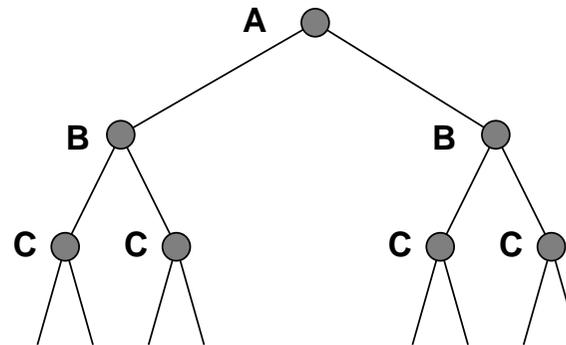
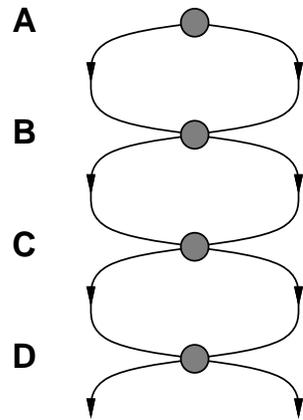


Stati ripetuti

Se non si evitano stati ripetuti, un problema con numero di stati lineari può generare un numero esponenziale di nodi!



Stati ripetuti

Il problema si affronta di solito in 3 possibili modi:

- evitare di generare il nuovo nodo se è uguale al nodo corrente (spazio: costante)
- evitare di generare il nuovo nodo se è uno degli avi (nel cammino dalla radice al nodo corrente) (spazio: $O(d)$)
- evitare di generare il nuovo nodo se è stato già generato (spazio: $O(b^d)$, in realtà $O(s)$)

Ricerca su Grafo

```
function RICERCA-GRAFO(problema, frontiera) returns una soluzione, o fallimento  
chiuso ← un insieme vuoto  
frontiera ← INSERISCI(CREA-NODO(STATO-INIZIALE[problema]), frontiera)  
loop do  
  if VUOTO?(frontiera) then return fallimento  
  nodo ← RIMUOVI-PRIMO(frontiera)  
  if TEST-OBIETTIVO[problema](STATO[nodo]) then return SOLUZIONE(nodo)  
  if STATO[nodo] non è in chiuso then  
    aggiungi STATO[nodo] a chiuso  
    frontiera ← INSERISCI-TUTTI(ESPANDI(nodo, problema), frontiera)  
end
```

Algoritmi di ricerca informati

CAPITOLO 4

– presentazione basata sui lucidi di S. Russell –

Sommario

- ◇ Ricerca best-first
- ◇ Ricerca A*
- ◇ Euristiche
- ◇ Hill-climbing
- ◇ Simulated annealing

Ripasso: algoritmo di ricerca generale

```
function RICERCA-ALBERO(problema, frontiera) returns una soluzione, o il fallimento
```

```
frontiera ← INSERISCI(CREA-NODO(STATO-INIZIALE[problema]), frontiera)
loop do
  if VUOTA?(frontiera) then return fallimento
  nodo ← RIMUOVI-PRIMO(frontiera)
  if TEST-OBIETTIVO[problema] applicato a STATO[nodo] ha successo
    then return SOLUZIONE(nodo)
  frontiera ← INSERISCI-TUTTI(ESPANDI(nodo, problema), frontiera)
```

```
function ESPANDI(nodo, problema) returns un insieme di nodi
```

```
successori ← l'insieme vuoto
for each ⟨azione, risultato⟩ in FUNZIONE-SUCCESSORE[problema](STATO[nodo]) do
  s ← un nuovo NODO
  STATO[s] ← risultato
  NODO-PADRE[s] ← nodo
  AZIONE[s] ← azione
  COSTO-DI-CAMMINO[s] ← COSTO-DI-CAMMINO[nodo] +
    COSTO-DI-PASSO(nodo, azione, s)
  PROFONDITÀ[s] ← PROFONDITÀ[nodo] + 1
  aggiungi s a successori
return successori
```

Una strategia è scelta definendo *l'ordine di espansione dei nodi*

Ricerca best-first

Idea: usare una *funzione di valutazione* $f(n)$ per ogni nodo n
– stima di “desiderabilità”

⇒ Espandere il nodo non espanso più desiderabile

Implementazione:

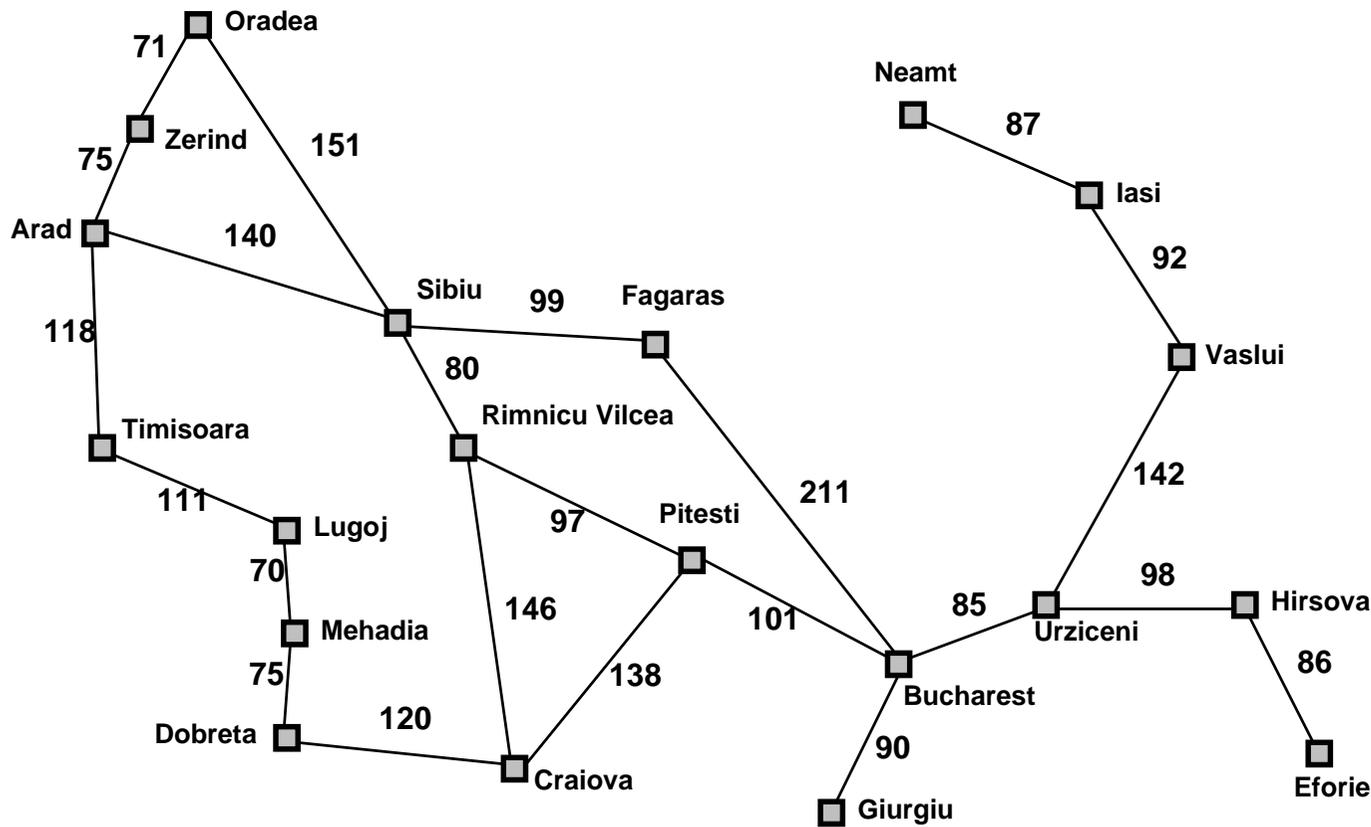
la frontiera è una coda ordinata in modo decrescente rispetto alla desiderabilità

Casi speciali:

ricerca greedy

ricerca A^*

Romania con costo dei passi in km



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Ricerca greedy

Funzione di valutazione definita sulla base di una *funzione euristica*

$h(n)$ = stima del costo dal nodo n al goal più vicino

Ad es., $h_{\text{SLD}}(n)$ = distanza in linea d'aria da n a Bucharest

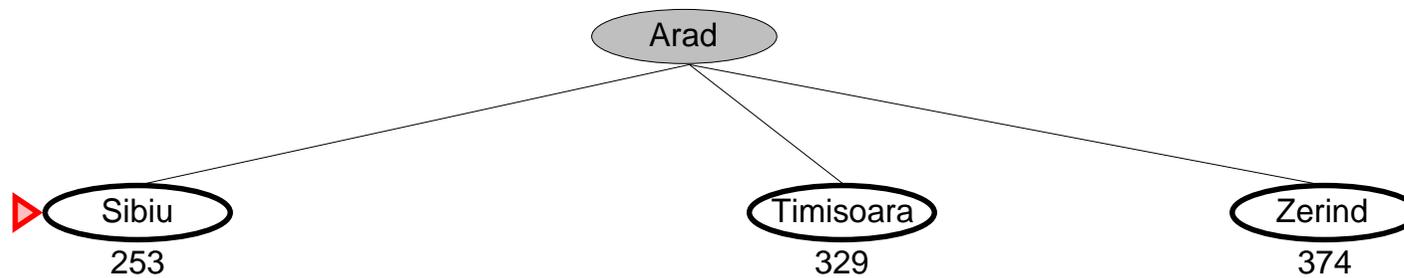
La ricerca greedy espande il nodo che *appare* essere il più vicino al goal:

$f(n) = h(n)$

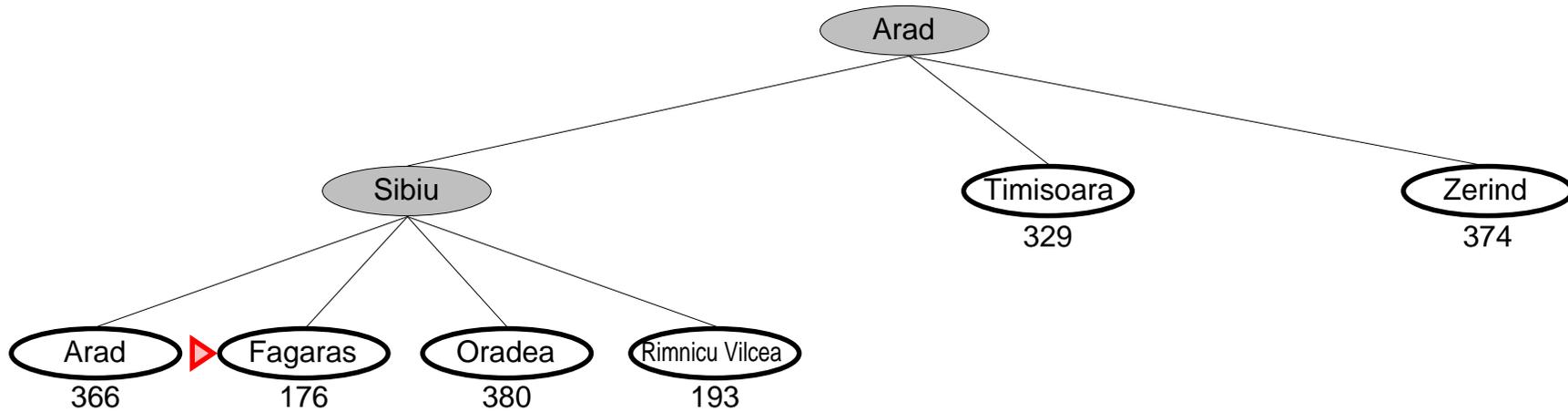
Esempio di ricerca greedy

▶ Arad
366

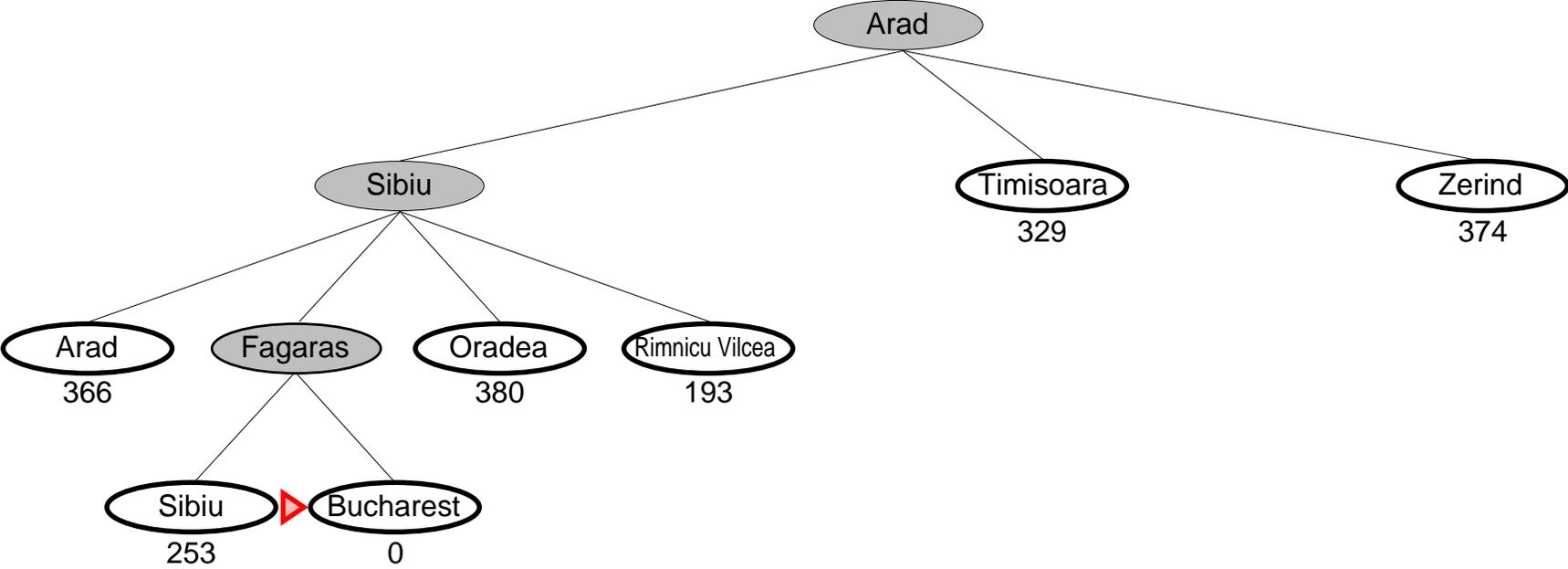
Esempio di ricerca greedy



Esempio di ricerca greedy



Esempio di ricerca greedy



Proprietà della ricerca greedy

Completa??

Proprietà della ricerca greedy

Completa?? No – può restare intrappolata in cicli, per es., avendo Oradea come goal,

lasi → Neamt → lasi → Neamt →

Completa in spazi finiti con controllo di ripetizione di stati

Time??

Proprietà della ricerca greedy

Completa?? No – può restare intrappolata in cicli, per es., avendo Oradea come goal,

lasi → Neamt → lasi → Neamt →

Completa in spazi finiti con controllo di ripetizione di stati

Tempo?? $O(b^m)$, ma l'uso di una buona euristica può dare miglioramenti enormi

Space??

Proprietà della ricerca greedy

Completa?? No – può restare intrappolata in cicli, per es., avendo Oradea come goal,

lasi → Neamt → lasi → Neamt →

Completa in spazi finiti con controllo di ripetizione di stati

Tempo?? $O(b^m)$, ma l'uso di una buona euristica può dare miglioramenti enormi

Spazio?? $O(b^m)$ – mantiene tutti i nodi in memoria

Ottima??

Proprietà della ricerca greedy

Completa?? No – può restare intrappolata in cicli, per es., avendo Oradea come goal,

lasi → Neamt → lasi → Neamt →

Completa in spazi finiti con controllo di ripetizione di stati

Tempo?? $O(b^m)$, ma l'uso di una buona euristica può dare miglioramenti enormi

Spazio?? $O(b^m)$ – mantiene tutti i nodi in memoria

Ottima?? No

Ricerca A*

Idea: evitare di espandere cammini che sono già costosi

Funzione di valutazione $f(n) = g(n) + h(n)$

$g(n)$ = costo già sostenuto per raggiungere n

$h(n)$ = costo stimato da n al goal

$f(n)$ = costo totale stimato del cammino che passa da n al goal

La ricerca A* usa una euristica *ammisibile*

cioè, $h(n) \leq h^*(n)$, dove $h^*(n)$ è il costo *effettivo* da n .

(si richiede anche $h(n) \geq 0$, così che $h(G) = 0$ per ogni goal G .)

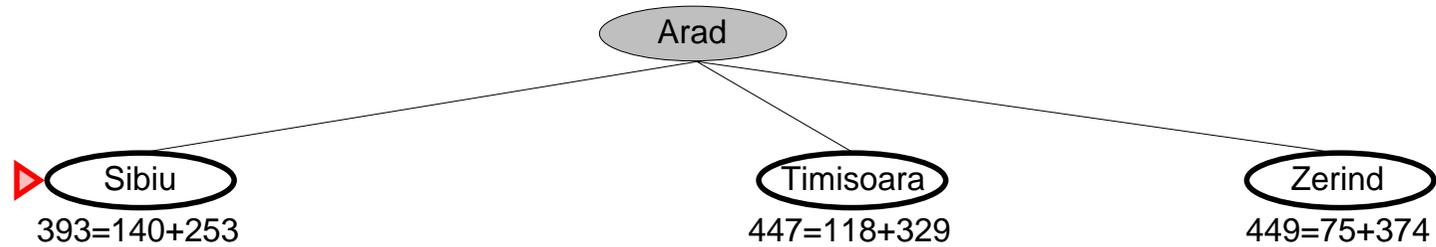
Es., $h_{\text{SLD}}(n)$ non sovrastima mai la distanza effettiva

Teorema: la ricerca A* è ottima

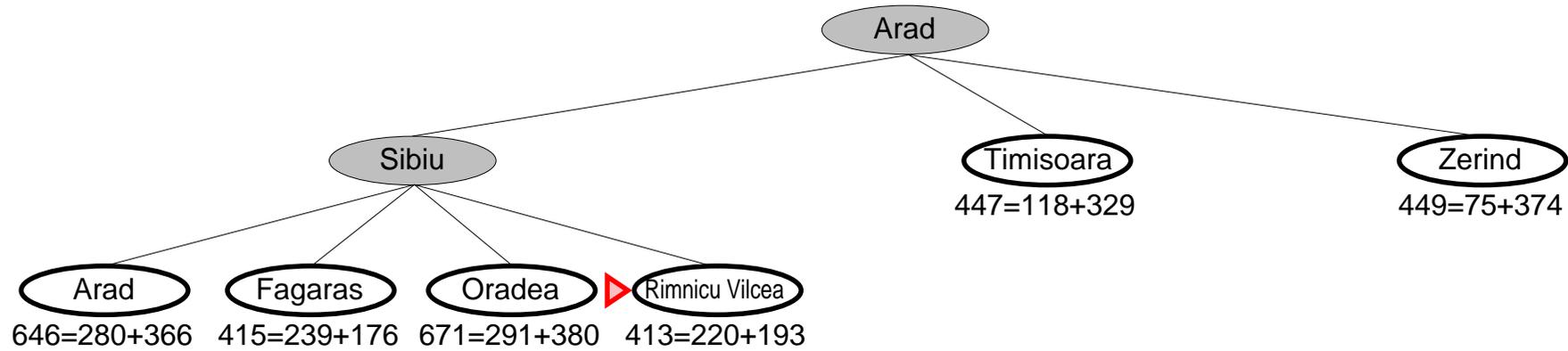
Esempio di ricerca A^*

▶ Arad
 $366=0+366$

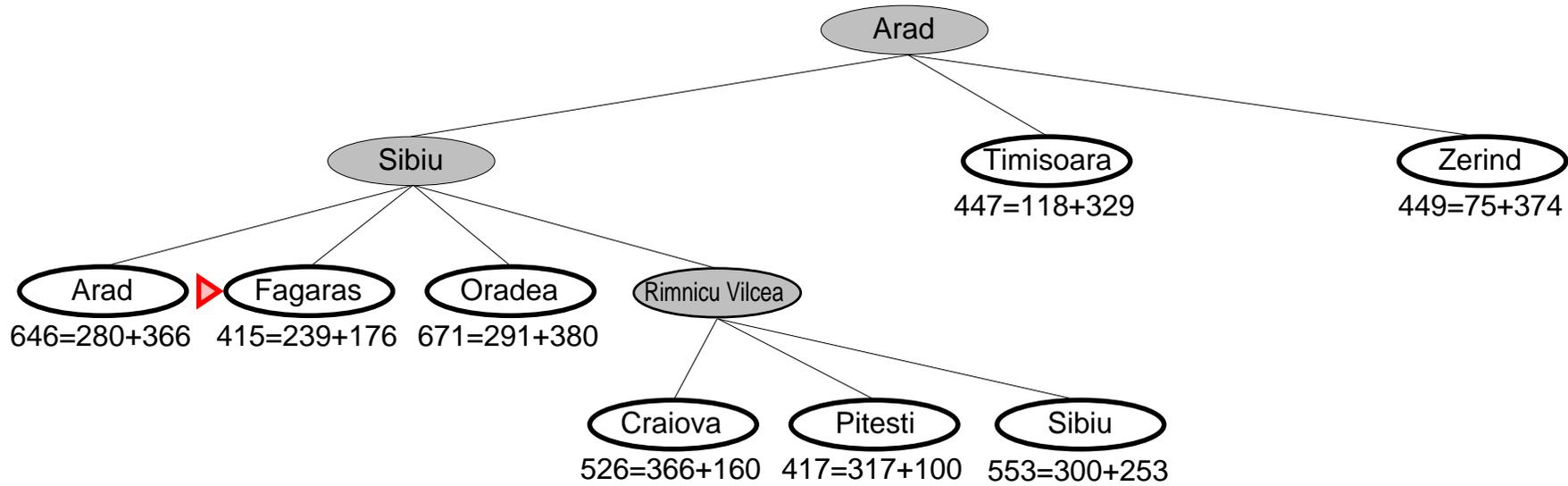
Esempio di ricerca A*



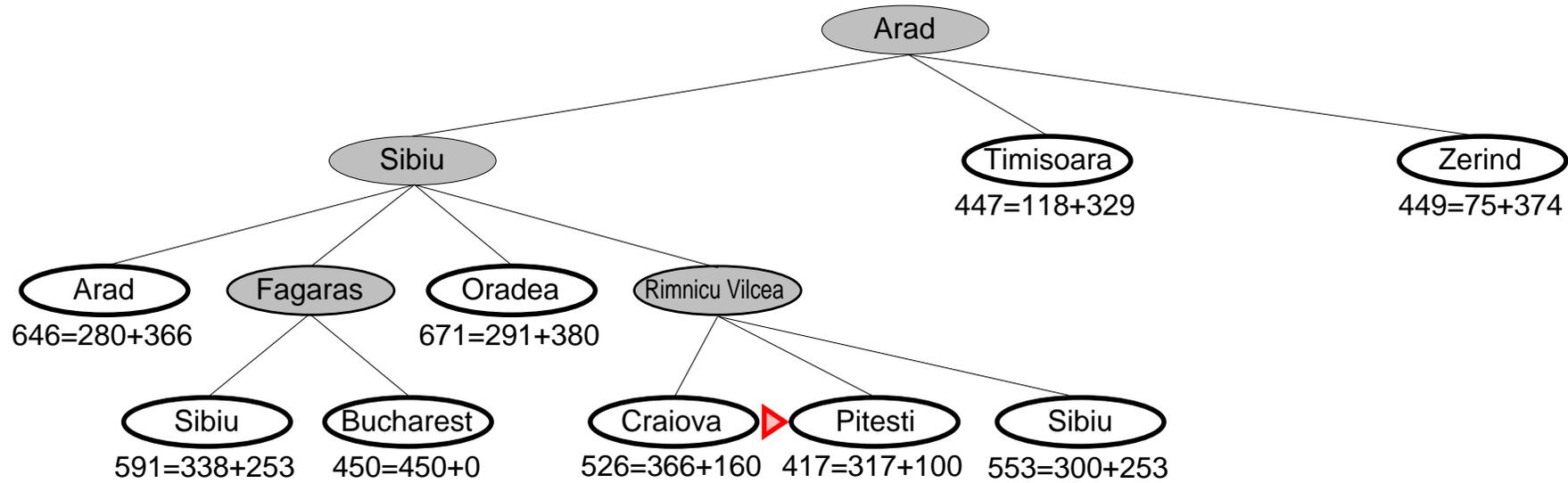
Esempio di ricerca A*



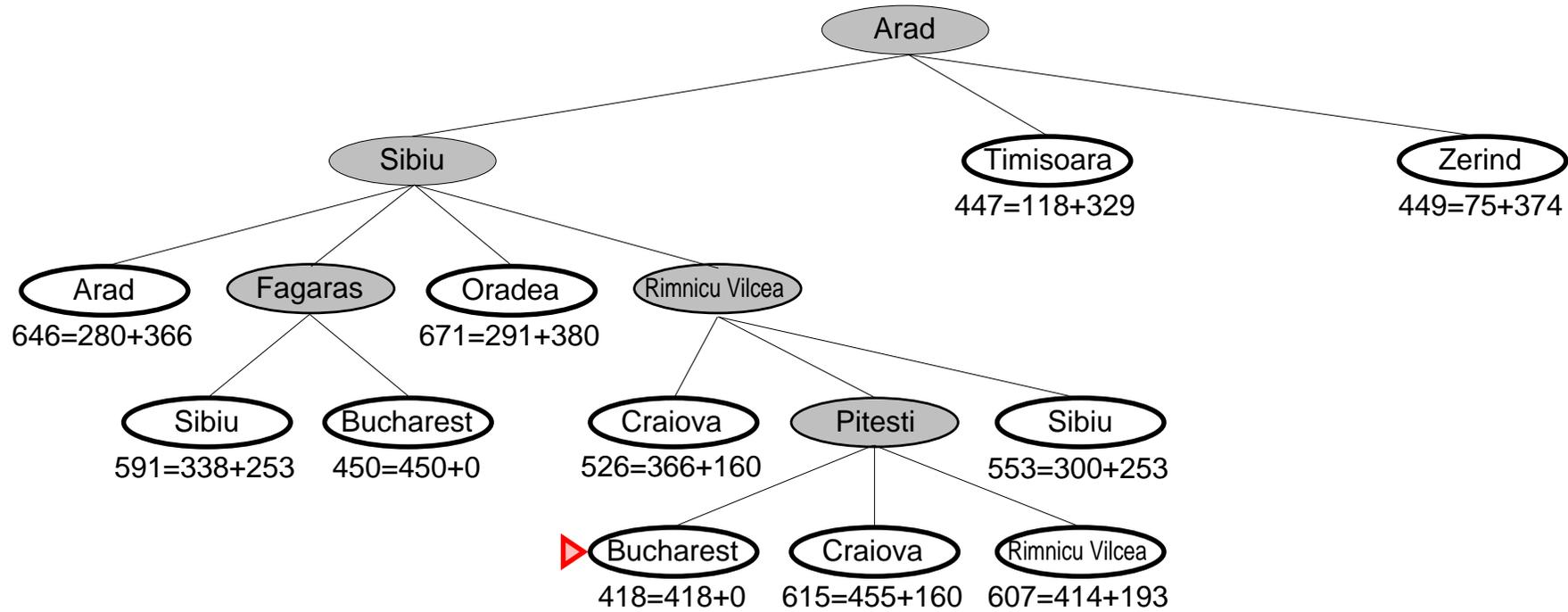
Esempio di ricerca A*



Esempio di ricerca A*

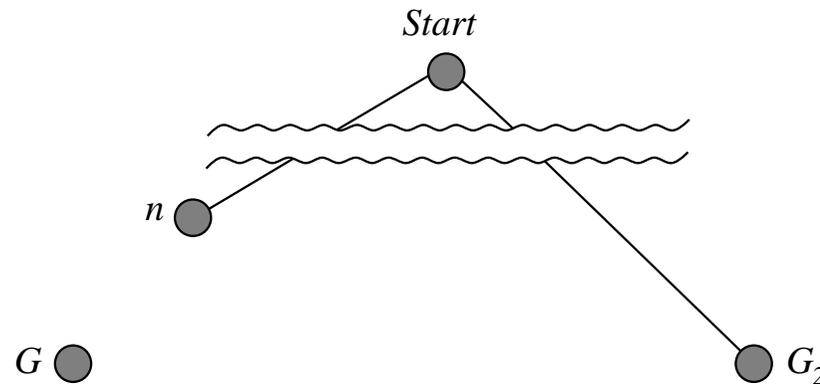


Esempio di ricerca A*



Ottimalità di A^* (prova per albero di ricerca)

Supponiamo che un goal sub-ottimo G_2 sia stato generato e che si trovi nella coda. Sia n un nodo non ancora espanso su un cammino minimo verso un goal ottimo G .



$$\begin{aligned} f(G_2) &= g(G_2) && \text{poiché } h(G_2) = 0 \\ &> g(G) && \text{poiché } G_2 \text{ è subottimo} \\ &\geq f(n) && \text{poiché } h \text{ è ammissibile} \end{aligned}$$

Poiché $f(G_2) > f(n)$, A^* non selezionerà mai G_2 per l'espansione

Ottimalità di A^* per ricerca su grafo

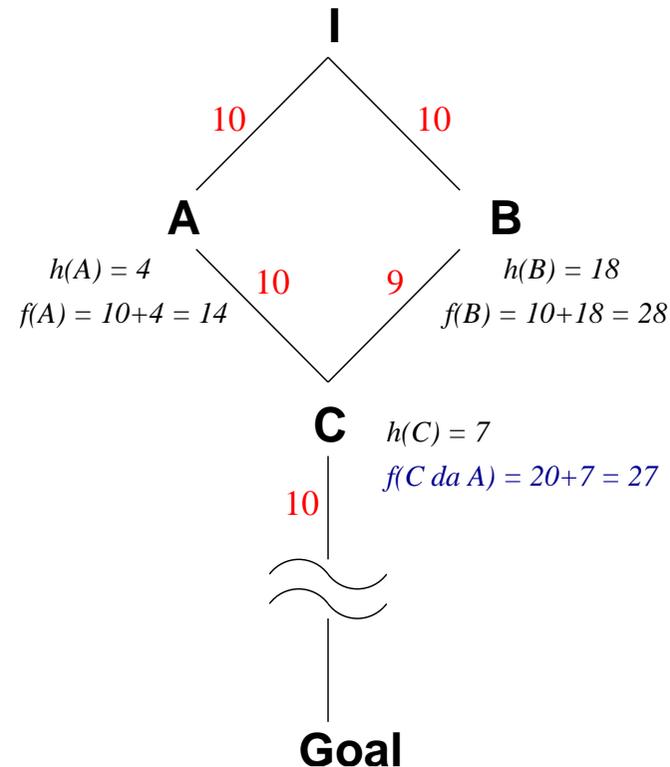
Notare che se si considera la versione dell'algoritmo di ricerca che evita di visitare più volte lo stesso stato (ricerca su grafo), allora la prova vista non funziona! Perché ?

Ottimalità di A^* per ricerca su grafo

Il motivo per cui la prova non funziona risiede nel fatto che si rischia di scartare un' occorrenza ripetuta di uno stato che si trova su un cammino ottimo!

Due soluzioni:

- scartare sempre il cammino più costoso ogni volta che si visita nuovamente uno stesso stato (complicato!)
- usare euristiche *consistenti*



Consistenza

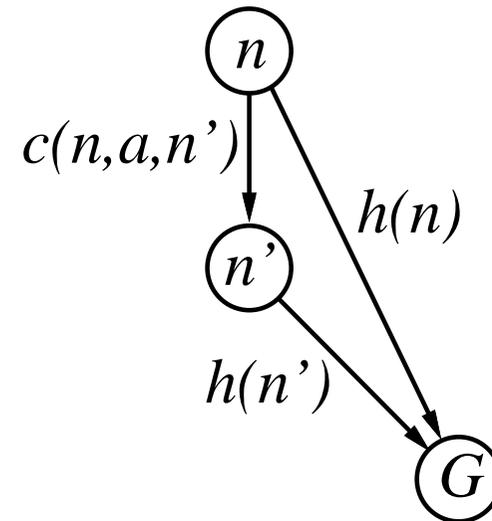
Una euristica è *consistente* se

$$h(n) \leq c(n, a, n') + h(n')$$

Se h è consistente, si ha

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

cioè, $f(n)$ è non decrescente lungo un cammino

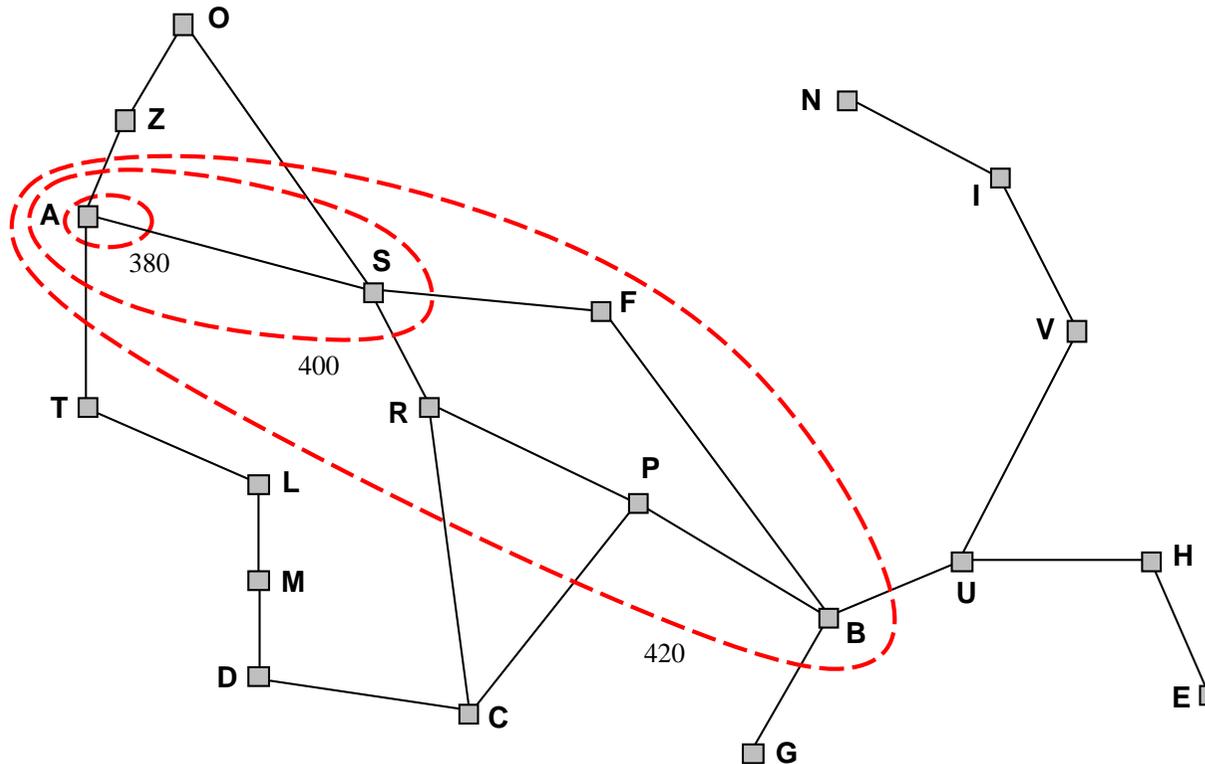


Ottimalità di A^* (più utile; euristica consistente)

Lemma: A^* espande i nodi in ordine di valore di f

Gradualmente, aggiunge dei “contorni di f ” dei nodi (si confronti con la ricerca breadth-first che aggiunge livelli)

Il contorno i possiede tutti nodi con $f = f_i$, dove $f_i < f_{i+1}$



Proprietà di A^*

Completa??

Proprietà di A^*

Completa?? Sì, a meno che non ci sia un numero infinito di nodi con $f \leq f(G)$

Tempo??

Proprietà di A^*

Completa?? Sì, a meno che non ci sia un numero infinito di nodi con $f \leq f(G)$

Tempo?? Esponenziale in [errore relativo in $h \times$ lunghezza di sol.]

Spazio??

Proprietà di A^*

Completa?? Sì, a meno che non ci sia un numero infinito di nodi con $f \leq f(G)$

Tempo?? Esponenziale in [errore relativo in $h \times$ lunghezza di sol.]

Spazio?? Mantiene tutti i nodi in memoria

Ottima??

Proprietà di A^*

Completa?? Sì, a meno che non ci sia un numero infinito di nodi con $f \leq f(G)$

Tempo?? Esponenziale in [errore relativo in $h \times$ lunghezza di sol.]

Spazio?? Mantiene tutti i nodi in memoria

Ottima?? Sì—non può espandere f_{i+1} finché f_i non è finita

A^* espande tutti i nodi con $f(n) < C^*$

A^* espande alcuni nodi con $f(n) = C^*$

A^* non espande alcun nodo con $f(n) > C^*$

Altra proprietà di A^*

A^* non è solo ottimo:

non esiste altro algoritmo ottimo che sicuramente espande meno nodi di A^*
(se non per risolvere “patte” sui nodi con f uguale al valore ottimo)

Questo è vero perché se così non fosse, allora tale algoritmo rischierebbe di non esplorare nodi ottimi, e quindi non sarebbe un algoritmo ottimo.

Euristiche ammissibili

Per esempio, consideriamo 8-puzzle:

$h_1(n)$ = numero di tasselli in posizione errata

$h_2(n)$ = distanza di **Manhattan** totale

(cioè numero di “quadrati” dalla posizione desiderata per ogni tassello)

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

$$h_1(S) = ??$$

$$h_2(S) = ??$$

Euristiche ammissibili

Per esempio, consideriamo 8-puzzle:

$h_1(n)$ = numero di tasselli in posizione errata

$h_2(n)$ = distanza di **Manhattan** totale

(cioè numero di “quadrati” dalla posizione desiderata per ogni tassello)

7	2	4
5		6
8	3	1

Start State

1	2	3
4	5	6
7	8	

Goal State

$$h_1(S) = ?? \quad 6$$

$$h_2(S) = ?? \quad 4+0+3+3+1+0+2+1 = 14$$

Dominanza

Se $h_2(n) \geq h_1(n)$ per tutti gli n (entrambe ammissibili)
allora h_2 *domina* h_1 ed è migliore per la ricerca

Tipici costi di ricerca:

$d = 14$ IDS = 3,473,941 nodi

$A^*(h_1) = 539$ nodi

$A^*(h_2) = 113$ nodi

$d = 24$ IDS \approx 54,000,000,000 nodi

$A^*(h_1) = 39,135$ nodi

$A^*(h_2) = 1,641$ nodi

Problemi rilassati

Euristiche ammissibili possono essere derivate dal costo *esatto* di una soluzione di una versione *rilassata* del problema

Se le regole di 8-puzzle sono rilassate così che un tassello può muoversi *ovunque*, allora $h_1(n)$ corrisponde alla soluzione sul cammino più breve

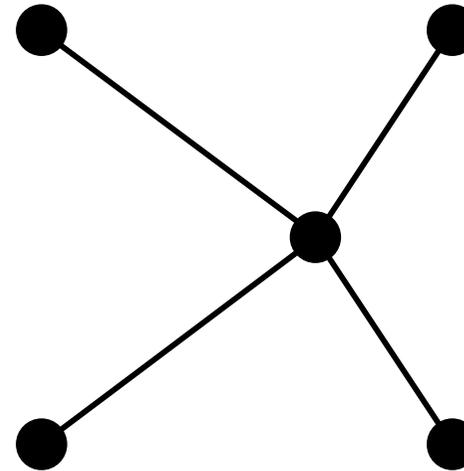
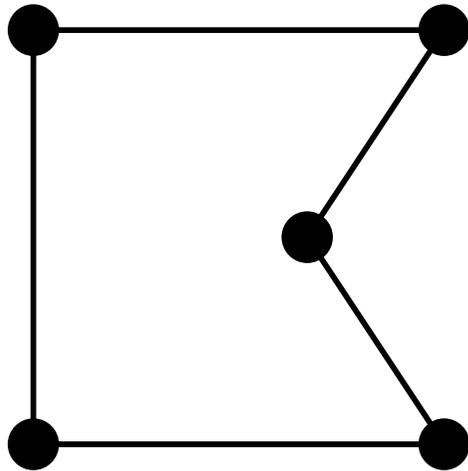
Se le regole di 8-puzzle sono rilassate così che un tassello può muoversi *ad ogni "quadrato" adiacente*, allora $h_2(n)$ corrisponde alla soluzione sul cammino più breve

Punto chiave: il costo della soluzione ottima di un problema rilassato non è più grande del costo della soluzione ottima del problema originario

Problemi rilassati

Esempi noti: **Problema del commesso viaggiatore (TSP)**

Trovare il percorso più corto che visiti tutte le città esattamente una volta



Albero di copertura minimo (Minimum spanning tree) può essere calcolato in $O(n^2)$ e fornisce un limite inferiore al percorso (aperto) più breve

Alcune considerazioni finali su A^*

A^* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Alcune considerazioni finali su A^*

A^* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Come si può ridurre l'occupazione di memoria??

Alcune considerazioni finali su A^*

A^* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Come si può ridurre l'occupazione di memoria??

La soluzione più semplice è quella di adattare l'idea dell'iterative deepening al contesto delle euristiche

Come??

Alcune considerazioni finali su A^*

A^* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Come si può ridurre l'occupazione di memoria??

La soluzione più semplice è quella di adattare l'idea dell'iterative deepening al contesto delle euristiche

Come??

Algoritmo Iterative Deepening A^* (IDA^*):

come A^* , però:

- non si inseriscono nella coda nodi con valore di f maggiori del valore di *cutoff*
- il valore di *cutoff* alla iterazione successiva si pone uguale al minimo valore di f dei nodi non inseriti nella coda

Alcune considerazioni finali su A^*

A^* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Come si può ridurre l'occupazione di memoria??

La soluzione più semplice è quella di adattare l'idea dell'iterative deepening al contesto delle euristiche

Come??

Esistono però soluzioni migliori, come RBFS, MA^* , SMA^*