ANNO ACCADEMICO 2024/2025

Sistemi Intelligenti

Teoria

Altair's Notes

DIPARTIMENTO DI INFORMATICA

Capitolo 1 Introduzione Pagina 5

1.1 Che Cos'è l'Intelligenza Artificiale? 5
Un Inizio — 6 • Test di Turing — 6 • Strong e Weak AI — 8 • Esempi — 9 • Definire AI — 10

1.2 Risoluzione Automatica di Problemi 12
I Problemi — 12 • Metodi di Ricerca non Informati (Blind Search) — 13 • Lista di Strategie — 15

1.3 Ricerca Informata 18
A* — 18 • Approfondimento su Euristiche — 21

1.4 Strategie di Ricerca con Avversario 22
Teoria delle Decisioni — 22 • Programmi che Giocano — 25

Capitolo 2 CSP e Rappresentazione della conoscenza Pagina 27

2.1 Constraint Satisfaction Problem 27
Introduzione — 27 • Domini e Vincoli — 28 • AC-3 — 30 • K-consistency — 31 • Backjumping — 32 • Applicazioni di CSP — 33

2.2 Rappresentazione della Conoscenza 33

Premessa

Licenza

Formato utilizzato

Box di "Concetto sbagliato":

Concetto sbagliato 0.1: Testo del concetto sbagliato

Testo contente il concetto giusto.

Box di "Corollario":

Corollario 0.0.1 Nome del corollario

Testo del corollario. Per corollario si intende una definizione minore, legata a un'altra definizione.

Box di "Definizione":

Definizione 0.0.1: Nome delle definizione

Testo della definizione.

Box di "Domanda":

Domanda 0.1

Testo della domanda. Le domande sono spesso utilizzate per far riflettere sulle definizioni o sui concetti.

Box di "Esempio":

Esempio 0.0.1 (Nome dell'esempio)

Testo dell'esempio. Gli esempi sono tratti dalle slides del corso.

Box di "Note":

Note:-

Testo della nota. Le note sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive.

Box di "Osservazioni":

Osservazioni 0.0.1

Testo delle osservazioni. Le osservazioni sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive. A differenza delle note le osservazioni sono più specifiche.

1 Introduzione

Note:-

DISCLAIMER: questi appunti sono stati scritti da una persona che ha dovuto dare questo esame in magistrale (yeah, l'ho skippato in triennale e ora mi tocca darlo): essendo che ho dato molti esami che dipendono da questo insegnamento e possibile che in alcune sezioni dia per scontato delle cose.

1.1 Che Cos'è l'Intelligenza Artificiale?

Nell'immaginario l'intelligenza artificiale viene solitamente assimilata a quella di un robot antropomorfo che risolve problemi complessi e impara da essi.

Però esistono altri tipi di IA:

- Servizi di streaming: portali per l'accesso a molti files. Utilizzano meccanismi di personalizzazione.
- Social network.
- Assistenti virtuali.
- Macchine fotografihche/Smartphone.

1.1.1 Un Inizio

Definizione 1.1.1: Intelligenza

Complesso di facoltà psichiche e mentali che consentono all'uomo di pensare, comprendere o spiegare i fatti o le azioni, elaborare modelli astratti della realtà, intendere e farsi intendere dagli altri, giudicare, e lo rendono insieme capace di adattarsi a situazioni nuove e di modificare la situazione stessa quando questa presenta ostacoli all'adattamento; propria dell'uomo, in cui si sviluppa gradualmente a partire dall'infanzia e in cui è accompagnata dalla consapevolezza e dall'autoconsapevolezza, è riconosciuta anche, entro certi limiti (memoria associativa, capacità di reagire a stimoli interni ed esterni, di comunicare in modo anche complesso, ecc.), agli animali.

Note:-

Artificiale indica che non è naturale.

Prospettiva storica:

- 1936: Alan Turing formalizza la Turing Machine.
- 1940: ENIAC: primo computer "moderno".
- 1950: Test di Turing, quando un computer può dirsi intelligente?
- Il dubbio nasce dal contesto bellico in cui vennerò sviluppati i primi computer: all'epoca solo poche persone istruite riuscivano a fare i calcoli necessari.
- 1956: Nasce l'intelligenza artificiale.

Breve storia dell'automazione:

- *Automazione del calcolo:* metà anni '50, pochi dati, molti calcoli.
- *Automazione di procedure amministrative e contabili:* metà anni '60, pochi calcoli, grandi molti di dati alfanumerici.
- *Automazione di fabbrica:* metà anni '70, primi robot industriali, ambiente predeterminato.
- *Automazione di ufficio:* metà anni '80, primi PC, primi strumenti per utenti non esperti.
- *Automazione della ricerca delle informazioni:* fine anni '90, internet, WEB, motori di ricerca.

Domanda 1.1

L'automazione è intelligenza?

Ragionando: la calcolatrice è automatica, ma non si può dire intellingente. Una lavatrice è automatica, ha diversi programmi e si adatta. Un rover che gira su Marte effettua esperimenti e si adatta, ha una certa autonomia. Infine, gli LLM eseguono un programma e hanno la capacità di comunicare mediante linguaggio naturale.

1.1.2 Test di Turing

Domanda 1.2

Quando un programma può dirsi intelligente?

Definizione 1.1.2: Turing Test (The Imitation Game)

Un'intervistatore deve capire se un'entità misteriosa è umana o è una macchina. Può fare tutte le domande che vuole su qualsiasi argomento e l'entità deve rispondere (il tutto per scritto). Al termine l'intervistatore enuncia il suo verdetto. Se dice uomo ed era macchina, la macchina ha superato il test.

AI:

- Data e luogo di nascita:
  - Darthmouth Conference (USA), 1956.
  - Nome scelto da John McCarthy.
- In precedenza:
  - Una macchina può pensare ed essere considerata intelligente?
  - Vari approcci: cybernetica, teoria degli automi, etc.
  - Turing test.
- Successivamente:
  - Scacchi.
  - Giochi.
  - Dimostrazioni automatiche.

Domanda 1.3

Basta produrre gli output attesi per dire che vi è comprensione?

- Si può dare una risposta "giusta" avendo certe conoscenze e ragionamento.
- Ma si può dare una risposta "giusta" anche tirando a caso.

Definizione 1.1.3: Esperimento della Stanza Cinese

Una persona interagisce con un computer, programmato per rispondere con certi ideogrammi cinesi ad altri ideogrammi cinesi ricevuti in input. Il computer parla cinese? Lo capisce?

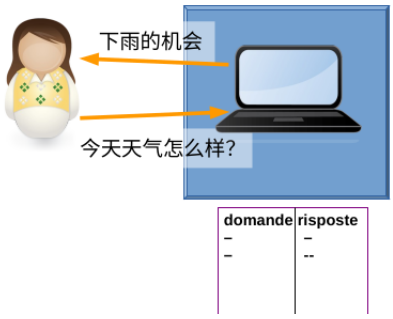


Figure 1.1: Esperimento di Searle.

Note:-

Ma supponiamo che una persona chiusa in una stanza ha istruzioni per rispondere con certi ideogrammi cinesi in risposta ad altri ideogrammi cinesi. Parla cinese? Lo capisce?

Definizione 1.1.4: Test di Turing Inverso

Usati per intercettare bot che cercano di accedere a form o a dati (C.A.P.T.C.H.A.).

Note:-

Una variante di questo test è usata in "Ma gli androidi sognano pecore elettriche?" (Blade Runner).

Figure 1.2: Esperimento di Searle.

1.1.3 Strong e Weak AI

Due tipi di intelligenza:

- *Strong AI:* è possibile riprodurre l'intelligenza umana?
- *Weak AI:* è possibile trovare dei modi per risolvere problemi che, se risolti dagli esseri umani richiederebbero intelligenza?

Obiettivo della weak AI:

- Programmare un agente artificiale in grado di:
  - Rilevare ostacoli.
  - Rilevare oggetti in movimento.
  - Costruire un piano d'azione.
  - Rilevare segnali significativi.
- In un ambiente che è:
  - Complesso.
  - Parzialmente prevedibile.
  - Parzialmente collaborativo.

Note:-

Nasce il binomio Agente-Ambiente.

Definizione 1.1.5: Agente

Un agente è un'astrazione che rappresenta un qualsiasi sistema che percepisce il proprio ambiente tramite i sensori e agisce su di esso tramite degli attuatori.

Osservazioni 1.1.1 Caratteristiche dell'ambiente

- Completamente osservabile: in ogni istante i sensori danno accesso a tutti gli aspetti dell'ambiente rilevanti per la scelta dell'azione.
- Parzialmente osservabile: i sensori danno accesso solo a parte dell'informazione rilevante (cause: sensori imprecisi oppure non in grado di rilevare alcuni dati).
- Deterministico: lo stato successivo è determinato dallo stato corrente e dall'azione applicata.
- Stocastico: applicando più volte una stessa azione in uno stesso stato si possono raggiugnere stati diversi. Si dice strategico quando è stocastico solo per quanto riguarda le azioni degli altri agenti.
- Epistodico: l'esperienza degli agenti è divisa in episodi atomici: un episodio è dato da una percezione seguita da una singola azione (esempio: classificazione).

- Sequenziale: attività composta da più passi ognuno dei quali in generale influenzerà i successivi.
- Statico: l'ambiente non cambia mentre l'agente "pensa".
- Dinamico: l'ambiente cambia mentre l'agente "pensa".
- Discreto: possono essere discreti stato, tempo, percezioni, azioni (esempio: gli scacchi hanno stati, percezioni, azioni discreti).
- Continuo: possono essere continui stato, tempo, percezioni, azioni (esempio: gli scacchi hanno tempo continuo).
- Singolo agente: viene modellata come agente una sola entità.
- Multi agente: vengono modellate come agenti più entità

Paradigmi di programmazione:

- Approccio tradizionale:
  - Imperativo.
  - A oggetti.
  - *Non è AI:* risolve un singolo compito ed è strutturato come una sequenza di passi.
  - Descrivono il COME.
- Approccio dichiarativo:
  - Separa una descrizione del COSA da un programma generale.
  - Lo stesso programma è applicato a diverse descrizioni per risolvere problemi diversi.

1.1.4 Esempi

Definizione 1.1.6: Mondo dei Blocchi

Tipico Toy Problem in ambito AI. Si hanno un tavolo con n posizioni e m blocchi. L'obiettivo è passare da uno stato iniziale a uno stato finale. Un agente può spostare un blocco per volta seguendo determinate regole.

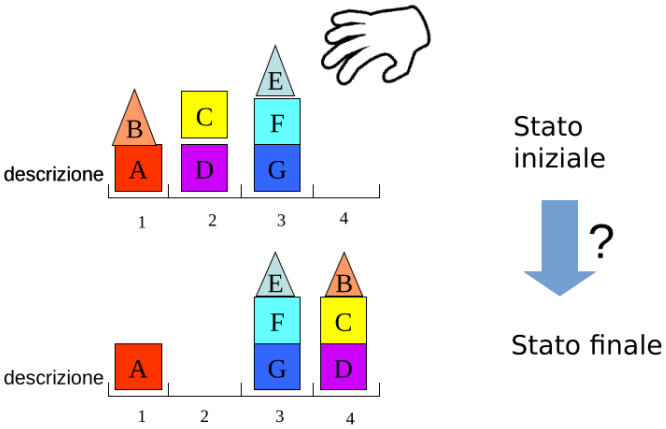


Figure 1.3: Mondo dei blocchi.

L'agente:

- Percepisce la situazione iniziale.
- Costruisce i passi per andare dalla situazione iniziale alla situazione finale.
- Deve determinare quali azioni lo avvicinano al *goal*.

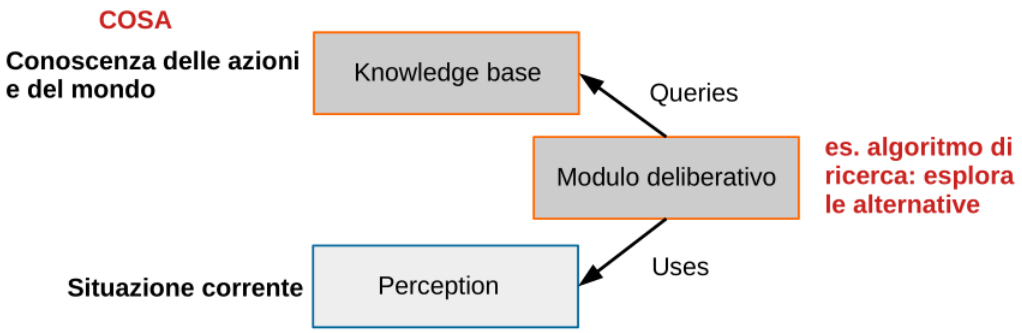


Figure 1.4: Meccanismo di deliberazione.

Domanda 1.4

Come scegliere le mosse?

- Dipende dal tipo di problema.
- A volte basta la prima mossa, a volte si vuole trovare una soluzione ottima.

1.1.5 Definire AI

Definizione 1.1.7: Automazione

Si deve programmare la macchina per fare ogni passo: è applicabile in domini fortemente ripetitivi.

Definizione 1.1.8: Autonomia

Un agente artificiale riceve compiti ad alto livello, l'utente demanda all'agente la risoluzione.

Un agente autonomo:

- Riceve solo compiti ad alto livello.
- Ragiona ed esplora alternative (molte mosse possibili a ogni istante).
- Riconosce quando non si può andare avanti su una strada [1].
- Riconosce che si è già stati in quella situazione.
- Prima si ragiona e poi si agisce.

Note:-

In AI gli agenti autonomi sono un modo di concepire i programmi, in cui controllo e logica (o modello) sono chiaramente separati.
Un agente fa sempre ciò che è programmato a fare[a].

[1] nota aggiuntiva: non proprio, dipende dal tipo di agente e dal suo control loop.

[a]No Terminator, sorry :'(

Domanda 1.5

Cosa vuol dire fare la cosa giusta?

Definizione 1.1.9: Funzione Deliberativa

La funzione deliberativa di un agente determina le azioni che saranno eseguite. In termini informali un agente è razionale quando "fa la cosa giusta", cioè opera per conseguire il "successo".

Note:-

Occorre quindi definire una *misura di prestazione*.

Il comportamento razionale di un agente dipende da 4 fattori:

1. Azioni nelle facoltà dell'agente.
2. Misura di prestazione.
3. Conoscenza dell'ambiente.
4. Percezione.

Corollario 1.1.1 Agente Razionale

Un agente razionale dovrebbe scegliere sempre un'azione che massimizza la misura di prestazione attesa, data la particolare sequenza percettiva in oggetto e le informazioni derivabili dalla conoscenza dell'ambiente.

Definizioni di AI:

- Sistemi che pensano come esseri umani:
  - Haugeland, 1985.
  - Bellman, 1978.
- Sistemi che agiscono come esseri umani:
  - Kuzweil, 1990.
  - Rich e Knight, 1991.
- Sistemi che pensano razionalmente:
  - Charniak e McDermott, 1985.
  - Winston, 1992.
- Sistemi che agiscono razionalmente:
  - Poole et al., 1998.
  - Nilsson, 1998.

Domanda 1.6

Quali problemi per l'AI:

- Non è adatta per:
  - Modelli matematici precisi.

  - Metodi algoritmici specifici.
- È utile/necessaria per:
  - Problemi non deterministici.
  - Più soluzioni.
  - Dati non numerici.
  - Grandi Knowledge Base (KB).
  - Interazione con ambiente ed esseri umani.

1.2 Risoluzione Automatica di Problemi

In questa parte si affronta la problematica di come definire il concetto di problema e di soluzione, di distinguere tra soluzione e soluzione ottima. Sono studiati tre approcci alla risoluzione di problemi: ricerca nello spazio degli stati, ricerca in spazi con avversario (giochi ad informazione completa), risoluzione di problemi mediante soddisfacimento di vincoli.

1.2.1 I Problemi

- La realtà che definisce un problema può essere astratta in un insieme di stati.
- La realtà transisce da uno stato ad un altro tramite l'esecuzione di azioni (o operazioni).

Caratteristiche:

- *Stati discreti* (o dentro o fuori, non ci sono stati graduali).
- Effetto *deterministico* delle azioni.
- *Dominio statico* (non cambia durante l'esecuzione delle azioni).

Esempio non deterministico:

- Eseguendo più volte la stessa azione si possono avere conseguenze diverse.
- Si hanno *stati continui*.

Definizione 1.2.1: Obiettivo

Un obiettivo (goal) è un risultato verso il quale gli sforzi sono diretti. È una condizione data in termini di:

- Situazione.
- Prestazione.

Note:-

L'insieme degli stati obiettivo sono tutti gli stati in cui vale la condizione che li definisce.

Definizione 1.2.2: Algoritmo di Ricerca

L'algoritmo di ricerca determina una soluzione che, a partire da uno stato iniziale, permette di raggiungere un dato stato obiettivo. Usa:

- Una descrizione del problema.
- Un metodo di ricerca attraverso lo spazio degli stati.

Corollario 1.2.1 Soluzione

Una soluzione è un percorso nello spazio degli stati.

Un problema di ricerca può essere definito come una tupla di 4 elementi:

1. Stato iniziale: cattura la situazione a partire dalla quale viene computata la soluzione.
2. Funzione successore: dato uno stato e un'azione legale in esso calcola lo stato a cui si transisce eseguendo quell'azione in quello stato.
3. Test obiettivo: determina se lo stato a cui è applicato è lo stato goal: può verificare una proprietà o verificare l'appartenenza dello stato all'insieme degli stati target.
4. Funzione di costo del cammino: dato un percorso possibile gli assegna un costo numerico.

Alcune astrazioni:

- *Stati:* occorre rappresentare solo l'informazione rilevante alla soluzione del problema.
- *Azioni:* occorre rappresentare solo gli aspetti funzionali alla soluzione del problema.
- *Toy problem:* un problema artificiale avente lo scopo di illustrare o mettere alla prova dei metodi di risoluzione. Ha una formulazione precisa e univoca. Utile per confrontare metodi diversi
- *Real-world problem:* problemi concreti, effettivi. Spesso non hanno una formulazione unica.

Note:-

E.g. di toy problems: problema dell'aspirapolvere, gioco dell'8, problema delle 8 regine.

Possibili approcci:

- *Blind:* usano esclusivamente la struttura del problema per cercare una soluzione.
- *Informati:* usano la struttura del problema e ulteriore conoscenza per guidare la ricerca.

1.2.2 Metodi di Ricerca non Informati (Blind Search)

Definizione 1.2.3: Albero di Ricerca

Un albero di ricerca è una struttura dati usata per trovare una soluzione a un problema di ricerca:

- Ogni nodo corrisponde a uno stato.
- I nodi figli sono costruiti tramite la funzione successore.
- Ogni nodo ha un riferimento al nodo padre (per ricostruire le soluzioni).
- L'albero è costruito a partire dal nodo corrispondente allo stato iniziale.
- L'albero diventa un grafo quando lo stesso nodo (NB: non lo stesso stato) può essere raggiunto tramite più percorsi.
- Un percorso che porta dal nodo iniziale a un nodo obiettivo è una soluzione.

Note:-

Gli alberi sono un caso specifico dei grafi.

Formalizzando:

- Un *grafo di ricerca* $G = (\{n_i\}, \{e_{ij}\})$ è costituito da un insieme di nodi n_i e di archi e_{ij}.
- $e_{pq} \in \{e_{ij}\}$ rappresenta l'esistenza di un arco dal nodo n_p al nodo n_q, quindi n_q è successore di n_p.
- Ciascun arco e_{ij} ha associato un costo c_{ij}.
- L'esistenza di e_{ij} non implica l'esistenza di e_{ji}.

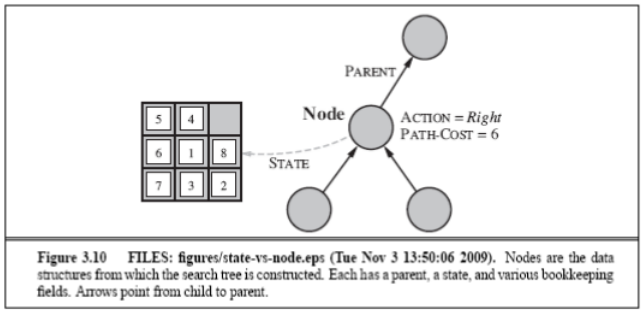


Figure 3.10 FILES: figures/state-vs-node.eps (Tue Nov 3 13:50:06 2009). Nodes are the data structures from which the search tree is constructed. Each has a parent, a state, and various bookkeeping fields. Arrows point from child to parent.

Figure 1.5: Esempio con il gioco dell'8.

Domanda 1.7

Se le strategie sono tante ve ne è una migliore? Come le confronto?

Criteri di valutazione:

- *Completezza:* garanzia di trovare una soluzione, se esiste.
- *Ottimalità:* garanzia di trovare una soluzione ottima (a costo minimo)[2].
- *Complessità temporale:* quanto tempo occorre per trovare una soluzione.
- *Complessità spaziale:* quanta memoria occorre per effettuare la ricerca.

Domanda 1.8

Come si valuta la complessità?

- *Complessità computazionale:* dato un problema esistono infiniti algoritmi che lo risolvono.
- *Termine di paragone:*
  - Tempo.
  - Spazio.
- *Criterio di preferenza:* economicità.

Note:-

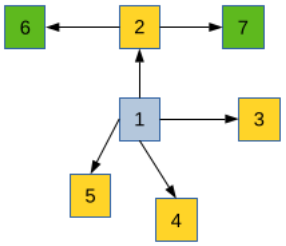
Per astrarre dal calcolatore utilizzato lo spazio e il tempo non sono metrici, ma parametrici. E.g. Numero di nodi creati o visitati.
È interessante vedere l'andamento del costo al variare della dimensione del problema.

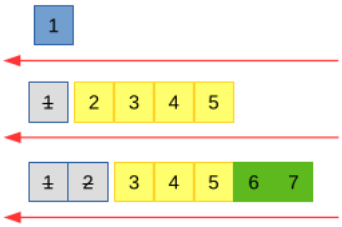
[2] Corso di "Algoritmi e Complessità

1.2.3 Lista di Strategie

Definizione 1.2.4: Ricerca in Ampiezza

La ricerca espande il nodo radice, poi tutti i suoi successori, poi tutti i discendenti di secondo livello, ecc. Si realizza gestendo la frontiera come una coda FIFO.





Valutazione:

- *Completezza:* se esiste un nodo obiettivo a una profondità finita d, la ricerca in ampiezza lo troverà a patto che il fattore di ramificazione b (cioè il numero di figli che un nodo può avere) sia finito.
- *Ottimalità:* la soluzione trovata è ottima solo se il costo del cammino è una funzione monotona crescente della profondità (es. tutte le azioni hanno lo stesso costo).
- *Complessità temporale:* $O(b^{d+1})$.
- *Complessità spaziale:* $O(b^{d+1})$, perché bisogna tenere in memoria sia la frontiera che gli antenati.

Definizione 1.2.5: Ricerca a Costo Uniforme

Nella ricerca a costo uniforme:

- Ogni nodo ha associato il costo del cammino con cui è stato raggiunto.
- La frontiera è mantenuta ottimale.
- A ogni iterazione espande il nodo appartenente a un cammino di costo minimo.

Note:-

Quando i costi sono tutti uguali diventa una ricerca in ampiezza.

Osservazioni 1.2.1

- Costo $\neq$ Numero dei passi: il numero dei passi effettuati non conta, conta solo il costo dei cammini.
- Quando trova il nodo obiettivo non si ferma subito, prima controlla se vi sono cammini aperti di costo inferiore e nel caso prova a espanderli.

Valutazione:

- *Completezza:* garantibile solo se tutti i passi hanno costo $\geqslant \epsilon > 0$. È il costo minimo delle operazioni.
- *Ottimalità:* garantibile solo se tutti i passi hanno costo $\geqslant \epsilon > 0$. Non sa gestire la nozione di guadagno unita a quella di costo.
- *Complessità temporale e spaziale:* ordine del branching factor elevato al numero di passi del percorso ottimale se i costi dei passi fossero uniformi ($O(b^{1+LC^*/\epsilon})$).

Definizione 1.2.6: Ricerca in Profondità

La ricerca in profondità espande sempre uno dei nodi più profondi della frontiera, cioè uno dei più lontani dalla radice. L'espansione produce tutti i successori di un nodo. Quando la ricerca tenta di espandere un nodo che non ha successori, l'effetto è che il nodo viene rimosso e si "torna indietro" nell'albero per esplorare eventuali alternative.

Note:-

Può essere con Backtracking o senza Backtracking[a].

[a]Meglio visto in "Intelligenza Artificiale e Laboratorio"

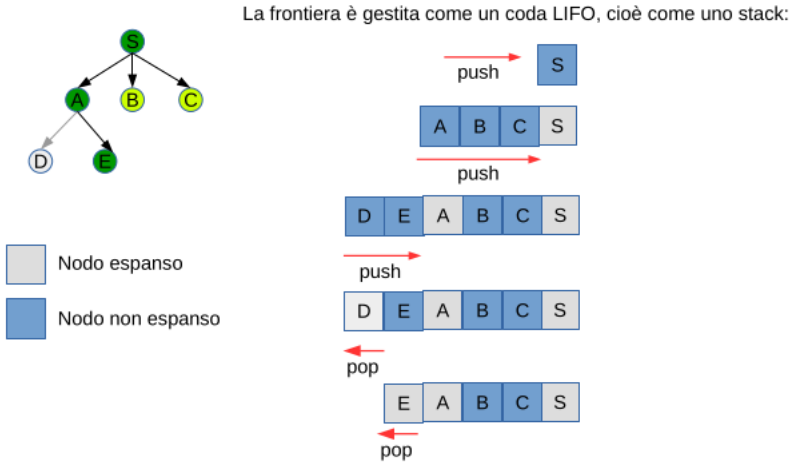


Figure 1.6: Ricerca in profondità.

Valutazione:

- *Completezza:* garantibile solo se tutti i cammini sono finiti.
- *Ottimalità:* in generale non è garantita.
- *Complessità temporale:* nel caso peggiore vengono percorsi tutti i nodi dell'albero ($O(b^m)$), dove b è il branching factor e m è la profondità massima.
- *Complessità spaziale:*
  - Senza Backtracking: $O(b * m)$, perché occorre mantenere tutti i nodi del cammino esplorato più tutti i loro fratelli.
  - Con Backtracking: $O(m)$

Corollario 1.2.2 Ricerca in Profondità Limitata

Per evitare di entrare in branch infiniti viene introdotto un limite artificiale l (punto di taglio):

- Un nodo viene espanso solo se la sua profondità $p \leqslant l$.
- Altrimenti viene trattato come un nodo privo di successori.

Note:-

Tutti i cammini saranno lunghi al più l.

Problemi:

- Si riduce la completezza, perché una soluzione potrebbe trovarsi a profondità maggiore di l.
- Se $l >> d$ si perde efficienza.

Definizione 1.2.7: Iterative Deepening

Esegue una ricerca in profondità limitata, con iterazioni successive in cui la profondità massima viene aumentata via via.

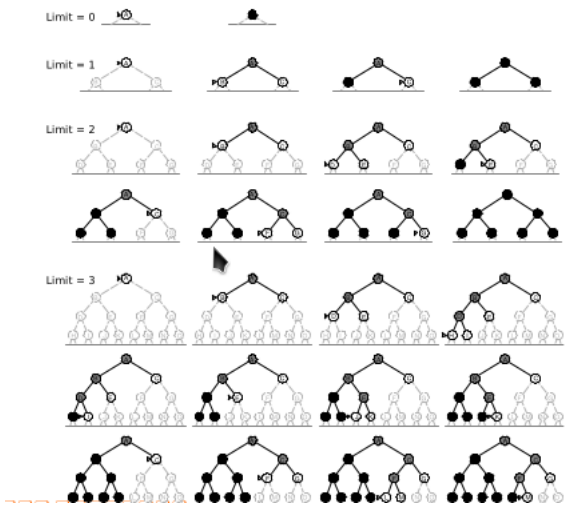


Figure 1.7: Iterative Deepening.

Valutazione:

- Combina ampiezza e profondità.
- È preferito quando lo spazio di ricerca è ampio e la profondità della soluzione non prevedibile.
- Ha una complessità spaziale modesta.
- La complessità temporale è alta.
- È completo se b è finito.
- È ottimo quando il costo è funzione non decrescente della profondità.

Definizione 1.2.8: Ricerca Bidirezionale

Composta da due ricerche:

- Forward dallo stato iniziale.
- Backward dallo stato obiettivo.

Termina quando le due ricerche si incontrano, quando le frontiere hanno intersezione non vuota.

Note:-

Se lo stato obiettivo non è unico si può introdurre uno stato fittizio raggiungibile da tutti gli stati obiettivo reali.

Sono possibili due andamenti:

- *A fronte d'onda:* quando non si ha informazione aggiuntiva si esplorano tutte le possibili operazioni.
- *A cono:* quando si ha informazione aggiuntiva si esplora parte delle operazioni.

1.3 Ricerca Informata

Domanda 1.9

Le strategie di ricerca blind non sono efficienti. E' possibile rendere la ricerca più efficiente utilizzando della conoscenza sul problema? La conoscenza permette di focalizzare la ricerca verso le direzioni più promettenti?

Definizione 1.3.1: Funzione di Valutazione

Una strategia di ricerca informata ordina la frontiera sulla base di una funzione di valutazione $f(n)$ applicata ai nodi:

- In base a $f(n)$ si ottengono strategie differenti.
- $f(n)$ comprende una componente $h(n)$ che restituisce una stima del minimo tra i costi che congiungono lo stato corrispondente al nodo n a uno stato goal.
- $h(n)$ è detta euristica.

Note:-

La strategià generale è detta *best-first search* e comprende greedy, A* e RBFS.

Definizione 1.3.2: Ricerca Greedy

Viene scelto il nodo stimato più vicino a quello obiettivo ($f(n) = h(n)$).

Problemi:

- Rischio di loop per vicoli ciechi.
- Stessi difetti della ricerca in profondità.

Intuizione:

- È possibile combinare la ricerca a costo uniforme con la ricerca greedy.
- *Costo uniforme:* espande per primi i nodi il cui raggiungimento dalla radice costa meno (guarda al passato).
- *Greedy:* espande per primi i nodi che promettono di raggiungere l'obiettivo spendendo meno (guarda al futuro).

1.3.1 A*

Definizione 1.3.3: A*

A* considera grafi, generati a partire da un singolo nodo iniziale, in cui un sottoinsieme T di nodi è costituito da nodi obiettivo (T sta per target). Dato un qualsiasi nodo n del grafo, un obiettivo t è detto preferito per n se e solo se il costo del cammino ottimo $h(n,t) \leqslant$ costo di qualsiasi altro cammino da n verso qualsiasi altro nodo di T.

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

- $g(n)$: costo minimo di tutti i percorsi, visti fino a ora, che consentono di raggiungere il nodo n a partire dallo stato iniziale s.
- $h(n)$: stima del costo minimo del proseguimento di percorso che consente di raggiungere un goal preferito di n.
- $f(n)$: stima del costo minimo per raggiungere un goal preferito di n partendo da s.

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

- $g^*(n)$:costo minimo per raggiungere il nodo n a partire dallo stato iniziale s (calcolato considerando tutti i cammini possibili).
- $h^*(n)$: costo minimo reale del proseguimento di percorso che consente di raggiungere un goal preferito a partire dal nodo n.
- $f^*(n)$: costo minimo per raggiungere un goal preferito di n da s.

Note:-

Se si sono esplorate tutte le alternative $f(x) = f*(n)$.

Algoritmo di A*:

1. Sia s il nodo iniziale.
2. Sia T l'insieme dei nodi obiettivo.
3. Segna s come aperto e calcola $f(s)$.
4. Seleziona il nodo aperto n avente valutazione minima.
5. Se n appartiene a T, marca n chiuso e termina.
6. Altrimenti marca n chiuso e applica l'operatore successore a n e:
   - Calcola il valore di f per tutti i successori n'.
   - Marca come aperti quei successori n' che non risultano già chiusi.
   - Rimarca come aperti quei successori n' che erano chiusi ma per cui è stato calcolato un valore f più basso di quello calcolato in precedenza (cioè sono stati raggiunti tramite un percorso migliore).

Definizione 1.3.4: Euristica Ammissibile

Un'euristica h è detta ammissibile quando

$$\forall n, h(n) \leqslant h^*(n)$$

dove $h^*(n)$ è il costo minimo reale per raggiungere il nodo goal a partire dal nodo n.

Note:-

Intuitivamente un'euristica è ammissibile quando non fa mai stime per eccesso.

Corollario 1.3.1 Ottimalità di A*

Se:

- Un'euristica h è ammissibile.
- Tutti i passi hanno costo maggiore di 0.

Allora:

- A^* termina e trova una soluzione ottima.
- In questo caso A^* è completa e ottimale.

Osservazioni 1.3.1

Perché manchi l'ottimalità deve accadere che durante la ricerca l'algoritmo:

- Scelga un nodo obbiettivo sub-ottimo.
- Al posto di un nodo.
- Che si trova su un cammino ottimo.

Note:-

Vedere dimostrazione su slides.

Corollario 1.3.2 Euristica Monotona

Un'euristica ammissibile è monotona quando

$$\forall n h(n) \leqslant c(n, a, n') + h(n')$$

dove c è la funziona costo per andare da n a n' mediante l'azione a.

Note:-

Tuttavia ammissibile non vuol sempre dire informativa: $h(n) = 0$ è ammissibile, ma non informativa.

Valutazione:

- A* è ottimameente efficiente per qualsiasi euristica: non esiste alcun altro algoritmo ottimo che garantisca di espandere meno nodi di quelli espansi da A*.
- Il numero di nodi espansi aumenta esponenzialmente con la profondità della soluzione ottima.
- A* mantiene in memoria tutti i nodi generati (è una ricerca in ampiezza).

Funzioni euristiche nel gioco dell'8:

- In media occorrono 22 mosse per arrivare alla soluzione.
- Il branching factor è pari a 3.
- Albero esaustivo di ricerca: contiene 3^22 nodi.
- Grafo esaustivo di ricerca: 180000 nodi.
- Ma se si passa al problema del 15 il grafo *esplode*: 10^23.

Figure 1.8: Il grafo be like.

1.3.2 Approfondimento su Euristiche

Possibili euristiche per A*:

- h1 (numero di tessere fuori posto): è ammissibile perché ogni tessera fuori posto deve essere spostata almeno una volta.
- h2 (*distanza di Manhattan*): è la somma della distanza di una tessera dalla sua posizione desiderata, contata in numero di tessere attraversate (originariamente di isolati attraversati) sulle ascisse più numero di tessere attraversate sulle ordinate. È ammissibile perché ogni mossa può spostare una tessera al più di una posizione più vicina al goal.

Qualità delle euristiche:

- La qualità di un'euristica può essere calcolata computando il branching factor effettivo b*.
- b* = branching factor di un albero uniforme di profondità d che contiene N+1 nodi.
- Le *euristiche migliori* hanno b* bassi, vicini a 1.

Definizione 1.3.5: Problemi Rilassati

Un problema ne rilassa un altro quando toglie qualche vincolo. Il grafo degli stati di un problema rilassato è un supergrafo di quello del problema originario perché include transizioni che i vincoli di quest'ultimo non consentono (meno vincoli, più transizioni possibili[a]).

[a] Magari fosse così ovunque.

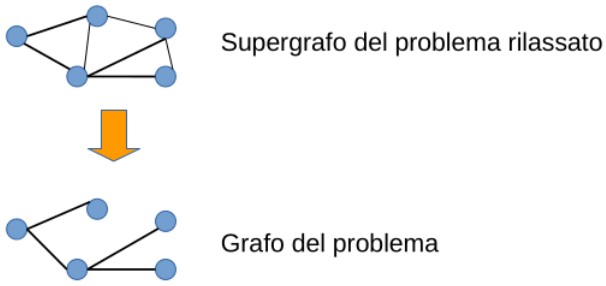


Figure 1.9: Rilassamento di un grafo.

Corollario 1.3.3 Absolver II

Absolver II è un esempio di programma che è in grado di generare automaticamente euristiche ammissibili per astrazione:

- Ha scoperto la prima euristica ammissibile per il cubo di Rubik.
- Ha scoperto un'euristica ammissibile per il problema dell'8 che è migliore di quelle precedentemente proposte

Note:-

Gli studi sulla generazione di euristiche continua oggi soprattutto nell'area di planning.

Definizione 1.3.6: Recursive Best First Search

RBFS trasforma la ricerca in ampiezza di A* in una ricerca in profondità.

1.4 Strategie di Ricerca con Avversario

Ambiente competitivo:

- Multi-agente.
- Ogni agente ha *obiettivi*.
- Gli obiettivi di agenti diversi sono conflittuali, cioè il conseguimento degli obiettivi di un agente impedisce il conseguimento degli obiettivi degli altri agenti.
- I problemi di ricerca con avversario sono anche detti *giochi*.

Tipologie di giochi:

- *Condizioni di scelta:*
  - Informazione perfetta: gli stati del gioco sono totalmente espliciti per tutti gli agenti.
  - Informazione imperfetta: gli stati del gioco sono solo parzialmente esplicitati.
- *Effetti della scelta:*
  - *Deterministici:* gli stati sono determinati unicamente dalle azioni degli agenti.
  - *Stocastici:* gli stati sono determinati anche da fattori esterni (es: dadi).

	Informazione Perfetta	Informazione Imperfetta
Giochi deterministici	Scacchi, Go, Dama, Otello, Forza4, tris	MasterMind
Giochi stocastici	Backgammon, Monopoli	Scarabeo, Bridge, Poker… (giochi di carte) Risiko

Figure 1.10: Tipi di giochi.

1.4.1 Teoria delle Decisioni

Elementi delle decisioni:

- *Andamenti:* non controllabili, dipendono da dinamiche esterne.
- *Scelte:* se ne può fare una sola.
- *Payoff:* ossono essere guadagno o perdite ma anche riferirsi ad altre misure (esempio: tempo, risorse). Sono specifici del problema.

Sono possibili tre approcci:

- Approccio maximax (ottimistico): Guarda i payoff più alti per ogni possibile scelta e fa la scelta che promette di più in assoluto: il massimo dei massimi. È ottimistica perché non ha garanzie che le dinamiche esterne faranno salire il fondo scelto.
- Approccio maximin (pessimistico): Questo approccio guarda le perdite maggiori legate a ciascuna scelta e poi esegue l'azione che minimizza le perdite (il massimo dei minimi).
- Approccio minimax (pentimento): best payoff - real payoff.

Guadagno e giochi con avversario:

- Non sappiamo come evolverà l'ambiente, non sappiamo quale scelta farà l'avversario.
- Dobbiamo far bastare la conoscenza dello stato corrente e delle mosse a disposizione.

Caratteristiche del gioco:

- Due giocatori.
- Ciascun giocatore non sa quali mosse farà l'altro ma le mosse possibili sono note e sono calcolabili i successori che produrranno una volte applicate a qualche stato.
- Osservabilità:
  - *Totale:* giochi con turno, i giocatori conoscono i risultati delle mosse precedenti.
  - *Parziale:* giochi ad azione simultanea: i giocatori non conoscono le mosse che i giocatori eseguono simultaneamente alla loro.
- Partendo da uno stato iniziale è possibile sviluppare un albero di possibili evoluzioni (*albero di gioco*), applicando le azioni eseguibili e calcolando così gli stati successori.
- Alcuni stati sono terminali, quando uno di essi è raggiunto la partita termina.
- I giocatori si avvalgono del calcolo dell'utilità degli stati.
- I giocatori devono tener conto dell'avversario quindi il calcolo dell'utilità comprende una valutazione del punto di vista dell'avversario.
- Giocatori pessimisti: suppongono che l'avversario faccia sempre la mossa che gli porta il guadagno maggiore.

Definizione 1.4.1: Strategia Ottima per un Agente

Sequenza di mosse che porta a uno stato terminale corrispondente alla vittoria dell'agente.

Note:-

L'agente non sa come muoverà l'altro, può solo "immedesimarsi".

Definizione 1.4.2: Minimax

Un algoritmo che rappresenta la strategia ottima per un agente è minimax in cui:

- Il max rappresenta l'agente che vuole massimizzare la propria utilità.
- Il min rappresenta l'avversario che vuole minimizzare l'utilità dell'agente.

Valutazione di minimax:

- Effettua una visita in profondità completa quindi la complessità temporale è esponenziale e quella spaziale è lineare.
- È completo in grafi finiti.
- È ottimale se MAX e MIN giocano in modo ottimale.

Corollario 1.4.1 Potatura Alfa-Beta

Per ridurre i tempi di ricerca si può fare pruning sui rami meno promettendi:

- α = massimo lower bound delle soluzioni possibili.
- β = minimo upper bound delle soluzioni possibili.

Ha senso esplorare un nodo se e solo se il suo valore stimato N è compreso tra i due estremi.

Minimax e Alpa-Beta Pruning sono equivalenti:

- Trovano la stessa mossa ottima.
- Attribuiscono alla radice la stessa valutazione.

Note:-

Alfa-Beta ha complessità temporale $O(b^{m/2})$ contro $O(b^m)$.

Osservazioni 1.4.1

- Alpha-beta pruning è più o meno efficace a seconda dell'ordine con cui i successori di ciascun nodo sono considerati.
- Se il successore più promettente è l'ultimo a essere considerato non è possibile evitare di esplorare i sottoalberi dei suoi fratelli.
- Quando l'ordinamento dei successori non è possibile, la complessità diventa $O(b^{3m/4})$.
- *Killer move:* espandere i figli più promettenti per primi.

Domanda 1.10

Come trovare le killer move?

- Tramite *apprendimento* per cui il sistema ricorda le esperienze passate e le usa per scegliere la mossa più promettente.
- Combinazione della potatura alfa-beta con iterative deepening.
- Uso di *tabelle di trasposizione* (talvolta in aggiunta ad alfa- beta + iterative deepening)

Definizione 1.4.3: Trasposizione

In alcuni problemi, eseguendo un certo insieme di mosse, è possibile ottenere sempre uno stesso risultato anche se le mosse sono ordinate differentemente. I diversi ordinamenti sono detti trasposizioni.

Note:-

Quando lo spazio degli stati è grande riconoscere le trasposizioni è importante per evitare di esplorare più volte gli stessi stati.

Corollario 1.4.2 Tabella di Trasposizione

Una hash table che contiene tutte le trasposizione: ogni volta che si genera un nuovo stato si controlla se corrisponde a uno stato già generato da una trasposizione. Se sì non viene esplorato.

Alfa-Beta in contesti real-time:

- Alfa-beta concentra la ricerca su una porzione limitata dello spazio degli stati ma deve comunque arrivare agli stati terminali.
- Può diventare troppo lento nel produrre la risposta quando il nodo terminale è situato a grande profondità (esempio: scacchi).
- In questo caso è necessario introdurre dei test di "*cutoff*" per produrre una decisione prima di raggiungere il nodo terminale.

Definizione 1.4.4: Funzione di Valutazione

Una funzione di valutazione fa una stima della bontà di uno stato intesa come percentuale di presenza degli stati che portano alla vittoria rispetto agli altri. Calcola la probabilità di essere in uno stato che porta a vittoria conoscendo solo la classe di appartenenza dello stato.

Note:-

Ma nel mondo reale non sempre è immediato capire come impostare una funzione di valutazione.

Definizione 1.4.5: Problema dell'Orizzonte

L'algoritmo non vede oltre il punto di taglio, ma in alcune fasi il gioco si può capovolgere il fretta o, in altri termini, la funzione di valutazione è instabile. Tagliare in questi punti è prematuro perché rischioso: l'avversario potrebbe successivamente forzare un forte cambiamento della valutazione.

Definizione 1.4.6: Quiescenza

La nozione di quiescenza concerne la permanenza della negatività (o positività) della valutazione. Si taglieranno nodi la cui valutazione è quiescente mentre quelli non quiescenti richiederanno un po' di esplorazione ulteriore dei sottoalberi che li vedono come radici.

1.4.2 Programmi che Giocano

Definizione 1.4.7: DeepBlue

Primo calcolatore a vincere una partita a scacchi contro un Campione del Mondo in carica, Garry Kasparov, con cadenza di tempo da torneo.

Grande potenza computazionale:

- Un computer a parallelismo massivo a 30 nodi basato su RS/6000, supportato da 480 processori specifici VLSI progettati per il gioco degli scacchi.
- Algoritmo in C.
- È capace di calcolare 200 milioni di posizioni al secondo.

Definizione 1.4.8: AlphaGo

Sviluppato da google, primo programma che ha battuto senza handicap a go un maestro umano, su un goban di dimensioni standard.

Nelle 500 partite disputate contro altri programmi ha vinto:

- Tutte le partite meno una quando eseguito su un solo computer.
- Tutte le partite quando eseguito su di un cluster che impiegava 1202 CPU e 176 GPU, circa 25 volte in più rispetto all'hardware del computer singolo.
- La versione cluster ha battuto la versione su singolo computer nel 77% delle partite.

Note:-

Utilizza deep learning neural networks e ricerca su alberi. Le reti neurali sono state addestrate su un dataset di 30.000.000 di mosse e poi raffinate giocando contro se stesse.

2

CSP e Rappresentazione della conoscenza

2.1 Constraint Satisfaction Problem

2.1.1 Introduzione

Definizione 2.1.1: Constraint Satisfaction Problem

Un constraint satisfaction problem (CSP) è definito da:

- Un insieme di variabili $X_1, \ldots, X_n$.
- Un insieme di vincoli $C_1, \ldots, C_m$.
- In alcuni casi è richiesta la massimizzazione di una funzione obiettivo.

Corollario 2.1.1 Stati

Gli stati di un CSP sono dati da tutti gli assegnamenti possibili per le variabili del CSP.
Un assegnamento $X_{i1} = v_{i1}, X_{i2} = v_{i2}$ è un'attribuzione di valori a un sottoinsieme delle variabili del CSP.

Un assegnamento è detto:

- *Completo:* se assegna valori a tutte le variabili del CSP.
- *Consistente:* se non viola alcun vincolo del CSP.
- *Soluzione:* se è completo e consistente.

Note:-

Quando esiste una soluzione per un vincolo si dice anche che esiste un mondo possibile che soddisfa il vincolo.
I vincoli binari possono essere rappresentati come archi di un grafo i cui nodi sono le variabili del CSP.

CSP come problemi di ricerca in uno spazio degli stati:

- *Stato iniziale* = {} assegnamento vuoto.
- *Successore* = assegnamento di un valore a una delle variabili che non ce l'hanno facendo attenzione che non sorgano conflitti.
- *Test obiettivo* = assegnamento completo.
- *Costo* = ogni passo ha costo costante.

2.1.2 Domini e Vincoli

- *Domini finiti:* è possibile enumerare i vincoli mettendo in relazione i diversi valori.
- *Domini infiniti:* non è possibile enumerare i vincoli, si usano linguaggi di specifica.
- *Domini continui:* la programmazione lineare permette di risolvere CSP in cui i vincoli sono disuguaglianze lineari che specificano una regione convessa.

Arità dei vincoli:

- *Unari:* coinvolgono una variabile e un valore.
- *Binari:* coinvolgono due variabili e possono essere rappresentati come archi di un grafo.
- *A tre o più variabili:* Vincoli a tre o più variabili coinvolgono un numero qualsiasi di variabili, possono essere rappresentati da ipergrafi[1].

Esempio 2.1.1 (Criptoaritmetica)
La criptoaritmetica è un gioco in cui a ogni lettera corrisponde una cifra diversa, bisogna trovare la sostituzione corretta.

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

questo vincolo coinvolge tutte le variabili (non è binario):

```
(1000*S+100*E+10*N+D+
 1000*M+100*O+10*R+E) = (10000*M+1000*O+100*N+10*E+Y)
```

Note:-
Un altro problema è quello delle 8 regine.

Vincoli e criteri di preferenza:

- I vincoli possono essere più o meno *rigidi*.
- Si distingue tra vincoli veri e *criteri di preferenza*.
- Una soluzione deve soddisfare tutti i vincoli.
- Una soluzione può violare uno o più criteri di preferenza.
- Il soddisfacimento dei criteri di preferenza permette di ordinare le soluzioni identificando quelle preferibili e quelle meno preferibili.

[1] Grafi con archi che connettono più di due nodi.

Definizione 2.1.2: Generate-and-test

È un metodo di risoluzione di CSP molto semplice:

- Finché non si ha una soluzione:
  1. Genera un assegnamento completo.
  2. Controlla se è consistente.
  3. Se è una soluzione, esci dal ciclo.
  4. Se non è una soluzione, torna al passo 1.
- Se si ha una soluzione la si restituisce.
- Altrimenti fallimento.

Note:-

Però Generate-and-test è inefficiente.

Ricerca di una soluzione in profondità:

- Esploriamo lo spazio degli stati (dei possibili assegnamenti) utilizzando una ricerca depth-first con backtracking.
- Ricerca non informata + vincoli per decidere quando potare un cammino.
- La limitatezza dei cammini rende ragionevole la scelta della ricerca in profondità, tuttavia spesso si ha branching factor elevato.
- Siano:
  - n = numero di variabili.
  - d = numero medio dei valori possibili per ciascuna variabile.
  - Uno qualsiasi dei valori può essere assegnato a una qualsiasi delle variabili.
- Il branching factor sarà n*d al primo livello, (n-1)*d al secondo (perché una variabile è stata fissata), eccetera.
- Quindi l'albero avrà $n! * d^n$ foglie.

Note:-

Per migliorare si possono aggiungere euristiche generali.

Euristiche:

- *Scelta della prossima variabile:*
  - Euristica Minimum Remaining Values (o fail-first): sceglie una delle variabili con il minor numero di valori alternativi consistenti con l'assegnamento corrente.
  - Euristica di grado: sceglie la variabile coinvolta con più vincoli.
  - Euristica del valore meno vincolante: prediligere il valore che lascia più libertà alle variabili adiacenti sul grafo dei vincoli.
- *Metodo di consistenza locale:*
  - Forward checking: si percorrono gli archi che collegano il nodo, corrispondente alla variabile assegnata, con i suoi vicini diretti e si riduce il range dei possibili valori di tali vicini in maniera conforme al vincolo.
  - Node consistency: riguarda singole variabili, vale quando i vincoli unari sono soddisfatti da tutti i valori dei domini delle rispettive variabili.

- Arc consistency: proprietà direzionale relativa a un vincolo binario, tutti i valori consistenti di una variabile x possono essere estesi a y tramite i vincoli.
- Path consistency: proprietà che lega una coppia di variabili a una terza tramite i vincoli.

Vincoli speciali:

- *Alldifferent*$(X_1, \ldots, X_n)$*:* i valori delle variabili elencate devono essere tutti differenti.
- *Atmost*$(N, A_1, \ldots, A_k)$*:* le attività $A_1, \ldots, A_k$ possono impegnare complessivamente al più N risorse.

Note:-

Atmost si utilizza per domini su grandi numeri.

2.1.3 AC-3

Definizione 2.1.3: AC-3

Algoritmo di arc consistency sviluppato nel 1977 da Alan Macworth. Si può usare come preprocessing oppure a valle degli assegnamenti per propagare le scelte fatte tramite i vincoli. Quest'ultimo uso realizza l'algoritmo MAC (Maintaining Arc Consistency).

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

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

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff we remove a value
  removed $\leftarrow$ *false*
  for each x **in** DOMAIN[X_i] **do**
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint between X_i and X_j
      then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ *true*
  return *removed*

Figure 5.7 The arc consistency algorithm AC-3. After applying AC-3, either every arc is arc-consistent, or some variable has an empty domain, indicating that the CSP cannot be made arc-consistent (and thus the CSP cannot be solved). The name "AC-3" was used by the algorithm's inventor (Mackworth, 1977) because it's the third version developed in the paper.

Figure 2.1: Algoritmo AC-3.

Arc consistency:

- La propagazione dei valori tramite arc consistency è un esempio di tecnica di inferenza.
- Proprietà:
  - Consente di ridurre i domini delle variabili di molti CSP.
  - Quando riduce i domini di tutte le variabili a un solo valore trova anche una soluzione.

  - Quando rende vuoto un dominio scopre che un particolare CSP non può essere risolto.
  - Non è però generalmente sufficiente a determinare una soluzione o ad accorgersi dell'irresolvibilità di un CSP.

Valutazione di AC-3:

- Un CSP con n variabili e vincoli binari ha al più n^2 archi.
- Sia d il numero massimo di valori di una variabile: il tempo per la verifica della consistenza sarà nel caso peggiore $O(n^2d^3)$.
- Più costoso della verifica forward ma più efficace.
- AC-3 è incompleto: alcuni assegnamenti inconsistenti non vengono rilevati.

2.1.4 K-consistency

La path consistency:

- È una proprietà più forte di arc consistency.
- Identifica vincoli impliciti, inferiti da triplette di variabili.
- Una coppia di variabili $\{X_1, X_2\}$ è path consistent rispetto a una terza variabile X_3 quando:
  - Per ogni assegnamento $\{X_1 = a, X_2 = b\}$ consistente con i vincoli su X_1 e X_2.
  - c'è un assegnamento di X_3 che soddisfa i vincoli esistenti sulle coppie di variabili $\{X_1, X_3\}$ e $\{X_2, X_3\}$.

Definizione 2.1.4: K-consistency

Un CSP è k-consistent quando per ogni sottoinsieme costituito da k-1 delle sue variabili e ogni loro assegnamento consistente, è possibile individuare un assegnamento consistente per qualsiasi k-ma variabile.

È la generalizzazione delle consistente viste in precedenza:

- 1-consistency: node consistency.
- 2-consistency: arc consistency.
- 3-consistency: path consistency.

Corollario 2.1.2 CSP Fortemente K-consistent

Un CSP è fortemente k-consistent quando è:

- k-consistent.
- (k-1)-consistent.
- ...
- 1-consistent.

È dimostrato che un CSP fortemente k-consistent può essere risolto senza backtracking:

- In pratica:
  - Poiché è 1-consistent basta iniziare da un valore consistente per una variabile X_1.
  - Poiché è 2-consistent è possibile individuare un valore consistente per le variabili direttamente in relazione con X_1.
  - Poiché è 3-consistente è possibile individuare un valore consistente per le variabili in relazione con coppie delle precedenti.
- Complessità temporale: $O(n * d)$ dove n è il numero di variabili e d è il numero di valori del dominio.
- La complessità temporale per decidere se un CSP è fortemente k-consistent è esponenziale.

2.1.5 Backjumping

Limiti del backtracking:

- Quando la ricerca con backtracking raggiunge un vicolo cieco, torna indietro alla variabile che era stata considerata al passo precedente (backtracking cronologico).
- Non sfrutta i vincoli, ha i limiti delle strategie di ricerca blind.

Note:-

Idea: fare backtracking a una variabile che potrebbe risolvere il problema.

Definizione 2.1.5: Backjumping

Variante del backtracking che utilizza i conflict set per decidere a quale variabile ritornare in caso di vicolo cieco.

Corollario 2.1.3 Conflict Set

Sia A un assegnamento parziale consistente, sia X una variabile non ancora assegnata. Se l'assegnamento A U {X=vi} risulta inconsistente per qualsiasi valore vi appartenente al dominio di X si dice che A è un conflict set di X.

Note:-

Un conflict set per una variabile è minimo quando togliendo uno qualsiasi degli assegnamenti che lo costituiscono non si ottiene più un conflict set.

Il forward checking può essere modificato in modo da costruire gli insiemi dei conflitti per ogni variabile:

- Quando si assegna un valore a una variabile, FC propaga questa scelta attraverso i vincoli cancellando valori dai domini di altre variabili.
- asta arricchire FC in modo che registri la relazione fra la variabile assegnata e quelle che hanno subito una riduzione di dominio.

Note:-

FC rileva esattamente gli stessi conflitti rilevati da BJ, quindi l'uso di BJ congiunto a FC è ridondante.

Corollario 2.1.4 NOGOOD

Assegnamenti parziali che non appartengono all'insieme delle possibili soluzioni.

Conflict-directed backjumping:

- Il conflitto dipende da due fattori: (1) dagli assegnamenti precedenti e (2) dalle variabili rimaste a cui dovremo assegnare valori in futuro.
- È possibile calcolare i conflict set in modo in modo tale che guidino in modo più efficace il backtracking (evitando di considerare T=R da cui non dipende l'inconsistenza e saltando direttamente ai "colpevoli"veri)?

Backjump:

- Sia X_j la variabile corrente.
- Indichiamo con conf(X) il conflict set della generica variabile X.
- Se tutti i valori possibili di X_j falliscono si fa un backjump alla variabile X_i che è stata aggiunta a conf(X_j) più di recente e si aggiorna conf(X_i):

$$conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{x_i\}$$

Osservazioni 2.1.1

- I vincoli forniscono una guida per evidenziare relazioni implicite fra le variabili.
- Gli stati NOGOOD minimali possono essere registrati dall'algoritmo che potrà così evitare di ripetere questi assegnamenti in futuro.
- Si tratta di una forma di *apprendimento*.

2.1.6 Applicazioni di CSP

Applicazioni reali di CSP:

- Sudoku.
- Location of facilities (es. warehouses): dato un elenco di magazzini e date delle richieste fatte da clienti, identificare un percorso che permetta di soddisfare le richieste visitando i magazzini così da minimizzare i costi.
- Job scheduling: supponiamo che una fabbrica produca un range di prodotti, ogni tipologia richiede la sequenzializzazione di determinate operazioni svolte da macchinari. Ogni operazione richiede un tempo di esecuzione. Lo scopo è trovare una sequenza di produzione che minimizzi i tempi di produzione.
- Car sequencing: nell'industria automobilistica la catena di montaggio assembla automobili partendo da un modello base a cui sono aggiunte caratterizzazioni scelte dai clienti (es. Optional). Automobili diverse avranno insiemi di optional diversi. Le auto sono portate da un nastro trasportatore e gli optional sono montati in aree di lavoro, ognuna delle quali ha una capacità massima. Il problema consiste nel definire un ordine di costruzione tale da non eccedere le capacità delle aree di lavoro.
- Cutting stock problem: un materiale deve essere tagliato in pezzi più piccoli per un cliente minimizzando lo spreco.
- Vehicle routing: n clienti vengono riforniti da uno stesso deposito. Il problema consiste nel trovare il percorso di costo minimo.
- Timetabling: costruzione automatica di orari (esempio per corsi di studi) tenendo conto dell'allocazione di risorse (aule, laboratori) e altri vincoli.
- Rostering e crew scheduling: definizione di turni e di equipaggi (esempio: per compagnie aeree).
- SAT solver: risolvono problemi a variabili booleane[2].
- ASP: per programmare e risolvere CSP[3].

2.2 Rappresentazione della Conoscenza

[2] Visto in "Logica per l'Informatica".
[3] Visto in "Intelligenza Artificiale e Laboratorio".