Algoritmi di ricerca informati

Capitolo 4, sezioni 1–2, 4

Sommario

◊ Ricerca best-first

◊ Ricerca A*

◊ Euristiche

◊ Hill-climbing

◊ Simulated annealing

Ripasso: ricerca generale

```
function GENERAL-SEARCH(problem, QUEUING-FN) returns a solution, or failure
   nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
   loop do
       if nodes is empty then return failure
       node ← REMOVE-FRONT(nodes)
       if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
       nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
   end
```

Una strategia è definita scegliendo *l'ordine di espansione dei nodi*

Ricerca Best-first

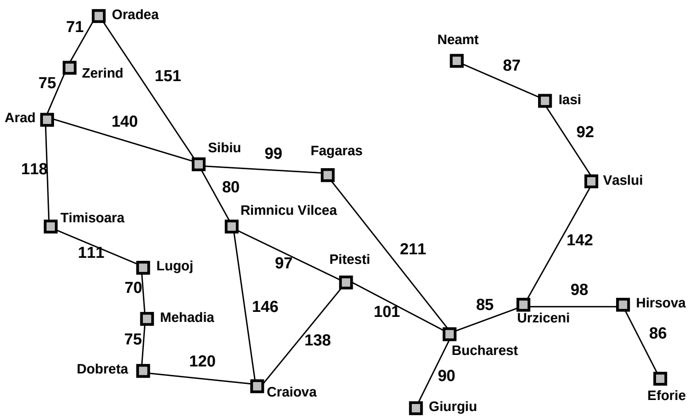
Idea: usare una *funzione di valutazione* per ogni nodo
– stima di "desiderabilità"

⇒ Espandere il nodo non espanso più desiderabile

Implementazione:
QueueingFn = inserire i successori in ordine decrescente di 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	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

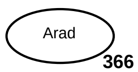
Ricerca greedy

Funzione di valutazione $h(n)$ (euristica)
$= $ stima del costo dal nodo n al *goal*

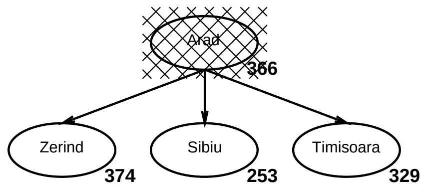
E.g., $h_{\mathrm{SLD}}(n) =$ distanza in linea d'aria da n a Bucharest

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

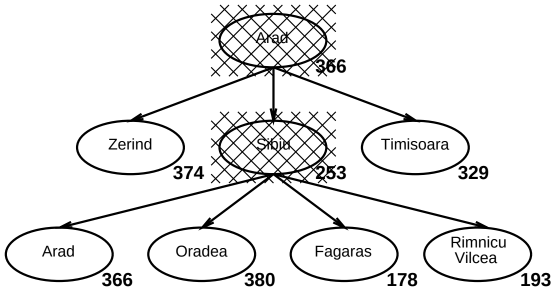
Esempio di ricerca greedy



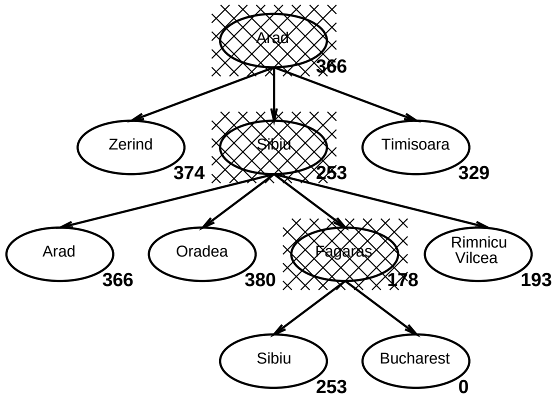
Esempio di ricerca greedy



Esempio di ricerca greedy



Esempio di ricerca greedy



Proprietà della ricerca greedy

Completa?? No – può restare intrappolata in cicli, per es.
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 gia' costosi

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

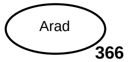
$g(n)$ = costo gia' avuto per raggiungere n
$h(n)$ = costo stimato da n al goal
$f(n)$ = costo totale stimato del cammino attraverso n al goal

La ricerca A* usa un'euristica *ammissibile*
cioe': $h(n) \leq h^*(n)$ dove $h^*(n)$ e' il costo *vero* da n.

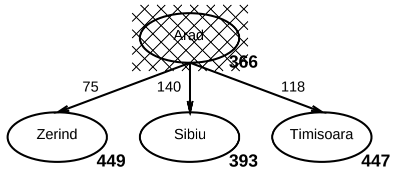
Es.: $h_{\text{SLD}}(n)$ non sovrastima mai la reale distanza

Teorema: la ricerca A* e' ottima

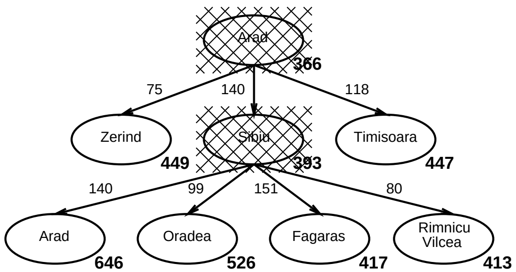
Esempio di ricerca A*



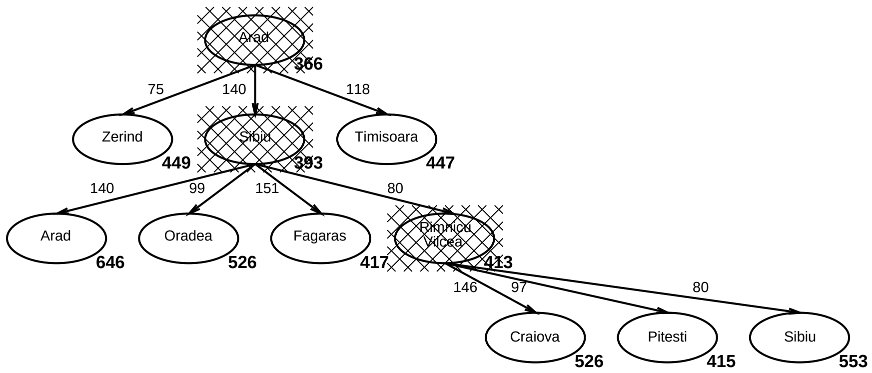
Esempio di ricerca A*



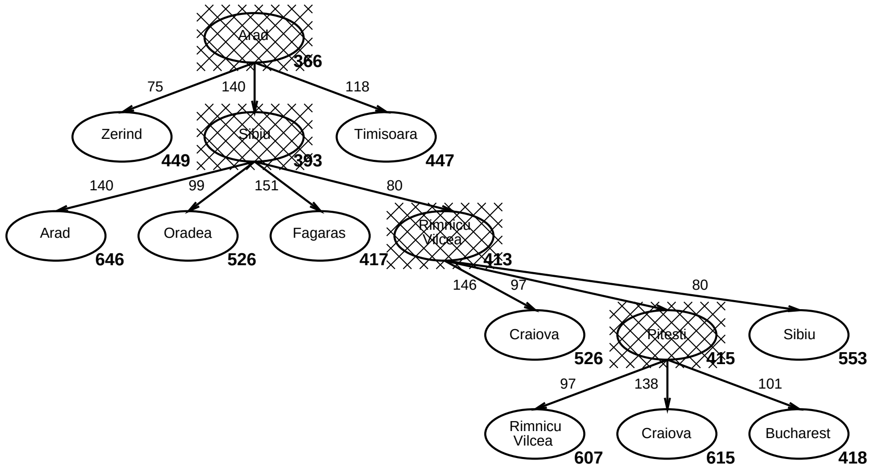
Esempio di ricerca A*



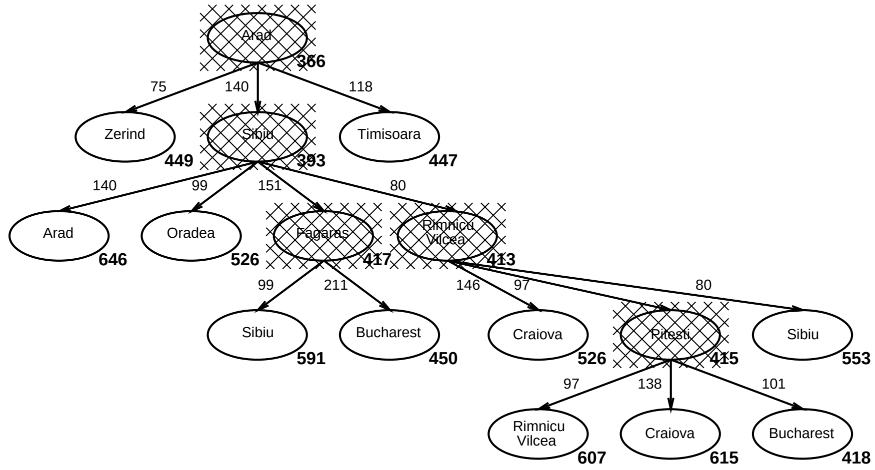
Esempio di ricerca A*



Esempio di ricerca A*

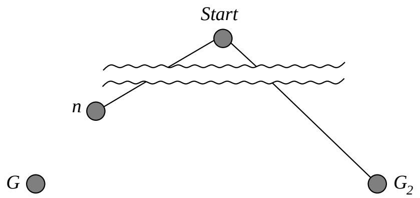


Esempio di ricerca A*



Ottimalita' di A*

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



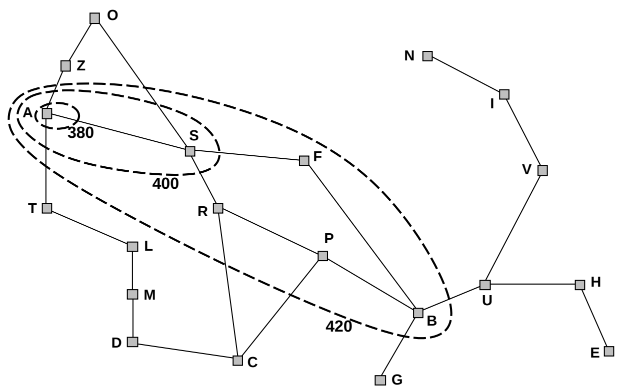
$$
\begin{aligned}
f(G_2) &= g(G_2) && \text{dato che } h(G_2) = 0 \\
&> g(G) && \text{dato che } G_2 \text{ e' sub} - \text{ottimo} \\
&\geq f(n) && \text{dato che } h \text{ e' ammissibile}
\end{aligned}
$$

Dato che $f(G_2) > f(n)$, A* non selezionera' mai G_2 per espanderlo

Ottimalita' di A* (piu' utile)

<u>Lemma</u>: A* espande i nodi in ordine di valore di f

Gradualmente, aggiunge dei "contorni di f" dei nodi
Il contorno i ha tutti i nodi con $f = f_i$, dove $f_i < f_{i+1}$



Proprieta' 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.]

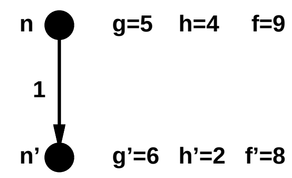
Spazio?? Mantiene tutti i nodi in memoria

Ottima?? Si—non puo' espandere f_{i+1} finche' f_i non e' finita

Prova del lemma: Pathmax

Per alcune euristiche ammissibili, f puo' *decrementare* lungo un cammino

Es.: supponiamo che n' sia un successore di n



Ma questo getta via dell'informazione!
$f(n) = 9 \Rightarrow$ il vero costo di un cammino che passa da n e' ≥ 9
Quindi anche il vero costo di un cammino che passa da n' e' ≥ 9

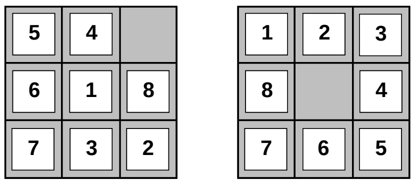
Modifica pathmax di A*:
Invece di $f(n') = g(n') + h(n')$, usare $f(n') = max(g(n') + h(n'), f(n))$

Con pathmax, f e' sempre non-decrescente lungo un qualsiasi cammino

Euristiche ammissibili

Per es., per 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)



Start State Goal State

$h_1(S) =$?? 7
$h_2(S) =$?? 2+3+3+2+4+2+0+2 = 18

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 = troppi 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 cosi' 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 cosi' che un tassello può muoversi *ad ogni "quadrato" adiacente*, allora $h_2(n)$ corrisponde alla soluzione sul cammino più breve

Algoritmi di miglioramento iterativo

In molti problemi di ottimizzazione, il *cammino* e' irrilevante;
la soluzione e' lo stato di goal stesso

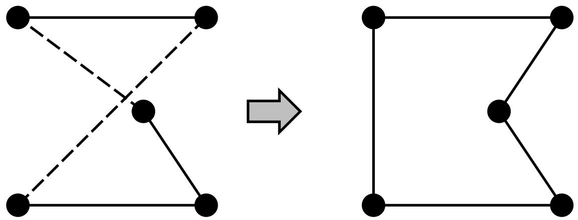
Allora lo spazio degli stati e' l'insieme delle configurazioni "complete";
trova una configurazione *ottima*, es.: TSP
o trova una configurazione che soddisfa dei vincoli, es.: n-regine

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

Spazio costante

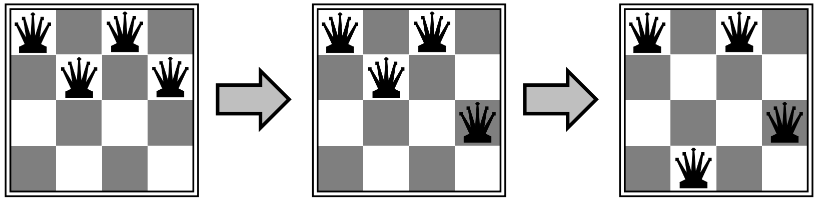
Esempio: Problema del commesso viaggiatore

Trova il cammino piu' breve che visita ogni citta' esattamente una volta



Esempio: n-regine

Mettere n regine su una scacchiera $n \times n$ senza che si attacchino
(senza che ci siano due regine nella stessa riga, colonna, o diagonale)



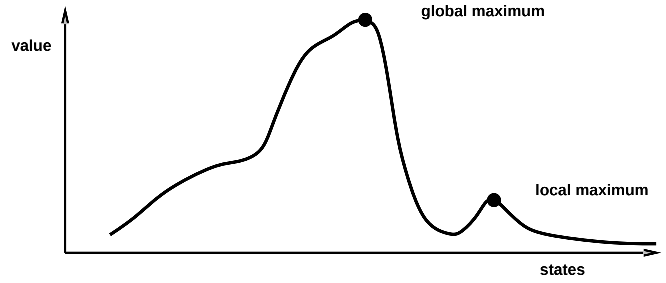
Hill-climbing (o discesa/ascesa di gradiente)

function Hill-Climbing(*problem*) **returns** a solution state
  inputs: *problem*, a problem
  local variables: *current*, a node
    next, a node

  current ← Make-Node(Initial-State[*problem*])
  loop do
    next ← a highest-valued successor of *current*
    if Value[next] < Value[current] **then return** *current*
    current ← *next*
  end

Hill-climbing

Problema: a seconda dello stato iniziale, puo' fermarsi su dei massimi locali



Simulated annealing

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

function Simulated-Annealing(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling the probability of downward steps

  current ← Make-Node(Initial-State[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[t]
    if T=0 **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ Value[*next*] – Value[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Se T e' alta, le mosse "cattive" sono piu' probabili.
T scende a mano a mano che il tempo passa, secondo schedule.