

Ottimalità dell'algoritmo A^*

- Si dimostra che quando:

- 1) l' euristica h è ammissibile
- 2) e tutti i passi hanno un costo maggiore di una costante positiva piccola a piacere

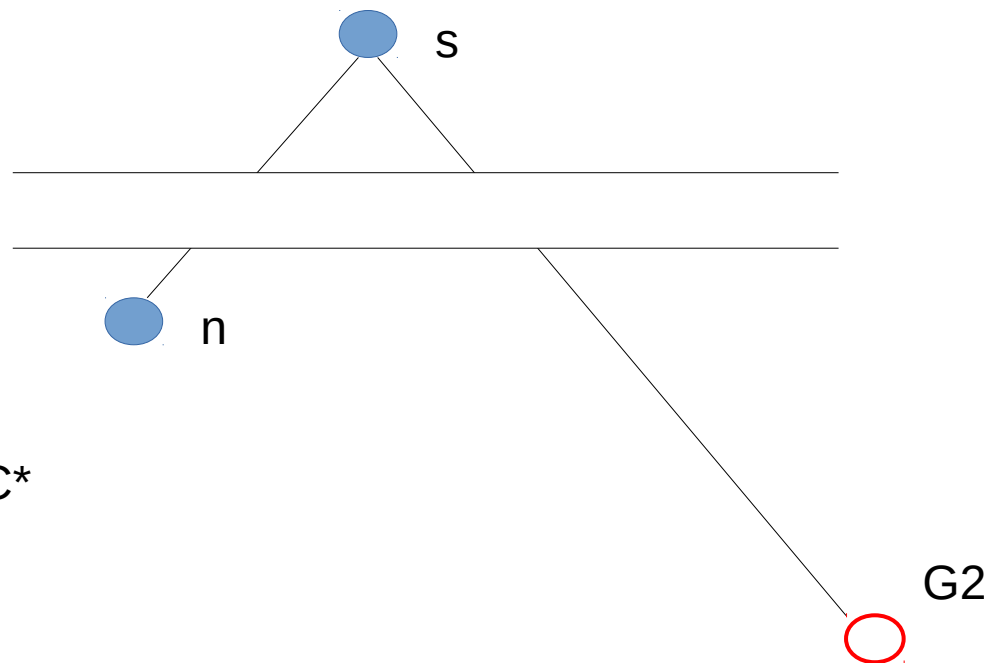
- Allora:

- A^* termina e trova una soluzione ottima (di costo minimo)
- In altri termini in questo caso A^* è completo e ottimale

A* ottimale per alberi di ricerca

- **Nota:** in un albero di ricerca ogni nodo ha un solo cammino assoluto
- Perché manchi l' ottimalità deve accadere che durante la ricerca l' algoritmo:
 - scelga un nodo obiettivo sub-ottimo (G2)
 - al posto di un nodo (n)
 - che si trova su un cammino ottimo (che porta al goal G)
- Dimostriamo che in un albero di ricerca ciò non può capitare laddove l' euristica è ammissibile e i passi hanno costo non nullo

A* ottimale per alberi di ricerca

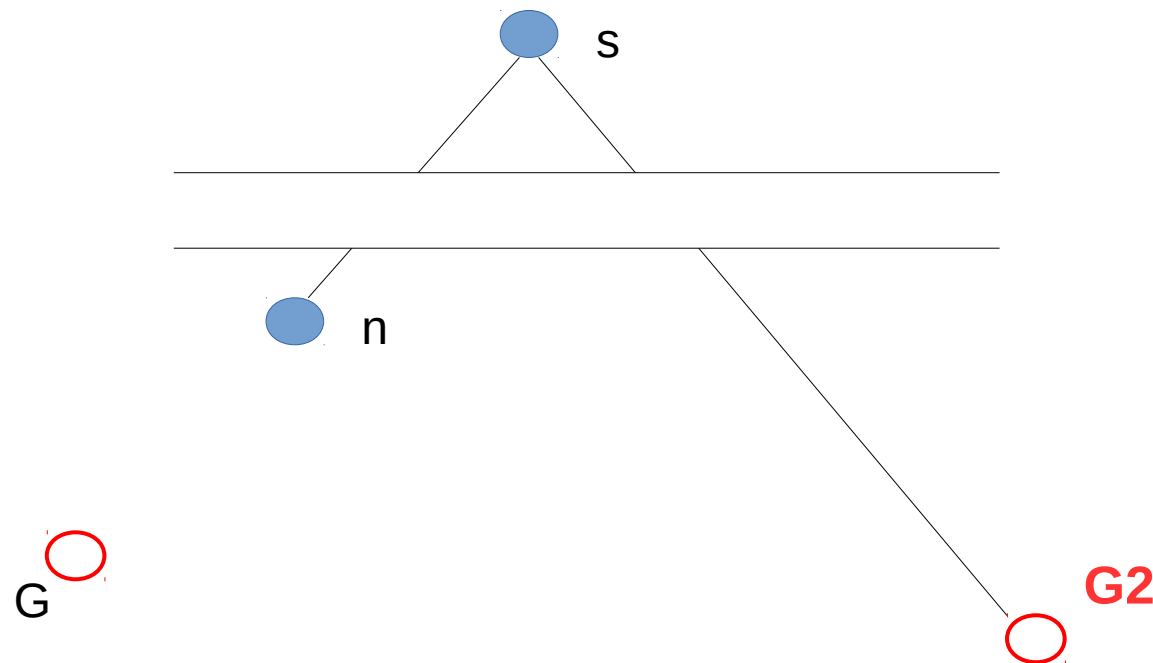


$$f^*(G) = g^*(G) + h^*(G) = C^*$$

G

- Chiamiamo:
 - **G2** = obiettivo subottimo
 - **C*** il costo reale della soluzione ottima

A* ottimale per alberi di ricerca



Consideriamo G2:

$$f(G2) = g(G2) + h(G2) > C^*$$

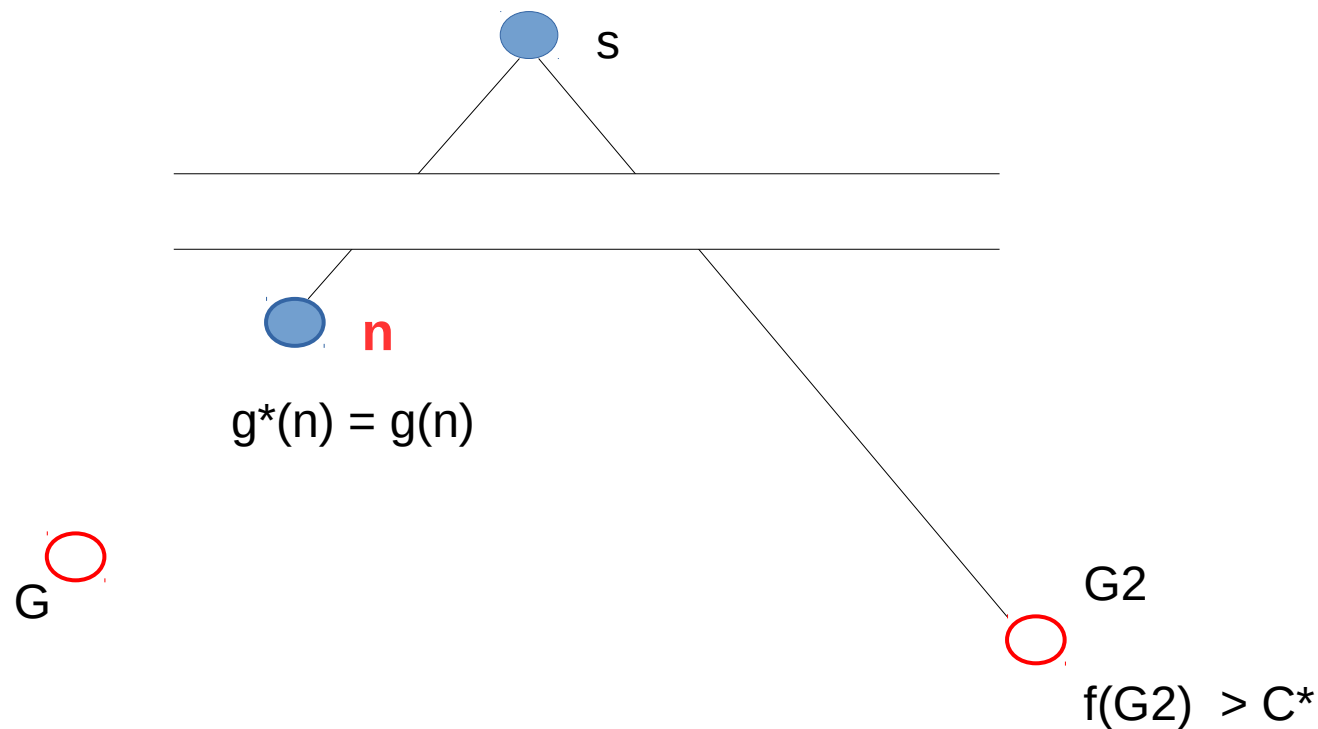
1) $h(G2) = 0$ perché G2 è un nodo obiettivo

2) $f(G2) = g(G2) + h(G2) = g(G2) + 0 = g(G2)$

3) Poiché per assunto G2 è subottimo, si ha che:

$$f(G2) = g(G2) > C^*$$

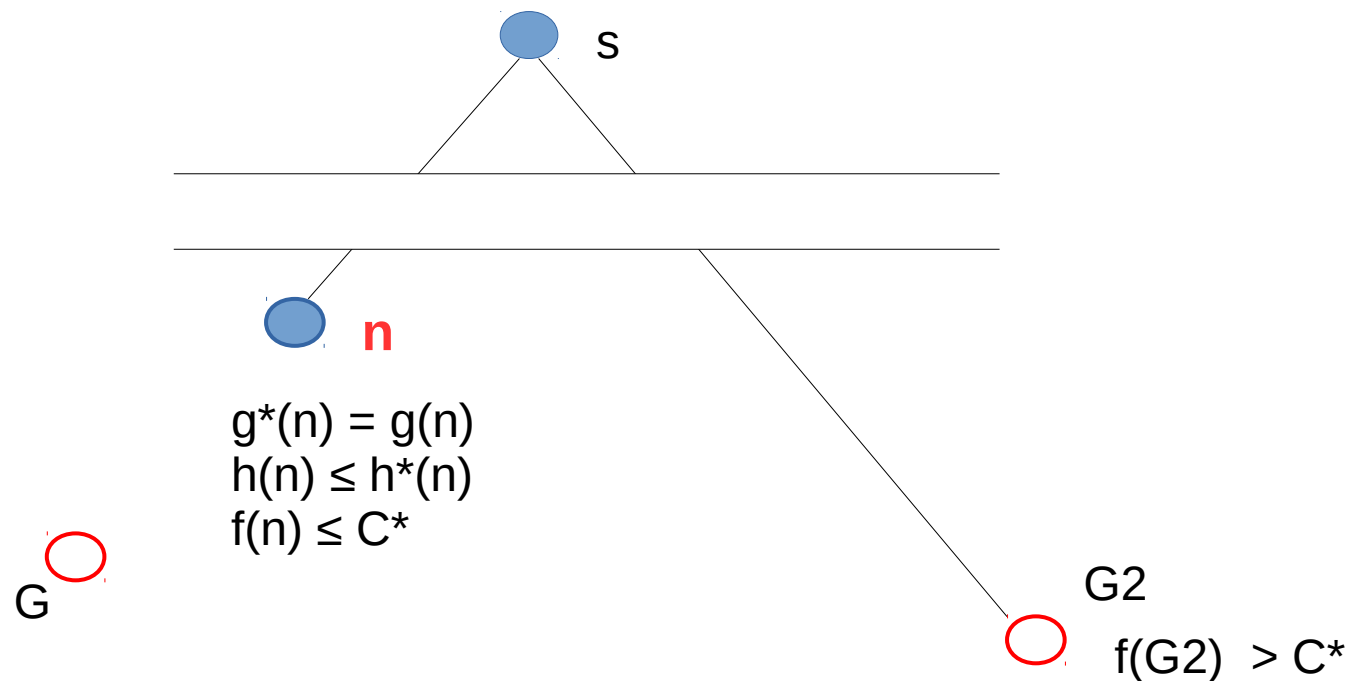
A* ottimale per alberi di ricerca



Sia n un **generico nodo** appartenente a un **cammino ottimo**:

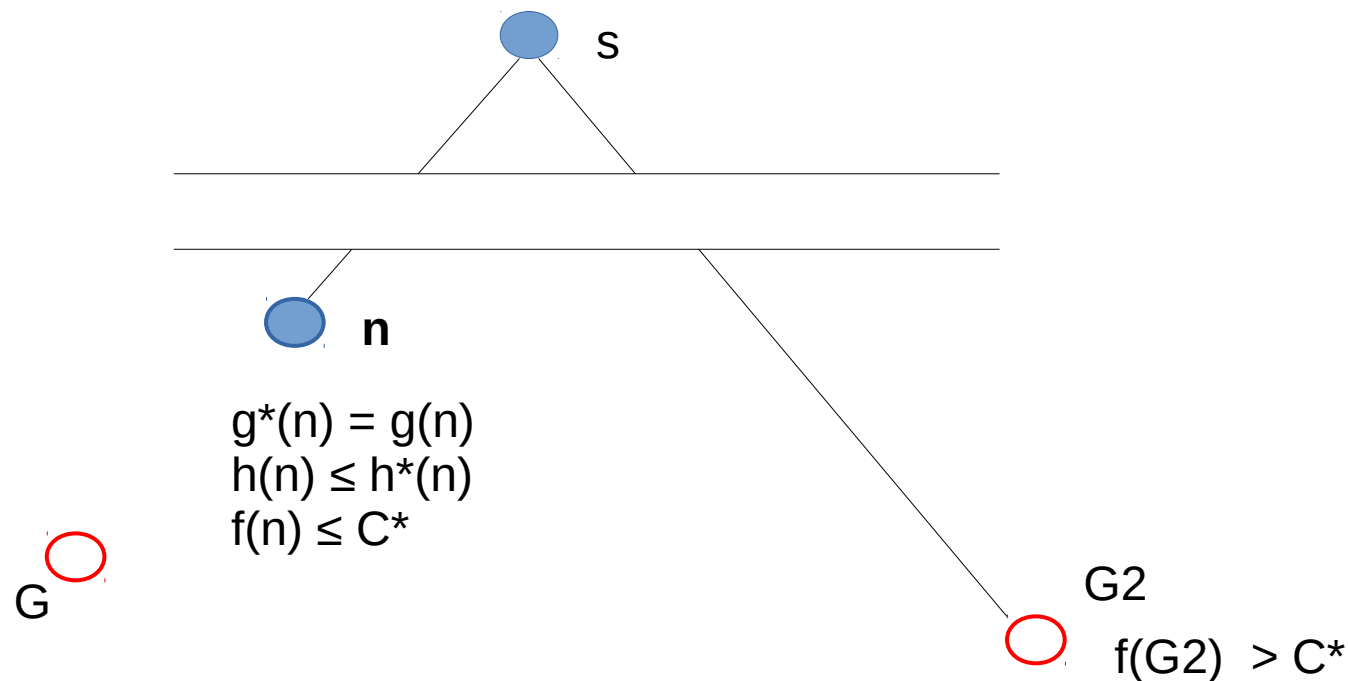
- $g^*(n) = g(n)$ perché lavoriamo su di un albero e ogni nodo è raggiungibile dalla radice lungo un solo percorso

A* ottimale per alberi di ricerca



- h è ammissibile per ipotesi, quindi $h(n) \leq h^*(n)$
- Quindi: $f(n) = g(n) + h(n) \leq g^*(n) + h^*(n) = C^*$
- Quindi $f(n) \leq C^* < f(G2)$

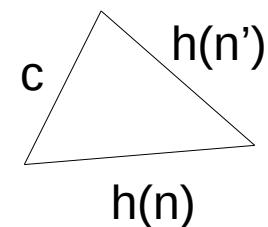
A* ottimale per alberi di ricerca



- Quindi:
 - 1) $f(n) < f(G2)$
 - 2) A* sceglie il nodo aperto con $f(.)$ minima,
- Di conseguenza fra **n** e **G2** verrà scelto **n** (q.e.d.)

A* ottimale per grafi di ricerca

- **Nei grafi vi è molteplicità di cammino:**
il primo cammino trovato durante la ricerca che porta a un certo stato non è necessariamente quello ottimo (nota: nell' esempio della Romania A* non si ferma quando incontra Bucarest la prima volta)
- **Questa proprietà invalida la precedente dimostrazione!**
- La dimostrazione generale è piuttosto complessa, occorre aggiungere l' ipotesi che **h** sia **monotona** (o consistente), cioè che dati un qualsiasi nodo n e un qualsiasi suo successore n' prodotto eseguendo l' azione a in n vale che $h(n) \leq c(n, a, n') + h(n')$
- Tale disuguaglianza è una disuguaglianza triangolare



A* ottimale per grafi di ricerca

- Si dimostra che quando l'euristica è monotona, per la disuguaglianza triangolare i costi $f(n)$ lungo un cammino sono non decrescenti
- A* espande i nodi in ordine non decrescente di f :
 - se un nodo CHIUSO viene incontrato più volte lungo un percorso, i nuovi valori di f saranno superiori a quello calcolato la prima volta
(cfr. Arad nell'esempio della Romania)
 - Di conseguenza sono i primi incontri con i nodi obiettivo che permettono di individuare una soluzione ottima
- Su questo assunto si dimostra l'ottimalità

A* ottimale per grafi di ricerca

- Dimostriamo che se l'euristica è monotona, i costi di $f(n)$ lungo un cammino sono non decrescenti:

1) Sia n' un nodo successore di n , per definizione:

$$g(n') = g(n) + c(n, a, n')$$

2) Sempre per definizione:

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

3) E sostituendo $g(n')$:

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

A* ottimale per grafi di ricerca

4) Applichiamo ora la disuguaglianza triangolare:

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

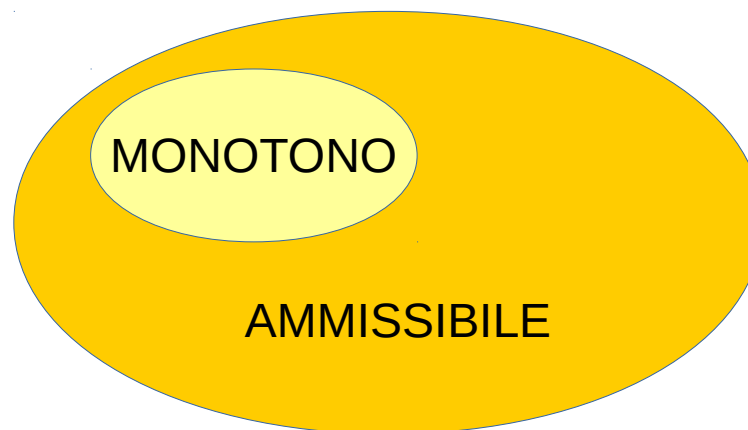
5) Sappiamo però che, per definizione: $g(n) + h(n) = f(n)$

6) Di conseguenza: $f(n') \geq f(n)$ (q.e.d.)

Euristiche e impatto sulla ricerca

Euristiche monotone e ammissibili

- La **monotonicità** è una proprietà più stringente dell' **ammissibilità**
 - Si dimostra che un' **euristica monotona** è anche **ammissibile**
 - Spesso ma non sempre le euristiche ammissibili sono anche monotone



Esempio di euristica monotona

- La **distanza in linea d'aria** è un' euristica:
 - **ammissibile** e **monotona** per il problema della Romania, infatti data una città (genericamente indicata da luogo) e un suo possibile successore (indicato da luogo1) avremo sempre che:
 - $h(\text{luogo}) < c(\text{luogo}, \text{vai}, \text{luogo1}) + h(\text{luogo1})$
- Cioè la distanza in linea d'aria da “luogo” a Bucharest è minore della distanza via terra fra “luogo” e il confinante “luogo1” più la distanza in linea d'aria da “luogo1” a Bucharest

Ammissibile non vuol dire informativo !!

- $h(n) = 0$ è un' euristica sempre ammissibile ma non è informativa della desiderabilità degli stati
 - Permette di valutare solo il costo del percorso fatto per raggiungere un nodo
 - La ricerca **diventa cieca** e richiede l' **espansione di un maggiore numero di nodi** rispetto a usare un' euristica ammissibile e informativa
- In particolare se abbiamo inoltre che tutte le operazioni che permettono di passare da un nodo a un successore hanno costo uniforme pari a 1, A^* diventa una ricerca in ampiezza

- **A* è ottimamente efficiente per qualsiasi euristica:**
non esiste alcun altro algoritmo ottimo che garantisca
di espandere meno nodi di quelli espansi da A*
- Purtroppo il numero di nodi espansi aumenta esponenzialmente con la profondità della soluzione ottima
- A* mantiene in memoria tutti i nodi generati

Ridurre i requisiti di memoria di A^*

- IDA*: unisce A^* e iterative deepening (non lo studiamo)
- RBFS: recursive best-first search:
 - Simile alla *ricerca ricorsiva in profondità* con una differenza importante: usa un “**upper bound**” dinamico che consente di focalizzare la ricerca **sul percorso più promettente** invece di continuare indefinitamente lungo lo stesso percorso.
 - Questo upper bound (limite superiore) ricorda la migliore alternativa fra i percorsi attualmente aperti

Recursive Best-First Strategy (RBFS)

- **Difetto:**
lo stesso nodo può essere visitato più volte, se l' algoritmo, dopo aver cambiato percorso, ritorna al percorso precedentemente abbandonato
- **Pregio:**
poche esigenze di spazio, come la *ricerca in profondità senza backtracking*: mantiene solo i nodi del percorso di ricerca corrente e i loro fratelli. È un vantaggio rispetto ad A* che mantiene informazione su tutti i nodi aperti e chiusi

Recursive best first strategy (RBFS)

- RBFS utilizza:
 - Un nodo
 - Un limite superiore (**upper bound**) locale del nodo
- Esplora:
 - il sottoalbero del nodo finché nella sua frontiera ci sono nodi i cui costi non eccedono il limite superiore
- Upper bound di un nodo:
 - $\min(\text{upper_bound_parent}, \text{valore_fratello_di_costo_minimo})$
 - memorizza la stima del costo del percorso alternativo migliore,

Dall'articolo di Korf 1993

- L'algoritmo ha 3 argomenti:
 - Un nodo **N**,
 - Un valore **F(N)** ad esso associato,
 - un upper bound **B**
- **f(n)**: la **funzione di valutazione**, è statica cioè il valore restituito non cambia nel tempo
- **F(n)**: valore associato al nodo n, è dinamico cioè cambia nel tempo e dipende dai discendenti di N:
 - $F(N) = f(N)$ se N è esplorato per la prima volta
 - $\min(F(S_N))$ con S_N sottoalbero di N altrimenti

Linear-space best-first search, Richard E. Korf, Artificial Intelligence 62 (1993) 41-78 Elsevier

Dall'articolo di Korf 1993

- L'algoritmo ha 3 argomenti:
 - Un nodo **N**,
 - Un valore **F(N)** ad esso associato,
 - un upper bound **B**
- **f(n)**: la **funzione di valutazione**, è statica cioè il valore restituito non cambia nel tempo
- **F(n)**: valore associato al nodo n, è dinamico
- **B**: è calcolato basandosi sui valori F(.) dei nodi fratelli, ricorda il secondo migliore

Linear-space best-first search, Richard E. Korf, Artificial Intelligence 62 (1993) 41-78 Elsevier

Dall'articolo di Korf 1993

- L'algoritmo ha 3 argomenti:
 - Un nodo **N**,
 - Un valore **F(N)** ad esso associato,
 - un upper bound **B**
- **La chiamata iniziale** a RBFS è:

RBFS (r, f(r), ∞)

dove r è il nodo radice r, il valore F(r) è pari a quello statico f(r) e l'upper bound è pari a ∞

- **Esempio: RBFS(Arad, 366, ∞)**

Linear-space best-first search, Richard E. Korf, Artificial Intelligence 62 (1993) 41-78 Elsevier

- RBFS lavora come A^* fintantoché la soluzione che costruisce rispetta l' upper bound (è la migliore)
- Sospende la ricerca lungo un cammino quando questo non appare più il migliore
- Il cammino viene dimenticato (nodi cancellati)
- Viene solo conservata traccia nella sua radice del costo che si era stimato esplorando quella via

Algoritmo RBFS (Korf 1993)

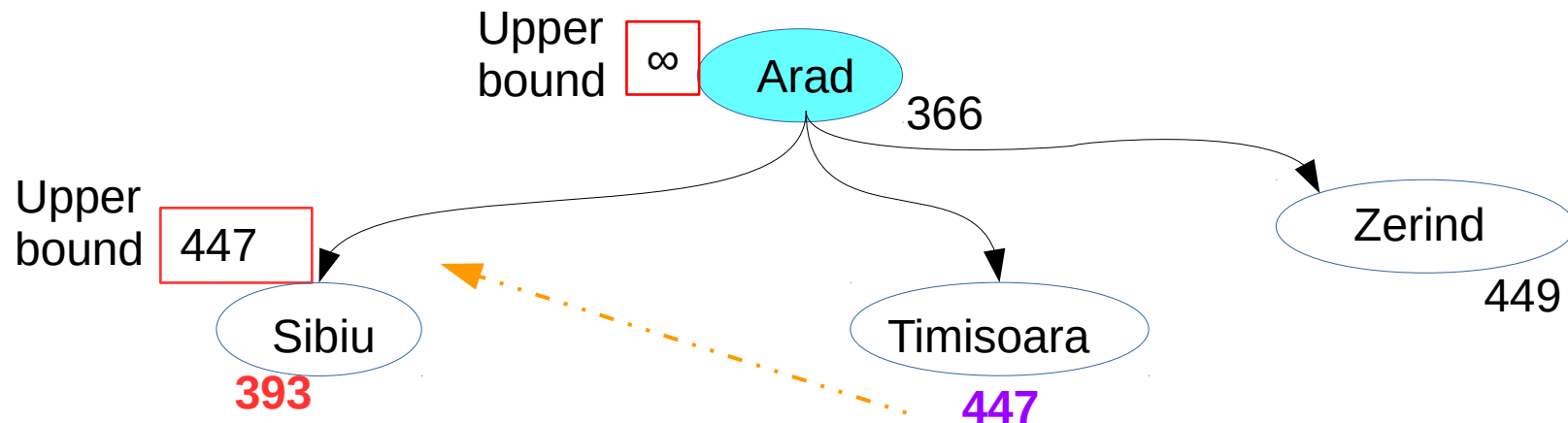
RBFS (node: N, value: F(N), bound: B)

```
IF f(N)>B, return f(N) // la fz. val. supera il limite superiore, cambia percorso
IF N is a goal, EXIT algorithm // successo!
IF N has no children, RETURN infinity // vicolo cieco, cambia percorso
FOR each child Ni of N, // inizializza F(.) per i successori di N
    IF f(N)<F(N), F[i] := MAX(F(N),f(Ni)) // N è il nodo padre, su cui siamo focalizzati
    ELSE F[i] := f(Ni)
sort Ni and F[i] in increasing order of F[i] // ordinamento per F crescenti, dopo
// F[1] sarà l'F del figlio più promettente

IF only one child, F[2] := infinity
WHILE (F[1] <= B and F[1] < infinity) // discesa ricorsiva solo se upper bound rispettato
    F[1] := RBFS(N1, F[1], MIN(B, F[2])) // applico ricorsivamente RBFS
    insert N1 and F[1] in sorted order
return F[1]
```

NOTA: F[i] è il valore F del nodo in posizione i dopo l'ordinamento. La posizione 1 è quella del nodo più promettente, la 2 quella dell'alternativa migliore e così via.

Esempio 1/5



Viene scelta Sibiu perché è il nodo sul percorso ritenuto più conveniente Viene associato a tale nodo il valore 447, che è la stima di costo della sua alternativa più conveniente

$F[\text{Arad}] = f(\text{Arad}) = 366$
 $\text{Upper bound}[\text{Arad}] = \infty$

For each child N_i of N :

$f(N) < F(N)?$

$366 < 366?$ No quindi $F[i] := f(N_i)$

esempio $F[\text{Sibiu}] := f(\text{Sibiu})$ cioè 393

Sort basato su F :

Sibiu (393), Timisoara (447), Zerind (449),
quindi $F[1]$ è $F[\text{Sibiu}]$

CICLO WHILE:

$F[\text{Sibiu}] < \text{Upper bound}[\text{Arad}]$? Sì!

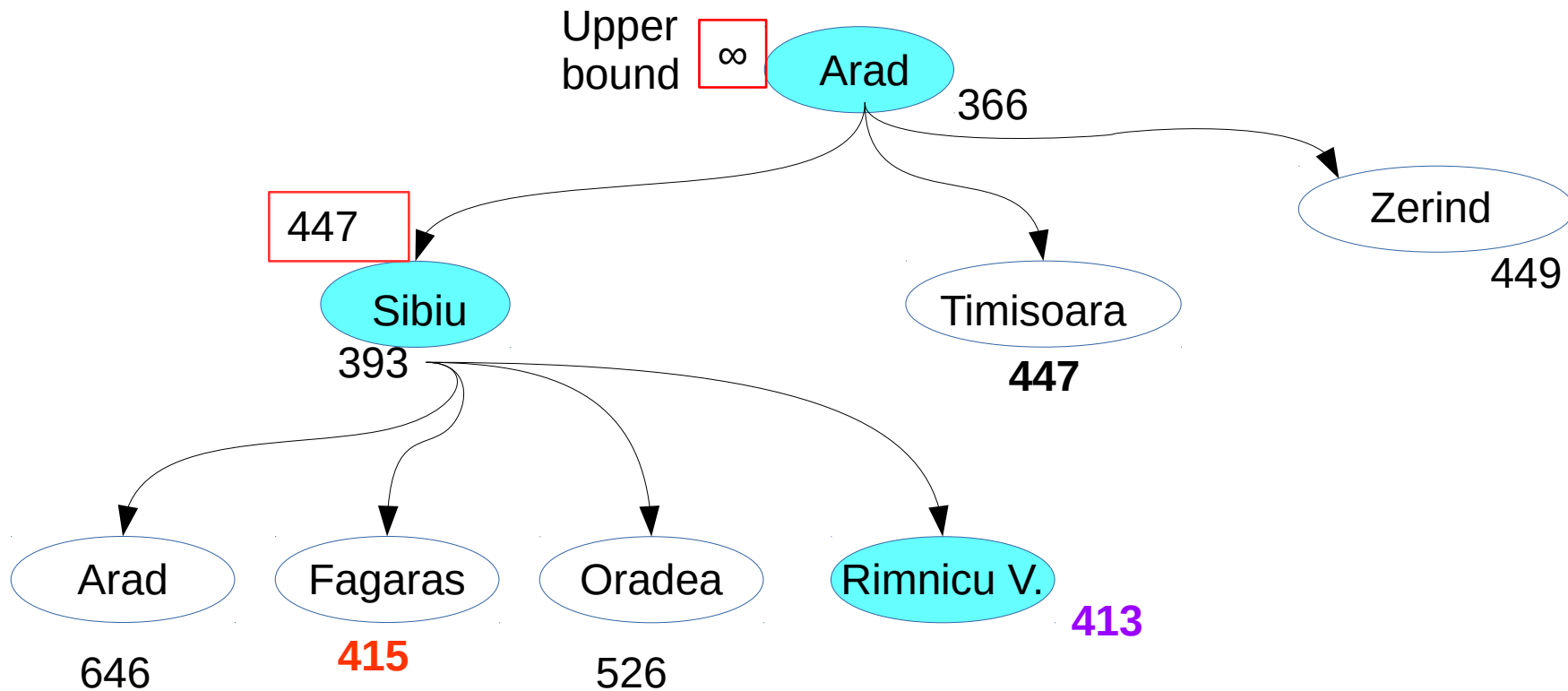
$F[\text{Sibiu}] < \text{infinity}$? Sì!

$F[1] := \text{RBFS}(N1, F[1], \text{MIN}(B, F[2]))$

Dove $B = \infty$, 2 corrisponde a Timisoara e $F[2] = 447$:

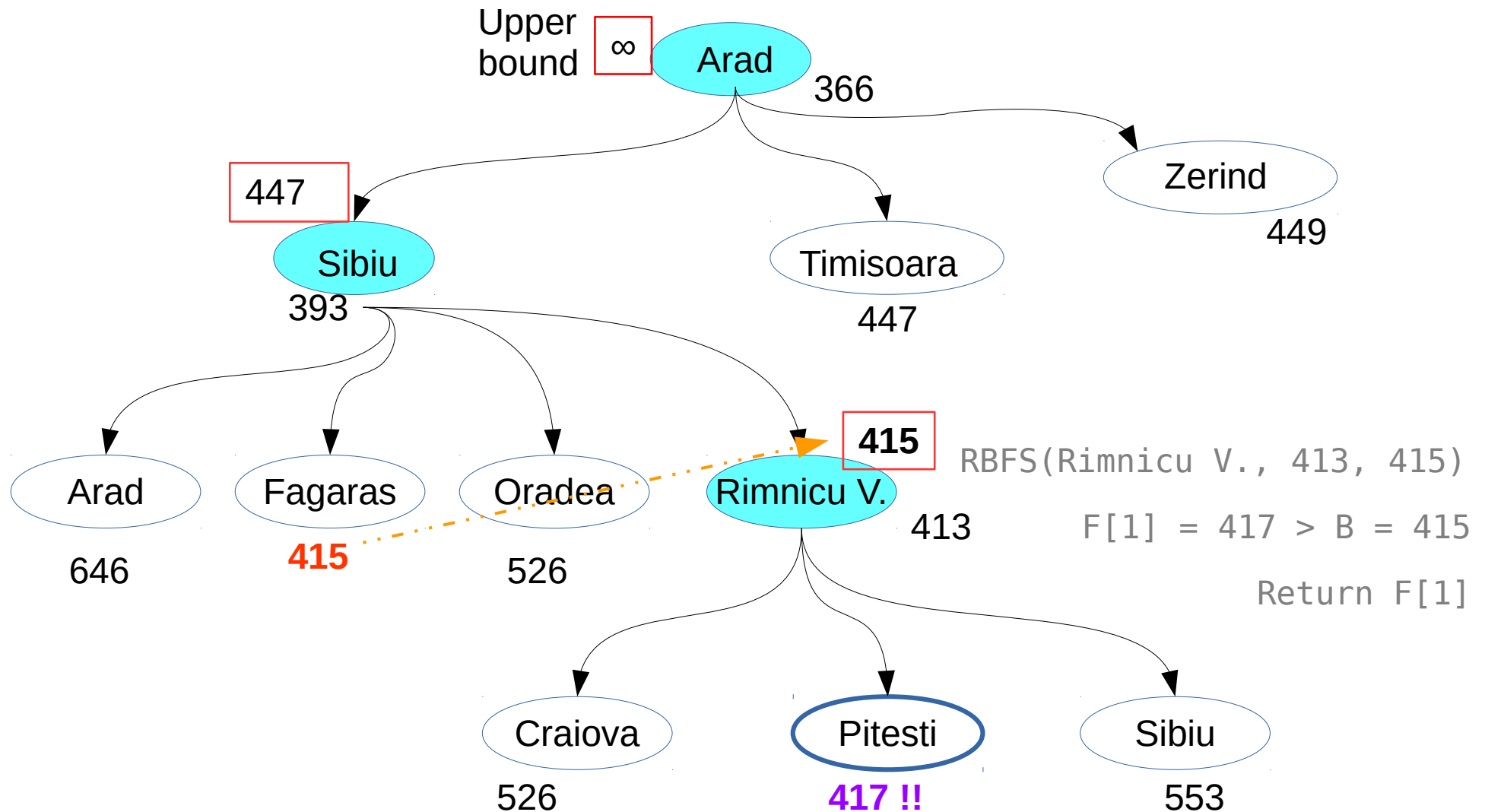
$F[1] := \text{RBFS}(\text{Sibiu}, 393, 447)$

Esempio 2/5



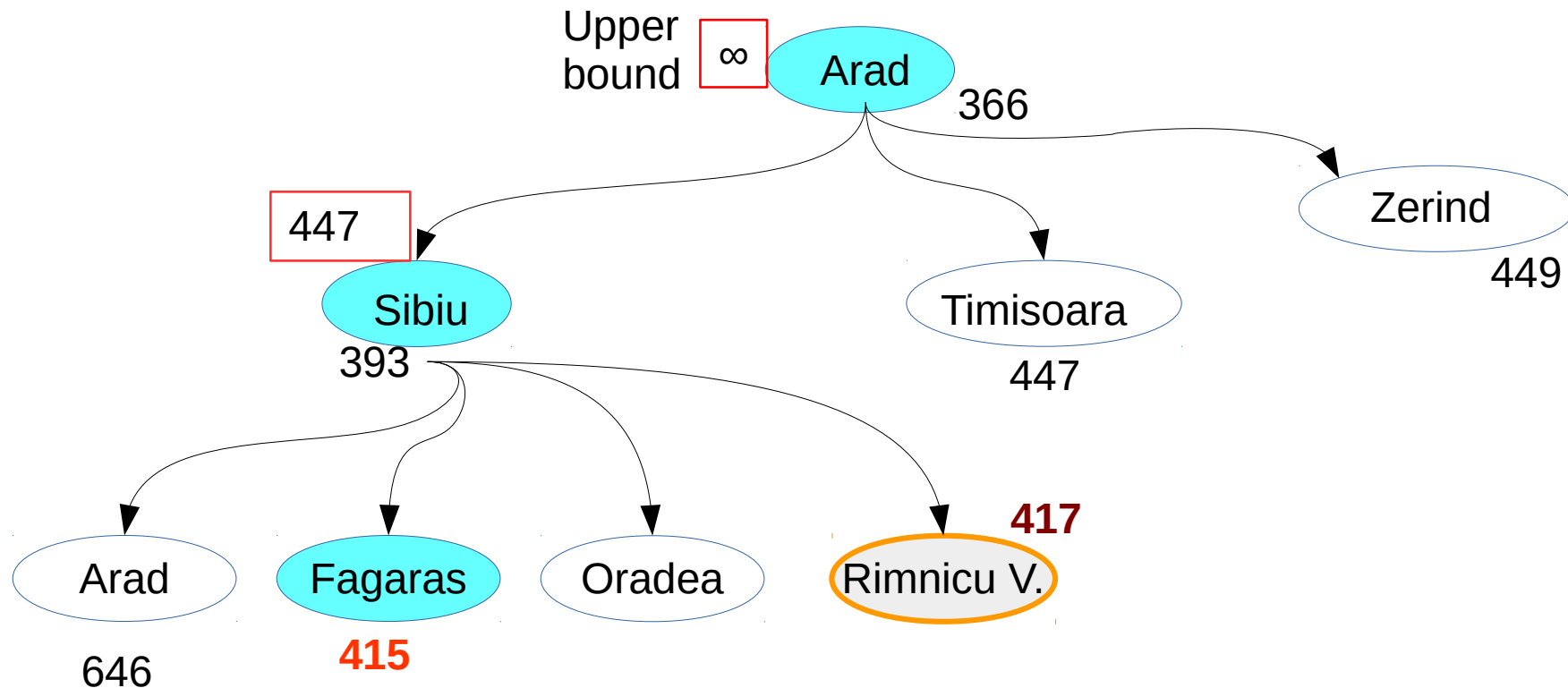
Rimnicu Vilcea è il nodo più promettente.
Fagaras è il secondo migliore, quindi 415 sarà l'upper bound per il richiamo ricorsivo su Rimnicu V.

Esempio 2/5



Pitesti ha una stima di costo più elevata dell'alternativa più conveniente a Rimnicu Vilcea, la condizione del while fallisce: si cambia percorso

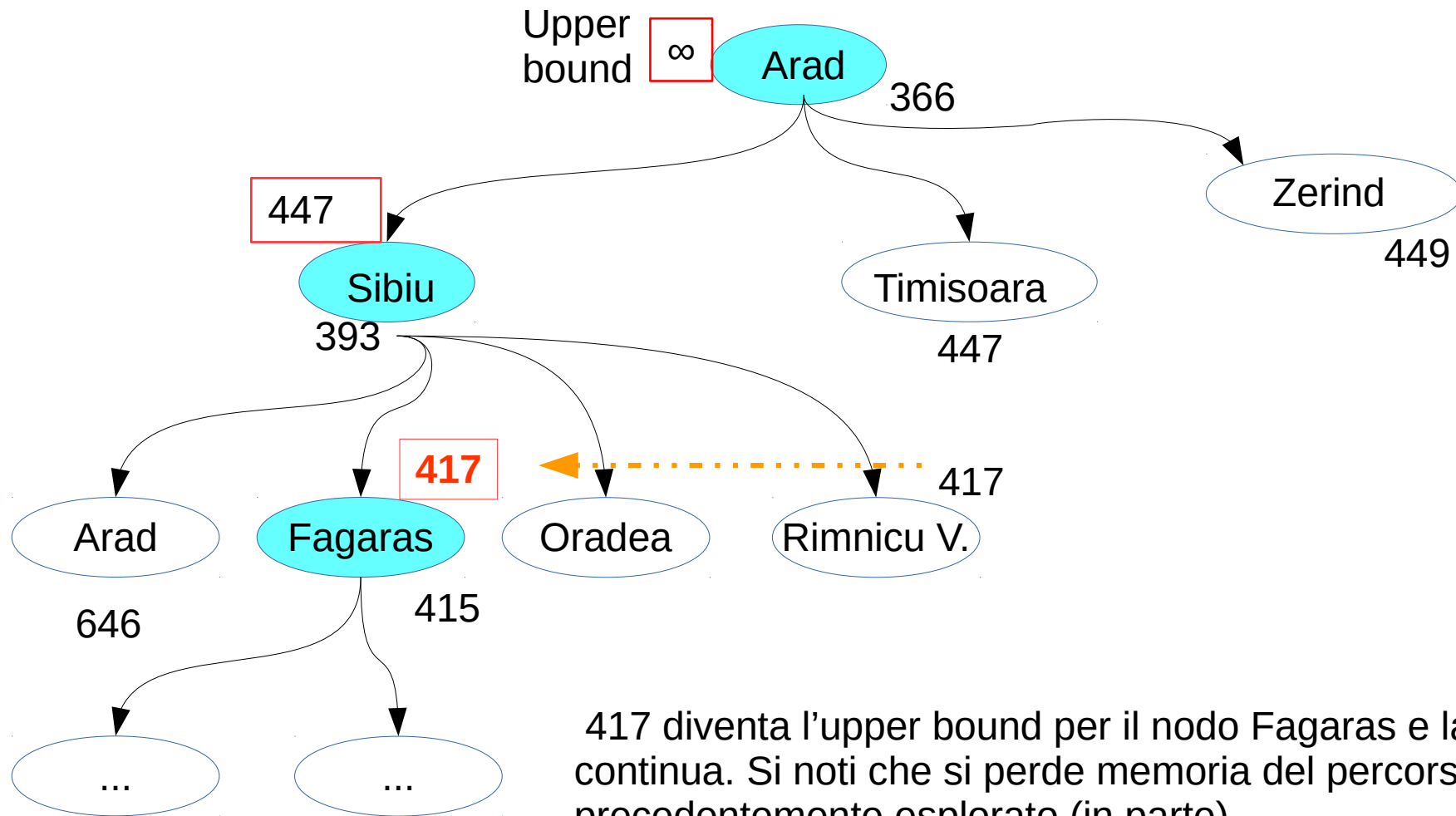
Esempio 3/5



La valutazione di Rimnicu Vilcea viene aggiornato alla valutazione del suo discendente più promettente (Pitesti). I nodi figli di Sibiu sono riordinati di conseguenza. Ora il nodo più promettente è Fagaras

Il sottoalbero di Rimnicu Vilcea è stato rimosso

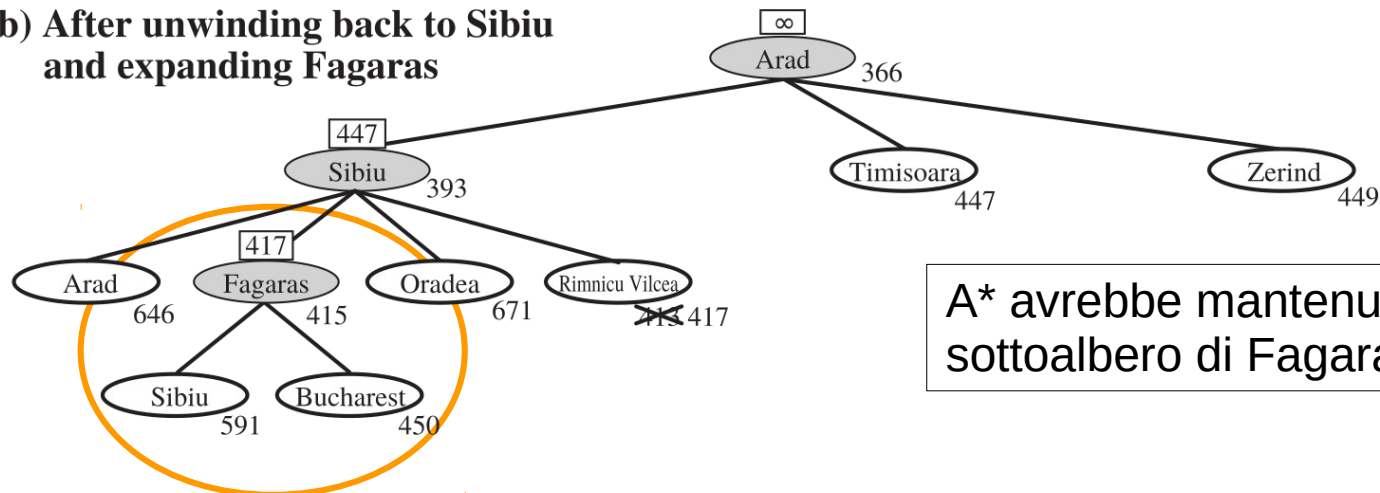
Esempio 4/5



417 diventa l'upper bound per il nodo Fagaras e la ricerca continua. Si noti che si perde memoria del percorso precedentemente esplorato (in parte)

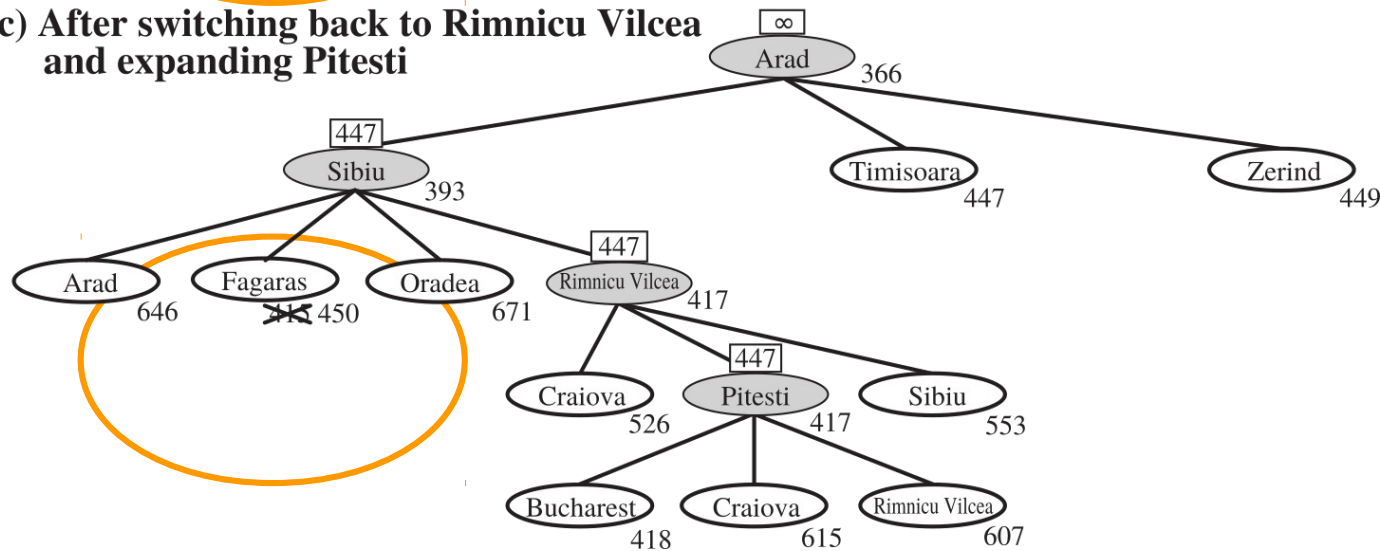
Esempio 5/5

(b) After unwinding back to Sibiu and expanding Fagaras



A* avrebbe mantenuto anche i nodi del sottoalbero di Fagaras

(c) After switching back to Rimnicu Vilcea and expanding Pitesti



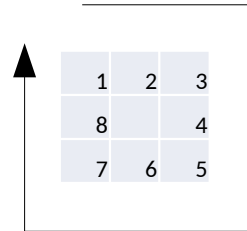
- RBFS è **ottimo** se l' euristica è **ammissibile**
- **Complessità spaziale lineare:** $O(bd)$
- **Complessità temporale:** difficile da definire perché dipende dall' accuratezza dell' euristica
- Non si accorge di cammini ripetuti
- Sfrutta poco la memoria: Non riesce a sfruttare aumenti nella disponibilità della memoria per incrementare l' efficienza

Altro esempio: gioco dell'8 (vedere da soli)

- Stato iniziale

2	8	3
	6	4
1	7	5

- Stato finale



1	2	3
8		4
7	6	5

- Costo operatori: unitario
- h = numero tessere fuori posto,
esempio: stato iniziale $h=5$ (sono a posto solo 3, 4 e 5)

	2	8	3
		6	4
1	7	5	

F-cost 5
h 5
F-limit inf

		8	3
2		6	4
1		7	5

F-cost 6
h 5
F-limit 6

2		8	3
6			4
1		7	5

F-cost 6
h 5

2	8	3	
1	6	4	
	7	5	

F-cost 6
h 5

2	8	3	
		6	4
1	7	5	

F-cost 7
h 5

	8		3
2		6	4
1		7	5

F-cost 7
h 5

	2	8	3
		6	4
1		7	5

F-cost 5
h 5
F-limit inf

F-cost 7
h 5

		8	3
2		6	4
1		7	5

	2	8	3
	6		4
1		7	5

F-cost 6
h 5
F-limit 6

F-cost 6
h 5

	2	8	3
1		6	4
		7	5

	2	8	3
		6	4
1		7	5

F-cost 7
h 5

	2		3
	6	8	4
1		7	5

F-cost 7
h 5

	2	8	3
	6	4	
1		7	5

F-cost 8
h 6

	2	8	3
	6	7	4
1			5

F-cost 7
h 5

	2	8	3
		6	4
1		7	5

F-cost 5
h 5
F-limit inf

		8	3
2		6	4
1		7	5

F-cost 7
h 5

2		8	3
6			4
1		7	5

F-cost 7
h 5

	2	8	3
	1	6	4
		7	5

F-cost 6
h 5
F-limit 7

2		8	3
		6	4
1		7	5

F-cost 7
h 5

2		8	3
1		6	4
7			5

F-cost 6
h 4
F-limit 7

	2	8	3
		6	4
1	7	5	

F-cost 5
h 5
F-limit inf

F-cost 7
h 5
F-limit 7

		8	3
2		6	4
1		7	5

2	8	3
6		4
1	7	5

F-cost 7
h 5
F-limit 7

2	8	3
1	6	4
	7	5

F-cost 6
h 5
F-limit 7

2	8	3
	6	4
1	7	5

F-cost 7
h 5

2	8	3
1	6	4
7		5

F-cost 6
h 4
F-limit 7

2	8	3
1	6	4
	7	5

F-cost 8
h 5

2	8	3
1		4
7	6	5

F-cost 6
h 3
F-limit 7

2	8	3
1	6	4
7		5

F-cost 8
h 5

F-cost 7
h 3
F-limit 7

2	8	3
	1	4
7	6	5

F-cost 8
h 4

2	8	3
1	6	4
7		5

F-cost 7
h 3

2		3
1	8	4
7	6	5

F-cost 8
h 4

2	8	3
1	4	
7	6	5

2	8	3
	6	4
1	7	5

F-cost 5
h 5
F-limit inf

F-cost 7
h 5
F-limit 7

	8	3
2	6	4
1	7	5

F-cost 7
h 5
F-limit 7

2	8	3
6		4
1	7	5

F-cost 6
h 5
F-limit 7

2	8	3
1	6	4
	7	5

F-cost 7
h 5
F-limit 7

2	8	3
	6	4
1	7	5

F-cost 6
h 4
F-limit 7

2	8	3
1	6	4
7		5

F-cost 8
h 5

2	8	3
1	6	4
	7	5

F-cost 6
h 3
F-limit 7

2	8	3
1		4
7	6	5

F-cost 8
h 5

2	8	3
1	6	4
7	5	

F-cost 7
h 3
F-limit 7

2	8	3
	1	4
7	6	5

F-cost 8
h 4

2	8	3
1	6	4
7		5

F-cost 7
h 3
F-limit 7

2		3
1	8	4
7	6	5

F-cost 8
h 4
F-limit 7

2	8	3
1	4	
7	6	5

F-cost 8
h 3

	8	3
2	1	4
7	6	5

F-cost 9
h 4

2	8	3
7	1	4
	6	5

F-cost 8
h 3

2	8	3
1		4
7	6	5

2	8	3
	6	4
1	7	5

F-cost 5
h 5
F-limit inf

F-cost 7
h 5
F-limit 7

	8	3
2	6	4
1	7	5

2	8	3
6		4
1	7	5

F-cost 7
h 5
F-limit 7

2	8	3
1	6	4
	7	5

F-cost 6
h 5
F-limit 7

F-cost 7
h 5
F-limit 7

2	8	3
	6	4
1	7	5

2	8	3
1	6	4
7		5

F-cost 6
h 4
F-limit 7

2	8	3
1	6	4
	7	5

F-cost 8
h 5

2	8	3
1		4
7	6	5

F-cost 6
h 3
F-limit 7

2	8	3
1	6	4
7	5	

F-cost 8
h 5

F-cost 8
h 3

2	8	3
	1	4
7	6	5

F-cost 8
h 4

2	8	3
1	6	4
7		5

2		3
1	8	4
7	6	5

F-cost 7
h 3
F-limit 7

2	8	3
1	4	
7	6	5

F-cost 8
h 4

F-cost 7
h 2
F-limit 7

	2	3
1	8	4
7	6	5

F-cost 9
h 4

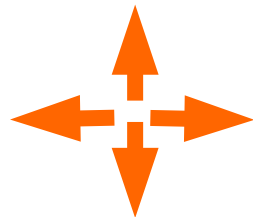
2	3	
1	8	4
7	6	5

F-cost 8
h 3

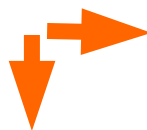
2	8	3
1		4
7	6	5

Funzioni euristiche

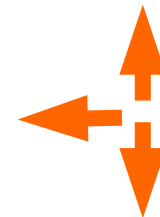
- Studiamo la natura delle euristiche usando il gioco dell' 8, uno dei primi problemi sui quali si è sperimentata la ricerca informata
- In media, generando in modo casuale lo stato iniziale:
 - occorrono 22 mosse per arrivare alla soluzione
 - Il branching factor è pari a 3:



Ho 4 mosse per la
tessera centrale



Ho 2 mosse per gli
spigoli

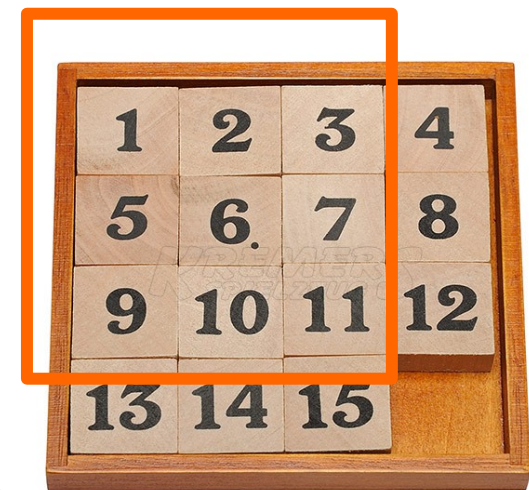


Ho 3 mosse per le
tessere sui lati

Albero e grafo esaustivo di ricerca

- **Albero esaustivo di ricerca:**
contiene 3^{22} nodi (oltre 30.000.000.000)
- **Grafo esaustivo di ricerca:**
contiene “solo” ~ 180.000 nodi, perché si evitano i duplicati
- E se passiamo dal problema dell' 8 al **problema del 15**? Non sembra molto più complesso, invece:
 - Il grafo esplode: avrebbe circa 10^{23} nodi

Problema dell'8



Problema del 15

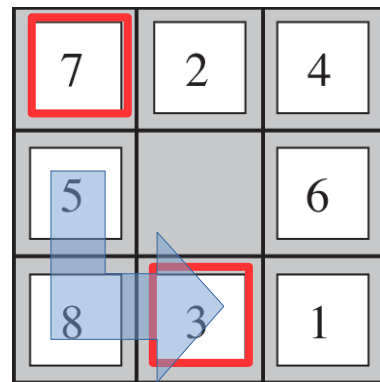
A*: euristiche per il problema del 15 (dell'8)

- A* necessita di euristiche ammissibili, cioè tali da non sovrastimare mai il numero dei passi che portano al goal
- Due possibili euristiche:
 - **h_1 = numero di tessere fuori posto.**
È ammissibile perché ogni tessera fuori posto deve essere spostata almeno una volta.
 - **h_2 = distanza di Manhattan (o block distance).** 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.

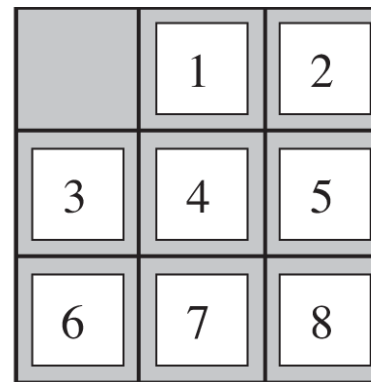


Esempio

- $h1(s) = 8$
tutte le tessere sono fuori posto
- $h2(s) = 3 + 1 + 2 + \dots = 18$
si sommano le distanze di Manhattan calcolate per ogni tessera



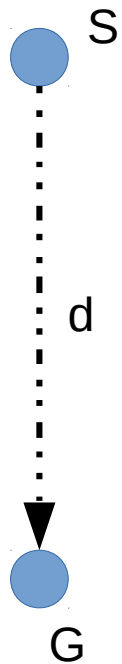
Start State



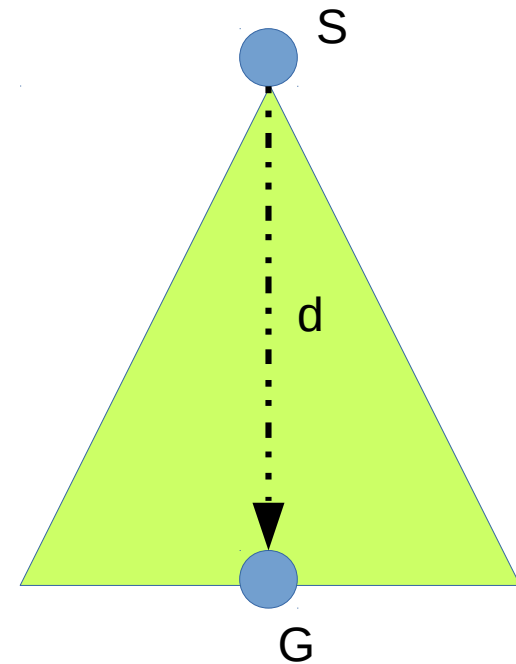
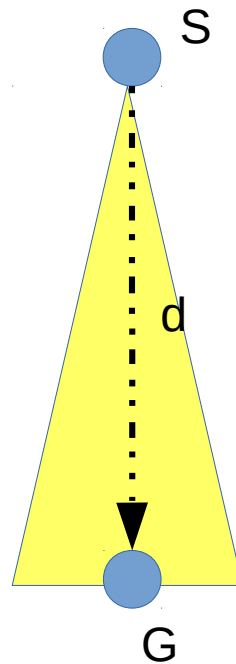
Goal State

Confronto sperimentale

Scopo: vogliamo decidere quale sia l'euristica migliore basandoci sui risultati derivanti dal loro utilizzo, in particolare i numeri di nodi che uno stesso algoritmo di ricerca informato (nel nostro caso A*) produce

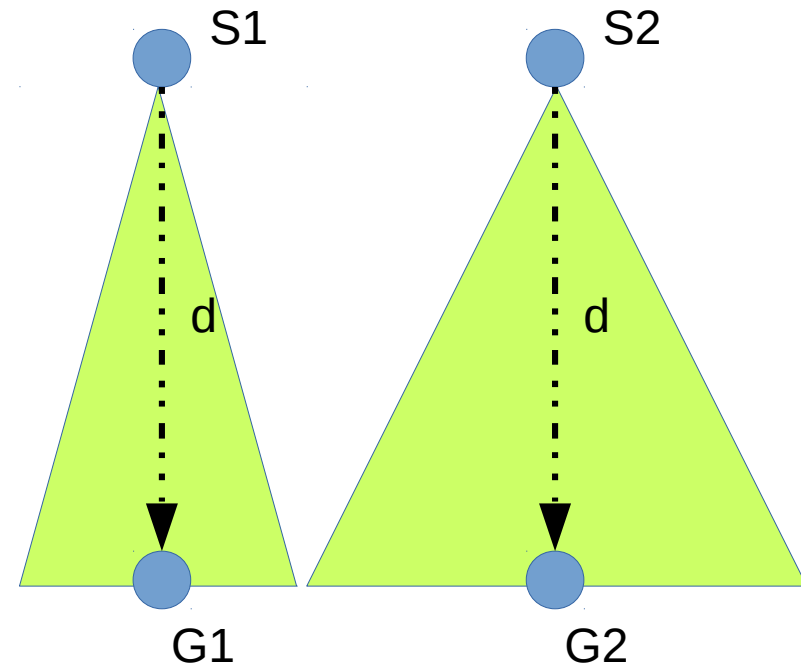
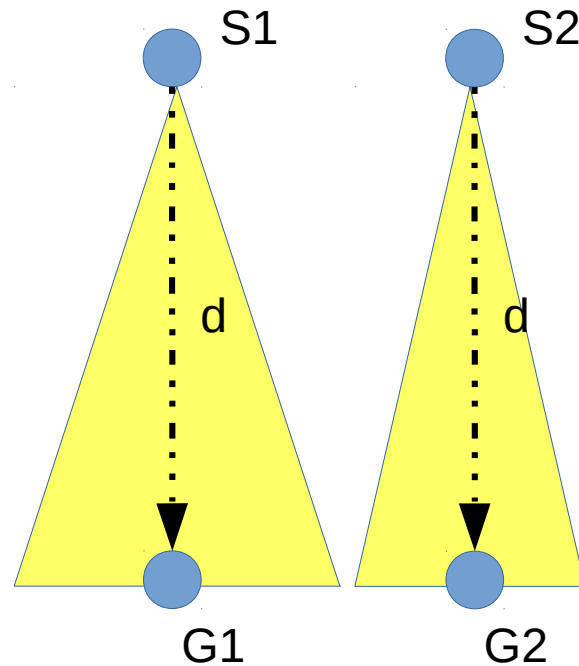
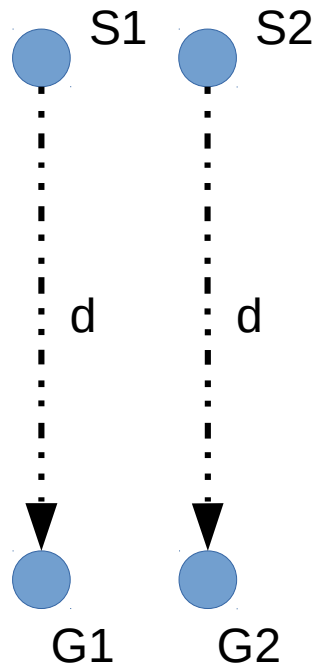


Sia d la profondità della soluzione



Euristiche diverse causeranno in generale l'esplorazione di numeri di nodi differenti.

Ogni istanza produce un risultato: come combinare questi dati?



Se considero due diverse istanze del problema con soluzioni a pari profondità

Il numero di nodi espansi su ciascuna istanza del problema cambia perché anche stato iniziale e goal hanno un'influenza. Come confrontare le euristiche in presenza di tanta variabilità?

Valutazione sperimentale

- La valutazione sperimentale delle euristiche h comprende i seguenti passi:
 - Generare un numero significativo di casi
 - Applicare lo stesso algoritmo di ricerca a ogni caso, tante volte quante sono le euristiche da valutare (una per ogni euristica)
 - Raccogliere i dati risultanti (numero di nodi generati, profondità della soluzione ...)
 - Calcolare i valori medi dei risultati ottenuti in casi affini (esempio quelli in cui la profondità della soluzione è la stessa)
 - Valutare e confrontare le prestazioni

Qualità delle euristiche

- La qualità di un' euristica può essere calcolata computando il **branching factor effettivo b^***
- Supponiamo di avere eseguito A^* su un certo problema, siano:
 - N = numero di nodi generati a partire da un nodo iniziale
 - d = profondità della soluzione trovata
- **b^* = branching factor di un albero uniforme di profondità d che contiene $N+1$ nodi**
- $N+1 = 1 + (b^*) + (b^*)^2 + \dots + (b^*)^d$

Branching factor effettivo (una stima)

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

$$N + 1 = ((b^*)^{d+1} - 1) / (b^* - 1)$$

$$N \approx (b^*)^d \Rightarrow b^* \approx \sqrt[d]{N}$$

Valutazione sperimentale: qualità delle euristiche

- Qualità di un' euristica calcolata a posteriori, a partire da alcuni casi d' uso, cioè problemi in cui viene applicato A^*
- Ogni istanza può produrre b^* differenti ma tali valori saranno tendenzialmente consistenti
- Quindi bastano alcune misure su un campione (piccolo insieme di problemi) per calcolare la bontà di un' euristica
- Le **euristiche migliori** hanno b^* bassi, vicini a 1
- Esse permettono di risolvere problemi complessi in tempi ragionevoli

Esempio: meglio h_1 o h_2 ?

- Sono stati generati in modo casuale 1200 problemi del 15 con profondità di soluzione compresa fra 2 e 24
- Sono stati risolti con Iterative Deepening e A^* , usando h_1 e poi h_2
- I numeri di nodi generati e l' effective branching factor sono stati calcolati caso per caso
- Sono state prodotte le medie per ogni profondità di soluzione

Esempio

d	Search Cost (nodes generated)			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	—	539	113	—	1.44	1.23
16	—	1301	211	—	1.45	1.25
18	—	3056	363	—	1.46	1.26
20	—	7276	676	—	1.47	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26

Figure 3.29 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A^* algorithms with h_1 , h_2 . Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d .

Esempio

d	Search Cost (nodes generated)			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	—	539	113	—	1.44	1.23
16	—	1301	211	—	1.45	1.25
18	—	3056	363	—	1.46	1.26
20	—	7276	676	—	1.47	1.27
22	—	18094	1219	—	1.48	1.28
24	—	39135	1641	—	1.48	1.26

Figure 3.29 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A^* algorithms with h_1 , h_2 . Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d .

Nodi prodotti
con $A(h_1)$

\gg

Nodi prodotti
con $A(h_2)$

b^* per $A(h_1)$

\gg

b^* per $A(h_2)$