



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Strategie di ricerca informate

Federico Chesani

DISI

Department of Informatics – Science and Engineering

STRATEGIE INFORMATE

- L'intelligenza di un sistema non è misurabile in termini di capacità di ricerca, ma nella capacità di utilizzare conoscenza sul problema per eliminare il pericolo dell'esplosione combinatoria.
- Se il sistema avesse un qualche controllo sull'ordine nel quale vengono generate le possibili soluzioni, sarebbe allora utile disporre questo ordine in modo che le soluzioni vere e proprie abbiano un'alta possibilità di comparire prima.
- L'intelligenza, per un sistema con capacità di elaborazione limitata consiste nella saggia scelta di cosa fare dopo...



STRATEGIE INFORMATE

- I metodi di ricerca precedenti in uno spazio di profondità d e fattore di ramificazione b risultano di complessità spaziale proporzionale a b^d per trovare un goal in una delle foglie.
- Questo è inaccettabile per problemi di una certa dimensione.
- Idea: invece di espandere i nodi in modo qualunque utilizziamo conoscenza euristica sul dominio (funzioni di valutazione) per decidere quale nodo espandere per primo.



STRATEGIE INFORMATE

Le funzioni di valutazione danno una stima computazionale dello sforzo per raggiungere lo stato finale. In particolare:

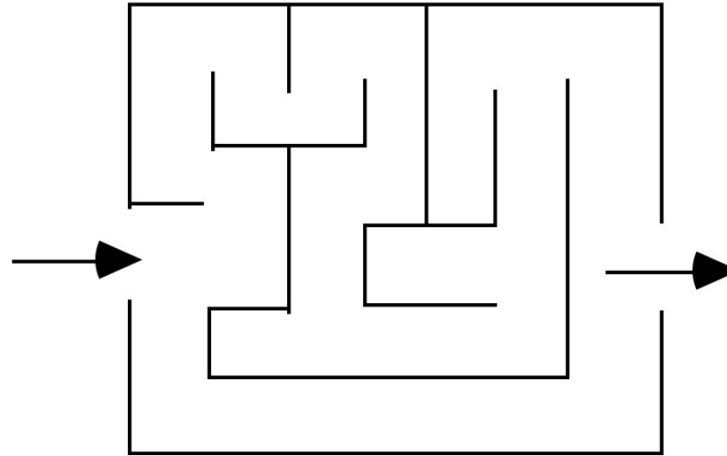
- Il tempo speso per valutare mediante una funzione euristica il nodo da espandere deve corrispondere a una riduzione nella dimensione dello spazio esplorato.
- Trade-off fra tempo necessario nel risolvere il problema (livello base) e tempo speso nel decidere come risolvere il problema (meta-livello).
- Le ricerche non informate non hanno attività di meta-livello.

PROBLEMI:

- Come trovare le funzioni di valutazione corrette, cioè come si fa a valutare bene quale è il nodo più "promettente"?
- È difficile caratterizzare numericamente la conoscenza empirica sul problema.
- Non sempre la scelta più ovvia è la migliore.



ESEMPIO



- Scelta euristica: muoversi sempre per ridurre la distanza dall'uscita.
- Nel caso di questo labirinto sceglieremmo la strada più lunga.



ESEMPIO: GIOCO DEL FILETTO

- La distanza dal goal è stimata in base al numero di caselle fuori posto:

stato

1	2	3
7	8	4
6		5

goal

1	2	3
8		4
7	6	5

- Sinistra: distanza 2
 - Destra: distanza 4
 - Alto: distanza 3
- Non è detto che la distanza stimi esattamente il numero di mosse necessarie

1	2	3
7		4
6	8	5

Distanza=3 ma in realtà sono necessarie 4 mosse per arrivare alla soluzione.



Ricerca Best-First

Best-first significa scegliere come nodo da espandere per primo quello che sembra più desiderabile.

- Usa una funzione di valutazione che calcola un numero che rappresenta la desiderabilità relativa per l'espansione di un nodo.
- Cerca di muoversi verso il massimo (minimo) di tale funzione che "stima" il costo per raggiungere il goal.
- QueuingFn inserirà i successori di un nodo espanso in ordine decrescente di desiderabilità.

Casi particolari in base alla funzione di valutazione adottata:

- ricerca **greedy** (golosa)
- ricerca **A***



Una realizzazione della ricerca Best-first

function BEST-FIRST-SEARCH(*problem*, EVAL-FN) **returns** a solution sequence

inputs: *problem*, a problem

Eval-Fn, an evaluation function

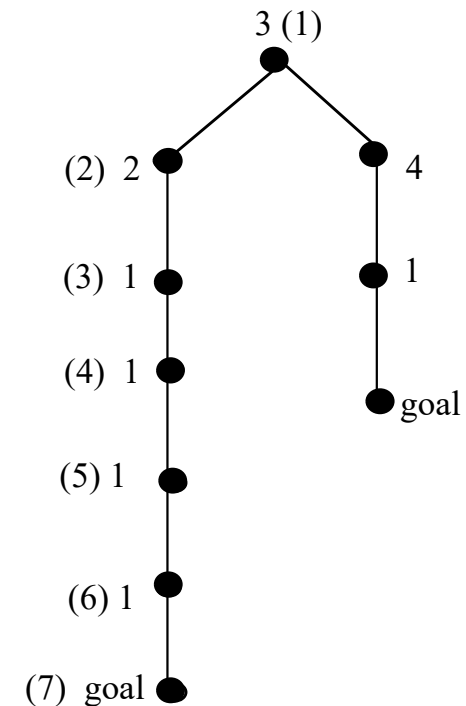
Queueing-Fn \leftarrow a function that orders nodes by EVAL-FN

return GENERAL-SEARCH(*problem*, *Queueing-Fn*)

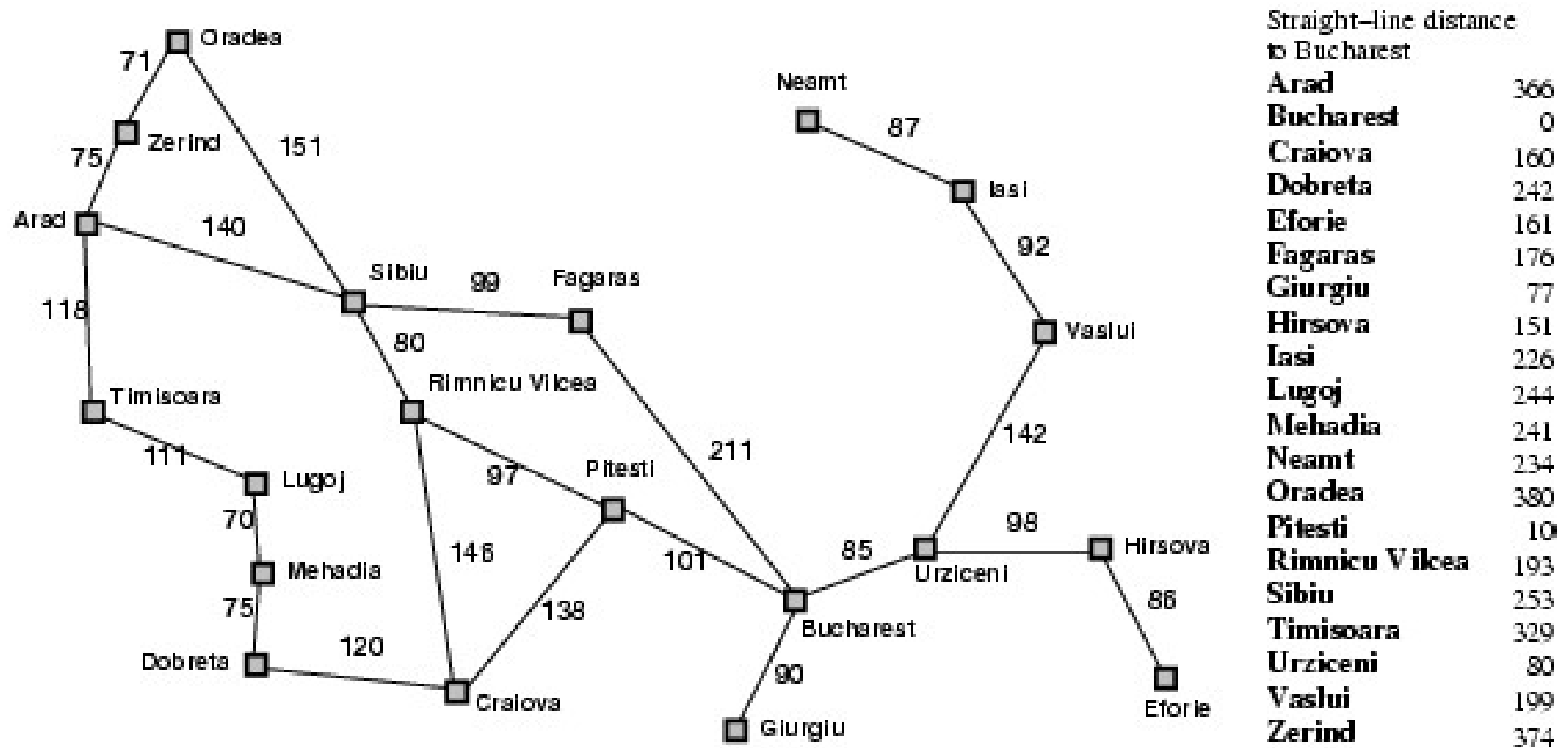
usa l'algoritmo di ricerca generale e la **funzione di valutazione** *EvalFn*

RICERCA Greedy-Best-first

- Utilizza per la funzione di valutazione la distanza (stimata) di un nodo n dal goal.
- Non considera il costo per raggiungere il nodo n dallo stato iniziale.
- La ricerca greedy non è detto che trovi la soluzione migliore, ovvero il cammino migliore per arrivare alla soluzione (si pensi all'esempio del labirinto).
- Questo perché la tecnica greedy cerca di trovare il più presto possibile un nodo con distanza 0 dal goal senza curarsi di trovare quel nodo che ha profondità più bassa.



Esempio Romania



Da: S. Russell & P. Norvig: "Intelligenza Artificiale: un approccio moderno", Pearson ed.



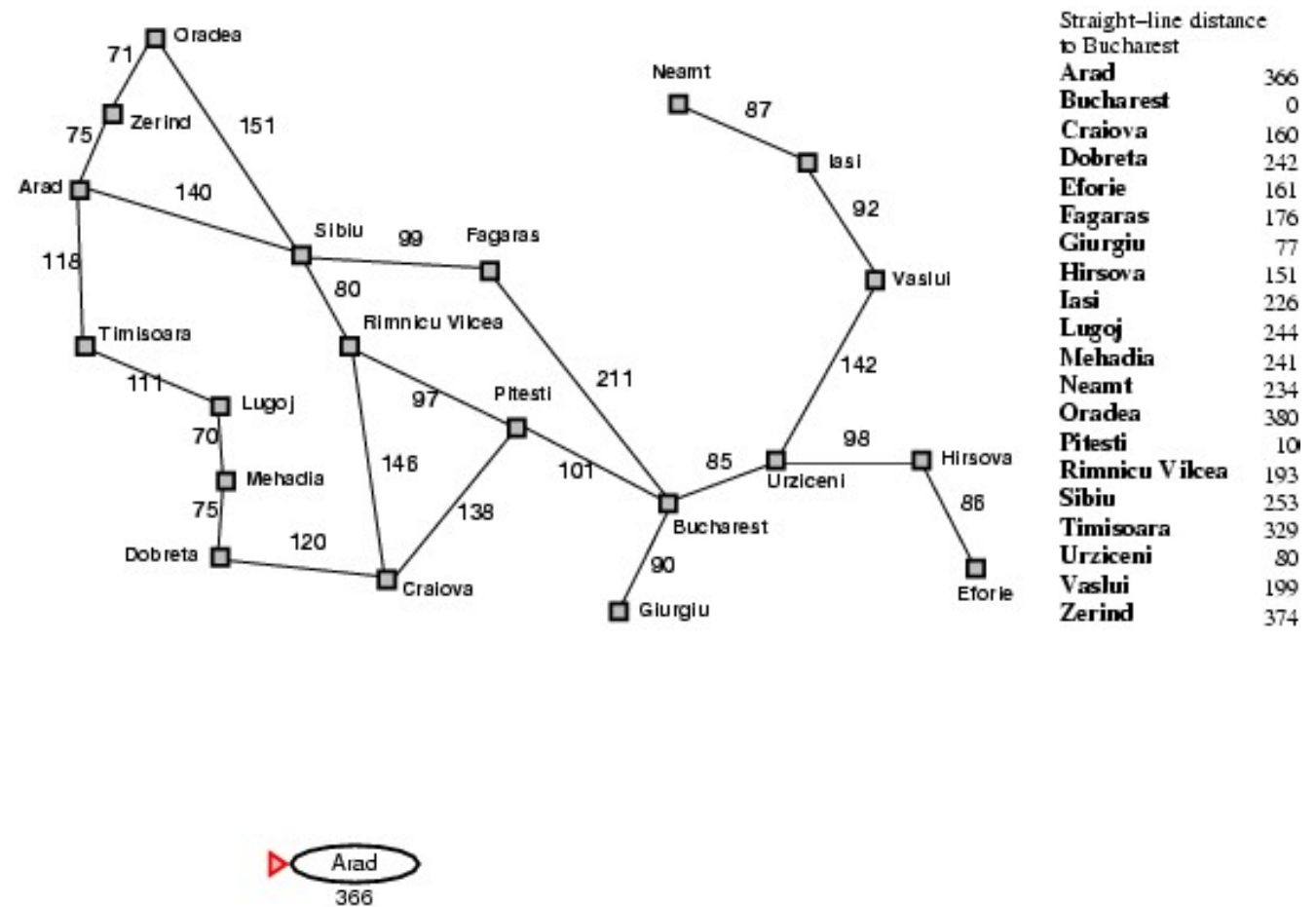
Ricerca greedy best-first

- Funzione di valutazione $f(n) = h(n)$
- $h(n)$ (euristica) = stima del costo da n al goal
- e.g., $h_{\text{SLD}}(n)$ = distanza in linea d'aria fra n e Bucharest

La ricerca greedy best-first espande il nodo che sembra più vicino al goal



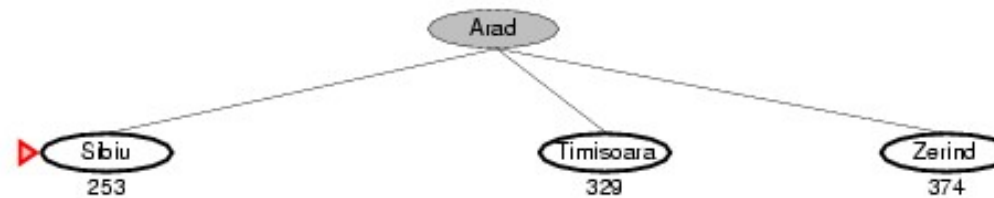
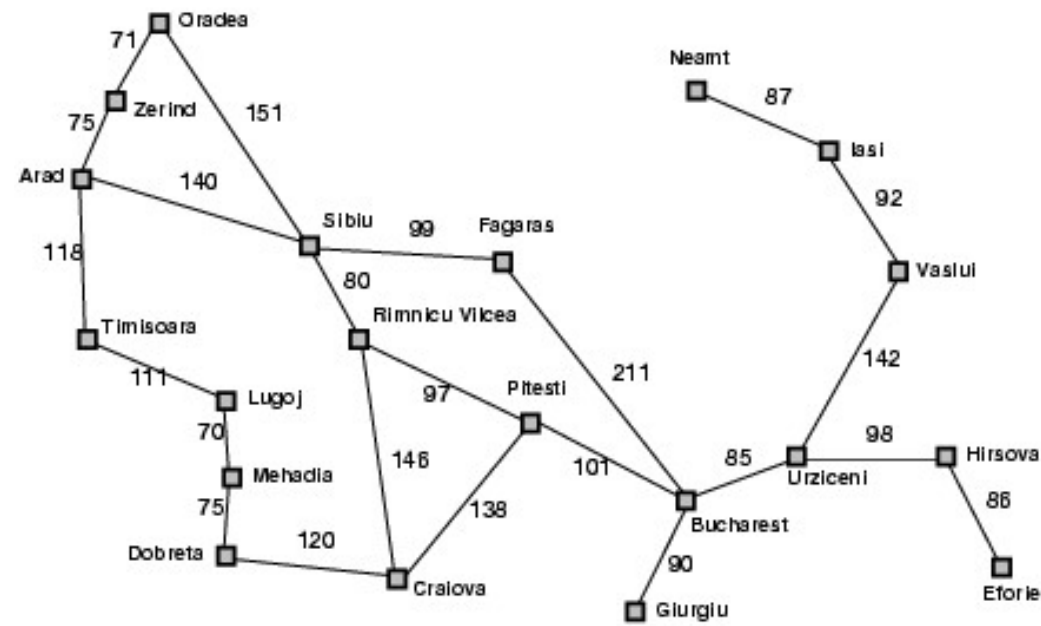
Greedy best-first: esempio



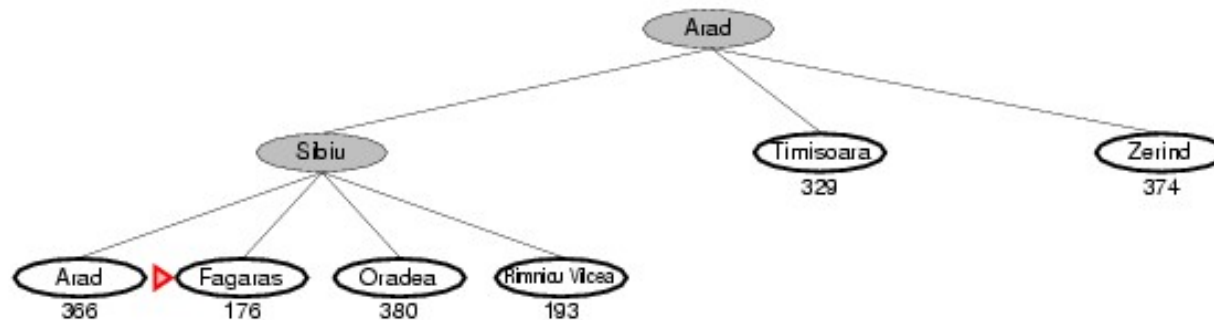
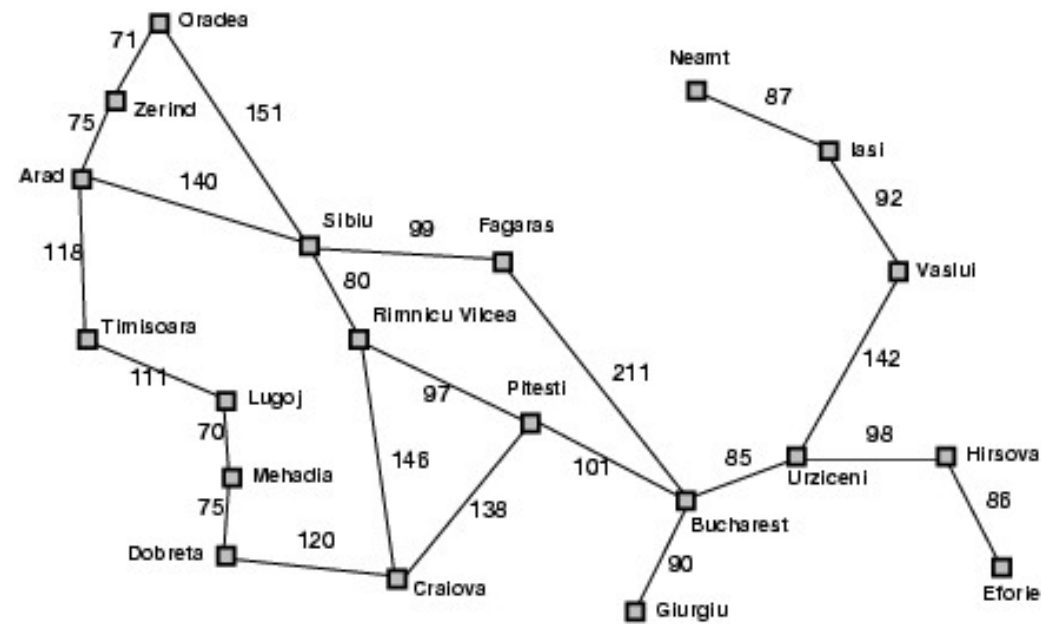
Da: S. Russell & P. Norvig: "Intelligenza Artificiale: un approccio moderno", Pearson ed.



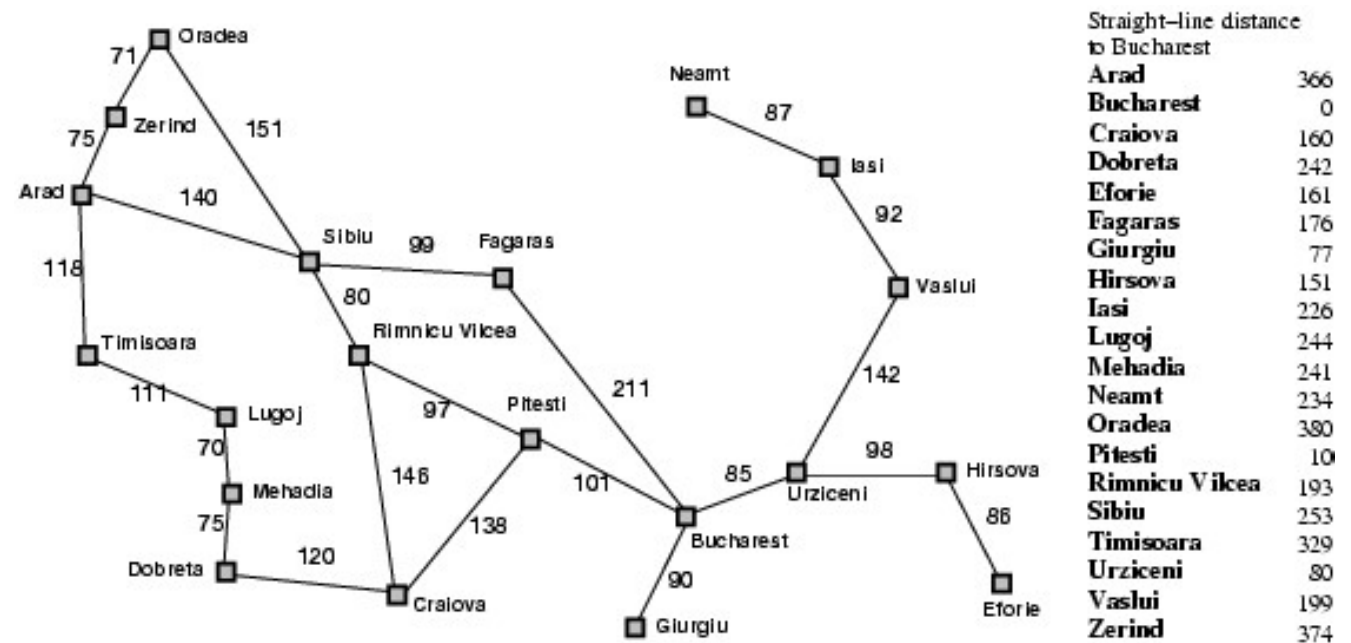
Greedy best-first: esempio



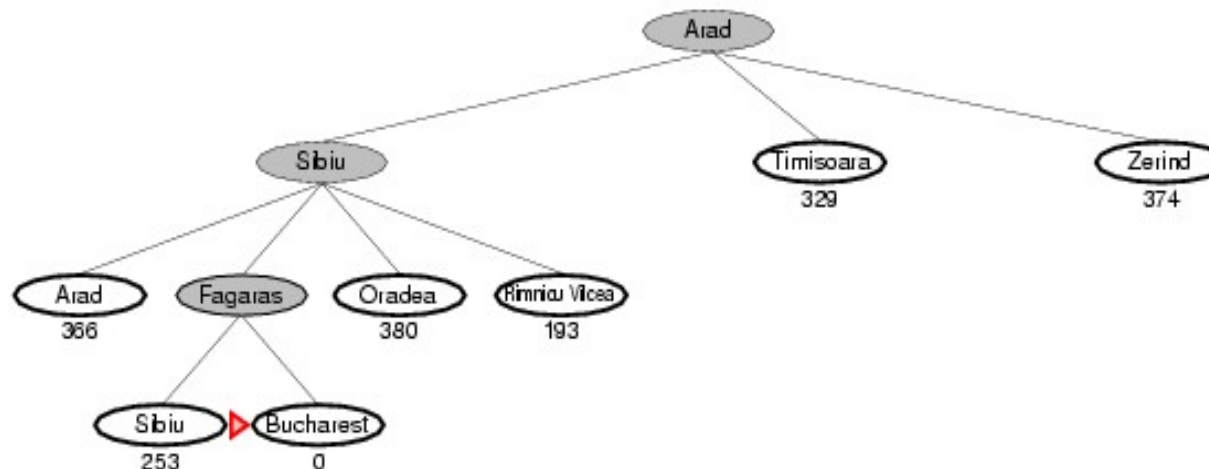
Greedy best-first: esempio



Greedy best-first: esempio



Nota: non è la strada “migliore” in termini di Km percorsi (meglio passare da Rimniku Vilcea e Pitesti).



Da: S. Russell & P. Norvig: "Intelligenza Artificiale: un approccio moderno", Pearson ed.



Strategia A*

Invece di considerare solo la distanza del nodo n dal goal, considera anche il "costo" nel raggiungere il nodo n dalla radice.

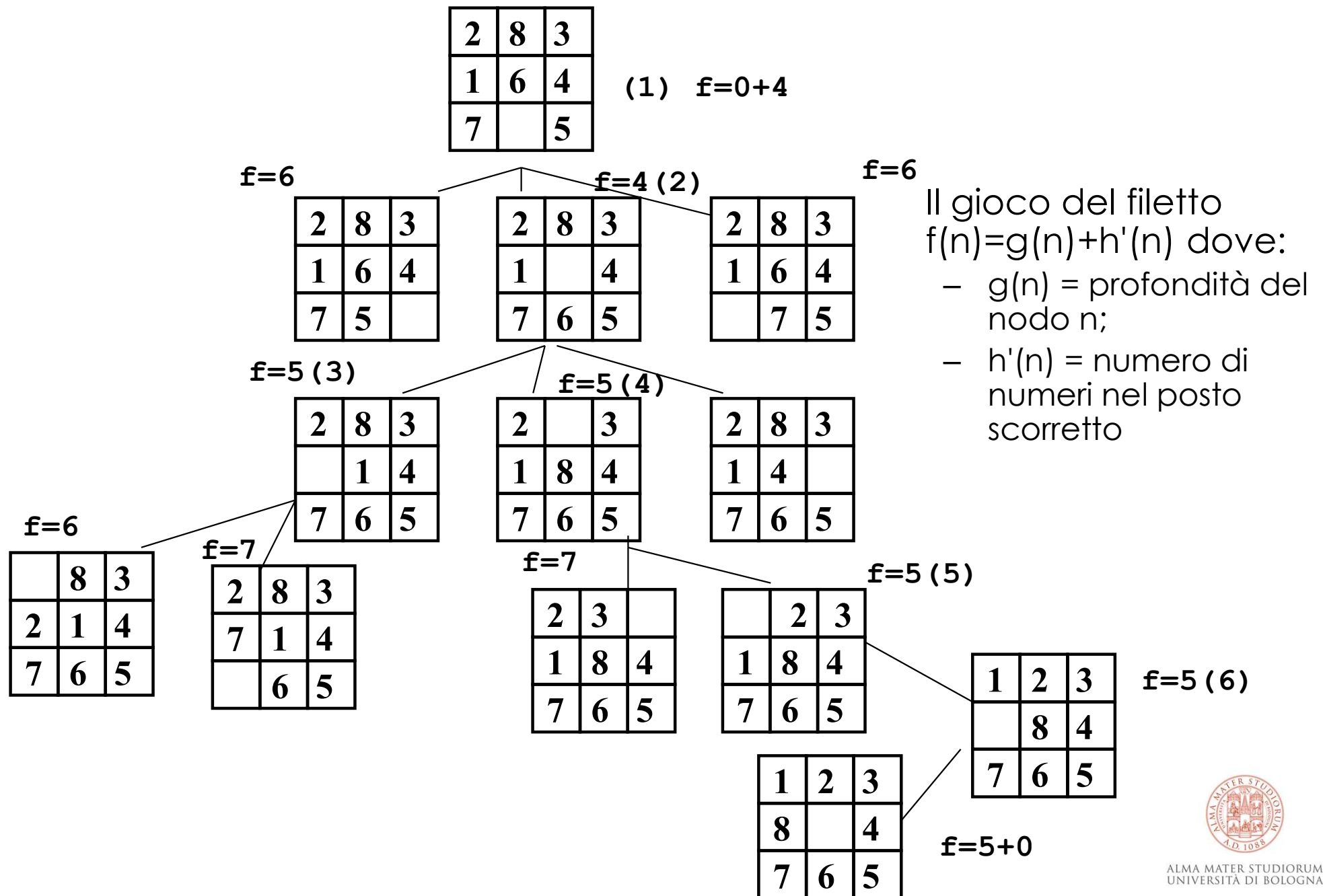
- Espandiamo quindi i nodi in ordine crescente di

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

- Dove $g(n)$ è la profondità del nodo e $h'(n)$ la distanza dal goal.
- Scegliamo come nodo da espandere quello per cui questa somma è più piccola.
- In pratica cerchiamo di combinare i vantaggi della ricerca in profondità (efficienza) e della ricerca a costo uniforme (ottimalità e completezza).

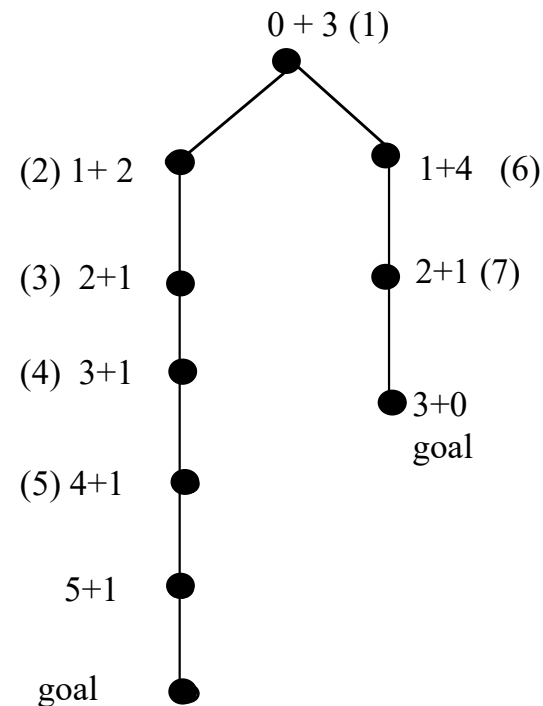


Esempio



Algoritmo A*

In questo esempio però l'algoritmo non garantisce di trovare la strada ottima. Nell'esempio, se il nodo con label 5 fosse stato il goal sarebbe stato trovato prima di quello sulla strada destra (ottimale). Perché?



Algoritmo A*

A* non garantisce di trovare la soluzione ottima, poiché dipende dalla funzione euristica.

- Si supponga di indicare con $h(n)$ la vera distanza fra il nodo corrente e il goal.
- La funzione euristica $h'(n)$ è ottimistica se abbiamo sempre che $h'(n) \leq h(n)$.

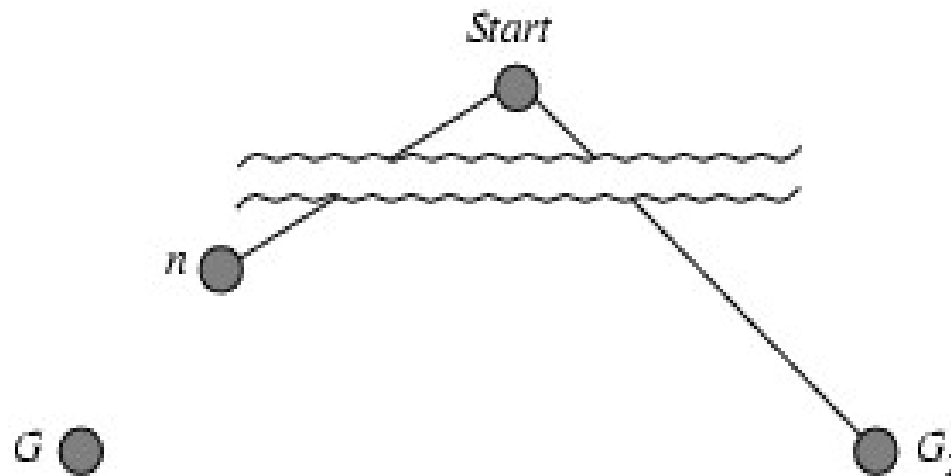
Tale funzione euristica è chiamata ammissibile.

Teorema:

- Se $h'(n) \leq h(n)$ per ogni nodo, allora l'algoritmo A* troverà sempre il nodo goal ottimale (in caso di alberi -TREE-SEARCH).
- Ovviamente l'euristica perfetta $h' = h$ è sempre ammissibile.
- Se $h' = 0$ otteniamo sempre una funzione euristica ammissibile → ricerca breadth-first (ovviamente la depth-first non lo sarà mai).

Ottimalita` di A^* (dimostrazione)

Si supponga di avere generato un goal sub-ottimo G_2 e di averlo inserito nella coda. Sia n un nodo non espanso nella coda tale che n è nella strada più breve verso il goal ottimo G .

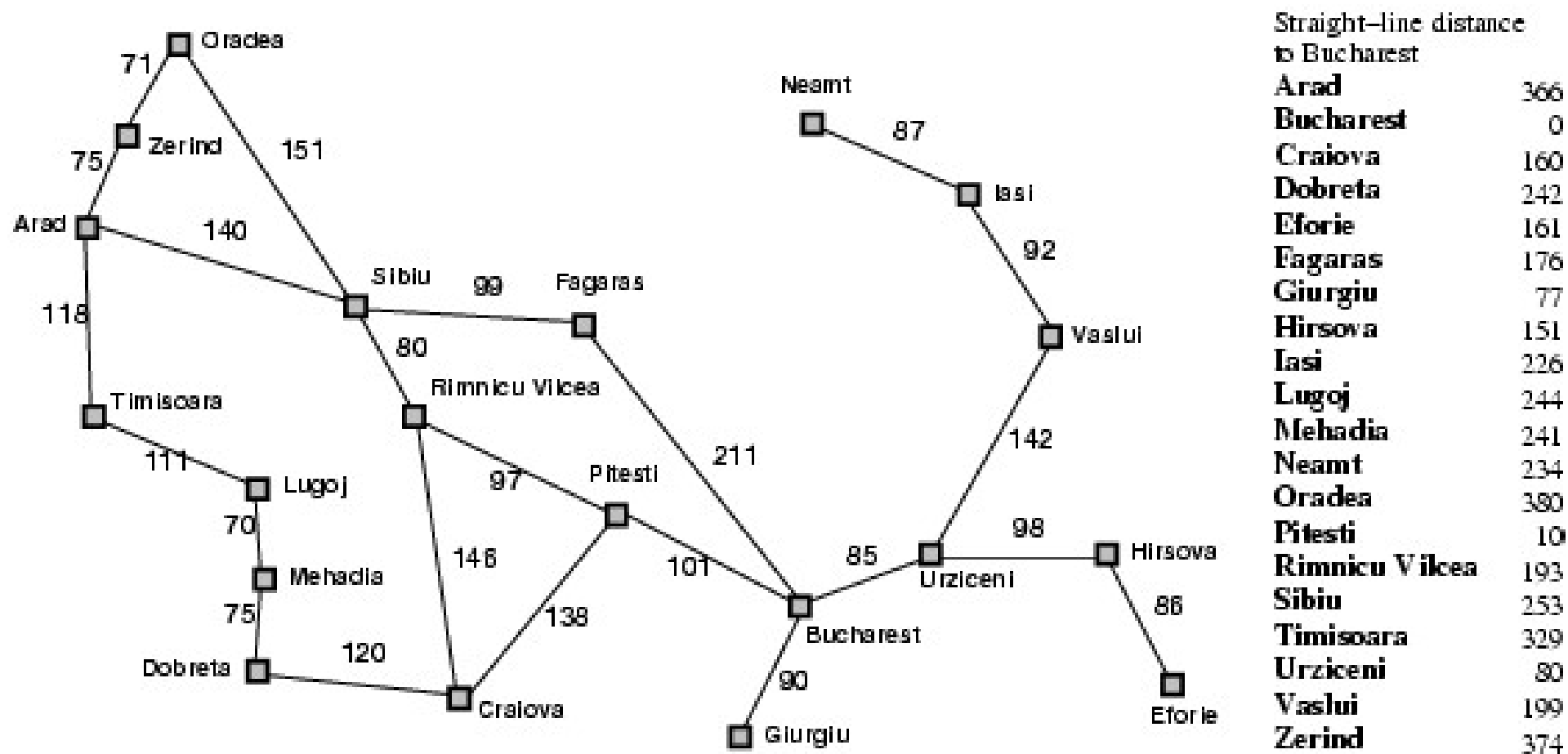


Ottimalita` di A^* (dimostrazione - cont)

- $f(G_2) = g(G_2)$ poichè $h(G_2) = 0$
 - $g(G_2) > g(G)$ poichè G_2 è sub-ottima
 - $f(G) = g(G)$ poichè $h(G) = 0$
 - $f(G_2) > f(G)$ da sopra
-
- $h'(n) \leq h(n)$ poichè h' è ammissibile
 - $g(n) + h'(n) \leq g(n) + h(n) = f(G)$ (n è sulla strada di G)
 - $f(n) \leq f(G)$
-
- quindi $f(G_2) > f(n)$, e A^* non selezionerà mai G_2 per l'espansione.



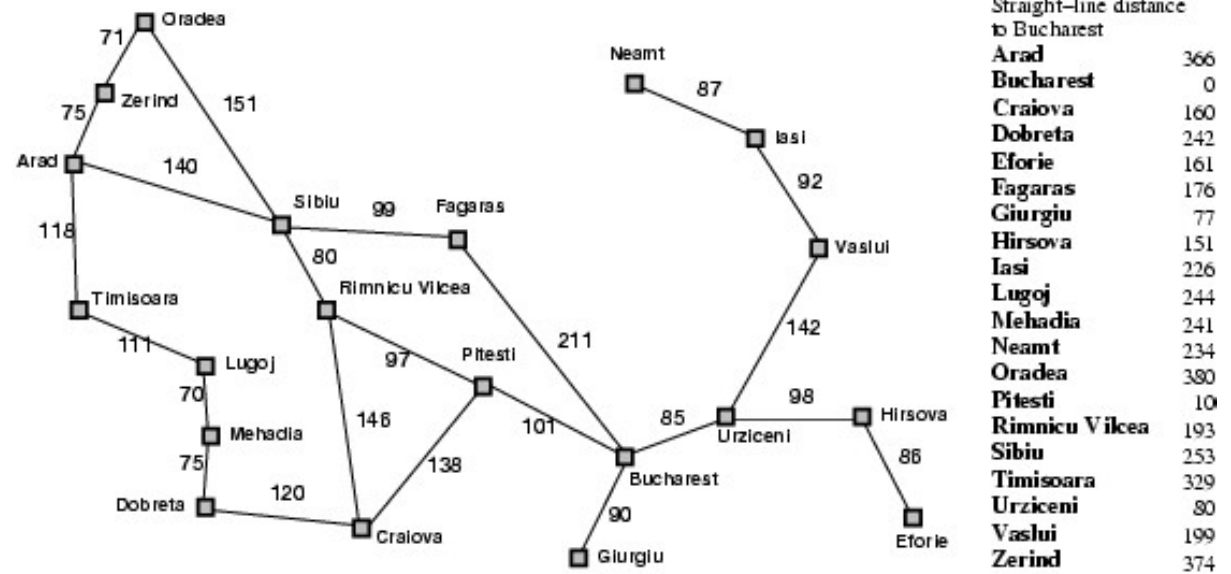
Esempio Romania A* con funzione ammissibile



Da: S. Russell & P. Norvig: "Intelligenza Artificiale: un approccio moderno", Pearson ed.

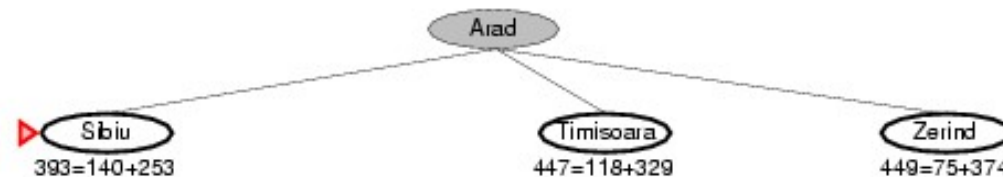
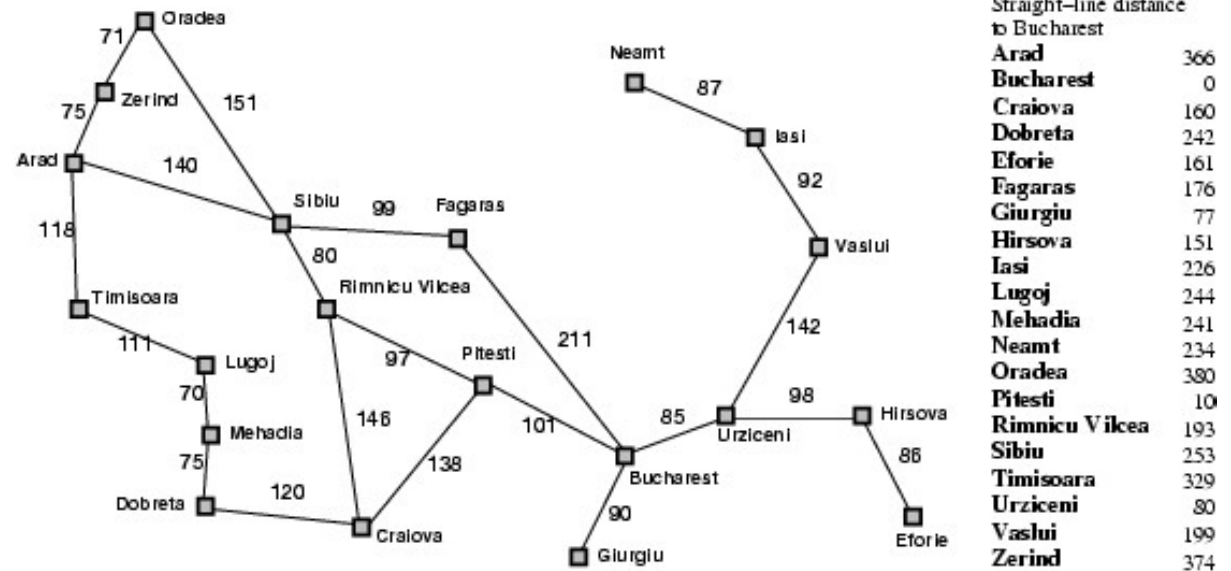


A* esempio

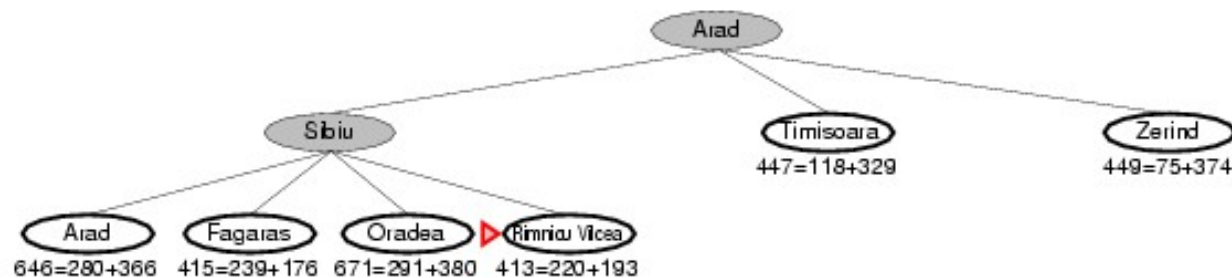
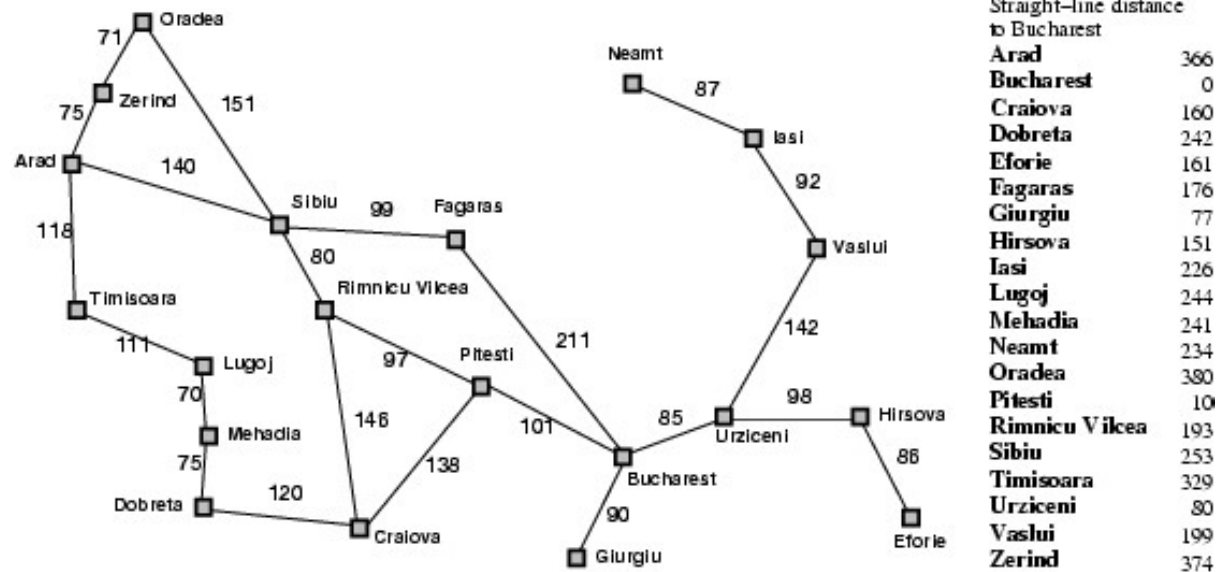


Arad
366=0+366

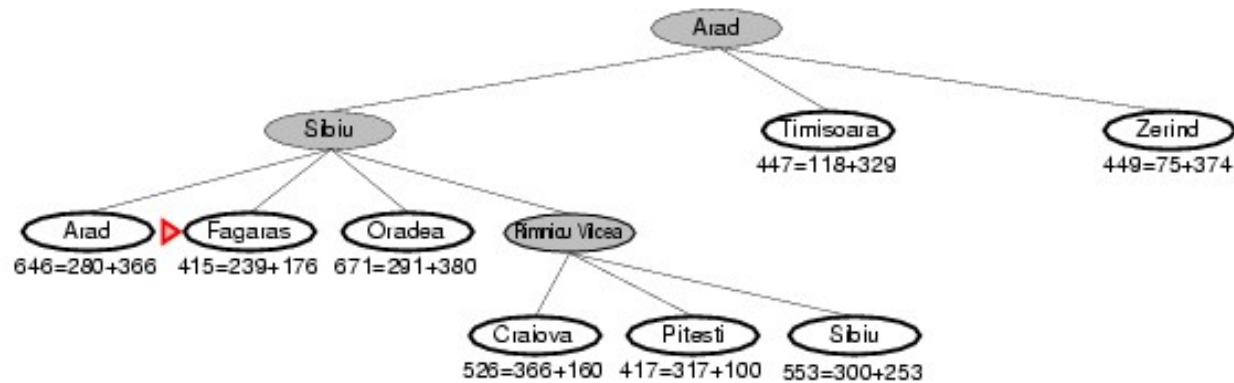
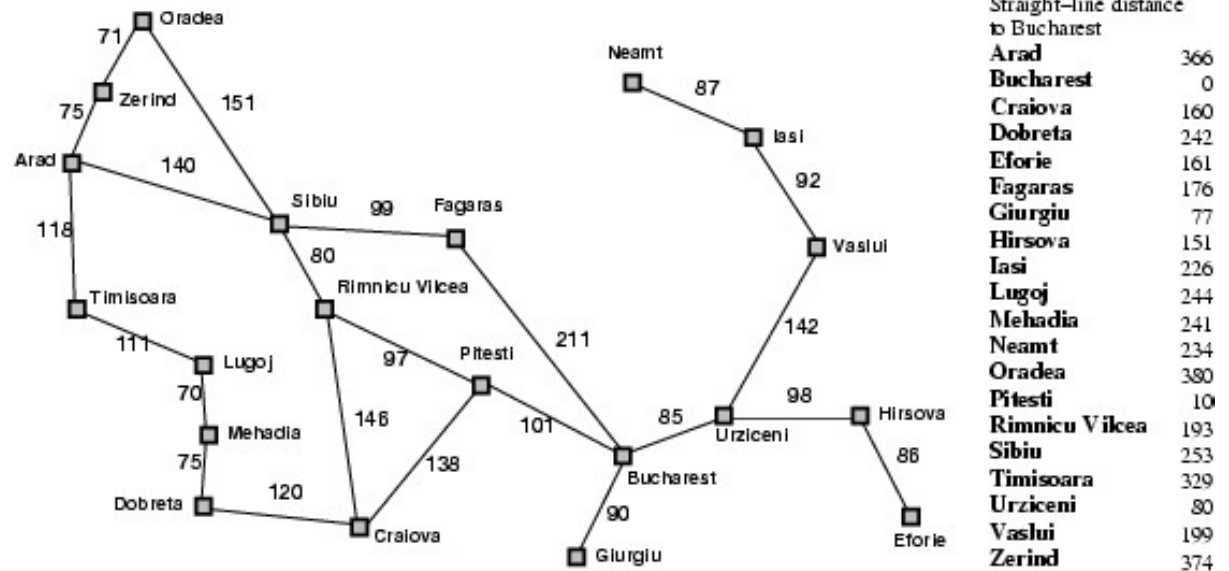
A* esempio



A* esempio



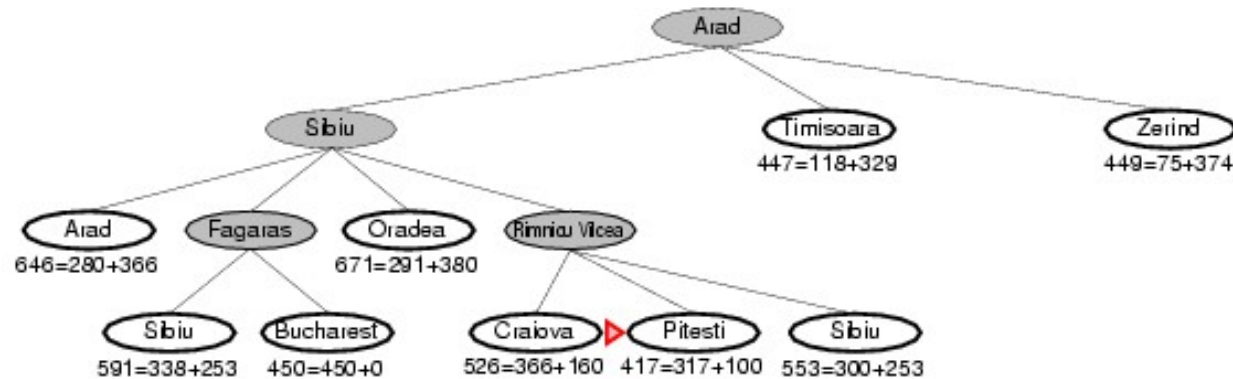
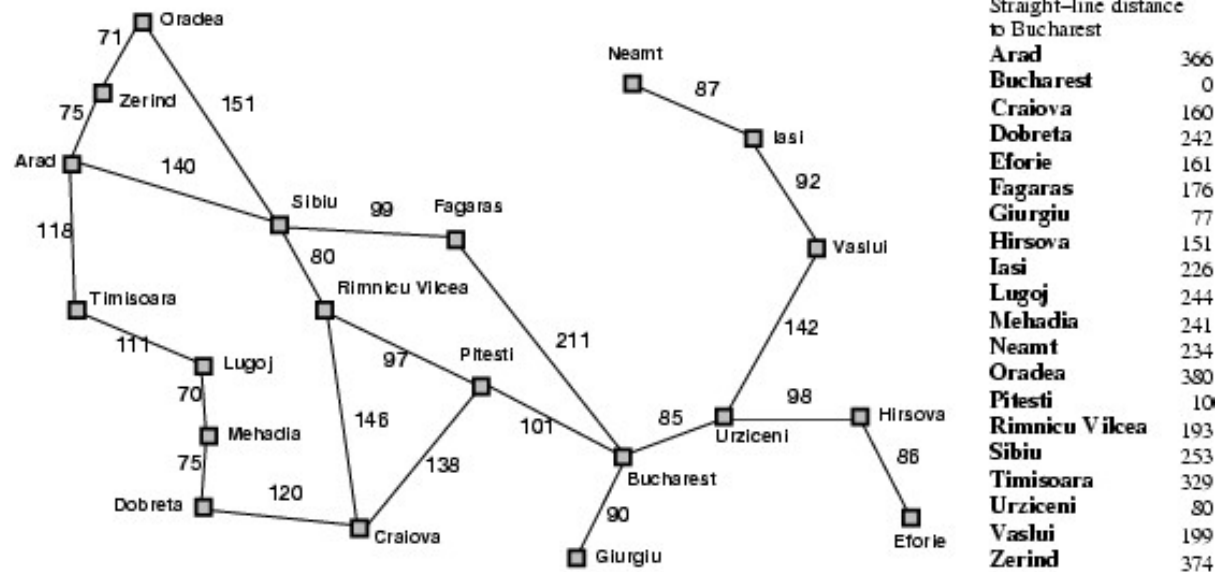
A* esempio



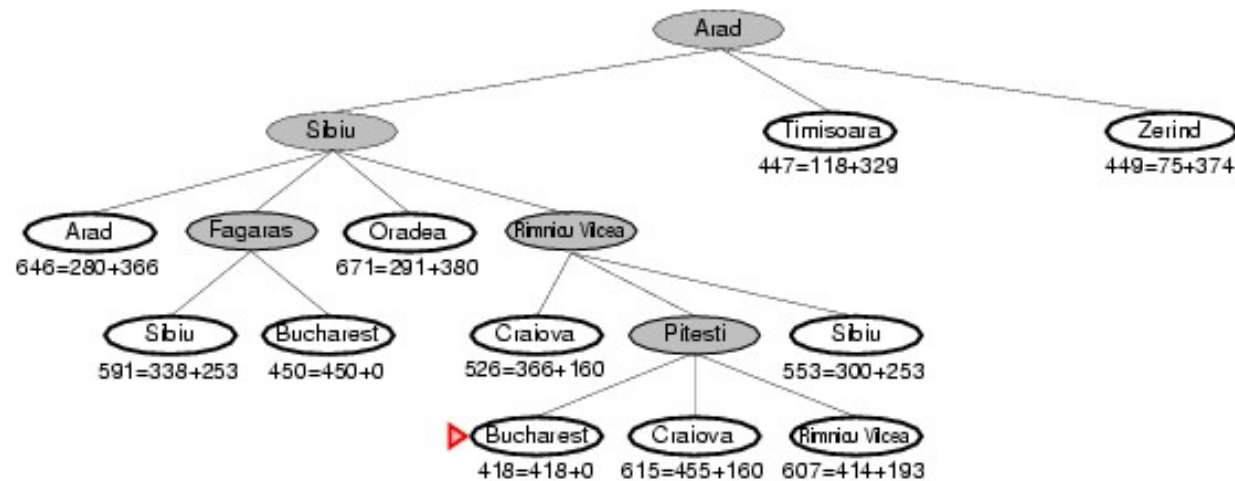
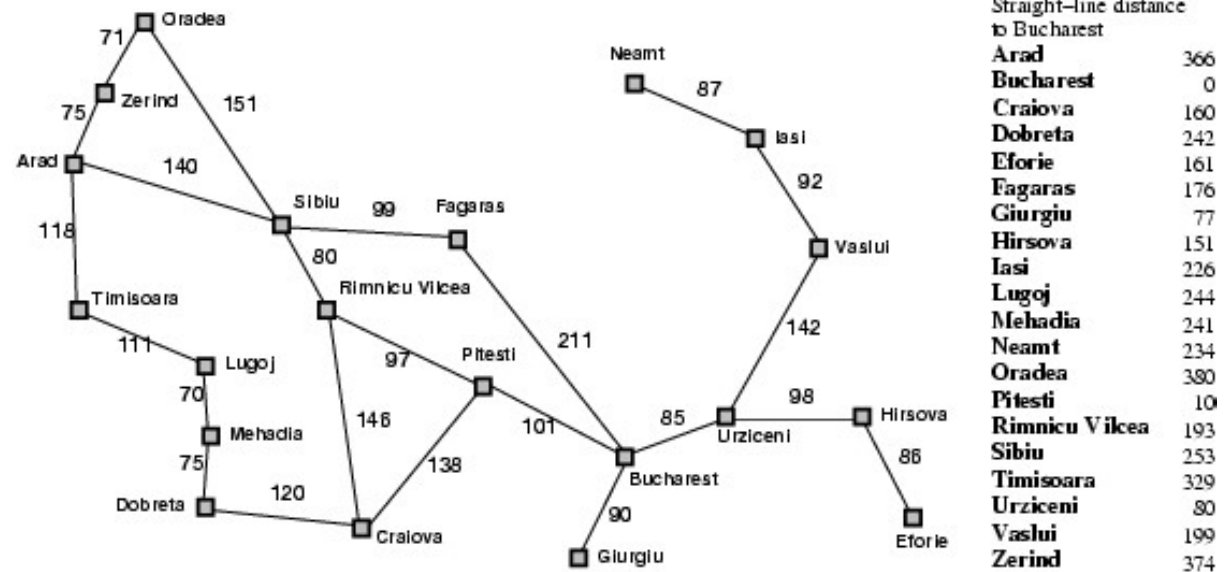
Da: S. Russell & P. Norvig: "Intelligenza Artificiale: un approccio moderno", Pearson ed.



A* esempio



A* esempio



Da: S. Russell & P. Norvig: "Intelligenza Artificiale: un approccio moderno", Pearson ed.



FUNZIONI EURISTICHE AMMISSIBILI

Poiché $h' \leq h^* \leq h$ la migliore è h^* .

- È meglio usare una funzione euristica con valori più alti, a patto che sia ottimista.

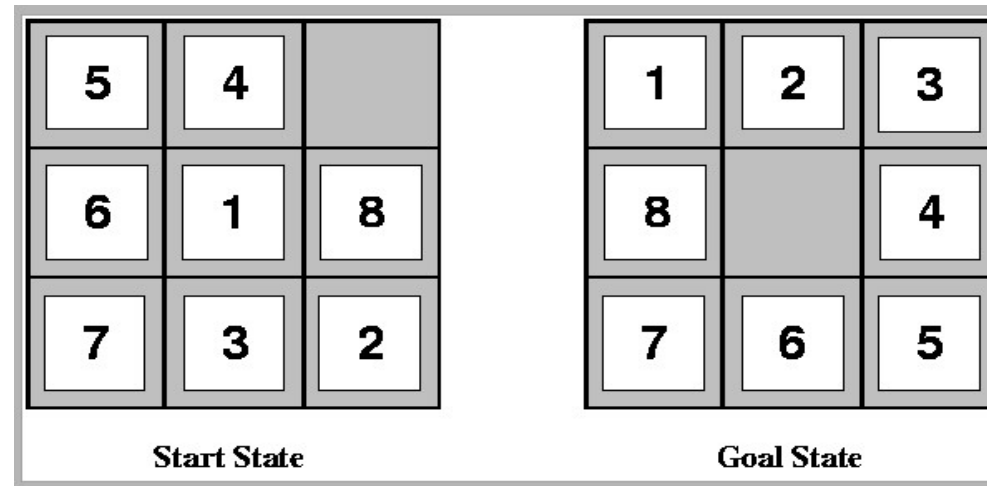
Come inventare funzioni euristiche?

- Spesso il costo di una soluzione esatta per un problema rilassato è una buona euristica per il problema originale.
- Se la definizione del problema è descritta in un linguaggio formale è possibile costruire problemi rilassati automaticamente.
- A volte si può utilizzare il massimo fra diverse euristiche.

$$h'(n) = \max(h_1(n), h_2(n), \dots, h_m(n)).$$



ESEMPIO FUNZIONI EURISTICHE AMMISSIBILI



E' possibile definire differenti funzioni euristiche. Ad esempio:

- h_1 = numero di tessere che sono fuori posto ($h_1 = 7$)
- h_2 = la somma delle distanze dalle posizioni che le tessere devono assumere nella configurazione obiettivo. La distanza è una somma delle distanze orizzontali e verticali (distanza di Manhattan). Le tessere da 1 a 8 nello stato iniziale danno una distanza $h_2 = 2+3+3+2+4+2+0+2 = 18$

Sono entrambe ammissibili



ESEMPIO: GIOCO DEL FILETTO

Descrizione: una tessera può muoversi dal quadrato A al quadrato B se A è adiacente a B e B è vuoto.

Problemi rilassati che rimuovono alcune condizioni:

- Una tessera può muoversi dal quadrato A al quadrato B se A è adiacente a B (distanza di Manhattan)
- Una tessera può muoversi dal quadrato A al quadrato B se B è vuoto.
- Una tessera può muoversi dal quadrato A al quadrato B (tesserine fuori posto).



DA ALBERI A GRAFI

Abbiamo assunto fin qui che lo spazio di ricerca sia un albero e non un grafo. **Non è quindi possibile raggiungere lo stesso stato da strade diverse**, ovvero se ci sono più strade si procede di nuovo all'espansione dello stato come se fosse nuovo.

- Non si tiene quindi traccia degli stati già considerati.

L'assunzione è ovviamente semplicistica: si pensi al gioco del filetto, ai missionari e cannibali ecc.

Come si estendono gli algoritmi precedenti per trattare con i grafi?

- Si deve tenere **memoria** non solo dei nodi da espandere (nodi aperti) ma anche dei nodi/stati già espansi (nodi chiusi)

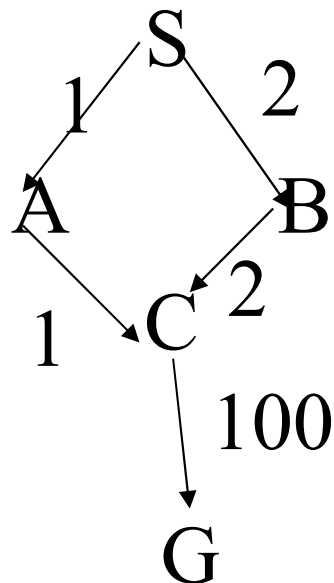


Stati ripetuti: Graph search

```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    if STATE[node] is not in closed then
      add STATE[node] to closed
      fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

Algoritmo A*: considerazioni sull'ottimalità

- Se in closed (lista dei nodi chiusi) conserviamo solo gli stati e la utilizziamo solo per evitare l'espansione ulteriore dei nodi che hanno stati in essa contenuti, siamo relativamente efficienti, ma potremmo non trovare la strada migliore.
- Viene a cadere l'assunzione che $g(n)$ valuta sempre la distanza minima dello stato/nodo dal nodo di partenza.
- Esempio: euristica h^* : $A=100$, $C=90$, $S=0$, $G=0$, $B=1$



S espansione Chiusi S Aperti B(3) A (101)
B espansione Chiusi B S Aperti A(101) C(94)
C espansione Chiusi B S C Aperti A(101) G(104)
A espansione Chiusi B S C A Aperti C eliminato
quindi G (104)
G goal ma no migliore strada



Algoritmo A* (considerazioni - consistenza)

- Nel caso precedente troveremmo la strada che passa per B (non è quella ottima)
- Il motivo è che a causa della stima troppo ottimista di B, C è espanso prima di trovare la strada alternativa che passa per A.

Ulteriore condizione su h: **consistenza** (o **monotonicità**)

Definizione: una euristica è consistente se per ogni nodo n , ogni successore n' di n generato da ogni azione a , vale:

- $h(n) = 0$ se lo stato corrispondente coincide con il goal, altrimenti
 - $h(n) \leq c(n,a,n') + h(n')$
-
- Con la monotonicità ogni nodo è raggiunto prima mediante la strada migliore. $g(n)$ valuta sempre la distanza minima dello stato/nodo dal nodo di partenza.

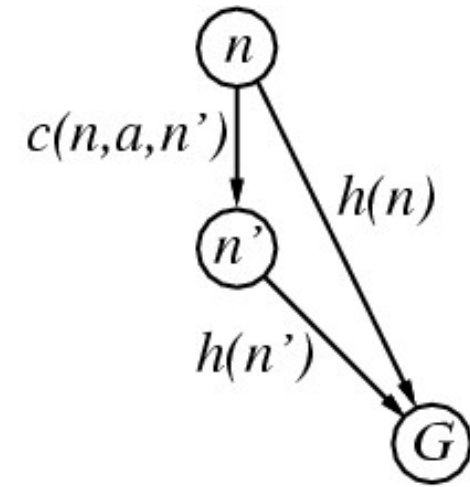


Euristica Consistente

Se h è consistente abbiamo che:

- $f(n') = g(n') + h(n')$
- $= g(n) + c(n, a, n') + h(n')$
- $\geq g(n) + h(n)$
- $= f(n)$

i.e., $f(n)$ non decresce mai lungo un cammino.



Theorema: Se $h(n)$ è consistente, A^* usando GRAPH-SEARCH è ottimale

- Nel caso in cui si utilizzi un'euristica ammissibile, ma non consistente per mantenere l'ottimalità l'algoritmo si complica inserendo in closed gli stati con il loro costo ($g(n)$) e ri-espandendo nuovi nodi n con stato già considerato (e in closed) se sono stati raggiunti da strade a minor costo $g(n)$.



ALGORITMI COSTRUTTIVI e ALGORITMI DI RICERCA LOCALE

Gli **algoritmi costruttivi** (visti fino ad ora) generano una soluzione ex-novo aggiungendo ad una situazione di partenza (vuota o iniziale) delle componenti in un particolare ordine fino a “costruire” la soluzione completa.

Gli **algoritmi di ricerca locale** partono da una soluzione iniziale e iterativamente cercano di rimpiazzare la soluzione corrente con una “migliore” in un intorno della soluzione corrente. In questi casi non interessa la “strada” per raggiungere l’obiettivo.

- Diversi problemi (ad esempio, il problema delle 8 regine) hanno la proprietà che la descrizione stessa dello stato contiene tutte le informazioni necessarie per la soluzione (il cammino è irrilevante).
- In questo caso gli algoritmi con miglioramenti iterativi spesso forniscono l’approccio più pratico.
- Ad esempio, possiamo cominciare con tutte le regine sulla scacchiera e poi spostarle per ridurre il numero di attacchi.



RICERCA LOCALE

- La ricerca locale è basata sull'esplorazione iterativa di “soluzioni vicine”, che possono migliorare la soluzione corrente mediante modifiche locali.
- Struttura dei “vicini” (**neighborhood**). Una struttura dei “vicini” è una funzione F che assegna a ogni soluzione s dell'insieme di soluzioni S un insieme di soluzioni $N(s)$ sottoinsieme di S .
- La scelta della funzione F è fondamentale per l'efficienza del sistema e definisce l'insieme delle soluzioni che possono essere raggiunte da s in un singolo passo di ricerca dell'algoritmo.
- Tipicamente è definita implicitamente attraverso le possibili mosse.
- La soluzione trovata da un algoritmo di ricerca locale non è detto sia ottima globalmente, ma può essere ottima rispetto ai cambiamenti locali.



RICERCA LOCALE (continua)

Massimo (minimo) locale:

- Un massimo locale è una soluzione s tale che per qualunque s' appartenente a $N(s)$, data una funzione di valutazione f , $f(s) \geq f(s')$.
- Quando risolviamo un problema di massimizzazione (minimizzazione) cerchiamo un massimo (minimo) globale s_{opt} cioè tale che per qualunque s , $f(s_{\text{opt}}) > f(s)$.
- Più è largo il neighborhood più è probabile che un massimo locale sia anche un massimo globale (ma ovviamente esiste un problema di complessità computazionale). Quindi, più è largo un neighborhood più aumenta la qualità della soluzione

L'algoritmo base a cui ci si riferisce è detto “**iterative improvement**” (miglioramento iterativo):

- Tipicamente, tale algoritmo parte con una soluzione di tentativo generata in modo casuale o mediante un procedimento costruttivo e cerca di migliorare la soluzione corrente muovendosi verso i vicini. Se fra i vicini c'è una soluzione migliore, tale soluzione va a rimpiazzare la corrente. Altrimenti termina in un massimo (minimo) locale.



Hill-climbing

"Like climbing Everest in thick fog with amnesia"

- Si noti che l'algoritmo non tiene traccia dell'albero di ricerca.

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                  neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
```


ALGORITMO HILL CLIMBING

- Un modo semplice per capire questo algoritmo è di immaginare che tutti gli stati facciano parte di una superficie di un territorio. L'altezza dei punti corrisponde alla loro funzione di valutazione.
- Dobbiamo muoverci cercando le sommità più alte tenendo conto solo dello stato corrente e dei vicini immediati.
- Hill climbing si muove verso i valori più alti (o bassi) che corrispondono a massimi (o minimi) locali

Problema: l'algoritmo può incappare in un minimo (massimo) locale di scarsa qualità:

- Ha il limite di essere un metodo locale che non valuta tutta la situazione, ma solo gli stati abbastanza vicini a quello corrente.

L'algoritmo poi non gestisce un albero di ricerca, ma solo lo stato e la sua valutazione.



PROBLEMI

Massimi locali:

- Stati migliori di tutti i vicini, ma peggiori di altri stati che non sono nelle vicinanze. Il metodo ci spinge verso il massimo locale, peggiorando la situazione.

Pianori o Altopiani:

- Zone piatte dello stato di ricerca in cui gli stati vicini hanno tutti lo stesso valore. In quale direzione muoversi (scelta casuale)?

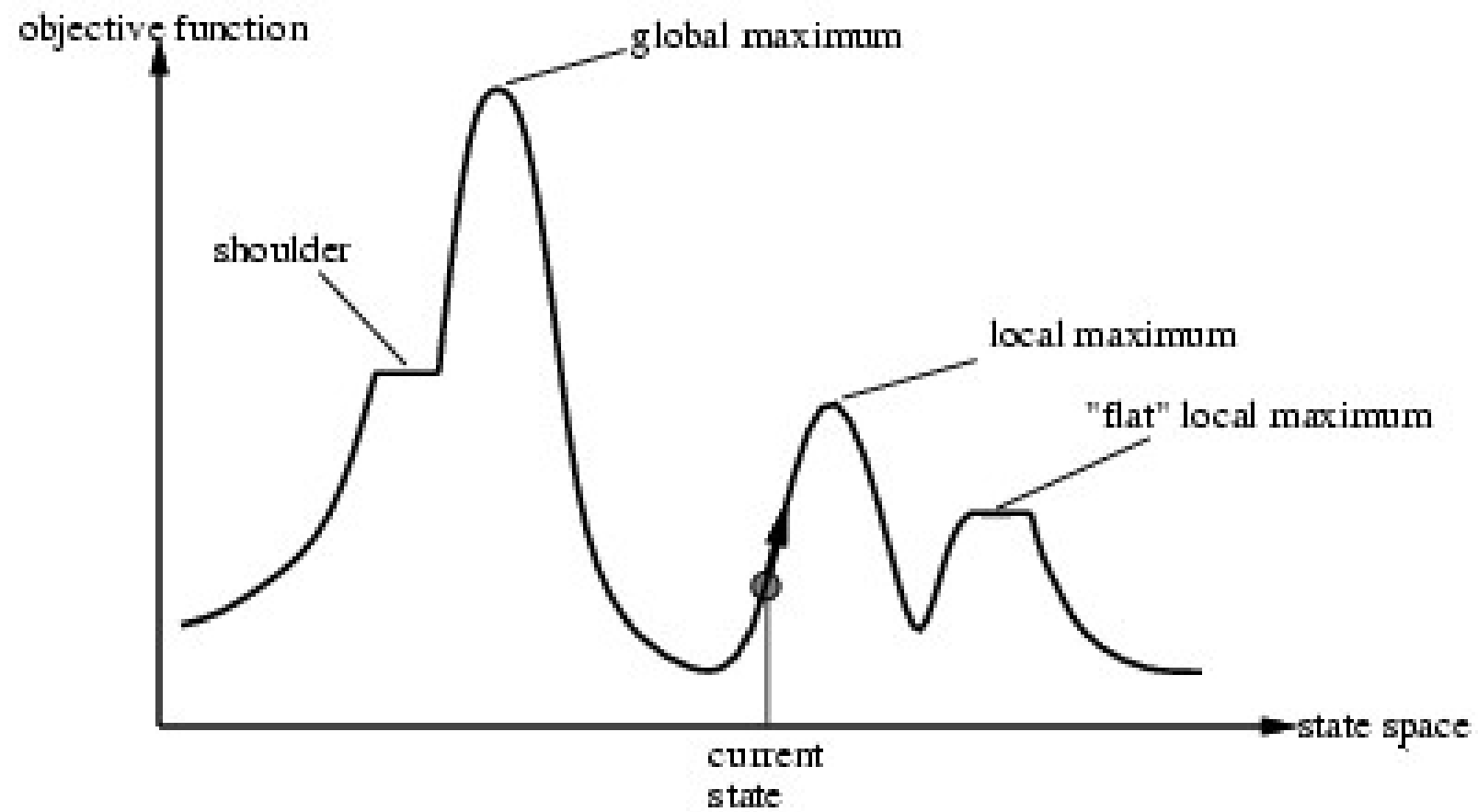
Crinali:

- È una zona più alta di quelle adiacenti verso cui dovremmo andare, ma non possiamo andarci in modo diretto. Dobbiamo quindi muoverci in un'altra direzione per raggiungerlo.

Una possibile (semplice) soluzione: ripartire con una nuova ricerca da una soluzione di partenza random o generata in modo costruttivo. Si salva poi la soluzione migliore dopo una serie di tentativi (bound dovuto al tempo di CPU o numero di iterazioni).

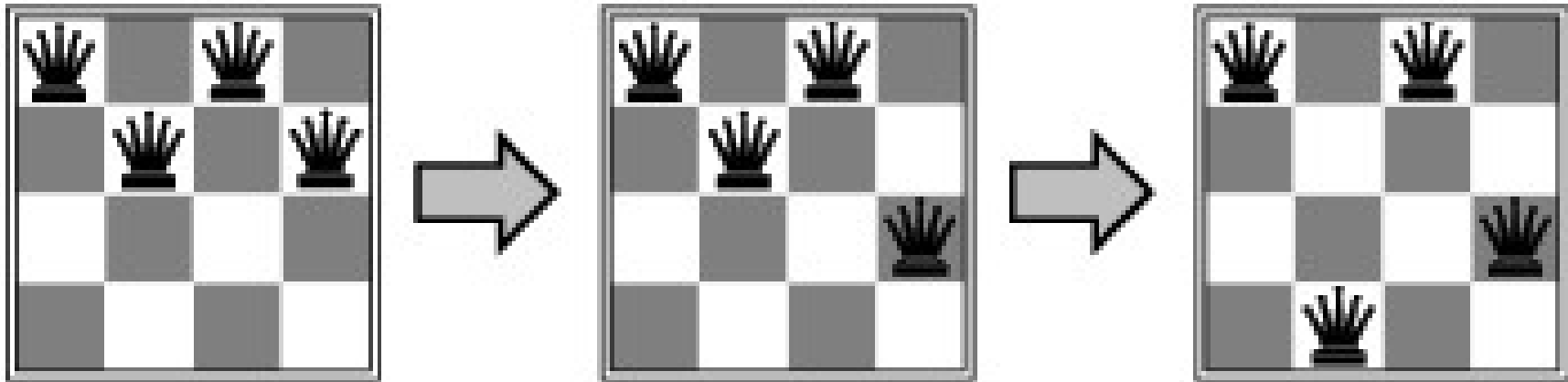


Hill-climbing



Esempio: N- regine

Inserire n regine in una scacchiera in modo che non si attacchino.



META EURISTICHE

- Si definiscono **meta-euristiche** l'insieme di algoritmi, tecniche e studi relativi all'applicazione di criteri euristici per risolvere problemi di ottimizzazione (e quindi di migliorare la ricerca locale con criteri abbastanza generali).

Ne citiamo alcuni:

- **ANT colony optimization**
 - ispirata al comportamento di colonie di insetti.
 - Tali insetti mostrano capacità globali nel trovare, ad esempio, il cammino migliore per arrivare al cibo dal formicaio (algoritmi di ricerca cooperativi)
- **Tabu search**
 - La sua caratteristica principale è l'utilizzo di una memoria per guidare il processo di ricerca in modo da evitare la ripetizione di stati già esplorati.



META EURISTICHE (continua)

- **Algoritmi genetici**

- Sono algoritmi evolutivi ispirati ai modelli dell'evoluzione delle specie in natura. Utilizzano il principio della selezione naturale che favorisce gli individui di una popolazione che sono più adatti ad uno specifico ambiente per sopravvivere e riprodursi.
- Ogni individuo rappresenta una soluzione con il corrispondente valore della funzione di valutazione (fitness). I tre principali operatori sono: selezione, mutazione e ricombinazione.



META EURISTICHE (Continua)

- **Simulated Annealing**

- Cerca di evitare il problema dei massimi locali, seguendo in pratica la strategia in salita, ma ogni tanto fa un passo che non porta a un incremento in salita.
- Quando rimaniamo bloccati in un massimo locale, invece di cominciare di nuovo casualmente potremmo permettere alla ricerca di fare alcuni passi in discesa per evitare un massimo locale.
- Ispirato al processo di raffreddamento dei metalli.
- In pratica, ci suggerisce di esaminare, ogni tanto, nella ricerca un nodo anche se sembra lontano dalla soluzione.



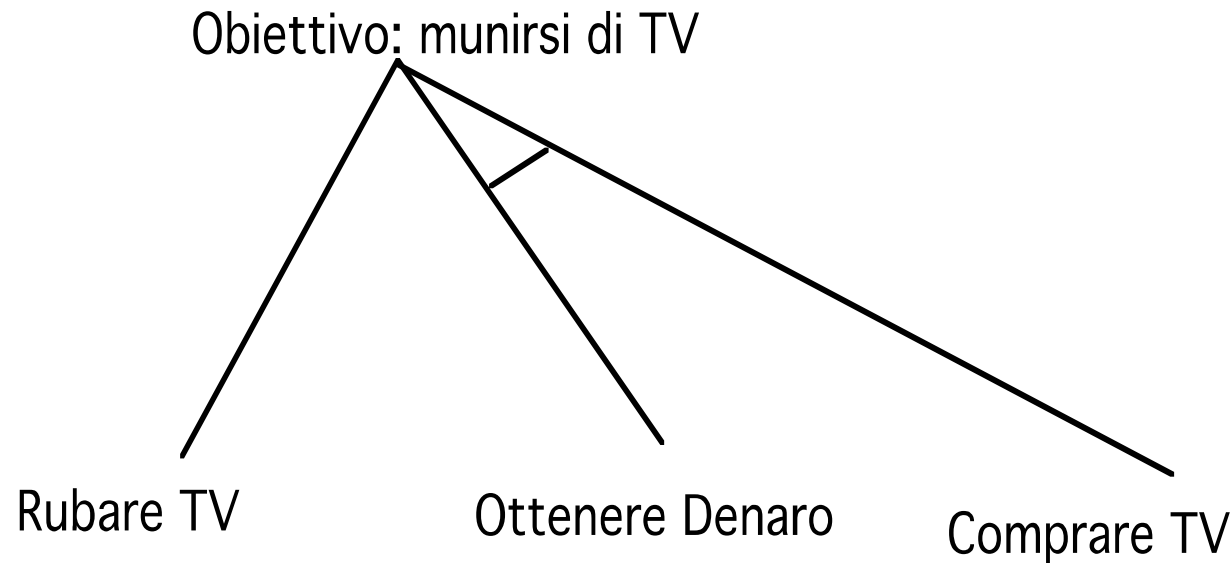
Ricerca in alberi/grafi AND-OR

Fino a questo momento abbiamo discusso strategie di ricerca per grafi OR nei quali vogliamo trovare un singolo cammino verso la soluzione.

- Il grafo (o albero) AND/OR risulta appropriato quando si vogliono rappresentare problemi che si possono risolvere scomponendoli in un insieme di problemi più piccoli che andranno poi tutti risolti.
- --> arco AND che deve puntare a un qualunque numero di nodi successori che si devono tutti risolvere per risolvere il nodo AND stesso.
- Dal nodo AND possono anche partire rami OR che indicano soluzioni alternative.



ESEMPIO

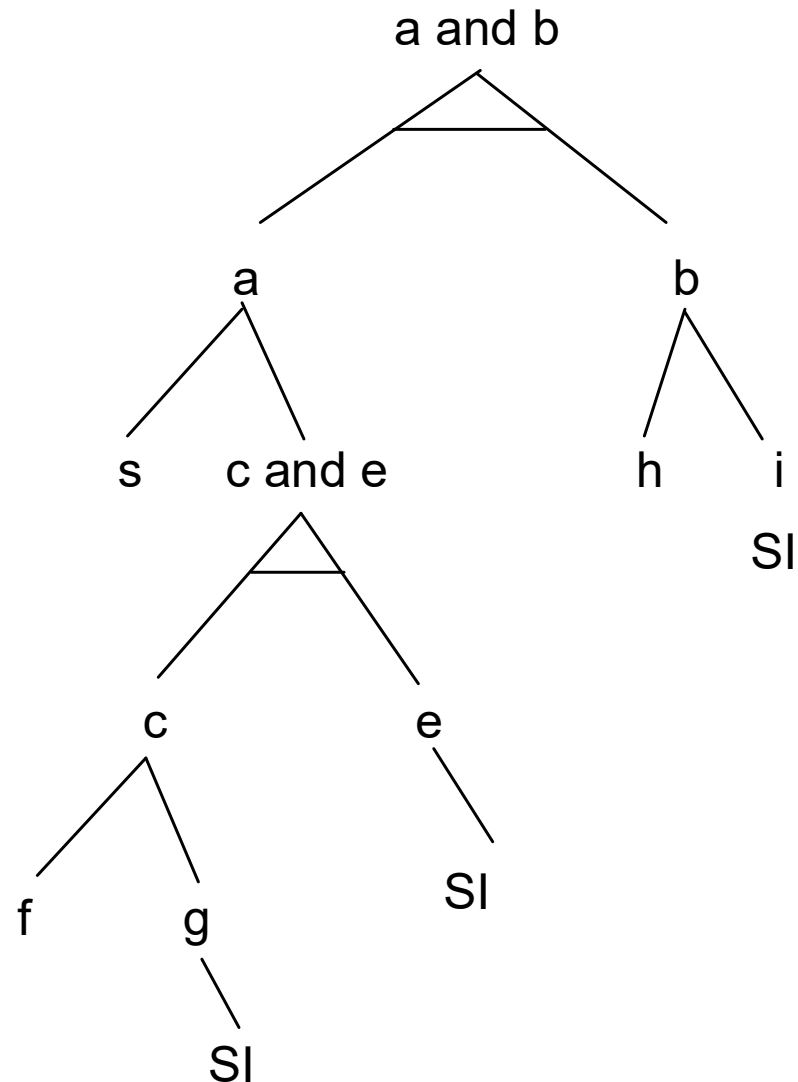


- L'algoritmo di ricerca va opportunamente modificato per essere in grado di trattare gli archi AND.
- Non faremo questa parte.
- Possiamo però pensare di trasformare gli alberi AND/OR in alberi OR e applicare le strategie già viste.



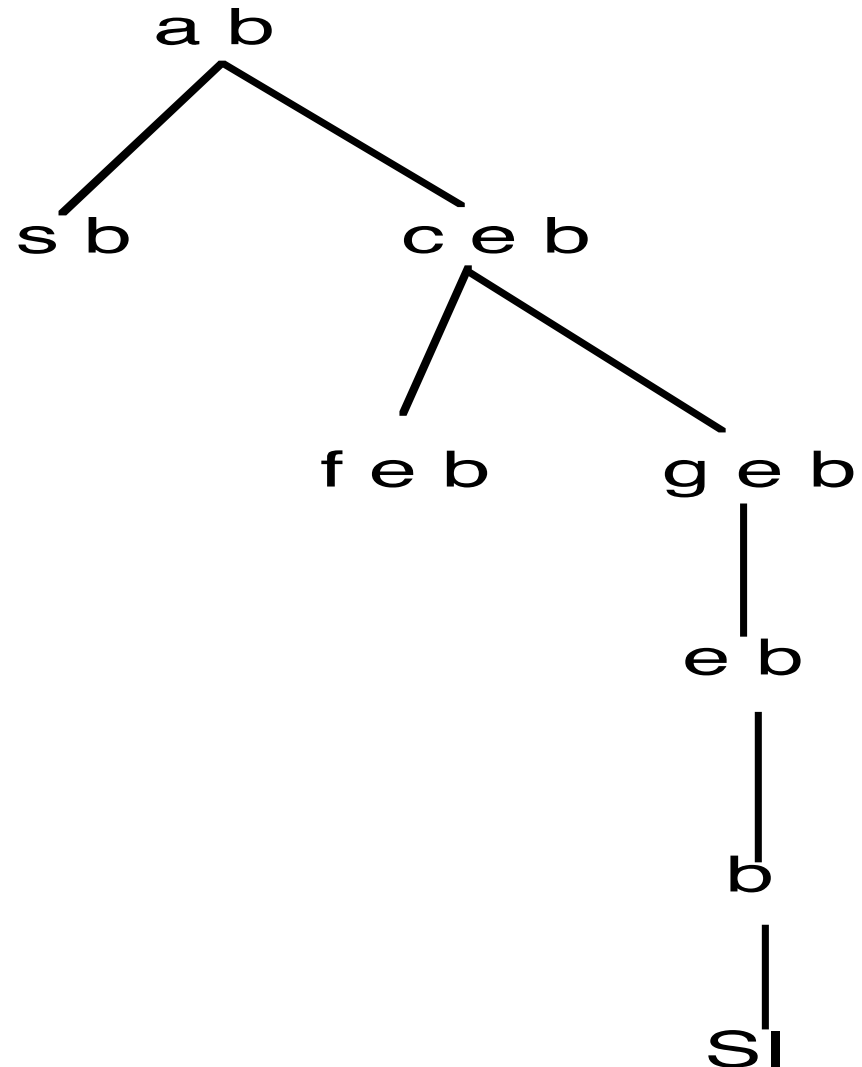
ESEMPIO: ALBERO AND/OR in un sistema logico

- g.
- e.
- i.
- $s \rightarrow a.$
- $c \text{ and } e \rightarrow a.$
- $f \rightarrow c.$
- $g \rightarrow c.$
- $h \rightarrow b.$
- $i \rightarrow b.$
- Goal: a and b



TRASFORMAZIONE IN ALBERO DI RICERCA OR:

- g.
- e.
- i.
- s \rightarrow a.
- c and e \rightarrow a.
- f \rightarrow c.
- g \rightarrow c.
- h \rightarrow b.
- i \rightarrow b.
- Goal: a and b



TRASFORMAZIONE DI ALBERI AND/OR IN ALBERI DI RICERCA:

Nota:

- Non ha alcuna influenza, al fine del raggiungimento della soluzione, l'ordine di espansione dei goal in AND **se il sistema è monotono** come in questo caso (sistema logico)
- Altrimenti, se i sottogoals interagiscono, non vale questa proprietà e si devono considerare come alternative i diversi ordini di espansione (planning).



ALTRE TECNICHE EURISTICHE

- SCELTA IN BASE AGLI OPERATORI
 - (regole, task inseriti in un'agenda):
- A) Insita nel controllo e influenzata IMPLICITAMENTE dall'utente:
 - Prolog: ordine testuale delle regole
 - OPS: strategie in base alla forma dell'antecedente (LEX MEA)
- B) Influenzata ESPLICITAMENTE dall'utente:
 - Attribuire alle regole dei valori di priorità rappresentati come numeri interi; in questo modo l'interprete selezionerà sempre la regola applicabile a più alta priorità.



ESEMPIO

R1: IF <la temperatura è maggiore di 100 gradi>,
 AND <la pressione ha valore p1>
 THEN <attiva la valvola>
 AND <segnala una situazione di allarme all'operatore>
 --- priorità 10

R2: IF <la pressione ha valore p1>
 THEN <attiva la procedura di funzionamento normale PROC5>
 --- priorità 5.

PRIORITÀ DINAMICHE:

- a) lo stato della memoria di lavoro;
- b) la presenza di particolari regole applicabili;
- c) la precedente esecuzione di regole correlate;
- d) l'andamento dell'esecuzione per particolari scelte fatte precedentemente.



META-CONOSCENZA=CONOSCENZA SULLA CONOSCENZA

PROBLEMI:

1. risulta particolarmente complesso individuare il corretto valore di priorità;
2. l'attribuzione dei valori di priorità è oscura: diventa dunque carente nel sistema la capacità di spiegazione.
3. la modificabilità della base di conoscenza diminuisce: controllo e conoscenza sul dominio non separati; le regole presentano dipendenze implicite.

META-CONOSCENZA=CONOSCENZA SULLA CONOSCENZA

- Può risolvere i problemi precedentemente citati (vasta applicazione non solo per il controllo).

meta-regola:

- MR1 "Usa regole che esprimono situazioni di allarme prima di regole che esprimono situazioni di funzionamento normale".



META-CONOSCENZA=CONOSCENZA SULLA CONOSCENZA

ESEMPI: Sistemi Esperti: Molgen, Neomycin, Teiresias.

VANTAGGI:

- il sistema è maggiormente flessibile e modificabile; un cambiamento nella strategia di controllo implica semplicemente il cambiamento di alcune meta-regole.
- la strategia di controllo è semplice da capire e descrivere.
- potenti meccanismi di spiegazione del proprio comportamento.



TEREISIAS: META CONOSCENZA PER INFLUENZARE IL CONTROLLO

Tale meta-conoscenza ha tre scopi fondamentali:

- a) sono utilizzate meta-regole per definire strategie di controllo;
- b) sono utilizzati "modelli" per descrivere regole di livello oggetto;
- c) sono utilizzati metodi descrittivi espliciti per descrivere la rappresentazione utilizzata e dunque poterla modificare.

CONTROLLO:

- Meta-Regola 1

IF <ci sono regole che menzionano.....>

AND <ci sono regole che menzionano.....>.

THEN <le prime dovrebbero essere eseguite prima delle seconde>

(FC=0.4).

Meta-Regola 2

IF <il paziente è.....>

AND <ci sono regole che menzionano ...>

THEN <è certo che nessuna di esse sarà utile ulteriormente> (FC=1).



TEREISIAS: META CONOSCENZA PER INFLUENZARE IL CONTROLLO

- meta-regola 1: ordine parziale sulle regole di livello oggetto;
- meta-regola 2: utilità di certe regole di livello oggetto;

NOTA: La sintassi delle meta-regole è identica a quella di livello oggetto

NUOVE FUNZIONI:

- MENTIONS: serve per referenziare le regole di livello oggetto;
- UTILITY e BEFORE: servono per influenzare la strategia di controllo.



TEREISIAS: META CONOSCENZA PER INFLUENZARE IL CONTROLLO

FUNZIONAMENTO:

- Trova le regole di livello oggetto applicabili (L);
- Trova le meta-regole associate (L');
- Valuta ogni meta-regola di L' e determina se è applicabile per alcune regole di L;
- Ordina o toglie elementi da L in accordo ai criteri delle meta-regole;
- Esegue ogni regola rimasta in L nell'ordine ottenuto.



META-CONOSCENZA PER IL CONTROLLO UNA NUOVA ARCHITETTURA

Così si espande anche l'interprete:

