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

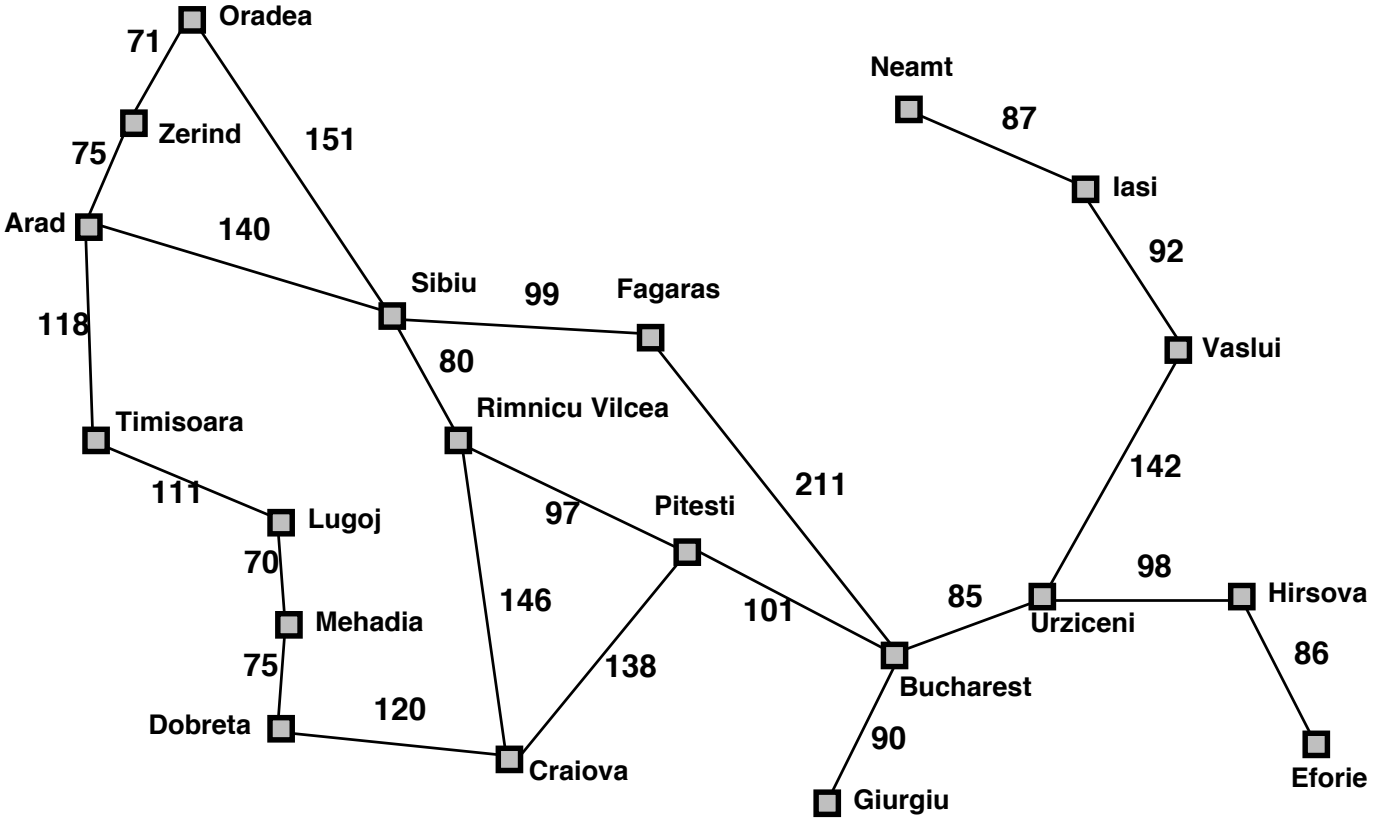
Idea: usare una *funzione di valutazione* $f(n)$ per ogni nodo n
– stima di "desiderabilità"

$\Rightarrow$ 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*

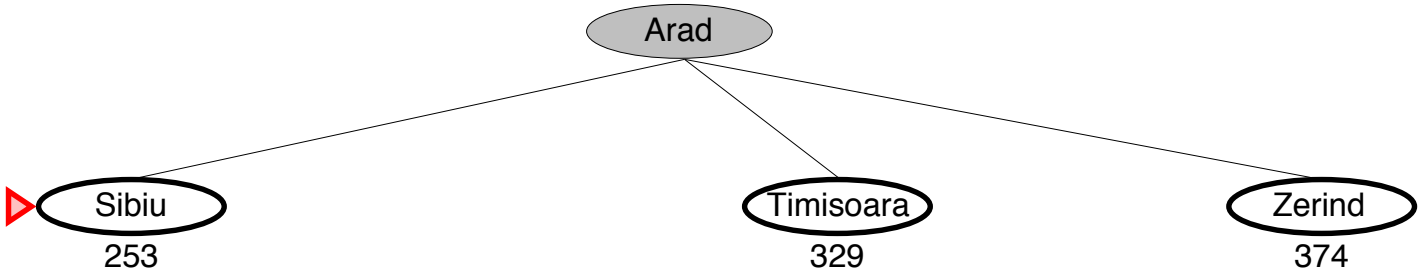
$h(n) =$ stima del costo dal nodo n al goal più vicino

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

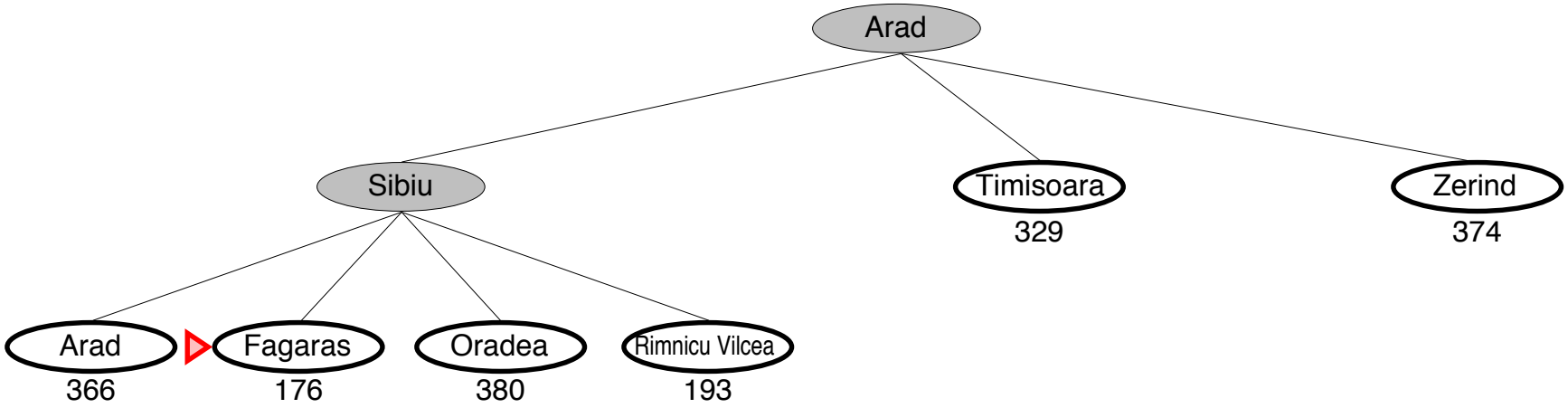
La ricerca greedy espande il nodo che *appare* essere il più vicino al goal: $f(n) = h(n)$

Esempio di ricerca greedy

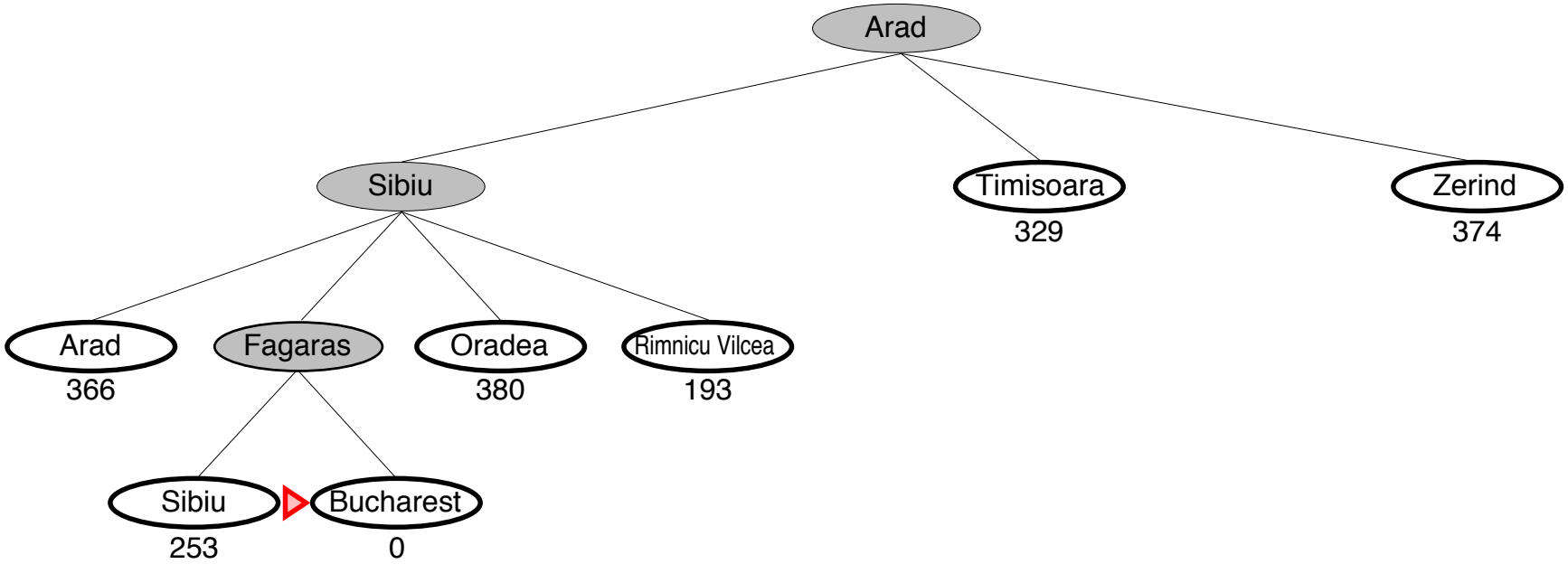
Esempio di ricerca greedy



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,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

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,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

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,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

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,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

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

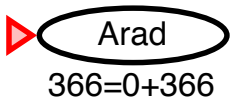
$g(n) =$ costo già sostenuto per raggiongere n
$h(n) =$ costo stimato da n al goal
$f(n) =$ costo totale stimato del cammino che passa da n al goal

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 $h(n) \geq 0$, cosìche $h(G) = 0$ per ogni goal G.)

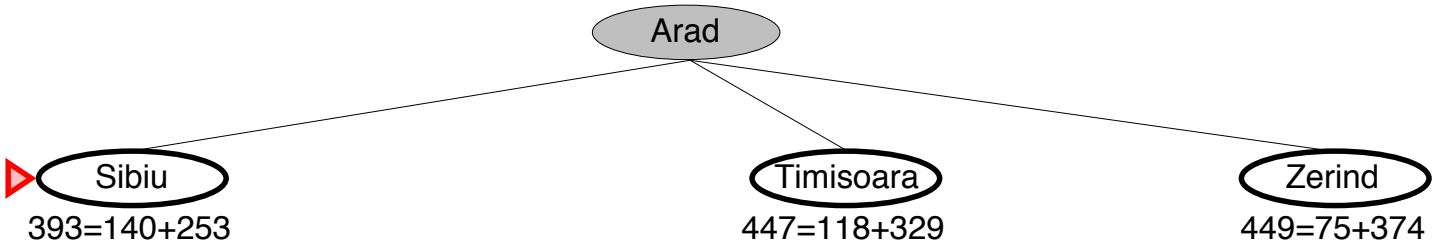
Es., $h_{\mathrm{SLD}}(n)$ non sovrastima mai la distanza effettiva

Teorema: la ricerca A* è ottima

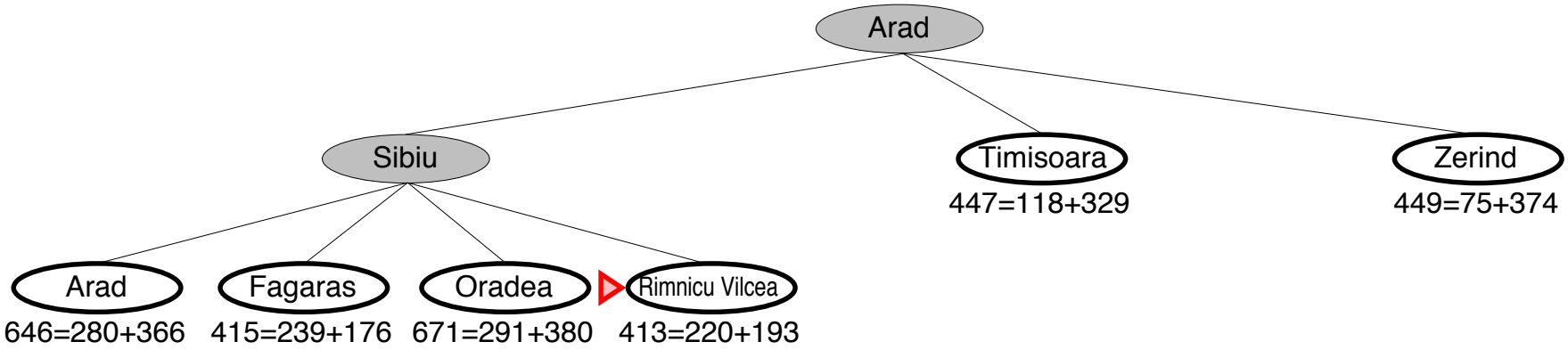
Esempio di ricerca A*



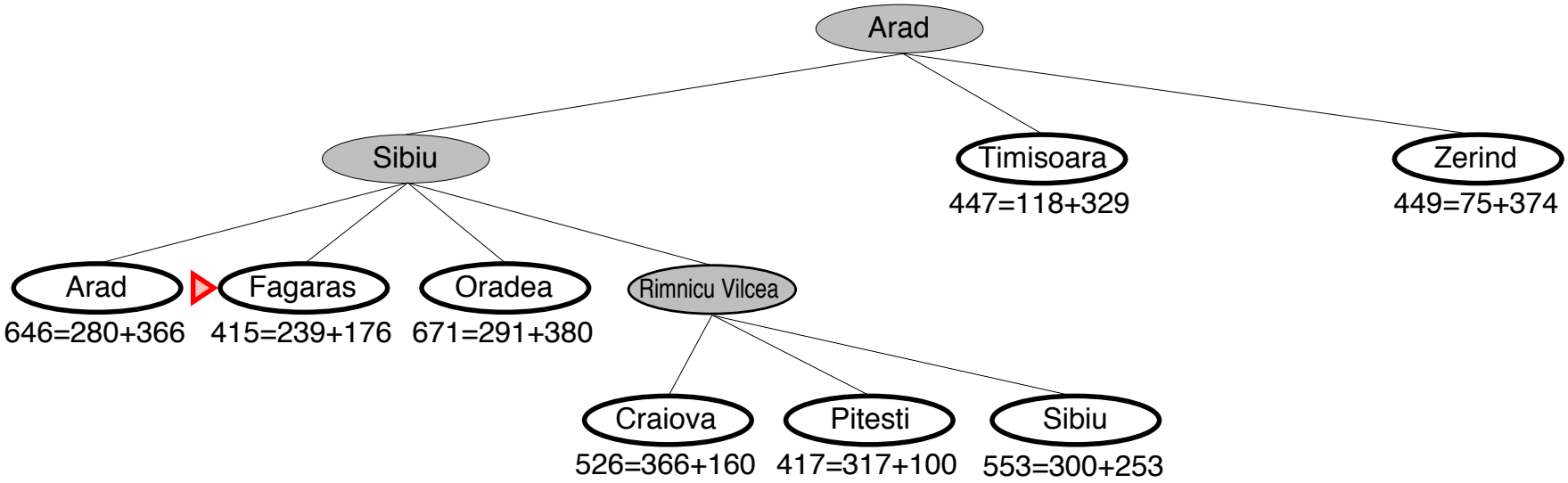
Esempio di ricerca A*



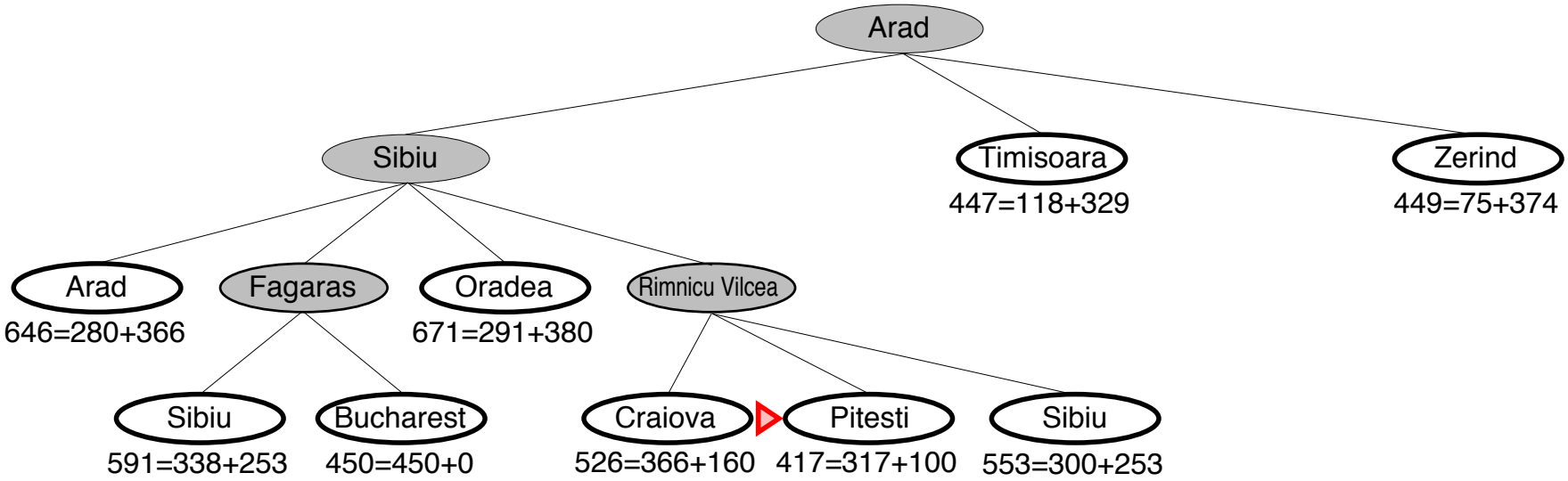
Esempio di ricerca A*



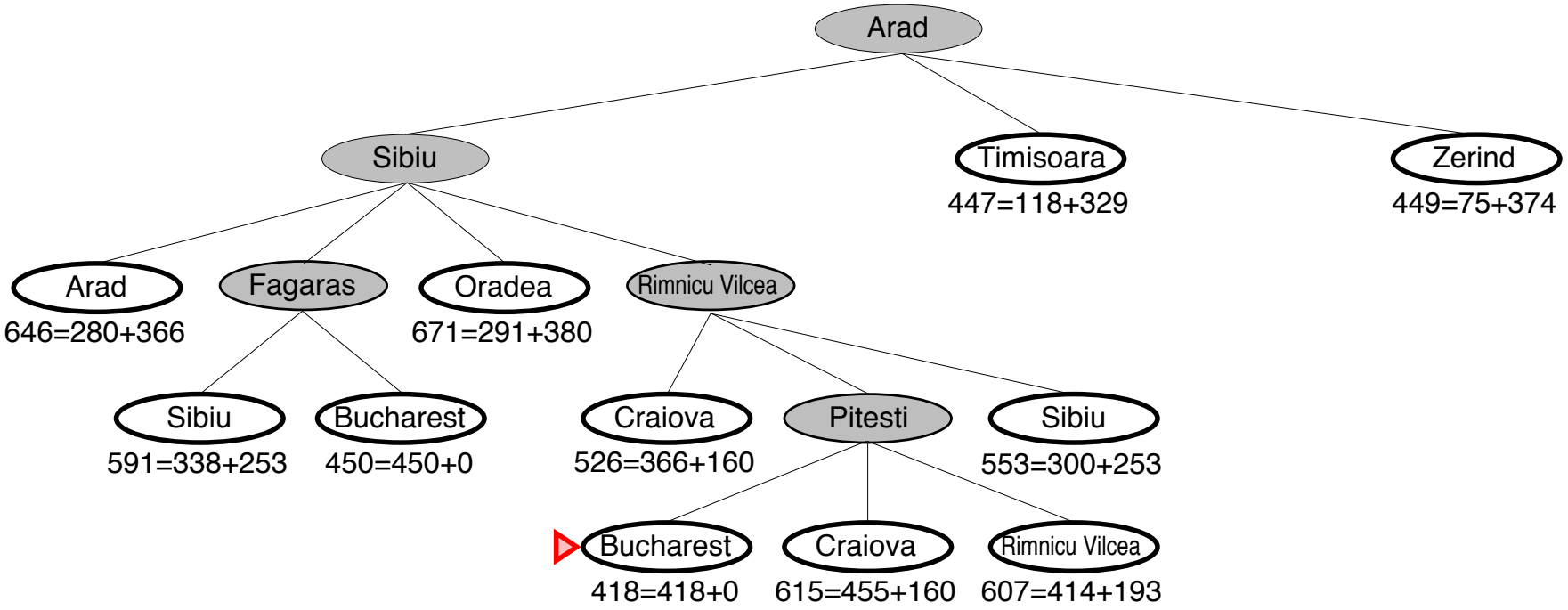
Esempio di ricerca A*



Esempio di ricerca A*

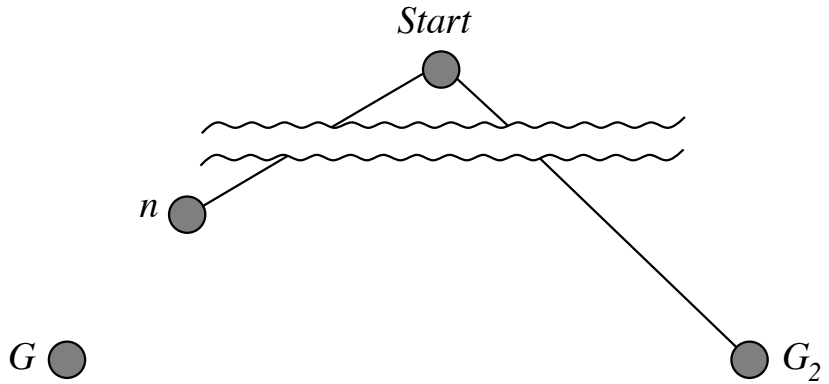


Esempio di ricerca A*



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

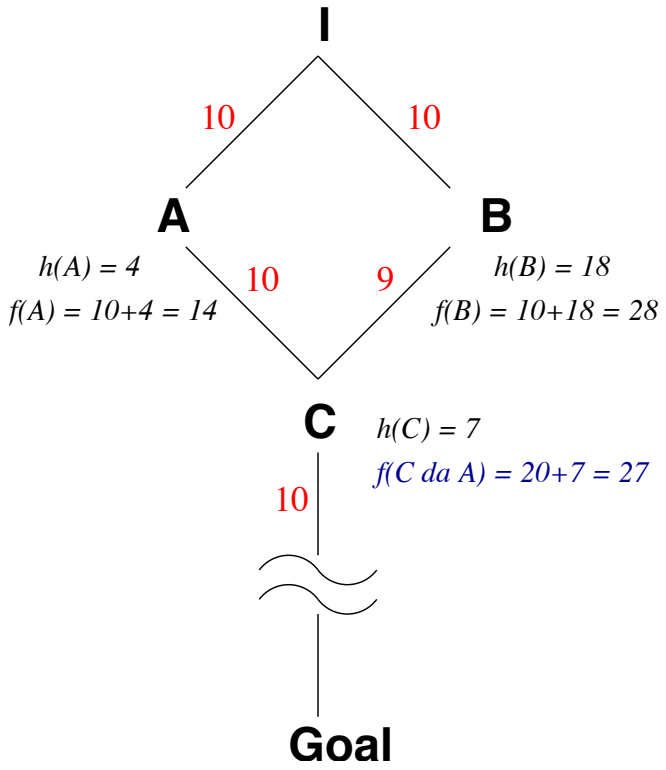
Notare che se si considera la versione dell'algoritmo di ricerca che evita di visitare più volte lo stesso stato (ricerca su grafo), allora la prova vista non funziona! Perchè ?

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:

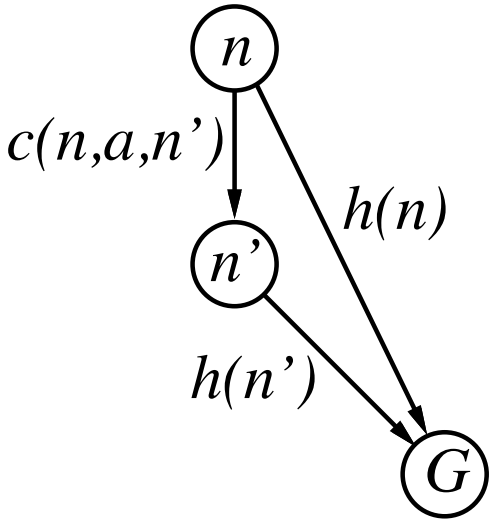
- scartare sempre il cammino più costoso ogni volta che si visita nuovamente uno stesso stato (complicato!)
- usare euristiche *consistenti*



Consistenza

Una euristica è *consistente* se

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



Se h è consistente, si ha

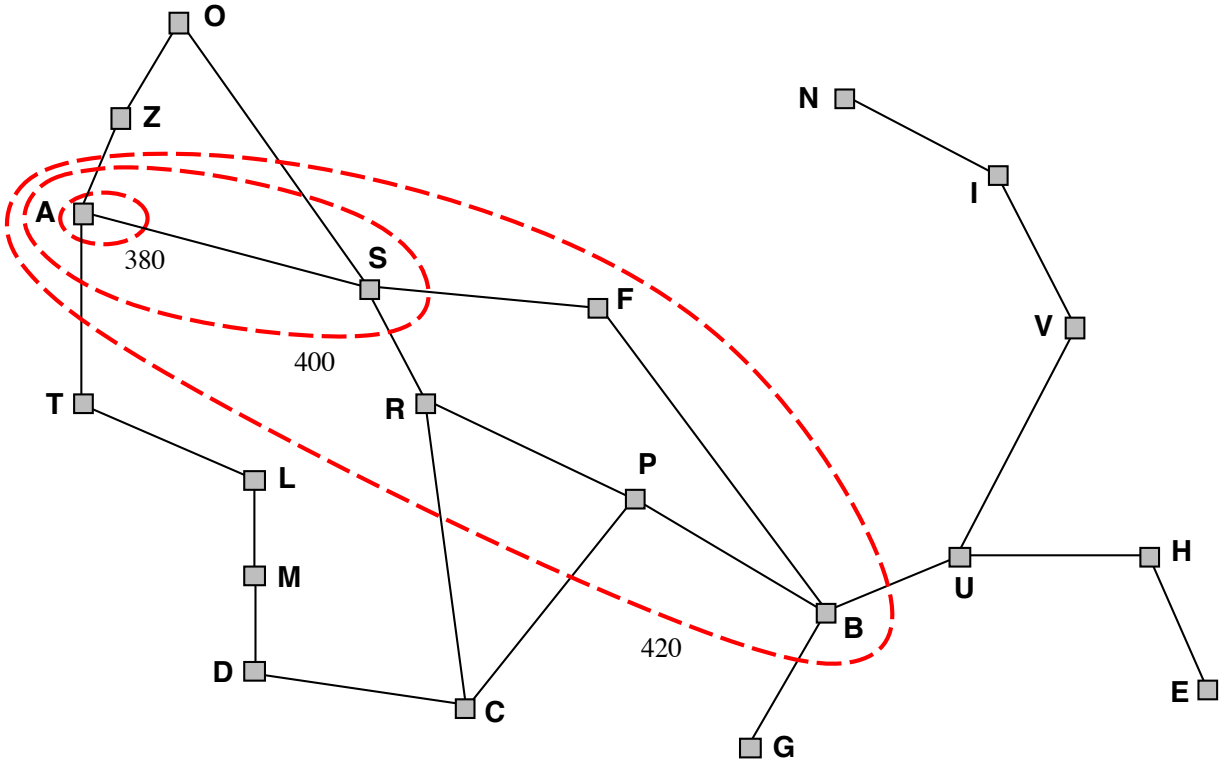
$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

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

Tempo??

Proprietà di A*

Completa?? Si, 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?? Si, 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?? Si, 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?? Si—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:

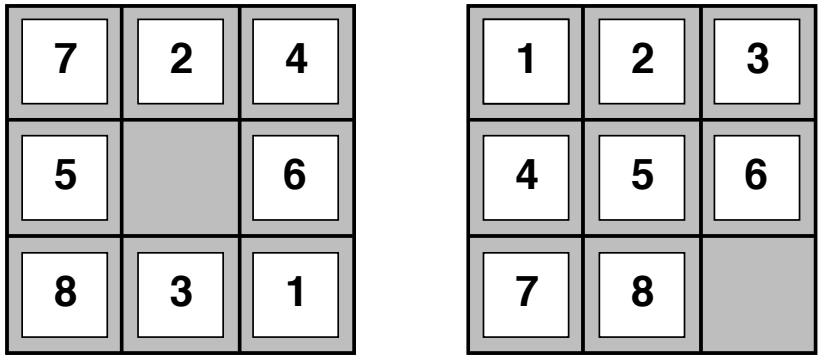
non esiste altro algoritmo ottimo che sicuramente espande meno nodi di A* (se non per risolvere "patte" sui nodi con f uguale al valore ottimo)

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)

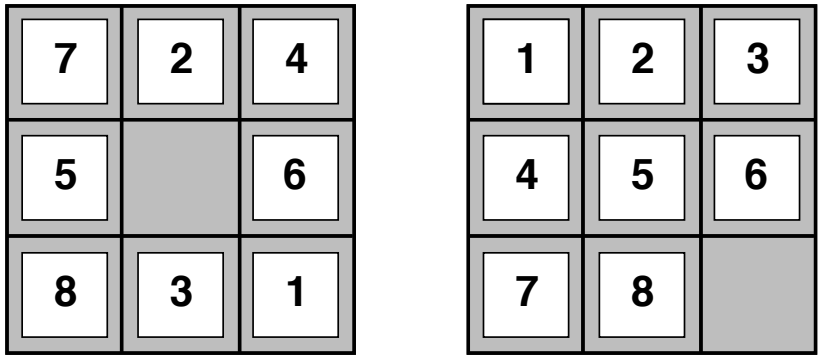


$h_1(S) =$??
$h_2(S) =$??

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)



$h_1(S) =$?? 6
$h_2(S) =$?? 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

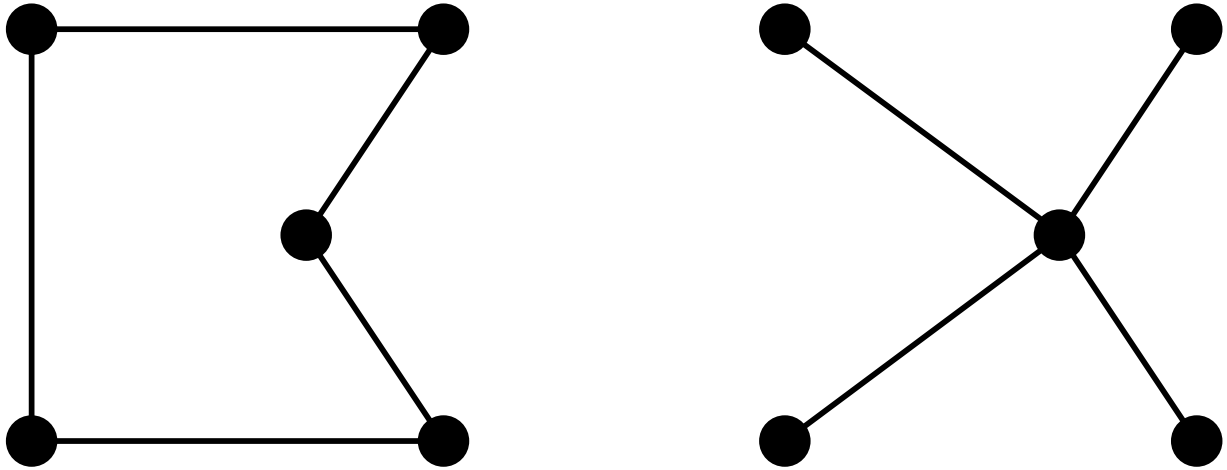
Se le regole di 8-puzzle sono rilassate così che un tassello può muoversi *ovunque*, allora $h_1(n)$ corrisponde alla soluzione sul cammino più breve

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 indiatro al cammino alternativo

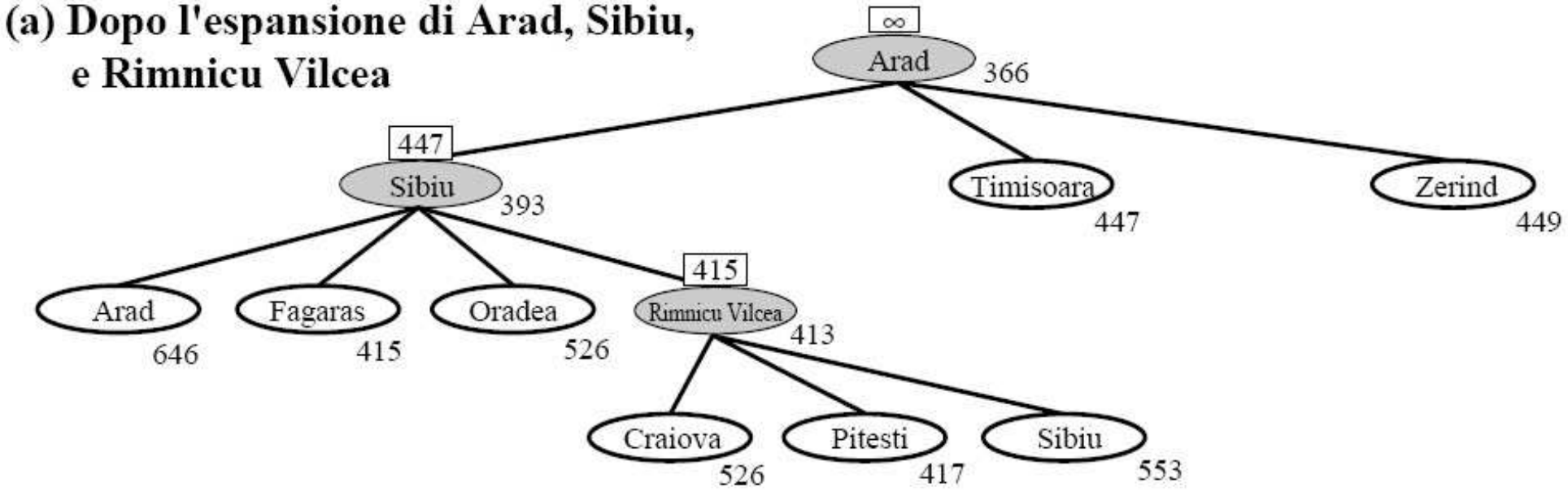
durante il ritorno, si sostituisce l'f-valore di ogni nodo lungo il cammino con il miglior f-valore dei suoi nodi figli

Ricerca Best First Ricorsiva

function RICERCA-BEST-FIRST-RICORSIVA(*problema*) **returns** una soluzione, o fallimento
    RBFS(*problema*, CREA-NODO(STATO-INIZIALE[*problema*]), ∞)

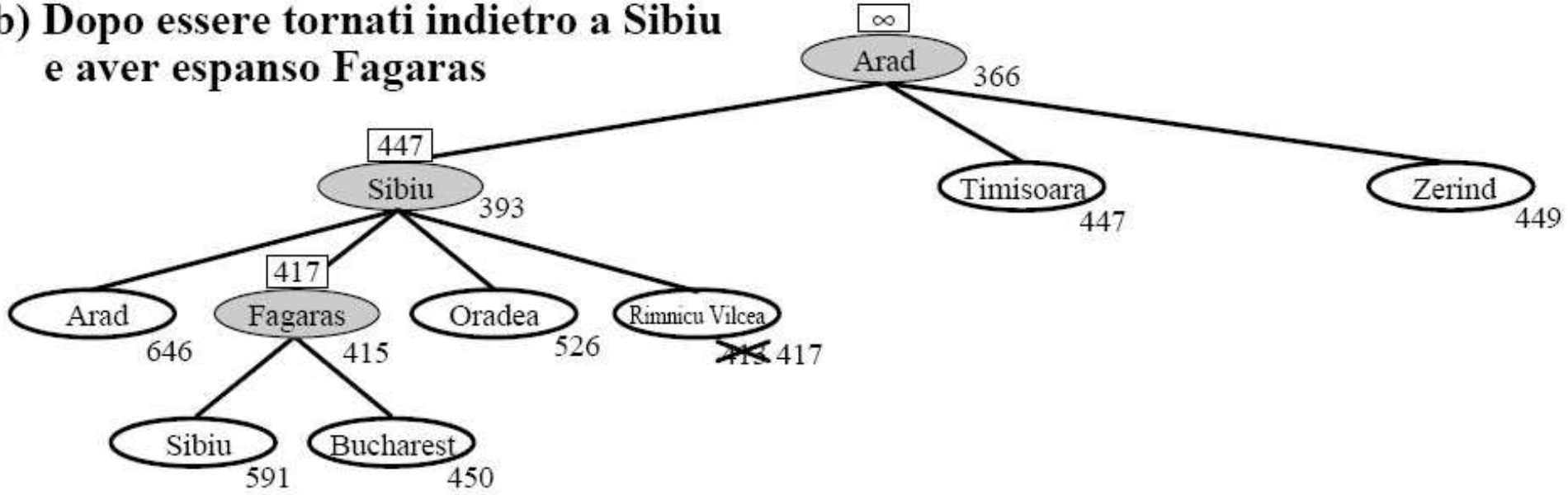
function RBFS(*problema*, *nodo*, *f-limite*) **returns** una soluzione, o fallimento e un nuovo limite all' f-costo
    if TEST-OBIETTIVO[*problema*](STATO[*nodo*]) **then return** SOLUZIONE(*nodo*)
    successori $\leftarrow$ ESPANDI(*nodo*, *problema*)
    if *successori* è vuoto **then return** *fallimento*, ∞
    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

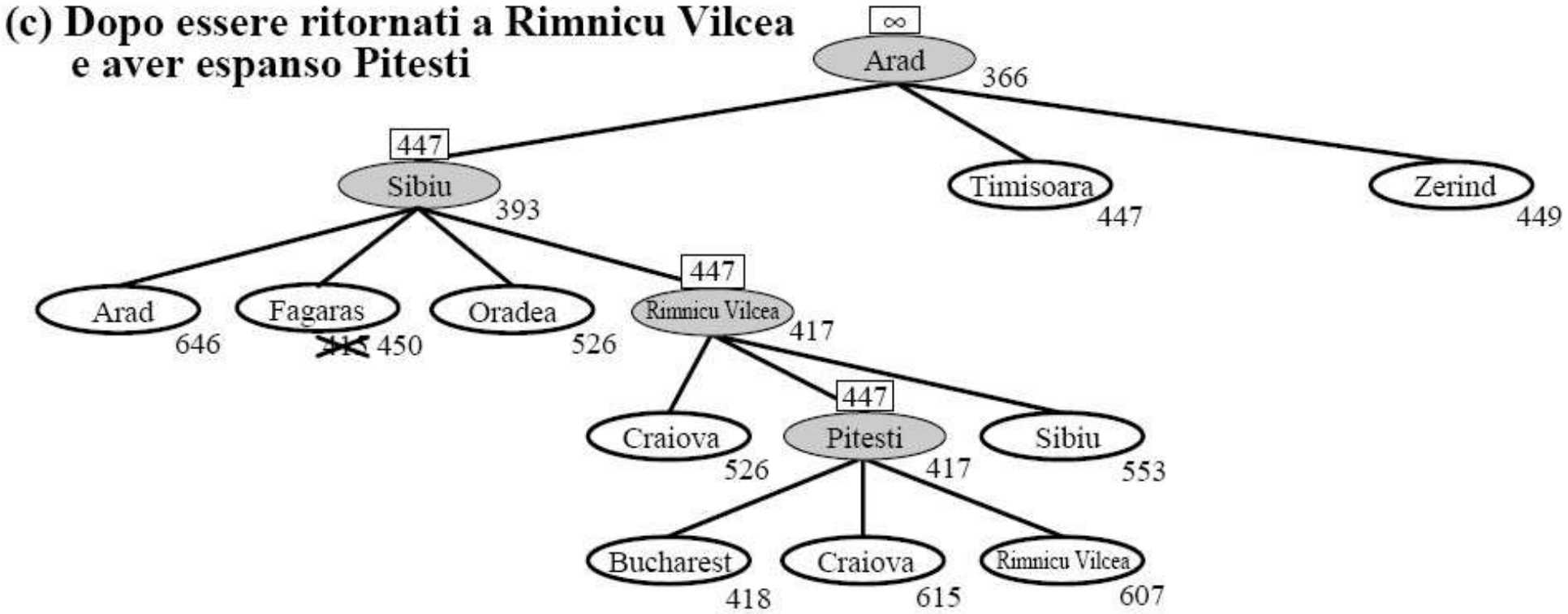


Esempio esecuzione

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



Esempio esecuzione



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

Algoritmi di miglioramento iterativo

In molti problemi di ottimizzazione, *il cammino* è irrilevante;
la soluzione è costituita dallo stato goal stesso

Quindi lo spazio degli stati è dato dall'insieme delle
configurazioni "complete";

trovare una configurazione *ottima*, es.: TSP

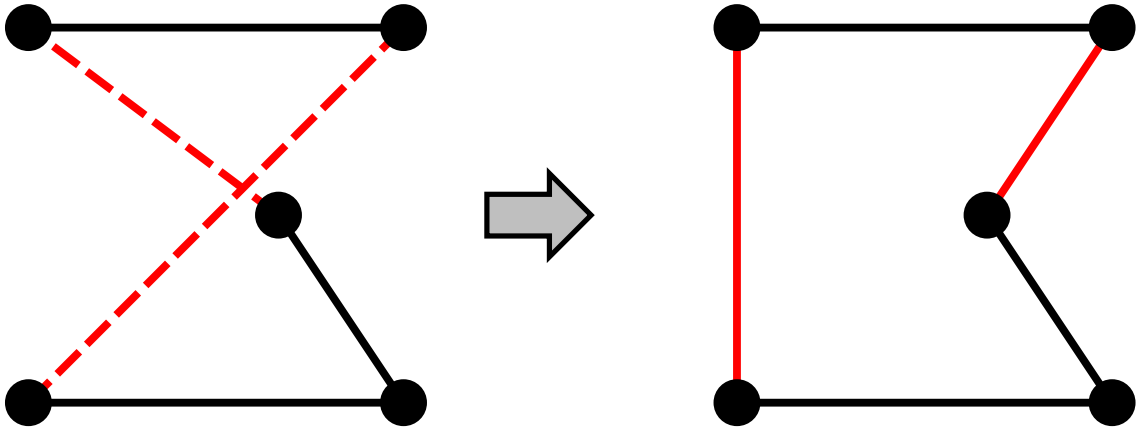
o, trovare una configurazione che soddisfi dei vincoli, es.: orario

In tali casi, si possono usare gli algoritmi di *miglioramento iterativo*;
Mantengono un singolo stato "corrente", e tentano di migliorarlo

Impiegano spazio costante, e sono quindi adatti sia per ricerca online che per ricerca offline

Esempio: Problema del commesso viaggiatore

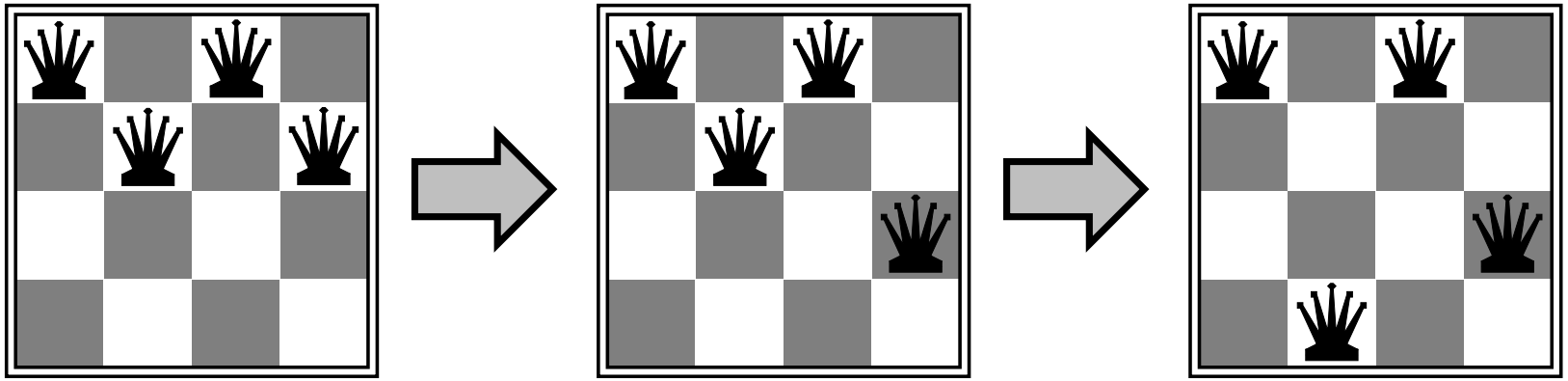
Si parte con un percorso qualsiasi, e si eseguono scambi a coppie



Esempio: n-regine

Disporre n regine su una scacchiera $n \times n$ senza che si minaccino
(non ci devono essere due regine sulla stessa riga, colonna, o diagonale)

Muovere una regina in modo da minimizzare il numero di minacce



Hill-climbing (o discesa/ascesa di gradiente)

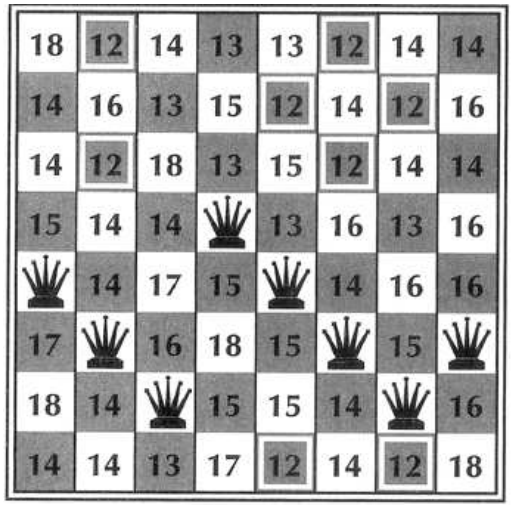
```
function HILL-CLIMBING(problema) returns uno stato che è un massimo locale
    inputs: problema, un problema
    variabili locali: nodo_corrente, un nodo
                      vicino, un nodo

    nodo_corrente ← CREA-NODO(STATO-INIZIALE[problema])
    loop do
        vicino ← il successore di nodo_corrente di valore più alto
        if VALORE[vicino] ≤ VALORE[nodo_corrente] then return STATO[nodo_corrente]
        nodo_corrente ← vicino
```

Hill-climbing: esempio delle 8 regine

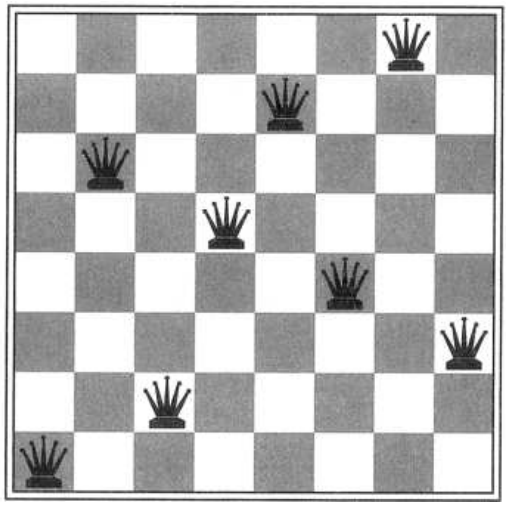
$h(s) =$ *numero di coppie di regine che si attaccano a vicenda*

Quanti successori per ogni stato ? 8×7

(# regine × caselle libere su colonna/riga [deve esserci 1 regina per colonna/riga])



Uno stato con $h = 17$ e valori di h per ogni successore indicati

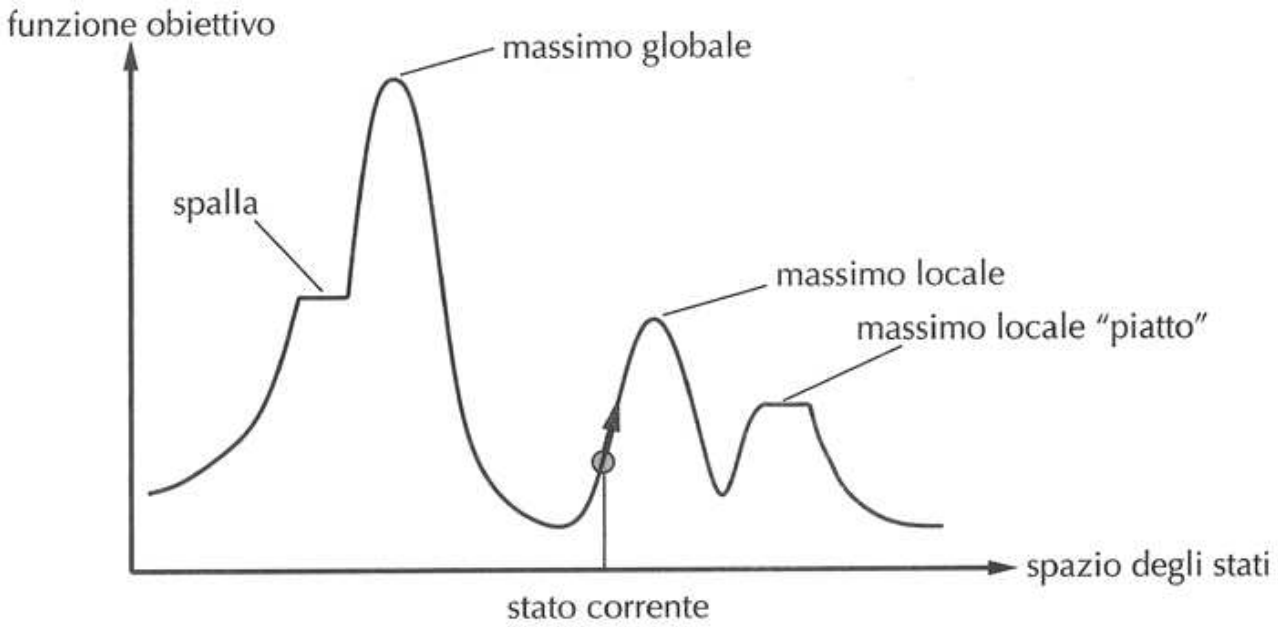


Un minimo locale con $h = 1$. Ogni successore di questo stato ha $h > 1$.

Lo stato a destra "dista" solo 5 passi da quello a sinistra

Hill-climbing: massimi locali ed altri problemi

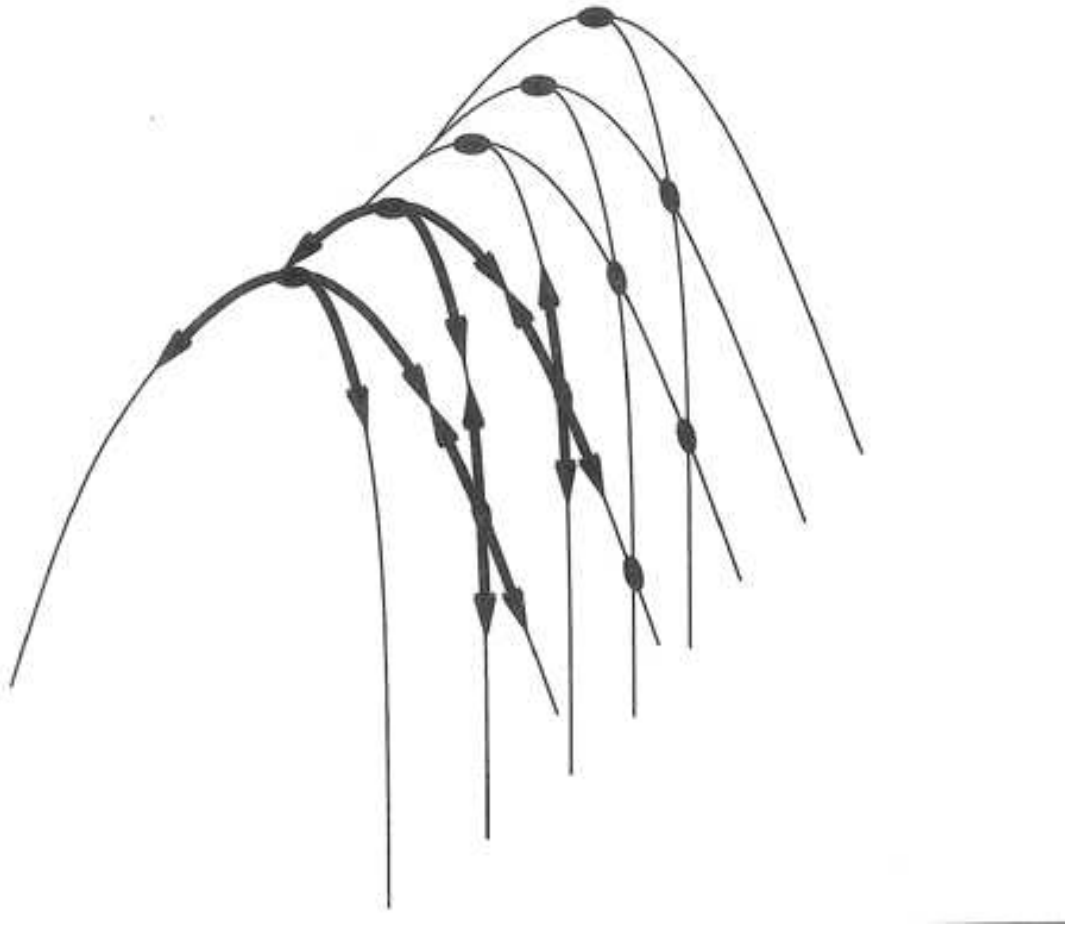
Problemi: a seconda dello stato iniziale, può fermarsi su dei massimi locali e/o "spalle"



Nel caso di spazi continui è difficile scegliere la dimensione del passo di progressione ed inoltre si può avere una convergenza molto lenta

Hill-climbing: cresta (*ridge*)

Sequenza di massimi locali (a valore crescente muovendosi verso l''interno" della figura) molto difficili da esplorare per Hill-climbing



Hill-climbing: alcune soluzioni

Possibili soluzioni:

- plateau (h piatta): "mossa laterale", cioè ci si sposta in uno stato con identico valore di h
  - bisogna stare attenti ad evitare cicli, specialmente nel caso di massimi (minimi) locali piatti
  - tipica soluzione: porre un limite massimo al numero consecutivo di mosse laterali
- massimi (minimi) locali: eseguire scelte stocastiche e/o più ricerche da stati iniziali diversi
  - Hill-climbing stocastico: scegliere a caso fra tutte le mosse che migliorano h, eventualmente usando una probabilità di selezione che è proporzionale al miglioramento (convergenza $+$ lenta, ma spesso soluzioni migliori)
  - Hill-climbing con riavvio casuale: esegue più ricerche a partire da stati iniziali diversi (scelti a caso). Se p è la probabilità di trovare una soluzione ottima per una singola ricerca, il numero di ricerche atteso prima di trovare una soluzione ottima è $1/p$

Numero ricerche prima di trovare una soluzione

Perché il numero di ricerche atteso prima di trovare una soluzione ottima è $1/p$??

Numero ricerche prima di trovare una soluzione

Perché il numero di ricerche atteso prima di trovare una soluzione ottima è $1/p$??

Consideriamo le variabili aleatorie binarie $\mathbf{x}_i$ tali che

$$\mathbf{x}_i = \begin{cases} 0 & \text{se la } i\text{-esima ricerca non trova una soluzione ottima} \\ 1 & \text{se la } i\text{-esima ricerca trova una soluzione ottima} \end{cases}$$

Sappiamo che $\forall i$, $P(\mathbf{x}_i = 1) = p$, e $P(\mathbf{x}_i = 0) = 1 - p$

Inoltre le variabili $\mathbf{x}_i$ sono mutuamente indipendenti:
il risultato di una ricerca non influenza il risultato di un'altra ricerca

Abbiamo quindi un Processo di Bernoulli per cui:
il numero di ricerche prima di trovare una soluzione ottima rispetta una distribuzione Geometrica

Numero ricerche prima di trovare una soluzione

Infatti, la probabilità che la k-esima ricerca (su k) sia la prima a trovare una soluzione ottima è

$$P((\sum_{j=1}^{k-1} \mathbf{x}_j = 0) \wedge (\mathbf{x}_k = 1)) = (1-p)^{k-1}p$$

Il valore atteso della quantità che a noi interessa è quindi

$$\begin{aligned}
\sum_{k=1}^{\infty} k(1-p)^{k-1}p &= p\sum_{k=1}^{\infty} k(1-p)^{k-1} \\
&= p\sum_{k=1}^{\infty} \frac{d(1-p)^k}{dp} \\
&= -p\frac{d}{dp}\left(\sum_{k=0}^{\infty}(1-p)^k - 1\right) = -p\frac{d}{dp}\left(\frac{1}{p} - 1\right) \\
&= p\frac{1}{p^2} = \frac{1}{p}
\end{aligned}$$

da cui deriva il risultato

Hill-climbing e le 8 regine

Numero stati: 8^8 (circa 17 milioni)

- Hill-climbing standard
  - soluzione (ottima) trovata il 14% delle volte
  - in media circa 4 passi per trovare una soluzione, altrimenti circa 3 passi in caso di soluzione subottima
- Hill-climbing con mosse lateriali (non più di 100 consecutive)
  - soluzione (ottima) trovata il 94% delle volte
  - in media circa 21 passi per trovare una soluzione, altrimenti circa 64 passi in caso di soluzione subottima
- Hill-climbing con riavvio casuale
  - soluzione (ottima) trovata con probabilità $p = 0,14$, quindi circa 7 ricerche per trovare una soluzione ottima ($(1-p)/p = 6,14$ fallimenti $+ 1$ successo). Numero di passi complessivo atteso: $3(1-p)/p + 4 = 22,43$
  - con mosse laterali: soluzione (ottima) trovata con probabilità $p = 0,94$, quindi circa $1,06$ ricerche per trovare una soluzione ottima ($(1-p)/p = 0,06$ fallimenti $+ 1$ successo). Numero di passi complessivo atteso: $0,06(1-p)/p + 21 = 25,08$

Simulated annealing

Idea: evitare i massimi locali permettendo delle mosse "cattive"
ma gradualmente decrementare la loro grandezza e frequenza

```
function SIMULATED-ANNEALING(problema, raffreddamento) returns uno stato soluzione
    inputs: problema, un problema
            velocità_raffreddamento, una corrispondenza dal tempo alla "temperatura"
    variabili locali: nodo_corrente, un nodo
                      successivo, un nodo
                      T, una "temperatura" che controlla la probabilità
                           di compiere passi verso il basso

    nodo_corrente ← CREA-NODO(STATO-INIZIALE[problema])
    for t ← 1 to ∞ do
        T ← velocità_raffreddamento[t]
        if T = 0 then return nodo_corrente
        successivo ← un successore scelto a caso di nodo_corrente
        ΔE ← VALORE[successivo] – VALORE[nodo_corrente]
        if ΔE > 0 then nodo_corrente ← successivo
        else nodo_corrente ← successivo solo con probabilità e^(ΔE/T)
```

Proprietà del simulated annealing

A "temperatura" fissata T, la probabilità di occupazione degli stati raggiunge la distribuzione di Boltzmann

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T diminuito abbastanza lentamente $\Longrightarrow$ si raggiunge sempre lo stato migliore (Metropolis et al., 1953, per problemi di modellazione di processi fisici)

Ampiamente usato in applicazioni pratiche, come la progettazione di circuiti VLSI e la definizione degli orari dei voli delle linee aeree