**Riassunti di**

# Gestione della conoscenza e intelligenza artificiale

## Politecnico di Torino
## Specializzazione in ing. informatica
## Anno 2005

**Diego Regis**
**Marco Scavarda**
**Marco Vallini**

# Rappresentazione di un problema

## Cos'è l'Intelligenza Artificiale

L'IA nasce negli anni 50-60 e viene utilizzata per:
- strategie di ricerca soluzione dei problemi
- dimostrazioni automatiche di teoremi
- giochi
- gestione della conoscenza: comprensione del linguaggio visivo, scritto e parlato

Permette quindi la realizzazione di **Sistemi Esperti**, sistemi in grado di dimostrare una competenza confrontabile con quella di un esperto umano in un campo specialistico (medicina,ingegneria…).
Altri campi applicativi: processi di produzione, automazioni di uffici, basi di dati, traduzioni automatiche.

Nell'informatica tradizionale si affronta la risoluzione di un problema tramite il meccanismo **Data Flow**, dove sono presenti dei dati che devono essere manipolati con delle operazioni di base secondo una certa logica (DATI - OPERAZIONI - CONTROLLO).
Nell'IA si può procedere secondo un **meccanismo a stati**, dove vengono rappresentate le configurazioni possibili e ci si sposta all'interno di esse secondo un controllo sequenziale e deterministico (successione di stati).

## Rappresentazione del problema nello spazio degli stati

È un grafo che presenta come nodi i possibili stati e come archi orientati le operazioni applicabili allo stato.
- Lo **stato iniziale** è il nodo di partenza del grafo (situazione iniziale)
  - Può esistere anche più di uno stato iniziale
- Lo **stato finale** è il nodo di arrivo del grafo (che coincide con l'obiettivo (goal) del problema)
  - Può esistere anche più di uno stato finale
- Il percorso tra lo stato iniziale e finale rappresenta la possibile soluzione

Gli **operatori** permettono di generare da uno stato di partenza, uno o più stati successivi (tramite una computazione).
La descrizione degli operatori può avvenire tramite tabelle. Nel caso delle stringhe, come convenzione, si adottano le produzioni grammaticali (A genera B).

La rappresentazione degli stati può essere realizzata **mediante alberi** (o grafi) e **mediante stringhe** (testo).
Nell'ultimo caso si adotta un linguaggio con notazioni pre-,in-,post- fisse (soluzione tramite regole di riscrittura).

## Descrizione formale di un problema

1. Definire uno spazio di stati (tutti gli stati, solo quelli realmente raggiungibili ecc…)
   a. S
2. Specificare gli stati iniziali
   a. $i \in S$
3. Specificare gli stati finali (goal)
   a. $g \in S$
4. Specificare un insieme di regole che descrivono le azioni (operatori)
   a. $O : S \to S$
5. La soluzione è la sequenza di operatori $o_1, o_2, \ldots, o_n$ tale che:
   a. $g = o_n( \ \ldots \ o_2(o_1(i)) \ \ldots \ )$

**Un approccio di IA** è differente in termini di tempo, risorse e metodologia nei confronti di quello per algoritmo.
(Esempio) TRIS
1. Si individua uno stato di partenza (quadrato 3x3 libero)
2. Si individua una regola per generare da uno stato tutti i possibili stati (inserimento di un valore O oppure X)
3. Si individua un metodo di valutazione di ciascuna delle possibili mosse
4. Si individua una strategia di gioco (tener conto di una o più mosse successive, anche dell'avversario)

Un altro esempio è il puzzle dell'8 con al massimo 4 possibili stati ad ogni mossa (SU,GIU', SINISTRA DESTRA).

## Grafi

Un grafo è un insieme di nodi

- Se un arco parte dal nodo *n* al nodo *m*
  - *m* è detto **successore** di *n*
  - *n* è detto **genitore** di *m*
- Se un nodo non ha genitori è detto **radice**
- Se un nodo non ha successori è detto **foglia**
- Un **albero** è un caso particolare di grafo in cui ogni nodo ha al massimo un genitore
- Un **cammino** è una sequenza di nodi $n_1, n_2, \ldots, n_k$ tale che ogni $n_i$ è successore di $n_{i-1}$
- Se fra i nodi *n* e *m* c'è un cammino
  - *m* è un **discendente** di *n*
  - *n* è un **antenato** di *m*
- Problema: trovare un cammino tra il nodo iniziale e un nodo appartenente ai nodi goal.

Assegnazione di un grafo in modo <u>implicito</u>: si assegna un insieme di nodi $\{s_i\}$ di partenza e un operatore di successione $\Gamma$ che, applicato ad un nodo, fornisce tutti i successori.

| | Vantaggi | Svantaggi |
|---|---|---|
| **Alberi** | * Procedure più semplici | * Lo stesso nodo può essere generato più volte |
| **Grafi** | * Nodi non duplicati | * Procedure più complesse<br>* Possibilità di cicli<br>* Costo elevato nello stabilire se un nodo è già stato generato |

## Scelta di una buona rappresentazione

La scelta della rappresentazione influenza lo sforzo di ricerca (hindsight).
In generale è preferibile una rappresentazione con piccoli spazi degli stati, ricorrendo a variabili di abbreviazione o valori numerici che identificano gli stati stessi (il tutto sotto forma di vettore di elementi, vedi Torre di Hanoi).

(Esempio) Problema della scimmia e delle banane
Una buona rappresentazione può essere la seguente:

$$
\begin{aligned}
(\mathbf{w}, 0, \mathbf{y}, 0) &\xrightarrow{goto(u)} (\mathbf{u}, 0, \mathbf{y}, 0) \\
(\mathbf{w}, 0, \mathbf{w}, 0) &\xrightarrow{pushbox(v)} (\mathbf{v}, 0, \mathbf{v}, 0) \\
(\mathbf{w}, 0, \mathbf{w}, 0) &\xrightarrow{climbox} (\mathbf{w}, 1, \mathbf{w}, 0) \\
(\mathbf{c}, 1, \mathbf{c}, 0) &\xrightarrow{grasp} (\mathbf{c}, 1, \mathbf{c}, 1)
\end{aligned}
$$

Dove w e y sono le posizioni iniziali di scimmia e banane, il 1° flag indica se la scimmia è sulla cassa, il 2° se ha preso la banana.

## Strategia di controllo

Una strategia di controllo deve <u>causare movimento</u>, cioè deve portare verso la soluzione e non fare eseguire sempre gli stessi passi, con la necessità di rilevare le situazioni critiche.
Inoltre deve <u>essere sistematica</u>, possedere quindi un *movimento globale* (nel corso di vari passi), ed un *movimento locale* (nel corso di un singolo passo). → Ricerca in ampiezza e Ricerca in profondità.

Ragionamento in avanti (**forward**): si procede dagli stati iniziali verso le mete
1. Si costruisce l'albero la cui radice è lo stato iniziale
2. Si cerca la regola la cui parte sinistra corrisponde al nodo e si generano i nodi corrispondenti alla parte destra

Ragionamento all'indietro (**backward**): si procede dalle mete verso gli stati iniziali
1. Si costruisce l'albero la cui radice è lo stato meta
2. Si cerca la regola la cui parte destra corrisponde al nodo e si generano i nodi corrispondenti alla parte sinistra

Per decidere se è meglio adottare un ragionamento in avanti o all'indietro:
- Si controlla quale stato (finale o iniziale) è più facile da verificare
- Qual è la direzione di maggior fattore di ramificazione

È necessario
- Stabilire l'ordine con cui applicare le operazioni
- Stabilire il tipo di ricerca (sequenziale / parallela)
- Stabilire il **regime di controllo**:
  - Irrevocabile
    - Decidere una regola senza poter revocare la decisione presa
    - (Esempio) negli Scacchi si studia e si effettua una mossa…ci si accorge successivamente che la mossa era dannosa, ma non si può tornare indietro nel gioco, è richiesto un maggior studio prima di fare la mossa
  - Backtracing
    - Avere l'opportunità di tornare indietro sul grafo nel caso in cui ci si accorge di aver sbagliato
    - (Esempio) Gioco dell'Otto
  - Ricerca su grafi
    - Ricercare la soluzione sull'intero grafo (molto costosa)

## Riduzione di problemi

La soluzione di un problema viene ridotta alla soluzione di uno o più sottoproblemi più semplici (Esempio: Torre di Hanoi)

1. Insieme dei problemi
   a. P
2. Insieme degli operatori che specificano come risolvere un problema dati n problemi risolti
   a. $O : pn \rightarrow P$
3. Problema da risolvere (goal)
   b. $g \in P$
4. Insieme dei problemi risolti
   a. $SP \subset P$
5. La soluzione è un insieme di operatori che, applicati ai problemi risolti, portino ad ottenere g

## Grafi AND/OR

Sono particolare tipi di grafo che permettono la riduzione dei problemi.
- Ogni arco è chiamato connettore e collega 2 nodi in modo AND oppure OR
- Nel modo AND il nodo genitore necessita di tutti i nodi figli per la risoluzione del problema
- Nel modo OR necessita anche solo di uno di essi

(Esempio: Torre di Hanoi – non ci sono nodi OR, quindi la soluzione è deterministica, segue dei passi prestabiliti)

Per risolvere un grafo AND/OR è richiesto che i sottoproblemi siano indipendenti gli uni dagli altri.

## Procedura MiniMax

Si cerca una buona mossa (ricerca limitata in profondità)
- Valutazione a 1 stadio
  - Si parte dalla posizione attuale, si generano le posizioni successive e si effettua una valutazione di ogni nono (statica), scegliendo la migliore. Si massimizza
- Valutazione a 2 stadi
  - Si valutano anche le possibili mosse dell'avversario. Per ogni nodo si applica il generatore di mosse plausibili. Per l'avversario si sceglie il massimo dei minimi → MiniMax
- Algoritmo
  - Procedura ricorsiva
  - GEN_MOSSE(Pos) è il generatore di mosse accettabili, fatte a partire da Pos
  - STATICA(Pos,Profondità) è la funzione di valutazione statica che fornisce un numero che rappresenta la bontà di Pos dal punto di vista corretto.

# Metodi di ricerca

**Metodo di ricerca Generale**
Si parte dallo stato iniziale e si applica un'operazione finché non viene raggiunto lo stato finale.

Dal momento che spesso è improponibile costruire lo spazio totale degli stati (Scacchi = 10^120), i metodi di ricerca costruiscono ad ogni passo un grafo di ricerca che è solamente una porzione dello spazio degli stati, ricavata espandendo uno specifico stato.

## 1) Metodi di ricerca in ampiezza: espandono i nodi nell'ordine in cui sono generati

OPEN = nodi da espandere
CLOSED = nodi già espansi

1. Si mette il nodo di partenza in una lista OPEN
2. Se OPEN è vuota si esce con insuccesso
3. Rimuovo il 1° nodo *n* da OPEN e lo metto in una lista CLOSED
4. Espando *n*: se ci sono successori li metto al fondo di OPEN, creo i puntatori ad essi e vado al punto 2
5. Se il successore è un nodo finale esco con la soluzione ottenuta percorrendo i puntatori all'indietro altrimenti vado al punto 2

**Ricerca in ampiezza del cammino di costo minimo (Algoritmo di Dijkstra)**

Ogni arco *n*-*m* ha associato un costo c(*n,m*). Il costo di un cammino è la somma dei costi degli archi che lo compongono.
Se g(*n*) è il costo del cammino dal nodo iniziale ad *n*, allora l'algoritmo precedente è modificato così:

1. Dal nodo OPEN si toglie quello con g(*n*) minimo
2. Per ogni successore *m* generato gli si associa g(*m*) = g(*n*) + c(*n,m*) e il puntatore ad *n*
3. Se si incontra un nodo *m* già inserito in precedenza, si controlla se il costo del *m* attuale è inferiore a quello del *m* precedentemente inserito, in tal caso si sostituisce in OPEN il nuovo *m* raggiunto dal percorso diverso.

Spesso è utile considerare una conoscenza **euristica** per orientare la ricerca. In generale esistono 3 tipi di regole (pratiche) euristiche:

- Nella scelta del nodo da espandere
- Per generare solo alcuni dei successori
- Per potare alcuni rami dell'albero di ricerca (non promettenti)

**Ricerca in ampiezza con potatura (Beam Search)**

È possibile quando si dispone di un'euristica per misurare quanto è promettente un nodo (Esempio: nel gioco dell'8 si possono contare il numero di tessere fuori posto). Si generano tutti i nodi di un certo livello di profondità, ma si tengono solo i β nodi più promettenti, cancellando tutti gli altri. → β è un parametro scelto in base all'intuizione ed alle risorse disponibili.

Viene introdotta una lista RESERVE che immagazzina tutti i k nodi generati, quindi i β migliori sono ordinati e trasferiti in OPEN, mentre i restanti k-β vengono cancellati.

L'algoritmo precedente è modificato:

1. Se OPEN è vuota:
   a. Se RESERVE è vuota si esce con insuccesso
   b. Se RESERVE non è vuota si selezionano i β nodi più promettenti e si copiano in OPEN mantenendo l'ordine
   c. RESERVE viene svuotata
2. Si estrae il primo nodo *n* da OPEN e si espande, ma se non ha figli si va al punto 1
3. Se un successore di *n* è il nodo finale si esce con successo
4. Se non ha successo, si cancellano i nodi che non rispettano i vincoli del problema (*renegade*)
5. I rimanenti li copio in RESERVE in un ordine qualsiasi

## 2) Metodi di ricerca in profondità: si espande per primo l'ultimo nodo generato

- La profondità della radice è 0.
- La profondità di un discendente è uguale alla profondità del genitore più 1.
- È necessario stabilire un limite di profondità Pmax e prevedere una procedura di ritorno (backtracing)
- Si introduce il concetto di stack per il ritorno (questo lavoro è realizzato dalla lista OPEN)

L'algoritmo è il seguente (molto simile a quello di ricerca in ampiezza – le parti differenti sono sottolineate):

1. Si mette il nodo di partenza in una lista OPEN
2. Se OPEN è vuota si esce con insuccesso
3. Rimuovo il 1° nodo *n* da OPEN e lo metto in una lista CLOSED
4. Se la profondità di n è uguale al limite di profondità si va al punto 2.
5. Espando *n*: se ci sono successori li metto all'inizio di OPEN, creo i puntatori ad essi
6. Se uno dei successori è un nodo finale esco con la soluzione ottenuta percorrendo i puntatori all'indietro altrimenti vado al punto 2

Un approccio alternativo all'utilizzo di uno stack esplicito è quello di una proceduta ricorsiva

**Procedura ricorsiva Backtrack**

Si parte da uno stato iniziale, si generano i successori secondo gli operatori applicabili. Per ogni successore si ripete ricorsivamente la generazione dei successori tramite gli operatori a loro compatibili. Quando una ricorsione ha termine si risale di un livello e si procede con il fratello. È da notare che la procedura può non terminare se esistono cammini infiniti.

**Procedura ricorsiva Backtrack1**

Effettua dei controlli supplementari per evitare cicli e rami troppo lunghi. Per far ciò si determina una variabile LIVELLO_MAX e il livello attuale di volta in volta è confrontato con questo limite: se è superiore è ritornato FAIL e si procede con un altro nodo.

**Ricerca in profondità con hindsight backtracking**

Un nodo *n* è detto *subversive* se e solo se qualunque percorso dal nodo sorgente passante per *n* è barricato in qualcuno dei suoi discendenti. Esso non può quindi appartenere al percorso risolutivo.
Se lungo un percorso si scopre che un nodo è subversive tutti i nodi sottostanti vengono cancellati.

OPEN = nodi da espandere o parzialmente espansi
PROGENY = nodi nuovi

1. Si mette il nodo di partenza in una lista OPEN
2. Si preleva da OPEN il nodo *n* più recente e si genera un nuovo figlio verificando che non sia già presente in PROGENY
3. Se non è possibile generare figli (nodo inerte o completamente espanso)
    a. Si elimina n da OPEN
    b. Si cerca un nodo subversive partendo dal fondo di OPEN, se esiste lo si elimina insieme ai suoi sottonodi
    c. Se OPEN è vuota si esce con fallimento, altrimenti si va al punto 2
4. Se *n* ha un nuovo figlio *m*, lo si mette in PROGENY
5. Se *m* è un nodo destinazione di esce con successo e con il percorso risolutivo che è in OPEN
6. Se *m* è renegade o insipid (che supera il limite di profondità o di costo) si va al punto 2
7. Pongo *m* in cima alla lista OPEN e si va al punto 2

**Ricerca in profondità con leap-frogging**

Invece di generare solo un figlio alla volta, sono generati tutti insieme.
Si sceglie un nodo tra quelli generati con un apposito criterio.
Se un percorso è barricato al nodo *n* è possibile saltare ad uno dei suoi fratelli e continuare l'algoritmo.
Se tutti i suoi fratelli sono subversive allora si deve risalire di un livello e provare dalla parte di un fratello del genitore.. e così via.
Nella realizzazione della ricerca in profondità con leap-frogging, si utilizzano due liste, OPEN e CLOSED.

*Algoritmo:*

1. Se il nodo di partenza è un nodo goal, uscire con *successo* e il percorso di soluzione vuoto.
2. Si inizializzano OPEN e CLOSED a vuote. Si pone il nodo sorgente in OPEN.
3. Se OPEN è vuota, si esce con *fallimento*.
4. Si preleva da OPEN il nodo *x* più recente *x*.
5. Si pone *x* in CLOSED.
6. Si espande *x*.
7. Se non si può generare nessun figlio di *x*, cioè è inerte, allora:

   a. se si è acquisita conoscenza specifica (*hindsight*), si cerca un antenato subversive di *x*; se lo si scopre, lo si rimuove da OPEN insieme a tutti i discendenti in OPEN;
   b. si va a 3.
8. Se il figlio di *x* è un nodo destinazione, si esce con *successo* e con il percorso risolutivo ottenuto inseguendo all'indietro i puntatori al padre.
9. Cancellare tutti i figli di *x* che siano *renegade*, ricorrenti o *insipid*.
10. Si pongono i figli sopravvissuti di *x* in ordine decrescente in cima alla lista OPEN.
11. si va a 3.

## Algoritmi di ricerca euristica

Come si effettua la previsione per decidere quale nodo devo scegliere ed espandere???

La scelta viene fatta in base al cammino già percorso (si guarda solo indietro):

Ricerca in ampiezza (bradth-first): Previsione(x) = livello(x)
Ricerca in profondità (depth-first): Previsione(x) = – livello(x)
Algoritmo di Dijkstra: Previsione(x) = costo del cammino fino a x

Per verificare quanto un nodo è **promettente**:

Previsione(X) = stima del costo minimo da START a X + stima del costo minimo da X a GOAL

Spesso è necessario ricorrere all'uso di una **funzione di valutazione** $\hat{f}$ arbitraria, che fornisce una stima del costo di un cammino (a costo minimo) dal nodo di partenza al nodo finale vincolato a passare per un nodo *n*.
$\hat{f}(n)$ è il valore della funzione in *n*. $\hat{f}$ viene usata per ordinare i nodi da espandere secondo valori crescenti di $\hat{f}$.
Si ha allora un algoritmo di ricerca ordinata (Algoritmo A).

*Algoritmo A: (ricerca ordinata per alberi e grafi)*

1. Si mette il nodo di partenza s in una lista OPEN e si calcola $\hat{f}(s)$
2. se OPEN è vuota, uscire con fallimento, altrimenti continuare
3. Si rimuove da OPEN il nodo *n* con il più piccolo valore di $\hat{f}$ e lo si copia in CLOSED
4. Se *n* è un nodo finale esco con la soluzione ottenuta percorrendo i puntatori all'indietro
5. Espando *n* e genero i suoi successori $n_i$, se non ce ne sono si va al punto 2
6. Calcolare tutte le $\hat{f}(n_i)$ dei successori e creare i puntatori ad *n*
7. Aggiorno nelle liste OPEN e CLOSED i nodi con il valore $\hat{f}$ più piccolo tra quello attuale e il precedente. Se $\hat{f}$ è diminuito sposto il nodo da CLOSED a OPEN e ridirigo a *n* il puntatore. Si va al passo 2

**Procedura GraphSearch**: rispetto ai Backtrack ricorda tutti i nodi già trattati e non solo quelli sul cammino attuale.

**Algoritmo di ricerca ottimo A***

L'obiettivo è definire una funzione che massimizza una misura di efficienza e insieme garantisce un cammino di costo minimo fino alla meta.

$k(n_i, n_j)$ è il costo effettivo di un cammino a costo minimo tra $n_i$ e $n_j$.
$T$ è un insieme di stati finali.
$h(n_i)$ è il costo di un cammino di costo minimo da ni ad uno di questi stati finali.

$h_i = \min_{n_j \in T} k(n_i, n_j)$ un cammino che passa per $n_i$ e ha costo $h(n_i)$ è detto **cammino ottimo**.

Se $g(n)=k(s,n)$ è il costo di un cammino ottimo dal nodo di partenza *s* ed il generico nodo *n*, allora $f(n)=g(n)+h(n)$ è il costo di un cammino ottimo vincolato a passare per il nodo *n*.
Supponiamo di usare $\hat{f}(n)=\hat{g}(n)+\hat{h}(n)$ come funzione di valutazione e chiameremo $A^*$ l'algoritmo di ricerca ordinata che ne fa uso. Si vuol dimostrare che se $\hat{h}$ è una stima per difetto di *h*, $A^*$ trova un cammino ottimo. N.B. Se $\hat{h}\equiv 0$ , l'algoritmo è quello a costo uniforme. Poiché $\hat{h}\equiv 0$ è sicuramente una stima per difetto di *h*, dimostrando che $A^*$ trova un cammino ottimo si dimostra che lo trova anche l'algoritmo a costo uniforme.

**Ammissibilità di A***

Un algoritmo di ricerca è ammissibile se per ogni grafo termina sempre dando un cammino ottimo, quando questo esiste. Se H(n) è una stima per difetto di h(n) allora A* è ammissibile.

**Alcune considerazioni e definizioni**
*Definizione:*
Si dice che l'algoritmo A è *più informato* dell'algoritmo di B se la conoscenza euristica utilizzata da A permette di calcolare una stima per difetto di $h(n)$ che è sempre strettamente maggiore (per tutti i nodi *n* non finali) di quella permessa della conoscenza euristica utilizzata da B.

*Teoremi:*

- Se A è "*più informato*" di B, allora B espanderà almeno tanti nodi quanti ne espande A.
- Se la "*restrizione di monoticità*" (se $n_k$ è un successore di $n_i$, allora $h(n_i) - h(n_k) \le c(n_i, n_k)$) è soddisfatta, allora quando $A^*$ seleziona un nodo per l'espansione ha già trovato un cammino ottimo fino ad esso.

*L'importanza di $\hat{g}$ :*
Se interessa una soluzione qualunque, indipendentemente dal costo (ma non indipendentemente dallo sforzo di ricerca !), vi sono ragioni per includere $\hat{g}$ e anche per non includerla.

<u>*Ragioni per includerla*</u>*:* se $\hat{g}$ è inclusa in $\hat{f}$, si è sicuri che un cammino venga trovato, prima o poi. $\hat{g}$ in pratica aggiunge alla ricerca una componente in ampiezza, mentre $\hat{h}$ tende a contribuire per la profondità.
<u>*Ragioni per non includerla*</u>*:* tendenzialmente non interessa il costo dei cammini costruiti fino ad ora, proprio perché lo si è già fatto. Interessa piuttosto lo sforzo necessario per arrivare alla fine. Anche se ambedue le cose sembrano ragionevoli, la prima è più corretta, anche se non sempre dimostrabile analiticamente.

*Conclusioni:*
L'algoritmo $A^*$ fa parte del metodo di ricerca detto **best-first**. Si rammenta che in questo metodo il nodo da espandere è selezionato tra *tutti* i nodi inespansi esistenti, senza riguardo a dove si trovano nel grafo di ricerca.
<u>*Invece*</u>:

- nella ricerca in ampiezza, il nodo da espandere è scelto *solo* tra i nodi inespansi allo stesso livello di profondità;
- nella ricerca in profondità, il nodo da espandere è scelto *solo* tra i nodi inespansi che sono fratelli.

Nel metodo best-first, se si pone h(x) = 0, f(x) = g(x). La ricerca è detta a *costo uniforme* (o *incurred cost search*). Se si pone g(x) = 0, f(x) = h(x). La ricerca è detta *greedy* (o *predicted cost search*).

## Metodo di Ricerca Best-First

Il nodo da espandere è scelto tra tutti i nodi inespansi esistenti senza badare a dove si trovano nel grafo. Un criterio è scegliere quello che, indipendentemente da quanto distante si trovi, presenta comunque la miglior funzione obiettivo.

## Riduzione a sottoproblemi

Lo scopo della riduzione a sottoproblemi è arrivare a sottoproblemi con soluzione ovvia (risolvibili al massimo in un passo).
La rappresentazione di questa riduzione è realizzata per mezzo di grafi AND/OR.

Per dimostrare che un nodo non terminale è risolvibile:

1. Se ha successori OR, esso è risolto se almeno uno dei successori è risolto
2. Se ha successori AND, esso è risolto se tutti i successori sono risolti

Specularmente, per dimostrare che un nodo non terminale è insolubile:

3. Se ha successori OR, esso è insolubile se tutti i successori sono insolubili
4. Se ha successori AND, esso è insolubile se almeno uno dei successori è insolubile

Il **grafo risolutivo** è il sottografo di nodi risolti che dimostra che il nodo di partenza è risolto.

## Pianificazione nella riduzione a sottoproblemi

Un problema di ricerca nello spazio degli stati è definito dalla terna (S, F, G) dove:

- S = stato/i iniziale/i
- F = insieme di operatori applicabiti
- G = stato/i finale/i

Se si trova una opportuna successione di stati "*chiave*" $g_1, g_2, ..., g_n$ il problema si può ridurre nell'insieme di problemi definiti dalle terne $(S, F, \{g_1\})$, $(\{g_1\}, F, \{g_2\})$, …, $(\{g_n\}, F, G)$ dove $\{g_1\} \equiv G_1$, $\{g_2\} \equiv G_2$, ecc.

Nei problemi reali spesso è facile identificare "*passi cruciali*" per la soluzione (operatori "*chiave*"). Sia *f* nell'insieme F un operatore chiave nel problema definito da *(S,F,G)*. Conviene allora cercare un cammino fino a uno stato in cui *f* è applicabile. Sia $G_f$ l'insieme degli stati a cui *f* è applicabile. Abbiamo identificato allora il sottoproblema *(S,F,* $G_f$*)*.
Ora se *g* di $G_f$ è uno stato chiave, ($\{g\}, F, \{f(g)\}$), dove $f(g)$ è lo stato che si ottiene applicando *f* a *g*, è un problema primitivo (basta applicare *f*!). Ci rimane quindi il sottoproblema descritto da $(f(g), F, G)$ per arrivare alla soluzione.
Per trovare gli aspiranti operatori chiave si usa un metodo basato sulle *differenze*. Una differenza per *(S,F,G)* è una lista parziale dei motivi per cui il criterio di stato finale (l'insieme G) non è soddisfatto dai membri di S. Gli aspiranti operatori chiave sono quelli che si applicano a una differenza (nel senso che "*rimuovono*" la differenza).

## Analisi mezzi-fini

Il processo di analisi può essere fondato su tabelle che, per ogni operatore, evidenziano la differenza (o le differenze) che l'operatore elimina. Queste liste sono solitamente ordinate rispetto a una qualche priorità. Inoltre queste liste devono includere le condizioni che devono essere soddisfatte per l'applicabilità delle regole. Su questo metodo sono basati i sistemi quali il GPS (General Problem Solver), sviluppato da Newell & Ernst.

## Ricerca su grafi AND/OR

*Lo schema generale di risoluzione è*:

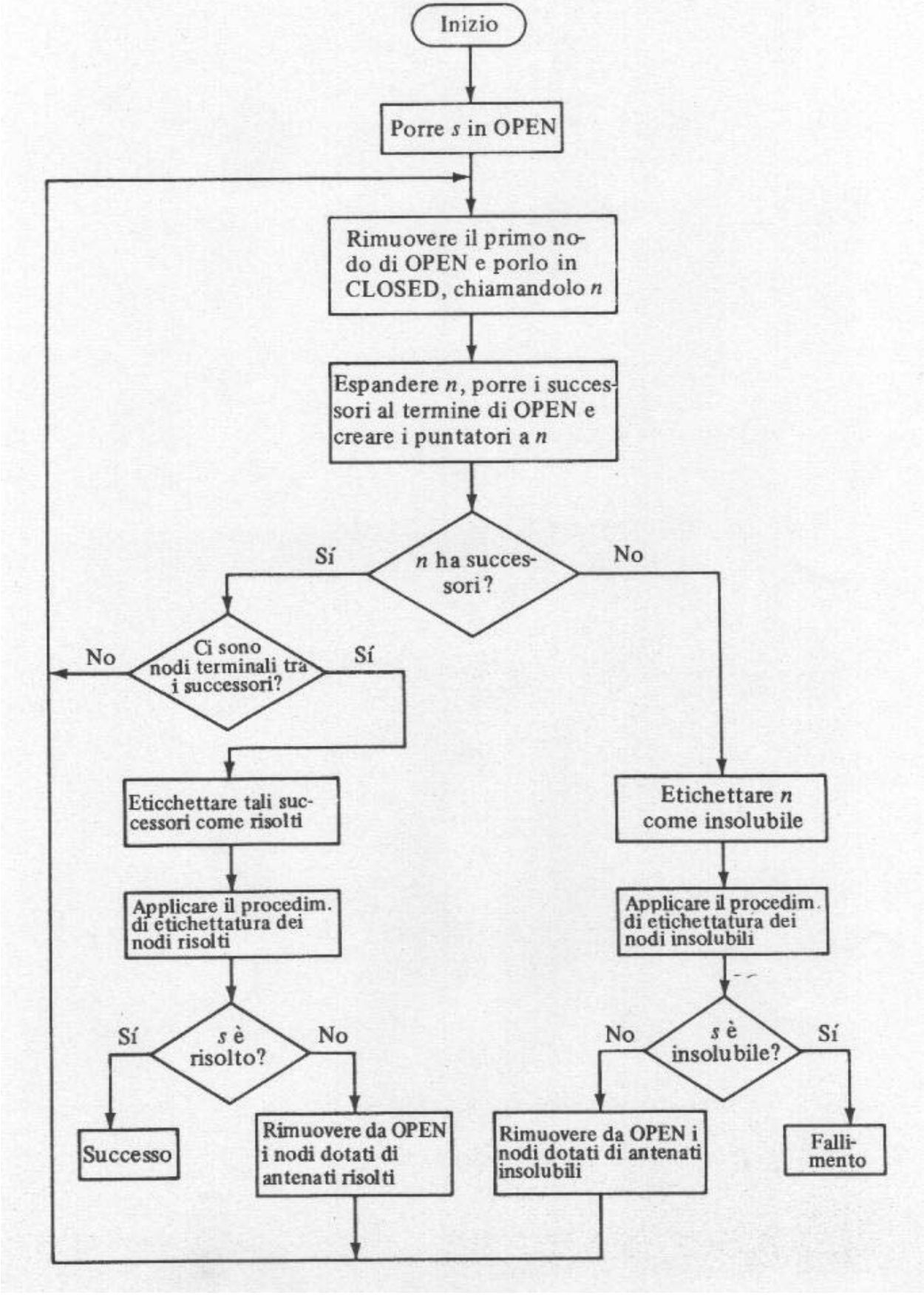
1. si associa un nodo di partenza alla descrizione del problema iniziale;
2. si calcolano insiemi di successori del nodo di partenza applicando i possibili operatori di riduzione a sottoproblemi. Sia Γ l'operatore combinato che calcola tutti i successori di un nodo; ancora chiameremo *espansione* di un nodo il processo di applicazione di Γ al nodo
3. (si ricordi che se si genera più di un insieme di successori AND ogni insieme non unitario va raggruppato sotto un nodo OR intermedio);
4. si predispongono puntatori da ogni nodo successore al nodo genitore. Questi vengono utilizzati nel processo di etichettatura dei nodi risolti e insolubili, e indicano il grafo risolutivo dopo la terminazione;
5. si continua il processo di espansione dei nodi e di predisposizione dei puntatori finché si può etichettare il nodo di partenza come risolto o insolubile.

**Ricerca in ampiezza (per alberi)**

1. porre il nodo *s* di partenza in una lista di nome OPEN;
2. rimuovere il primo nodo di OPEN e porlo in una lista di nome CLOSED, chiamandolo *n*;
3. espandere il nodo *n*, generandone tutti i successori; porre questi alla *fine* di OPEN predisponendo puntatori ad *n*. Se non ci sono successori, etichettare *n* come insolubile e proseguire, altrimenti andare a 8.
4. applicare il procedimento di etichettatura dei nodi insolubili all'albero di ricerca;
5. se il nodo di partenza è etichettato come insolubile, uscire con un fallimento, altrimenti continuare;
6. rimuovere da OPEN tutti i nodi con antenati insolubili (questo passo ci permette di evitare lo sforzo superfluo di tentare di risolvere problemi insolubili);
7. andare a 2;
8. se alcuni dei successori sono nodi terminali, etichettarli come risolti e proseguire, altrimenti andare a 2;
9. applicare il procedimento di etichettatura dei nodi risolti all'albero di ricerca;
10. se il nodo di partenza è etichettato come risolto, uscire con l'albero risolutivo che verifica ciò; altrimenti continuare;
11. rimuovere da OPEN tutti i nodi risolti o con antenati risolti (questo passo ci permette di evitare lo sforzo superfluo di risolvere un problema in più di un modo);
12. andare a 2.

Può essere utile in questo caso lo schema a blocchi di questo algoritmo:

*... – Schema a blocchi della procedura di ricerca in ampiezza per gli alberi AND/OR*



**Ricerca in profondità (per alberi)**
Come per gli alberi ordinari, si definisce la *profondità* di un nodo in un albero AND/OR come segue:

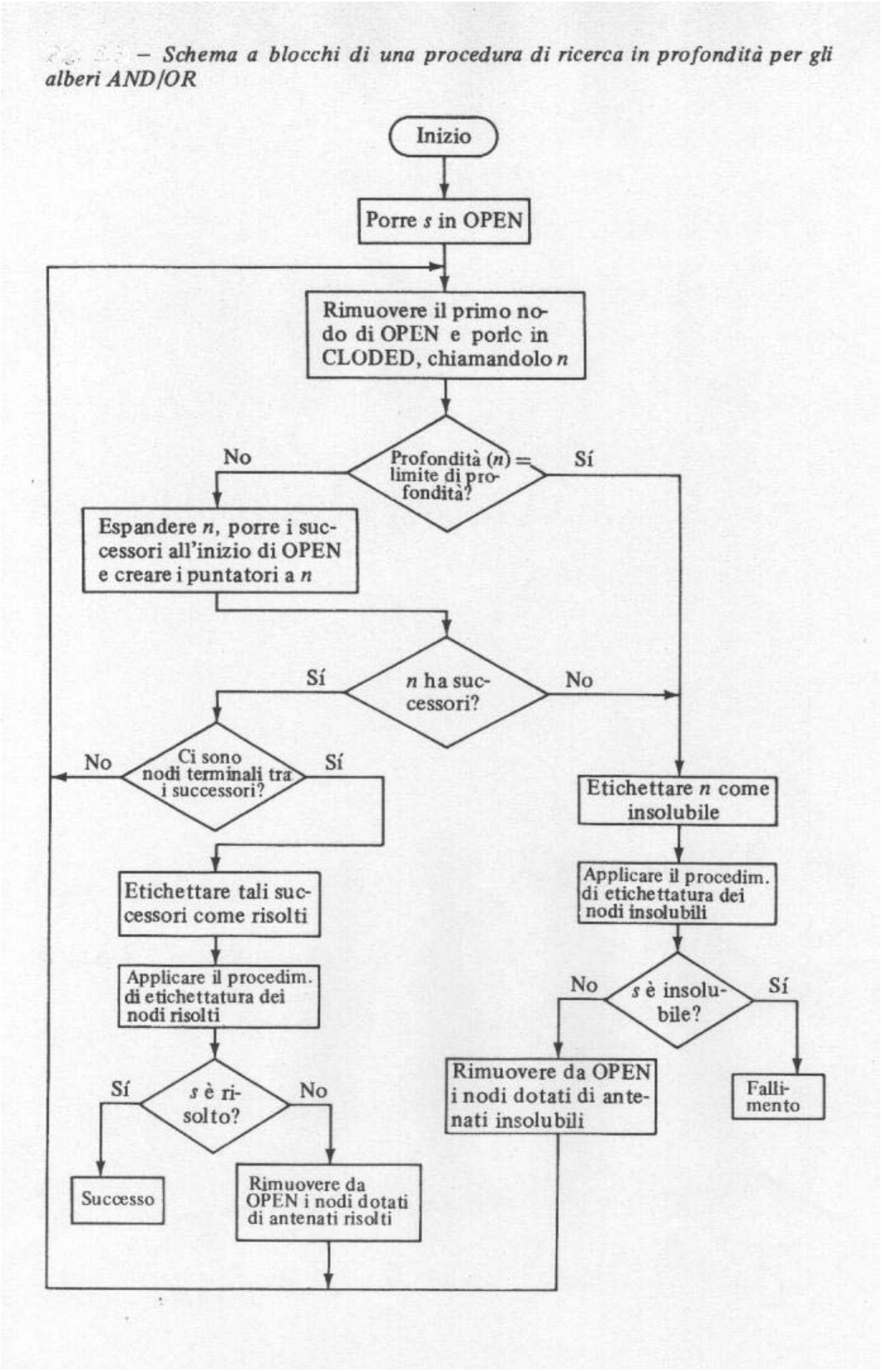
- La profondità del nodo di partenza è zero
- La profondità di ogni altro nodo è uguale alla profondità del suo genitore più 1.

**Algoritmo:**
avvertenza: i nodi non terminali che si trovano oltre un limite di profondità vengono etichettati come insolubili.

1. porre il nodo *s* di partenza in una lista di nome OPEN;
2. rimuovere il primo nodo di OPEN e porlo in una lista di nome CLOSED, chiamandolo *n*;
3. se la profondità di *n* è uguale al limite di profondità etichettare *n* come insolubile e andare a 5, altrimenti proseguire;
4. espandere il nodo *n*, generandone tutti i successori. Porre questi all'inizio di OPEN (in ordine arbitrario) e predisporre puntatori a *n;* se non ci sono successori etichettare *n* come insolubile e continuare, altrimenti andare a 9;
5. applicare all'albero di ricerca il procedimento d'etichettatura dei nodi insolubili;
6. se il nodo di partenza è stato etichettato come insolubile, uscire con un fallimento; altrimenti proseguire;
7. rimuovere da OPEN tutti i nodi con antenati insolubili;
8. andare a 2;
9. se alcuni dei successori sono nodi terminali etichettarli come risolti e proseguire, altrimenti andare a 2;
10. applicare all'albero di ricerca il procedimento di etichettatura dei nodi risolti;
11. se il nodo di partenza viene etichettato come risolto, uscire con l'albero risolutivo che verifica ciò; altrimenti proseguire;
12. rimuovere da OPEN i nodi risolti o con antenati risolti;
13. andare a 2.

Ecco lo schema a blocchi dello stesso algoritmo:

*– Schema a blocchi di una procedura di ricerca in profondità per gli alberi AND/OR*




Per trattare grafi AND/OR occorre un algoritmo come $A^*$, ma in più capace di trattare gli archi AND. Infatti $A^*$ è inadeguato.

## Considerazioni sull'Algoritmo $AO^*$

Per la ricerca su un grafo AND/OR occorre:

- Attraversare il grafo a partire dal nodo iniziale seguendo il cammino ottimo attuale; accumulare l'insieme dei nodi presenti sul cammino e non ancora espansi
- Prendere uno dei nodi non espansi ed espanderlo. Aggiungere i successori al grafo e calcolare $\hat{f}$ (si usa solo *h*)
- Cambiare la stima di $\hat{f}$ del nodo appena espanso (per tener conto dei successori). Propagare all'indietro questo cambiamento. Ad ogni nodo visitato mentre si risale il grafo, decidere quale dei suoi archi successori è il più promettente e marcarlo come parte dell'attuale cammino migliore.

N.B.:

- l'attuale cammino ottimo può cambiare
- la propagazione della stima all'indietro non era necessaria in $A^*$, perché si esaminavano solo i nodi non espansi. Qui, invece, i nodi espansi devono essere riesaminati.

## Concetto di "costo"

Nella ricerca negli spazi degli stati è stata usata una funzione di valutazione euristica (stima il costo di un cammino ottimo tra un nodo e l'obiettivo). Per gli alberi AND/OR occorre introdurre il concetto di costo di un albero risolutivo radicato in un dato nodo.

**Iª definizione**:
*Costo complessivo*: somma dei costi di tutti gli archi nell'albero risolutivo.

**IIª definizione:**
*Costo massimo*: costo del cammino a massimo costo nell'albero risolutivo. Un cammino radicato in *n* è una successione di nodi da un nodo iniziale *n* a un nodo finale $n_k$ di cui uno è successore dell'altro.

Si cerca un albero risolutivo a minimo costo all'interno dell'intero albero AND/OR implicito (*albero risolutivo* ≡ *sottoalbero*). Sarà detto *albero risolutivo ottimo*. Sia $h(s)$ il costo di un albero risolutivo ottimo radicato nel nodo di partenza *s*. Se $C(n_i, n_j)$ è il costo dell'arco fra $n_i$ ed il successore $n_j$, valgono le seguenti definizioni:

1. se *n* è un nodo terminale (corrispondente a un problema primitivo) $h(n=0)$
2. se *n* è un nodo non terminale con successori OR $n_1, n_2, \ldots, n_k$, $h(n) = \min_i [c(n,n_i) + h(n_i)]$
3. se *n* è un nodo non terminale con successori AND $n_1, n_2, \ldots, n_k$, $h(n) = \sum_{i=1}^{k} [c(n,n_i) + h(n_i)]$ per il costo complessivo $h(n) = \max_i [c(n,n_i) + h(n_i)]$ per il costo massimo.

Naturalmente $h(n)$ è indefinito se *n* è un nodo insolubile. Anche per l'albero AND/OR si farà uso del concetto di *costo stimato*, cioè si introduce una funzione $\hat{h}(n)$ che è una stima di $h(n)$, costo di un albero risolutivo ottimo radicato in *n*. $\hat{h}$ è una *funzione euristica*.

Ad ogni passo, all'estremo dell'albero di ricerca ci possono essere nodi, detti foglie, che ricadono in uno di questi tre casi :

1. nodi già identificati come terminali dal processo di ricerca;
2. nodi già identificati come non terminali privi di successori dal processo di ricerca;
3. nodi i cui successori non sono ancora stati generati dal processo di ricerca.

Quindi per ogni nodo *n* dell'albero di ricerca, la funzione $\hat{h}(n)$ può essere così definita:

1. se *n* è un nodo foglia:
   a. se *n* è stato identificato come terminale $\hat{h}(n) = 0$ ;
   b. se *n* è stato identificato come non terminale privo di successori, $\hat{h}(n)$ è indefinito;

c. se $n$ è un nodo i cui successori non sono ancora stati generati $\hat{h}(n)$ è una stima euristica del costo $h(n)$ di un albero risolutivo ottimo radicato nel nodo $n$; tale stima si deve basare sulla conoscenza euristica disponibile sul dominio del problema rappresentato dall'albero AND/OR.

2. se $n$ è un nodo non foglia con successori OR $n_1, n_2, \ldots, n_k$: $\hat{h}(n) = \min_i \left[ c(n, n_i) + \hat{h}(n_i) \right]$
3. Se $n$ è un nodo non foglia con successori AND $n_1, n_2, \ldots, n_k$, $\hat{h}(n) = \sum_{i=1}^{k} \left[ c(n, n_i) + \hat{h}(n_i) \right]$ per il costo complessivo, $\hat{h}(n) = \max_i \left[ c(n, n_i) + \hat{h}(n_i) \right]$ per il costo massimo

Si introduce ancora il concetto di *albero risolutivo potenziale per s*, cioè un sottoalbero radicato in $s$ che potrebbe divenire la parte superiore di un albero risolutivo completo. Ad ogni passo della ricerca si estrae l'albero risolutivo potenziale $\tau_0$ radicato in $s$ che si ***stima*** essere la parte superiore di un albero risolutivo ottimo radicato in $s$.
I valori di $\hat{h}$ usati come stime sono basati su queste definizioni:

1. il nodo di partenza appartiene a $\tau_0$
2. se il nodo $n$ appartiene a $\tau_0$, allora:
   a. se il nodo $n$ ha successori OR $n_1, n_2, \ldots, n_k$ nell'albero di ricerca il successore con il minimo valore di $\left[ c(n, n_i) + \hat{h}(n_i) \right]$ appartiene a $\tau_0$ (eventuali conflitti sono risolti arbitrariamente);
   b. se il nodo $n$ ha successori AND nell'albero di ricerca, tutti questi successori appartengono a $\tau_0$.

In somma, la strategia che verrà usata sarà quella di estendere l'albero risolutivo potenziale più promettente (e non il nodo più promettente). Un buon criterio è scegliere un nodo la cui espansione ha la maggiore provabilità di ***refutare*** l'ipotesi che $\tau_0$ sia veramente la parte superiore di un albero risolutivo ottimo (in modo da tagliare al più presto i rami inutili!).

**Algoritmo** $AO^*$

1. porre il nodo di partenza $s$ in una lista di nome OPEN e calcolare $\hat{h}(s)$ ;
2. determinare l'albero risolutivo potenziale $\tau_0$, che si ritiene essere la parte superiore dell'albero risolutivo ottimo radicato in $s$, usando i valori di $\hat{h}$ ai nodi dell'albero di ricerca;
3. scegliere una foglia di $\tau_0$ che si trovi in OPEN e porla in CLOSED, chiamandola $n$;
4. se $n$ è un nodo terminale etichettarlo come risolto e continuare, altrimenti andare a 9;
5. applicare a $\tau_0$ il procedimento di etichettatura dei nodi risolti;
6. se il nodo di partenza è etichettato come risolto uscire con $\tau_0$ come albero risolutivo, altrimenti continuare;
7. rimuovere da OPEN i nodi con antenati risolti;
8. andare a 2;
9. applicare a $n$ l'operatore di successione Γ, generandone tutti i successori. Se non ci sono successori etichettare $n$ come insolubile e continuare, altrimenti andare a 14;
10. applicare a $\tau_0$ il procedimento di etichettatura dei nodi insolubili;
11. se il nodo di partenza è etichettato come insolubile uscire con un fallimento, altrimenti continuare;
12. rimuovere da OPEN i nodi con antenati insolubili;
13. andare a 2;
14. porre i successori in OPEN e predisporre i puntatori a $n$. Calcolare i valori di $\hat{h}$ per i successori e ricalcolare i valori di $\hat{h}$ per $n$ e i suoi antenati;
15. andare a 2.

## Giochi a due giocatori

Caratteristiche:

- mosse alternate
- non c'è intervento del caso (no giochi di carte)
- ogni giocatore ha informazioni complete sullo stato del gioco

Si può applicare il metodo di riduzione a sottoproblemi per trovare una strategia vincente mediante la "*dimostrazione*" che il gioco può essere vinto.

### Procedura minimax

Nei giochi complessi (dama, scacchi) non si può dimostrare la vittoria (alberi con $10^{40}$ e $10^{120}$ nodi!). Ci si limita a cercare una "buona" mossa (ricerca limitata in profondità). Schematicamente:

- Guarda se è una mossa vincente. Se sì, la considera la migliore dandole un valore molto alto.
- In caso contrario considera tutte le mosse che l'avversario può fare successivamente. Vede qual è la peggiore per il giocatore (chiamando ricorsivamente questa procedura). Assume che l'avversario farà la mossa che pensa sia la peggiore (e che l'avversario considererà la migliore). Qualunque sia il valore della mossa peggiore, lo passa come il valore del caso che sta considerando.
- Il caso migliore è allora quello con valore più alto.

L'ipotesi è che l'avversario farà la mossa a lui più conveniente.

*Valutazione ad 1 stadio:*
Si parte dalla posizione attuale, si generano le posizioni successive e si effettua una valutazione di ogni nodo (valutazione statica), si sceglie la migliore. Si riporta questo valore all'indietro (*backed-up value*) e lo si attribuisce al nodo corrente. In questo stadio *si massimizza*.

*Valutazione a 2 stadi:*
Si valutano anche le possibili mosse dell'avversario. Per ogni nodo si applica il generatore di mosse plausibili. Le mosse dell'avversario sono da considerare come AND (per ciascun nodo OR).

### Algoritmo minimax

Procedura ricorsiva che utilizza:
GEN MOSSE(Pos): è il generatore di mosse accettabili che fornisce una lista di nodi che rappresentano le mosse che potrebbero essere fatte a partire da Pos.
STATICA(Pos, Profondità): è la funzione di valutazione statica che fornisce un numero che rappresenta la bontà di Pos dal punto di vista corretto. Valori alti indicheranno una buona posizione per la parte che sta per muovere, come indicato dal valore di Profondità pari o dispari. Chi ha chiamato STATICA cercherà di massimizzare il suo punteggio. Quando il valore calcolato viene passato indietro al livello più alto successivo, verrà negato per riflettere la possibilità della situazione dal punto di vista del giocatore avversario. Invertendo i valori a livelli alternati, la procedura MINI-MAX può essere molto semplice; in effetti fra i valori a disposizione sceglie sempre quello massimo.
Elementi che influenzano la decisione di fermare la ricorsione (e quindi chiamare la funzione di valutazione statica):

- Una delle due parti ha vinto?
- Quanti stadi abbiamo già esplorato?
- Quanto è promettente questo cammino?
- Quanto tempo rimane?
- Quanto è stabile la configurazione?

Tutti questi elementi (e altri) sono valutati dalla funzione booleana:

ABBASTANZA_PROFONDO che può assumere i valori: $\begin{cases} TRUE \rightarrow smettere \\ FALSE \rightarrow continuare \end{cases}$

MINI-MAX restituisce:

- Il valore riportato indietro del cammino che sceglie (VALORE)
- Il cammino stesso. Avremo l'intero cammino anche se probabilmente è effettivamente necessario solo il passo più alto (CAMMINO).

Chiamata iniziale: MINIMAX(ATTUALE, 0)

1. ATTUALE è il nodo a partire dal quale si vuole la valutazione
2. 0 indica la profondità nulla del nodo iniziale

MINIMAX (POSIZIONE, PROFONDITÀ)

1. Se ABBASTANZA-PROFONDO (PROFONDITA), allora fornisce la struttura VALORE = STATICA(POSIZIONE, PROFONDITA); CAMMINO = nil. Ciò indica che non vi è un cammino a partire da questo nodo e che il suo valore è quello determinato dalla funzione di valutazione statica.
2. Altrimenti genera un altro stadio dell'albero chiamando GENMOSSE(POSIZIONE) e dando a SUCCESSORI il valore della lista che fornisce.
3. Se SUCCESSORI è vuoto, allora non vi sono mosse da fare, quindi si ha la stessa struttura che si sarebbe avuta se ABBASTANZA-PROFONDO avesse dato vero.
4. Se SUCCESSORI non è vuoto, allora lo attraversa esaminando ogni elemento e tenendo traccia del migliore, nel modo seguente.
5. Inizializza PUNTEGGIO-MIGLIORE col valore minimo che possa dare STATICA; esso verrà aggiornato per riflettere il punteggio ottimo che può essere raggiunto da un elemento di SUCCESSORI.
6. Per ogni elemento di SUCCESSORI (che chiameremo SUCC) si fanno le operazioni seguenti:
    a. Si attribuisce a RISULTATO-SUCC il valore MINIMAX (SUCC, PROFONDITA + 1). Questa chiamata ricorsiva di MINIMAX attuerà effettivamente l'esplorazione di SUCC.
    b. Si attribuisce a NUOVO-VALORE il valore meno VALORE(RISULTATO-SUCC). Ciò farà in modo che rifletta le possibilità della posizione dalla prospettiva opposta a quella del livello più successivo.
    c. Se NUOVO-VALORE > PUNTEGGIO-MIGLIORE, allora abbiamo trovato un successore migliore di tutti quelli esaminati fino ad ora e lo si memorizza nel modo seguente:
        i. Si attribuisce a PUNTEGGIO-MIGLIORE il valore NUOVO-VALORE.
        ii. Il cammino noto migliore è ora quello che va da ATTUALE a SUCC e poi lungo il cammino appropriato che parte da SUCC come determinato dalla chiamata ricorsiva di MINIMAX. Quindi si attribuisce a CAMMINO-MIGLIORE il risultato derivante dall'aver appeso SUCC a CAMMINO(RISULTATO-SUCC).
7. Ora che sono stati esaminati tutti i successori, sappiamo qual è il valore di NODO, e quale cammino si deve prendere a partire da esso. Quindi abbiamo la struttura VALORE = PUNTEGGIO-MIGLIORE CAMMINO = CAMMINO-MIGLIORE

**Potatura Alfa-Beta**

Posporre la *valutazione* alla completa *generazione* dell'albero di ricerca è inefficiente: meglio valutare mentre si genera.

*Alcune considerazioni:*
È sufficiente memorizzare i valori sintetizzati provvisori, e aggiornarli man mano che si generano i successori.
Osservando che:

- il VSP di un nodo AND (compreso il nodo di partenza) non può mai diminuire;
- il VSP di un nodo OR non può mai aumentare

Le regole per interrompere la ricerca sono:

a. si può tralasciare la ricerca sotto i nodi OR aventi VSP *minore o uguale* del VSP di uno dei loro antenati AND; a tali nodi OR si può assegnare il loro VSP come valore sintetizzato finale[1];
b. si può tralasciare la ricerca sotto i nodi AND aventi VSP *maggiore o uguale* del VSP di uno dei loro antenati OR; a tali nodi AND si può assegnare il loro VSP come valore sintetizzato finale.

Durante la ricerca, i VSP sono calcolati come segue:

1. i VSP dei nodi AND (compreso il nodo di partenza) sono posti uguali al *più grande* valore sintetizzato finale corrente dei suoi successori;
2. i VSP dei nodi OR sono posti uguali al *più piccolo* valore sintetizzato finale corrente dei suoi successori.

I VPS dei nodi AND sono chiamati *valori alfa.*
I VPS dei nodi OR sono chiamati *valori beta*.
Nel caso a) si ha un *troncamento alfa*
Nel caso b) si ha un *troncamento beta*
Da qui la dizione di *procedura alfa-beta*. La procedura termina quando tutti i successori del nodo di partenza hanno ricevuto un valore sintetizzato finale. La migliore mossa è quella corrispondente al successore con il massimo valore sintetizzato finale.
N.B: stessa mossa di MINI-MAX ma con meno ricerca.

In conclusione l'effettivo uso di alfa e beta consiste nel terminare la ricerca:

- ad un livello di minimizzazione quando viene scoperto un valore minore di alfa
- ad un livello di massimizzazione quando è stato trovato un valore maggiore di beta.

### Algoritmo ricorsivo per la ricerca minimax con potatura alfa-beta

(The following version, which evaluates the minimax value of a node $n$ relative to the cut-off values $\alpha$ and $\beta$, is adapted from [Pearl 1984, p. 234])

AB(n; $\alpha$, $\beta$)

1. If $n$ at depth bound, return AB($n$) = static evaluation of $n$. Otherwise, let $n_1$, … $n_k$, ... $n_b$ be the successors of $n$ (in order), set $k \leftarrow 1$ and, if $n$ is a *MAX* node, go to step 2; else go to step 2′.
2. Set $\alpha \leftarrow \max[\alpha, \text{AB}(n_k; \alpha, \beta]$.
3. If $\alpha \geq \beta$, return $\beta$; else continue.
4. If $k = b$, return $\alpha$; else proceed to $n_{k+1}$, i.e., set $k \leftarrow k + 1$ and go to step 2.

2' Set $\beta \leftarrow \min[\beta, \text{AB}(n_k; \alpha, \beta)]$.
3' If $\beta \leq \alpha$, return $\alpha$; else continue.
4' If $k = b$, return $\beta$; else proceed to $n_{k+1}$, i.e., set $k \leftarrow k + 1$ and go to step 2'.

We begin an alpha-beta search by calling AB($s$; $-\infty$, $+\infty$ ), where $s$ is the start node.

Throughout the algorithm, $\alpha < \beta$.

The ordering of nodes in step 1 of the algorithm has an important effect on its efficiency.

[1] Si noti che questi possono non essere i "veri" valori sintetizzati di questi nodi, ma sono dei confini che portano a un appropriato calcolo del VSP dei loro antenati

# Logica

Scopo della logica è quello di formalizzare i meccanismi di ragionamento. Essa studia proposizioni, ossia espressioni che rappresentano affermazioni..

## Logica proposizionale

In essa ad ogni proposizione elementare viene associata una variabile preposizionale.
Essa usa i simboli:
$\wedge$ congiunzione
$\rightarrow$ implicazione
esempio: $(p \wedge q) \rightarrow r$
Il problema della logica preposizionale è che le proposizioni elementari sono unità atomiche non scomponibili. Per esaminare la *struttura interna* delle proposizioni è però necessario uno strumento più potente: la **logica dei predicati**

Semantica: richiede l'introduzione di valori di verità, T (true) e F (false), quindi l'insieme B=(T,F)

| P | $\neg$P |
|---|---|
| T | F |
| F | T |

| P | Q | P$\wedge$Q |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | F |
| F | F | F |

| P | Q | P$\vee$Q |
|---|---|---|
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

| P | Q | P$\rightarrow$Q |
|---|---|---|
| T | T | T |
| T | F | F |
| F | T | T |
| F | F | T |

Il simbolo $\rightarrow$ è detto di *implicazione materiale*, non è intuitivo e non significa causalità.
Due formule F e G sono equivalenti (scritto F $\Leftrightarrow$ G) se e solo se esse hanno lo stesso valore di verità in tutte le interpretazioni.
Ogni formula è finita e contiene un numero finito di formule atomiche, quindi è sempre possibile determinare se essa è valida, inconsistente, o né l'una ne l'altra. La logica preposizionale è quindi DECIDIBILE

**Formule**
La logica preposizionale tratta formule, una formula è composta da:
- Formule atomiche (o atomi)
- Connettivi logici

**Formule ben formate (FBF)**
Si possono rappresentare fatti del mondo mediante proposizioni logiche scritte come formule ben formate (fbf). Per superare alcuni limiti della logica preposizionale, si usa la logica dei predicati. Per l'uso del calcolo dei predicati si deve definire la sintassi e la semantica (in quanto è un linguaggio per modellare il mondo):
Sintassi: specifica un alfabeto di simboli,
l'alfabeto è formato da
- simboli di interpunzione come , ()
- simboli logici o connettivi come $\neg \vee \wedge \rightarrow \Leftrightarrow$
- lettere funzionali n-adiche $f_i^n$
- lettere predicative n-adiche $p_i^n$

con questi simboli si costruiscono varie espressioni quali:
- termini $\rightarrow$ lettere costanti
- formule atomiche $\rightarrow$ lettere proposizionali
- formule ben formate (fbf) $\rightarrow$ ogni formula atomica è una fbf, in buona sostanza lo è se è riconducibile a Vero o Falso

Una formula è BEN FORMATA (FBF) se e solo se essa è ottenibile applicando le seguenti regole:
1) Un atomo è una FBF
2) Se F è una FBF, allora ($\neg$ F) è una FBF
3) Se F e G sono FBF, allora (F$\vee$G), (F$\wedge$G), (F$\rightarrow$G) e (F$\Leftrightarrow$G) sono FBF

Semantica:
una fbf assume "significato" quando è interpretata come affermazione sul dominio del discorso.
- Dominio: insieme non vuoto (anche infinito)
- Funzioni sul dominio: dato un dominio D, una funzione m-adica manda ogni n-pla di elementi di D in un elemento di D

Validità e soddisfacibilità:
si dice valida una fbf con valore Vero in tutte le interpretazioni e si dimostra che se una fbf è valida, esiste una procedura che ne verifica la validità.
Se una interpretazione dà il valore V a tutte le fbf di un insieme, si dice che le soddisfa. Una fbf W segue logicamente da un insieme di fbf S se ogni interpretazione che soddisfa S soddisfa pure W.
Per dimostrare che un insieme di fbf è insoddisfacibile, occorre provare che non esiste nessuna interpretazione in cui tutte le fbf hanno valore Vero.

## Logica dei predicati

Introduce individui e variabili individuali, funzioni e predicati.
Utilizza i quantificatori:
∃ esiste
∀ per ogni
che usano una variabile individuale.
Esempio: ∀x romano(x) → fedele(x,Cesare) ∨ odia(x,Cesare)
Variabili individuali $H_i$:
sono usate per indicare una generica costante
Quantificatore universale:
esprime il concetto di "per tutti i valori assunti" e si indica con ∀
Quantificatore esistenziale:
esprime il concetto di "esiste almeno un elemento di D". Si indica con ∃

Uso della logica dei predicati:
determinare la validità di una proposizione, date certe premesse

La logica dei predicati è INDECIDIBILE, quindi non esiste nessun algoritmo che permetta sempre di determinare se una data proposizione segue logicamente da un insieme di proposizioni dato.
Estensioni: in alcuni casi è necessario introdurle, per esempio se in una formula vera si sostituisce ad una sua parte qualcosa di equivalente, non sempre si ottiene ancora una formula vera

Si possono progettare algoritmi che funzionano se la proposizione segue logicamente, altrimenti possono operare all'infinito. Vi sono due metodi validi: la *deduzione naturale* e il *metodo di risoluzione*
Ci sono vari metodi di operare:

- Si può partire dagli assunti, fare le sostituzioni e vedere se si arriva alla meta (alto fattore di ramificazione)
- In alternativa si ragiona all'indietro, partendo dalla meta (basso fattore di ramificazione)

Per provare la metà, si usano le regole di inferenza per trasformarla in altra meta e così via finchè sono soddisfatte tutte le mete (si ottiene un grafo AND-OR)
Per arrivare alla dimostrazione occorrono processi quali

- L'unificazione (cioè trovare il matching tra le componenti le diverse affermazioni)
- La sostituzione
- L'applicazione del *modus ponens:* stabilisce che (P∧ (P→Q)) → Q ossia a parole: x esempio se passare il test di Turing implica che la macchina può pensare AND un certo computer ha passato il test di Turing, allora l'implicazione è che la macchina può pensare

## La formula a clausole

Si applicano in successione queste operazioni:

1) Eliminazione delle implicazioni
2) Riduzione del campo dei segni di negazione
3) Standardizzazione delle variabili
4) Eliminazione dei quantificatori esistenziali
5) Conversione in forma premessa
6) Trasformazione della matrice in forma normale congiuntiva
7) Eliminazione dei quantificatori universali
8) Eliminazione dei segni di congiunzione

| Forma normale | Sostituita da |
|---|---|
| A → B | not A v B |
| not (A ^ B) | not A v not B |
| not (A v B) | not A ^ not B |
| not (∀x) A | (∃ x) (not A) |
| not (∃ x) A | (∀x) (not A) |
| (∀x) (P(x) → (∃ x) Q(x)) | (∀x) (P(x) → (∃ y) Q(y)) |
| (∀y ∃ x) P(x,y) | (∀y) P(g(y), y) |
| (∃ x) P(x) | P(a) |
| A v (B ^ C) | (A v B) ^ (A v C) |

# Risoluzioni

## La base della risoluzione

la risoluzione è un processo iterativo che, ad ogni passo, confronta (risolve) due clausole genitori e permette di inferire una nuova clausola. Per la risoluzione si considerano due clausole che contengono la stessa formula atomica, una volta affermata e una volta negata. La risolvente è ottenuta combinando tutte le formule atomiche delle clausole genitori eccetto quelle che cancella. Tale risoluzione vale nella logica preposizionale.

## L'universo di Herbrand

Nel caso della logica dei predicati, c'è il problema di dimostrare che un insieme finito S di clausole è insoddisfacibile, ossia bisogna dimostrare che non esiste nessuna interpretazione che lo soddisfi.
Herbrand ha dimostraro che è possibile enumerare una adeguata lista di nomi per gli elementi del dominio, tale che se mostriamo che non esiste una interpretazione soddisfacente su domini i cui elementi possono essere nominati dai nomi di quella lista, ciò equivale a mostrare che non esiste nessuna interpretazione soddisfacente.
Universo di Herbrand:
è la lista dei nomi adeguati a un insieme S di clausole. In pratica H(S) è il dominio più generale, e se si dimostra che S è insoddisfacibile sul dominio H(S), si è sicuri che è insoddisfacibile su tutti i domini.
L'universo di Herbrand è infinito ma numerabile e quindi i suoi elementi possono essere ordinati secondo un criterio.
Base di Herbrand:
è l'insieme degli esempi base di tutte le formule atomiche di S, ottenuti usando l'universo di Herbrand per nominare gli elementi del dominio (gli elementi della base sono detti *atomi*). Una interpretazione di H(S) è completa per tutte le clausole di S quando ad ogni atomo della base si assegna un valore di verità.
Anche la base è un insieme numerabile e quindi ordinabile → si può costruire quindi un ALBERO SEMANTICO, che è un albero binario che parte da un nodo radice ed ha i rami etichettati con ciascun atomo, assunto una volta con il valore Vero e un'altra con il valore Falso.
Una *interpretazione* per l'insieme S si ottiene percorrendo un cammino dalla radice ad una foglia.
In generale un albero semantico ha cammini infiniti, è da notare però che non è indispensabile percorrere cammini infiniti per determinare che certe interpretazioni non soddisfano l'insieme di clausole.
Un albero si dice chiuso se tutti i cammini sono interrotti. Un albero semantico di un insieme insoddisfacibile S di clausole è chiuso per S e contiene un numero finito di nodi al di sopra dei nodi di fallimento (Teorema di Herbrand). Provare l'insoddisfacibilità mediante lo sviluppo dell'albero semantico non è pratico, però se applicando ripetutamente la risoluzione, si arriva all'*insieme vuoto*, si verifica subito l'insoddisfacibilità.
Si dimostra che il principio di risoluzione è corretto e completo:

- corretto: se la clausola vuota viene prodotta, l'insieme originario è sicuramente insoddisfacibile
- completo: se l'insieme originario è insoddisfacibile, la clausola vuota viene prima o poi prodotta

## Risoluzione nella Logica Proposizionale

Procedimento:

1. trasformare tutte le proposizioni in F in forma a clausole
2. Negare S e trasformare il risultato in forma a clausole. Aggiungerlo all'insieme di clausole del passo 1
3. Ripetere fino a quando viene trovata una contraddizione o non si può più andare avanti

Algoritmo di unificazione:
nella logica preposizionale è facile verificare la contraddizione di due formule atomiche: L e $\neg$L ; nella logica dei predicati entrano in gioco i legami delle variabili.
L'unificazione confronta due formule atomiche per determinare se esiste un insieme di sostituzioni che le rende identiche.
IDEA: immaginare che ogni formula atomica sia rappresentata da una lista.
Si controllano i primi elementi, se uguali si procede e si continua il confronto.
REGOLE : 1) costanti, funzioni e predicati si possono unificare solo se identici
2) una variabile può unificarsi con: variabile, costante o funzione
COMPLICAZIONE: la stessa sostituzione deve valere per l'intera formula atomica
OSSERVAZIONE: l'obiettivo è una sostituzione che causi l'unificazione di due parti letterali. Si possono avere più sostituzioni.

**Logica dei predicati: la sostituzione**
Una sostituzione è la serie di coppie ordinate:
$\alpha = \{ t_1/y_1, t_2/y_2, \ldots t_n/y_n \}$
dove t sono termini, le y sono variabili e $t_i \neq y_i$ per ogni i.
Quando una sostituzione $\alpha$ è applicata ad una espressione k, ogni occorrenza di $y_i$ in k è sostituita da $t_i$. L'espressione risultante $K_\alpha$ è detta istanza di k

**Composizione di sostituzioni**
Siano α e β due sostituzioni. Si dice composizione di α e β (α o β) la sostituzione tale che
$K(\alpha \circ \beta) = (k\ \alpha)\ \beta$ per ogni espressione K
In pratica si applica prima α e poi β.

- PASSO 1: da α e β si costruiscono i set $\lambda_1$, $\lambda_2$ e $\lambda_3$
- PASSO 2: si deriva la composizione α o β come: $\alpha \circ \beta = \lambda_1 - \lambda_2 - \lambda_3$

PROPRIETA':

1. $(\alpha \circ \beta) \circ \gamma = \alpha \circ (\beta \circ \gamma)$
2. $\varepsilon \circ \alpha = \alpha$ (ε sostituzione vuota)
3. $\alpha \circ \varepsilon = \alpha$ (identità a sinistra e a destra)
4. $\alpha \circ \beta \neq \beta \circ \alpha$ (solitamente → non commutativa)

## Unificazione

Le espressioni $K_1$ e $K_2$ sono unificabili se e solo se esiste una sostituzione γ tale per cui $K_1\ \gamma = K_2\ \gamma$. La sostituzione γ è detta UNIFICATORE. $K_1\gamma$ e $K_2\gamma$ sono istanze comuni delle due espressioni
Unificatori più generali:
Un unificatore δ delle espressioni $K_1$ e $K_2$ è detto *mgu* (most general unifier) se e solo se, per ogni unificatore γ di $K_1$ e $K_2$, l'istanza comune $K_i\ \delta$ è più generale che l'istanza comune $K_i\ \gamma$. In altre parole, $K_i\ \gamma$ è una istanza di $K_i\ \delta$.
ESEMPI DI ESPRESSIONI CHE NON POSSONO ESSERE UNIFICATE:
P(x) con P(f(x)) , P(f(x)) con P(x) , P(x) con Q(x) , P(f(x) con P(g(x))

## Risoluzione nella Logica dei Predicati

Due formule atomiche sono contraddittorie se l'una può essere unificata con la negazionedell'altra. Si applica quindi l'unificazione prodotto dall'algoritmo di unificazione per generare la clausola risolvente.
Algoritmo di risoluzione:
dato un insieme F di enunciati ed un enunciato S da provare

1. Trasformare tutti gli enunciati di F in forma a clausole
2. Negare S e trasformare il risultato in forma a clausole. Aggiungerlo all'insieme di clausole ottenuto al passo 1
3. Ripetere i passi seguenti, fino a quando non viene trovata una contraddizione, oppure non si può più procedere, oppure è già stata utilizzata una predeterminata quantità di risorse:
    a. Selezionare 2 clausole (genitori)
    b. Risolvere insieme
    c. Se la risolvente è la clausola vuota, allora è stata trovata una contraddizione, sennò aggiungerla all' insieme di clausole

Strategie nella scelta delle clausole:

- Selezionare coppie di clausole che contengono formule atomiche complementari
- Eliminare subito le clausole generate che non possono partecipare a risoluzioni successive (tautologie,clausole sussunte da altre)
- Adoperare la *strategia dell'insieme di supporto*: se possibile, risolvere con una delle clausole che è parte dell'enunciato che si sta tentando di refutare
- Adoperare la *strategia di preferenza per le clausole unitarie*: se possibile, risolvere con clausole con una sola formula atomica

Migliorie:
funzioni computabili, predicati computabili e relazioni di uguaglianza possono migliorare l'efficienza della risoluzione. In questi casi alla regola di base si possono aggiungere 2 modi per generare clausole:

- Sostituire un valore con un altro uguale
- Ridurre i predicati computabili

Osservazione:
il metodo di risoluzione riduce il numero di sostituzioni rispetto a quelle date dal teorema di Herbrand, ma non elimina la necessità di tentare più di una sostituzione
Risposta a domande:
non è sufficiente una risposta SI/NO, ma occorre "riempire caselle vuote"

# Ragionamento non-monotono

## Motivazioni:

Spesso la conoscenza sul mondo di cui si dispone è incompleta e per derivare conclusioni plausibili è necessario fare delle assunzioni; tali assunzioni sono scelte facendo riferimento a regole generali sulle proprietà degli oggetti.

Il ragionamento non-monotono si occupa di come conclusioni plausibili (ma non infallibili) possono essere derivate da una base di conoscenza (insieme di formule). Le assunzioni vengono assunte in assenza di informazioni contrarie.

**Altre motivazioni: CWA**

Nella teoria dei databases le informazioni negative non si esplicitano, quindi se un fatto positivo non è nel DB, si assume che valga la sua negazione (CWA)

**Inadeguatezza della logica classica**

La logica classica è monotona, per affermare una regola generica, è necessario rappresentare una lista completa di tutte le eccezioni.

**Il Frame Problem:**

E' il problema di rappresentazione di mondi dinamici. Come rappresentare gli aspetti del mondo che rimangono invariati rispetto a certi cambiamenti di stato. Nella logica classica occorre descrivere esplicitamente tutto ciò che non cambia. Serve quindi una forma di ragionamento che permetta di trattare le eccezioni.

**Default Logic:**

Permette regole di inferenza non standard per esprimere le proprietà che valgono per default.

In generale sono permesse default rules del tipo:

$$\frac{\alpha(x) : \beta(x)}{\gamma(x)}$$ va letta come: se $\alpha(x)$ vale e $\beta(x)$ può essere assunto in modo consistente, allora posso concludere $\gamma(x)$

Dato un insieme di formule (una KB), generalmente incompleta, le default rules permettono di completare (estendere) la KB → ESTENSIONI

**Default Theories:**

una default theory è una coppia <D,W> dove D è un insieme di default rules e W è un insieme di formule del I ordine. Esempio:

$$D: \left\{ \frac{uccello(x) : vola(x)}{vola(x)} \right\}$$

W: { uccello (Tweety) $\forall$x (pinguino(x) → ¬ vola (x)) }

Se nell'insieme W si aggiunge pinguino(Tweety), si blocca l'applicazione della default rule in D.

Se l'insieme D ha due default rules, può succedere che ci siano 2 ESTENSIONI, una con vera la prima, l'altra con vera la seconda.

**Logica non monotona**

Permette ci cancellare e aggiungere enunciati alla base dati

**Ragionamento probabilistico**

Rende possibile rappresentare inferenze probabili ma incerte

**Logica sfumata (fuzzy logic)**

Permette di rappresentare proprietà di oggetti continue (non binarie) o sfumate

**Concetto di spazi di credenza**

Permette di rappresentare modelli di insieme di credenze inseriti l'uno nell'altro

**Ragionamento non monotono**

È basato sull'assunto: "se x non può essere provato (entro un determinato tempo (logica indecidibile)), allora concludere Y"

## Cancellazione enunciati

Nel sistema non monotono gli enunciati possono essere cancellati o aggiunti alla base dati. Se si cancella un enunciato, può succedere di dover cancellare tutti gli enunciati la cui prova è dipesa dal primo.

Nel controllo delle incompatibilità si ottiene un albero di ricerca orientato con ritorno all'indietro, e se si rileva una incompatibilità, si elimina l'assunzione iniziale, più tutte quelle inferite da questa.

**Sistema di ragionamento non monotono TMS (Truth Maintenance System)**

È un processo di ritorno all'indietro guidato dalla dipendenza. Ogni enunciato viene chiamato nodo, e potrà essere in uno dei due stati seguenti:

- IN : creduto vero
- OUT : non creduto vero

Ad ogni nodo si associa una lista di giustificazioni:

- Nodi IN : hanno almeno una giustificazione valida
- Nodi OUT : non hanno alcuna giustificazione valida

Per poter modellare situazioni diverse, le giustificazioni possono essere costituite da:

- Liste di supporto: (SL (nodi_in) (nodi_out))
- Prove condizionali: (CP conseguente (ipotesi_in) (ipotesi_out))

**Liste di supporto**

Le giustificazioni basate su SL sono valide se tutti i nodi elencati nella lista (nodi_in) sono IN e tutti quelli in (nodi_out) sono in OUT (attualmente).

Nomenclatura: se la lista (nodi_out) in SL non è vuota, il nodo è detto assunzione.

**Prove condizionali**

Permettono di rappresentare argomentazioni ipotetiche, valide se il nodo conseguente è IN ogni volta che sono IN i nodi in (ipotesi_in) e sono OUT i nodi in (ipotesi_out). TMS le trasforma in giustificazioni SL (più facili da manipolare).

# Ragionamento Probabilistico

## Motivazioni

- Il mondo rilevante è realmente casuale
- Dato un numero sufficiente di informazioni il mondo rilevante nonè casuale, ma il nostro programma non sempre avrà accesso a tutti i dati necessari
- Il mondo sembra casuale perché non lo abbiamo descritto al livello giusto

Considerazione metodologica: nei primi due casi l'approccio probabilistico è corretti, nel terzo è meglio salire di livello per una descrizione più adeguata.

## Teoria delle probabilità

Si usa per correlare eventi che dipendono da altri eventi

Probabilità congiunta:

se A e B sono indipendenti: prob(A e B) = prob(A) * prob(B)

Valutazione probabilistica:

Punteggio $= \sum^{N}_{i=1}$ prob(i)*valore(i) dove N è il numero dei possibili eventi

Teorema di Bayes:

$$P(H_i \mid E) = \frac{P(E \mid H_i) * P(H_i)}{\sum^{k}_{n=1} P(E|H_n) * P(H_n)}$$

Permette di calcolare la probabilità di un certo evento a partire da un insieme di osservazioni effettuate

## Problemi deterministici trattati in modo probabilistico

Sono problemi complessi quando manca informazione sufficiente. Una possibile soluzione e calcolare una funzione lineare, pesando ogni contributo. Il problema connesso è valutare i pesi, e si usa una procedura di addestramento. Per affrontare i problemi legati alle influenze dei pesi, può essere opportuno passare ad una

RAPPRESENTAZIONE BASATA SU REGOLE. Occorre poi associare ad ogni regola una probabilità. Il sistema di regole può essere accresciuto inserendo nuove regole o modificandolo: è dinamico, oltre che strutturato.

Conclusione: un sistema di più alto livello conduce la ricerca con l'algoritmo MiniMax nell'albero del gioco. A più basso livello ci si basa sulle regole

Vantaggi:

- si possono trattare con le stesse modalità e in modo integrato situazioni in cui la conoscenza è incompleta e/o vi sono situazioni intrinsecamente probabilistiche
- si modella meglio il mondo reale
- associando la valutazione probabilistica a delle regole il sistema mantiene capacità esplicative

**Soluzione MYCIN:**

- sistema basato su regole
- usa il ragionamento inesatto
- è destinato alla diagnostica medica
- interagisce con TEIRESIAS

MYCIN opera in un dominio di CONOSCENZA INCERTA,usa il ragionamento all'indietro, parte dalla meta e usa molte regole. In alternativa a Bayes, usa due indici:

1. MB misura di CREDENZA (è la diminuzione proporzionale della non credenza in un'ipotesi h). In pratica misura quanto l'evidenza avvalora l'ipotesi e vale 0 se non la avvalora.
2. MD misura di NON CREDENZA (è la diminuzione proporzionale della credenza in h). Misura quanto l'evidenza avvalora la negazione dell'ipotesi ed è 0 se l'evidenza rende certa l'ipotesi.

Da questi indici si ricava:

- CF fattore di INCERTEZZA : CF[h,e] = MB[h,e] – MD[h,e]
- Se CF > 0 il sistema desume che l'ipotesi è vera
- Se CF < 0 il sistema desume che l'ipotesi è falsa

OSSERVAZIONI:

- struttura di una regola
  1. ipotesi
  2. collezione di elementi di evidenza

3. fattore di certezza

- vengono usate regole complesse (con più elementi di evidenza) piuttosto che regole semplici.
- Sorgono dei problemi deduttivi se, rispetto ad un fatto, si inserisce una regola che descrive una relazione causale e un'altra che descrive una relazione causale inversa

SOLUZIONI:

- Usare un solo tipo di regola
- Partizionare i due tipi in modo che non inferiscano

# Logica modale e Logica intenzionale

Una logica modale contiene dei connettivi come possibile e vincolato:

- (possibile p) è vero se p è possibile (inteso che le cose avrebbero potuto andare in un modo diverso dalla realtà in cui sono andate)
- (vincolato p q) significa che la verità di p comporta inevitabilmente la verità di q.

La logica classica non è compatibile con i seguenti connettivi:

- Non distingue tra una affermazione falsa (ma possibilmente vera) ed una necessariamente falsa
- Non tratta il caso in cui una affermazione falsa potrebbe, se vera, vincolarne un'altra ad essere vera a sua volta.

MONDO POSSIBILE: introdotto da Kripke, è un mondo in cui le cose avrebbero potuto manifestarsi, il mondo reale è uno dei mondi possibili. Quindi (possibile p) è vera se p è vera in un mondo possibile.
Si introduce (conosce a p), ovvero "a conosce che p", la cui semantica viene fornita attraverso un insieme di mondi possibili, fra cui quelli compatibili con ciò che a conosce. (conosce a p) indica che p è vero in tutti i mondi compatibili con ciò che a conosce
CONDIZIONALI CONTROFATTUALI: sono nella forma " se p fosse vero, q sarebbe vero"
ESTENSIONE: è l'oggetto denotato da un termine (sono corrispondenti a insiemi)
INTENSIONE: è il significato di un termine, è una funzione che specifica la sua estensione in ogni mondo possibile. (sono corrispondenti a formule logiche).
Gli studiosi di logica ritengono che estensione ed intensione devono essere separate e generare due tipi di teorie, una sugli insiemi (collezioni di oggetti), ed una sui predicati-come-oggetti.

## La logica di Ordine superiore

La logica del primo ordine ha come referente i predicati, proprietà asserite circa il mondo. Se si estende la logica permettendo asserzioni sui predicati, si ha una logica di ordine superiore.
Si possono operare in essa quantificazioni sui predicati stessi: per esempio si può definire l'uguaglianza asserendo "due oggetti sono diversi se e solo se differiscono per almeno una proprietà".
Le logiche di ordine superiori possono però portare facilmente a paradossi.

# Pattern recognition

## Concetti di base

Col termine di *pattern* si intende genericamente una descrizione di un oggetto. Le descrizioni possono provenire da fenomeni fisici catturati mediante sensori, oppure essere frutto di processi di astrazione. Qui interessano quelli del primo tipo (figure, suoni, ecc.). Applicazioni tipiche:

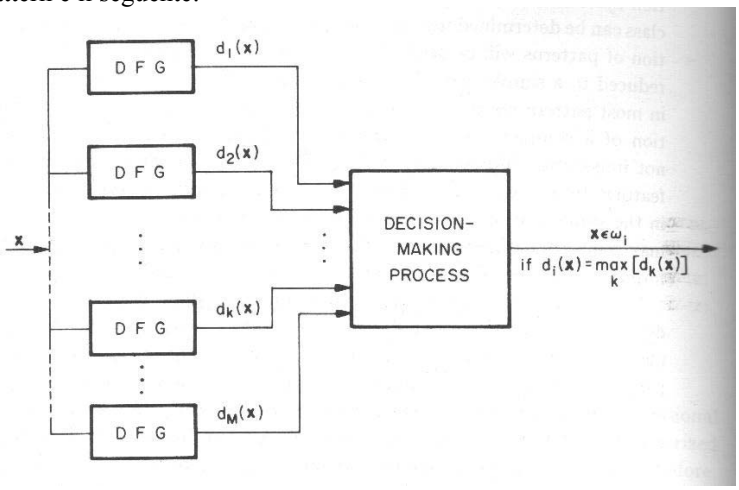
| Task of Classification | Input Data | Output Response |
|---|---|---|
| Character recognition | Optical signals or strokes | Name of character |
| Speech recognition | Acoustic waveforms | Name of word |
| Speaker recognition | Voice | Name of speaker |
| Water prediction | Water maps | Water forecast |
| Medical diagnosis | Symptoms | Disease |
| Stock market prediction | Financial news and charts | Predicted market ups and downs |

Una classe di pattern è una categoria determinata da alcuni attributi comuni. Riconoscere è pertanto ricondotto all'attribuizione di un pattern d'ingresso ad una data classe. Un pattern è dato mediante la misura degli attributi che lo costituiscono, solitamente in forma vettoriale:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{pmatrix}$$

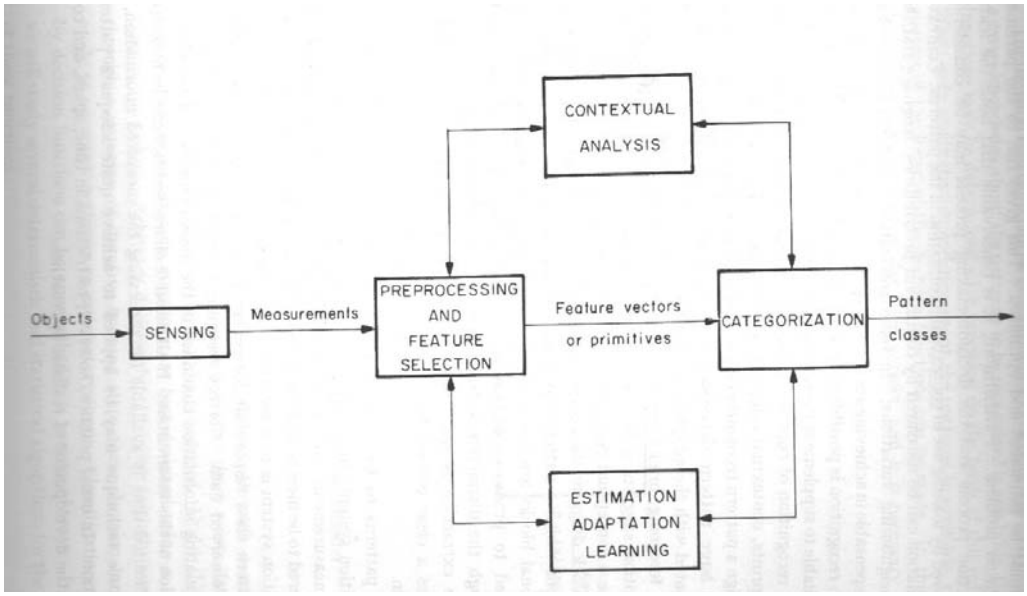
## Problemi implicati

Se le misure degli attributi sono forniti mediante numeri reali, è usuale pensare i vettori-pattern come punti in uno spazio Euclideo n-dimensionale. Altro oggetto di studio è l'estrazione di caratteristiche distintive (*features*) o attributi dai dati in input. Quest'attività è solitamente accompagnata da una riduzione di dimensionalità del vettore del pattern. Si parla di preelaborazione ed estrazione delle caratteristiche (*prepocessing* and *features extraction* problem). Per il riconoscimento e la classificazione è necessario ancora disporre di una procedura per determinare la decisione ottima (criteri e algoritmi). Il problema è ricondotto alla determinazione di superfici di separazione (*boundaries*) tra le classi a partire dall'osservazione di un certo numero di vettori. Se, ad esempio, si usa un criterio di decisione che fornisce un indice di somiglianza o vicinanza del pattern d'ingresso rispetto a ciascuna classe, lo schema di un classificatore di pattern è il seguente:




**DFG** → decision function generator

Associato al problema di preelaborazione ed estrazione delle caratteristiche è la stima e l'ottimizzazione dei parametri del riconoscitore (addestramento dei modelli). Quando il sistema di riconoscimento deve essere resistente alle distorsioni, flessibile rispetto ad una grande deviazione dei pattern e capace di aggiustare da sè i parametri occorre affrontare il problema dell'adattamento.




**Funzioni di decisione**

Il compito principale di un sistema di riconoscimento è determinare la classe di appartenenza dei pattern in analisi. Occorre stabilire le regole su cui basare queste decisioni. Un possibile approccio è l'uso di funzioni di decisione. Questo approccio si può estendere al caso in cui ci sono più di due classi, e si può scegliere come funzione discriminante qualsiasi funzione dello spazio euclideo, non necessariamente lineare.

*Funzioni di decisione lineari*

Una funzione di decisione lineare nel caso più generale di *n* dimensioni assume la forma:

$$d(\bar{x}) = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_{n+1} = \bar{w}'\bar{x} + w_{n+1}$$

$\bar{w}$ è detto vettore dei pesi o dei parametri.

Per convenzione si aggiunge una componente unitaria (cioè un 1) al vettore pattern e il peso $w_{n+1}$ al vettore dei pesi, per cui:

$$\bar{x} = (x_1, x_2, \ldots, x_n, 1)'$$
$$\bar{w} = (w_1, w_2, \ldots, w_n, w_{n+1})' \quad \text{e} \quad d(\bar{x}) = \bar{w}'\bar{x}$$

*Caso 1:*

Le classi sono separabili mediante singole superfici di decisione. In questo caso si ha:

$$d_i(\bar{x}) = \bar{w}_i'\bar{x} \begin{cases} > 0 \; se \; \bar{x} \in \omega_i \\ < 0 \; \text{altrimenti} \end{cases} \text{ con } i=1,2,\ldots,M$$

dove $w_i = (w_{i1}, w_{i2}, \ldots, w_{i,n+1})'$ è il vettore pesi associato con la i-esima funzione di decisione.

*Caso 2:*

Ogni classe è separabile da ciascuna altra mediante una distinta superficie di decisione, cioè le classi sono separabili a coppie. In questo caso ci sono $\frac{M(M-1)}{2}$ superfici di decisione (la combinazione di M classi prese a due a due).

La forma delle funzioni di decisione: $d_{ij}(\bar{x}) = \bar{w}'_{ij}\bar{x}$

Se $\bar{x}$ appartiene alla classe $\omega_i$ allora $d_{ij}(\bar{x}) > 0$ per tutti i $j \neq i$.

Proprietà di queste funzioni: $d_{ij}(\bar{x}) = -d_{ji}(\bar{x})$

*Caso 3:*

Esistono M funzioni di decisione $d_k(\bar{x}) = \bar{w}'_k\bar{x}$ con *k=1,2,...M* con la proprietà che se $\bar{x}$ appartiene alla classe $\omega_i$, sia $d_i(\bar{x}) > d_j(\bar{x})$ per tutti i $j \neq i$

Può essere considerato una particolarizzazione del caso 2, poiché si può definire:

$$d_{ij}(\bar{x}) = d_i(\bar{x}) - d_j(\bar{x})$$
$$= (\bar{w}_i - \bar{w}_j)'\bar{x}$$
$$= \bar{w}'_{ij}\,\bar{x}$$

Si può verificare facilmente che se $d_i(\bar{x}) > d_j(\bar{x})$ per tutti i $j \neq i$, allora si verifica anche che $d_{ij}(\bar{x}) > 0$ per tutti i $j \neq i$: se una classe è separabile con le condizioni del caso 3, lo è anche con quelle del caso 2 (il viceversa non è generalmente vero).

**Funzioni di decisione generalizzate**

I confini tra le classi non sono sempre esprimibili mediante piani. Si può stabilire una funzione di discriminazione non lineare introducendo opportune funzioni $f_i(\bar{x})$, così:

$$d(\bar{x}) = w_1 f_1(\bar{x}) + w_2 f_2(\bar{x}) + \ldots + w_k f_k(\bar{x}) + w_{k+1}$$

Le $f_i(\bar{x})$ sono funzioni reali a singolo valore. Poiché una volta valutate le $f_i(\bar{x})$ forniscono un solo valore, si può porre:

$$\bar{x}^* = \begin{pmatrix} f_1(\bar{x}) \\ f_2(\bar{x}) \\ \vdots \\ f_k(\bar{x}) \\ 1 \end{pmatrix}$$

e riscrivere l'espressione precedente: $d(\bar{x}) = \bar{w}'\bar{x}^*$

Generalizzazione della forma quadratica: $d(\bar{x}) = \sum_{j=1}^{n} w_{jj} x_j^2 \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} w_{jk} x_j x_k + \sum_{j=1}^{n} w_j x_j + w_{n+1}$

Generalizzazione per funzioni polinomiali di ordine r: $f_i(\bar{x}) = x_{p_1}^{S_1} x_{p_2}^{S_2} \ldots x_{p_r}^{S_r}$ con $p_1, p_2, \ldots p_r = 1,2,\ldots,n$ e $s_1, s_2, \ldots, s_r = 0,1$

Forma recursiva per ricavare d(x): $d^r(\bar{x}) = \sum_{p_1=1}^{n} \sum_{p_2=p_1}^{n} \ldots \sum_{p_r=p_r-1}^{n} w_{p_1 p_2 \ldots p_r} x_{p_1} x_{p_2} \ldots x_{p_r} + d^{r-1}(\bar{x})$

dove *r* è il grado di non linearità e $d^0(\bar{x}) = w_{n+1}$

**Lo spazio dei pattern e lo spazio dei pesi**

Si supponga che esistano solo due classi $\omega_1$ e $\omega_2$, ciascuna rappresentata da due punti $(x_1^1, x_2^1)$ e $(x_1^2, x_2^2)$. Se le classi sono linearmente separabili esiste una funzione $d(\bar{x})$ che, se è $d(\bar{x}) > 0$ per una classe è $d(\bar{x}) = 0$ per l'altra. Quindi il vettore dei pesi $\bar{w} = (w_1, w_2, w_3)'$ deve soddisfare le relazioni:

$$w_1 x_{11}^1 + w_2 x_{12}^1 + w_3 > 0$$
$$w_1 x_{21}^1 + w_2 x_{22}^1 + w_3 > 0$$
$$w_1 x_{11}^2 + w_2 x_{12}^2 + w_3 < 0$$
$$w_1 x_{21}^2 + w_2 x_{22}^2 + w_3 < 0$$

$\bar{w}$ è pertanto la soluzione del sistema di disuguaglianze.

Si preferisce porre il sistema moltiplicando per -1 le ultime due disuguaglianze:

$$w_1 x_{11}^1 + w_2 x_{12}^1 + w_3 > 0$$
$$w_1 x_{21}^1 + w_2 x_{22}^1 + w_3 > 0$$
$$-w_1 x_{11}^2 - w_2 x_{12}^2 - w_3 > 0$$
$$-w_1 x_{21}^2 - w_2 x_{22}^2 - w_3 > 0$$

Si osservi che le *x* sono i coefficienti e le *w* le variabili. Il luogo dei punti che soddisfa le disuguaglianze è detto *spazio dei pesi*.

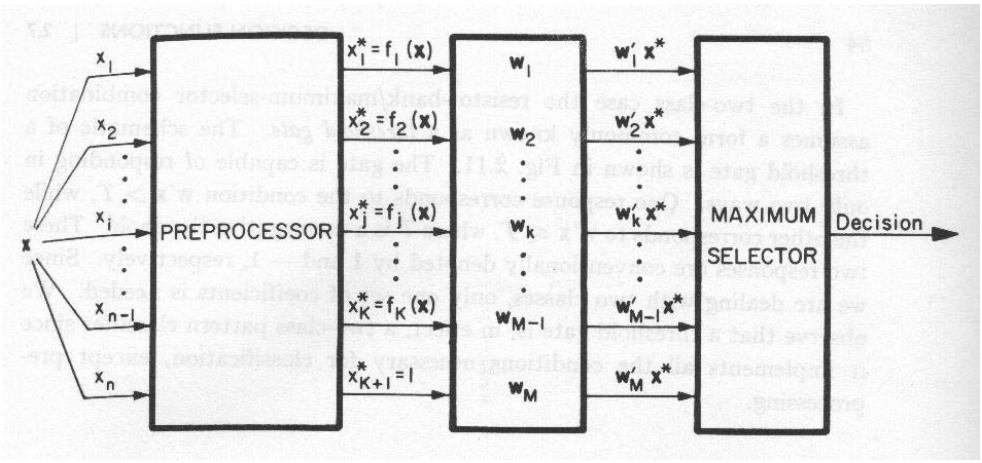
*Osservazione:*
Lo spazio dei pattern è uno spazio euclideo n-dimensionale. In questo spazio, $\overline{w}$ è un insieme di coefficienti che determinano una superficie di decisione. Lo spazio dei pesi è uno spazio euclideo *(n+1)-dimensionale.* In questo spazio, ogni disuguaglianza rappresenta il lato positivo o negativo di un iperpiano che passa per l'origine (con equazioni tipo $w_1 x_{11}' + w_2 x_{12}' + w_3 = 0$).
La regione delle soluzioni è caratterizzata dall'essere una figura conica (più precisamente, un *cono poliedrico convesso*).

**Dicotomia**
Una misura del potere discriminante delle funzioni di decisione è il numero di modi con cui possono classificare un dato set di pattern. Si può dimostrare che il numero di dicotomie lineari di N punti in uno spazio euclideo n-dimensionale, se i punti sono ben distribuiti, è dato da:

$$D(N,m) = \begin{cases} 2\sum_{k=0}^{n} C_k^{N-1} \; se \; N > n+1 \\ 2^N \; se \; N \le n+1 \end{cases} \text{ con } C_k^{N-1} = \frac{(N-1)!}{(N-1-k)!k!}$$

N.B.: $D(N,m)$ cresce rapidamente al crescere di N e n!

**Realizzazione delle funzioni di decisione**
Se la determinazione dei parametri delle funzioni di decisione è complessa, la realizzazione di classificatori multiclasse è relativamente semplice. Un possibile diagramma a blocchi, ipotizzando di trattare una multiclasse di tipo 3 ( $d_i(\overline{x}) > d_j(\overline{x})$ per tutti i $j \neq i$ ) è il seguente:




Se la funzione di decisione è di tipo qualsiasi, il preprocessor realizza il calcolo necessario in:

$$d(\overline{x}) = w_1 \underline{f_1(\overline{x})} + w_2 \underline{f_2(\overline{x})} + ... + w_k \underline{f_k(\overline{x})} + w_{k+1}$$

Se la funzione di valutazione è lineare si può avere una realizzazione semplice e poco costosa in hardware: per ogni classe si utilizza una rete resistiva. Se poi si hanno solo due classi, il sistema si semplifica ulteriormente mediante l'uso di "threshold gate".

**Classificazione di pattern mediante funzioni di distanza**
Per stabilire una misura di similarità fra pattern, che possono essere considerati punti in uno spazio euclideo, si può ricorrere al concetto di prossimità. A volte è intuitivo stabile se un punto appartiene ad una classe e questo vale quando i punti di una classe sono aggregati in "cluster". Se questo non accade si possono avere delle difficoltà nella classificazione.

**Classificazione a minima distanza**
È usata quando le classi dei pattern presentano un limitato grado di variabilità. Si possono verificare varie situazioni, a seconda di come si possono rappresentare le classi:

*Singoli prototipi:* Ogni classe è rappresentata da un unico elemento (prototipo). È il caso in cui la variabilità dei pattern e altre fonti di disturbo possono essere trascurate. Esempio: riconoscimento di caratteri OCR ben stampati.
La funzione di decisione è:

$$d_i(\overline{x}) = \overline{x}' \overline{z}_i - \frac{1}{2} \overline{z}_i' \overline{z}_i \text{ con } i = 1,2,...,M$$

dove il pattern $\overline{x}$ è assegnato alla classe $\omega_i$ se $d_i(\overline{x}) > d_j(\overline{x})$ per tutti i $j \neq i$

*Conclusioni:* il classificatore a minima distanza è un particolare caso si classificatore lineare. Poiché questa classificazione è basata sull'accoppiamento (*match*) tra un pattern e il prototipo di classe più vicino, questo approccio è detto anche *correlazione* oppure *cluster watching*.

*Prototipi multipli:* Le considerazioni fatte nel caso di singoli prototipi si possono estendere al caso in cui i pattern di ciascuna classe $\omega_i$ tendano ad aggregarsi ("clusterizzare") intorno ad un set di prototipi $z_i^1, z_i^2, ..., z_i^{N_i}$, dove $N_i$ è il numero di prototipi nella i-esima classe.
La funzione di decisione è:

$$d_i(\bar{x}) = \max_l \left\{ \left( \bar{x}' \bar{z}_i^l - \frac{1}{2} \left(\bar{z}_i^l\right)' \left(\bar{z}_i^l\right) \right) \right\} \text{ con } l = 1,2,...,N_i$$

e $\bar{x}$ è assegnata alla classe $\omega_i$ se $d_i(\bar{x}) > d_j(\bar{x})$ per tutti i $j \neq i$.

L'approccio basato sulla minima distanza è un caso particolare di una forma più generale di classificatori a spezzate (*piecewise linear classifier*). Le funzioni di decisione in questo caso assumono la forma:

$$d_i(\bar{x}) = \max_l \left\{ d_i^{\,l}(\bar{x}) \right\} \text{ con } i = 1,2,...,M \text{ e } l = 1,2,...,N_i$$

dove ciascuna distanza è:

$$d_i^{\,l}(\bar{x}) = w_{i_1}^l x_1 + w_{i_2}^l x_2 + .... + w_{i_n}^l x_n + w_{i,n+1}^l x = \bar{w}_i^l \bar{x}$$

### Estensione del concetto di minima distanza

Nel caso più generale, si ha un set di pattern di classificazione noto $\{\bar{s}_1, \bar{s}_2, ..., \bar{s}_N\}$ dove si assume che ciascun pattern appartiene ad una delle classi $\omega_1, \omega_2, ..., \omega_M$.
Si definisce regola di classificazione dell'intorno più vicino (*nearest neighbor*, *NN*) quella che assegna un vettore incognito $\bar{x}$ alla classe del vicino più prossimo, dove si dice che $\bar{s}_i (\in \{\bar{s}_1, \bar{s}_2, ..., \bar{s}_N\})$ è il vicino più prossimo (nearest neighbor) se:

$$D(\bar{s}_i, \bar{x}) = \min_l \{ D(\bar{s}_l, \bar{x}) \} \text{ con } l = 1,2,...,N$$

Poiché si usa un solo vicino per la classificazione, questo schema è detto anche "*1-NN rule*". Se, invece, si considerano *q* vicini a $\bar{x}$ e si adotta un criterio di maggioranza, la regola è detta "*q-NN rule*" (si assegna $\bar{x}$ alla maggioranza dei *q* vicini più prossimi). Come è intuitivo, in questo modo si riduce la probabilità di errata attribuzione.

### Formazione dei cluster

Gioca un ruolo fondamentale per la classificazione basata sul criterio di minima distanza. La definizione dei cluster di dati deve essere fondata di una misura di similarità. Un criterio, già introdotto, è quello basato sulla distanza definita come: $D = \|\bar{x} - \bar{z}\|$. Il criterio stabilisce che "minore distanza → più grande similarità". Ovviamente non è l'unica definizione di distanza che si possa adottare. Se ad esempio si tiene conto di proprietà statistiche, una definizione di distanza molto usata è quella detta "distanza di Mahalanobis":

$$D = (\bar{x} - \bar{m})' \, \bar{C}^{-1} (\bar{x} - \bar{m})$$

dove $\bar{x}$ è una variabile pattern
$\bar{m}$ rappresenta il vettore medio
$\bar{C}$ è la matrice di covarianza di una popolazione di pattern.

Le misure di similarità non sono necessariamente ristrette alle misure di distanza. È infatti importante rilevare dai dati (e dal problema) l'attributo - o gli attributi - comuni che rendono simili i simili e differenziano i dissimili. Se, ad esempio, le regioni di cluster tendono a distendersi lungo degli assi, si può scegliere come misura di similarità l'angolo tra i vettori (o un parametro a questo correlato). Ad esempio:

$$S(\bar{x}, \bar{r}) = \frac{\bar{x}' \bar{r}}{\|\bar{x}\| \, \|\bar{r}\|}$$

fornisce il coseno tra i vettori $\bar{x}$ e $\bar{r}$, ovvero la proiezione di uno sull'altro. Nelle applicazioni di recupero dell'informazione (information retrieval), di nosologia e di tassonomia si usa una variante detta misura Tanimoto così definita:

$$S(\bar{x}, \bar{r}) = \frac{\bar{x}' \bar{r}}{\bar{x}' \bar{x} + \bar{r}' \bar{r} - \bar{x}' \bar{r}}$$

che ha l'ovvia interpretazione di rapporto tra elementi comuni ed elementi dissimili nei due vettori.

### Criteri di clustering

Dopo aver adottato una misura di similarità, occorre fissare il criterio di aggregazione: può essere basato su uno schema euristico oppure sulla minimizzazione (o massimizzazione) di qualche indice di prestazione. Ad esempio, ricade nel primo gruppo il criterio basato sulla distanza euclidea: infatti la vicinanza di due pattern è una misura "relativa" di similarità, per cui occorre stabilire ancora una soglia al fine di definire il grado di accettabilità di un pattern come simile ad un altro (o appartenente ad un cluster).

Nel secondo gruppo ricade, ad esempio, il criterio di minimizzare l'errore quadratico medio definito come:

$$J = \sum_{j=1}^{N_c} \sum_{\bar{x} \in S_j} \left\| \bar{x} - \overline{m}_j \right\|^2 \text{ dove:}$$

$N_c$ è il numero dei cluster

$S_j$ è il set di campioni appartenenti al j-esimo cluster

$\overline{m}_j = \frac{1}{N_j} \sum_{x \in S_j} \bar{x}$ è il vettore medio del set $S_j$

Esistono anche algoritmi ibridi, che mescolano l'ottimizzazione rispetto a qualche indice con l'euristica.

**Semplice algoritmo di clusterizzazione**

- Si abbiano N pattern $\{\bar{x}_1, \bar{x}_2, ..., \bar{x}_N\}$.
- Si sceglie un pattern a caso come centro del primo cluster, cioè come $\bar{z}_1$.
- Si sceglie ancora una soglia non negativa *T*.
- Si supponga di aver scelto $\bar{x}_1$ come $\bar{z}_1$ (il primo del set).
- Si calcola poi la distanza di $\bar{x}_2$ da $\bar{z}_1$: sia $D_{21}$ questa distanza.
- Se $D_{21} > T$, allora $\bar{x}_2$ è assunto come $\bar{z}_2$, il centro del secondo cluster, altrimenti si scarta come appartenente al primo cluster.
- Si supponga $D_{21} > T$: si calcola allora $D_{31}$ e $D_{32}$, rispettivamente la distanza di $\bar{x}_3$ da $\bar{z}_1$ e $\bar{z}_2$. Se sono entrambe >*T*, allora $\bar{x}_3$ è il centro del terzo cluster, altrimenti si assegna al cluster più prossimo.

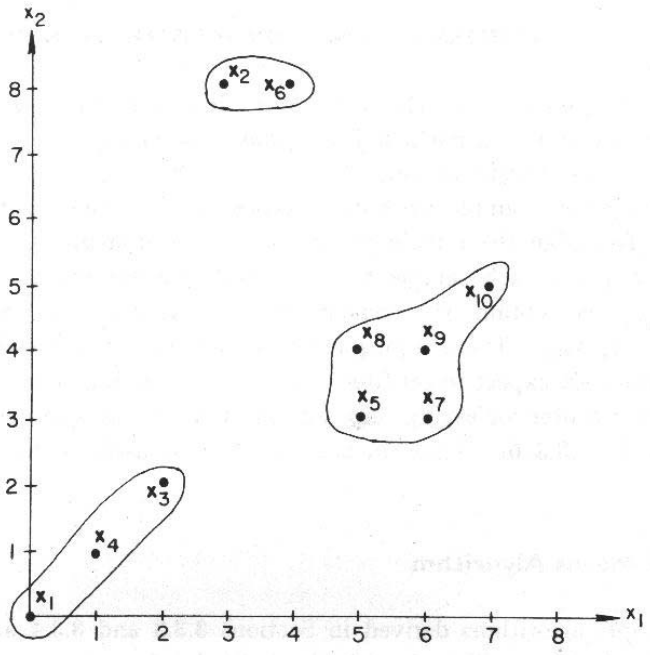
Si procede così per tutti gli altri pattern del set. Il risultato dipende dalla scelta iniziale, dall'ordine con cui si analizzano i pattern e dalla soglia T. Nonostante i limiti evidenti, è un algoritmo semplice, rapido (il set di pattern è visitato una volta sola), può dare indicazioni di massima sulla disposizione dei dati. Nella pratica, si usano alcuni accorgimenti aggiuntivi:

- Si prova con diversi valori di soglia;
- si considerano le distanze tra i centri dei cluster e il numero di campioni che cadono in ciascun cluster;
- si considera la distanza dal centro del più vicino e del più lontano pattern per ciascun cluster, nonché la varianza.

Questa procedura è efficace particolarmente quando i dati esibiscano un'aggregazione caratteristica "a sacchettini" (o "nuvolette").

**Algoritmo Maximin-Distance**

È simile al precedente, ma cerca prima i cluster più lontani. Si pongono i campioni in una tabella (a sinistra). All'inizio la tabella di destra è vuota.




1. Si sceglie un campione a caso, ad esempio $\bar{x}_1$, come primo centro di cluster, e lo si designa come $\bar{z}_1$.
2. A questo punto si seleziona il punto più lontano, $\bar{x}_6$, come secondo centro di cluster $\bar{z}_2$.
3. Si calcola la distanza degli altri punti rispetto a $\bar{z}_1$ e $\bar{z}_2$, memorizzando la minima tra le due.
4. Si seleziona la massima tra queste distanze minime: se è una frazione apprezzabile della distanza tra $\bar{z}_1$ e $\bar{z}_2$, per esempio la metà, si sceglie il campione corrispondente come centro di cluster $\bar{z}_3$ (nell'esempio $\bar{x}_7$).
5. Si procede così, selezionando sempre la massima delle distanze minime. Se questa distanza è una frazione apprezzabile delle "tipiche" distanze massime precedenti (per esempio si sceglie il valore medio come valore "tipico"), si designa il pattern corrispondente come nuovo centro di cluster, altrimenti l'algoritmo si ferma.
6. Alla fine si assegnano i pattern al cluster di appartenenza e, per avere un valore più rappresentativo, si sceglie il valore medio calcolato all'interno del cluster come centro $\bar{z}_i$.

**Algoritmo k-Means**

È basato sul criterio di minimizzare un indice di prestazione, definito come somma dei quadrati delle distanze tra tutti i punti di ciascun cluster rispetto al proprio centro di cluster. È un algoritmo basato sui seguenti passi:

1. Si scelgono *k* centri di cluster iniziali in modo arbitrario, e si designano come $\bar{z}_1(1), \bar{z}_2(1), \ldots, \bar{z}_k(1)$. Ad esempio, si scelgono i primi *k* campioni.
2. Alla k-esima iterazione, si distribuiscono i campioni $\{\bar{x}\}$ tra i *k* cluster usando la relazione:

$$\bar{x} \in S_J(k) \text{ se } \left\|\bar{x} - \bar{z}_j(k)\right\| < \left\|\bar{x} - \bar{z}_i(k)\right\| \text{ per } i = 1,2,\ldots,k \text{ e } i \neq j$$

$S_J(k)$ indica il cluster che ha come centro $\bar{z}_J(k)$

3. Si calcola per ogni cluster il nuovo centro $\bar{z}_J(k+1)$ con $j = 1,2,\ldots,k$ in modo che sia minimizzata la distanza quadratica di tutti i punti del cluster rispetto al centro. In formule, si vuole che sia minimo l'indice di prestazione:

$$J_J = \sum_{\bar{x} \in S_J} \left\|\bar{x} - \bar{z}_J(k+1)\right\|^2 \text{ con } j = 1,2,\ldots,k .$$

È noto che basta calcolare il valore medio di $\bar{x}$ : $\bar{z}_J(k+1) = \dfrac{1}{N_J} \sum_{\bar{x} \in S_J(k)} \bar{x}$, donde il nome "k-Means"

4. Se $\bar{z}_J(k+1) = \bar{z}_J(k)$ per $j = 1,2,\ldots,k$, la procedura è terminata, altrimenti si va al passo 2.

Il comportamento di questo algoritmo, di cui non esiste una prova di convergenza di tipo generale, è influenzato da:

- il numero k di centri prescelto
- la scelta iniziale dei centri di cluster

- l'ordine con cui vengono presi i campioni
- le proprietà geometriche (di disposizione) dei dati.

**Algoritmo ISODATA**

L'algoritmo Isodata (Iterative Self-Organizing Data Analysis Technique A[2]) è simile alla procedura K-Means: si aggiungono in più delle procedure euristiche (importante, tra le altre, quella detta di *split and merge*). All'inizio si fissa un numero di cluster iniziali $N_c$ (non necessariamente quello che si desidera come definitivo) e i loro centri (arbitrari) $\bar{z}_1, \bar{z}_2, ..., \bar{z}_{N_c}$.

1. Si specificano i seguenti parametri di processo:
   *K* numero dei centri di cluster desiderati
   $\vartheta_N$ numero di campioni con cui confrontare il numero di campioni in un cluster
   $\vartheta_S$ parametro di deviazione standard
   $\vartheta_C$ parametro di aggregazione di un blocco
   *L* massimo numeri di coppie di centri di cluster che possono essere aggregate
   *I* numero di iterazioni permesse

2. Si distribuiscono gli *N* campioni $\{\bar{x}_1, ..., \bar{x}_N\}$ tra gli attuali centri di cluster in base alla relazione:
   $$\bar{x} \in S_J \text{ se } \left\|\bar{x} - \bar{z}_j\right\| < \left\|\bar{x} - \bar{z}_i\right\|, \text{ con } i = 1,2,...,N_c; i \neq j$$

3. Si scaricano i cluster con meno di $\vartheta_N$ campioni; formalmente:
   Se per un *j* è $N_j < \vartheta_N$, allora si elimina $S_j$ e si riduce $N_c$ di 1.

4. Si ricalcolano i centri dei cluster $\bar{z}_j$, $j = 1,2,...,N_c$, ponendoli uguali alla media dei campioni in ciascun cluster,
   $$\bar{z}_j = \frac{1}{N_j} \sum_{\bar{x} \in S_j} \bar{x}, \text{ con } j = 1,2,...,N_c$$

5. Si calcola la distanza media: $\bar{D}_j = \frac{1}{N_j} \sum_{\bar{x} \in S_j} \left\|\bar{x} - \bar{z}_j\right\|$, con $j = 1,2,...,N_c$

6. Si calcola la media pesata delle distanze medie: $\bar{D} = \frac{1}{N} \sum_{j=1}^{N_c} N_j \bar{D}_j$

7.
   *a)* Se questa è l'ultima iterazione porre $\vartheta_C = 0$ e andare al passo 11.
   *b)* Se $N_c \leq k/2$ (pochi cluster rispetto al previsto), andare al passo 8 (fase di *split*).
   *c)* Se questa è un'iterazione pari oppure $N_c \geq 2k$ (troppi cluster), andare al passo 11 (fase di *merge*); altrimenti continuare.

8. Calcolare il vettore di deviazioni standard $\bar{\sigma}_j = (\sigma_{1j}, \sigma_{2j}, ..., \sigma_{nj})$ per ciascun cluster, secondo la relazione $\sigma_{ij} = \sqrt{\frac{1}{N_j} \sum_{\bar{x} \in S_j} (x_{ik} - z_{ij})^2}$, $i = 1,2,...,n$, $j = 1,2,...,N_c$ dove:

   n è la dimensionalità dei vettori $\bar{x}$
   $x_{ik}$ è la i-esima componente del k-esimo campione di $S_j$
   $z_{ij}$ è la i-esima componente del j-esimo centro di cluster

   In pratica, il vettore $\bar{\sigma}_j$ rappresenta la deviazione standard dei campioni in $S_j$ lungo gli assi principali delle coordinate.

9. Si cerca la componente massima di ciascun $\bar{\sigma}_j$, con $j = 1,2,...,N_c$ e si denota come $\sigma_{j\max}$

10. Se per ogni $\sigma_{j\max}$ ci si trova di fronte a questa situazione:

[2] La A finale è aggiunta per comodità di pronuncia

$$\sigma_{j\max} > \vartheta_S \quad \text{e} \quad \begin{cases} a)\overline{D}_j > \overline{D} \text{ e } N_j > 2(\vartheta_N + 1) \\ \text{oppure} \\ b)\ N_C \le K/2 \end{cases}$$

si spezza $\bar{z}_j$ in due cluster $\bar{z}_j^{+}$ e $\bar{z}_j^{-}$.

Il nuovo centro di cluster $\bar{z}_j^{+}$ è ricavato da $\bar{z}_j$ sommando una data quantità $\gamma_j$ alla componente di $\bar{z}_j$ che corrisponde alla massima componente di $\overline{\sigma}_j$;

per $\bar{z}_j^{-}$ invece $\gamma_j$ viene sottratto. $\gamma_j$ in pratica viene calcolato come $\gamma_j = k\sigma_{j\max}$ con $0 \le k \le 1$.

L'obiettivo è quello di creare una perturbazione, cioè una differenza sensibile nella distanza tra un generico campione e i due nuovi centri di cluster, senza tuttavia modificare la disposizione degli altri cluster in modo apprezzabile. Se avviene lo "splitting", si va al passo 2 altrimenti si continua; (è evidente che la $\bar{z}_j$ originaria scompare, $N_C$ aumenta di 1)

11. Si calcolano le distanze a coppie $D_{ij}$ tra i centri dei cluster: $D_{ij} = \left\| \bar{z}_i - \bar{z}_j \right\|$, $i = 1,2,...,N_C - 1$ e $j = 1,2,...,N_C$
12. Si confrontano le distanze $D_{ij}$ con il parametro $\vartheta_C$. Si ordinano le *L* più piccole distanze che sono minori di $\vartheta_C$ in ordine crescente: $[D_{i_1 j_1}, D_{i_2 j_2}, ..., D_{i_L j_L}]$ (si ricordi che *L* è il numero massimo di centri di cluster che si possono aggregare).
13. A partire dalla distanza più piccola $D_{i_e j_e}$ si aggregano i centri a coppie $\bar{z}_{ie}$ e $\bar{z}_{je}$ (a condizione che $\bar{z}_{ie}$ e $\bar{z}_{je}$ non siano già stati usati per una aggregazione). In pratica da $\bar{z}_{ie}$ e $\bar{z}_{je}$ si crea un nuovo centro come somma pesata dei due (i pesi sono il numero di campioni in ogni cluster):

$$\bar{z}_e^{*} = \frac{1}{N_{i_e} + N_{j_e}} \left[ N_{i_e} \cdot (\bar{z}_{i_e}) + N_{j_e} \cdot (\bar{z}_{j_e}) \right]$$

( Praticamente $\bar{z}^{*}$ è il baricentro)

Si cancellano quindi $\bar{z}_{ie}$ e $\bar{z}_{je}$, riducendo $N_C$ di 1. Si osservi che, poiché ogni centro di cluster è usato al più una volta, non è detto che si riescano a fare *L* aggregazioni.

14. Se questa è l'ultima iterazione, fine. Altrimenti andare al passo 2 (è previsto che si possa andare anche al passo 1 per modificare qualche parametro, se l'utente lo ritiene opportuno per una migliore adattatività).

**Algoritmo di quantizzazione vettoriale**

Si può pensare come un algoritmo derivato da K-MEANS e da ISODATA.

1. All'inizio tutti i vettori sono attribuiti ad un unico cluster. Si calcola l'unico rappresentante come valor medio di tutti i vettori;
2. Per ciascun rappresentante finora trovato:
   a. si crea una perturbazione (casuale o meglio ancora lungo la direzione di massima dispersione) in modo da determinare due nuovi rappresentanti;
   b. si assegna ciascun vettore del rappresentante padre ad uno dei due figli, secondo il criterio della minima distanza;
   c. si ricalcolano i nuovi rappresentanti come il vettore medio dei vettori assegnati a ciascun figlio: saranno quelli i nuovi centri di cluster (si abbandonano i vettori "perturbati").
3. Se *non* è soddisfatto il "criterio di fine", si va a 2.

Alla fine i vettori rappresentanti, detti *codeword*, sono etichettati con un intero da 0 a $2^N - 1$, dove N è il numero di bit utilizzato per codificare i vettori. Il "criterio di fine" può essere legato:

- al raggiungimento del numero di bit N stabilito a priori per la codifica;
- al superamento di una soglia minima di "distorsione" accettata, se si è definito un opportuno indice di distorsione riferita all'applicazione che si sta trattando.

NB: sostituire i vettori veri con i rispettivi rappresentanti può essere considerato come una *distorsione* dello spazio vero; ovvero, se si stanno trattando sequenze di vettori, ciascuno dei quali è un punto nello spazio, al *pattern* (traiettoria) vero si sostituisce un *pattern distorto*, quello che è costretto a passare per i punti (vettori) rappresentanti piuttosto che per i punti veri. In quest'ultimo caso il numero di bit N utilizzato per la codifica non è stabilito a priori, ma è quello trovato.

*Vantaggi:*

- non è richiesto di assegnare a priori i K rappresentanti iniziali come negli algoritmi primitivi K-MEANS e ISODATA;
- è basato su criteri oggettivi (geometrici) e non su euristiche (come, ad esempio, ISODATA)

*Limiti:*

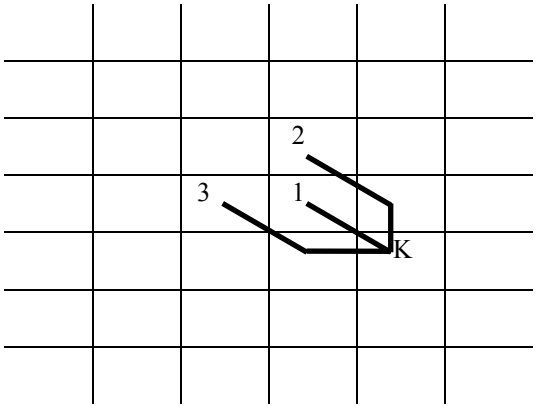
- fissare a priori il numero N di bit su cui rappresentare i codici può essere inefficiente o inadeguato
- trattare tutto lo spazio dei vettori alla stessa stregua può essere inefficiente o inadeguato (qui non si effettuano "aggiustamenti" se un cluster è poco rappresentativo o troppo ricco)

*Campi di utilizzo:*

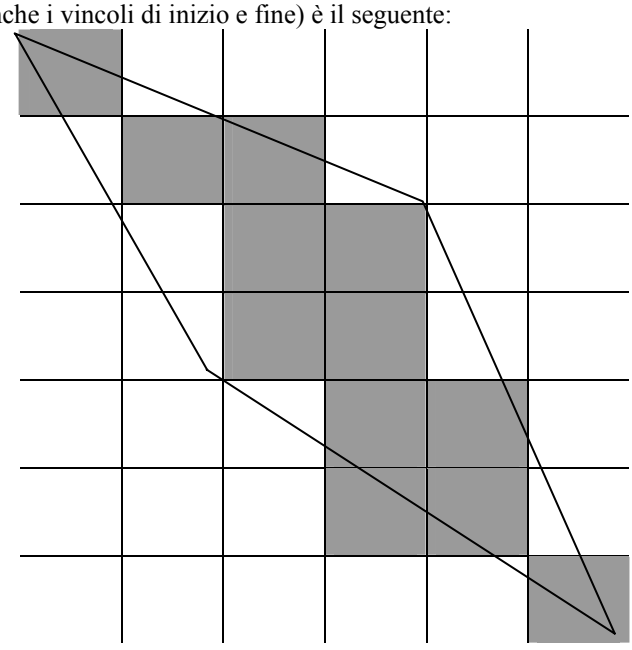
- dove serve una *riduzione di dimensionalità* (compressione di banda per qualsiasi applicazione)
- dove serve passare da variabili continue a variabili simboliche (associando etichette ai codici)

# Programmazione dinamica

La programmazione dinamica ha lo scopo di confrontare delle sequenze di eventi non identici, all'interno della problematica più generale del confronto di modelli. Per esempio le sequenze di eventi possono essere delle stringhe da confrontare come nel caso di comandi impartiti ad un interprete. Per determinare la maggiore o migliore somiglianza (decidere per esempio se accettare o rifiutare un comando e come interpretarlo) è necessario considerare un percorso ed i relativi costi associati. Si ricorda che avendo un percorso ottimo, anche ogni suo sottoinsieme è ottimo (principio di ottimalità di Bellman). Sostanzialmente, avendo un punto finale è possibile considerare facendo backtrack quali sono i percorsi per arrivare a quel punto con i relativi costi associati. Osservando i percorsi all'indietro ci si può trovare in 3 situazioni: *sostituzione*, *cancellazione*, *inserzione* (è anche possibile considerare situazioni composte). Più praticamente:
Per ogni punto K, si ipotizza di essere arrivati da un numero limitato di punti: il costo del punto K è quindi valutato come il costo del punto di partenza + il costo del salto al punto di arrivo, pesato con un coefficiente che tiene conto del fatto che si effettua un'inserzione, una cancellazione o una sostituzione. In realtà, si deve tener conto che anche nell'inserzione e nella cancellazione si ha una sostituzione (nel punto K).
La formula è: coeff $*$ distanza(**d**, **o**)
dove *coeff* è 1.0 nel caso di sostituzione, e un valore maggiore nel caso di cancellazione o inserzione. Essendo stati posti dei vincoli nei punti di partenza, nei punti di arrivo e nei salti possibili, i punti di valutazione cadono in uno spazio limitato. Ad esempio, se i "passi" permessi sono i seguenti:



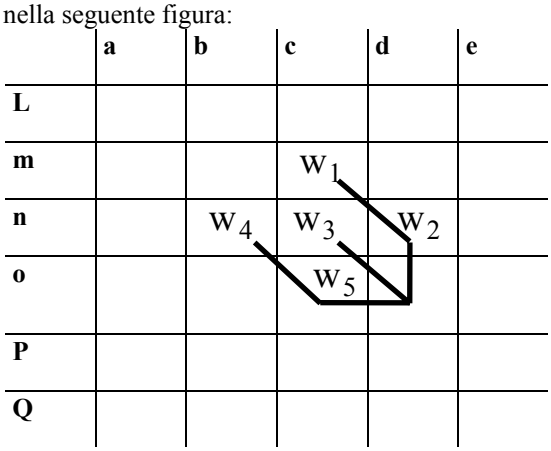

lo spazio che viene esplorato (dati anche i vincoli di inizio e fine) è il seguente:



Si pone così una limitazione allo spazio di ricerca, ma anche alle possibili "distorsioni".

**Determinazione dei costi**

Per i pesi, si può usare uno schema come nella seguente figura:




$$C(i,j) = \min \begin{cases} C(i-2,j-1) + w_1 \cdot dist(i-1,j) + \\ w_2 \cdot dist(i,j) \\ C(i-1,j-1) + w_3 \cdot dist(i,j) \\ C(i-1,j-2) + w_4 \cdot dist(i,j-1) + \\ w_5 \cdot dist(i,j) \end{cases}$$

Le pesature possono essere di 2 tipi diversi: simmetrica e asimmetrica.

*Applicazioni:*

Solitamente si ha a disposizione una serie di "modelli" (nel nostro esempio, la forma corretta dei comandi) e si utilizza la programmazione dinamica per il confronto. Si confronta la stringa da analizzare con *tutti* i modelli e si assegna la stringa al modello che ha fornito il miglior indice di accoppiamento. Usualmente l'assegnamento è condizionato con il superamento di una soglia di accettazione: se la stringa è distante da tutti i modelli, la si rigetta.

(NB: non è detto che questo metodo vada bene: anche qui sarebbe meglio l'ipotesi di "mondo chiuso"!).

# Rappresentazione della Conoscenza

**Esigenze:**
rappresentare:
4) Fatti complessi
5) Oggetti complessi
6) Scenari
7) Sequenze di eventi

**Caratteristiche:**
- Adeguatezza rappresentativa
- Adeguatezza inferenziale : inferire nuova conoscenza
- Efficienza inferenziale : poter incorporare informazioni in più
- Efficienza nell'acquisizione

**Classificazione delle tecniche:**
- Metodi *dichiarativi*: conoscenza rappresentata come collezione statica di fatti, affiancata da un piccolo insieme di procedure generali per la manipolazione. Vantaggi:
  - Ogni fatto va immagazzinato solo una volta
  - È facile aggiungere nuovi fatti al sistema
- Metodi *procedurali*: la conoscenza viene rappresentata come procedure per il suo uso. Vantaggi:
  - È facile rappresentare la conoscenza su come fare le cose
  - È facile rappresentare il ragionamento per default e quello probabilistico
  - È facile rappresentare la conoscenza euristica

NB: la maggior parte dei sistemi usa una combinazione dei due metodi

**Schemi:**
la descrizione delle strutture di conoscenza avviene mediante schemi
Definizione: il termine schema si riferisce ad un'organizzazione attiva di relazioni (esperienze) passate, che si deve sempre supporre siano operanti in ogni risposta organica adattiva
Tipi di schemi:
4. Frame: per descrivere una collezione di attributi
5. Script: per sequenze di eventi comuni
6. Stereotipi: per insiemi di caratteristiche
7. Modelli a regole: per caratteristiche comuni

Problemi dell'uso degli schemi:
- Esistono proprietà di oggetti così di base?
- A quale livello dovrebbe essere rappresentata la conoscenza?
- Come si può accedere alle parti rilevanti per propri scopi?

**Strutture di conoscenza basilari:**
4. Oggetti complessi → composizione di oggetti semplici
5. Classi complesse → composizione di classi più piccole

Primitive relazionali:
- Relazione ISA: categorizzazione, tassonomia gerarchica
- Relazione PARTE_DI: esprime la composizione strutturale

Sono entrambi ordinamenti parziali all'interno di un dominio
Proprietà di queste relazioni:
- Transitività: se A ISA B e B ISA C, allora A ISA C
- Ereditarietà delle proprietà: le proprietà dell'oggetto più generale, sono anche di quello più specifico

**Livello di rappresentazione:**
le primitive ISA e PART_OF non permettono di descrivere le azioni, servono altre primitive. Conclusione: è meglio trasformare tutti gli enunciati in una rappresentazione costruita su un piccolo insieme di primitive, quasi in forme canoniche.

**Argomenti contro l'uso di primitive a basso livello:**
- Trasformare eventi complessi in primitive a basso livello comporta molto lavoro spesso non necessario
- È richiesto grande spazio di memoria per le rappresentazioni

**Accesso alle strutture di conoscenza:**
varie proposte:
- Indicizzare le strutture mediante le parole del contenuto (dipendenza concettuale)
- Considerare ogni parola del testo come puntatore a tutte le strutture in cui potrebbe essere contenuta. Data una frase, occorre fare l'intersezione per avere la struttura che copre tutte le parole del contenuto. Controindicazioni:
  - Se è presente una parola leggermente estranea, l'intersezione è vuota

- Il calcolo di tutti gli insiemi di possibilità e poi dell'intersezione è oneroso
- Individuare una parola chiave del testo e selezionare una struttura iniziale. Controindicazioni:
  - Come identificare la parola chiave?
  - Come decidere l'importanza relativa delle parole chiave?

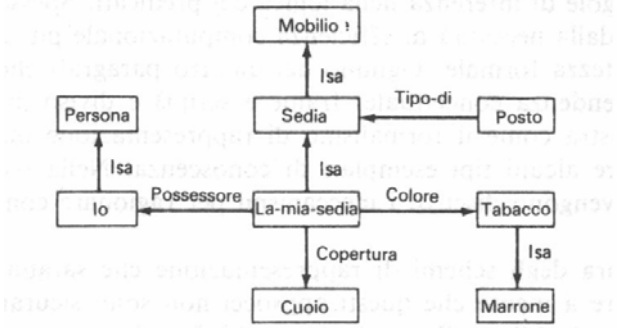
## Strutture per rappresentare la conoscenza

Sono basate su di una idea comune: le entità complesse sono rappresentate come collezione di attributi-valori.
Operativamente sono basate su:

- Memorie associative
- Liste di proprietà

## Reti semantiche

Esempio:




- → Progettate all'inizio per rappresentare il significato delle parole nel linguaggio naturale
- → L'informazione è rappresentata come insieme di nodi connessi da archi le cui etichette rappresentano le relazioni fra i nodi
- → Gli archi sono unidirezionali
- → Operativamente sono rappresentate da strutture di memoria basate sulla coppia attributo-valore
- → Le reti semantiche sono in stretta correlazione con la logica dei predicati

Manipolazione delle reti semantiche:
si può usare il metodo detto "ricerca dell'intersezione": si fa partire un processo di attivazione da due nodi e si vede dove l'attivazione si incontra. Questo metodo "mima" i processi cognitivi: è stato abbandonato a favore di metodi più diretti.
Problemi connessi alle rappresentazioni con reti semantiche:

- conviene immagazzinare il concetto nella forma più generale possibile. Per tener conto dei casi specifici si può introdurre un legame del tipo ESEMPLARE-DI
- per risolvere il problema della quantificazione, conviene partizionare la rete semantica in un insieme gerarchico di spazi

**Algoritmi per l'ereditarietà**
Algoritmo (di principio) dell'ereditarietà
Per ottenere il valore V dell'attributo A di un esemplare O:

1. si trova O nella base di conoscenza
2. se c'è un valore per l'attributo A, si riporta tale valore
3. sennò si vede se c'è un valore per l'attributo ESEMPLARE-DI, se non c'è → FALLIMENTO e TERMINA
4. sennò ci si sposta sul nodo corrispondente al valore dell'attributo ESEMPLARE-DI e si cerca un valore per l'attributo, se c'è si riporta il valore
5. sennò si ripetono le seguenti operazioni finchè o non c'è nessun valore per l'attributo ISA o si determina una risposta:
   a. ci si sposta sul nodo corrispondente al valore dell'attributo ISA
   b. si vede se è presente un valore per l'attributo A, se si, viene riportato il valore

Limiti dell'algoritmo: dubbio su cosa fare quando c'è più di un valore per l'attributo ESEMPLARE-DI o per l'attributo ISA
Problema: l'algoritmo non riesce a rilevare intrinseche ambiguità di alcune rappresentazioni: si potrebbe pensare di risolvere il problema inserendo un controllo relativo alla lunghezza dei cammini. La lunghezza però non sempre corrisponde al livello di generalità, quindi si può definire il risultato dell'ereditarietà come: l'insieme dei valori in competizione per una casella S in un frame F contiene tutti quei valori che possono essere derivati da qualche frame X che sta sopra F nella gerarchia ISA, e non sono contraddetti da qualche frame Y la cui distanza inferenziale da F sia più corta di quella esistente tra X e F

Algoritmo generale di eredità delle proprietà
Per determinare un valore V per una casella S di un esemplare F si esegue:

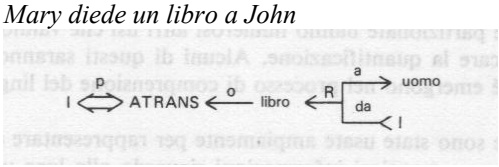
- si pone CANDIDATI a vuoto
- si esegue una ricerca in ampiezza o in profondità a partire da F, verso l'alto nella gerarchia ISA seguendo tutti gli archi. Ad ogni passo si guarda se c'è memorizzato un valore per S:

    - se si trova un valore, lo si aggiunge a CANDIDATI e si termina quel ramo della ricerca
    - se non si trova alcun valore ma ci sono archi ESEMPLARE-DI e ISA verso l'alto, si seguono
    - sennò si termina il ramo
- per ogni elemento C di CANDIDATI
    - si guarda se c'è qualche altro elemento di CANDIDATI che è derivato da una classe più vicina ad F della classe dalla quale proviene C
    - se c'è, si rimuove C da CANDIDATI
- si controlla la cardinalità di CANDIDATI
    - se è 0 si segnala che non è stato trovato alcun valore
    - se è 1 si riporta come V l'unico elemento di CANDIDATI
    - se è maggiore di 1 si riporta una contraddizione

## La dipendenza concettuale

- è intesa a modellare il processo dinamico di comprensione di una frase
- è indipendente dalla lingua
- la sintassi e la semantica sono integrate
- è basata su un limitato numero di azioni primitive

Esempio:
*Mary diede un libro a John*




I simboli hanno il seguente significato:

- le frecce indicano la direzione di dipendenza
- la doppia freccia indica il doppio legame fra l'attore e l'azione
- la freccia a tre linee (non nell'esempio) indica causalità
- p indica il tempo passato
- ATRANS è una delle azioni primitive usate dalla teoria. Indica il passaggio di possesso
- o indica la relazione di oggetto
- R indica la relazione di ricevente

Azioni primitive:

- ATRANS esempio: dare
- PTRANS es: andare
- PROPEL es: spingere
- MOVE es: dare un calcio
- GRASP es: mangiare
- EXPEL es: piangere
- MTRANS es: raccontare
- MBUILD es: decidere
- SPEAK es: dire
- ATTEND es: ascoltare

Costituiscono i blocchi elementari e sono i nuclei intorno ai quali è incentrata la concettualizzazione.
Una concettualizzazione è la descrizione di un evento mediante una azione primitiva più un insieme di casi ad essa collegati
Modificatori:

- *p* passato
- *f* futuro
- *t* transizione
- $t_s$ inizio della transizione
- $t_f$ fine della transizione
- *k* in corso
- *?* interrogativo
- */* negativo
- *nil* presente
- *delta* senza tempo
- *c* condizionale

Vantaggi:

- Sono necessarie meno regole di inferenza grazie alle primitive

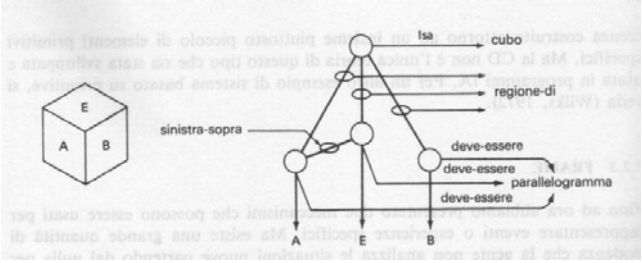
- Molte inferenze sono già contenute nella rappresentazione stessa
- La struttura iniziale avrà dei buchi da riempire, buchi che possono servire per focalizzare l'attenzione del programma

Svantaggi:

- Le primitive sono tutte a basso livello, possono causare inefficienza
- Permette di rappresentare solo eventi

## Frame:

Esempio:




- Sono strutture di rappresentazione a caselle da riempire, adatte per situazioni complesse
- Idea Base (Minsky): quando si affrontano nuove situazioni, si fa uso di conoscenze preesistenti
- Utilizza una collezione di caselle (slot) che descrivono aspetti degli oggetti e che possono essere riempite con altri frame. Ad uno slot possono essere associati insiemi di condizioni da soddisfare o informazioni procedurali quali:
    9) procedura if-added: descrivono che cosa si deve fare quando lo slot è riempito
    10) procedure if-needed o to-establish: descrivono come si può calcolare, se richiesto, l'elemento da usare per il riempimento
- Una situazione complessa è rappresentata da frame correlati, ossia un sistema di frame

Frame:

- I frame sono strutture complesse che ricalcano la complessità delle strutture concettuali
- La struttura a frame può essere usata per il riconoscimento, i frame contengono attributi degli oggetti che devono essere veri
- I frame descrivono esemplari tipici dei concetti che rappresentano
- Uso dei frame:
    - Si utilizza una parte di evidenza parziale per selezionare un frame
    - Si cerca di riempire gli slot vuoti
    - Se c'è fallimento, le sue modalità possono dare informazioni su quale altro frame selezionare
    - Si usano i legami specifici tra frame per individuare le direzioni da esplorare
    - Si può risalire la struttura gerarchica in cui eventualmente sono inseriti i frame

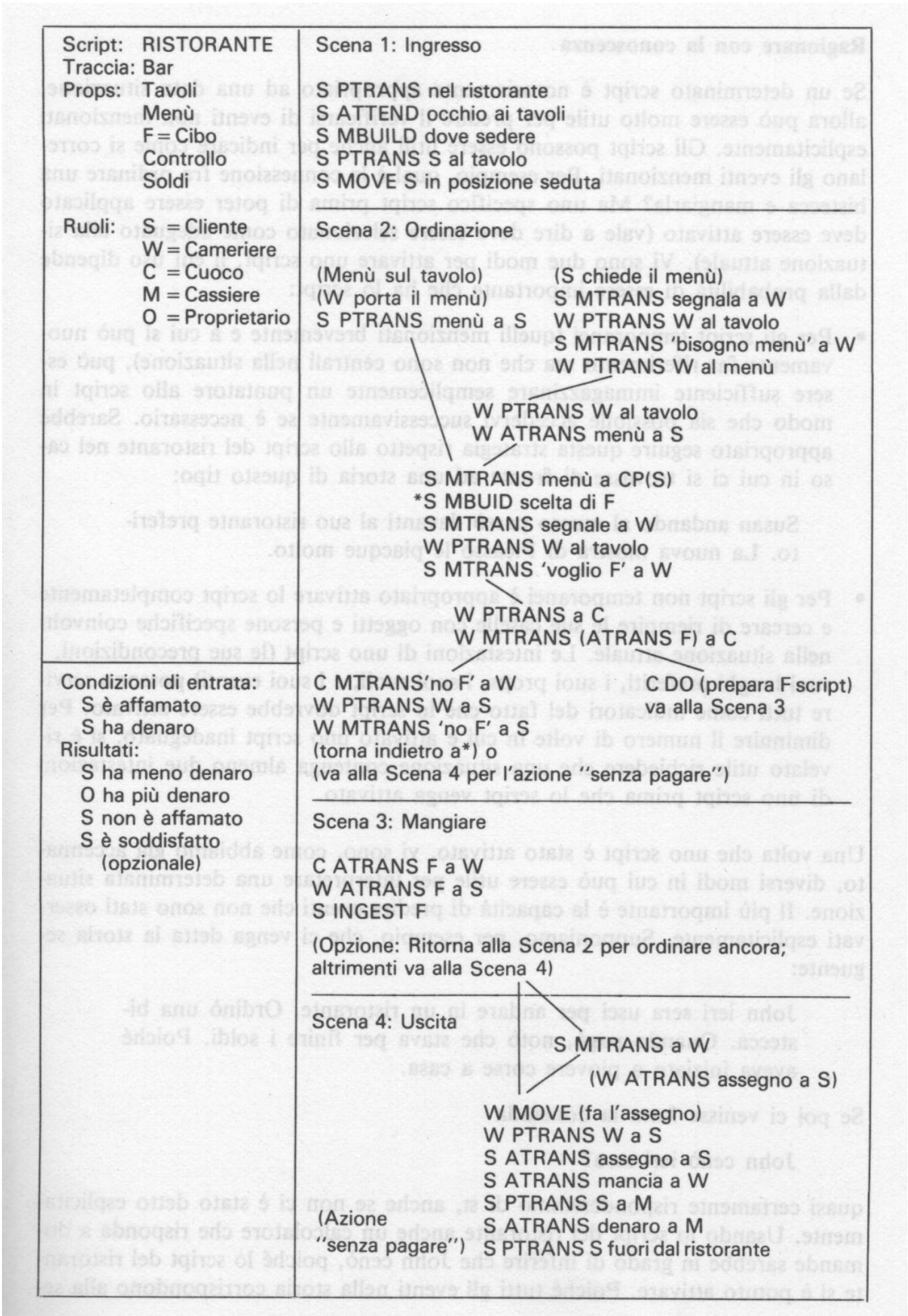
## Script:

- Descrive una sequenza stereotipata di eventi
- È costituita da slot, a cui è associata l'informazione su quali tipi di valori può contenere o i valori di default
- Si differenzia dai frame in quanto ha un ruolo specializzato

Componenti:

- Condizioni di entrata: condizioni che in generale devono essere soddisfatte prima degli eventi dello script
- Risultato: condizioni che in gen. saranno vere dopo il verificarsi degli eventi
- Props: caselle che rappresentano oggetti implicati negli eventi
- Ruoli: caselle che rappresentano persone coinvolte negli eventi
- Traccia: la variazione specifica rispetto ad uno schema più generale
- Scene: le effettive sequenze di eventi che si verificano

Esempio:
Lo script "Ristorante":
gli script descrivono catene causali di eventi e la loro forza risiede nella capacità di predire eventi non osservati esplicitamente. Lo script come i frame contiene meccanismi per la focalizzazione dell'attenzione.


Script: RISTORANTE
Traccia: Bar
Props: Tavoli
Menù
F = Cibo
Controllo
Soldi
Ruoli: S = Cliente
W = Cameriere
C = Cuoco
M = Cassiere
O = Proprietario
Condizioni di entrata:
S è affamato
S ha denaro
Risultati:
S ha meno denaro
O ha più denaro
S non è affamato
S è soddisfatto
(opzionale)
Scena 1: Ingresso
S PTRANS nel ristorante
S ATTEND occhio ai tavoli
S MBUILD dove sedere
S PTRANS S al tavolo
S MOVE S in posizione seduta
Scena 2: Ordinazione
(Menù sul tavolo)
(W porta il menù)
S PTRANS menù a S
(S chiede il menù)
S MTRANS segnala a W
W PTRANS W al tavolo
S MTRANS 'bisogno menù' a W
W PTRANS W al menù
W PTRANS W al tavolo
W ATRANS menù a S
S MTRANS menù a CP(S)
*S MBUID scelta di F
S MTRANS segnale a W
W PTRANS W al tavolo
S MTRANS 'voglio F' a W
W PTRANS a C
W MTRANS (ATRANS F) a C
C MTRANS'no F' a W
W PTRANS W a S
W MTRANS 'no F' a S
(torna indietro a*) o
(va alla Scena 4 per l'azione "senza pagare")
C DO (prepara F script)
va alla Scena 3
Scena 3: Mangiare
C ATRANS F a W
W ATRANS F a S
S INGEST F
(Opzione: Ritorna alla Scena 2 per ordinare ancora;
altrimenti va alla Scena 4)
Scena 4: Uscita
S MTRANS a W
(W ATRANS assegno a S)
W MOVE (fa l'assegno)
W PTRANS W a S
S ATRANS assegno a S
S ATRANS mancia a W
S PTRANS S a M
S ATRANS denaro a M
S PTRANS S fuori dal ristorante
(Azione
"senza pagare")

# Teoria delle probabilità

P($A$): probabilità **incondizionata** (o probabilità **a priori**): quella che si ha *in mancanza di qualsiasi altra informazione*.
Ogni variabile casuale $X$ ha un **dominio** di possibili valori ($x_{1,\dots,}x_n$).
P($A|B$): probabilità **condizionata** o **a posteriori**: probabilità di $A$ dato che *tutto ciò che sappiamo* è $B$.
Vale la **regola del prodotto**:
$P(A \wedge B) = P(A|B)\, P(B)$

**Assiomi della teoria della probabilità:**
1) $0 \le P(A) \le 1$
2) Le proposizioni necessariamente vere (cioè valide) hanno probabilità 1, quelle necessariamente false (cioè insoddisfacibili) hanno probabilità 0; in formule: P(*Vero*) = 1 e P(*Falso*) = 0.
3) $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$ (si desume dal diagramma di Venn).

**Distribuzione di probabilità congiunta:**
Siano $X_1 \dots X_n$ variabili casuali che assumono valori determinati in un dominio, ciascuna con una certa probabilità.
Un **evento atomico** è un'assegnazione di valori particolari a tutte le variabili.
La distribuzione di probabilità congiunta $\mathbf{P}(X_1 \dots X_n)$ assegna probabilità a tutti gli eventi atomici (è una tabella $n$-dimensionale).

**Regola di Bayes:**

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

Se considero E prove di fondo:
$\mathbf{P}(Y|X,E) = \mathbf{P}(X|Y,E)\, \mathbf{P}(Y|E) / \mathbf{P}(X|E)$

**Normalizzazione:**

$$P(M|S) = \frac{P(S|M)P(M)}{P(S|M)P(M) + P(S|\neg M)P(\neg M)}$$

Tale processo è detto normalizzazione, poiché tratta 1/P(S) come costante di normalizzazione che permette ai termini condizionali di dare 1 come somma.
Nel caso generale multivalore, la regola di Bayes diventa:
$\mathbf{P}(Y|X) = \alpha\mathbf{P}(X|Y)\, \mathbf{P}(Y)$

**La combinazione delle prove:**
occorre valutare la probabilità a priori per la causa e le probabilità condizionali per ognuno degli effetti.

**Rete di credenze o Rete bayesiana:**
Una rete di credenze è un grafo per cui valgono le seguenti proprietà:
1. Un insieme di variabili casuali costituiscono i nodi della rete.
2. Un insieme di archi con un verso connette le coppie di nodi. Il significato intuitivo di una freccia dal nodo $X$ al nodo $Y$ è che $X$ ha *un'influenza diretta* su $Y$.
3. Ogni nodo ha una tabella di probabilità condizionate che quantifica gli effetti che i genitori hanno sul nodo. I genitori di un nodo sono tutti quei nodi che hanno frecce che puntano al nodo.
4. Il grafo non ha cicli diretti (dunque si tratta di un grafo diretto aciclico, DAG, Directed Acyclic Graph).

Specificata la topologia, occorre specificare, per ogni nodo, la **tabella delle probabilità condizionate**. Ogni riga della tabella contiene la probabilità condizionata del valore di ogni nodo per un **caso condizionante**.

**Metodo per la costruzione delle reti di credenze:**
la rete di credenze è una rappresentazione corretta del dominio solo se ogni nodo è condizionalmente indipendente dai suoi predecessori nell'ordinamento dei nodi, dati i suoi genitori.
La procedura generale per la costruzione incrementale della rete è la seguente:
1. Si scelga un insieme di variabili rilevanti $X_i$ che descrivono il dominio.
2. Si scelga un ordinamento per le variabili.
3. Finché sono rimaste variabili:
   a) Si prenda una variabile $X_i$ e si aggiunga un nodo alla rete.
   b) Si ponga *Genitori*($X_i$) come un qualche insieme minimo di nodi già presenti nella rete, tale che la proprietà di indipendenza condizionale sia soddisfatta.
   c) Si definisca la tabella delle probabilità condizionate per $X_i$.

Siccome ogni nodo è connesso solo ai nodi precedenti, la rete è aciclica.

# Reti Neurali

**Cosa sono:**
Sono una NUOVA TECNOLOGIA INFORMATICA, hanno:

- Architettura innovativa : non Von Neumann, ma molti semplici processori paralleli, con rete di connessioni distribuita
- Analogia con la struttura del cervello : $10^{10}$ neuroni fortemente connessi da sinapsi
- Nuove prestazioni ottenibili : tempo reale in problemi complessi, autoapprendimento,resistenza a guasti ed errori, degradazione graduale

**Funzionamento:**
Le reti neurali usano una architettura elaborativi che presenta una serie di analogie,nel trattamento delle informazioni, con l'intelligenza naturale:

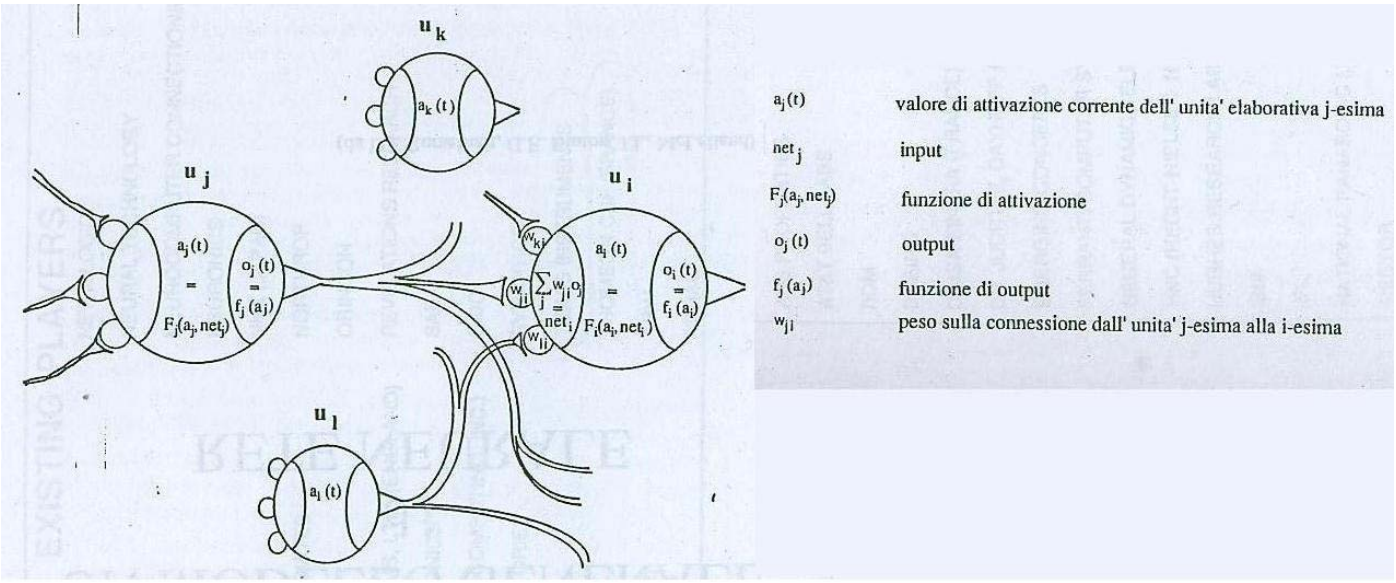
- Forte parallelismo e interconnessione
- Unità elaborative semplici
- Molta memoria
- Non programmata ma addestrata mediante apprendimento automatico

**Definizione:**
La RN è un modello computazionale parallelo formato da numerose unità elaborative omogenee fortemente interconnesse mediante collegamenti di varia intensità. Ogni unità ha una attività semplice. I dati del problema da risolvere vengono forniti alle unità di Input, e la computazione si propaga in parallelo nella rete fino all'output (risultato). Una RN non viene programmata ma addestrata mediante una serie di esempi della realtà da modellare.
**Sistemi Hardware di Reti Neurali (6^ generazione) e Computers convenzionali (4-5^ gen.)**
Le RN possono essere simulate in software su computers tradizionali o implementate in hardware dando origine a una nuova generazione di strumenti di calcolo. Le RN si differenziano dai computers tradizionali per l'architettura, l'approccio a RN evita infatti il collo di bottiglia dell'accesso alla memoria ed è intrinsecamente parallelo (PC è sequenziale): integra memorizzazione (nelle connessioni) e computazione (in tante unità elaborative autonome).
**Reti Neurali e Intelligenza Artificiale**
Le RN rientrano a pieno titolo fra le tecniche di IA, costituendo il naturale complemento dei metodi simbolici della IA, con cui dovranno integrarsi nei futuri sistemi intelligenti
**Applicazioni delle RN:**

- Riconoscimento di segnali percettivi
- Diagnosi e gestione di apparati complessi
- Controllo del movimento di robot e veicoli autonomi
- Memoria associativa
- Ricostruzione di informazioni parziali o corrotte da rumore
- Soluzioni approssimate in tempo reale di problemi computazionalmente intrattabil

## Modello generale di Rete Neurale

Esistono vari modelli di RN chiaramente distinti come teoria sottostante, meccanismo di apprendimento e scopo. Si può però caratterizzare un MODELLO GENERALE che riassuma le caratteristiche comuni
Componenti di base di un modello generale:



Componenti:
- Un insieme di unità elaborative
- Uno stato di attivazione
- Una funzione di output $f_i$ per ogni unità
- Le connessioni fra le unità
- Una regola di propagazione per propagare i valori di output
- Una funzione di attivazione $F_i$ per combinare gli input
- Una regola di apprendimento per modificare le connessioni

Unità elaborative:
- Le unità elaborative sono semplici e uniformi
- L'attività di una unità consiste nel ricevere un input da un insieme di unità e calcolare un valore di output da inviare ad un altro insieme di unità
- Il sistema è inerentemente parallelo, le unità possono lavorare contemporaneamente
- Vi sono 3 tipi di unità:
    1. di Input
    2. di Output
    3. interne

Stato di attivazione della rete:
Ogni unità $u_i$ è caratterizzata da un valore di attivazione al tempo t : $a_i(t)$
Modelli diversi fanno assunzioni diverse circa i valori di attivazione che le unità possono assumere:
- valori discreti
- valori continui

Output delle unità elaborative:
Le unità interagiscono trasmettendo segnali ai loro vicini (unità connesse). La forza dei loro segnali e il grado di influenza sui vicini dipende dal loro grado di attivazione.
Connessioni:
Le unità sono connesse in una struttura a rete (digrafo). È definita una matrice di connessioni W in cui ogni connessione $w_{ji}$ è caratterizzata da:
- unità di partenza
- unità di arrivo
- forza della connessione

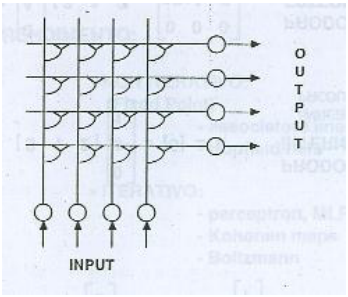
l'insieme delle connessioni costituisce la CONOSCENZA della RN e determina la risposta agli Input
- La MEMORIA risiede nelle CONNESSIONI
- Le CONNESSIONI non sono programmate ma APPRESE AUTOMATICAMENTE

Regola di propagazione:
E' la modalità con cui gli output delle unità o(t) sono propagati attraverso le connessioni della rete per produrre i nuovi input. Se vi sono più tipi di connessioni diverse la propagazione avviene indipendentemente per ogni tipo di connessione. Spesso questa regola prevede semplicemente la somma degli input all'unità pesati dall'intensità delle connessioni: $net_a = w_a\, o$
Funzione di attivazione:
E' una funzione F che prende il valore corrente di attivazione a(t) e il vettore $net_k$ di input all'unità e produce un nuovo valore di attivazione. F di solito è una funzione o di soglia, o quasi-lineare, o stocastica.
Apprendimento come modifica delle connessioni della rete neurale:
Le RN non sono programmate da un esperto umano ma si autodefiniscono mediante APPRENDIMENTO AUTOMATICO, che consiste nella modifica delle connessioni effettuata mediante una regola di apprendimento.
Ci possono essere 3 tipi di modifiche:
1. la creazione di nuove connessioni
2. la perdita di connessioni esistenti
3. la modifica della forza di connessioni esistenti

Regole di apprendimento:
Quasi tutte possono considerarsi varianti della regola di Hebb la cui idea di base è:
"se una unità $u_i$ riceve un input da una unità $u_j$ e se entrambe sono fortemente attive, il peso $w_{ji}$ ,da $u_j$ a $u_i$ deve essere rafforzato". La regola dice che il cambiamento della connessione da $u_j$ a $u_i$ è dato dal prodotto di una funzione g (attivazione $u_i$ e input di addestramento) e di una funzione h (output $u_j$ e forza connessione).
Nella versione più semplice (Hebb classica): $\Delta w_{ji} = \eta\, a_i\, o_j$ dove $\eta$ è la velocità di apprendimento
Una importante variante è la DELTA RULE, così detta perché l'apprendimento è proporzionale alla differenza (delta) fra il valore corrente di attivazione e quello desiderato, fornito dal maestro.

# Modelli di Reti Neurali

## Associatore Lineare (Linear Pattern Associator) (PA)




**Descrizione:**

- Ci sono due insiemi di unità: di Input e di Output (valori binari $\epsilon$ {-1,+1})
- Gli input sono completamente connessi agli output mediante una matrice di pesi
- Apprende associazioni fra pattern di Input e di Output mediante la regola di Hebb
- Serve per CLASSIFICAZIONE o MEMORIA ASSOCIATIVA

**Funzionamento:**

Memorizzata una associazione fra un pattern di input e uno di output, il PA apprende i pesi di una associazione mediante il prodotto esterno dei vettori da associare. Il valore di attivazione delle unità di output è:

$$o_t = \varepsilon\, o_p\, (i_p^T\, i_t)$$

L'output del test è proporzionale al prodotto dell'output dell'associazione appresa per il prodotto interno dell'input dell'associazione appresa e dell'input del test.

Sapendo che il prodotto interno tra due vettori da una misura della somiglianza, e se gli elementi dei vettori sono $\epsilon$ {-1,+1} come nel PA, allora si ha un prodotto interno normalizzato che potrà valere:

- +1 : vettori identici
- -1 : vettori opposti
- 0 : vettori ortogonali

Si può scrivere: $o_t = n\varepsilon\, o_p\, (i_p^T \cdot i_t)$

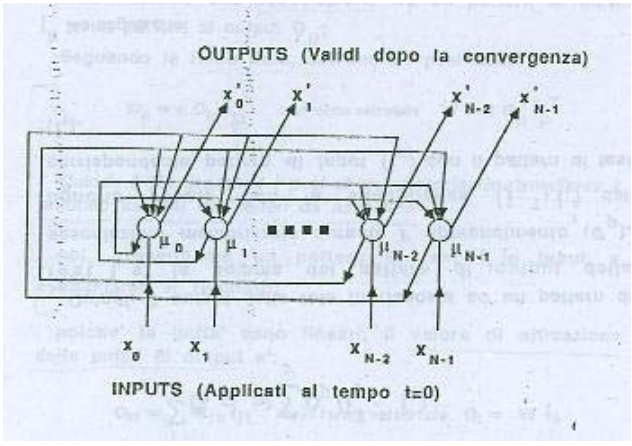
Quindi, l'output del test è uguale all'output dell'associazione appresa moltiplicato per uno scalare dipendente dal grado di somiglianza dell'input di test con l'input dell'associazione appresa.

Nel caso di apprendimento di più associazioni, l'output sarà una combinazione lineare degli output delle associazioni apprese, ognuno di essi pesato dal grado di somiglianza del rispettivo input con quello di test.

**Osservazioni:**

- Apprende col prodotto esterno NB ( prodotto esterno = $u\, v^T$ )
- Opera col prodotto interno NB ( prodotto interno = $v^T u$ )
- Riesce a memorizzare più associazioni in una stessa rete se i pattern di input sono ortogonali. E' un limite forte nei casi reali.

## Hopfield Neural Net

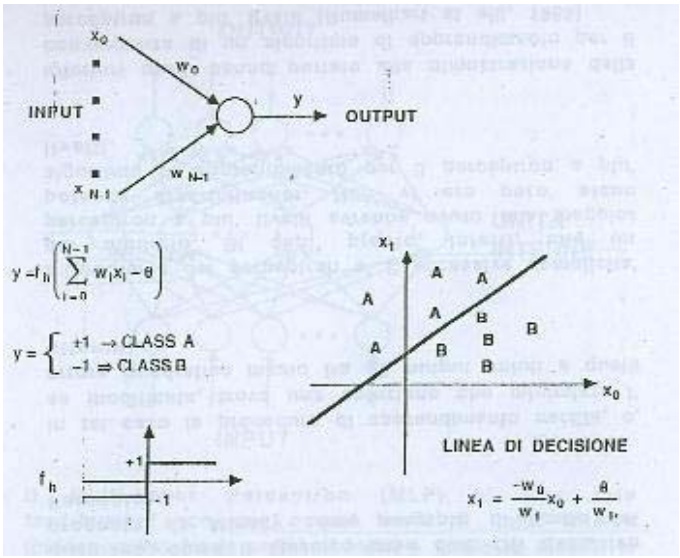


**Descrizione:**
- È usata come MEMORIA ASSOCIATIVA e CLASSIFICATORE
- Input e Output BINARI {+1,-1}
- I pesi sono fissati con un APPRENDIMENTO NON ITERATIVO
- A t=0 si da in Input un pattern sconosciuto, la rete cerca il pattern più simile ad esso

**Algoritmo:**
1. apprendimento (fixed point) : si assegnano i pesi delle connessioni
2. elaborazione di un pattern : scopo è riottenere il pattern più simile a quello in input, fra quelli memorizzati
3. si inizializzano le unità con il pattern di Input
4. si itera fino alla convergenza : il processo viene iterato finchè l'output dei nodi rimane immutato con ulteriori iterazioni

**Osservazioni:**
- apprende col prodotto esterno
- accede alla memoria iterando il prodotto interno
- più potente del PA, può memorizzare vettori non ortogonali
- LIMITI:
  - Il numero di vettori che può essere memorizzato e riottenuto è limitato dal vincolo dell'indipendenza lineare
  - Se si memorizzano vettori troppo simili, il pattern corrispondente diventa instabile

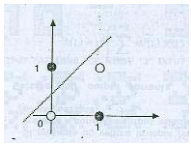
## Perceptron




**Descrizione:**
- Usata per CLASSIFICAZIONE
- Apprende a classificare semplici patterns
- Input BINARIO o CONTINUO; Output BINARIO
- Ha un SINGOLO NODO che calcola una somma pesata degli input, gli sottrae una soglia e passa il risultato ad una funzione scalino che restituisce +1 o -1 decidendo la classe del pattern (A o B)

**Algoritmo:**
1. inzializzazione di pesi e soglie : si assegnano piccoli valori casuali a $w_i(0)$ e $\theta$
2. presentazione di un nuovo input e dell'output desiderato
3. calcolo dell'output corrente
4. modifica dei pesi (delta rule)
5. iterazione ripartendo dal passo 2

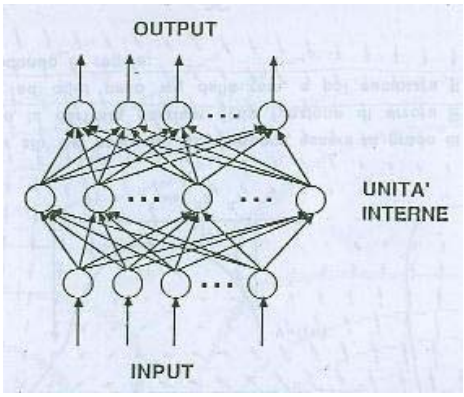
**Osservazioni:**
- se l'input proviene da due classi separabili, l'algoritmo di apprendimento converge e posiziona l'IPERPIANO DISCRIMINANTE fra le due classi
- spesso però due classi non sono separabili da un iperpiano, come nel caso dell'XOR



classe 0 = {(0,0) , (1,1)} classe 1 = {(1,0) , (0,1)}

- LIMITI:
  - Eccessiva semplicità del modello
  - Necessita di più livelli

## Multi-Layer Perceptron




**Descrizione:**
- È una rete feed-forward con uno o più livelli di unità interne fra le unità di input e quelle di output
- Input CONTINUO, output CONTINUO $\epsilon$ (0,1), funzione di attivazione di Sigmoide
- Supera le limitazioni del Perceptron, è promettente per varie applicazioni

**Back Error Propagation:**
Con la BP si modificano i pesi del MLP in modo da minimizzare la funzione di errore, un metodo classico per farlo è la discesa del gradiente. Per realizzarla nel MLP bisogna essere in grado di calcolare la derivata parziale della funzione di errore E rispetto ad ogni peso $w_{ji}$ della rete, e poi cambiare il peso secondo la regola (qui derivata):

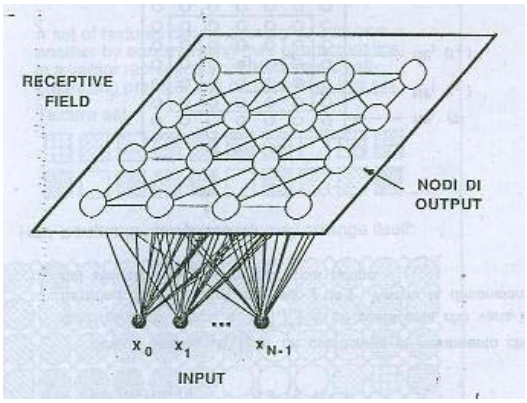
$$\Delta_p w_{ji} = \eta \, \delta_{pi} \, o_{pj}$$

Si dimostra che $\delta$ differisce nel caso di unità di output e di unità interne.

**Algoritmo:**
minimizza l'errore quadratico medio tra l'output corrente e quello desiderato di un MLP
1. inizializzazione di pesi e soglie : a piccoli valori casuali
2. presentazione di un esempio (vettore di input e corrispondente vettore di output)
3. calcolo degli output correnti
4. modifica dei pesi
5. iterazione a partire dal passo 2

**Osservazioni:**
- si possono ottenere, in base al n° di livelli, le seguenti superfici di discriminazione nello spazio degli input:
  - 1 livello : IPERPIANO
  - 2 livelli : REGIONE CONVESSA
  - 3 livelli : REGIONI COMUNQUE COMPLESSE, anche concave e disgiunte
- PROBLEMI:
  - la convergenza è garantita solo a minimi locali
  - può essere molto lenta

## Kohonen's Self Organizing Feature Maps

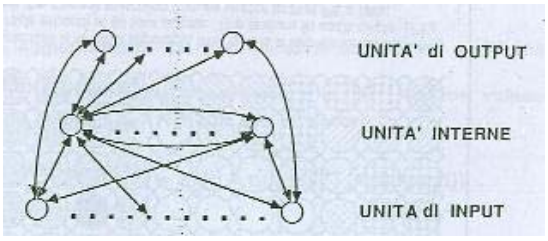



**Descrizione:**
- si AUTO ORGANIZZA per creare dei raggruppamenti in classi (CLUSTERS) di esempi non classificati. È usata per costruire QUANTIZZATORI VETTORIALI e per la COMPRESSIONE DEI DATI
- la rete è una GRIGLIA BIDIMENSIONALE di nodi che rappresentano i centri delle classi
- Input è un VETTORE DI REALI, i nodi in input sono completamente connessi al campo recettivo
- L'apprendimento è NON-SUPERVISONATO

**Algoritmo:**
1. inizializzazione dei pesi e del raggio di "vicinato" : i pesi sono inizializzati con piccoli valori casuali. Si inizializza il raggio di "vicinato" (distanza entro cui la modifica dei pesi di un nodo si ripercuote sui vicini)
2. presentazione di un input
3. calcolo della distanza da tutti i nodi : si calcola la distanza $d_i$ fra il vettore in input e ogni nodo di output j
4. selezione del nodo di output con la minima distanza
5. aggiornamento dei pesi verso il nodo $j^*$ e i suoi vicini
6. iterazione dal passo 2

**Osservazioni:**
- dimostrata la convergenza dell'algoritmo di apprendimento
- usato in: riconoscimento del parlato (quantizzatore vettoriale),casi di input rumorosi, controllo, classificazione
- dopo l'addestramento la rete è in grado di classificare nuovi input sconosciuti in una (o più) delle classi che si è autodefinite

## Boltzmann Machine (Modelli Termodinamici)




**Descrizione:**
- è una RN stocastica, completamente connessa, con due tipi di unità : VISIBILI (Input,Output) e INVISIBILI (o Interne)
- Input e Output BINARIO {0, 1}, PESI SIMMETRICI, funzione di attivazione STOCASTICA, dipendente dalla TEMPERATURA della rete. Usa il metodo del SIMULATED ANNEALING
- È in grado di COMPLETARE dei pattern parziali dati sulle unità visibili. Può effettuare una CLASSIFICAZIONE oppure fare da MEMORIA ASSOCIATIVA

**Funzionamento:**

Simulated Annealing:

tecnica di ottimizzazione per trovare il Minimo Assoluto di una funzione, che simula una agitazione termica per sfuggire ai minimi locali.

Simulated Annealing e BM:

La BM serve ad apprendere a SIMULARE un ambiente: dato un input la BM converge verso uno stato stabile (si rilassa) e ricostruisce l'output. Con il SA la rete converge verso lo stato a minima energia. La funzione da minimizzare è l'ENERGIA TOTALE della BM. All'equilibrio termico la probabilità relativa di due stati è determinata solamente dalla differenza delle loro energie: vi è una forte preferenza per gli stati a bassa energia, più marcata alle basse temperature.

**Procedimento:**

per ottenere l'Output di una BM dati gli Input (rilassamento mediante SA)

1. gli input sono imposti alla rete
2. si itera, cominciando con una temperatura elevata
3. le unità aggiornano gli output per un certo intervallo di tempo
4. la temperatura viene diminuita secondo la politica di raffreddamento
5. quando la rete si stabilizza, si ferma il raffreddamento (equilibrio termico) e si leggono gli output

Dato che la BM è una RN STOCASTICA, gli output continuano ad evolvere anche all'equilibrio, senza cambiare gli input, occorre perciò collezionarli per un certo tempo per avere una STIMA DELLA LORO FREQUENZA. L'output reale è quindi una DISTRIBUZIONE DI PROBABILITA'.

**Algoritmo di Apprendimento:**

una BM può essere usata per modellare una realtà fisica. L'apprendimento avviene tramite la modifica dei pesi della rete, minimizzando una misura di distanza fra le prestazioni della rete e quelle dell'ambiente.. All'equilibrio termico, tutta l'informazione richiesta per cambiare il peso $w_{ij}$ sta nel comportamento delle unità che esso connette $u_i$ e $u_j$. La regola di modifica dei pesi è:

$$\Delta w_{ij} = \eta \cdot \text{SIGN}\,(p_{ij}^{+} - p_{ij}^{-})$$

- $p^{+}_{ij}$ : probabilità (all'equilibrio) che le unità $u_i$ e $u_j$ siano entrambe attive (=1) quando la rete opera con l'input imposto dall'ambiente (fase$^{+}$)
- $p^{-}_{ij}$ : stessa probabilità quando la rete opera liberamente senza input imposto (fase$^{-}$)

Per ottenere una prestazione dipendente solo dall'input "sottraiamo" la prestazione dovuta alla struttura interna

**Osservazioni:**

- è un modello PIU' COMPLESSO DI ALTRI sia come struttura (stocastica) che come modalità nel generare l'output (SA) e l'algoritmo di apprendimento
- Applicazioni nel riconoscimento di classi vocaliche e classificazione dello shift del vettore di input
- PROBLEMI:
    - Determinazione dei parametri dell'algoritmo di apprendimento
    - Lentezza dell'apprendimento
    - Input binari