

Advanced School in Artificial Intelligence

Strategie di ricerca

Strategie non informate

Marco Alberti

marco.alberti@unife.it



Progetto di alta formazione in ambito tecnologico economico e culturale per una regione della conoscenza europea e attrattiva approvato e cofinanziato dalla Regione Emilia-Romagna con deliberazione di Giunta regionale n. 1625/2021



Università
degli Studi
di Ferrara

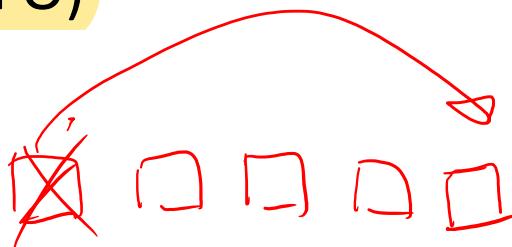
Strategie non informate

- Non utilizzano stime del costo da uno stato alla soluzione
- Adottano una strategia fissa per selezionare dalla frontiera il nodo da espandere
- Algoritmi:
 - Breadth-first
 - Uniform cost
 - Depth-first
 - Depth-first a profondità limitata
 - Approfondimento iterativo



Breadth-first

- nodi aggiunti in coda alla frontiera (FIFO)



```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node  $\leftarrow$  NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier  $\leftarrow$  a FIFO queue, with node as an element
  reached  $\leftarrow$  {problem.INITIAL}
  while not Is-EMPTY(frontier) do
    node  $\leftarrow$  POP(frontier)
    for each child in EXPAND(problem, node) do
      s  $\leftarrow$  child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier 
    return failure
```

Proprietà breadth-first

[A] [T so] [S o L]

- Completezza: sì.

- Se la soluzione di profondità minima è a profondità n , la strategia esplora tutti i nodi a profondità $< n$ e tutti quelli di profondità n fino alla soluzione

- Ottimalità: no

- La strategia trova una soluzione di profondità minima, che non corrisponde necessariamente a costo minimo
 - ASFB: profondità 3, costo 450
 - ASRPB: profondità 4, costo 418

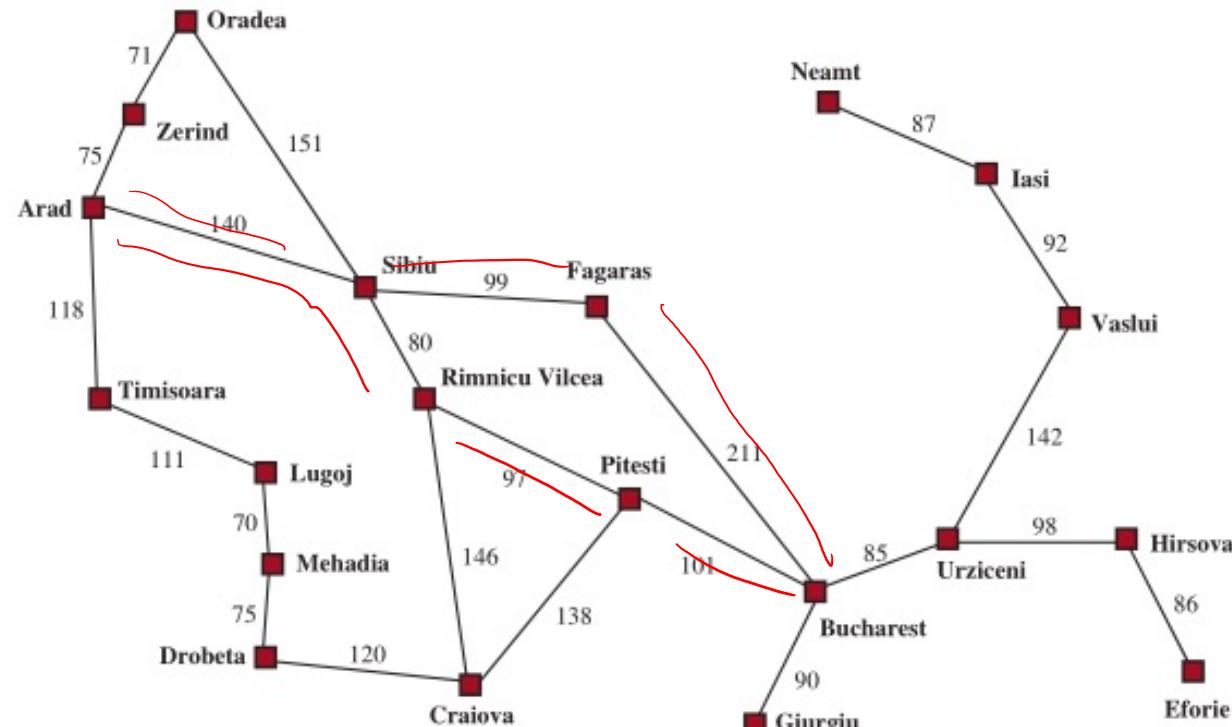


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

Proprietà breadth-first



- b branching factor, d profondità della soluzione più breve
- Complessità temporale (nodi espansi):

$$1 + b + b^2 + \dots + b^{d-1} + b^d + b(b^d - 1) = O(b^{d+1})$$

- Complessità spaziale: tutti i nodi della stessa profondità di quello della soluzione sono, nel caso peggiore,

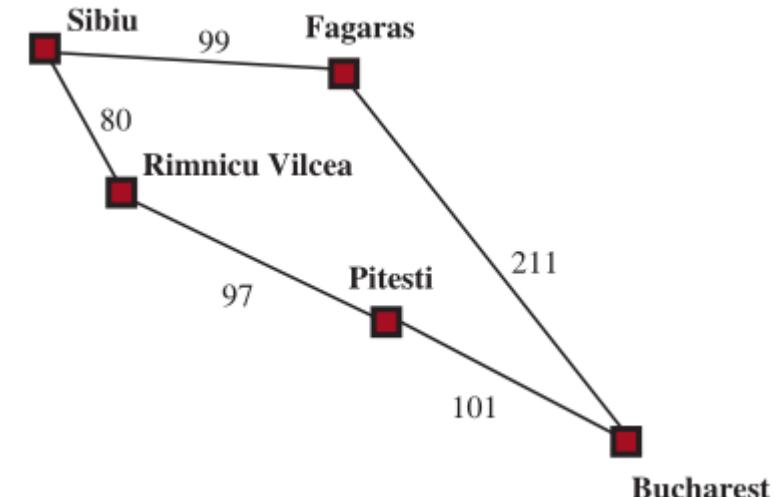
$$b(b^d - 1) = O(b^{d+1})$$

- Conclusione: breadth-first
 - non garantisce ottimalità
 - usa molta memoria

Uniform cost search

```
function UNIFORM-COST-SEARCH(problem) returns a solution node, or failure
  return BEST-FIRST-SEARCH(problem, PATH-COST)
```

- Add inserisce i nodi nella coda in ordine crescente di costo di cammino (e non di profondità)



SRPB (soluzione ottima) verrebbe espanso prima di SFB (soluzione più breve)

Proprietà uniform-cost search

- Completezza: se ogni transizione ha costo $\geq \varepsilon > 0$ 
- Ottimalità: se ogni transizione ha costo $\geq \varepsilon > 0$.
 - il costo dei cammini corrispondenti ai nodi esplorati è non decrescente, quindi trova prima la soluzione di costo minimo
 $\downarrow d$
 $\downarrow c$
- Complessità temporale e spaziale: $O(b^{\frac{C}{\varepsilon}+1})$
 - b è il branching factor
 - C è il costo della soluzione ottima .
 - ε è il costo minimo di una transizione .
 - La soluzione ottima avrà profondità, al massimo, C/ε , da cui la complessità come nel caso breadth-first

Depth-first

- La frontiera è una coda LIFO (o uno stack)
- Vengono espansi prima gli ultimi nodi aggiunti

Ordine visite se si evitano loop:

1. A
2. AS AT AZ
3. ASA ASR AT AZ
4. ASFB ASR AT AZ

Altrimenti:

1. A
2. AS AT AZ
3. ASA ASF ASR AT AZ
4. ASAS ASAT ASAZ ASF ASR AT AZ
5. ASASA... (loop) 

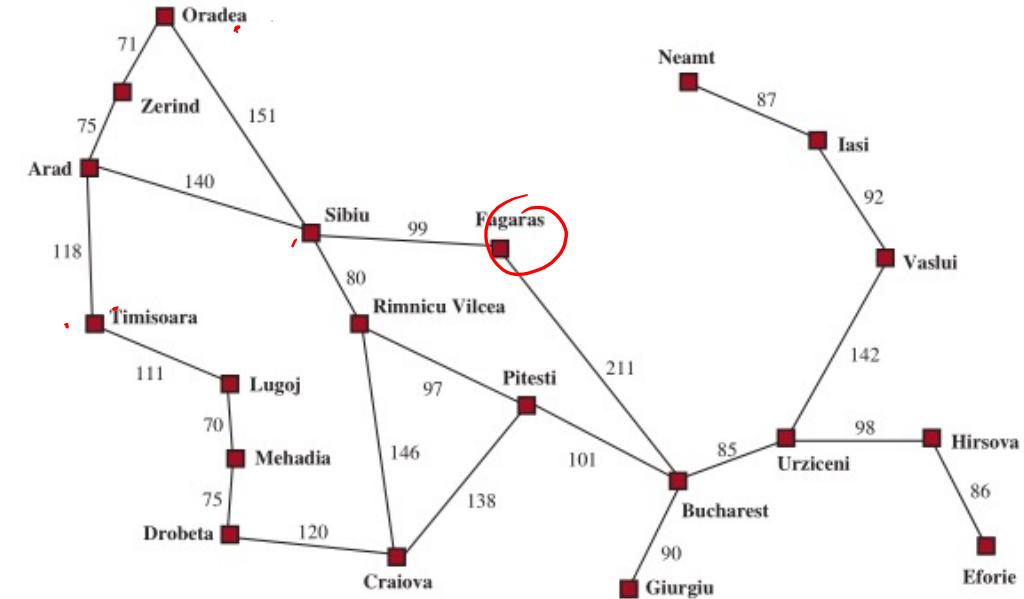
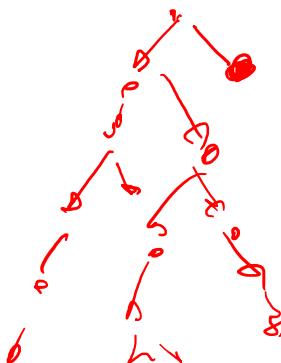


Figure 3.1 A simplified road map of part of Romania, with road distances in miles.

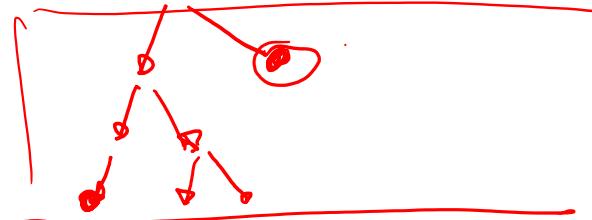
Proprietà depth-first

- Prima le buone notizie: complessità spaziale $O(bm)$ dove b è il branching factor e m è la profondità dell'albero di ricerca
 - è necessario memorizzare solo il primo fratello non esplorato di ogni nodo lungo il cammino dall'ultimo nodo visitato alla radice (backtracking)
- ma:
 - non completa in caso di path infiniti
 - soggetta a loop in-path e a cicli (se tree search anziché graph search)
 - non ottimale
 - complessità temporale $O(b^m)$



Depth limited search

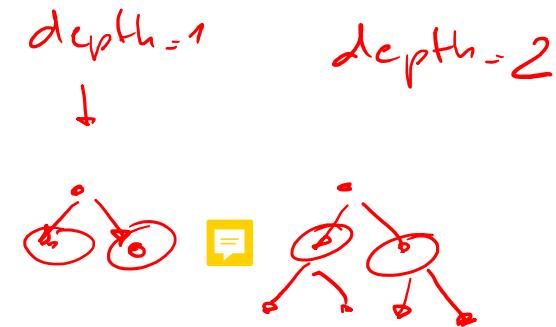
- Esplora solo i nodi di profondità $\leq l$
- Risolve (ovviamente) il problema dei cammini infiniti
- Ma trova solo le soluzioni di profondità $d \leq l$ (se ce ne sono)



```
function DEPTH-LIMITED-SEARCH(problem,  $\ell$ ) returns a node or failure or cutoff
  frontier  $\leftarrow$  a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result  $\leftarrow$  failure
  while not IS-EMPTY(frontier) do
    node  $\leftarrow$  POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node)  $>$   $\ell$  then
      result  $\leftarrow$  cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

Iterative deepening

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to  $\infty$  do
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)
    if result  $\neq$  cutoff then return result
```



- Ripete la ricerca depth-limited incrementando il limite fino a trovare una soluzione

Proprietà iterative deepening

- Completezza: sì
- Ottimalità: trova la soluzione più breve, non quella di costo minimo.
 - Per avere ottimalità si può imporre bound sul costo (espandere solo i nodi di costo $\leq C$ anziché quelli di profondità $\leq l$);
 - ma così la profondità può aumentare o diventare illimitata (se il costo delle transizioni non è $\geq \varepsilon > 0$; v. uniform cost search)
- Complessità temporale: $(d + 1)b^0 + db^1 + \dots + b^d = O(b^d)$
 - è vero che i nodi di profondità $\leq d$ vengono visitati più volte
 - ma (soprattutto se il branching factor è grande) la maggior parte dei nodi sta a profondità d , quindi la complessità non peggiora di molto

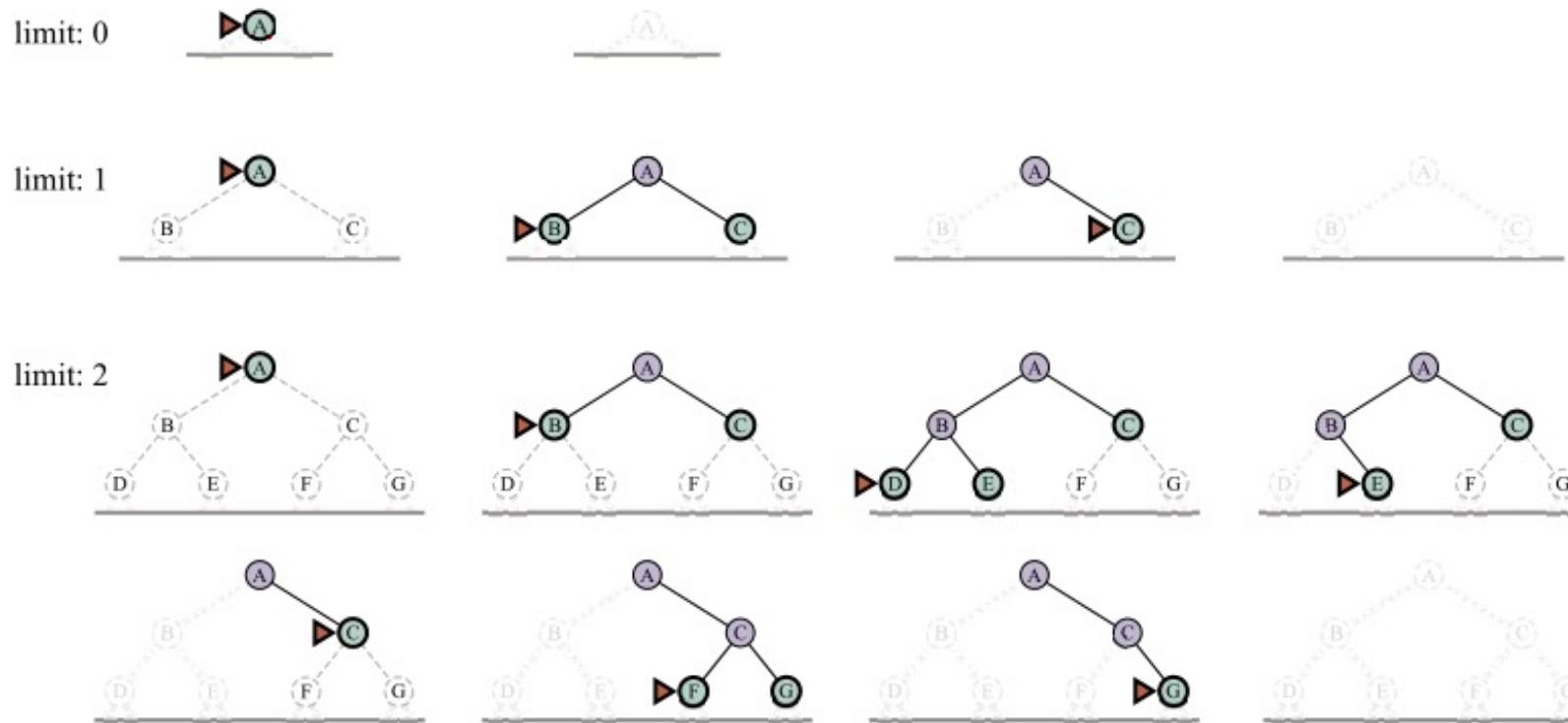
1 120 1000
10000 0

Proprietà iterative deepening

- Complessità spaziale: $O(bd)$, lineare!
- Complessivamente iterative deepening è
 - completa come breadth-first
 - ottimale come uniform-cost (nella variante con costo limite e con le stesse ipotesi sul costo delle transizioni)
 - memory-efficient come depth-first

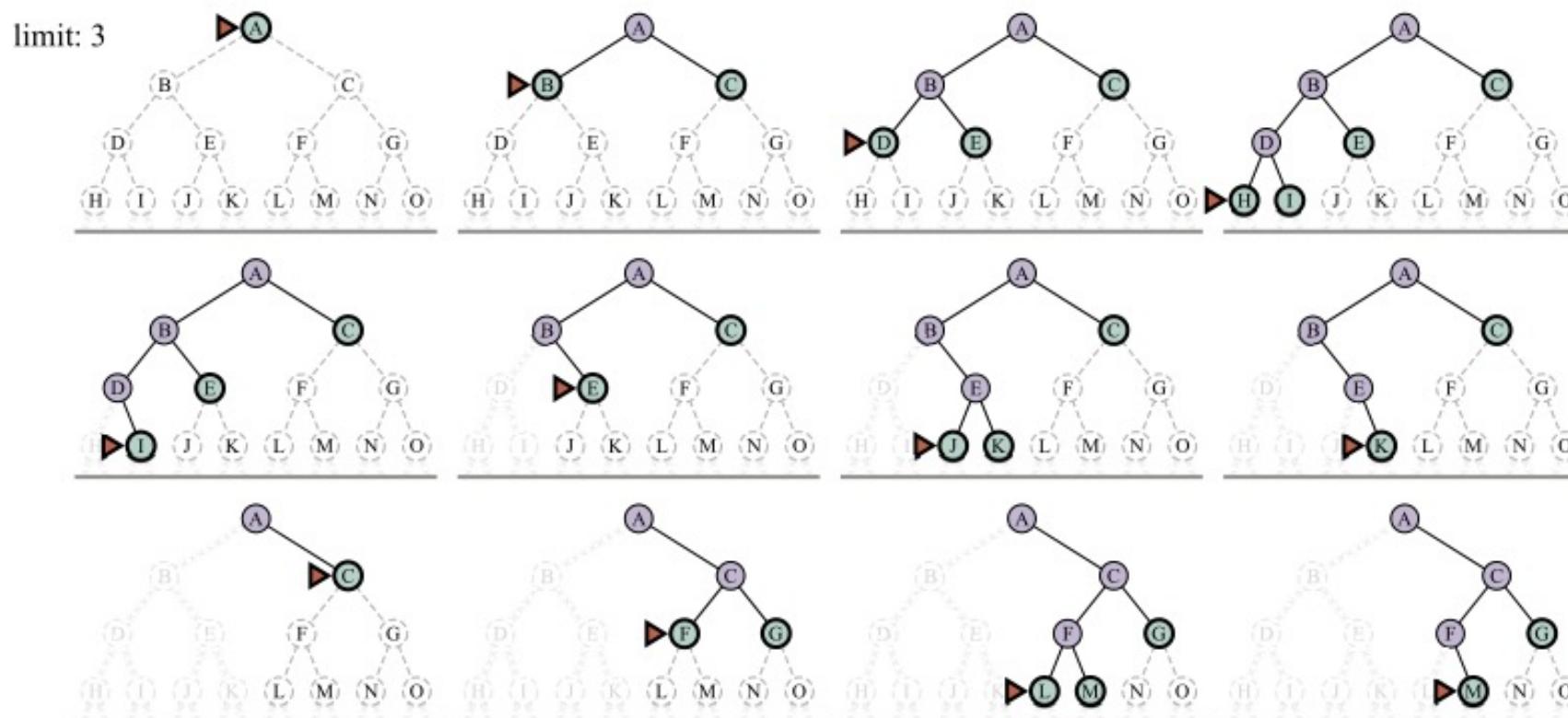


Esempio iterative deepening



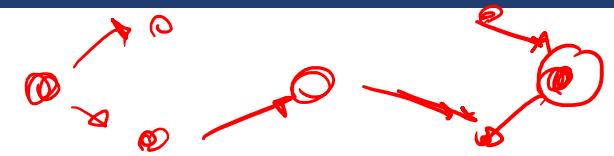
Triangolo: prossimo nodo da espandere. Verde: frontiera. Azzurro: esplorati.
Schiariti: di fallimento al limite impostato

Esempio iterative deepening



Triangolo: prossimo nodo da espandere. Verde: frontiera. Azzurro: esplorati.
Pallidi: di fallimento al limite impostato

Valutazione algoritmi di ricerca



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ¹	Yes ^{1,2}	No	No	Yes ¹	Yes ^{1,4}
Optimal cost?	Yes ³	Yes	No	No	Yes ³	Yes ^{3,4}
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	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Figure 3.15 Evaluation of search algorithms. b is the branching factor; m is the maximum depth of the search tree; d is the depth of the shallowest solution, or is m when there is no solution; ℓ is the depth limit. Superscript caveats are as follows: ¹ complete if b is finite, and the state space either has a solution or is finite. ² complete if all action costs are $\geq \epsilon > 0$; ³ cost-optimal if action costs are all identical; ⁴ if both directions are breadth-first or uniform-cost.