

	BF	UC	DF	DL	ID	Bidir.
Completa	SI	SI (-)	NO	SI (+)	SI	SI
Ottimale	SI (*)	SI (-)	NO	NO	SI (*)	SI
Tempo	$O(b^d)$	$O(b^{1+\lfloor C^*/\alpha \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Spazio	$O(b^d)$	$O(b^{1+\lfloor C^*/\alpha \rfloor})$	$O(b*m)$	$O(b*l)$	$O(b*d)$	$O(b^{d/2})$

UC = BF se anche tutti con lo stesso costo. $O(b^{d+d})$

Att: $b^{1+\lfloor C^*/\alpha \rfloor}$ può essere $\gg b^d$ perché può esplorare lunghe cammini con archi poco costosi

(*) se gli operatori hanno tutti lo stesso costo

(-) per costo degli archi x tale che $x \geq \alpha > 0$

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

Best First: Completa, ottimale ne \neq , tempo: $O(b^{1+d})$, spazio: $O(b^d)$

Greedy best First: Ne completa, ne ottima, tempo: $O(b^m)$, spazio: $O(b^m)$

BEAM SEARCH: Ne ottimo ne completo, Tempo: $O(K \cdot b)$
spazio: $O(K)$

IDA: Ottimo e completo se:

- le azioni hanno costo costante k e l'f-limit viene aumentato di k
- le azioni hanno costo variabile e l'incremento di f-limit è \leq del costo minimo degli archi (ε) Tempo: $O(b^d)$ Spazio: $O(b \cdot d)$
- il nuovo f-limit è il minimo valore $f(n)$ dei nodi generati ed esclusi all'iterazione precedente

A*: costo spazio: $O(b^d)$, tempo: $O(b^d)$

• A* è completa se $g(n) \geq d(n) \cdot \alpha$ con $\alpha > 0$, cioè costo archi $\geq \alpha > 0$

• A* albero ottimale se h ammisibile cioè h è una sotto stima di $h^*(n)$
check: calcolo h^* e vedo se $\forall n, 0 \leq h(n) \leq h^*(n)$ $0 \leq h \leq h^*(n)$
ammis.

• A* su grafo ottimale se h ammis e monotona (consistente), $f(n)$ monotona crescente

$h(n)$ monotona \Rightarrow Quando n espande uno stato esso sarà sempre sul cammino ottimo
(Viene trovato per primo il camm. meno costoso)

$$\begin{aligned} &\cdot h(goal) = 0 \\ &\cdot \forall n. h(n) \leq c(n, a, n') + h'(n) \end{aligned}$$

check monod: Verifico queste proprietà $\forall n$

Caratteristiche A*

- A* espande tutti i nodi con $f(n) < C^*$
- A* espande alcuni i nodi con $f(n) = C^*$
- A* non espande alcun nodo con $f(n) > C^*$
Assumendo C^* come costo della soluzione ottima.

Attenzione:

Consistente \rightarrow Ammisibile

$\not\rightarrow$ Posso avere esempi ottimali non ammis.

\rightarrow Ottimale

Licerca A* ad albero

A* \Rightarrow completo (derivata da A)

Una euristicca h_2 è più informata (domina su) di h_1 se:

$\forall n h_2(n) \geq h_1(n)$, quindi i nodi espauriti da A* con h_2 sono un sottoinsieme di quelli espauriti con h_1
Se alcuni sì, alcuni no, non ci sono relazioni di dominanza

Fattore di ramificazione effettivo $b^* t.c N+1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$

Più alto è, più nodi espaurde ad uguale profondità, meglio se vicino a 1 ≈ 1.5

Casi speciali A:

- $f(n) = g(n) \Rightarrow UC$
- $f(n) = h(n) \Rightarrow Greedy best first$
- $f(n) = d(n) \Rightarrow BF$

DF come Best First

$$\bullet f(n) = -d(n) \Rightarrow DF$$

Satto il punto se costi archi uguali

Attenzione: A*, UC, GBF, ricorda a grafo

- 1) Se nodo in esplorati \Rightarrow Nullle
- * 2) Se nodo in frontiera \Rightarrow Sost in front. se costo <
- 3) Se nodo ne in front \Rightarrow rimuovi che in esplor. in front.

IDA Algo

IDA* combina A* con ID: ad ogni iterazione si ricerca in profondità con un limite (cut-off) dato dal valore della funzione $f(n)$ e non dalla profondità. Il valore f-limit viene aumentato ad ogni iterazione fino a trovare la soluzione (come in ID si aumenta di un livello alla volta). Ma di quanto aumento?

- Nel caso di costo fisso di ogni azione il limite viene aumentato di questo costo.
- Nel caso in cui i costi siano variabili il limite viene aumentato del costo minimo oppure ad ogni passo scelgo il valore minimo delle $f(n)$ scartate perché in quanto superavano il limite all'iterazione precedente.

Invenzione empirica

- 1) Rilassamento dei vincoli del prob.
- 2) Max delle heuristiche ammissibili
- 3) Somma dei costi dei pattern disgiunti in un db
- 4) Combinazione lineare $h(n) = c_1h_1(n) + \dots + c_kh_k(n)$
- 5) Apprendere dall'esperienza

Hill climbing: Ne ottima, Ne completa, tempo: $O(\infty)$, spazio: $O(b)$

→ Prende solo mosse che migliorano $f(n)$, cioè la funz. obb. (Vogliamo massimizzarla o minimizzarla) non ci interessa del cammino fatto

Possibili versioni migliorate:

- Consente un numero limitato di mosse laterali, cioè l'algoritmo si modifica fermandosi quando $nodoVicino.Valore < nodoCorrente.Valore$ invece che \leq .
- Si sfrutta Hill-Climbing stocastico scegliendo a caso tra le mosse in salita. L'algoritmo converge più lentamente ma a volte trova soluzioni migliori.
- Si sfrutta Hill-Climbing con prima scelta, può generare mosse a caso fino a trovarne una migliore dello stato corrente, diventa più efficace quando i successori sono molti.
- Si sfrutta Hill-Climbing con rinvio casuale (random restart), cioè si riparte da un punto scelto a caso. Se la probabilità di successo è p saranno necessarie in media $1/p$ ripartenze per trovare la soluzione. Questo algoritmo è tendenzialmente completo.

Tempo Simulato: Fa una mossa in salita, che discesa. Quelle in discesa diminuiscono nel tempo.

Algo:

- se migliora lo stato viene espanso (migliora lo stato significa che la funzione di valutazione è maggiore, cioè $[\Delta E = f(n') - f(n)] \geq 0$)
- altrimenti se peggiora lo stato (caso $[\Delta E = f(n') - f(n)] < 0$) scelgo il nodo n' con probabilità $p = e^{\Delta E/T}$ dove T è un parametro che nel progredire dell'algoritmo decresce, facendo così decrescere anche p rendendo improbabili le mosse peggiorative.

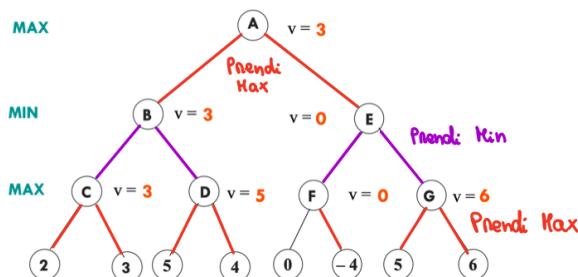
T è un parametro dato in input che decresce man mano.

Se decresce lentamente si raggiunge un ottimo globale con prob tendente a 1.

Algo Min-Max

ricorreva

Ricerca in profondità prendendo il max-value al livello max e min value al livello min.



Tempo: $O(b^m)$

Spazio: $O(m)$

Pratica alfa beta

Si initializza la coppia Max Value (α) e Min Value (β) con: $\alpha = -\infty$ e $\beta = +\infty$ e man mano mi salvo i valori min e massimi. Taglio quando

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

Att: Taglio gli archi succ., non quelli orco!

Tempo: $O(b^{m/2})$ Spazio: $O(m)$ → può arrivare a prof. doppia rispetto a Min-Max

Ottimale delle mosse per la postura:

- 1) Nodi max: Nodi ordinati per valore minmax decrescente }
- 2) Nodi min: Nodi ordinati per valore minmax crescente }

Problemi CSP

- Problema: descritto da tre componenti
 1. X insieme di variabili
 2. D insieme di domini (ogni dominio D_i è a sua volta un insieme di valori possibili per la variabile X_i)
 3. C insieme di vincoli (ogni vincolo C_i è una coppia [variabili, relazione])
- Stato: un assegnamento parziale o completo di valori a variabili (ad esempio $X_i = v_i, X_j = v_j \dots$)
- Stato iniziale: {}
- Azioni: assegnamento di un valore ad una variabile
- Soluzione: un assegnamento completo della formula, dove tutte le variabili hanno un valore consistente, cioè dove tutti i vincoli sono soddisfatti

Ricerca: Ad ogni passo si amena una variabile, usiamo DL con backtracking

Max Profondità: Numero di variabili

Ampiezza spazio di ricerca: $|D_1| \cdot |D_2| \cdot \dots \cdot |D_n|$

Numero foglie: d^n

Vincoli: $x_i \neq 3$ (Vincolo unario)

$x_i \neq x_2$ (Vincolo binario)

1) 2.5.1 Select-Unassigned-Variable()

Scelta delle variabili: qual è la prossima variabile da scegliere?

- Se non (la trovo) MRV (Minimum Remaining Values): scegliere la variabile che ha meno valori legali (residui) da poter assegnare, cioè la variabile più vincolata. Si scoprono prima i fallimenti (fail first)!
- Euristica del grado: scegliere la variabile coinvolta in più vincoli con le altre variabili (la variabile più vincolante o di grado maggiore). (Usata a parità di MRV)

2) 2.5.2 Order-Domain-Values()

Scelta dei valori: una volta scelta la variabile, come scegliere il valore da assegnare?

- Valore meno vincolante: scegliamo il valore che esclude meno valori per le altre variabili direttamente collegate alla variabile scelta. Meglio valutare prima un assegnamento che ha più probabilità di successo, se volessimo tutte le soluzioni l'ordine non sarebbe importante.

3) 2.5.3 Inference()

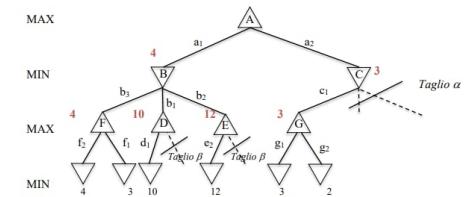
Propagazione di vincoli: qual è l'influenza di un assegnamento sulle altre variabili?

- Verifica in avanti (Forward Checking o FC): assegnato un valore ad una variabile, si possono eliminare i valori incompatibili per le altre variabili direttamente collegate da vincoli.
- Consistenza di nodo e arco: si restringono i valori dei domini delle variabili tenendo conto dei vincoli unari e binari su tutto il grafo (se una variabile ad un certo punto ha il dominio ridotto all'insieme vuoto, si esegue immediatamente backtracking). Altrimenti si itera finché tutti i nodi ed archi sono consistenti).

4) 2.5.4 Backtrack()

Quando la ricerca arriva ad un assegnamento che viola i vincoli, come posso evitare di ripetere in futuro lo stesso errore?

- Backtracking Cronologico: viene cambiato il valore dell'ultima variabile assegnata, continuando a fallire fino al soddisfacimento dei vincoli. (Problema se non è l'ultima var. che crea il problema)
- Backtracking Intelligente: si considerano alternative solo per le variabili che hanno causato il fallimento (X_i, X_j, \dots), cioè un "insieme dei conflitti". Ad esempio, nel problema delle regine, si parte con tutte le variabili assegnate (tutte le regine sulla scacchiera) e ad ogni passo si modifica l'assegnamento ad una variabile per cui un vincolo è violato (si muove una regina minacciata su una colonna). Questo è un algoritmo di riparazione euristica. Un'altra euristica potrebbe consistere nello scegliere un nuovo valore che crea meno conflitti (euristica dei conflitti minimi).



2.5 Codice Algoritmo backtracking

```

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

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp) Quala variabile scegliere? ①
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) Quali valori scegliere? ②
        if value is consistent with assignment then controllo anticipo
            add {var = value} to assignment
            inferences ← INFERENCE(csp, var, value) Qual è l'influenza di un assegnamento sulle altre variabili? come restringe i domini? ③
            if inferences ≠ failure then
                add inferences to assignment
                result ← BACKTRACK(assignment, csp) Come evitare di ripetere i fallimenti? ④
                if result ≠ failure then
                    return result
                remove {var = value} and inferences from assignment
            return failure
    
```

Esempio:

Soluzione

Variabili: Rossa, Bianca, Gialla, Svizzero, Italiano, Greco, Farfalle, Serpente, Gatto
Domini: {1, 2, 3}

Vincoli:

Gialla = Svizzero

Bianca = 1

Serpente = Greco - 2

Farfalle = Serpente ≠ Gatto

Italiano ≠ Greco ≠ Svizzero

Bianca ≠ Rossa ≠ Gialla

Farfalle ≠ Serpente ≠ Gatto

Risolviamo con MRV (a parità di MRV euristica del grado) e FC. Risolviamo subito i vincoli unari.

Step	Rossa	Bianca	Gialla	Svizzero	Italiano	Greco	Farfalle	Serpente	Gatto
0	{1, 2, 3}	{1}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}
1	{2, 3}	1	{2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}
2	{3}	1	2	{2}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}
3	{3}	1	2	{2}	{1, 3}	{1, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}
4	3	1	2	2	{1, 3}	{1, 3}	{1, 2, 3}	{1, 2, 3}	{1, 2, 3}
5	3	1	2	2	{1}	{1}	{1, 2}	{1}	{1, 2, 3}
6	3	1	2	2	{1}	3	{2}	1	{3}
7	3	1	2	2	1	3	{2}	1	{3}
8	3	1	2	2	1	3	2	1	{3}
9	3	1	2	2	1	3	2	1	3

Step 0. Abbiamo risolto i vincoli unari riducendo i domini di Bianca e di Gatto.

Step 1. Per MRV scegliamo Bianca. Assegniamo Bianca=1 e applichiamo FC.

Step 2. Per MRV potremmo scegliere Rossa, Bianca o Gatto. Scegliamo Gialla per il grado. Poi FC.

Step 3. Per MRV e grado scegliamo Svizzero. Valore 2.

Step 4. Per MRV Rossa = 3

Step 5. MRV: Italiano, Greco o Gatto. Meglio Greco per il grado. Greco = 1. Il FC, che ci impone Serpente = Greco - 2 fallisce: il dominio di Serpente si svuota. Devo fare backtracking e provare con Greco = 3.

Step 6. MRV: Italiano o Serpente. Serpente = 1.

Step 7. Italiano = 1

Step 8. Farfalle = 2

Step 9. Gatto = 3

Tipi di agente:

- 1) Reattivo semplice: Si basa solo sulla percezione corrente, sceglie le azioni in base alle regole if-else programmate. L'ambiente deve essere compl. osservabile!
- 2) Basato su modello: Mantiene lo stato del mondo esterno aggiornandolo in base a:
 - Storia delle percezioni, come il mondo si evolve, cosa fa l'azione e la percezione corrente
 - Decide l'azione con regole if-else basate sullo stato del mondo agg.
- 3) Con obiettivo: Sono guidati dal raggiungere un certo obiettivo.
 - Mantengono lo stato del mondo estero e si preoccupano di come cambierà in base alle azioni.
 - Sceglie l'azione in base allo stato attuale e a come cambierà il mondo estero con quell'azione.
- 4) Con valutazione di utilità: L'agente ha più stati obiettivo e sceglie verso quale muoversi in base alla funzione di utilità, la quale dà la misura di performance (arriva un num. reale allo stato).
 - Le azioni scelte sono basate sull'obb. da raggi. che viene scelto in base alla funz. di utilità.

↓
Tiene conto dell'utilità e le prob. di successo.

- 5) Che apprendono: È presente una componente di apprendimento che produce cambiamenti al programma agente, migliora le prestazioni adattando i suoi componenti, apprendendo dall'ambiente. L'elemento esecutivo è il programma agente, l'elemento critico osserva e dà feedback sul comportamento. Infine è presente un generatore di problemi, suggerisce nuove situazioni da esplorare.

Proprietà dell'ambiente:

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

Descrizione PEAS dei problemi

- Descrizione PEAS dei problemi
- P erformance | prestazione
 - E nvironment | ambiente
 - A ctuators | attuatori
 - S ensors | sensori

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

Un problema può essere definito formalmente mediante 5 componenti:

1. Stato iniziale → Nodo
 2. Azioni possibili nello stato ($Azioni(stato)$) → Anchi uscenti da un nodo
 3. Modello di transizione: $Risultato(stato, azione) = nuovo_stato \rightarrow$ Nodo
 4. Test obiettivo: insieme di stati obiettivo ($GoalTest(stato) = \{true, false\}$) → Nodo/lo obiettivo
 5. Costo del cammino: somma dei costi delle azioni, costo di passo definito come $c(s, a, s') \rightarrow$ Somma Costi anche fino a s'
- 1, 2 e 3 definiscono implicitamente lo spazio degli stati.

Formulazione di un problema