Agenti basati su goal (*goal-based*) ⇒problem-solving agent

- Formulazione del problema
- Esempio di problema
- Alcune strategie di ricerca

function SIMPLE-PROBLEM-SOLVING-AGENT(*percept*) **returns** an action
  persistent: *seq*, an action sequence, initially empty
    state, some description of the current world state
    goal, a goal, initially null
    problem, a problem formulation

  state ← UPDATE-STATE(*state*, *percept*)
  if *seq* is empty **then**
    goal ← FORMULATE-GOAL(*state*)
    problem ← FORMULATE-PROBLEM(*state*, *goal*)
    seq ← SEARCH(*problem*)
    if *seq* = *failure* **then return** a null action
  action ← FIRST(*seq*)
  seq ← REST(*seq*)
  return *action*

Formulazione del problema

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

Formulare il goal:

essere a Bucharest

Formulare il problema:

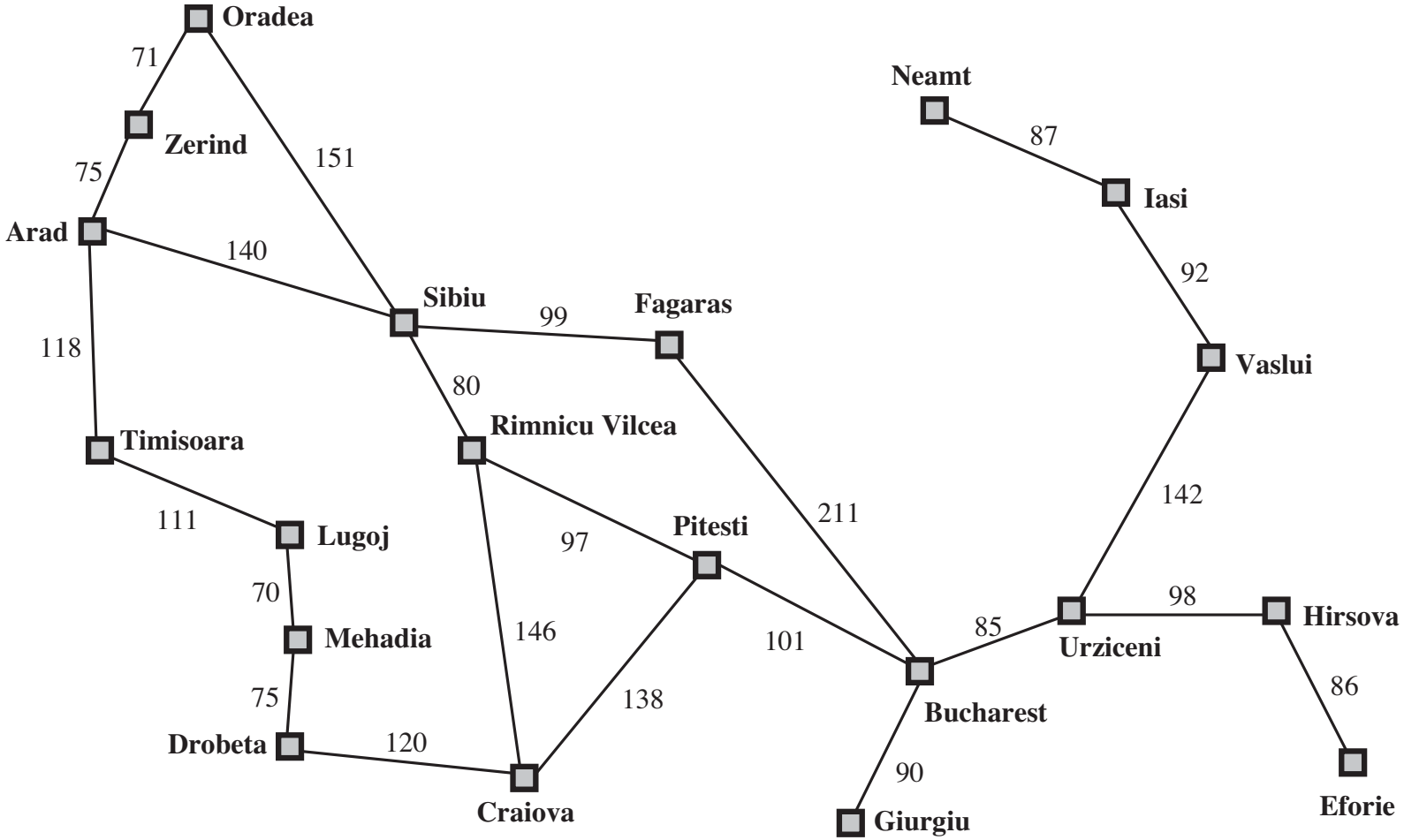
stati: varie città

azioni: viaggi fra città

Trovare una soluzione:

sequenza di città, esempio: Arad, Sibiu, Fagaras, Bucharest

Esempio di problema: osservabile, discreto, deterministico



Formulazione del problema "single-state"

Un *problema* è definito da quattro elementi:

stato iniziale es., "ad Arad"

funzione successore $S(x)$ = insieme di coppie azione–stato
es., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind\rangle, \ldots\}$

test per il goal, può essere
esplicito, es., $x =$ "a Bucharest"
implicito, es., $Bucharest(x)$

costo di un cammino (additivo)
es., somma di distanze, numero di azioni eseguite, etc.
$c(x, a, y)$ è il costo del singolo passo, assunto essere ≥ 0

Una *soluzione* è una sequenza di azioni
che conduce dallo stato iniziale ad uno stato di goal

Alternativa alla funzione successore

Formulazione alternativa per la funzione successore: definire separatamente

- le azioni eseguibili in un dato stato
- *transition model*: l'effetto ottenuto dalla applicazione di ogni azione (che potrebbe essere sconosciuto)

Selezionare uno spazio degli stati

Il mondo reale è molto complesso
$\Rightarrow$ lo spazio degli stati deve essere *astratto* per risolvere il problema

Stato (astratto) = insieme di stati reali

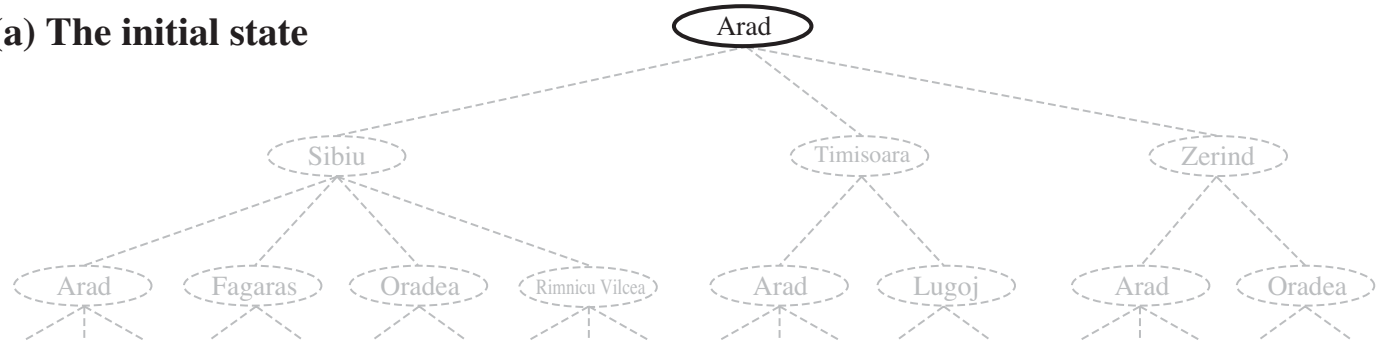
Azione (astratta) = combinazione complessa di azioni reali
es., "Arad $\rightarrow$ Zerind" rappresenta un insieme complesso
di possibili strade, detour, fermate, etc.
Per garantire la realizzabilità, qualsiasi stato reale "in Arad"
deve condurre a *qualche* stato reale "in Zerind"

Soluzione (astratta) =
insieme di cammini reali che sono soluzioni nel mondo reale

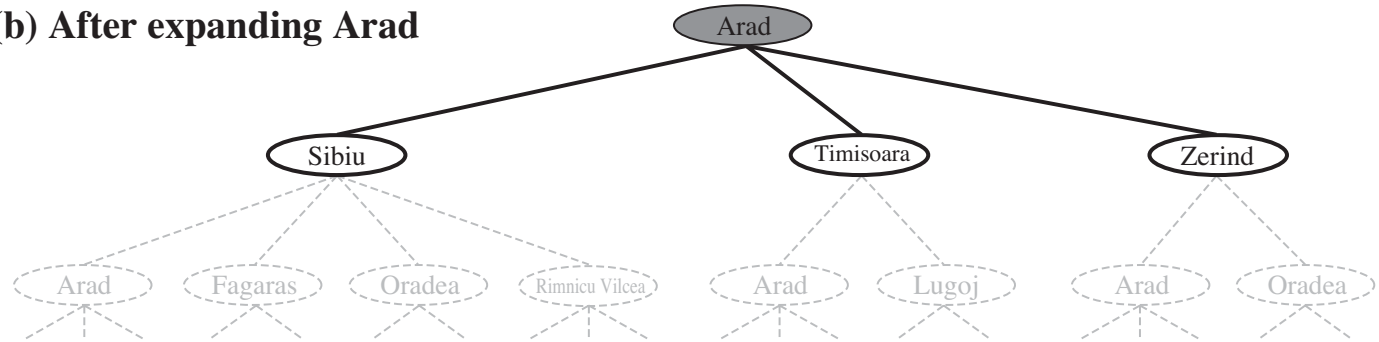
Ogni azione astratta dovrebbe essere "più semplice" del problema originale!

Esempio di ricerca

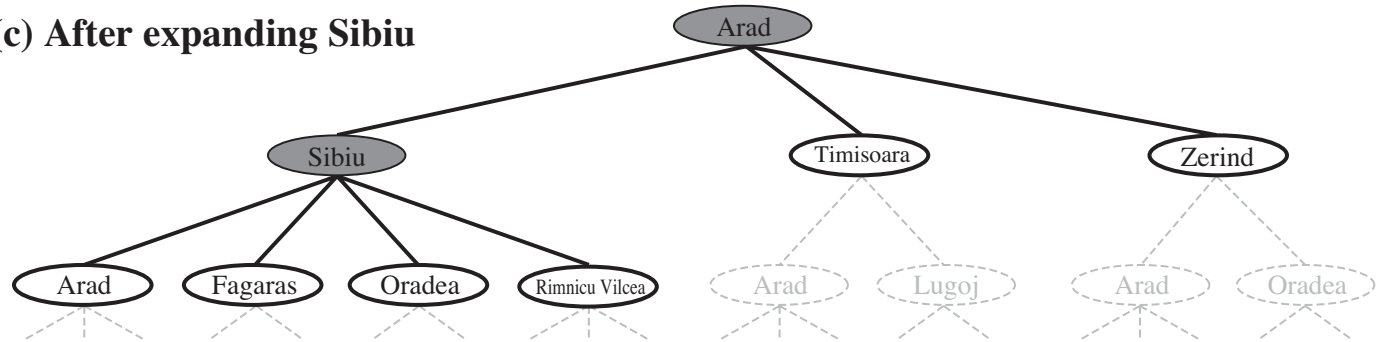
(a) The initial state



(b) After expanding Arad



(c) After expanding Sibiu



Alternative: ricerca su albero o su grafo

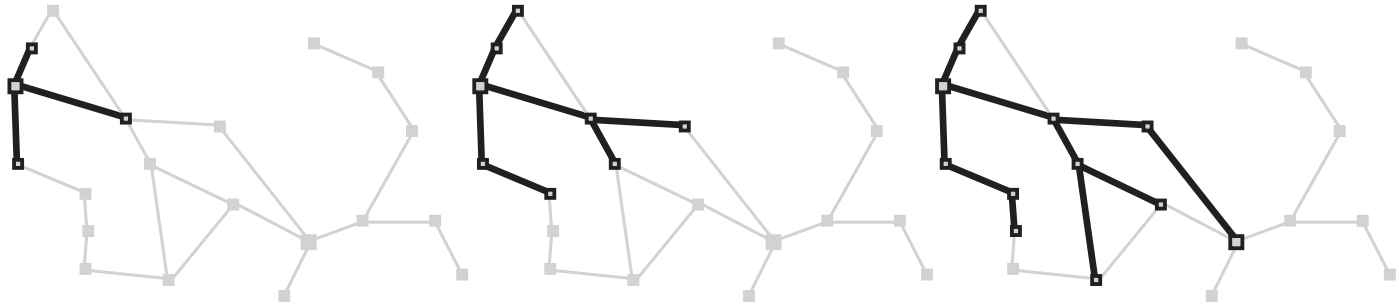
function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

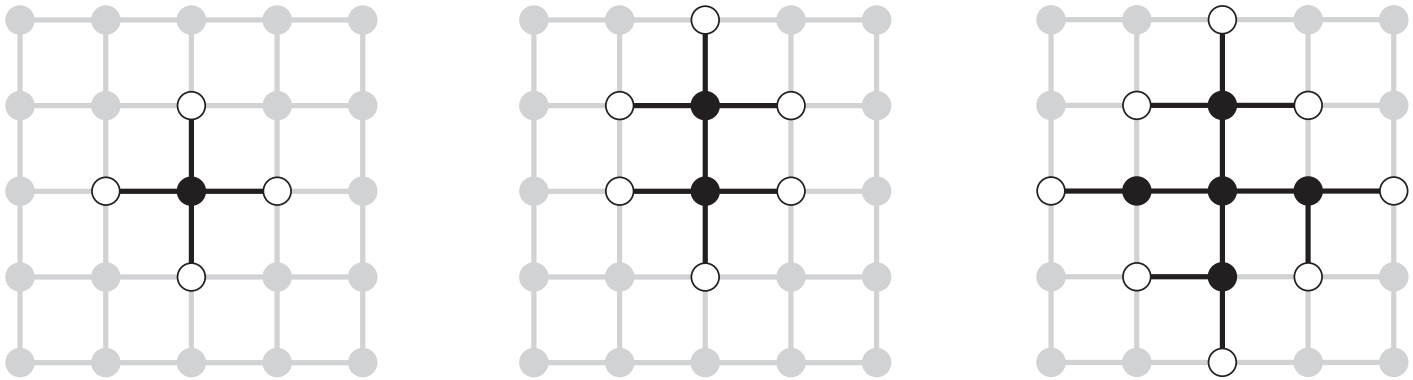
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set

Proprietà della ricerca su grafo

Parsimonia: al più una copia dello stesso stato



Separazione: la frontiera separa stati esplorati da quelli non esplorati



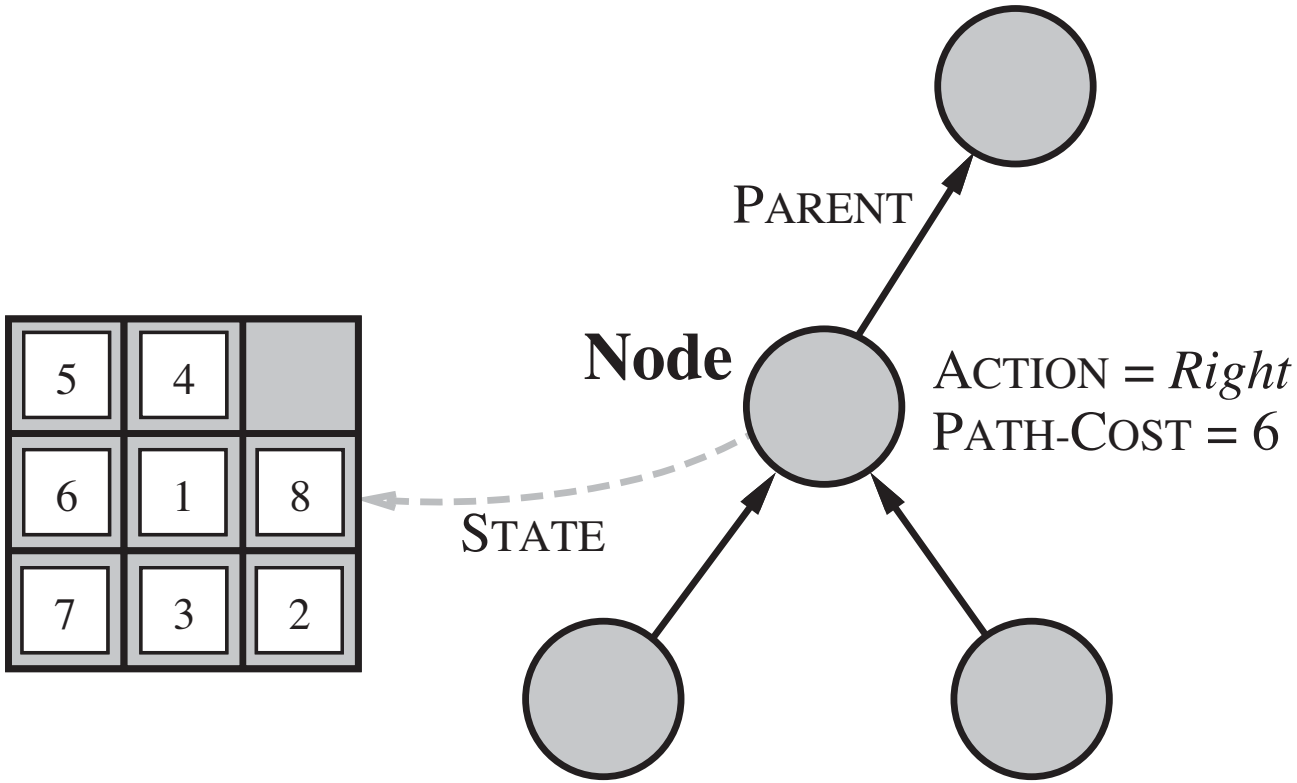
Implementazione

Uno *stato* è una rappresentazione di una configurazione fisica

Un *nodo* è una struttura dati che fa parte di un albero di ricerca

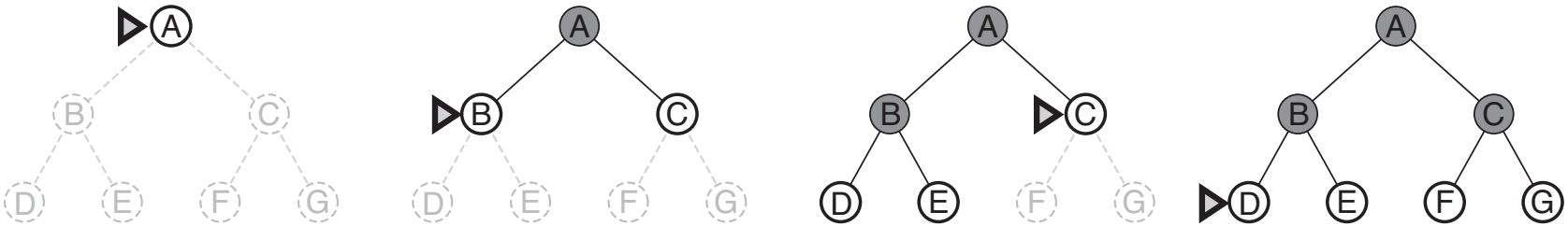
include *genitori*, *figli*, *profondità*, *costo del cammino* $g(x)$

Stati non hanno genitori, figli,...



Ricerca in Ampiezza (BFS) e a costo uniforme

Ricerca in ampiezza



Ricerca a costo uniforme

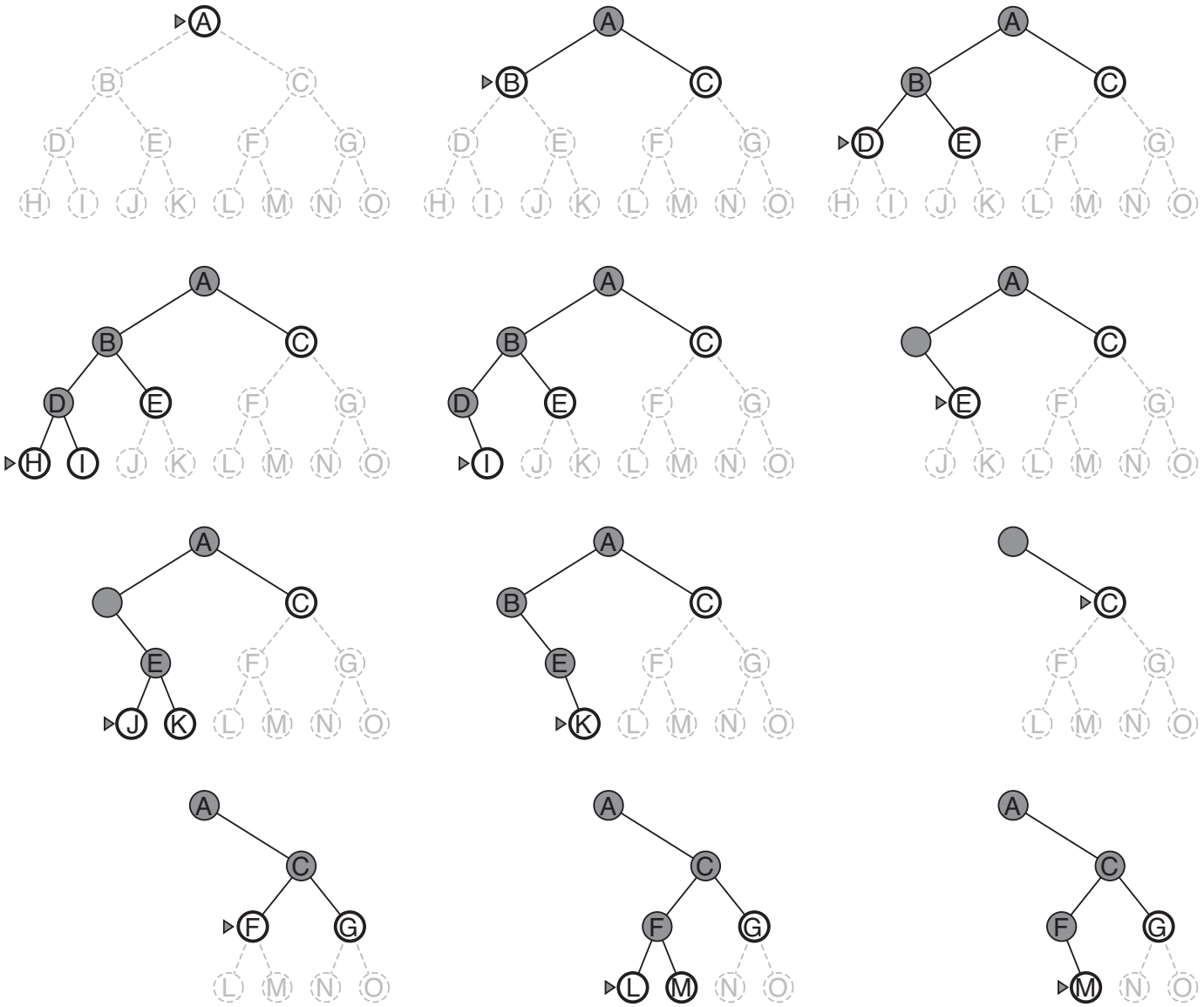
Espande il nodo a costo (di cammino) inferiore

Implementazione:

$$frontiera = \text{coda (a priorità) ordinata per costo di cammino}$$

Equivalente alla ricerca breadth-first se i costi dei singoli passi è identico

Ricerca in Profondità (DFS)

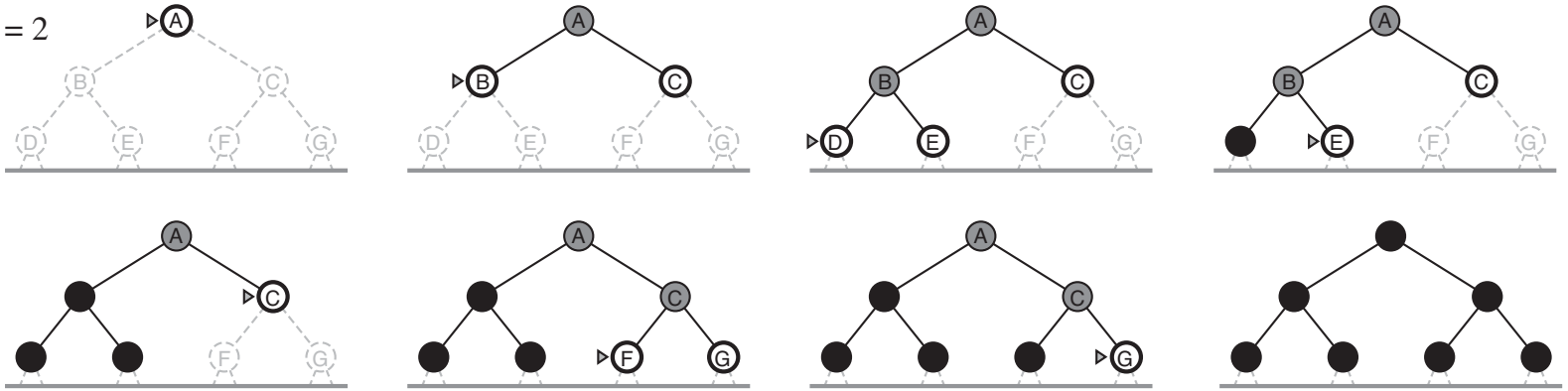


Ricerca in Profondità Iterativa (IDS)

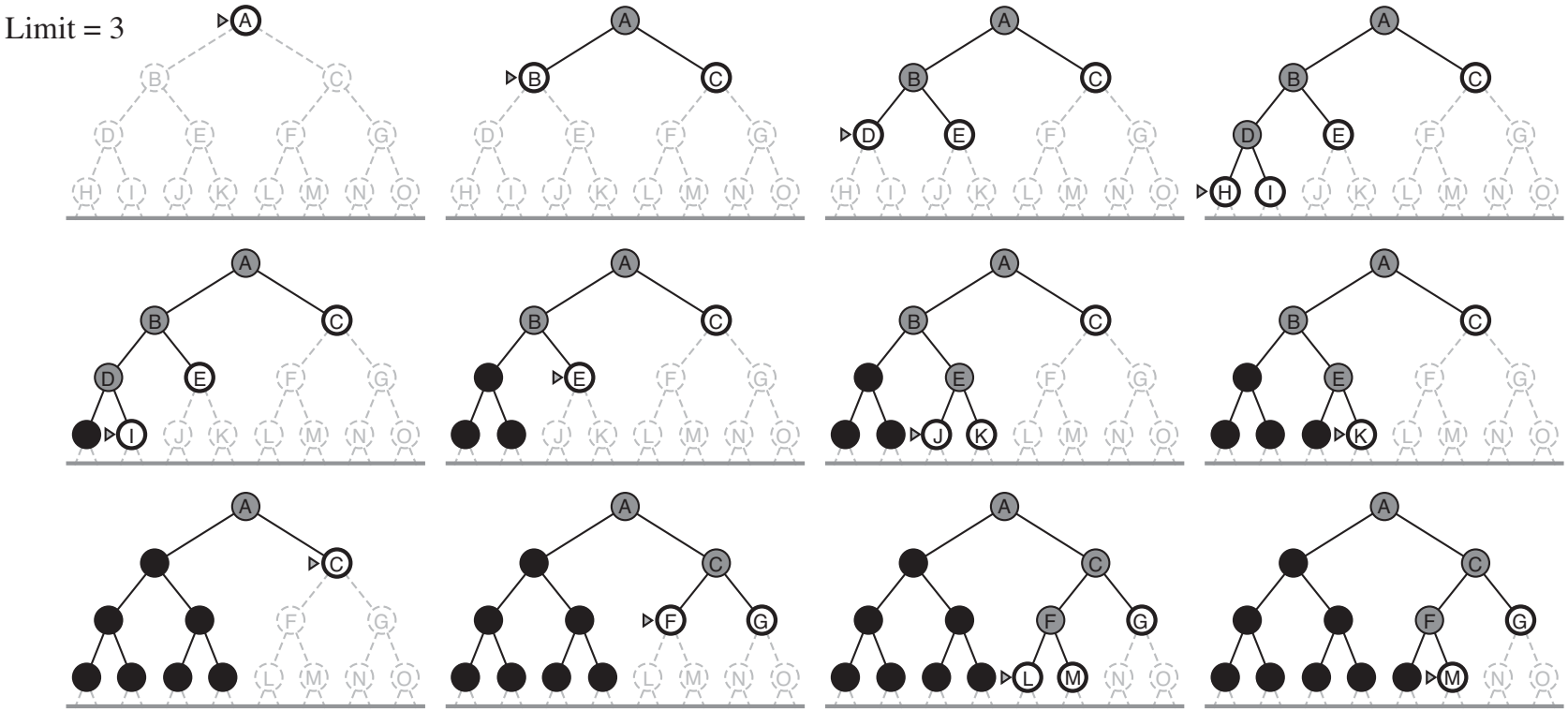
Limit = 0

Limit = 1

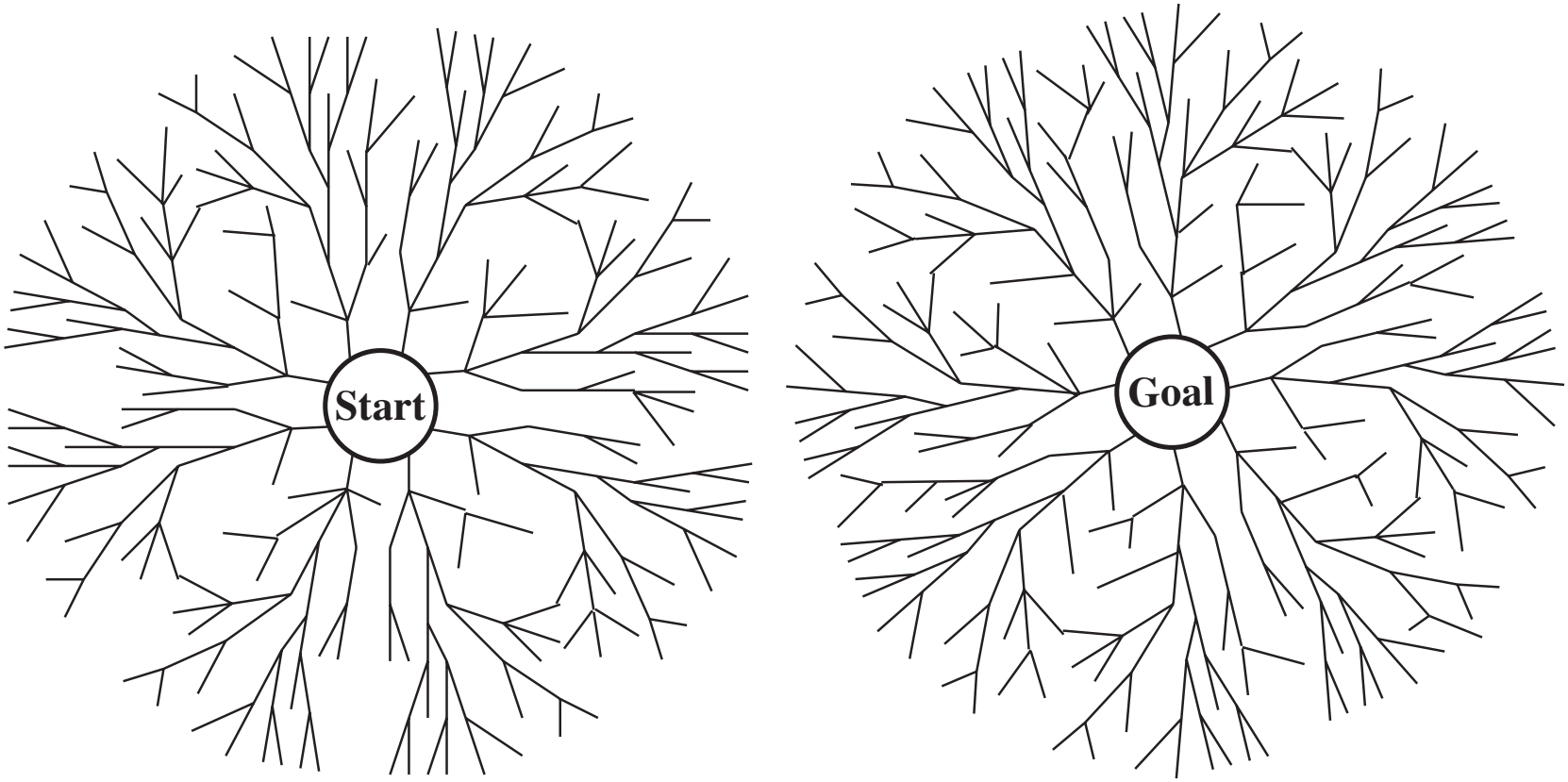
Limit = 2



Ricerca in Profondità Iterativa (IDS)



Ricerca bi-direzionale (BS)



Confronto fra strategie di ricerca

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Note: [a] completa se b finito; [b] completa se un passo ha costo $\geq \epsilon > 0$;
[c] ottima se i costi sono tutti identici; [d] se entrambi le direzioni usano BFS.

- b fattore di ramificazione (branching factor)
- d profondità della soluzione più vicina alla radice
- m profondità massima dell'albero di ricerca
- l limite alla profondità