Ricerca in uno spazio di soluzioni

Outline

♢ Formulazione del problema

♢ Alcuni algoritmi di ricerca

Esempio: Romania

In vacanza in Romania; ora ad Arad.
Il volo parte domani da Bucharest

Formulare il goal:

essere a Bucharest

Formulate il problema:

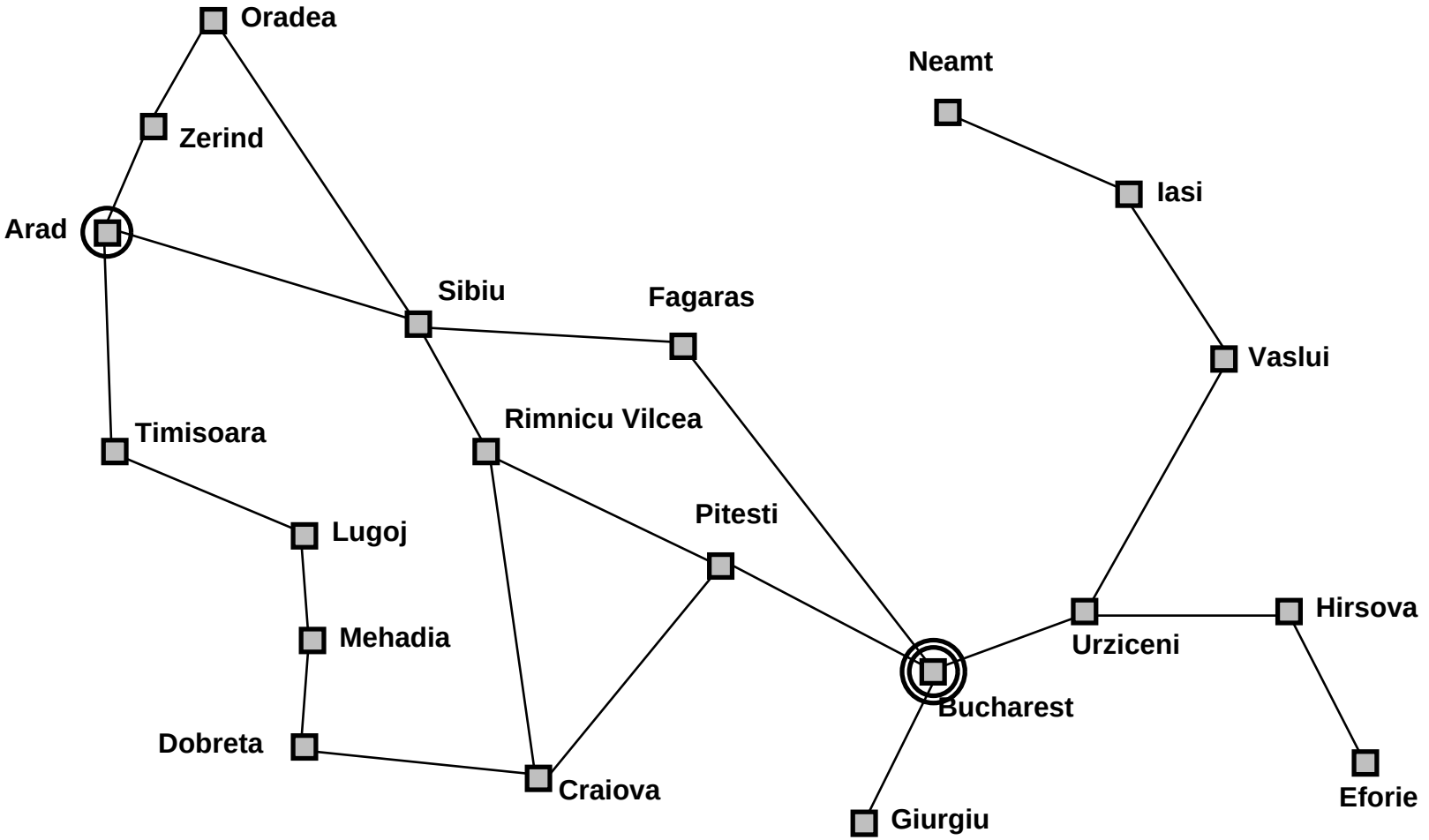
stati: varie citta'

operatori: viaggi tra le citta'

Tovare una soluzione:

sequenza di citta', esempio: Arad, Sibiu, Fagaras, Bucharest

Esempio: Romania



Formulazione del problema (singoli stati)

Un *problema* e' definito da quattro elementi:

stato iniziale es.: "ad Arad"

operatori (o *funzione successore* $S(x)$)
es.: Arad $\rightarrow$ Zerind Arad $\rightarrow$ Sibiu etc.

Un test per il goal, puo' essere
esplicito, es.: $x =$ "a Bucharest"
implicito, es.: Bucharest(x)

costo di un cammino (additivo)
es.: somma di distanze, numbero di operatori eseguiti, ...

Una *soluzione* e' una sequenza di operatori
che porta dallo stato iniziale ad uno stato di goal

Selezionare uno spazio degli stati

Il mondo reale e' molto complesso

$\Rightarrow$ lo spazio degli stati deve essere $astratto$ per risolvere il problema

Stato (astratto) = insieme di stati reali

Operatore (astratto) = combinazione complessa di azioni reali

es.: "Arad $\rightarrow$ Zerind" rappresenta un insieme complesso di possibili strade, detour, fermate, ...

Per garantire la realizzabilita', qualsiasi stato reale "in Arad" deve portare a $qualche$ stato reale "in Zerind"

Soluzione (astratta) =

insieme di cammini reali che sono soluzioni in quel mondo reale

Ogni azione astratta dovrebbe essere "piu' semplice" del problema originale.

Algoritmi di ricerca

Idea di base:

offline, esplorazione simulata dello spazio degli stati
generando successori di stati gia' esplorati
(cioe' *espandendo* stati)

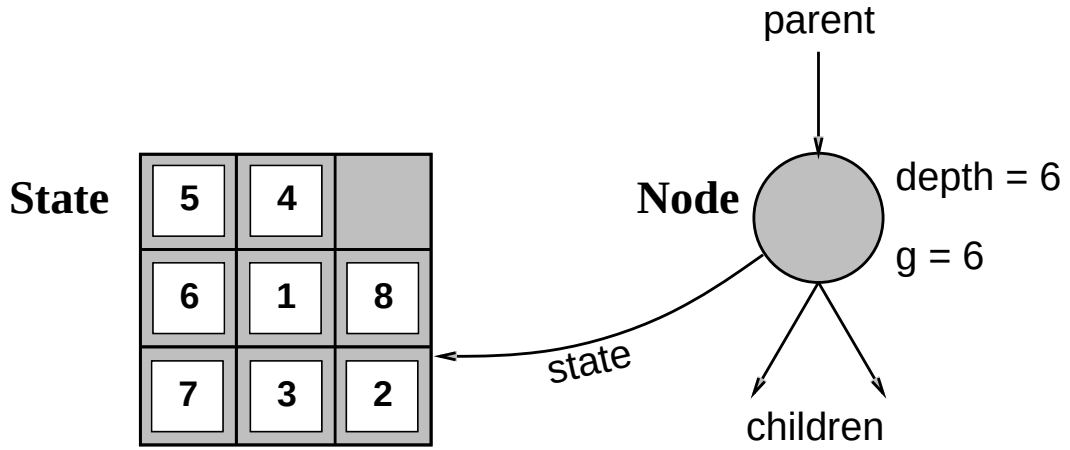
```
function GENERAL-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

Implementazione degli algoritmi di ricerca

```
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
```

Implementazione: stati vs. nodi

Uno *stato* e' una rappresentazione di una configurazione fisica
Un *nodo* e' una struttura dati che fa parte di un albero di ricerca
include *genitori*, *figli*, *profondita'*, *costo del cammino* $g(x)$
Stati non hanno genitori, figli, ...



La funzione EXPAND crea nuovi nodi, riempiendo i vari campi e usando gli OPERATORS (o SUCCESSORFN) del problema per creare gli stati corrispondenti.

Strategie di ricerca

Una strategia e' definita scegliendo un *ordine di espansione dei nodi*

Le strategie sono valutate secondo le seguenti dimensioni:

- completezza—trova sempre una soluzione se ne esiste una?
- complessita' in tempo—numero di nodi generati/espansi
- complessita' in spazio—massimo numero di nodi in memoria
- ottimalita'—trova sempre una soluzione di costo minimo?

La complessita' in spazio e tempo sono misurate in termini di

- b—massimo fattore di branching dell'albero di ricerca
- d—profondita' della soluzione di costo minimo
- m—massima profondita' dello spazio degli stati (puo' essere ∞)

Strategie di ricerca non informate

Le strategie *non informate* usano solo l'informazione disponibile nella definizione del problema

Ricerca breath-first

Ricerca con costo uniforme

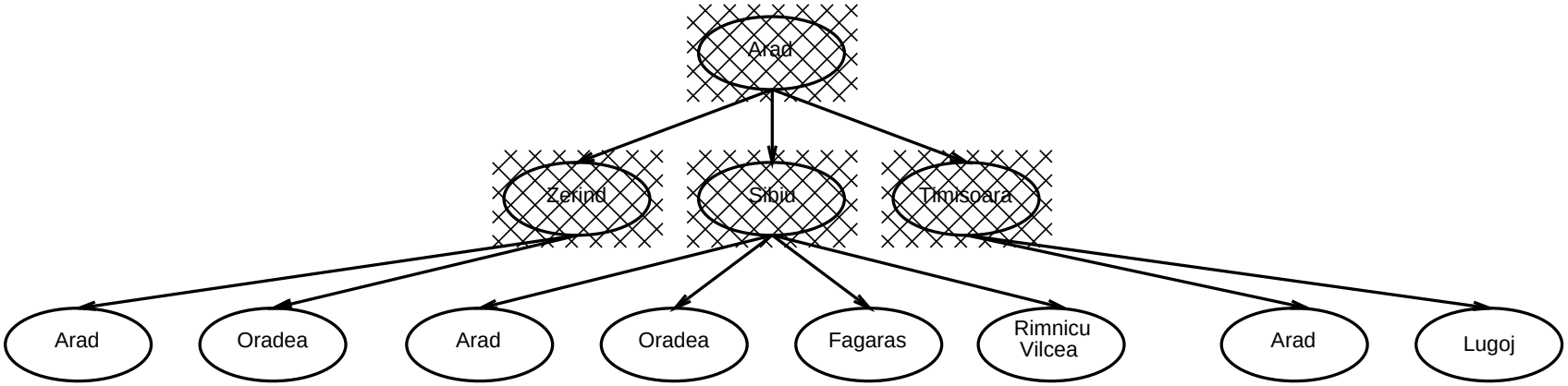
Ricerca depth-first

Ricerca depth-limited

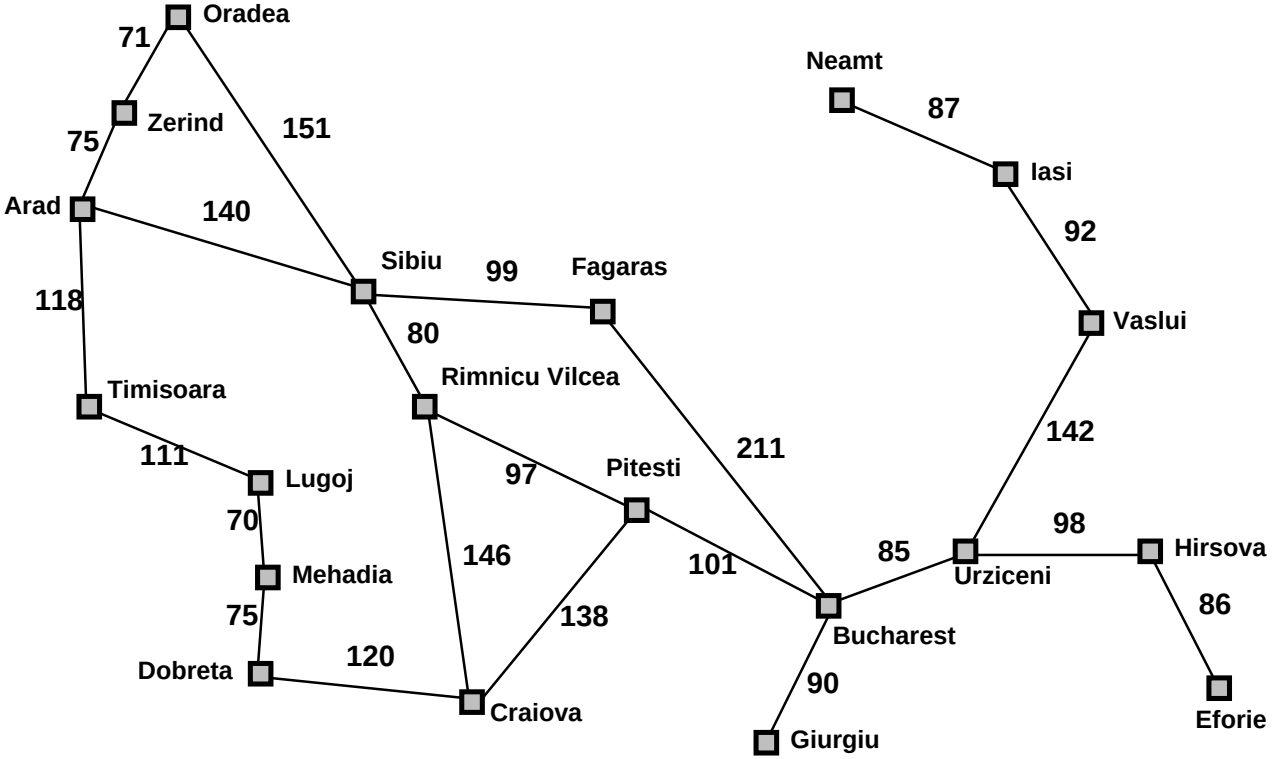
Ricerca iterative deepening

Ricerca breadth-first

Espande sempre il nodo meno profondo



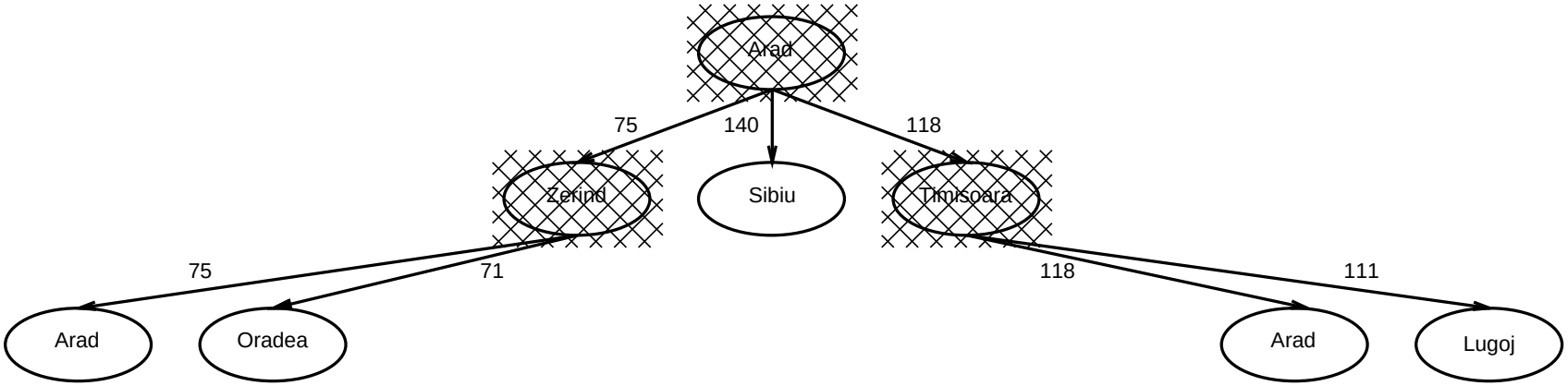
Romania con costo dei passi in Km



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

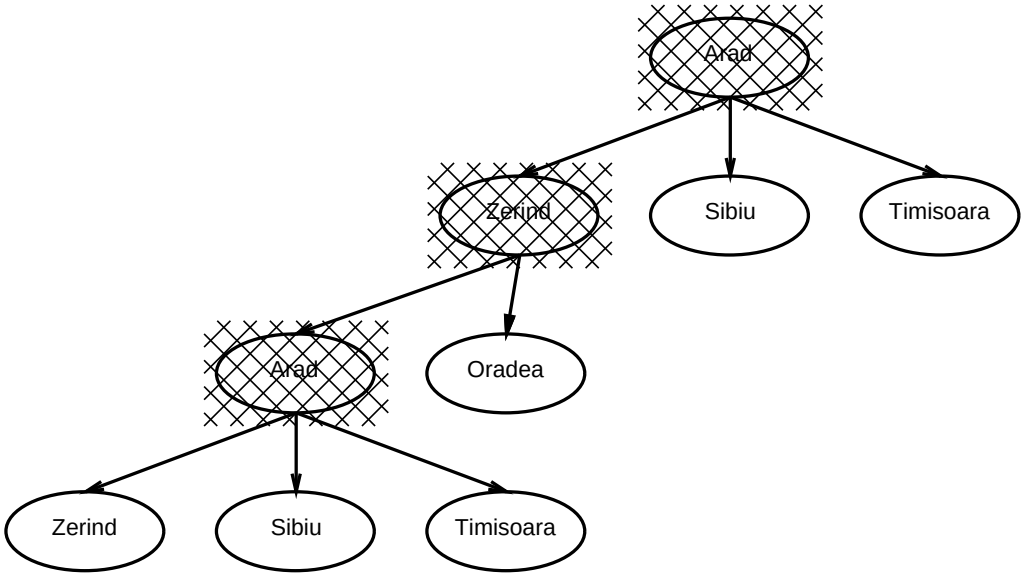
Ricerca a costo uniforme

Espande il nodo con costo minimo



Ricerca depth-first

Espande il nodo piu' profondo



Nota: puo' avere dei cammini infiniti $\rightarrow$ ha bisogno di uno spazio di ricerca finito e non ciclico (o si deve controllare che gli stati non si ripetano)

Ricerca depth-limited

= ricerca depth-first con profondita' limitata (a l)

Implementazione:

Nodi a profondita' l non hanno successori

Completa se l $\geq$ d.
Non ottima.
Tempo: $O(b^l)$
Spazio: $O(b \times l)$

Ricerca iterative deepening

Prova tutti i possibili limiti di profondita'.

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution sequence
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

Confronto tra le strategie

Criterio	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Tempo	b^d	b^d	b^m	b^l	b^d
Spazio	b^d	b^d	bm	bl	bd
Ottima ?	si	si	no	no	si
Completa ?	si	si	no	si, se $l \geq d$	si