e colonne con probabilità diverse da 1/2, il valore di $E(X)$ cambia. Abbiamo affermato prima e dimostreremo più avanti che se R e C utilizzano le strategie ottimali scegliendo righe e colonne con probabilità (7/12, 5/12) il valore atteso diventa -1/12 e cioè il gioco è favorevole a C.

Il *teorema fondamentale della teoria dei giochi*, dovuto a von Neumann, afferma che in un qualsiasi gioco con due persone e a somma zero esistono strategie ottimali cioè tali che massimizzano il guadagno di un giocatore e minimizzano la perdita dell'altro. Il valore del massimo guadagno coincide con quello della minima perdita e rappresenta il *valore del gioco*.

Se il valore del gioco è zero si dice che il gioco è *equo*.

In altri termini, il valore del gioco è la cifra che uno dei giocatori in media guadagnerà utilizzando la sua migliore strategia e corrisponde a quanto il suo avversario in media perderà utilizzando anche lui la sua migliore strategia.

Possiamo anche dire che, in presenza di strategie ottimali, il valore del gioco coincide con il valore atteso.

Prima di formalizzare meglio il discorso, vediamo altri due esempi. Il primo esempio lo discutiamo prima in modo intuitivo e, diversamente dal gioco delle dita, vedremo che la formalizzazione confermerà quanto abbiamo dedotto in modo informale.

Esempio 3

Due ragazzi, R e C, coprono ciascuno una moneta e poi la scoprono contemporaneamente: se entrambe sono testa o entrambe sono croce vince R un euro, se differiscono l'euro lo vince C.

Anche in questo caso, fare sempre la stessa mossa non conviene: se, ad esempio, R mostra sempre croce, C mostrerà testa e vince. Bisogna pertanto alternare in modo casuale le mosse. Con quali probabilità ?

Se R mostra testa con probabilità, ad esempio, p=6/10, C, dopo un pò, si accorgerà che R mostra testa più spesso di croce e a questo punto comincia a mostrare sempre croce. E' chiaro che in tal modo dopo n partite C avrà ricevuto da R, in media, n 6/10 euro e R da C n 4/10 euro con una perdita netta per R di n 2/10 euro.

Analogamente, se R mostra testa con probabilità, ad esempio, 4/10 C, alla lunga, si accorgerà che R mostra croce più spesso di testa e a questo punto comincia a mostrare sempre testa. In tal modo dopo n partite C avrà vinto, in media, n 6/10 euro e R n 4/10 euro con una perdita netta per R di n 2/10 euro.

Pertanto l'unico modo per R di massimizzare la sua vincita è di giocare testa e croce con la stessa probabilità p=1/2. Ovviamente il discorso fatto per R vale anche per C il quale anche lui può minimizzare la sua perdita giocando testa e croce con p=1/2.

Il gioco appare equo e in effetti vedremo che non favorisce nè R nè C.

La matrice di gioco è:

	testa	croce
testa	1	-1
croce	-1	1

Il valore atteso è dato da:

$$E(X) = \frac{1}{2}(1) + \frac{1}{2}(-1) + \frac{1}{2}(-1) + \frac{1}{2}(1) = 0$$

e viene confermato che il gioco non è molto intelligente ma equo.

Vedremo più avanti che le conclusioni alle quali siamo giunti ragionando sul problema saranno confermate passando alla sua formalizzazione.

Anche la morra a due dita può essere rappresentata da questa matrice nell'ipotesi che la vincita consista sempre in un euro (somma pari R riceve un euro, somma dispari R paga un euro):

	uno	due
uno	1	-1
due	-1	1

Come si vede, cambiando le vincite la morra è diventata un gioco equo.

E' utile riaffermare che se la teoria dei giochi servisse solo a formalizzare giochi del tipo visto o anche più complessi, la sua utilità sarebbe molto limitata. Invece la teoria dei giochi viene utilizzata in moltissime aree e cerca di suggerire un approccio razionale in situazioni in cui ci sono dei concorrenti che si pongono un obiettivo.

Ovviamente la realtà è molto complessa e una matrice di gioco può solo approssimare una situazione reale, però anche gli aspetti elementari della teoria ai quali ci limiteremo potranno dare un'idea della varietà di situazioni che si possono affrontare.

A tal fine facciamo ora un esempio che descrive una situazione estremamente semplificata ma utile per rendersi conto che la teoria dei giochi è applicabile anche in aree molto lontane dai giochi.

Esempio 4

Supponiamo di avere due aziende (R e C) che vendono acqua minerale nella stessa area geografica e che hanno dei costi mensili ciascuna di 5000 euro. Ogni azienda deve decidere se vendere ogni bottiglia a 1 euro o a 2 euro.

Facciamo queste ipotesi semplificative:

a) al prezzo di 2 euro si possono vendere, in un mese, 5000 bottiglie ricavando 10,000 euro

b) al prezzo di 1 euro si possono vendere, in un mese, 10,000 bottiglie ricavando 10,000 euro

c) se entrambe le aziende applicano lo stesso prezzo, le vendite si ripartiscono equamente tra le due aziende (cioè, se il prezzo è 2 euro, ciascuna vende 2500 bottiglie e ricava 5000 euro, se il prezzo è 1 euro, ciascuna vende 5000 bottiglie e ricava 5000 euro)

d) se una sola azienda applica il prezzo di 2 euro, non vende nulla e l'altra vende, nel mese, 10,000 bottiglie

e) nella matrice di gioco inseriamo i ricavi meno il costo mensile (5000 euro)

La matrice di gioco viene pertanto ad essere:

	1 euro	2 euro
1 euro	0	5000
2 euro	-5000	0

In questo gioco non ci sono euro che un'azienda versa all'altra. Il gioco è comunque a somma zero in quanto, ad esempio, i 5000 euro guadagnati da R rappresentano anche la perdita subita da C.

Vediamo il ragionamento di R:

vendo a 2 euro: se C decide di vendere a 1 perdo 5000 euro, se C vende anche lei a 2 guadagno zero.

vendo a 1 euro: se C vende a 1 guadagno zero, se C vende a 2 guadagno 5000 euro.

Pertanto vendendo a 2 o perdo o guadagno zero, vendendo a 1 o guadagno zero o guadagno 5000 pertanto mi conviene vendere a 1 euro.

Ovviamente lo stesso ragionamento fa C e conclude anche lei che è razionale vendere a 1 euro.

Più avanti vedremo che effettivamente, nelle ipotesi semplificative fatte, vendere a 1 euro è la scelta più razionale. Osserviamo che l'esempio è poco realistico in quanto la scelta migliore porta a una vincita nulla.

Strategie minimax

Esempio 5

Consideriamo di nuovo il gioco delle dita la cui matrice è:

	uno	due
uno	2	-3
due	-3	4

Abbiamo visto prima che non esistono mosse migliori o peggiori. Bisogna alternare probabilisticamente le mosse possibili. Nel caso in cui righe e colonne vengono scelte con probabilità 1/2, abbiamo visto che, in media, nessuno dei due giocatori guadagna o perde e il gioco sembra equo.

Il fatto interessante è che invece il gioco favorisce C il quale può guadagnare, in media, 1/12 di euro ad ogni partita a patto che adotti una strategia opportuna.

Per arrivare all'individuazione della strategia ottimale dobbiamo innanzitutto vedere cosa sono le *strategie minimax*.

Una volta convintisi R e C che non esistono mosse migliori da fare, i due giocatori si possono chiedere se esiste comunque una strategia che garantisca loro la minima perdita, *qualsiasi* sia la mossa dell'avversario.

Vediamo prima il ragionamento di C. In sostanza, C vuole rischiare il meno possibile, trascurando anche possibili guadagni, e vuole individuare una strategia che gli assicuri la minima perdita *indipendentemente* dalla scelta che effettua R.

Individuare questa strategia è semplice. Infatti se C trova i massimi valori in ogni colonna (2 e 4) questi rappresentano le massime vincite di R e quindi le massime perdite cui è esposto C: C perde 2 euro se C sceglie la colonna 1 e R la riga 1, C perde 4 euro se C sceglie la colonna 2 e R la riga 2. Ricordiamoci che C vuole perdere il meno possibile e questo gli viene garantito, *qualunque* sia la mossa di R, se sceglie la colonna che contiene il minimo (2) tra questi massimi, cioè la colonna 1.

Pertanto la *strategia minimax* di C è:

Scegliere la colonna che contiene il minimo tra i massimi valori di ciascuna colonna.

Scegliendo la colonna 1 C pone un limite superiore al *valore del gioco*: il *valore minimax* del gioco è 2. Qualunque scelta faccia R, C non potrà perdere più di 2 euro.

Osserviamo che non stiamo dicendo che il valore del gioco è 2 (vedremo fra poco che il valore è -1/12), stiamo solo affermando che il valore del gioco certamente non è maggiore di 2, in quanto che esiste la strategia minimax che, adottata da C, gli fa perdere al più 2 euro ad ogni partita.

Per quanto riguarda R, ricordiamoci che anche lui è un tipo prudente e vuole individuare una strategia che, *qualsiasi* sia la scelta di C, gli faccia avere la minima perdita. Il ragionamento che fa R è esattamente lo stesso, solo che, avendo i guadagni di R segno positivo e i guadagni di C segno negativo, la strategia di R è speculare a quella di C.

In sostanza R va a vedere i valori minimi in ciascuna riga (i massimi guadagni di C e quindi le massime perdite per R) e prende il massimo tra questi minimi e questo rappresenta la minima perdita di R a prescindere dalla scelta di C.

Ora, il minimo della riga 1 è -3, il minimo della riga 2 è -3. Il massimo tra questi minimi è -3 ed è contenuto sia nella riga 1 che nella 2. Pertanto scegliere una delle due righe è equivalente ai fini della minimizzazione della perdita di R. Allora la *strategia maximin* che adotta R è:

Scegliere la riga che contiene il massimo tra i minimi valori di ciascuna riga.

Si dice che, scegliendo la riga 1 o 2, R pone un limite inferiore al *valore del gioco*: il *valore maximin* del gioco è -3. Qualunque scelta faccia C, R non potrà perdere più di 3 euro.

Ribadiamo che non stiamo dicendo che il valore del gioco è -3, stiamo solo affermando che il valore del gioco certamente non è inferiore a -3, in quanto che esiste la strategia maximin che, adottata da R, gli fa perdere al più 3 euro ad ogni partita.

Questa è la *strategia maximin* di R Si tratta della strategia meno rischiosa che lo cautela contro perdite maggiori.

Riassumendo, la strategia minimax di C è scegliere la colonna 1: la perdita massima di C sarà di 2 euro. La strategia maximin di R è scegliere la riga 1 o 2: in tal modo R può essere sicuro che al massimo perderà 3 euro. Il valore maximin del gioco è -3, il valore minimax del gioco è 2. Il valore v del gioco non lo abbiamo ancora individuato ma certamente possiamo affermare che è compreso fra -3 e 2.

Per semplicità, le due strategie vengono denotate complessivamente come le *strategie minimax*.

Osserviamo subito che adottare stabilmente le strategie minimax serve solo ad arricchire uno dei giocatori. Infatti se R sceglie sempre la riga 1 e C la colonna 1, R ad ogni partita incassa 2 euro. E' chiaro che la cosa non piace a C il quale comincia a scegliere la colonna 2 per incassare 3 euro, R si accorge del cambio di strategia e comincia a scegliere la riga 2 per vincere 4 euro, e così via.

Se invece, all'inizio del gioco, R sceglie sempre la riga 2 e C la colonna 1, C guadagna ad ogni partita 3 euro. R a questo punto sceglie la riga 1 per vincere 2 euro, C replica scegliendo la colonna 2 per vincere 3 euro, e così via.

Teorema di von Neumann

Finora, certamente, tutte le cose che abbiamo detto non ci hanno portato a risultati brillanti, abbiamo solo concluso che le strategie minimax vanno bene per i paurosi e non sono per nulla utili, non abbiamo individuato le strategie ottimali, abbiamo solo ricavato che il valore del gioco è compreso fra i valori maximin e minimax.

La domanda che allora ci poniamo è:

Esiste una strategia che garantisca a R una vincita *media* maggiore del valore maximin e una strategia che assicuri a C una vincita *media* inferiore al valore minimax ?

Il teorema fondamentale della teoria dei giochi, dovuto a von Neumann e anche detto *teorema minimax*, afferma che:

Per ogni matrice di gioco esiste una coppia di strategie ottimali tali che il valore maximin coincide con il valore minimax. Tale valore è il valore del gioco.

Prima di vedere come ricavare le strategie ottimali e il valore del gioco, esaminiamo un altro esempio.

Esempio 6

R e C scrivono contemporaneamente un numero da 1 a 3 e lo mostrano. Se la somma dei due numeri è pari (2, 4, 6), R riceve da C il corrispettivo in euro, se la somma è dispari (3, 5) R paga a C 3 o 5 euro. La matrice è:

	1	2	3
1	2	-3	4
2	-3	4	-5
3	4	-5	6

R vuole perdere il meno possibile. Pertanto gli conviene andare a vedere i valori minimi in ciascuna riga (le massime perdite per R), prendere il massimo tra questi minimi e questo rappresenta la minima perdita di R a prescindere dalla scelta di C.

Nel nostro esempio, il minimo della riga 1 è -3, il minimo della riga 2 è -5, il minimo della riga 3 è -5. Il massimo tra questi minimi è -3 ed è contenuto nella riga 1.

Pertanto R sceglie la riga 1: infatti se C sceglie la colonna 1, R guadagna 2, se C sceglie la 3, R guadagna 4, il peggio che può capitare a R è che C scelga la colonna 2 nel qual caso R perde 3. E' comunque la scelta migliore in quanto la scelta della riga 2 o 3 può portare alla perdita di 5 euro (riga 2- colonna 3, riga 3 - colonna 2).

Scegliendo la riga 1, R pone un limite inferiore al *valore del gioco*: qualunque scelta faccia C, R non potrà perdere più di 3 euro. Questa è la *strategia maximin* di R Il *valore maximin* del gioco è -3.

Il criterio che deve adottare C per minimizzare la sua perdita è simmetrico a quello di R. C trova i massimi in ciascuna colonna, questi rappresentano le massime perdite per C, se ora prende il minimo tra questi massimi questo valore rappresenta la massima perdita che C può subire *qualunque* sia la scelta di R.

Nel nostro esempio il massimo della colonna 1 e 2 è 4, il massimo della colonna 3 è 6. Pertanto C sceglie la colonna 1 o la 2 perchè in tal modo al massimo deve dare a R 4 euro. Se C scegliesse la colonna 3, R potrebbe scegliere la riga 3 e C perderebbe 6 euro.

Scegliendo la colonna 1 o 2 C pone un limite superiore al valore del gioco: il *valore minimax* del gioco è 4.

In conclusione, la strategia maximin di R è scegliere la riga 1: in tal modo R può essere sicuro che al massimo perderà 3 euro.
La strategia minimax di C è scegliere la colonna 1 o la 2: la perdita massima di C sarà di 4 euro.
Il valore del gioco sarà compreso fra -3 e 4.

Anche in questo esempio, servirsi sempre delle strategie minimax non funziona. Infatti se R sceglie sempre la riga 1 e C la colonna 2, C a ogni partita incassa 3 euro. R, stanco di perdere, comincia a scegliere la riga 2 per incassare 4 euro, C si accorge del cambio di strategia e comincia a scegliere la colonna 3 per vincere 5 euro, e così via.

Anche in questo caso le strategie migliori per R e C sono strategie probabilistiche. Si dimostra che il gioco è equo (v = 0) e che le strategie ottimali sono per R scegliere le righe 1, 2, 3 e per C le colonne 1, 2, 3 con probabilità 1/4, 1/2, 1/4, rispettivamente.

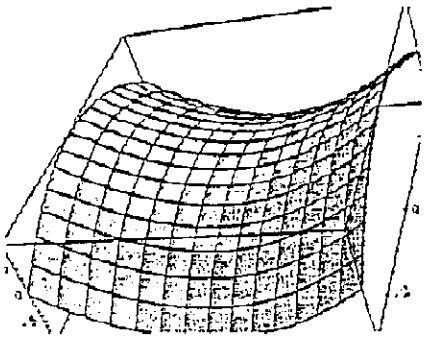
Per realizzare queste strategie R e C possono usare, ad esempio, due carte di quadri, una di cuori e una di fiori, mescolarle e prenderne una a caso, se esce cuori riga o colonna 1, se esce quadri riga o colonna 2, se esce fiori riga o colonna 3.

A differenza degli altri esempi visti sinora, in questo caso abbiamo una matrice 3 x 3 e trovare le strategie ottimali richiede l'utilizzo del *metodo del simplesso*, un classico algoritmo di *programmazione lineare* che fa parte del corso di Ricerca operativa.

Poichè i nostri obiettivi sono molto limitati, ci limiteremo nel seguito ad esaminare solo giochi particolari e cioè i giochi strettamente determinati, i giochi 2 x 2 e i giochi descritti da matrici riconducibili a matrici 2 x 2.

Giochi strettamente determinati.

Un gioco è detto *strettamente determinato* se la sua matrice ha un elemento che è contemporaneamente minimo nella sua riga e massimo nella sua colonna. Tale elemento è detto *punto di sella* in quanto gli altri elementi nella sua riga sono maggiori e gli altri elementi nella sua colonna sono minori, e ciò ricorda la forma di una sella:



Teorema: se v è un punto di sella in un gioco strettamente determinato, allora una strategia ottimale per R è scegliere la riga che contiene v e una strategia ottimale per C è scegliere la colonna che contiene v. Il valore del gioco è proprio v.

Esempio 7

Abbiamo prima visto la matrice:

	uno	due
uno	- 2	4
due	2	3

e, a livello intuitivo, abbiamo visto che conviene a R scegliere la riga 2 e a C la colonna 1. Per individuare il valore maximin poniamo tra parentesi tonde i valori minimi di ciascuna riga:

(-2)	4
(2)	3

Il valore maximin, cioè il massimo dei minimi è 2.

Poniamo ora tra parentesi quadre i massimi di ciascuna colonna:

(-2)	[4]
[(2)]	3

Il valore minimax, cioè il minimo dei massimi, è 2.

L'elemento 2 ha sia le parentesi tonde sia le quadre ed è il segnale che 2 è punto di sella.

Pertanto il valore del gioco è 2. La strategia di R è p = (0, 1), la strategia di C è q = (1, 0). Viene pertanto confermata la validità delle strategie prima individuate in modo informale.

E' chiaro che in questo modo, ad ogni partita, R guadagna 2 euro, però, a differenza dei giochi visti in precedenza, se C cambia strategia perde ancora di più. Infatti se C sceglie la seconda colonna perde 3 euro. Pur essendo una strategia perdente quella di C, è comunque ottimale in quanto gli consente di perdere il meno possibile. Chiaramente il gioco non è equo e infatti il valore del gioco è 2 cioè R vince ad ogni partita 2 euro.

Esempio 8

Non è detto che, dato un gioco, il punto di sella, se esiste, sia unico. Ad esempio, data la matrice:

$$\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

trovando i minimi di riga e i massimi di colonna abbiamo:

$$\begin{pmatrix} [(0)] & [1] & [(0)] \\ (-1) & 0 & (-1) \\ [(0)] & [1] & [(0)] \end{pmatrix}$$

Abbiamo quattro punti di sella cui corrisponde lo stesso valore zero.

Teorema. Se una matrice contiene più di un punto di sella, tali punti assumono tutti lo stesso valore, il valore del gioco.

Dimostrazione. Supponiamo che a e b siano due punti di sella. Se a e b si trovano sulla stessa riga allora entrambi, per definizione, sono i minimi della riga e quindi a=b. Analogamente, se a e b si trovano sulla stessa colonna, entrambi sono i massimi della colonna e pertanto a=b.

Assumiamo allora che a e b si trovano su righe e colonne diverse, e siano c e d due altri elementi della matrice:

$$\begin{pmatrix} * & * & * & * & * & * & * \\ * & * & a & * & * & d & * \\ * & * & * & * & * & * & * \\ * & * & c & * & * & b & * \\ * & * & * & * & * & * & * \end{pmatrix}$$

Poichè a è un punto di sella, $a \leq d$ e $c \leq a$. Ma anche b è punto di sella, quindi $b \leq c$ e $d \leq b$. Pertanto abbiamo:

$$a \leq d \leq b \leq c \leq a$$

da cui ricaviamo: $a = b$

Nell'Esempio 4 avevamo questa matrice di gioco:

	1 euro	2 euro
1 euro	0	5000
2 euro	-5000	0

Trovando i minimi di riga e i massimi di colonna abbiamo:

	1 euro	2 euro
1 euro	[(0)]	[5000]
2 euro	(-5000)	0

C'è un punto di sella per cui ritroviamo i risultati già individuati in precedenza: il valore del gioco è 0 e la strategia ottimale è vendere le bottiglie a un euro. Non utilizzare la strategia ottimale porta ad accrescere la perdita.

Quanto ora esemplificato vale in generale per tutti i giochi strettamente determinati:

Teorema: dato un gioco strettamente determinato, se uno dei giocatori segue la strategia ottimale mentre l'altro non lo fa, quest'ultimo non potrà mai migliorare la sua vincita, la quale diminuisce o al più resta inalterata.

Esempio 9

Consideriamo la seguente matrice, relativa a un gioco non meglio precisato:

$$\begin{pmatrix} 3 & 0 & -4 & -3 \\ 2 & 3 & 1 & 2 \\ -4 & 2 & -1 & 3 \end{pmatrix}$$

Trovando i minimi e i massimi abbiamo:

$$\begin{pmatrix} [3] & 0 & (-4) & -3 \\ 2 & [3] & [(1)] & 2 \\ (-4) & 2 & -1 & [3] \end{pmatrix}$$

L'elemento 1 è un punto di sella per cui il valore del gioco è 1 e le strategie ottimali sono pure e consistono per R nel vettore $p = (0, 1, 0)$ e per C nel vettore $q = (0, 0, 1, 0)$. In altri termini R sceglie sempre la seconda riga e C la terza colonna.

Anche in questo caso il gioco non è equo e R guadagna, ad ogni partita, 1 euro. Se C cambia strategia perde ancora di più; infatti se C sceglie la prima o la terza colonna perde 2 euro, se sceglie la seconda colonna perde 3 euro.

Come accennato, il teorema minimax ci assicura l'esistenza di una coppia di strategie *miste* ottimali tali che il valore maximin coincide con il valore minimax. Tale valore è il valore del gioco.

I giochi strettamente determinati sono molto semplici, i problemi più interessanti danno luogo a matrici di gioco che non hanno punto di sella, cioè i cui valori maximin e minimax non coincidono.

Vediamo comunque un gioco che descrive in modo estremamente semplificato una situazione riguardante il vantaggio competitivo.

Esempio 10

Supponiamo che diventi disponibile un progresso tecnologico (DVD, WiFi, etc.). Se un'azienda adotta la nuova tecnologia ottiene un vantaggio competitivo sulle aziende concorrenti. Se tutte le aziende del settore la adottano, il vantaggio scompare. La seguente matrice potrebbe rappresentare la situazione descritta:

	adotta	non adotta
adotta	0	5
non adotta	-5	0

L'azienda R fa bene ad adottare la nuova tecnologia: infatti, se C non la adotta R si avvantaggia di 5, se C la adotta annulla lo svantaggio -5. Pertanto, qualunque cosa decida di fare C, è meglio per R adottare la nuova tecnologia. Ovviamente la stessa cosa vale per C e abbiamo quindi che (adotta, adotta) è la strategia ottimale.

D'altra parte, trovando i minimi delle righe e i massimi delle colonne abbiamo:

	adotta	non adotta
adotta	[(0)]	[5]
non adotta	(-5)	0

Abbiamo un punto di sella individuato dalle strategie (adotta, adotta) il cui valore è lo stesso che si ottiene con le strategie (non adotta, non adotta). Il fatto è che nè R nè C possono permettersi il lusso di non adottare la tecnologia, se lo fanno i concorrenti si avvantaggiano. In altri termini, e questo corrisponde effettivamente al mondo reale, ogni nuova tecnologia va adottata pena l'uscita dal mercato (vedi LCD-tubi catodici).

Righe e colonne dominanti e dominate

Come abbiamo accennato, quando la matrice di gioco non è 2 x 2 le cose si complicano, però a volte siamo in grado di ridurre una matrice più grande a una equivalente 2 x 2.

Sia A una matrice di gioco. Supponiamo che la riga j abbia tutti i suoi elementi maggiori o uguali dei corrispondenti elementi della riga i. Allora la riga j è detta *dominante* la riga i e la riga i è detta *dominata* da j.

Chiaramente, il giocatore R preferisce giocare la riga j e non la i in quanto la vincita è sempre maggiore o uguale a quanto otterrebbe scegliendo la riga i. Pertanto la riga i, svantaggiosa per R, può essere cancellata dalla matrice senza che il gioco venga influenzato.

Infatti, è vero che la riga i potrebbe contenere una grossa vincita per C e cancellando la riga i C appare danneggiato, ma le cose non stanno così in quanto, per ottenere la vincita C, R dovrebbe scegliere la riga i, ma R non sceglierà mai la riga i e pertanto la sua scomparsa non danneggia C.

Analogamente, supponiamo che A contenga la colonna i tale che tutti i suoi elementi sono maggiori o uguali dei corrispondenti elementi della colonna j. Allora la colonna i, svantaggiosa per C, è detta *dominata* da j e la colonna j si dice che *domina* la colonna i.

Ovviamente, il giocatore C preferisce giocare sempre la colonna j invece della i, per cui la colonna dominata i può essere cancellata dalla matrice.

In altri termini, una riga dominata contiene elementi più piccoli di quelli di un'altra riga, una colonna dominata contiene elementi più grandi di quelli di un'altra colonna.

Esempio 11

Consideriamo la seguente matrice:

$$\begin{pmatrix} -5 & -3 & 1 \\ 2 & (-1) & 2 \\ -2 & [3] & 4 \end{pmatrix}$$

e notiamo che ogni elemento nella prima riga è minore del corrispondente elemento nella seconda riga. Perciò la prima riga è dominata e può essere cancellata. La matrice diventa:

$$\begin{pmatrix} 2 & -1 & 2 \\ -2 & 3 & 4 \end{pmatrix}$$

Osserviamo ora che la terza colonna è dominata in quanto ogni elemento è maggiore del suo corrispondente nella seconda colonna. Cancelliamo quindi la terza colonna:

$$\begin{pmatrix} 2 & -1 \\ -2 & 3 \end{pmatrix}$$

Il gioco non è strettamente determinato non essendoci punto di sella:

$$\begin{pmatrix} [2] & (-1) \\ (-2) & [3] \end{pmatrix}$$

Applicando le formule che ricaveremo tra poco, otteniamo le strategie ottimali e il valore del gioco per la matrice 2 x 2:

$p = (5/8, 3/8)$ $\quad q = (1/2, 1/2)$ $\quad v = 1/2$

Aver cancellato la prima riga e la terza colonna significa che non vanno mai scelte per cui le strategie ottimali per la matrice 3 x 3 originaria sono:

$p = (0, 5/8, 3/8)$ $\quad q = (1/2, 1/2, 0)$ $\quad v = 1/2$

Matrici di gioco 2 x 2

Finalmente siamo ora in grado di risolvere un qualsiasi gioco 2 x 2. Consideriamo la seguente matrice:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Se la matrice è strettamente determinata, strategie e valore sono immediatamente ricavati. Altrimenti vale il seguente:

Teorema minimax per giochi 2 x 2

Supponiamo che la matrice di gioco A non sia strettamente determinata. Sia D la quantità $a + d - b - c$.

Allora $p = (p_1, p_2)$ è una strategia ottimale per R, $q = (q_1, q_2)$ è una strategia ottimale per C e v è il valore del gioco, dove:

$$p_1 = \frac{d - c}{D} \qquad p_2 = \frac{a - b}{D}$$

$$q_1 = \frac{d - b}{D} \qquad q_2 = \frac{a - c}{D}$$

$$v = \frac{ad - bc}{D}$$

Prima di dimostrare il teorema, vediamo alcuni esempi che ci chiariscono alcuni aspetti importanti delle strategie ottimali.

Esempio 12

Ora siamo in grado di calcolare le strategie ottimali per il gioco delle dita (Esempio 1), quello la cui matrice è:

$$\begin{array}{c} \\ \text{uno} \\ \text{due} \end{array} \begin{array}{c} \begin{array}{cc} \text{uno} & \text{due} \end{array} \\ \begin{pmatrix} 2 & -3 \\ -3 & 4 \end{pmatrix} \end{array}$$

Per prima cosa vediamo che il gioco non è strettamente determinato non essendoci un punto di sella:

$$\begin{pmatrix} [2] & (-3) \\ (-3) & [4] \end{pmatrix}$$

Applichiamo quindi le formule e abbiamo:

$$p_1 = 7/12 \qquad p_2 = 5/12$$

$$q_1 = 7/12 \qquad q_2 = 5/12$$

$$v = \frac{2\ 4 - (-3)(-3)}{2+4+3+3} = \frac{-1}{12}$$

Il valore del gioco coincide con il valore atteso della variabile X calcolato in precedenza.

Il gioco non è equo e C guadagna, in media, dopo n partite, n/12 euro.

Verifichiamo ora una importante proprietà delle strategie ottimali. Supponiamo che R, stanco di perdere applicando la strategia ottimale, provi a cambiare strategia per cui, ad esempio,

$p = (1/2, 1/2) \quad q = (7/12, 5/12)$

Calcoliamo E(X) ed abbiamo:

$$E(X) = \begin{matrix} 2 & 1 & 7 \\ & 2 & 12 \end{matrix} - \begin{matrix} 3 & 1 & 5 \\ & 2 & 12 \end{matrix} - \begin{matrix} 3 & 1 & 7 \\ & 2 & 12 \end{matrix} + \begin{matrix} 4 & 1 & 5 \\ & 2 & 12 \end{matrix} = \frac{14}{24} - \frac{15}{24} - \frac{21}{24} + \frac{20}{24} =$$

$$= \frac{34 - 36}{24} = \frac{-2}{24} = \frac{-1}{12}$$

Non è cambiato nulla, R potrebbe insistere, pensare che forse è meglio scegliere più spesso la seconda riga che gli può dare una vincita maggiore della prima, e cambiare di nuovo strategia:

$p = (1/10, 9/10) \qquad q = (7/12, 5/12)$

$$E(X) = \begin{matrix} 2 & 1 & 7 \\ & 10 & 12 \end{matrix} - \begin{matrix} 3 & 1 & 5 \\ & 10 & 12 \end{matrix} - \begin{matrix} 3 & 9 & 7 \\ & 10 & 12 \end{matrix} + \begin{matrix} 4 & 9 & 5 \\ & 10 & 12 \end{matrix} = \frac{14 - 15 - 189 + 180}{120} =$$

$$= \frac{194 - 204}{120} = \frac{-10}{120} = \frac{-1}{12}$$

Il valore atteso non cambia ancora. Il caso limite è se R usa una strategia pura, cioè sceglie sempre la stessa riga, ad esempio la seconda:

$p = (0, 1) \qquad q = (7/12, 5/12)$

$$E(X) = \begin{matrix} 2 & 0 & 7 \\ & & 12 \end{matrix} - \begin{matrix} 3 & 0 & 5 \\ & & 12 \end{matrix} - \begin{matrix} 3 & 1 & 7 \\ & & 12 \end{matrix} + \begin{matrix} 4 & 1 & 5 \\ & & 12 \end{matrix} = \frac{-21 + 20}{12} = \frac{-1}{12}$$

Come si vede, se C resta fedele alla sua strategia ottimale, il valore atteso coincide sempre con il valore del gioco.

Visto che non riesce a non perdere, R, disperato, potrebbe adottare l'altra strategia pura possibile, cioè:

$p = (1, 0)$ $\quad q = (7/12, 5/12)$

$$E(X) = 2\,1\,\frac{7}{12} - 3\,1\,\frac{5}{12} - 3\,0\,\frac{7}{12} + 4\,0\,\frac{5}{12} = \frac{14 - 15}{12} = \frac{-1}{12}$$

Non c'è niente da fare per R. Solo se *entrambi* i giocatori si allontanano dalla strategia ottimale, cambia il valore atteso. Abbiamo infatti visto che le strategie:

$p = (1/2, 1/2)$ $\quad q = (1/2, 1/2)$

portano a $E(X) = 0$.

R può anche guadagnare se, ad esempio, le strategie sono:

$p = (0, 1)$ $\quad q = (1/2, 1/2)$

$$E(X) = -3\,\frac{1}{2} + 4\,\frac{1}{2} = \frac{1}{2}$$

Quanto ora detto implica che, nel caso che *solo* C lasci la sua strategia ottimale, C continua a vincere ! Infatti, ad esempio, se abbiamo:

$p = (7/12, 5/12)$ $\quad q = (1, 0)$

$$E(X) = 2\,\frac{7}{12}\,1 - 3\,\frac{7}{12}\,0 - 3\,\frac{5}{12}\,1 + 4\,\frac{5}{12}\,0 = \frac{14 - 15}{12} = \frac{-1}{12}$$

In generale, vale il seguente:

Teorema 1: una volta individuata una coppia di strategie ottimali miste, se un giocatore segue la sua strategia ottimale mista, la vincita rimane uguale al valore v del gioco qualsiasi cosa faccia l'altro giocatore purchè egli non scelga righe o colonne dominate. Nel caso lo faccia la sua vincita diminuisce.

Segue dal teorema che i giocatori non hanno motivo di tenere nascosta la strategia ottimale che utilizzano, possono tranquillamente comunicarla all'avversario senza che quest'ultimo possa modificare a suo favore il risultato del gioco.

Il limite posto da questo teorema (no righe o colonne dominate) si chiarisce rifacendosi all'Esempio 9.

Siamo partiti dalla matrice:

$$\begin{pmatrix} -5 & -3 & 1 \\ 2 & -1 & 2 \\ -2 & 3 & 4 \end{pmatrix}$$

Essendo la prima riga dominata siamo passati alla matrice:

$$\begin{pmatrix} 2 & -1 & 2 \\ -2 & 3 & 4 \end{pmatrix}$$

e poichè la terza colonna è dominata, siamo giunti a:

$$\begin{pmatrix} 2 & -1 \\ -2 & 3 \end{pmatrix}$$

Le formule precedenti ci danno le strategie ottimali per il gioco 2 x 2:

$$p = (5/8, 3/8) \qquad q = (1/2, 1/2) \qquad v = 1/2$$

La cancellazione della prima riga e della terza colonna significa che non vanno mai scelte per cui abbiamo ottenuto subito le strategie ottimali per la matrice 3 x 3 originaria:

$$p = (0, 5/8, 3/8) \qquad q = (1/2, 1/2, 0) \qquad v = 1/2$$

Ora, in base al teorema enunciato se *uno solo* dei giocatori cambia la sua strategia, il valore atteso non cambia, questo a patto che il giocatore non scelga righe o colonne dominate.

Infatti, se C muta la sua strategia e sceglie sempre la prima colonna:

$$p = (0, 5/8, 3/8) \qquad q = (1, 0, 0)$$

$$E(X) = 2\,\frac{5}{8}\,1 - 2\,\frac{3}{8}\,1 = \frac{10-6}{8} = \frac{4}{8} = \frac{1}{2}$$

Come si vede $E(x) = v$. Se però C sceglie la terza colonna (dominata), il guadagno atteso di R aumenta, e la cosa è ragionevole in quanto la terza colonna è svantaggiosa per C:

$$p = (0, 5/8, 3/8) \qquad q = (0, 0, 1)$$

$$E(X) = 2\,\frac{5}{8}\,1 + 4\,\frac{3}{8}\,1 = \frac{10+12}{8} = \frac{22}{8} = \frac{11}{4}$$

Prima di passare alla dimostrazione del teorema minimax per matrici 2 x 2, enunciamo un altro importante:

Teorema 2. Una matrice 2 x 2 contiene un punto di sella se e solo se vi è almeno una riga o colonna dominata.

Ad esempio, nella matrice:

$$\begin{pmatrix} [5] & [(3)] \\ (1) & 2 \end{pmatrix}$$

il punto di sella è 3 e la seconda riga è dominata. Nella matrice:

$$\begin{pmatrix} 2 & [(-3)] \\ [3] & (-5) \end{pmatrix}$$

il punto di sella è -3 e la prima colonna è dominata. Infine, nella matrice:

$$\begin{pmatrix} (0) & 2 \\ [(1)] & [3] \end{pmatrix}$$

la prima riga e la seconda colonna sono dominate.

Viceversa, data la matrice:

$$\begin{pmatrix} 5 & 3 \\ 1 & 2 \end{pmatrix}$$

la seconda riga è dominata, può essere omessa e abbiamo:

$$\begin{pmatrix} 5 & 3 \end{pmatrix}$$

Ora la prima colonna è dominata e abbiamo il punto di sella 3.

Teorema minimax per giochi 2 x 2.

Dimostrazione

Supponiamo che il gioco 2 x 2 non abbia punto di sella. Ora il fatto che il gioco non abbia punto di sella implica che non ci sono righe o colonne dominate. Infatti nel momento in cui c'è una riga o colonna dominata compare un punto di sella.

Ci proponiamo di individuare la strategia ottimale $p = (p_1, p_2)$ di R. Il Teorema 1 ci assicura che C può variare la sua strategia ottimale $q = (q_1, q_2)$ senza cambiare il valore v del gioco. In particolare, allora, C può adottare le strategie pure $q = (1, 0)$ oppure $q = (0, 1)$.

Se C adotta la strategia pura $q = (1, 0)$ abbiamo:

$$E(X) = a\,p_1\,1 + b\,p_1\,0 + c\,p_2\,1 + d\,p_2\,0 = a\,p_1 + c\,p_2 = v$$

Se C adotta la strategia pura $q = (0, 1)$ si ha:

$$E(X) = a\,p_1\,0 + b\,p_1\,1 + c\,p_2\,0 + d\,p_2\,1 = b\,p_1 + d\,p_2 = v$$

Abbiamo due equazioni e, ricordando che $p_1 + p_2 = 1$, $p_2 = 1 - p_1$ e eguagliando le parti sinistre ricaviamo:

$$a\,p_1 + c\,(1 - p_1) = b\,p_1 + d\,(1 - p_1)$$

$$a\,p_1 + c - c\,p_1) = b\,p_1 + d - d\,p_1$$

$$a\,p_1 - c\,p_1 - b\,p_1 + d\,p_1 = d - c$$

$$p_1\,(a + d - b - c) = d - c$$

da cui:

$$p_1 = \frac{d - c}{a + d - b - c}$$

$$p_2 = 1 - p_1 = 1 - \frac{d - c}{a + d - b - c} = \frac{a + d - b - c - d + c}{a + d - b - c} = \frac{a - b}{a + d - b - c}$$

Sostituendo i valori di p_1 e p_2 in una delle precedenti equazioni abbiamo:

$$a\,p_1 + c\,p_2 = v = a\,\frac{d - c}{a + d - b - c} + c\,\frac{a - b}{a + d - b - c} =$$

$$= \frac{ad - ac + ac - bc}{a + d - b - c} = \frac{ad - cb}{a + d - b - c}$$

per cui concludiamo che:

$$v = \frac{ad - cb}{a + d - b - c}$$

E' facile ora individuare la strategia ottimale $q = (q_1, q_2)$ di C. Il Teorema 2 ci assicura, non essendoci per ipotesi punti di sella, che la strategia di R non coinvolge righe dominate per cui R può variare la sua strategia $p = (p_1, p_2)$ senza cambiare il valore v del gioco. In particolare, allora, R può adottare la strategia pura $p = (1, 0)$ e otteniamo:

$$E(X) = a\,1\,q_1 + b\,1\,q_2 + c\,0\,q_1 + d\,0\,q_2 = a\,q_1 + b\,(1 - q_1) =$$

$$a\,q_1 + b - b\,q_1 = q_1\,(a - b) + b = v$$

da cui:

$$q_1 = \frac{v - b}{a - b} = \frac{\dfrac{ad - bc}{a + d - b - c} - b}{a - b} = \frac{\dfrac{ad - bc - ab - bd + b^2 + bc}{a + d - b - c}}{a - b} =$$

$$= \frac{(a - b)\,d + (a - b)\,(-b)}{a + d - b - c}\,\frac{1}{a - b} = \frac{(a - b)\,(d - b)}{a + d - b - c}\,\frac{1}{a - b} = \frac{d - b}{a + d - b - c}$$

$$q_2 = 1 - q_1 = 1 - \frac{d - b}{a + d - b - c} = \frac{a + d - b - c - d + b}{a + d - b - c} = \frac{a - c}{a + d - b - c}$$

Esempio 13

La matrice di gioco dell'Esempio 3 è:

	testa	croce
testa	1	-1
croce	-1	1

e non ha punto di sella:

$$\begin{pmatrix} [1] & (-1) \\ (-1) & [1] \end{pmatrix}$$

Applicando le formule ricaviamo le strategie ottimali e il valore:

$p = (1/2, 1/2)$ $q = (1/2, 1/2)$ $v = 0$

in accordo con i risultati ottenuti intuitivamente.

E' interessante osservare che, come abbiamo detto all'inizio, la teoria dei giochi cerca di individuare delle strategie razionali di comportamento al fine di massimizzare la vincita dei giocatori: ebbene, il teorema fondamentale ci dice in sostanza che il comportamento *razionale* è un comportamento *probabilistico,* in cui la prossima mossa viene decisa, ad esempio, lanciando un dado. In altri termini, *è razionale affidarsi al caso.*

Quando c'è un punto di sella è chiaro che le strategie minimax sono le più razionali per i due giocatori. Quando il punto di sella manca dove va a finire il comportamento razionale ?

In realtà il punto di sella scompare ma solo per riapparire a un livello superiore: il teorema di von Neumann ci dice che i valori maximin e minimax coincidono di nuovo utilizzando le strategie ottimali. In sostanza, il punto di sella esiste di nuovo non più come valore *assoluto* (vincita ad ogni partita) ma come valore *medio.*

Il teorema minimax stabilisce come procedere razionalmente in situazioni prima considerate non gestibili razionalmente.

Giochi non a somma zero

I giochi con due giocatori e a somma zero vengono completamente descritti dalle matrici di gioco e risolti applicando il teorema di von Neumann. Tale teorema è però applicabile solo a giochi a somma zero e questa è una limitazione molto forte se vogliamo applicare la teoria dei giochi alle interazioni umane del mondo reale.

Ad esempio, la corsa agli armamenti non è un gioco a somma zero: entrambi i giocatori possono perdere. Quando stipulano un contratto i due contraenti sperano di vincere entrambi e spesso così accade. L'inquinamento ambientale è un gioco in cui i vantaggi che ricava l'inquinatore sono molto inferiori ai danni provocati a tutti gli altri e lo stesso accade per la distruzione dei banchi di pesce.

In sostanza i giochi a somma zero sono proprio quelli che l'uomo ha inventato per divertirsi o per analizzarli. Ad esempio il poker è a somma zero in quanto ad ogni mano ciò che vince un giocatore viene perso dagli altri.

Invece i "giochi" causati dalle interazioni umane sono essenzialmente non a somma zero. Von Neumann sperava di estendere la sua teoria a giochi con n partecipanti e non a somma zero, ma i problemi diventano molto più difficili da analizzare.

Il grande merito di von Neumann è stato di aver dato una risposta scientifica alla domanda "Qual è il comportamento razionale in un gioco a somma zero ?". La restrizione ai giochi a somma zero limita fortemente l'applicabilità della teoria fuori del campo dei giochi ma ha comunque dato una prima risposta parziale al problema del comportamento umano in situazioni complesse.

Un progresso decisivo è stato compiuto da John Nash con l'introduzione del concetto di *equilibrio*. Prima di introdurre tale concetto fondamentale è utile fare qualche esempio.

Equilibri in strategie dominanti ed equilibri di Nash

Esempio 14

L'Esempio 1 riguarda la morra a due dita, vediamo di nuovo lo stesso gioco però con la seguente matrice:

	uno	due
uno	7, 7	8, 0
due	0, 8	5, 5

Ogni elemento della matrice ora ha due componenti, il primo indica la vincita di R., il secondo la vincita di C. Ad esempio, (0, 8) indica che, se R mostra due dita e C uno solo, R vince 0 e C vince 8 euro.

Al solito, questa matrice potrebbe anche nascere dal gioco testa-croce o da altri giochi ancora. Per analizzare il gioco è sufficiente esaminare la matrice, non ci interessa da quale gioco abbia origine, cosa rappresentino i valori (euro, mele, televisori, etc.), chi tiri fuori la vincita (potrebbe essere uno zio generoso).

Chiaramente, il gioco è a somma diversa da zero per cui il teorema minimax non è applicabile. Vediamo però che accade qualcosa di analogo a quanto visto prima per le righe e le colonne dominanti: a R conviene scegliere sempre la prima riga in quanto vince 7 o 8. Scegliendo la seconda riga invece vince 0 o 5: è vero che la vincita effettiva dipende dal quale colonna sceglie C, però quale che sia la scelta di C, la vincita per R è sempre maggiore scegliendo la riga 1.

A sua volta, C guarda la matrice e si accorge che scegliendo la colonna 1 ottiene sempre una vincita maggiore che scegliendo la colonna 2. Si dice che la riga 1 (prime componenti) *domina* la riga 2 (prime componenti) e pertanto la scelta della riga 1 è per R la *strategia dominante.* Analogamente la colonna 1 (seconde componenti) domina la colonna 2 (seconde componenti) e quindi scegliere la colonna 1 rappresenta per C la strategia dominante.

La coppia di strategie (riga 1, colonna 1) individua un *equilibrio in strategie dominanti*. Spesso nel seguito una tale coppia di strategie verrà per brevità denotata come "la strategia dominante".

Supponiamo ora che la matrice del gioco cambi come segue:

	uno	due
uno	7, 7	8, 9
due	0, 8	5, 5

Le prime componenti non sono cambiate per cui la riga 1 continua a dominare la riga 2 e R ha la strategia dominante "scegliere la riga 1", però l'elemento (8,0) è diventato (8,9) per cui la colonna 2 non è più dominata dalla colonna 1 (seconde componenti).

C ragiona così: io non ho una strategia dominante, però R ce l'ha (riga 1) per cui a me conviene scegliere la colonna 2 in modo da vincere 9, se scegliessi colonna 1 vincerei 7. Pertanto la strategia migliore è (riga 1, colonna 2).

Infine, consideriamo la seguente matrice:

	uno	due
uno	7, 7	8, 0
due	0, 8	9, 9

In questo caso non ci sono righe o colonne dominanti per cui occorre far intervenire il concetto di equilibrio di Nash:

Definizione. Un *equilibrio di Nash* è un insieme di strategie pure o miste, una per ogni giocatore, tale che a nessun giocatore conviene cambiare la propria strategia, nell'ipotesi che gli altri giocatori non cambino la loro. In altri termini, i giocatori sono in equilibrio se cambiando uno di essi la sua strategia, mentre gli altri non la cambiano, la sua vincita diminuisce.

Osserviamo che un equilibrio in strategie dominanti è un equilibrio di Nash, ma si può avere un equilibrio di Nash anche in assenza di strategie dominanti.

Nash ha dimostrato un teorema che estende il teorema di von Neumann al caso di giochi con n persone e non a somma zero:

Teorema di Nash: in un gioco (a somma zero o non zero) con n partecipanti, esiste almeno un equilibrio di Nash, cioè una strategia collettiva ottimale in cui nessun giocatore può, modificando la sua strategia, migliorare la sua vincita nell'ipotesi che gli altri giocatori non modifichino la propria strategia.

Tre sono i risultati importanti contenuti nel teorema di Nash:

1) estende il teorema minimax a un qualsiasi numero di giocatori

2) generalizza il teorema minimax in quanto assicura l'esistenza di un punto di equilibrio sia per giochi a somma zero sia per quelli che non lo sono.

3) dimostra che vi può essere più di una soluzione

Torniamo alla nostra matrice:

	uno	due
uno	7, 7	8, 0
due	0, 8	9, 9

Vediamo se la strategia (riga 1, colonna 1) è un equilibrio di Nash: supponendo che C scelga la colonna 1, a R conviene scegliere la riga 1 (se sceglie la riga 2 la sua vincita scende da 7 a 0. Nell'ipotesi che R scelga la riga 1, se C sceglie la colonna 2 invece che la colonna 1 il suo guadagno scende da 7 a 0. Pertanto la strategia (riga 1, colonna 1), in base alla definizione precedente, è un equilibrio di Nash.

Strategia (riga 1, colonna 2): nell'ipotesi che C scelga la colonna 2, a R conviene la riga 2 invece che la riga 1 (guadagno 9 invece che 8). Analogamente, supponendo che R scelga la riga 1, per C è meglio scegliere la colonna 1 (vincita 7 invece che 0). Ne consegue che la strategia (riga 1, colonna 2), in base alla definizione, non è un equilibrio.

Strategia (riga 2, colonna 1): il ragionamento è simmetrico a quello ora fatto. Se C sceglie la colonna 1 a R non conviene scegliere la riga 2, è meglio la riga 1 (vincita 7 invece che 0). Anche a C non conviene la colonna 1: nell'ipotesi che R scelga la riga 2, è preferibile la colonna 2 (vincita 9 anzichè 8). Concludiamo che tale strategia non è un equilibrio.

Strategia (riga 2, colonna 2): se C sceglie la colonna 2, a R non conviene scegliere la riga 1, è meglio la riga 2 (vincita 9 invece che 8). Anche C non ha interesse a cambiare colonna: se passa dalla 2 alla 1 la sua vincita scende da 9 a 8. Anche la strategia (riga 2, colonna 2) è un equilibrio di Nash.

Abbiamo pertanto due equilibri distinti in questo gioco individuati dalle strategie pure (riga 1, colonna 1) e (riga 2, colonna 2). Vedremo più avanti che esiste un terzo equilibrio individuato da una strategia mista, in cui righe e colonne vengono scelte con probabilità (1/8, 7/8).

E' importante sottolineare che i giochi a somma zero hanno *un'unica soluzione* che è certamente la *più razionale* in quanto massimizza le vincite dei due giocatori. Passando ai giochi non a somma zero le cose si complicano.

Infatti:

1) Possono esserci più equilibri di Nash per lo stesso gioco e la cosa non ci piace. Dato un gioco vorremmo un'unica soluzione non un insieme di soluzioni. Inoltre la definizione di equilibrio non ci dice nulla su come scegliere il migliore equilibrio, cioè quello che massimizza le vincite dei due giocatori.

2) Anche se, dato un gioco, l'equilibrio di Nash è unico è sbagliato pensare che tale equilibrio rappresenti *sempre* la soluzione migliore.

3) Anche se un gioco ammette un equilibrio in strategie dominanti, non è detto che tale equilibrio sia *sempre* la soluzione migliore.

Ad esempio, abbiamo appena individuato due equilibri che sono *assolutamente equivalenti* dal punto di vista della definizione la quale richiede solo nessuno dei giocatori abbia un incentivo a cambiare strategia. Noi guardiamo i due equilibri e ci accorgiamo che il secondo equilibrio è senz'altro migliore.

In modo più formale, un equilibrio si dice *efficiente* se non esiste un altro equilibrio preferibile da tutti i giocatori, che cioè li avvantaggi di più.

Nel nostro esempio il secondo equilibrio è efficiente in quanto consente a entrambi di vincere 9 anzichè 7, però in giochi più impegnativi il problema di individuare la soluzione migliore è molto complesso.

Prima di vedere come si può individuare la strategia mista, facciamo un altro esempio.

Esempio 15

Supponiamo di avere la seguente matrice relativa a un gioco non meglio precisato, in cui R ha tre mosse a disposizione e C solo due:

	1	2
1	9, 18	7, 5
2	6, 5	10, 6
3	15, 17	4, 19

Si verifica facilmente che non vi sono strategie dominanti per cui dobbiamo trovare gli equilibri di Nash.

La strategia (riga 1, colonna 1) non è un equilibrio in quanto, supponendo che C scelga la colonna 1, a R conviene scegliere la riga 3 (vincita 15 invece che 9). A C invece, nell'ipotesi che R scelga la riga 1, conviene la colonna 1 invece che la 2 (18 meglio di 5). Pertanto C non ha interesse a cambiare colonna ma a R conviene e questo basta per dire che la strategia (riga 1, colonna 1) non è un equilibrio.

Si ragiona in modo analogo per le altre cinque strategie e ci accorgiamo che solo la strategia (riga 2, colonna 2) è un equilibrio: infatti, supponendo che C scelga la colonna 2, a R conviene la riga 2 (vincita 10 invece che 7 o 4). Anche a C, nell'ipotesi che R scelga la riga 2, sta bene la colonna 2 (vincita 6 invece che 5).

In questo gioco pertanto abbiamo un solo equilibrio di Nash individuato dalla strategia (riga 2, colonna 2). L'equilibrio è unico ma non massimizza le vincite dei giocatori: infatti la strategia (riga 3, colonna 1) consente a entrambi di vincere di più.

La strategia (riga 3, colonna 1) non è un equilibrio in quanto se C sceglie colonna 1 a R sta bene la riga 3, ma se R sceglie la riga 3, C preferisce la seconda colonna (vincita 19 invece che 17). In questo gioco perciò l'unico equilibrio di Nash non è il più vantaggioso.

Trovare gli equilibri puri di Nash

Ci chiediamo ora quale rapporto esiste fra gli equilibri di Nash e i concetti visti per i giochi a somma zero.

I giochi a somma zero sono un caso particolare dei giochi a somma diversa da zero, possono essere riscritti nella forma non a somma zero e quindi analizzati.

Una volta riscritto un gioco strettamente determinato, una riga o colonna dominante diventa una strategia dominante, il punto di sella diventa un equilibrio di Nash.

Anche il metodo parentesi tonde-quadre per individuare il punto di sella si modifica per trovare gli equilibri dati da strategie pure e ora lo esemplifichiamo.

Data la matrice dell'Esempio 12:

	uno	due
uno	7, 7	8, 0
due	0, 8	5, 5

Per ogni colonna mettiamo tra parentesi tonde le massime prime componenti:

	uno	due
uno	(7), 7	(8), 0
due	0, 8	5, 5

Per ogni riga mettiamo tra parentesi quadre le massime seconde componenti:

	uno	due
uno	(7), [7]	(8), 0
due	0, [8]	5, 5

Gli equilibri di Nash individuati da strategie *pure* sono tutti e soli quelli individuati dagli elementi che hanno sia le parentesi tonde sia le quadre. Nel nostro caso la strategia (riga 1, colonna 1) è un equilibrio.

Il metodo ora descritto per individuare gli equilibri è corretto in quanto il fatto che l'elemento 7, 7 abbia la prima componente con parentesi tonde significa che, fissata la colonna 1, il primo 7 è il massimo in questa colonna.

Il fatto che la seconda componente abbia le parentesi quadre significa che, fissata la riga 1, il secondo 7 è il massimo in questa riga. Ma queste sono proprio le condizioni presenti nella definizione di equilibrio di Nash.

Nel caso della matrice:

	uno	due
uno	7, 7	8, 9
due	0, 8	5, 5

mettendo le parentesi abbiamo:

	uno	due
uno	(7), 7	(8), [9]
due	0, [8]	5, 5

e pertanto l'unico equilibrio di Nash è dato dalla strategia (riga 1, colonna 2).

Ancora, consideriamo la matrice:

	uno	due
uno	7, 7	8, 0
due	0, 8	9, 9

mettiamo le parentesi:

	uno	due
uno	(7), [7]	8, 0
due	0, 8	(9), [9]

e abbiamo i due equilibri (riga 1, colonna 1) e (riga 2, colonna 2).

Nel caso della matrice:

	1	2
1	9, 18	7, 5
2	6, 5	10, 6
3	15, 17	4, 19

mettendo le parentesi otteniamo:

	1	2
1	9, [18]	7, 5
2	6, 5	(10), [6]
3	(15), 17	4, [19]

e pertanto l'unico equilibrio è (riga 2, colonna 2).

Consideriamo il gioco testa-croce dell'Esempio 3:

$$\begin{array}{c} \\ \text{testa} \\ \text{croce} \end{array} \begin{array}{c} \begin{array}{cc} \text{testa} & \text{croce} \end{array} \\ \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \end{array}$$

Essendo un gioco a somma zero abbiamo usato la notazione semplificata, ma ovviamente possiamo anche usare la notazione valida per i giochi non a somma zero:

$$\begin{array}{c} \\ \text{testa} \\ \text{croce} \end{array} \begin{array}{c} \begin{array}{cc} \text{testa} & \text{croce} \end{array} \\ \begin{pmatrix} 1, -1 & -1, 1 \\ -1, 1 & 1, -1 \end{pmatrix} \end{array}$$

mettendo le parentesi abbiamo:

$$\begin{array}{c} \\ \text{testa} \\ \text{croce} \end{array} \begin{array}{c} \begin{array}{cc} \text{testa} & \text{croce} \end{array} \\ \begin{pmatrix} (1), -1 & -1, [1] \\ -1, [1] & (1), -1 \end{pmatrix} \end{array}$$

e concludiamo che non vi sono equilibri individuati da strategie pure.

Infine, consideriamo il gioco dell'Esempio 8:

$$\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$$

Lo riscriviamo nella forma:

$$\begin{pmatrix} 0,0 & 1,-1 & 0,0 \\ -1,1 & 0,0 & -1,1 \\ 0,0 & 1,-1 & 0,0 \end{pmatrix}$$

troviamo i massimi di colonna e riga e abbiamo:

$$\begin{pmatrix} (0),[0] & (1),-1 & (0),[0] \\ -1,[1] & 0,0 & -1,[1] \\ (0),[0] & (1),-1 & (0),[0] \end{pmatrix}$$

e troviamo quattro equilibri di Nash che corrispondono ai quattro punti di sella dell'Esempio 8.

Equilibri con strategie miste

Vediamo ora come possiamo individuare gli equilibri individuati da strategie miste. Se vi sono strategie dominanti vi sono solo equilibri dati da strategie pure, se non ve ne sono possono esserci sia equilibri puri sia equilibri misti. Consideriamo la matrice:

$$\begin{array}{c} \\ \text{uno} \\ \text{due} \end{array} \begin{array}{c} \begin{array}{cc} \text{uno} & \text{due} \end{array} \\ \begin{pmatrix} 7,7 & 8,0 \\ 0,8 & 9,9 \end{pmatrix} \end{array}$$

Non vi sono strategie dominanti e abbiamo individuato gli equilibri puri (uno, uno) e (due, due).

Come fare per individuare le strategie miste che portano a un terzo equilibrio ? Il ragionamento da fare è il seguente.

Abbiamo prima visto, parlando dei giochi a somma zero, che il valore atteso della variabile aleatoria X che assume come valori gli elementi x_{ij} di una matrice di gioco è dato, fissata la riga i, da:

$$E(\text{riga i}) = \sum_{j=1}^{m} x_{ij}\ p_i\ q_j$$

Ricordiamo che la *probabilità condizionale* di un evento A dato l'evento H è definita come segue:

$$P(A/H) = \frac{P(A \wedge H)}{P(H)}$$

Pertanto, nel caso in cui R abbia scelto la riga 1:

$$p_1 = P(\text{riga 1} / \text{riga 1}) = \frac{P(\text{riga 1} \wedge \text{riga 1})}{P(\text{riga 1})} = \frac{P(\text{riga 1})}{P(\text{riga 1})} = 1$$

e la vincita media di R dipenderà solo dalla strategia adottata da C.

Supponiamo che C adotti la strategia mista $(q_1, q_2) = (q, 1-q)$, cioè C scelga la colonna 1 con probabilità q e la colonna 2 con probabilità $(1-q)$, allora la vincita media di R sarà:

$$E(\text{riga } 1) = \sum_{j=1}^{2} x_{1j}\ p_1\ q_j = 7 \times 1 \times q + 8 \times 1 \times (1-q) = 7q + 8(1-q)$$

Analogamente, una volta che R ha scelto la riga 2, $p_2 = 1$ e la vincita media di R sarà:

$$E(\text{riga } 2) = \sum_{j=1}^{2} x_{2j}\ p_2\ q_j = 0 \times 1 \times q + 9 \times 1 \times (1-q) = 9(1-q)$$

Ora, quello che vuole C è che per R sia indifferente scegliere riga 1 o riga 2, cioè scegliere una riga invece che l'altra non deve dare a R alcun vantaggio. In altri termini, la vincita media di R scegliendo la riga 1 deve essere uguale a quella ottenuta scegliendo la riga 2, cioè deve essere:

$7q + 8(1-q) = 9(1-q)$

$7q + 8 - 8q = 9 - 9q$

$7q = 1 - q$

$8q = 1$

$q = 1/8$

$(1 - q) = 7/8$

In conclusione, la strategia di C che rende a R indifferente ai fini della vincita media la scelta della riga 1 o della riga 2 è individuata dal vettore $(q_1, q_2) = (1/8, 7/8)$.

Simmetricamente, anche R vuole che C non abbia motivo per preferire la colonna 1 alla colonna 2 o viceversa.

Ora, nel momento in cui C sceglie la colonna 1, $q_1 = 1$ e il valore atteso della sua vincita dipende solo dalla strategia di R, descritta dal vettore $(p_1, p_2) = (p, 1-p)$. La vincita media di C relativa alla scelta della colonna 1 è:

$$E(\text{colonna } 1) = \sum_{i=1}^{2} x_{i1}\, p_i\, q_1 = 7 \times p \times 1 + 8 \times (1-p) \times 1 = 7p + 8(1-p)$$

Analogamente, una volta che C ha scelto la colonna 2, $q_2 = 1$ e la vincita media di C sarà:

$$E(\text{colonna } 2) = \sum_{i=1}^{2} x_{i2}\, p_i\, q_2 = 0 \times p \times 1 + 9 \times (1-p) \times 1 = 9(1-p)$$

L'obiettivo di R è che per C sia indifferente scegliere colonna 1 o colonna 2, cioè scegliere una colonna invece che l'altra non deve avvantaggiare R.

In altri termini, la vincita di C scegliendo la colonna 1 deve essere uguale a quella ottenuta scegliendo la colonna 2, cioè deve essere:

$7p + 8(1-p) = 9(1-p)$

$7p + 8 - 8p = 9 - 9p$

$p(7 - 8 + 9) = 1$

$p = 1/8$

In conclusione, la strategia di R che rende a C indifferente, ai fini della vincita media, la scelta della colonna 1 o della colonna 2 è individuata dal vettore $(p_1, p_2) = (1/8, 7/8)$. La strategia è identica a quella che adotta C essendo il gioco simmetrico.

Abbiamo perciò individuato un terzo equilibrio di Nash descritto dalla coppia di strategie miste:

$(p_1, p_2) = (1/8, 7/8)$

$(q_1, q_2) = (1/8, 7/8)$

Ci si può chiedere se esistono altri equilibri di Nash. Altri equilibri puri certamente non ne esistono in quanto vi sono solo quattro strategie pure possibili e solo due rappresentano equilibri. La procedura che abbiamo visto ci ha portato al terzo equilibrio. Può esistere un altro equilibrio in cui un giocatore adotta una strategia mista e l'altro una strategia pura ?

La risposta è negativa in quanto supponiamo che R stia adottando una strategia mista, C ovviamente non vuole che scegliere una riga piuttosto che un'altra avvantaggi R per cui non può adottare una strategia pura e deve anche lui adottare una strategia mista.

Come si vede, passando ai giochi non a somma zero, le cose si sono notevolmente complicate. Nei giochi a somma zero esiste il teorema di von Neumann che ci assicura l'esistenza di una coppia di strategie ottimali tali che il valore maximin coincide con il valore minimax. Tale valore è il valore del gioco.

Ora invece abbiamo, in generale, più equilibri cui corrispondono strategie e valori attesi completamente diversi.

La vincita media di R e C quando si adotta la strategia relativa all'equilibrio puro (riga 1, colonna 1) è 7, invece il valore atteso della vincita per l'altro equilibrio (riga 2, colonna 2) è 9, per cui tale equilibrio è vantaggioso sia per R che per C.

Qual'è la vincita media per il terzo equilibrio individuato, sia per R che per C, dal vettore di probabilità (1/8, 7/8) ?

La strategia di R è tale che per lui è indifferente scegliere la riga 1 o la riga 2 per cui la vincita media è data da:

E(riga 1) = 7 x 1/8 + 8 x 7/8 = 63/8 = E(riga 2) = 0 x 1/8 + 9 x 7/8 = 63/8

La vincita media di C è invece data da:

E(colonna 1) = 7 x 1/8 + 8 x 7/8 = 63/8 = E(riga 2) = 0 x 1/8 + 9 x 7/8 = 63/8

La matrice è simmetrica per cui i valori attesi per R e C coincidono. Se il gioco non è simmetrico le vincite medie possono essere diverse.

Si vede allora che le vincite medie per i tre equilibri individuati sono scalate come segue:

$7 < 63/8 < 9$

l'equilibrio misto porta a una vincita maggiore del primo equilibrio puro ma inferiore al secondo equilibrio puro che si conferma il migliore per entrambi i giocatori.

Esempio 16

Abbiamo visto che il gioco testa-croce dell'Esempio 3:

		testa	croce
	testa	1, -1	-1, 1
	croce	-1, 1	1, -1

non presenta strategie dominanti e non ha nemmeno equilibri di Nash. Ad esempio, la strategia (croce, testa) non è un equilibrio: a C, nell'ipotesi che R scelga la riga 2, sta bene la colonna 1, però a R conviene scegliere la riga 1 invece che la riga 2 (vincita 1 invece che -1). Questo basta per dire che la strategia (croce, testa) non è un equilibrio.

La mancanza di equilibri non è in contraddizione con il teorema di Nash che assicura l'esistenza di almeno un equilibrio per ogni gioco. Infatti, il teorema garantisce l'esistenza di equilibri che possono essere anche dati da strategie *miste*, in cui intervengono le probabilità.

Noi abbiamo solo visto che non esistono equilibri dati da strategie *pure*. Nell'Esempio 3, ragionando sul gioco, abbiamo concluso che la soluzione del gioco è data dalla strategia che sceglie righe e colonne con probabilità 1/2. Applicando il metodo appena esemplificato per trovare le strategie miste abbiamo la conferma che l'equilibrio è individuato per entrambi i giocatori dal vettore di probabilità (1/2, 1/2).

Esempio 17

Ci sono due stazioni radio R e C che devono scegliere quale format scegliere (musica, notizie, tempo, etc.). Vi sono tre possibili format F1, F2, F3 i quali hanno un indice di ascolto pari a 50%, 30%, 20%, rispettivamente.

Se R e C scelgono lo stesso format si dividono a metà la audience per quel format, se scelgono format diversi ognuno si prende tutta la audience per quel format. La matrice delle vincite è la seguente:

	F1	F2	F3
F1	25,25	50,30	50,20
F2	30,50	15,15	30,20
F3	20,50	20,30	10,10

Non vi sono strategie dominanti per cui troviamo gli equilibri di Nash:

	F1	F2	F3
F1	25,25	(50),[30]	(50),20
F2	(30),[50]	15,15	30,20
F3	20,[50]	20,30	10,10

Vi sono due equilibri in cui una stazione sceglie F2 e l'altra F1 o viceversa. Nei giochi a somma zero esiste una sola soluzione, invece, nei giochi non a somma zero, in generale, ne abbiamo più di una.

Entrambi gli equilibri non sono efficienti in quanto l'equilibrio (F1, F2) avvantaggia F1 mentre l'altro è favorevole a F2.

Le soluzioni sembrano buone in quanto hanno entrambe un vincita totale di 80 e tale valore rappresenta la vincita complessiva massima nella matrice ma non sono certo equivalenti dal punto di vista di R e C.

Vi sono anche altri problemi. Infatti entrambe le stazioni, in assenza di comunicazione, potrebbero scegliere il format F1 più redditizio e in tal caso scenderebbero entrambe a 25.

Vi è anche un pericolo maggiore: se R, ad esempio, è convinta che C sceglierà F1 le conviene scegliere F2 vincendo in tal modo 30 invece che 25. Se sia R che C hanno questa convinzione, entrambe scelgono F2 e vincono 15 ciascuna. Nasce pertanto un problema di *cooperazione* che va risolto per evitare una vincita inferiore a quella prevista dall'equilibrio.

Esempio 18

Supponiamo che R sia una casa produttrice di consolle per videogiochi e deve decidere se per la sua nuova consolle adottare i DVD oppure i CD. Contemporaneamente, la società di software C deve decidere quale supporto (DVD o CD) usare per il suo nuovo videogioco. R e C guadagneranno raggiungendo un accordo, perderanno altrimenti. La matrice del gioco è la seguente:

	dvd	cd
dvd	9,9	-4,-1
cd	-3,-1	5,5

Non ci sono strategie dominanti per cui troviamo gli equilibri di Nash:

	dvd	cd
dvd	(9),[9]	-4,-1
cd	-3,-1	(5),[5]

Ci sono due equilibri: (dvd, dvd) e (cd, cd). Quale dei due scegliere ?
Poichè l'equilibrio (dvd, dvd) è efficiente esso va scelto da R e da C.

Osserviamo che è essenziale che sia R che C facciano la stessa scelta, se R sceglie la strategia di equilibrio "dvd" e C la strategia di equilibrio "cd" entrambi vanno in perdita.

Notiamo ancora che se R, per qualche motivo, decide di usare i vecchi CD, la cosa migliore per C è adeguarsi a tale scelta.

Anche in questo gioco, come nel caso delle stazioni radio, è importante allora che i giocatori comunichino e coordinino le loro azioni.

Esempio 19

Consideriamo il gioco del "fifone". R e C su due auto si lanciano l'uno contro l'altro, se nessuno dei due sterza cimitero, se entrambi sterzano si salvano entrambi, se solo uno sterza è un "chicken" e perde la faccia mentre l'altro è un coraggioso. La matrice diventa:

	non sterza	sterza
non sterza	-10,-10	6,-1
sterza	-1,6	2,2

Non ci sono strategie dominanti. Troviamo gli equilibri di Nash:

	non sterza	sterza
non sterza	-10,-10	(6),[-1]
sterza	(-1),[6]	2,2

Vi sono due equilibri in strategie pure: (sterza, non sterza) e (non sterza, sterza). Il problema per R è di convincere C che egli non sterzerà per cui il meglio che C può fare è sterzare. Lo stesso problema ha C e in conclusione può finire male per tutti e due.

Mancando le strategie dominanti possiamo cercare anche un equilibrio in strategie miste.

Supponiamo che R scelga la riga 1 per cui $p_1 = 1$ e C adotti la strategia mista $(q_1, q_2) = (q, 1-q)$, allora la vincita media di R sarà:

$$E(\text{riga } 1) = \sum_{j=1}^{2} x_{1j}\ p_1\ q_j = -10 \times 1 \times q + 6 \times 1 \times (1-q) = -10q + 6(1-q)$$

Analogamente, una volta che R ha scelto la riga 2, $p_2 = 1$ e la vincita media di R sarà:

$$E(\text{riga } 2) = \sum_{j=1}^{2} x_{2j}\ p_2\ q_j = -1 \times 1 \times q + 2 \times 1 \times (1-q) = -q + 2(1-q)$$

Ora, quello che vuole C è che per R sia indifferente scegliere riga1 o riga 2, cioè scegliere una riga invece che l'altra non deve dare a R alcun vantaggio. In altri termini, la vincita media di R scegliendo la riga 1 deve essere uguale a quella ottenuta scegliendo la riga 2, cioè deve essere:

$-10q + 6(1-q) = -q + 2(1-q)$

$-10q + 6 - 6q = -q + 2 - 2q$

$-16q + 3q = -6 + 2$

$-13q = -4$

$q = 4/13 \approx 1/3$

$1-q = 1 - 4/13 = 9/13 \approx 2/3$

In conclusione, la strategia di C che rende a R indifferente ai fini della vincita media la scelta della riga 1 o della riga 2 è individuata dal vettore $(q_1, q_2) = (4/13, 9/13)$.

Simmetricamente, anche R vuole che C non abbia motivo per preferire la colonna 1 alla colonna 2 o viceversa.
Ora, nel momento in cui C sceglie la colonna 1, $q_1 = 1$ e il valore atteso della sua vincita dipende solo dalla strategia di R, descritta dal vettore $(p_1, p_2) = (p, 1-p)$. La vincita media di C relativa alla scelta della colonna 1 è:

$$E(\text{colonna 1}) = \sum_{i=1}^{2} x_{i1}\, p_i q_1 = -10 \times p \times 1 + 6 \times (1-p) \times 1 = -10p + 6(1-p)$$

Analogamente, una volta che C ha scelto la colonna 2, $q_2 = 1$ e la vincita media di C sarà:

$$E(\text{colonna 2}) = \sum_{i=1}^{2} x_{i2}\, p_i q_2 = -1 \times p \times 1 + 2 \times (1-p) \times 1 = -p + 2(1-p)$$

L'obiettivo di R è che per C sia indifferente scegliere colonna 1 o colonna 2, cioè scegliere una colonna invece che l'altra non deve avvantaggiare R. In altri termini, la vincita di C scegliendo la colonna 1 deve essere uguale a quella ottenuta scegliendo la colonna 2, cioè deve essere:

$-10p + 6(1-p) = -p + 2(1-p)$

$-10p + 6 - 6p = -p + 2 - 2p$

$-16p + 3p = -6 + 2$

$-13p = -4$

$p = 4/13 \approx 1/3$

$1-p = 1 - 4/13 = 9/13 \approx 2/3$

In conclusione, la strategia di R che rende a C indifferente, ai fini della vincita media, la scelta della colonna 1 o della colonna 2 è individuata dal vettore $(p_1, p_2) = (4/13, 9/13)$. La strategia è la stessa di C essendo il gioco simmetrico.

Abbiamo perciò individuato un terzo equilibrio di Nash descritto dalla coppia di strategie miste:

$(p_1, p_2) = (4/13, 9/13)$

$(q_1, q_2) = (4/13, 9/13)$

Abbiamo individuato due equilibri puri e uno misto per il gioco. Quali sono le vincite medie per i tre equilibri ?

1) (sterza, non sterza): costantemente, R vince -1, C vince 6

2) (non sterza, sterza): ogni volta, R vince 6, C vince -1

3) (4/13, 9/13): le vincite medie sono le seguenti:

La strategia di R è tale che per lui è indifferente scegliere la riga 1 o la riga 2 per cui la vincita media è data da:

E(riga 1) = -10 x 4/13 + 6 x 9/13 = 14/13 =

= E(riga 2) = -1 x 4/13 + 2 x 9/13 = 14/13

La vincita media di C è invece data da:

E(colonna 1) = -10 x 4/13 + 6 x 9/13 = 14/13 =

= E(colonna 2) = -1 x 4/13 + 2 x 9/13 = 14/13

Anche in questo esempio, il gioco è simmetrico e i valori attesi coincidono.

L'individuazione dell'equilibrio migliore è uno dei problemi centrali della teoria dei giochi. Nel caso dell'Esempio 14 abbiamo visto che il secondo equilibrio puro era efficiente e anche superiore rispetto all'equilibrio misto, nel caso dell'Esempio 19 l'equilibrio (dvd, dvd) era anche lui efficiente, ora invece nessuno dei due equilibri puri lo è.

Infatti, i due equilibri sono asimmetrici: l'equilibrio (sterza, non sterza) fa vincere -1 a R e 6 a C, con l'equilibrio (non sterza, sterza) R guadagna 6 e C perde 1. Il terzo equilibrio in media fa vincere 14/13, poco più di 1 e, rispetto ai due equilibri puri, ha il vantaggio di essere simmetrico, entrambi i giocatori hanno la stessa vincita.

Osserviamo che il buon senso suggerisce che la strategia più razionale sia (sterza, sterza) la quale però non è un equilibrio di Nash.

Esempio 20

La matrice ora vista può anche rappresentare un problema completamente diverso e più interessante. Supponiamo che Boeing e Airbus stiano pensando di costruire un superjumbo ma l'analisi di mercato dice che c'è posto per solo un costruttore.

Se costruiscono entrambi è un disastro economico. Se uno solo costruisce il superjumbo grandi profitti e piccole perdite per l'altro, se entrambi rinunciano profitti discreti. La matrice è la stessa dell'Esempio 19:

	costruisce	non costruisce
costruisce	-10,-10	6,-1
non costruisce	-1,6	2,2

I due equilibri sono (costruisce, non costruisce) e (non costruisce, costruisce). Il punto è che Boeing per guadagnare 10 deve convincere Airbus che costruirà qualsiasi cosa succeda, se Airbus crede ciò le conviene non costruire (meglio perdere 1 che 10).

Il simmetrico avviene per Airbus per cui si vede che la cosa migliore è, se possibile, trovare un accordo mediante compensazioni per chi non costruisce.

In mancanza di accordo entrambe le aziende possono annunciare la ferma decisione di costruire il superjumbo sperando che l'altra ci creda e rinunci.

Il dilemma del prigioniero

Abbiamo visto sinora vari giochi e abbiamo accennato ai problemi che sorgono quando non vi è un equilibrio in strategie dominanti e dobbiamo gestire gli equilibri di Nash. Sembra perciò che quando esiste una strategia dominante tutto vada liscio e senza problemi. Le cose non stanno così, infatti consideriamo la seguente matrice:

$$\begin{pmatrix} -5, -5 & 0, -10 \\ -10, 0 & -1, -1 \end{pmatrix}$$

Si vede subito che esiste l'equilibrio in strategie dominanti (riga 1, colonna 1) il quale è ovviamente anche un equilibrio di Nash:

$$\begin{pmatrix} (-5),[-5] & (0), -10 \\ -10,[0] & -1, -1 \end{pmatrix}$$

Sembra un gioco assolutamente "innocente" la cui soluzione è banale. Nulla di più sbagliato, questa matrice rappresenta il cosiddetto "dilemma del prigioniero", un gioco di grande importanza teorica e applicativa che si può formulare come segue:

Avviene in città un furto con scasso. Due sospetti, R e C, sono arrestati dalla polizia la quale però non ha prove sufficienti per incriminarli. La polizia li visita separatamente e suggerisce a entrambi di accusare l'altro del furto. Se uno accusa e l'altro tace, il delatore viene rilasciato e l'accusato prende 10 anni di prigione. Se entrambi tacciono passano un anno in prigione per porto abusivo di armi. Se si accusano a vicenda cinque anni di prigione ciascuno.

In questa storia le strategie sono: accusare, tacere e le "vincite" sono gli anni di prigione. La matrice di gioco è pertanto la seguente:

	accusa	tace
accusa	-5, -5	0, -10
tace	-10, 0	-1, -1

Si vede subito che sia R che C hanno la strategia dominante "accusa" per cui l'unico equilibrio è (accusa, accusa).

Sembrerebbe quindi tutto chiaro, a entrambi i prigionieri conviene accusare l'altro. Le cose non sono così semplici, dimentichiamo un momento la teoria dei giochi, mettiamoci nei panni di R e C che non sanno nulla di strategie dominanti ed equilibri di Nash e vediamo se effettivamente la strategia (accusa, accusa) è la più razionale per R e C al fine di minimizzare gli anni di prigione.

R ragiona così: C può fare solo due cose: o mi accusa o tace. Se C mi accusa e io taccio becco dieci anni, se C mi accusa e io accuso lui, mi danno 5 anni, quindi, nel caso che C mi accusi, è meglio che io faccia lo stesso.

L'altra possibilità è che C taccia: in tal caso se anch'io taccio mi danno un anno, se invece lo accuso sono libero. Conclusione, anche nel caso in cui C taccia, mi conviene accusarlo.

Nel frattempo, anche C cerca di fare una scelta razionale e giunge ovviamente anche lui alla conclusione che gli conviene accusare R per minimizzare la pena.

Come si vede il ragionamento fatto con il solo buon senso da R e da C conferma che la strategia (accusa, accusa) è la migliore.

Pertanto, R e C si accusano l'un l'altro e si fanno cinque anni di prigione ciascuno, se fossero stati zitti entrambi avrebbero avuto un solo anno.

Questa storiella mostra che azioni individuali scelte razionalmente per minimizzare una perdita possono provocare una perdita individuale maggiore di una scelta apparentemente irrazionale. Non sempre le scelte *razionali* sono *sensate*.

Si potrebbe perciò concludere che la scelta migliore per R e C è di tacere. In effetti la cosa è più sottile di quanto appare in quanto supponiamo che:

1) la polizia, fatta la proposta, dice che il giorno dopo tornerà per acquisire separatamente la risposta di ciascuno e concede anche ai due di incontrarsi.

2) R e C si incontrano, analizzano il problema, concordano che la cosa migliore è tacere, giurano che l'indomani taceranno entrambi.

3) il giorno dopo la polizia va nella cella di R e gli chiede cosa ha deciso di fare: R, il cui obiettivo primario è minimizzare la pena, può:

a) credere che C terrà fede al giuramento e tacerà, ma allora perchè farsi un anno di galera, meglio tradire, accusare C ed essere rilasciato

b) credere che C verrà meno al giuramento, nel qual caso è stupido tacere e farsi 10 anni di prigione, meglio accusare C e farsi soli 5 anni.

C ragiona ovviamente nello stesso modo e, in conclusione, R e C si accusano a vicenda ed entrambi si fanno 5 anni invece che uno solo.

Il dilemma nasce dal fatto che anche dopo che R e C hanno concordato sulla utilità di tacere, entrambi hanno interesse a venir meno all'accordo.

Esiste un conflitto fra l'interesse del gruppo e l'interesse individuale. Se i due prigionieri cooperano il gruppo ci guadagna e loro stessi minimizzano la pena. Se invece ognuno guarda il proprio interesse, conviene accusare, e la conseguenza è che il gruppo e loro stessi ci vanno a perdere.

Questo dilemma, apparentemente di scarsa importanza, è stato ampiamente studiato in quanto vi sono molti esempi di interazione nel mondo reale che rappresentano situazioni analoghe.

Il dilemma esemplifica la difficoltà di analizzare razionalmente alcune situazioni e ha dato luogo a studi in campi tra loro diversissimi quali, ad esempio, scienze sociali, economia, politica, biologia, etnologia.

Ad esempio, pensiamo allo sfruttamento eccessivo dei banchi di pesce. Se la pesca viene limitata, il pesce ha il tempo sufficiente per riprodursi e tutti ci guadagnano, però se un peschereccio pensa che gli altri non si limiteranno, pescherà il più possibile e alla fine il pesce sparirà per tutti.

Il problema può essere rappresentato mediante la matrice del dilemma oppure anche, per semplicità, con quest'altra equivalente:

	pesca intensiva	pesca moderata
pesca intensiva	1,1	3,0
pesca moderata	0,3	2,2

Osserviamo che i rapporti maggiore-minore tra i valori degli elementi sono identici a quelli presenti nel dilemma del prigioniero, per cui anche ora abbiamo la strategia dominante (pesca intensiva, pesca intensiva) la quale però dà una vincita inferiore a (pesca moderata, pesca moderata).

Un altro esempio è la corsa agli armamenti fra due stati. Entrambi hanno due opzioni: aumentare le spese militari oppure accordarsi per ridurre gli armamenti. Nessuno dei due stati può essere sicuro che l'altro terrà fede all'accordo, pertanto entrambi ritengono "razionale" continuare la corsa giungendo a un risultato "irrazionale" (maggiori probabilità di conflitto, miliardi sprecati in armi, etc.).

Un altro esempio è dato da ciò che accade spesso in una corsa ciclistica. Supponiamo che due ciclisti siano in fuga e il gruppo insegua a una certa distanza. I due ciclisti possono cooperare, alternandosi in testa, oppure non cooperare, uno dei due fa il furbo e sta sempre dietro per risparmiare energie in vista della volata finale, quello davanti ovviamente non è d'accordo, rallenta e la conseguenza è che vengono raggiunti dal gruppo.

Ancora, in Nuova Zelanda i giornali sono collocati in cassette aperte: si prende il giornale e si inserisce la moneta in una scatola. E' possibile perciò prendere il giornale senza pagare, quasi nessuno lo fa in quanto, se il fenomeno si diffondesse, sparirebbe questo modo comodo di comprare il giornale. La massima parte dei cittadini rinuncia al vantaggio individuale di rubare il giornale per guadagnare il vantaggio in comodità dell'acquisto.

Il dilemma illustra un conflitto fra razionalità individuale e di gruppo. Un gruppo in cui ciascun componente persegue interessi personali può avere risultati peggiori di un gruppo che persegue interessi collettivi. Altri esempi possono essere l'inquinamento e la congestione del traffico.

Il conflitto fra obiettivi individuali e il bene comune è alla radice di molte situazioni reali e da ciò nasce l'importanza del dilemma. Si tratta di situazioni in cui azioni individuali apparentemente razionali provocano un danno collettivo e individuale maggiore.

Le cose come si vede sono abbastanza complicate. In effetti lo sono ancora di più in quanto negli esempi che abbiamo finora visto la strategia "cooperare" è vantaggiosa rispetto a quella "pensa solo a te stesso". Si tratta però sempre di un vero vantaggio ?

Esempio 21

Nell'Esempio 4 abbiamo visto un gioco artificiosamente reso a somma zero in cui c'era da decidere il prezzo delle bottiglie di acqua minerale. Vogliamo ora vedere un gioco, un poco più realistico, in cui supponiamo che R e C siano Coca Cola e Pepsi le quali fanno un'analisi del mercato al fine di decidere una strategia di prezzi per le loro lattine. Se entrambe vendono a 1 euro guadagnano 3 milioni mensili ciascuna, se vendono a 2 euro guadagnano 6 milioni ciascuna. Se R vende a 1 euro e C vende a 2, R guadagna 8 milioni e C solo 2. Nella situazione inversa accade l'inverso. La matrice di gioco è la seguente:

	1 euro	2 euro
1 euro	3, 3	8, 2
2 euro	2, 8	6, 6

La matrice è ancora una volta sostanzialmente quella del dilemma. C'è un'unica strategia dominante (1 euro, 1 euro). Chiaramente a Coca e Pepsi conviene fare un "cartello" e vendere a 2 euro, raddoppiando in tal modo i profitti. Tale cartello va però a unico vantaggio dei giocatori, chi ci rimette sono i consumatori. In altri termini, la cooperazione all'interno di un gruppo spesso non è nell'interesse del resto della società.

Inoltre va detto che, nell'ipotesi che Coca e Pepsi si siano accordate per vendere a 2 euro, entrambe possono guadagnare di più barando al gioco e vendendo a 1 euro: il "traditore" lo fa per aumentare il profitto da 6 a 8 milioni ma in tal modo finisce col favorire i consumatori.

In questo esempio la strategia dominante (1 euro, 1 euro) è quella che, a differenza degli altri esempi, effettivamente appare la migliore per la società nel suo complesso. E' forte però la tentazione di fare un cartello ed eventualmente barare poi al gioco.

Quest'ultimo esempio è senz'altro utile per capire la complessità dei problemi che si celano dietro una storiella apparentemente banale. Il dilemma del prigioniero ha un solo equilibrio di Nash e anzi un unico equilibrio in strategie dominanti: quando entrambi i prigionieri accusano.

E' vero che la strategia "entrambi tacciono" porta a risultati migliori della "entrambi accusano" però la prima strategia è instabile in quanto un prigioniero può, se l'altro tace, migliorare il suo risultato accusando. Vediamo perciò che anche la presenza di una strategia dominante non mi garantisce di ottenere il risultato migliore.

Ricordando che l'obiettivo essenziale della teoria dei giochi è individuare il comportamento razionale, possiamo concludere che, nel caso dei giochi a somma zero, la soluzione minimax è effettivamente la migliore ma è riferita a giochi di scarsa importanza applicativa, invece, nel caso dei giochi non a somma zero, i giochi sono molto più interessanti ma i relativi equilibri possono essere molteplici e, anche se sono unici, possono fornire, come nel dilemma del prigioniero, soluzioni problematiche.

In altri termini alla domanda "Qual è il comportamento razionale in questo gioco ?" vi è una risposta chiara e univoca nel caso dei giochi a somma zero e una o più risposte problematiche nell'altro caso.

Abbiamo solo accennato ai concetti di base della teoria dei giochi, ma dovrebbe essere chiara la sua rilevanza in una molteplicità di contesti. Il testo di Russell-Norvig ci dice che essa "è utilizzata per prendere decisioni molto importanti che includono i procedimenti per bancarotta, la vendita delle frequenze radio, lo sviluppo dei prodotti e il relativo prezzo sul mercato, la difesa nazionale e altre situazioni che coinvolgono miliardi di dollari e centinaia di migliaia di vite".

Sommario

La *teoria dei giochi* cerca di individuare il *comportamento razionale* in presenza sia di *giochi a somma zero* sia di *giochi non a somma zero*.

Campi diversissimi tra loro sono campi di applicazione per la teoria dei giochi: ricordiamo, ad esempio, l'economia, l'ambiente, l'evoluzione, le strategie militari, la psicologia.

Giochi a somma zero: in presenza di un *punto di sella* nella *matrice del gioco* vi è una *strategia pura ottimale* per entrambi i giocatori. Altrimenti il *teorema di von Neumann* ci assicura l'esistenza di *strategie miste ottimali*.

Giochi non a somma zero: il *teorema di Nash* estende il teorema minimax a un qualsiasi numero di giocatori. Inoltre assicura l'esistenza di almeno un *punto di equilibrio* sia per giochi a somma zero sia per quelli che non lo sono.

Il *dilemma del prigioniero* mostra come nei giochi non a somma zero il concetto di comportamento razionale sia molto più difficilmente individuabile rispetto ai giochi a somma zero.

7. Elaborazione del linguaggio naturale

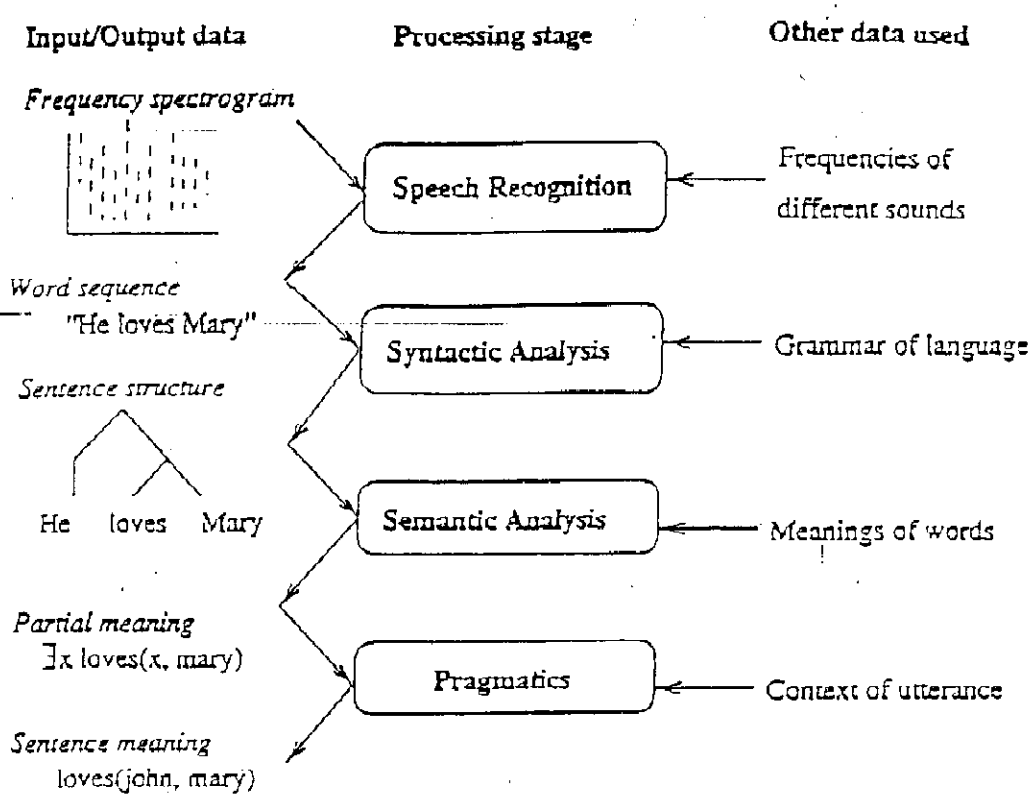
Vogliamo ora esaminare le problematiche di un'area specifica dell'AI, cioè il trattamento del linguaggio naturale. Si tratta di un problema che un bambino risolve brillantemente mentre il computer incontra difficoltà notevolissime.

L'elaborazione del linguaggio naturale include la sua *comprensione* e la sua *generazione*. La comprensione parte da una frase scritta o parlata e si propone di capire il suo significato in modo da poter poi intraprendere un'azione opportuna. La generazione riguarda il problema opposto: partire da una rappresentazione formale di ciò che vuoi dire e trovare il modo di esprimerlo in inglese o un altro linguaggio naturale. Noi esamineremo principalmente i problemi della comprensione, con l'obiettivo di trovare il significato di una frase inglese.

I sistemi per l'elaborazione del linguaggio naturale sono costruiti sia per esplorare le teorie riguardanti i linguaggi elaborati dall'uomo sia per motivi pratici quali l'interfacciamento in linguaggio naturale. In questo caso bisogna capire la frase dell'utente e tradurla in una forma adatta per l'applicazione. Ad esempio, un front end in linguaggio naturale per un database deve tradurre una richiesta in inglese in una query formale al database.

L'utente può voler comunicare con il sistema o con la voce o con una tastiera. Capire il linguaggio parlato è molto più difficile che capire lo scritto, l'input sono dei segnali prelevati con un microfono. Prima di affrontare il problema di comprendere la frase pronunciata dobbiamo trasformare i segnali in parole. Questo è il problema di *riconoscimento del parlato*, è il primo passo per la comprensione del linguaggio, e il suo output è una sequenza di parole.

Una volta che la sequenza di parole è ottenuta, il resto del processo di comprensione è suddiviso nelle seguenti fasi: *analisi sintattica, analisi semantica* e *analisi pragmatica*. La seguente figura illustra le fasi coinvolte nel processo di comprensione del linguaggio:



Riconoscimento del parlato: i segnali analogici provenienti dal microfono vengono analizzati e viene ricavata la sequenza di parole.

Analisi sintattica: La sequenza di parole viene analizzata utilizzando la *grammatica* del linguaggio e viene ricavata la *struttura* della frase.

Analisi semantica: Utilizzando l'informazione circa la struttura della frase e il significato delle parole presenti, viene ricavata una rappresentazione parziale del significato della frase.

Analisi pragmatica: Infine, la rappresentazione parziale del significato della frase è completata utilizzando informazione riguardante il *contesto*, cioè quando e dove è stata detta la frase, e chi l'ha detta a chi.

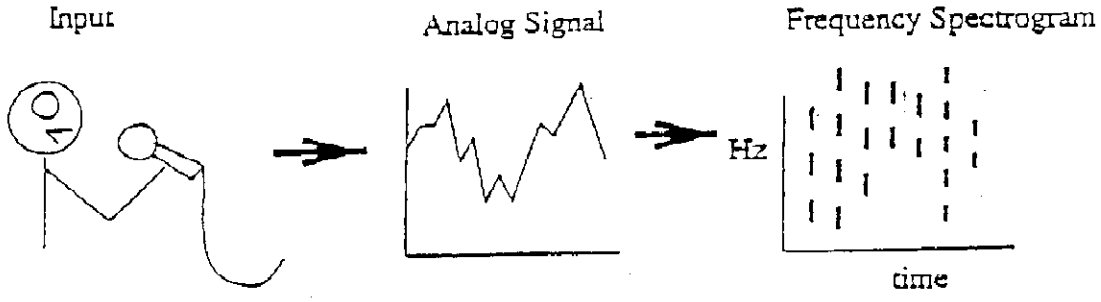
Esamineremo ora queste quattro fasi, con particolare attenzione all'analisi sintattica. Discuteremo quindi il problema dell'*ambiguità* del linguaggio e infine accenneremo all'altro aspetto dell'elaborazione del linguaggio, cioè la sua generazione.

Riconoscimento del parlato

Vediamo ora come un segnale analogico proveniente da un microfono può essere analizzato al fine di ricavare la sequenza di parole pronunciate. Il compito è difficile ma la posta in gioco è alta: comunicare con il computer parlandogli invece che con una tastiera è molto attraente. Non bisogna digitare tasti, si può lavorare al telefono, ad alta velocità e con le mani libere.

Certamente ci vorrà ancora del tempo per poter dialogare con il computer su qualsiasi argomento però anche una comprensione del parlato limitata e specializzata risulta utile e sistemi siffatti sono oggigiorno ampiamente utilizzati.

La figura che segue illustra il modo in cui viene ottenuto lo *spettrogramma di frequenze*, che è il punto di partenza per l'analisi del parlato. L'output di un microfono è un *segnale analogico* che può essere suddiviso in *frequenze* utilizzando appositi filtri. Questa operazione separa il suono in toni alti e toni bassi. Si può misurare l'energia presente in ciascuna frequenza e ricaviamo uno spettrogramma che illustra l'andamento temporale delle frequenze.



Una volta ricavato lo spettrogramma dobbiamo individuare le parole da cui ha avuto origine. Ci sono due fasi: la prima consiste nell'individuare i suoni di base.

Tutti i linguaggi naturali usano un piccolo numero (circa 50) di suoni di base detti *fonemi*. Un fonema, ad esempio, è il suono "a" in "cat" o il suono "sh" in "shopping". Ogni fonema ha un suo spettrogramma caratteristico (*template*) e possiamo quindi ottenere un catalogo che contiene, per ogni fonema, il suo template.

Ora, avendo a disposizione uno spettrogramma di parlato dobbiamo scoprire quali fonemi sono stati pronunciati cercando di accoppiare frammenti dello spettrogramma con i templates nel catalogo, cercando il match migliore. Questo processo è detto *template matching*.

La seconda fase parte dai fonemi individuati e cerca di arrivare alle parole. Per esempio, partendo dai fonemi "th", "i", e "ng" deduciamo che la parola è "thing". Se la fase precedente ha correttamente individuato i fonemi è sufficiente fare il matching con il dizionario fonetico della lingua in esame.

Il problema è che ottenere i fonemi giusti non è affatto semplice per le ragioni che vedremo, per cui al termine della prima fase abbiamo solo delle probabilità associate a fonemi diversi a seconda della bontà del match. Ad esempio, un suono potrebbe essere il fonema "th" al 50%, ma anche "f" al 30%. Le cose sono ulteriormente complicate dal fatto che a volte una parola può essere pronunciata in modi diversi (ad esempio, tomato).

Trovare quale è la la parola pronunciata richiede perciò informazione *statistica*, cioè le probabilità dei singoli fonemi, le probabilità delle differenti pronunce, le probabilità delle parole stesse.

Notiamo inoltre che la probabilità di una parola dipende dalle parole che la precedono, ad esempio è altamente improbabile che dopo "a" si presenti "the". Bisogna dire che i metodi utilizzati per trovare le probabilità delle parole hanno avuto pieno successo nel riconoscimento del parlato.

Applicazioni pratiche

Semplici riconoscitori del parlato sono ampiamente usati, specialmente per applicazioni telefoniche. Ad esempio, in caso di telefonate con pagamento a carico del chiamato, il sistema chiede e registra il nome del chiamante, chiama il numero desiderato, dice il nome e chiede al chiamato se accetta il pagamento.

Il numero di risposte è limitato (ad esempio, yes, no, OK) ma non per questo il suo compito è banale in quanto deve essere *speaker independent* pertanto deve avere molti templates per gestire tutte le possibili pronunce delle risposte.

Altri tipici sistemi di riconoscimento sono in uso per il pagamento di bollette, ricariche telefoniche, applicazioni bancarie, anche in questi casi l'insieme di risposte è molto limitato (cifre, si, no) e il sistema chiede conferma delle informazioni acquisite, in caso di incomprensione il sistema smista la chiamata a un operatore.

Problemi e complicazioni

Il problema principale nel riconoscimento del parlato è che non vi è una semplice corrispondenza tra segnale e parola. La stessa parola può essere pronunciata in modi molto diversi da persone diverse che possono usare dialetti diversi ed essere di differenti età. Anche la stessa persona può pronunciare la parola in modi diversi (lento, veloce, intonazione). Inoltre lo stesso suono può corrispondere a diverse parole (bear-bare, sun-son).

Poiché normalmente non vi è stacco tra le parole pronunciate è anche difficile stabilire dove una parola finisce e un'altra comincia. Un esempio classico è la frase "How to recognize speech" la cui pronuncia è vicinissima a quella di "How to wreck a nice beach".

Il rumore di fondo complica le cose, il segnale che contiene "Hello" può contenere anche il rumore del vento. Se le parole sono dette a telefono altri problemi possono sorgere in quanto le frequenze alte vengono eliminate per cui certi fonemi vengono a coincidere (ad esempio, "f" e "s").

A causa di tutti questi problemi i sistemi di riconoscimento spesso fanno delle assunzioni semplificative. Un'assunzione è che essi gestiscono solo un utente a differenza dei sistemi indipendenti dall'utente. In un sistema singolo utente i templates usati per fare il matching dei fonemi provengono proprio dall'utente il quale deve "allenare" il sistema pronunciando suoni e parole specificati.

Un'altra assunzione può essere che ci devono essere pause tra le parole. Un'altra ancora è avere un vocabolario ristretto, ad esempio "sì", "no" e le cifre zero-nove.

L'output di un riconoscitore del parlato è una sequenza di parole e per alcune applicazioni questo è sufficiente. Ad esempio un sistema telefonico può dover riconoscere solo "yes" e "no". Anche un sistema per la dettatura non ha problemi di comprensione del testo in quanto deve solo ricavare il testo che sarà poi gestito tramite un word processor.

Però per applicazioni più complesse dobbiamo capire il significato delle frasi e per questo abbiamo bisogno delle altre fasi prima elencate.

Analisi sintattica

L'*analisi sintattica* ci aiuta a capire come le parole sono raggruppate insieme per formare le frasi e ci fornisce il punto di partenza per risolvere il problema della comprensione delle frasi. Per esempio, consideriamo le due frasi:

1) Il cane mangiò l'osso
2) L'osso fu mangiato dal cane

Analizzare la *struttura* della frase, tramite le regole della sintassi, ci aiuta a capire che è l'osso ad essere mangiato e non il cane. Una regola del tipo "E' il secondo sostantivo che viene mangiato" non funziona.

L'analisi sintattica ci permette di individuare possibili raggruppamenti di parole all'interno di una frase. A volte un solo raggruppamento sarà possibile e avremo fatto un passo notevole verso la comprensione della frase. Per esempio, nella frase seguente:

3) Il coniglio dalle lunghe orecchie mangiò una grande lattuga

possiamo ricavare dalle regole della sintassi che il raggruppamento di parole "Il coniglio dalle lunghe orecchie" è il soggetto e "una grande lattuga" l'oggetto.

Quando passiamo all'individuazione del significato della frase, il problema è semplificato in quanto possiamo innanzitutto capire il significato di questi due raggruppamenti e poi combinarli per ottenere il significato dell'intera frase.

In altri casi possono esserci più raggruppamenti. Ad esempio, nella frase "John vide Mary con un telescopio" vi sono due raggruppamenti possibili:

4i) John vide (Mary con un telescopio) Mary aveva un telescopio con sé

4ii) John (vide Mary con un telescopio) la vide mediante un telescopio

Quando vi sono più possibili raggruppamenti si dice che la frase è *sintatticamente ambigua*. A volte possiamo usare conoscenza di tipo generale per individuare il raggruppamento giusto.

Ad esempio, se la frase era "John vide Mary con un libro" solo la 4i) diventa possibile.

Ancora, se abbiamo:

5) I saw the Forth bridge flying into Edimburgh

Vi sono due possibili significati, uno in cui vedo da un aereo il ponte e l'altro in cui è il ponte che vola. Conoscendo le caratteristiche dei ponti possiamo escludere il secondo significato.

Nell'esempio 4) la conoscenza di tipo generale è insufficiente a decidere il significato, dovremmo conoscere anche le abitudini di John. Comunque l'analisi sintattica ci aiuta a determinare il significato di una frase ricavando i possibili raggruppamenti di parole.

Scrivere una grammatica

Le regole della sintassi specificano la possibile organizzazione delle parole nelle frasi e ci permettono di determinare la struttura di una specifica frase, cosa che aiuterà a stabilire il suo significato. Le regole costituiscono la *grammatica* del linguaggio.

Per analizzare la struttura di una frase abbiamo bisogno di un *parser* (analizzatore) che, data una frase e una grammatica, stabilisce se la frase è corretta per quella grammatica e in caso positivo restituisce una rappresentazione, *parse tree,* della struttura della frase. Il parse tree mostra i possibili raggruppamenti delle parole in costrutti sintattici di livello superiore.

Vediamo ora come possono essere scritte delle semplici grammatiche, successivamente discuteremo come può essere effettuato il processo di parsing.

La grammatica per un linguaggio naturale specifica le strutture ammissibili per le frasi in termini di categorie sintattiche di base quali i sostantivi e i verbi. E' definita in modo simile alle grammatiche per il linguaggi di programmazione, ma sarà certamente più complessa e difficilmente potrà coprire tutte le possibili frasi del linguaggio.

Nel linguaggio naturale noi analizziamo una frase per determinare la sua struttura e come primo passo per arrivare al suo significato. La maggior parte delle grammatiche si preoccupano solo delle frasi grammaticalmente corrette in quanto l'analisi di frasi scorrette ma significative è molto più difficile. Ci limitiamo pertanto a semplici grammatiche che riconoscono le frasi corrette ma non sono in grado di gestire frasi grammaticalmente scorrette.

Il punto di partenza per descrivere la struttura di un linguaggio naturale è usare una semplice *context free grammar* (grammatica libera dal contesto), queste grammatiche hanno la caratteristica che la parte sinistra della regola contiene una sola categoria sintattica. Supponiamo che vogliamo una grammatica che riconosca frasi del tipo:

1) John ate the biscuit

2) The lion ate the schizophrenic

3) The lion kissed John

ma non sia in grado di riconoscere frasi scorrette del tipo:

1) Ate John biscuit the

2) Schizophrenic the lion ate

3) Biscuit lion kissed

Una semplice grammatica per i nostri scopi è la seguente:

```
sentence --> noun_phrase, verb_phrase.
noun_phrase --> proper_noun.
noun_phrase --> determiner, noun.
verb_phrase --> verb, noun_phrase.

proper_noun --> [mary].
proper_noun --> [john].
noun --> [schizophrenic].
noun --> [biscuit].
verb --> [ate].
verb --> [kissed].
determiner --> [the].
```

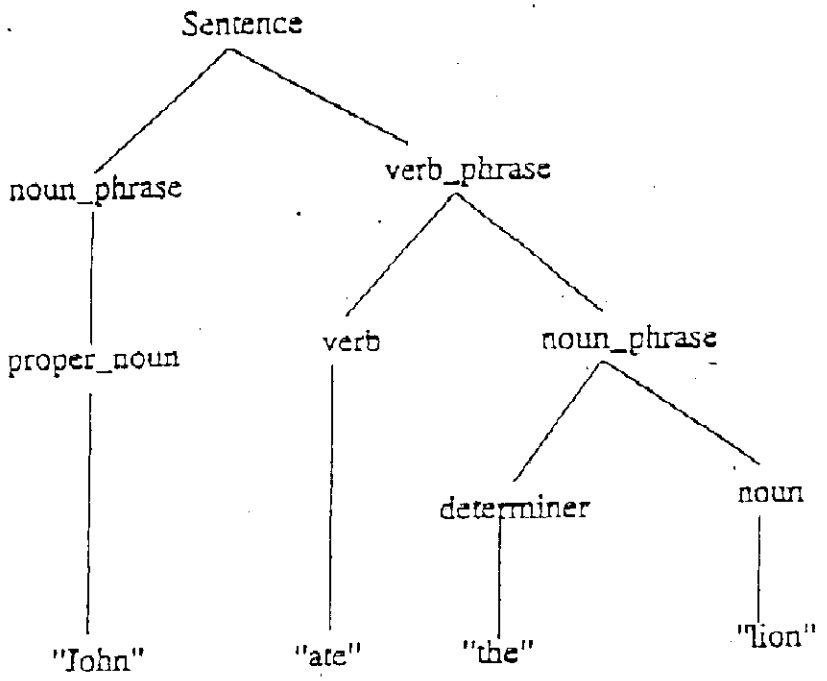
La prima regola dice che una frase *consiste* di una *noun phrase* seguita da una *verb phrase*. La noun phrase corrisponde al *soggetto* della frase mentre la verb phrase corrisponde al *predicato*.

Ad esempio, in "John ate the biscuit", "John" è la noun phrase e "ate the biscuit" è la verb phrase.

Le altre regole si possono leggere come segue. Una noun phrase consiste di un nome proprio oppure di un articolo seguito da un nome. Una verb phrase consiste di un verbo seguito da una noun phrase. Le ultime regole si limitano ad attribuire a varie parole la loro categoria sintattica.

Data una grammatica possiamo illustrare la struttura sintattica della frase mediante il parse tree, che mostra come la frase può essere scissa nei suoi costituenti sintattici. Questa informazione tornerà utile in fase di analisi semantica.

La seguente figura mostra il parse tree per "John ate the lion", riconosciuta dalla nostra grammatica se si aggiunge la regola noun -- > [lion] :



Le frasi scorrette non sono riconosciute dalla grammatica. Ad esempio "biscuit lion kissed" inizia con due sostantivi, la qual cosa non è ammessa dalla grammatica.

Notiamo però che una frase del tipo "The biscuit kissed John" è sintatticamente corretta e viene riconosciuta dalla grammatica in quanto la sua scorrettezza è di tipo semantico.

La grammatica pertanto *genera*, partendo dalla categoria sintattica "sentence", le frasi appartenenti al linguaggio *riconosciuto* o *generato* dalla grammatica. A ogni passo viene *espansa* un categoria sintattica *riscrivendola* nel modo indicato dalla regola applicata.

Accordo del numero soggetto-verbo

La grammatica precedente è estremamente semplice, consideriamo le due frasi:

- Mary eat the lion
- Mary eats the ferocious lion

Se aggiungiamo le due regole verb --> [eat] e verb --> [eats] la prima frase viene riconosciuta mentre la seconda no in quanto gli aggettivi non vengono gestiti dalla grammatica. Ora la prima frase è scorretta in quanto il verbo dovrebbe essere "eats" mentre la seconda è corretta e dovrebbe essere accettata modificando opportunamente la grammatica.

Per gestire il primo problema dobbiamo imporre il rispetto del numero fra soggetto e verbo in modo che "I am" e "We are" sono accettate mentre "I are" e "We am" non lo sono. Per risolvere il problema degli aggettivi bisogna aggiungere altre regole alla grammatica.

Per imporre il rispetto soggetto-verbo possiamo aggiungere argomenti alle regole della grammatica cioè avere:

```
sentence --> noun_phrase(Num), verb_phrase(Num).
noun_phrase(Num) --> proper_noun(Num).
noun_phrase(Num) --> determiner(Num), noun(Num).
verb_phrase(Num) --> verb(Num), noun_phrase(_).

proper_noun(s) --> [mary].
noun(s) --> [lion].
noun(p) --> [lions].
det(s) --> [the].
det(p) --> [the].
verb(s) --> [eats].
verb(p) --> [eat].
```

Gli argomenti che iniziano con una maiuscola (Num) sono delle variabili, mentre gli argomenti che iniziano con le minuscole (s, p) indicano specifiche entità, nel nostro caso nome e verbo singolare oppure plurale.

Num si riferisce al *numero* di un sostantivo o un verbo che può essere singolare (s) o plurale (p).

Pertanto ora una frase consiste ancora di una noun phrase seguita da una verb phrase però queste due categorie hanno un numero associato Num che deve essere lo stesso per entrambe. A sua volta una noun phrase ha anch'essa un numero associato e lo stesso accade per la verb phrase.

Notiamo però che per quest'ultima la noun phrase non deve accordarsi con il numero del verbo (ad esempio, sono corrette sia "John eats the lion", sia "John eats the lions") e questo viene indicato con il simbolo "-".

In generale, esprimere le giuste concordanze mediante una grammatica è molto complesso e bisogna che il dizionario contenga, per limitare le sue dimensioni, anche delle informazioni di carattere generale. Ad esempio, il fatto che in genere la terza singolare del presente si ottiene aggiungendo "s" al verbo. La *morfologia* è l'area della linguistica che studia la struttura delle parole.

Per estendere la grammatica al fine di gestire gli aggettivi dobbiamo aggiungere qualche regola:

```
noun_phrase(Num) -->
    determiner(Num), adjectives, noun(Num).
adjectives --> [].
adjectives --> adjective, adjectives.

adjective --> [ferocious].
adjective --> [ugly].
```

Ora una noun phrase consiste di un articolo, alcuni aggettivi e un sostantivo. La categoria adjectives, a sua volta, consiste o di niente, o di un adjective seguito da adjectives. In tal modo possiamo analizzare la frase "The ferocious ugly lion eats Mary" e altre simili.

Notiamo che la regola per la categoria adjectives è *ricorsiva*, e consente di generare un qualsiasi numero di aggettivi.

Pertanto, partendo da una grammatica semplicissima, abbiamo visto come è possibile arricchirla di funzionalità interessanti. Per arrivare a riconoscere una porzione significativa di un linguaggio naturale occorre avere grammatiche con centinaia di regole e un formalismo grammaticale sofisticato.

Abbiamo detto all'inizio che lo scopo dell'analisi sintattica è di determinare la struttura della frase in modo da facilitare la sua successiva comprensione. Vedremo più avanti come ciò può accadere, ora osserviamo che anche la seguente grammatica:

sentence --> det, nounverbdet, noun.

nounverbdet --> noun, verb, det.

genera frasi del tipo visto finora però è certamente peggiore della prima grammatica in quanto raggruppa le parole in modo non significativo.

Ad esempio la frase "The dog ate the lion" ha con questa grammatica il raggruppamento "The (dog ate the) lion" mentre la prima produce il raggruppamento "(The dog) (ate (the lion))" che è senz'altro più significativo in quanto vengono individuati i componenti sintattici della frase.

Questo raggruppamento semplifica il lavori di analisi semantica in quanto possiamo trovare il significato della frase cercando il significato delle sue componenti.

Esempio

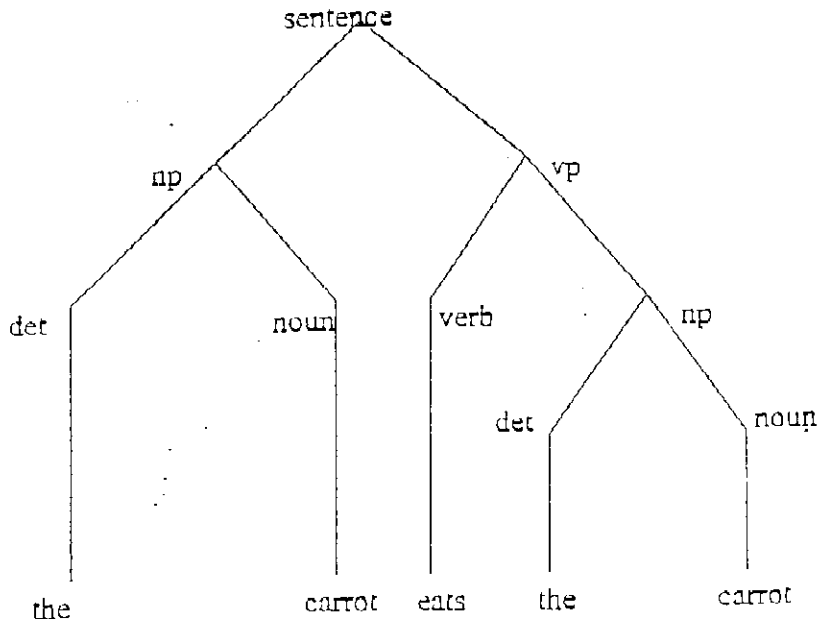
Data la seguente grammatica:

sentence --> noun-phrase (N), verb-phrase (N).
noun-phrase (N) --> det, noun (N).
verb-phrase (N) --> verb (N), noun-phrase (_).
noun (s) --> [carrot].
noun (s) --> [rabbit].
noun (p) --> [rabbits].
verb (s) --> [eats].
det --> [the].

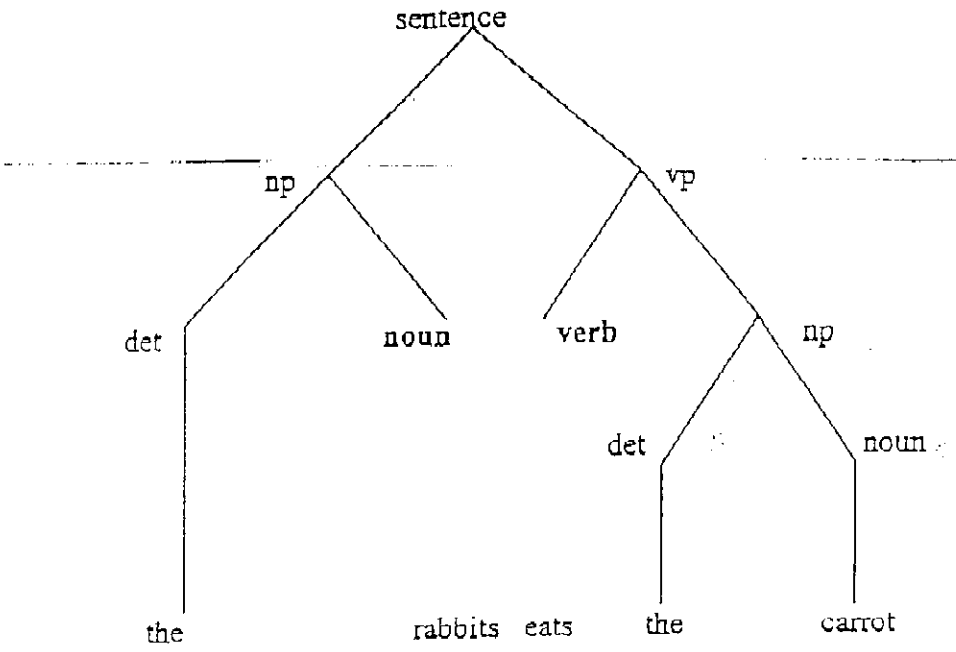
individuiamo quali delle seguenti frasi possono essere riconosciute dalla grammatica e per esse disegnare il parse tree:

a) the carrot eats the carrot
b) the rabbits eats the carrot
c) the rabbit eats carrots

a) the carrot eats the carrot

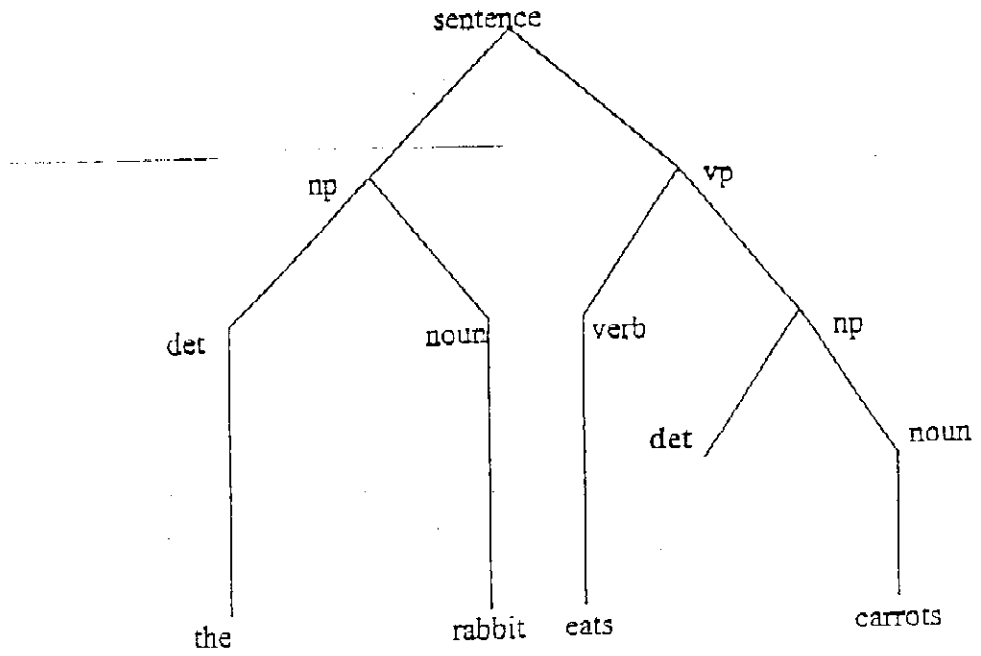


b) the rabbits eats the carrot



Il numero della categoria noun-phrase e quello della categoria verb-phrase devono essere gli stessi: ora rabbits può essere generato da noun (p) mentre eats può essere generato da verb (s) pertanto non c'è un parse tree per la frase che non viene riconosciuta dalla grammatica, cioè non appartiene al linguaggio generato dalla grammatica.

c) the rabbit eats carrots



Anche in questo caso la frase non è generabile a partire dalla categoria sentence: infatti la categoria det può generare solo "the" ma questo articolo non è presente nella frase. Se ci fosse anche la regola

det --> [ ].

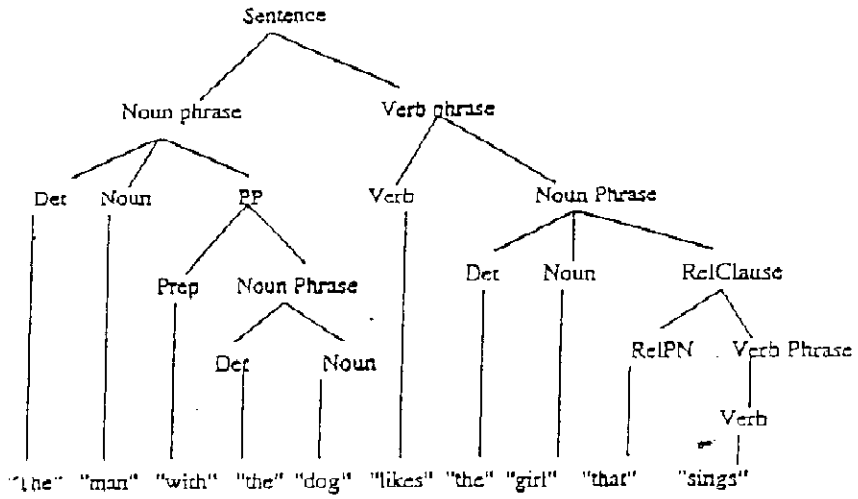
allora avremmo il parse tree per la frase.

Parsing

La grammatica ci consente di generare frasi appartenenti al linguaggio da essa generato, però per analizzare una frase specifica ci occorre un analizzatore (parser) il quale cerchi i possibili modi in cui le regole della grammatica possono essere usate per analizzare la frase.

In generale, vi possono essere molte regole che possono essere applicate per espandere una categoria sintattica e il parser deve decidere quale delle regole applicare. Ad esempio, nella grammatica che gestisce l'accordo del numero soggetto-verbo, la categoria noun-phrase può essere espansa in due modi diversi.

L'analisi della frase può essere fatta in più modi. Un modo molto semplice di operare è fare essenzialmente una DFS attraverso il parse tree, facendo backtracking quando una parola nella frase non fa match con la categoria sintattica attesa. La seguente figura mostra il parse tree di una frase generata da una grammatica più complessa:



Questo modo di operare viene adottato dal linguaggio Prolog e ha il difetto che ci dice solo se la frase è sintatticamente corretta oppure no. Giacchè noi vogliamo giungere al significato della frase ci occorre un metodo di analisi diverso.

Un approccio possibile è combinare analisi sintattica e semantica insieme. L'idea alla base è di ottenere, man mano che procede l'analisi sintattica, il significato della frase combinando insieme i significati delle sue parti componenti.

Analisi semantica

Le altre due fasi di analisi, semantica e pragmatica, cercano di ricavare il *significato* della frase. Nella prima fase (semantica) si ottiene una rappresentazione parziale del significato basandosi sulla struttura sintattica della frase e sui significati delle sue parole componenti. Nella seconda fase (pragmatica) il significato viene completato basandosi sulla conoscenza del contesto e del mondo reale.

Noi accenneremo alla *semantica composizionale* il cui scopo è trovare un modo di comporre il significato dell'intera frase partendo dal significato delle sue parti componenti. Non è sempre possibile fare ciò, ma se lo è semplifica molto la vita. La divisione della frase in parti significative viene effettuata dall'analisi sintattica.

In generale il significato di una frase può essere rappresentato in molti modi, usando i KRL prima visti. Per semplicità noi assumiamo che il significato venga rappresentato usando la logica dei predicati. Questo non va bene per rappresentare in generale le frasi dei linguaggi naturali ma permette di presentare in modo semplice le idee di base.

Nella logica dei predicati rappresentiamo la frase "John loves Mary" così:
loves (john, mary).

Se abbiamo la frase "The man loves Mary" non sappiamo il nome per cui usiamo un simbolo, ad esempio m1, e ricaviamo la semantica
man (m1) $\wedge$ *loves (m1, mary).*

Se abbiamo "A man loves Mary" la semantica è:
$\exists x$ *(man (x)* $\wedge$ *loves (x, mary)).*

Se la frase contiene aggettivi le cose si complicano: "A tall bearded man loves Mary" ha la semantica:
$\exists$ *x (man (x)* $\wedge$ *tall (x)* $\wedge$ *bearded (x)* $\wedge$ *loves (x, mary)).*

Se cerchiamo di rappresentare frasi come quelle presenti in questa pagina le cose si fanno molto complicate e la logica dei predicati si rivela inadeguata.

Ad ogni modo, per illustrare come funziona la semantica composizionale cominciamo con la frase semplicissima "John jumps" e passiamo poi a "John loves Mary".

La semantica composizionale richiede che si possa rappresentare ogni parola nella frase in modo che le parole possano essere combinate in modo da trovare il significato del tutto.

Rappresentare "John" è semplice, basta usare il simbolo *john*. Rappresentare "jumps" è più difficile. Per fare semantica composizionale non è necessario sapere che "jumps" significa saltare.

Noi dobbiamo solo rappresentare in qualche modo che "jumps" è un verbo che necessita di un soggetto in modo tale che quando viene combinato con quest'ultimo otteniamo un'espressione del tipo *jumps (john)* oppure *jumps (mary)*.

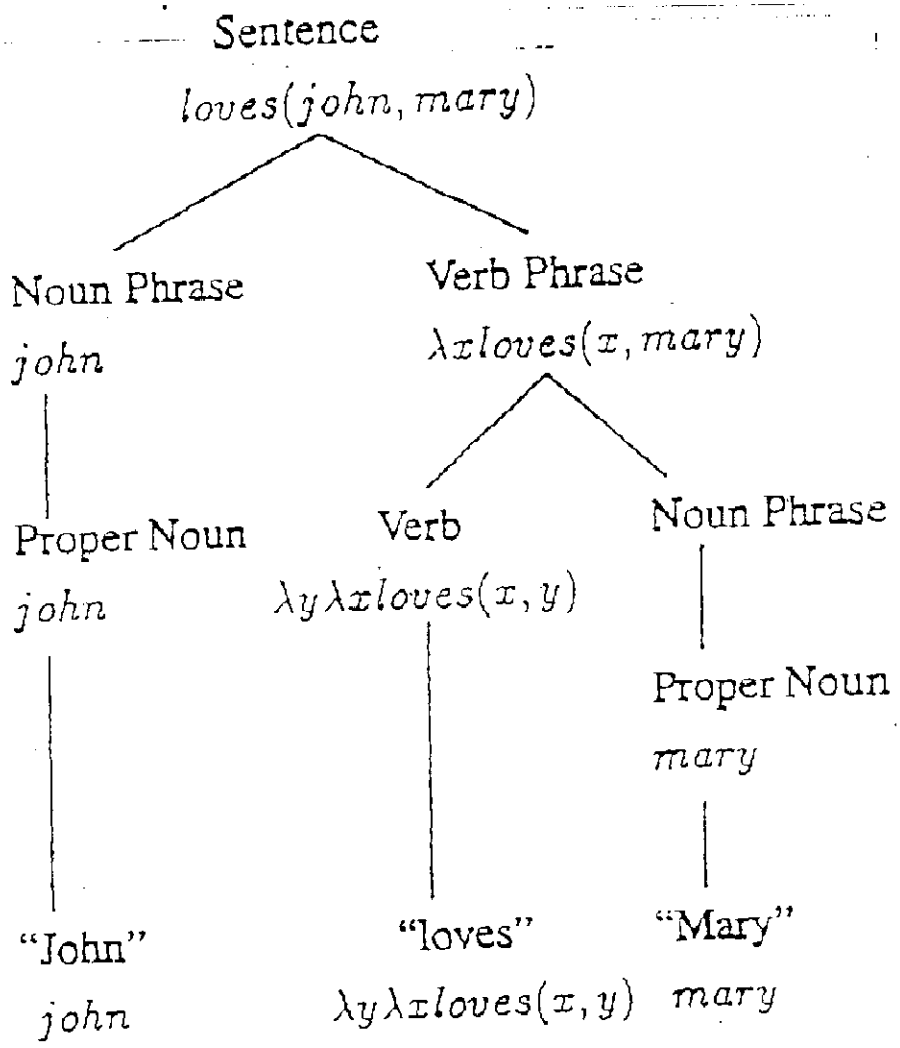
Un modo di fare ciò è rappresentare "jumps" come un predicato con una variabile: *jumps (X)*. Il problema è come combinare *jumps (X)* con *john* in modo da ottenere *jumps (john)*.

Quanto accennato può essere discusso in modo più formale usando la *λ-notation*. Un'espressione come $\lambda\ x.\ jumps(x)$ è utilizzata per rappresentare un'espressione che quando applicata a un argomento (ad esempio, *john*) fa sì che quell'argomento prenda il posto di X, in modo da avere *jumps (john)*.

Applicando $\lambda y.\ \lambda\ x.\ loves\ (x, y)$ a *mary* abbiamo $\lambda x.\ loves\ (x, mary)$.

Applicando questo risultato a *john* si ha *loves (john, mary)*.

La figura che segue illustra come questo si correla alla struttura della frase ottenuta mediante il parsing. Viene mostrato il parse tree per "John loves Mary" integrato dai significati.



Notiamo che il significato per la verb phrase è basato semplicemente sulla combinazione dei significati delle sue parti, mentre il significato dell'intera frase si ottiene combinando il significato della verb phrase con quello della noun phrase.

Pragmatica

Da quanto abbiamo sinora visto possiamo utilizzare il riconoscimento del parlato e l'analisi sintattica e semantica per arrivare, partendo dalla frase pronunciata da qualcuno "John loves Mary", all'espressione nella logica dei predicati *loves (john, mary)*.

Per fare ciò occorre analizzare la struttura dei suoni, la struttura del linguaggio e il significato delle parole componenti la frase. Però per comprendere realmente il linguaggio la conoscenza dei significati delle singole parole non è sufficiente.

E' necessario conoscere anche il contesto in cui è stata pronunciata la frase: dove è stata detta, da chi e perché, e cosa è stato detto in precedenza.

Linguaggio come azione

Per rendersi conto che non è sufficiente fermarsi dopo aver effettuato l'analisi semantica accenniamo al ruolo che il linguaggio ha nella *comunicazione* e nell'*azione*.

Quando parliamo a qualcuno ogni frase ha uno scopo. A volte lo scopo è semplicemente quello di comunicare un fatto. Ad esempio, se diciamo "John loves Mary" forse il solo scopo della frase è che colui al quale è rivolta aggiunga il fatto *loves (john, mary)* alla sua "base di conoscenza" umana. Però spesso abbiamo uno scopo più profondo, come si vede nei seguenti esempi:

1) "Dov'è il caffè ?" Lo scopo immediato è sapere dove si trova, lo scopo sottostante è avere un caffe.

2) "Puoi chiudere la finestra ?" Lo scopo vero è che la persona chiuda la finestra. Letteralmente però la richiesta è solo circa la capacità della persona a farlo.

3) "Sai l'ora ?" Lo scopo vero è sapere che ora è però, a rigore, come prima, la risposta potrebbe essere sì-no.

4) "Quando torni a casa ?" Lo scopo vero dipende dalla persona che parla. Potrebbe essere il coniuge che vuole sapere a che ora preparare la cena o un amico che vuole passare a salutare.

5) "Sei in ritardo" Lo scopo immediato è informare la persona che ha fatto tardi. Lo scopo vero potrebbe essere un rimprovero.

In questi esempi il linguaggio viene usato per ottenere qualcosa. Le frasi possono essere viste come azioni, simili all'atto di aprire una porta. Ad esempio, l'effetto della frase 2 dovrebbe essere identico all'andare personalmente a chiudere la finestra. Il termine *speech act* è usato per riferirsi alle azioni che possono essere effettuate mediante il linguaggio. Tipici speech acts riguardano informazioni, richieste e promesse. Dietro ogni speech act c'è un obiettivo che la persona vuole raggiungere.

Ovviamente quanto riferito al parlato si può trasportare pari pari al caso in cui c'è una persona che scrive e una che legge.

Possiamo cercare di esprimere i possibili obiettivi che giacciono dietro una frase. Denotando con S lo speaker e con H l'hearer, abbiamo:

1. S vuole che H gli porti un caffe.
2. S vuole che H chiuda la finestra.
3. S vuole che H gli dica che ora è.
4. S vuole che H gli dica quando torna a casa per passare a salutarlo
5. S vuole che H sappia che è arrabbiato perché è in ritardo.

L'interlocutore H deve essere in grado di capire l'obiettivo in modo da rispondere appropriatamente. Non è sufficiente sapere il significato letterale della frase, bisogna sapere anche qual è il messaggio sottostante. Quest'analisi degli obiettivi che si nascondono dietro una frase è solo un aspetto della pragmatica.

Idee desunte dalla pianificazione (*planning*) risultano ora utili. Se il linguaggio serve per raggiungere obiettivi, decidere cosa dire è un processo di planning. Estrarre gli obiettivi che stanno dietro a una frase pronunciata, in modo da rispondere appropriatamente, è un problema di riconoscimento di un piano *(plan recognition)* ed è l'opposto del planning.

Infatti mentre nel planning si parte da un obiettivo e si individuano le azioni necessarie per conseguirlo, nel riconoscimento di un piano si parte dalle azioni (nel nostro caso frasi) e si cerca qual è l'obiettivo sottostante.

La conoscenza del contesto è utile in questa analisi. Ad esempio, se la domanda "Sai che ora è ?" è pronunciata quando la persona è arrivata in ritardo a una riunione, la frase è interpretabile come una rimprovero e la risposta giusta non è "Sì". Se invece la domanda è fatta per sapere veramente che ora è, la risposta appropriata è "Sì, sono le cinque".

Gestione dei pronomi

Un problema molto differente si presenta quando dobbiamo analizzare frasi che contengono pronomi. Consideriamo le due frasi:

"Hamish kisses Shona. He loves her"

Chiaramente "he" si riferisce ad Hamish, mentre "her" riguarda Shona. L'interpretazione della seconda frase dovrebbe pertanto essere *loves (hamish, shona)*. Ma per ottenere questa interpretazione dobbiamo necessariamente guardare la prima frase. Cioè la frase precedente fornisce il contesto in cui va interpretata la frase successiva. Gestire correttamente il problema nel nostro esempio è facile e funziona in molti casi ma non sempre le cose sono così semplici. Ad esempio:

"John buys a new telescope. He sees Mary in the distance. He gets out his telescope. He looks at her through it".

Il pronome "her" si riferisce chiaramente a Mary, che però non compare nella frase precedente. Inoltre notiamo che il telescopio di John compare in tre modi diversi: "a new telescope", "his telescope", "it". Capire che il nuovo telescopio che compare nella prima frase è lo stesso che compare nella terza non è banale in quanto richiede la comprensione che comprare qualcosa significa possederla.

Questo problema di scoprire a cosa ci si riferisce riguarda non solo i pronomi. Ad esempio, consideriamo "John vide una Micra blu e una Fiesta rossa dal concessionario. Egli decise di comprare l'auto blu". L'auto blu ovviamente è la Micra e va capito esaminando la prima frase.

Un esempio più significativo può essere dato dalle domande "When is the next flight to Budapest ? Does it have any seats left ?". Il sistema deve stabilire che "it" si riferisce al volo per Budapest e non a Budapest stessa.

Ambiguità nel linguaggio

Uno dei maggiori problemi per la comprensione di un linguaggio è l'*ambiguità*. Intendiamo con ciò che le frasi hanno più di una possibile interpretazione. Le ambigutà possono comparire in tutte le fasi di comprensione del linguaggio. Ad esempio, abbiamo:

Ambiguità nel riconoscimento del parlato: parole differenti sono *omofone*, cioè hanno lo stesso suono (bear-bare, sun-son).

Ambiguità sintattica: vi sono due problemi, il primo riguarda il fatto che una parola può appartenere a più di una categoria sintattica , ad esempio, "bat" può essere sia un nome che un verbo. Il secondo problema nasce dal fatto che una frase può avere più di una struttura. Abbiamo visto prima la frase "John vide Mary con un telescopio".

Ad esempio, nella frase "Fruit flies like a banana" la parola "flies" può essere sia un nome che un verbo. Se la interpretiamo come verbo la frutta comincia a volare come le banane, se come nome abbiamo che alle mosche della frutta piace la banana.

Nella frase "She criticized his apartment so he knocked her flat", "her flat" può essere sia aggettivo possessivo-sostantivo (distrusse il suo appartamento) sia pronome-aggettivo (la buttò a terra).

Ambiguità semantica: molte parole possono avere più di un significato e si parla di ambiguità lessicale. Ad esempio, "bank" può essere la riva di un fiume o una banca.

Ambiguità pragmatica: a volte non è chiaro a quale oggetto si riferisce un pronome e si parla di *ambiguità referenziale* "Chi è l'amico di tuo cugino? E *lui* quanti figli ha ?". Possono altresì essere non chiari gli obiettivi. Abbiamo visto la frase "Sai che ora è ?" la quale può essere un rimprovero o una vera domanda.

A volte le ambiguità che inizialmente si presentano possono essere chiarite in seguito. Ad esempio, supponiamo che il sistema di riconoscimento del parlato ha suggerito che la sequenza di parole può essere una delle due seguenti:

"He read the book"

"He red the book"

La seconda alternativa non ha un valido parsing grammaticale per cui viene scartata in fase di analisi sintattica.

Comunque a volte è necessario fare ricorso alla conoscenza del mondo reale, allo speaker e al contesto per risolvere le ambiguità. Ad esempio l'interpretazione sbagliata della frase "Fruit flies like a banana" può essere scartata se sappiamo che la frutta non ha l'abitudine di volare.

La versione corretta di "bank" nella frase "Richard went to the bank to ask about a mortgage" può essere individuata se sappiamo che le banche si occupano di mutui.

Chiaramente risolvere tutte le ambiguità richiede moltissima conoscenza del mondo reale. Per minimizzare il numero di frasi che sfuggono alla corretta interpretazione occorre approntare una base di conoscenza che include una grande quantità di informazioni sul mondo reale.

Vi sono oggigiorno pochi sistemi che sono in grado di dialogare con un essere umano, interpretando le domande e fornendo risposte.

A tutt'oggi nessun sistema è in grado di interpretare correttamente qualsiasi frase, si stanno però diffondendo sistemi che gestiscono via telefono bollettazione e prenotazioni, chiedendo ogni volta conferma della correttezza dell'interpretazione.

Generazione

Oltre a *comprendere* le frasi pronunciate bisogna anche saper rispondere. Pertanto, partendo da un'idea di ciò che si vuole comunicare si vuole *generare* una sequenza di parole appropriate.

Ad esempio, supponiamo che abbiamo un sistema di interrogazione di una database via telefono. Supponiamo che la risposta da dare all'utente riguardi i voli a basso costo da Londra a Boston in un certo giorno. Il sistema dovrebbe comporre una frase di questo genere:

"Vi sono due voli che potrebbero andar bene. Il primo è con British Airways, costa 330 sterline a parte alle 11am. Il secondo è con American Airlines, costa 340 e parte alle 2pm. Sono possibili altri voli, ma sono tutti più cari."

La generazione del linguaggio naturale può essere decomposta in fasi in qualche modo simili a quella che gestiscono la comprensione delle frasi. Il sistema deve organizzare le informazioni da fornire in modo che le frasi siano chiare e precise.

La prima cosa da decidere è cosa dire. Nell'esempio ci sono da menzionare i due voli economici individuati. Il problema di decidere cosa dire viene indicato come *text planning*. L'idea è di partire con un obiettivo e comporre una serie di frasi che raggiungono l'obiettivo. Nel nostro caso le informazioni essenziali sono la compagnia, il costo e l'orario.

Avendo deciso cosa dire il sistema deve individuare come dirlo cioè quale sequenza di parole comporre. Supponendo che la rappresentazione semantica sia:

$$cost(flight1, 330) \wedge leavetime(flight1, 1100) \wedge airline(flight1, ba)$$

il sistema deve decidere che una frase appropriata è "Il primo è con British Airways, costa 330 sterline a parte alle 11am."

Questo passaggio è complesso. Innanzitutto, va deciso come distribuire l'informazione in frasi separate. Ad esempio, l'informazione contenuta nella seconda e terza frase potrebbe anche essere espressa così:

"Vi è un volo British Airways e uno American Airlines, che partono alle 11am e alle 2pm. Il volo con BA costa 330 e quello con AA costa 340."

Una volta deciso il modo migliore di distribuire l'informazione, ogni frase va composta. La frase deve essere grammaticalmente corretta, usare appropriatamente i pronomi e usare parole di facile comprensione.

In generale questo problema è impegnativo, è chiaro però che se il numero di possibili frasi è limitato è possibile preparare delle frasi tipo che vanno riempite con le specifiche informazioni. Ad esempio, una frase del tipo:

"Vi è un volo con la compagnia X che costa Y e parte a Z".

I metodi usati per la generazione del linguaggio naturale dipendono dal livello di flessibilità richiesto. Se, come nel nostro esempio, abbiamo un'applicazione in cui solo poche frasi vanno composte, si possono usare frasi tipo, altrimenti occorrono soluzioni basate su tecniche più sofisticate.

Infine consideriamo il problema della sintesi del parlato. Bisogna passare dalle parole a un segnale audio. Se vi sono poche frasi di base è possibile registrare le frasi tipo e aggiungere solo i dati specifici. Questa tecnologia viene usata, ad esempio, negli annunci nelle stazioni ferroviarie.

Un approccio più sofisticato registra i fonemi di base delle parole e li compone insieme per formare le parole richieste. La giustapposizione dei fonemi spesso produce suoni poco chiari e sgradevoli. Infatti i fonemi vengono pronunciati in modo leggermente diverso a seconda dei fonemi precedenti e successivi. Inoltre l'intonazione e la temporalizzazione sono importanti (ad esempio, quando si chiede qualcosa si aumenta l'acutezza alla fine della frase, dopo un punto importante in genere si fa una pausa), e questo non può essere determinato guardando solo le parole.

Nonostante le intrinseche difficoltà i progressi che si stanno facendo sono importanti e la qualità del parlato generato migliora continuamente.

Sommario

L'elaborazione del linguaggio naturale richiede sia la *comprensione* della frase pronunciata, tirando fuori il suo significato, sia la *generazione* del parlato a partire da informazioni che vanno espresse a voce.

Il problema della comprensione del linguaggio naturale può essere diviso in riconoscimento del parlato, analisi sintattica, analisi semantica e pragmatica.

Il *riconoscimento del parlato* richiede l'individuazione della sequenza di parole a partire da una segnale audio. Richiede il matching di frammenti del segnale con suoni noti (fonemi) al fine di comporre le parole.

L'*analisi sintattica* richiede l'utilizzo di una *grammatica* del linguaggio per trovare la struttura della frase.

L'*analisi semantica* utilizza la struttura della frase e la conoscenza del significato delle parole per trovare una rappresentazione del significato della frase.

La *pragmatica* si serve del *contesto* per completare il significato della frase, ad esempio, per trovare a quali oggetti si riferiscono i pronomi.

Tutte queste fasi possono avere un output *ambiguo* cioè sono possibili più risultati. Ad esempio, "bank" ha due significati. A volte la conoscenza del mondo reale ci permette di eliminare l'ambiguità.

Anche la generazione del linguaggio naturale può essere divisa in fasi e cioè decidere cosa dire, decidere come dirlo e, se si vuole un segnale audio, elaborare il segnale a partire dalle parole.

8. Machine learning

La capacità di apprendere è certamente una delle caratteristiche fondamentali di un essere intelligente. Un sistema in grado di imparare è più flessibile essendo capace di gestire nuovi problemi e situazioni.

Problemi che concernono l'apprendimento automatico (machine learning) si ritrovano in molte aree dell'AI: nella pianificazione e nella robotica si vuole costruire sistemi che imparino regole di comportamento dall'esperienza ricavata in un certo ambiente; nel linguaggio naturale un sistema può apprendere regole ricavandole da frasi di esempio; nella visione un sistema può imparare a riconoscere un oggetto a partire da alcune immagini; nei sistemi esperti le regole possono essere apprese a partire da casi specifici.

Anche a livello applicativo l'area dell'apprendimento automatico riveste un grande interesse. Ad esempio, è interessante analizzare i dati ricavati dai clienti abituali di un supermercato per ricavare regole da usare in campagne pubblicitarie mirate.

Vi sono diversi tipi di apprendimento, che coinvolgono un diverso impegno da parte del docente e dell'allievo. Ad esempio, un insegnante può dire una data importante che bisogna ricordare, può fornire degli esempi tipo "Fido e Rover sono dei cani", presentare un'analogia "la corrente nei fili scorre come l'acqua nelle tubature".

Finora gran parte della ricerca sull'apprendimento automatico si è concentrata sull'*apprendimento induttivo*, in cui il punto di partenza sono *esempi*.

L'apprendimento può riguardare categorie concettuali (il concetto di "cane" partendo da esempi di cane), regole per predire il tempo, regole per diagnosticare una malattia, e così via. In tutti questi casi vengono forniti degli esempi e il sistema cerca di trovare regole o descrizioni generali a partire dagli esempi. Gli esempi possono essere sia *positivi* (Fido è un cane) che *negativi* (Carlo non lo è).

In generale, l'apprendimento induttivo è usato per addestrare un sistema a risolvere *problemi di classificazione*. In un problema di classificazione vi sono delle *caratteristiche input* che influenzano il risultato, e un insieme di possibili *categorie output*. Ad esempio, la diagnosi medica è un problema di classificazione, in cui le caratteristiche input sono i sintomi del paziente, e le categorie output sono le possibili diagnosi.

Nel riconoscimento automatico di caratteri le caratteristiche input sono le specificità del carattere (ad esempio, un trattino in basso a sinistra), e le categorie output i caratteri possibili (ad esempio, la lettera "p").

Le tecniche di apprendimento induttivo vengono usate per costruire sistemi che, avendo in input le caratteristiche, generano la corretta classificazione.

Noi nel seguito accenneremo a due importanti tecniche. La prima consiste nell'utilizzo di *algoritmi genetici*. Tali algoritmi si basano sul concetto che buone soluzioni possono "evolvere" in una popolazione combinando possibili soluzioni per ottenere la "prole" ed eliminando le soluzioni poco valide.

La seconda tecnica prevede l'utilizzo di *reti neurali*. Tali reti, che ricordano molto alla lontana la struttura del cervello, sono oggi largamente diffuse, al pari degli algoritmi genetici, in molte aree applicative.

Le applicazioni reali di *machine learning* richiedono generalmente migliaia di esempi per giungere ad un apprendimento significativo. Ad esempio, per apprendere le regole per diagnosticare una specifica malattia, a partire dal sintomo "dolori allo stomaco", i dati su moltissimi pazienti sono necessari insieme ai sintomi addizionali dei malati e le diagnosi finali dei medici.

Al fine di chiarire il modo di operare delle due tecniche considereremo un semplice problema e un piccolo numero di esempi.

Esempio

Supponiamo di avere alcune informazioni sulle caratteristiche di alcuni studenti e sul loro comportamento l'anno passato, vogliamo individuare una regola che ci dica se è probabile che gli studenti quest'anno abbiano una borsa di studio.

Assumiamo che i dati sono i seguenti:

- se l'anno passato hanno avuto la borsa
- se studiano molto
- maschio - femmina
- se sono soliti bere molto

Inoltre sappiamo anche se quest'anno hanno la borsa di studio.

Coloro che quest'anno hanno la borsa sono esempi positivi, mentre gli altri sono esempi negativi.

La seguente figura mostra i dati a nostra disposizione:

Student	First last year?	Male?	Works hard?	Drinks?	First this year?
Richard	yes	yes	no	yes	no
Alan	yes	yes	yes	no	yes
Alison	no	no	yes	no	no
Jeff	no	yes	no	yes	no
Gail	yes	no	yes	yes	yes
Simon	no	yes	yes	yes	no

Si nota subito che i due studenti che hanno la borsa quest'anno (Alan e Gail) entrambi l'hanno avuta l'anno scorso e studiano molto. Inoltre nessuno degli altri ha entrambe queste caratteristiche. Pertanto viene fuori subito una regola del tipo:

IF borsa l'anno scorso AND studiare molto THEN borsa quest'anno

Osserviamo però che dai dati disponibili è possibile estrarre anche un'altra regola:

IF maschio AND non bere molto OR femmina AND bere molto THEN borsa quest'anno

La prima regola appare ragionevole mentre la seconda è abbastanza strana. Notiamo che le due regole forniscono previsioni diverse per altri studenti. Supponiamo che Robert l'anno scorso ha avuto la borsa, studia molto ma beve. Per la prima regola dovrebbe avere la borsa anche quest'anno, ma per la seconda no.

In genere la regola più semplice è quella che fornisce predizioni migliori in quanto tiene conto di caratteristiche generali (studiare molto) invece di caratteristiche particolari (bere o non bere).

In questo esempio estremamente semplificato abbiamo solo quattro caratteristiche. Ovviamente altre informazioni sono certamente rilevanti: ad esempio, le materie di studio e i voti riportati finora nell'anno. Si potrebbe anche, ad esempio, raffinare la caratteristica "beve molto" dando invece dei valori sì-no, i valori "troppo", "moderato", "occasionale", "mai". E' evidente che la scelta delle caratteristiche e dei loro valori è molto importante se si vogliono risultati soddisfacenti.

Fatte queste premesse, l'esempio comunque è utile per delineare i possibili approcci al problema dell'apprendimento induttivo. Poichè abbiamo quattro caratteristiche ciascuna con valori sì-no, vi sono 16 diversi tipi di studente. Nel seguito denotiamo i valori di uno specifico studente con sequenze di T e F (vero-falso). Pertanto i valori di Richard sono TTFT.

Nel seguito vedremo come regole del tipo di quelle ora desunte intuitivamente possono essere dedotte automaticamente a partire dagli esempi, utilizzando gli algoritmi genetici o le reti neurali.

Gli algoritmi genetici

Un *algoritmo genetico* può essere visto come una particolare tecnica di ricerca. Tale tecnica può essere applicata non solo a problemi di machine learning ma anche a un'ampia classe di problemi di ricerca.

Ad esempio, gli algoritmi genetici sono stati applicati con successo a problemi di organizzazione di orari, in cui bisogna organizzare eventi con certi vincoli (ad esempio, distribuire le lezioni nelle aule tenendo conto delle preferenze e senza che vi siano sovrapposizioni). La costruzione di orari è un problema in cui vi sono moltissime possibili soluzioni, ed è difficile individuare la migliore.

Il machine learning pone problemi analoghi. Infatti vi possono essere moltissime possibile regole che potrebbero combaciare con i dati forniti, e nasce il problema di individuare le regole migliori.

L'idea alla base degli algoritmi genetici è partire da una popolazione di *genomi* che rappresentano le possibili soluzioni, mediante *mutazione* e *combinazione* dei genomi ricavare nuovi genomi e valutare le prestazioni di tale *prole* utilizzando opportune funzioni. La prole che riceve il maggiore punteggio sopravvive e si accoppia di nuovo per generare nuova prole.

Al di là dei termini suggestivi che richiamano le teorie dell'evoluzione, quanto abbiamo detto non è molto differente dagli algoritmi euristici di ricerca prima visti. Infatti, ad esempio, nell'algoritmo best first abbiamo dei nodi successore (che possono essere possibili soluzioni di un problema) ai quali si giunge a partire da nodi "padre". I nodi vengono valutati e il migliore viene utilizzato per derivare nuovi nodi successore.

La differenza principale con gli algoritmi genetici è che ora nuovi successori sono ricavati a partire da due genitori e non uno solo. In presenza di circostanze favorevoli, se vengono utilizzati due genitori con alto punteggio, il "figlio" può risultare veramente buono, e l'algoritmo converge rapidamente verso una eccellente soluzione.

Volendo precisare meglio il modo di operare degli algoritmi genetici, dato l'insieme S di possibili soluzioni, un algoritmo genetico procede come segue:

Finchè S non contiene una soluzione valida:

- assegna un punteggio alle soluzioni in S

- scegli coppie di soluzioni in S, favorendo coppie con alto punteggio

- genera la prole combinando e/o mutando tali soluzioni

- sostituisci le soluzioni più deboli con la nuova prole

In genere, le soluzioni iniziali sono scelte a caso. Le coppie sono selezionate usando una funzione semi-casuale che favorisce le coppie con alto punteggio, ma non può trascurare la possibilità che l'accoppiamento con una soluzione debole generi un buon figlio.

Al pari di tutti gli algoritmi euristici di ricerca, gli algoritmi genetici devono valutare le soluzioni candidate. Nel caso che ci interessa, l'apprendimento induttivo, una semplice funzione di valutazione può essere la percentuale di esempi che sono correttamente classificati dalla regola in esame.

Ad esempio, nel problema studenti, la regola "se hai avuto la borsa l'anno scorso, l'avrai anche quest'anno" funziona correttamente per cinque dei sei studenti. Solo Richard rappresenta un'eccezione. Pertanto il punteggio per questa regola è 5/6.

Dobbiamo anche individuare una rappresentazione delle soluzioni che si presti alle operazioni di mutazione e incrocio. Possiamo usare delle stringhe del tipo "T # T F", che rappresenta la regola "avrai la borsa se l'hai avuto l'anno scorso, lavori sodo e non bevi molto". Il carattere "#" viene utilizzato per indicare che non importa che tu sia maschio o femmina.

Questa semplice rappresentazione va bene per rappresentare qualsiasi formula che coinvolge congiunzioni di condizioni. Non va bene se ci sono disgiunzioni (regole del tipo "avrai la borsa se l'hai avuta l'anno scorso *oppure* lavori sodo").

Abbiamo perciò individuato un modo di rappresentare le possibili soluzioni, e un modo di valutare la bontà delle stesse. Dobbiamo solo disporre di metodi per generare soluzioni "figlio". Questo può essere fatto utilizzando due operatori genetici: *incrocio* e *mutazione*.

L'operatore *incrocio* ha in input due soluzioni e mescola i loro elementi per generare due nuovi candidati. La forma più semplice di questo operatore divide ogni soluzione in due e forma due nuove soluzioni combinando la prima metà di una con la seconda metà dell'altra.

Ad esempio, la soluzione:

T # F # incrociata con:

F # T F genera:

T # T F e:

F # F #

In sostanza, abbiamo accoppiato la categoria degli studenti (borsa, pigri) con la categoria degli studenti (no borsa, studiano, no bere). Come risultato abbiamo ottenuto la categoria (borsa, studiano, no bere) e la categoria (no borsa, pigri).

In generale, se un genitore eccelle nella prima metà della stringa e l'altro genitore nella seconda metà, allora uno dei figli è migliore di entrambi genitori e l'altro figlio peggiore di entrambi. Il secondo figlio viene in genere eliminato ripetendo il ciclo.

L'altro operatore genetico, la *mutazione*, agisce su un singolo candidato e cambia a caso alcuni suoi caratteri. Utilizzare il solo operatore mutazione non è opportuno in quanto cancella alcune caratteristiche dei candidati, ma si rivela utile nel caso che la popolazione iniziale non includa alcune caratteristiche importanti della soluzione.

Supponiamo di avere la seguente popolazione iniziale di soluzioni:

S = {#T##, F#TF, TT##, #TT#, ###F}

Il primo elemento di S corrisponde alla regola "borsa quest'anno se maschio", il secondo alla regola "borsa quest'anno se non l'hai avuto l'anno scorso ma studi molto e non bevi", e così via.

Ognuna di queste stringhe può essere valutata calcolando la percentuale di esempi correttamente classificati.

La prima soluzione # T # # predice che tutti gli uomini avranno la borsa. Vediamo però che la regola fornisce solo due predizioni giuste (Alan che ottiene la borsa e Alison che non la ottiene) per cui il punteggio della stringa è 2/6.

La seconda stringa F # T F, classifica correttamente Richard, Jeff e Simon e pertanto riceve il punteggio 3/6.

La terza soluzione T T # # è verificata per Alan, Alison, Jeff e Simon e ha il punteggio 4/6 .

La quarta stringa # T T # va bene per Richard, Alan, Alison e Jeff e riceve il punteggio 4/6.

La quinta soluzione # # # F classifica correttamente Alan, Jeff e Simon e ha punteggio 3/6.

Il migliore punteggio (4/6) è ottenuto dalla terza e quarta soluzione per cui sono queste le stringhe cui applicare l'operatore incrocio. La soluzione:

T T # # incrociata con:

T T # genera:

T T T # (punteggio 5/6) e:

T # # (punteggio 2/6)

Queste due nuove soluzioni prendono il posto della stringa # T # # (punteggio 2/6) e di una delle due stringhe 3/6, diciamo la # # # F.
Pertanto l'insieme S diventa:

S = { T T T #, # T # #, F # T F, T T # #, # # T # }

Ora si potrebbero prendere le soluzioni migliori, incrociarle e andare avanti così senza però giungere alla soluzione ottimale T # T #. In effetti stiamo usando la versione più semplice dell'operatore incrocio, esistono varianti più complicate in cui vengono incrociate porzioni arbitrarie dei genitori.

Esiste però anche l'operatore mutazione che può essere utile per far comparire una caratteristica importante non presente negli esempi. Supponiamo che la mutazione venga applicata alla migliore soluzione finora individuata (T T T #). Sono tante le mutazioni possibili, effettuandole casualmente verrà fuori anche la soluzione ottimale T # T # (punteggio 6/6).

L'esempio visto è estremamente semplificato, in problemi reali migliaia di cicli possono essere necessari per individuare la migliore soluzione. Ad ogni modo questa tecnica spesso risulta migliore rispetto ai tradizionali algoritmi di ricerca.

La caratteristica interessante degli algoritmi genetici è che incrociando due buone soluzioni vi è un'alta probabilità che almeno uno dei figli sia migliore di entrambi i genitori.

Le reti neurali

Le *reti neurali* rappresentano un approccio al machine learning molto diverso dagli algoritmi genetici. Un rete neurale è costituita da molti elementi di elaborazione (*neuroni*) connessi insieme. Il comportamento di ciascun neurone è semplicissimo, ma la rete nel suo insieme può avere un comportamento sofisticato e svolgere compiti complessi. Vi sono molti tipi di reti neurali, noi ci limitiamo ad accennare ai cosiddetti *perceptron*.

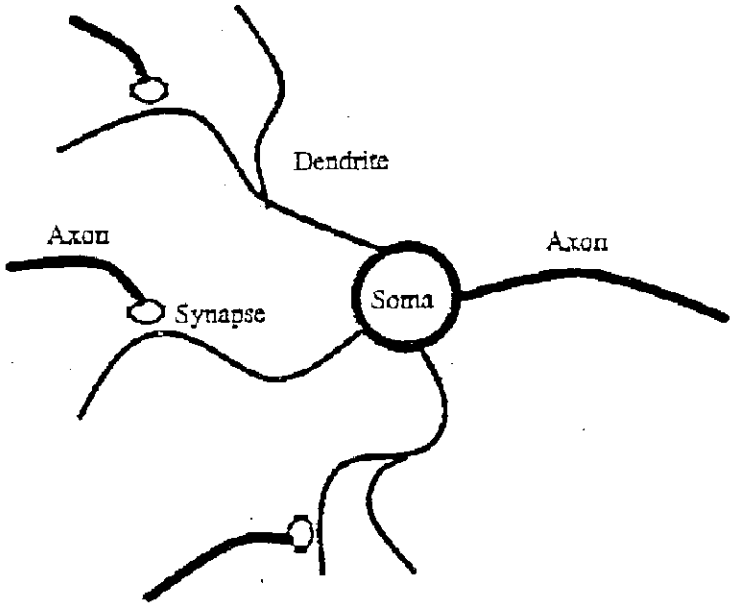
In tale tipo di reti il comportamento della rete dipende dai *pesi* presenti sulle connessioni tra neuroni. Questi pesi possono essere "appresi" a partire dagli esempi forniti.

Le reti neurali sono, al pari degli algoritmi genetici, ispirati dalla biologia, pertanto è utile accennare alle caratteristiche dei neuroni biologici per comprendere le differenze con i neuroni di cui parlano le reti neurali.

I neuroni biologici

Il cervello umano consiste di circa 100 miliardi di neuroni. Ogni neurone è connesso a migliaia di altri neuroni. Il modo di operare di un neurone è molto complicato e ancora non del tutto chiaro, ma sinteticamente può essere descritto come segue: un neurone riceve inputs dai suoi vicini, e se sufficienti inputs sono presenti contemporaneamente il neurone viene attivato e *"spara"*, emettendo un output che viene ricevuto dai neuroni ad esso connessi.

La seguente figura mostra le caratteristiche essenziali di un neurone:



Il *soma* è il corpo della cellula, i *dendriti* sono filamenti tramite i quali arrivano gli inputs alla cellula, l'*assone* manda i segnali output, le *sinapsi* sono connessioni specializzate che possono essere potenziate o indebolite per far passare i segnali. In funzione dei segnali ricevuti un neurone può essere in uno stato di *eccitazione* o di *inibizione*. Se eccitato manda il segnale output mediante l'assone.

Il perceptron semplice

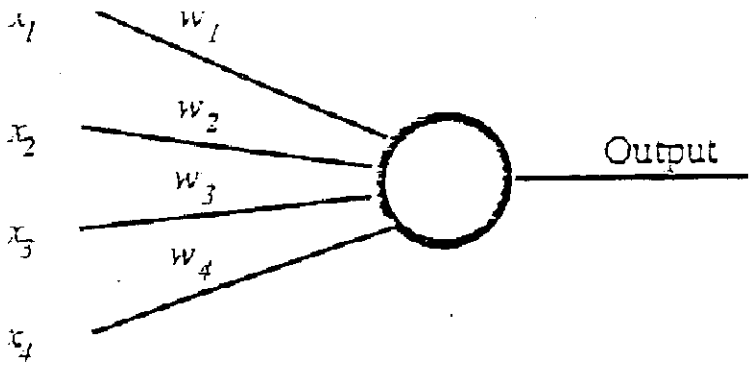
Un semplice neurone artificiale può essere facilmente realizzato. Esso riceve un certo numero di inputs, utilizza dei pesi per simulare il comportamento delle sinapsi, somma i valori pesati, e spara se la somma supera una certa soglia.

Più precisamente, se vi sono n inputs ciascuno con un peso associato, il neurone calcola la somma pesata degli inputs e manda in output 1 se la somma supera la soglia t, 0 altrimenti. Se gli inputs sono $x_1, x_2, \ldots, x_n$ con pesi $w_1, w_2, \ldots, w_n$:

se $w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > t$ output 1

altrimenti output 0

La seguente figura visualizza il comportamento di questo semplice perceptron:



Una rete neurale realistica include centinaia o migliaia di neuroni. Però anche disponendo di un solo neurone, è possibile mostrare come avviene l'apprendimento in una rete.

Apprendere, nelle reti neurali, implica partire dagli esempi e modificare opportunamente i pesi. Ogni esempio avrà valori input-output specificati. Gli esempi sono esaminati uno alla volta, e i pesi aggiustati se la rete fornisce valori incorretti.

Ciò viene realizzato aumentando i pesi sulle connessioni attive (quelle con input 1) se l'output della rete è 0 mentre l'esempio ci dice che il valore è 1. Viceversa, i pesi vengono decrementati se il valore output è 1 mentre dovrebbe essere 0.

L'intero insieme di esempi deve essere preso in esame più volte finchè la rete fornisce i risultati corretti per il massimo numero di esempi.

Per comprendere il comportamento dell'algoritmo di apprendimento del perceptron, è utile fare un esempio.

Esempio

Consideriamo i nostri studenti, ogni loro caratteristica può essere rappresentata da un input: ad esempio, $x_1 = 1$ se lo studente ha avuto la borsa l'anno scorso, $x_2 = 0$ se è una ragazza, e così via.

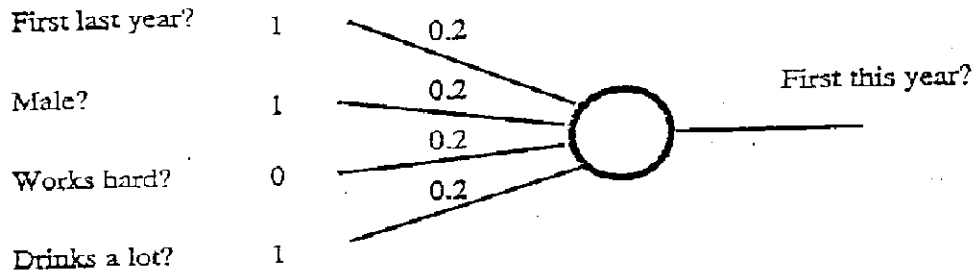
L'output vale 1 se lo studente ottiene la borsa quest'anno.

Inizialmente i pesi assumono un valore casuale piccolo, supponiamo che tale valore sia $w_i = 0.2$ per tutti i pesi.

Inoltre supponiamo che il valore della soglia sia $t = 0.5$.

Infine il valore col quale modifichiamo ogni volta i pesi sia $d = 0.05$.

Allora i dati relativi al primo studente (Richard) portano alla seguente rappresentazione:



L'output della rete (costituita da un solo neurone) è 1, in quanto la somma pesata degli input è:

$0.2 \times 1 + 0.2 \times 1 + 0.2 \times 0 + 0.2 \times 1 = 0.6 > t$

Ma sappiamo che Richard non ha avuto la borsa quest'anno, per cui l'output dovrebbe essere zero. Quindi tutti i pesi sulle connessioni attive vanno decrementati di d. I nuovi pesi sono:

0.15, 0.15, 0.2, 0.15

Il prossimo esempio riguarda Alan. I suoi inputs sono 1, 1, 1, 0. L'output della rete è 0 poichè la somma è:

$0.15 \times 1 + 0.15 \times 1 + 0.2 \times 1 + 0.15 \times 0 = 0.5 = t$

Alan però ha avuto la borsa quest'anno per cui l'output dovrebbe essere 1. Aumentiamo i pesi per gli inputs che valgono 1 e otteniamo i nuovi pesi:

0.2, 0.2, 0.25, 0.15

Nel caso di Alison abbiamo che la somma vale

$0.2 \times 0 + 0.2 \times 0 + 0.25 \times 1 + 0.15 \times 0 = 0.25 < t$

per cui l'output è 0 e va bene in quanto Alison non ha avuto la borsa quest'anno.

Per Jeff la somma vale:

$0.2 \times 0 + 0.2 \times 1 + 0.25 \times 0 + 0.15 \times 1 = 0.35$ output 0 e anche è ok.

Per Gail la somma vale 0.6, output 1 e va bene.

Nel caso di Simon la somma vale $0.6 > t$ per cui l'output è 1 mentre dovrebbe essere 0 per cui decrementiamo i pesi di d e otteniamo i nuovi pesi:

0.2, 0.15, 0.2, 0.1

E' terminato il primo ciclo di apprendimento, ora tutti gli esempi vanno considerati di nuovo finchè la rete dà in output il massimo numero di risultati giusti.

Il secondo ciclo può essere così schematizzato:

Richard	0.2 + 0.15 + 0.1 = 0.45	OK
Alan	0.2 + 0.15 + 0.2 = 0.55	OK
Alison	0.2	OK
Jeff	0.15 + 0.1 = 0.25	OK

Gail 0.2 + 0.2 + 0.1 = 0.5 l'output è 0 mentre dovrebbe essere 1 per cui calcoliamo i nuovi pesi:

0.25, 0.15, 0.25, 0.15

Simon 0.15 + 0.25 + 0.15 = 0.55 l'output è 1 mentre dovrebbe essere 0 per cui decrementiamo i pesi e otteniamo:

0.25, 0.1, 0.2, 0.1

Il terzo ciclo ci porta alla soluzione ottimale:

Richard	0.25 + 0.1 + 0.1 = 0.45	OK
Alan	0.25 + 0.1 + 0.2 = 0.55	OK
Alison	0.2	OK
Jeff	0.1 + 0.1 = 0.2	OK
Gail	0.25 + 0.2 + 0.1 = 0.55	OK
Simon	0.1 + 0.2 + 0.1 = 0.4	OK

Poichè ora coincidono gli output calcolati con quelli attesi l'algoritmo si ferma. Possiamo dire che l'apprendimento dei pesi da parte della rete è terminato.

Se ora si presenta un nuovo studente ci serviamo dei pesi finali (0.25, 0.1, 0.2, 0.1) per prevedere se quest'anno avrà o meno la borsa. Ad esempio, se Tim l'ha avuta lo scorso anno, studia molto e beve, abbiamo:

$$0.25 + 0.1 + 0.2 = 0.55 \geq t$$

per cui l'output è 1 e prevediamo che quest'anno avrà la borsa.

In questo esempio possiamo attribuire un significato ai pesi finali: aver avuto la borsa l'anno passato e lavorare sodo "pesano" giustamente più del sesso e del bere. Per reti più complesse diventa molto difficile attribuire un significato ai pesi, l'unica cosa che sappiamo è che gli esempi vengono correttamente classificati oppure no.

A questo punto possiamo formulare l'algoritmo di base per l'apprendimento del perceptron:

- Inizializzare casualmente i pesi

- Ripetere

    - per ogni esempio

        - se l'output della rete è 1 e il valore atteso è 0 decrementare di d i pesi per le connessioni attive

        - se l'output della rete è 0 e il valore atteso è 1 incrementare di d i pesi per le connessioni attive

finchè la rete fornisce gli outputs corretti per il massimo numero di esempi.

Occorre dire che, sebbene quest'algoritmo abbia lavorato bene per gli studenti, vi sono insiemi di esempi per i quali non esiste alcun insieme di pesi in grado di fornire il comportamento corretto.

Un esempio classico è dato dalla funzione *or esclusivo*, che fornisce 1 solo se uno solo degli input è 1, altrimenti 0. Si dimostra che non è possibile trovare un insieme di pesi per un perceptron tale che questa funzione sia realizzata.

Per risolvere problemi di questo genere sono stati introdotti perceptron più complessi privi dei limiti delle versioni precedenti e vi è oggigiorno un notevole interesse per le applicazioni delle reti neurali nel machine learning.

Noi abbiamo ora visto cosa si può fare con un singolo neurone, usando il metodo più semplice per calcolare l'output. Questo modo di operare va bene a fini esemplificativi per illustrare le idee di base, ma per problemi più impegnativi dobbiamo fare intervenire una rete interconnessa di neuroni, accrescendo in tal modo il potere di apprendimento del sistema.