

## 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

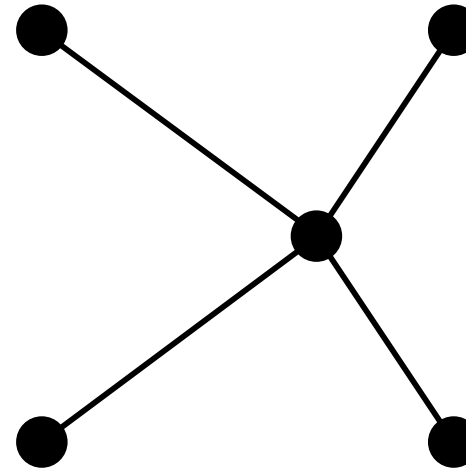
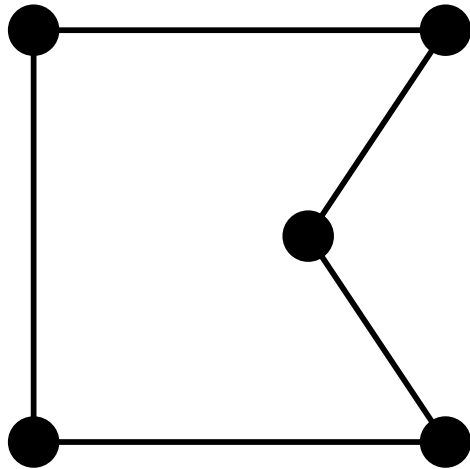
Se le regole di 8-puzzle sono rilassate così che un tassello può muoversi *ad ogni "quadrato" adiacente*, allora  $h_2(n)$  corrisponde alla soluzione sul cammino più breve

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,  $MA^*$ ,  $SMA^*$

## Ricerca Best First Ricorsiva (RBFS)

La Ricerca Best First Ricorsiva è un algoritmo ricorsivo che imita una ricerca in profondità, utilizzando **spazio lineare**

In particolare:

invece di seguire indefinitamente il cammino corrente, **tiene traccia dell'f-valore del miglior cammino alternativo** che parte da uno degli avi

se il nodo corrente **supera l'f-valore alternativo**, la ricorsione **torna indietro** al cammino alternativo

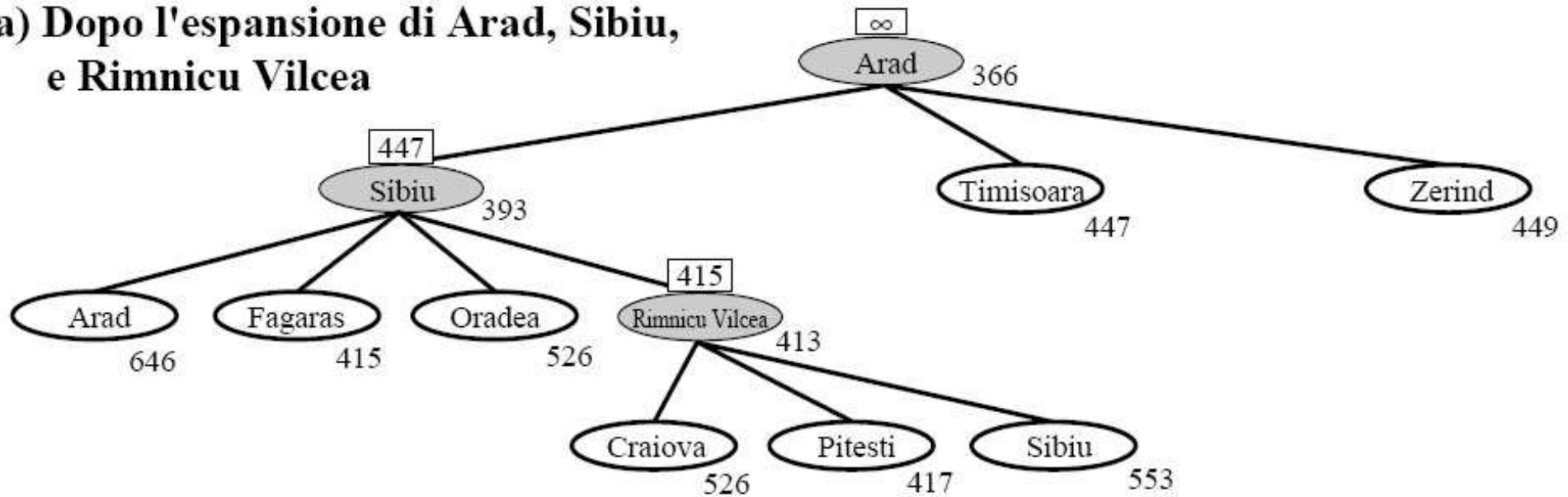
durante il ritorno, si **sostituisce l'f-valore** di ogni nodo lungo il cammino con **il miglior f-valore dei suoi nodi figli**

# Ricerca Best First Ricorsiva

```
function RICERCA-BEST-FIRST-RICORSIVA(problema) returns una soluzione, o fallimento  
  RBFS(problema, CREA-NODO(STATO-INIZIALE[problema]),  $\infty$ )  
  
function RBFS(problema, nodo, f-limite) returns una soluzione, o fallimento e un nuovo  
limite all' f-costo  
  if TEST-OBIETTIVO[problema](STATO[nodo]) then return SOLUZIONE(nodo)  
  successori  $\leftarrow$  ESPANDI(nodo, problema)  
  if successori è vuoto then return fallimento,  $\infty$   
  for each s in successori do  
    f[s]  $\leftarrow$  max(g(s) + h(s), f[nodo])  
  repeat  
    best  $\leftarrow$  il nodo con f-valore minimo in successori  
    if f[best] > f-limite then return fallimento, f[best]  
    alternativa  $\leftarrow$  il secondo f-valore più basso in tutti i successori  
    risultato, f[best]  $\leftarrow$  RBFS(problema, best, min(f-limite, alternativa))  
    if risultato  $\neq$  fallimento then return risultato
```

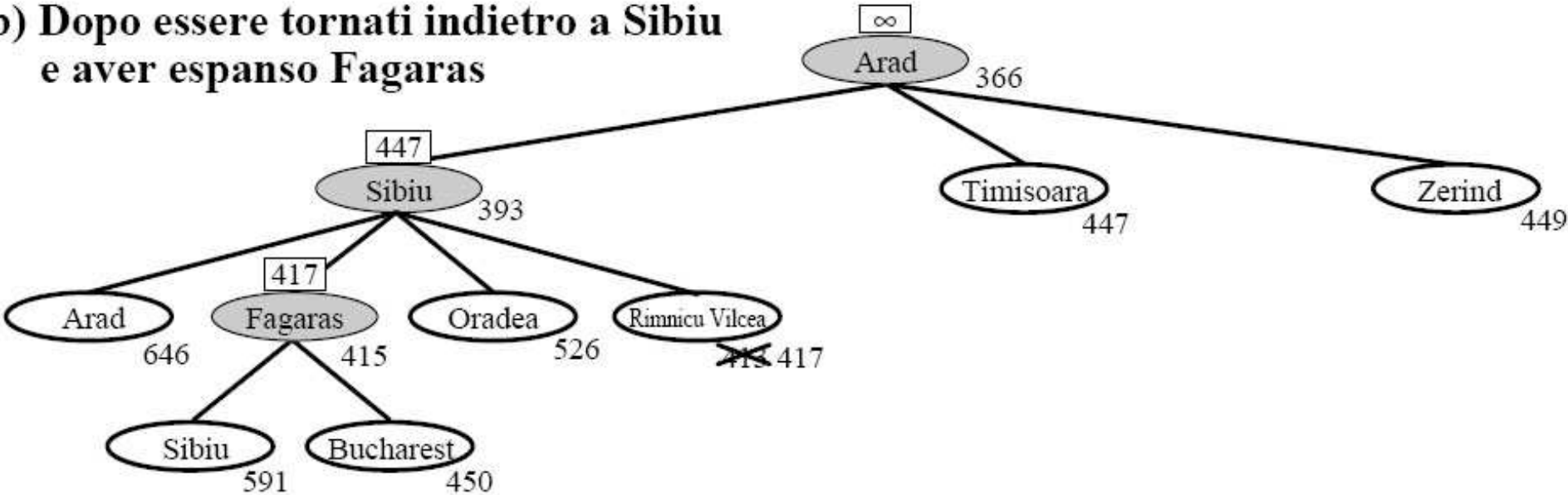
# Esempio esecuzione

(a) Dopo l'espansione di Arad, Sibiu, e Rimnicu Vilcea



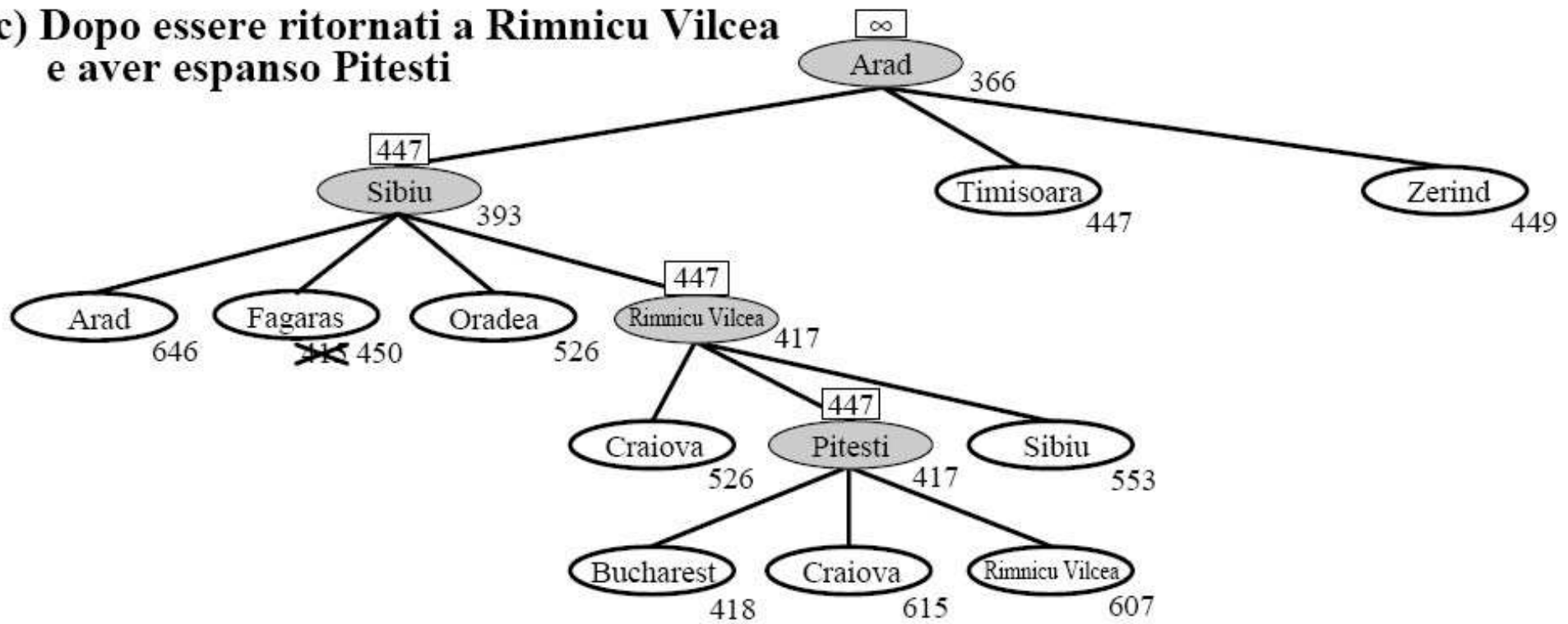
# Esempio esecuzione

(b) Dopo essere tornati indietro a Sibiu e aver espanso Fagaras



# Esempio esecuzione

**(c) Dopo essere ritornati a Rimnicu Vilcea e aver espanso Pitesti**





## Proprietà Ricerca Best First Ricorsiva

Come  $A^*$ , la Ricerca Best First Ricorsiva è ottima se la funzione euristica  $h(n)$  è **ammissibile**

Inoltre:

- Complessità spaziale:  $O(bd)$
- Complessità temporale: difficile da definire (ma esponenziale, nel caso pessimo...)

In ogni caso, il problema di RBFS (ed anche  $IDA^*$ ) consiste nell'uso di **troppa poca memoria**

Gli algoritmi  $MA^*$  (*memory-bounded  $A^*$* ) e  $SMA^*$  (*simplified  $MA^*$* ) utilizzano esattamente **tutta la memoria disponibile**

## Simplified MA\*

La Simplified MA\* procede come A\*, espandendo la foglia migliore fino a quando la memoria è piena

Successivamente, per poter proseguire, deve cancellare dalla memoria un nodo **vecchio** per far posto ad un nodo **nuovo**

In particolare:

scarta sempre il nodo foglia **peggiore** (f-valore più alto, cioè quello in fondo alla coda)

l'f-valore del nodo scartato viene **riportato sul nodo padre**

si espande di nuovo il nodo padre quando **tutti gli altri cammini sono peggiori**

## Simplified MA\*: complicazione...

Cosa succede se *tutte* le foglie hanno lo stesso f-valore??

SMA\* rischierebbe di scegliere lo stesso nodo sia per la cancellazione che per l'espansione!

Politica adottata:

rimuove la foglia **più vecchia** ed espande la foglia **più recente**

se queste **coincidono**, significa che **la soluzione che passa da quel nodo non può essere contenuta dalla memoria disponibile...**

... e quindi **è giusto rimuoverla!**

Quindi SMA\* è

- **completa solo se** la soluzione può essere contenuta in memoria
- **ottima** se la soluzione è **raggiungibile**  
... altrimenti restituisce la **migliore soluzione raggiungibile**