INTELLIGENZA ARTIFICIALE

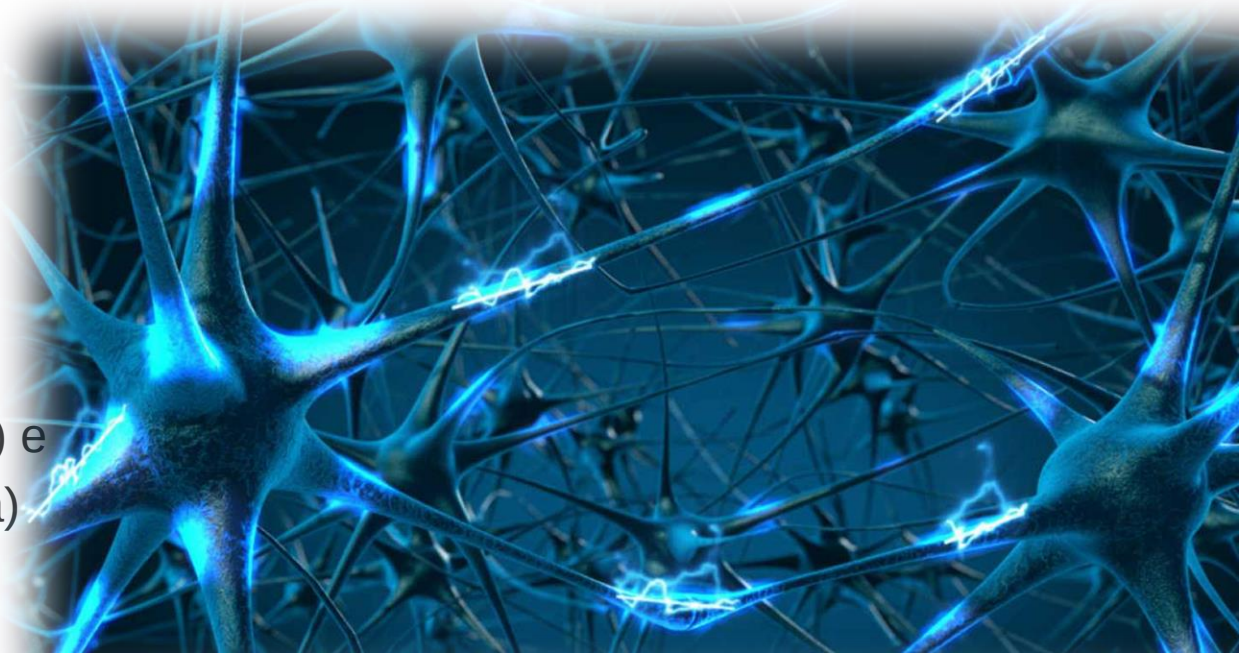
ALGORITMI DI RICERCA INFORMATA (*)

Corsi di Laurea in Informatica, Ing. Gestionale, Ing. Informatica, Ing. di Internet

(a.a. 2022-2023)

Roberto Basili

(*) alcune *slides* sono di
Andrew NG (Stanford) e
Maria Simi (Univ. Pisa)



Overview

- Algoritmi euristici: la funzione euristica
  - Scopi e proprietà
- Algoritmo Greedy Best-first
- Algoritmo A*
  - Proprietà
- Funzioni Euristiche
- Algoritmi basati su A*
  - Beam Search
  - IDA*

Ricerca euristica

- La ricerca esaustiva non è praticabile in problemi di complessità esponenziale
- Noi usiamo conoscenza del problema ed esperienza per riconoscere i cammini più promettenti.
- La *conoscenza euristica* (dal greco “eureka”) aiuta a fare scelte “oculate”
  - non evita la ricerca ma la riduce
  - consente in genere di trovare una buona soluzione in tempi accettabili.
  - sotto certe condizioni garantisce completezza e ottimalità

Ricerca euristica

- La *conoscenza euristica* (dal greco “eureka”) aiuta a fare scelte “oculate”

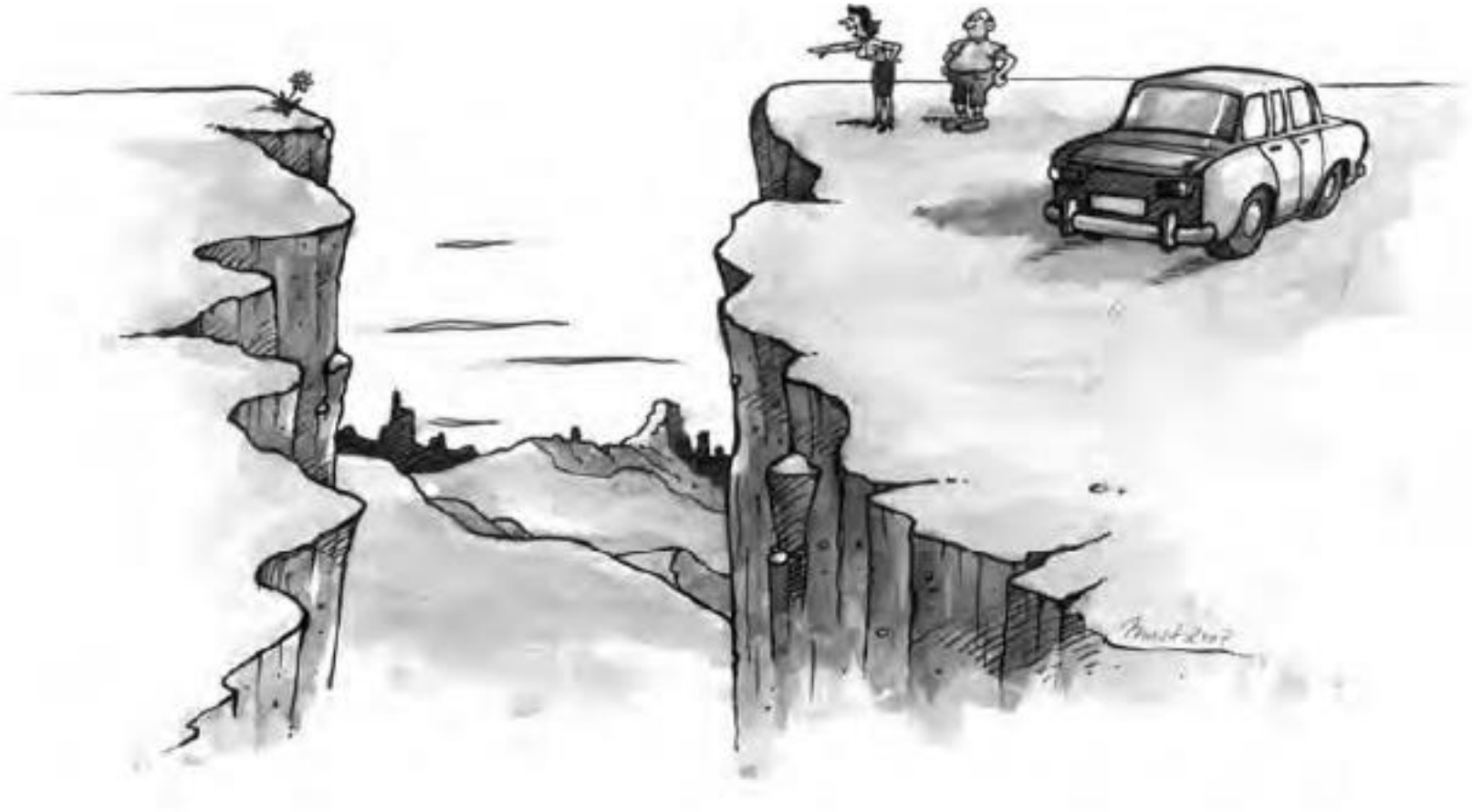


Fig. 6.13 He: “Dear, think of the fuel cost! I’ll pluck one for you somewhere else.” She: “No, I want that one over there!”

Funzioni di valutazione euristica

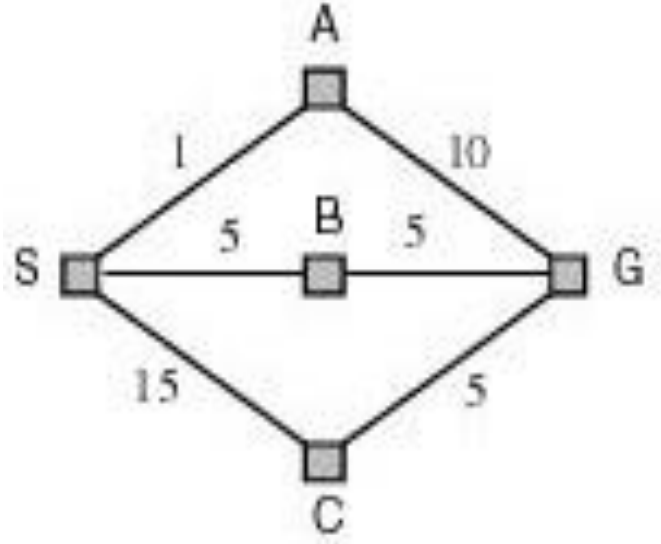
Conoscenza del problema data tramite una *funzione di valutazione* dello stato, detta funzione di valutazione euristica:

$$f : n \rightarrow \mathrm{R}$$

La funzione si applica al nodo ma dipende solo dallo stato (*n*.Stato)

Esempi di euristica

- La città più vicina (o la città più vicina alla mèta in linea d'aria) nel problema dell'itinerario



- Il numero delle caselle fuori posto nel gioco dell'otto
- Il vantaggio in pezzi nella dama o negli scacchi

Algoritmo di ricerca Best-First

- Ad ogni passo si sceglie il nodo sulla frontiera per cui il valore della *f* è migliore (il nodo più promettente).
- Migliore significa 'minore' in caso di un’euristica che stima la distanza della soluzione
- Implementata da una *coda con priorità* che ordina in base al valore della funzione di valutazione euristica.

Ricerca Best First

$f(n)=g(n)$ con g pari al PATH-COST

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure

node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
frontier ← a priority queue ordered by PATH-COST, with *node* as the only element
explored ← an empty set
loop do
  if EMPTY?(*frontier*) **then return** failure
  node ← POP(*frontier*) /* chooses the lowest-cost node in *frontier* */
  if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  add *node*.STATE to *explored*
  for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
    child ← CHILD-NODE(*problem*, *node*, *action*)
    if *child*.STATE is not in *explored* or *frontier* **then**
      frontier ← INSERT(*child*, *frontier*)
    else if *child*.STATE is in *frontier* with higher PATH-COST **then**
      replace that *frontier* node with *child*

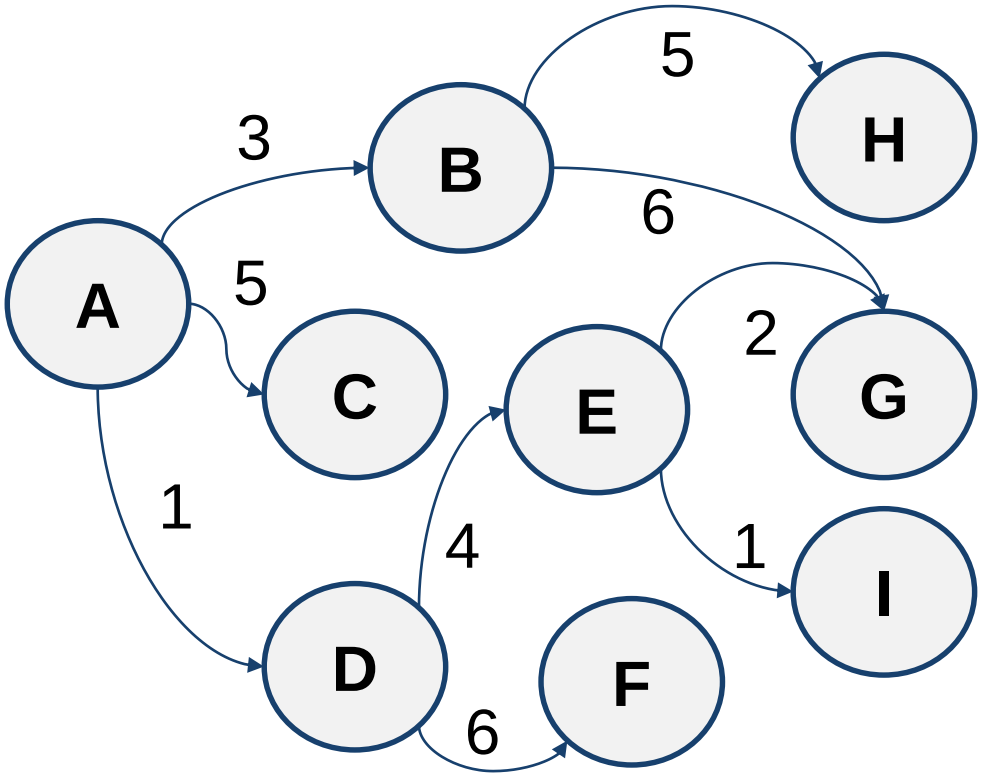
Ricerca Best First

f(n)=h(n)

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE,
  frontier ← a priority queue ordered by f(n), with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher f(n) then
        replace that frontier node with child
```

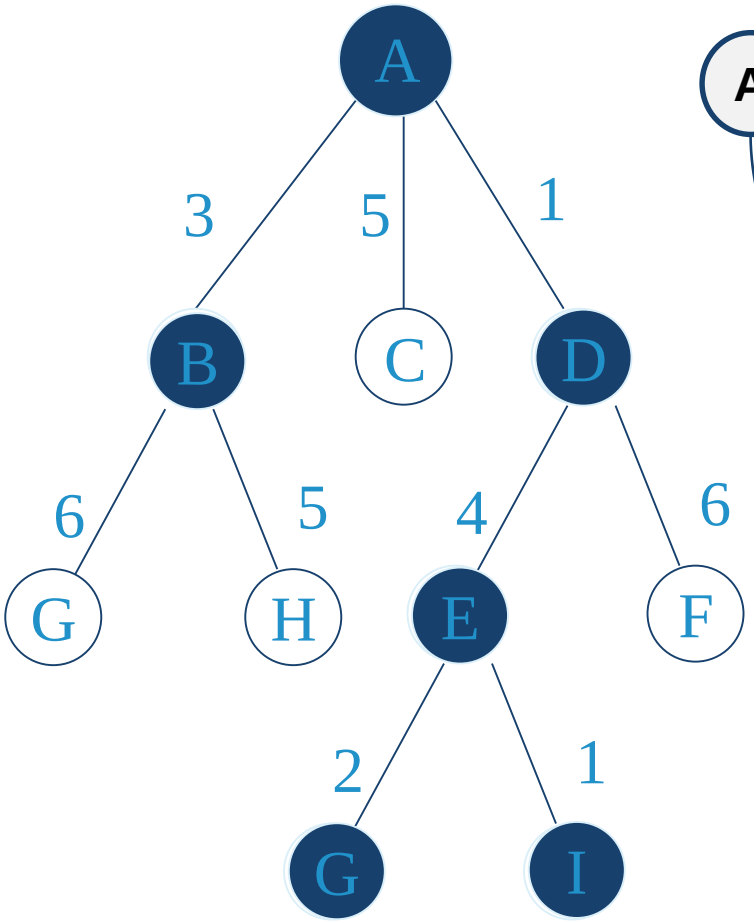
Greedy Best First

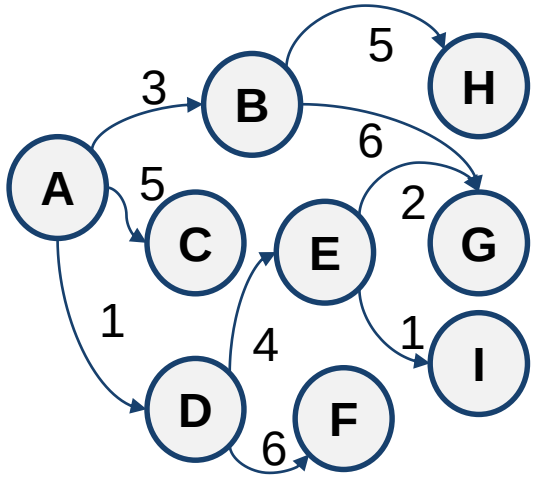
Esempio:
GBF per il *minimum spanning tree*



Strategia best-first: esempio

Passo 7





La Best First non è in generale completa, né ottimale

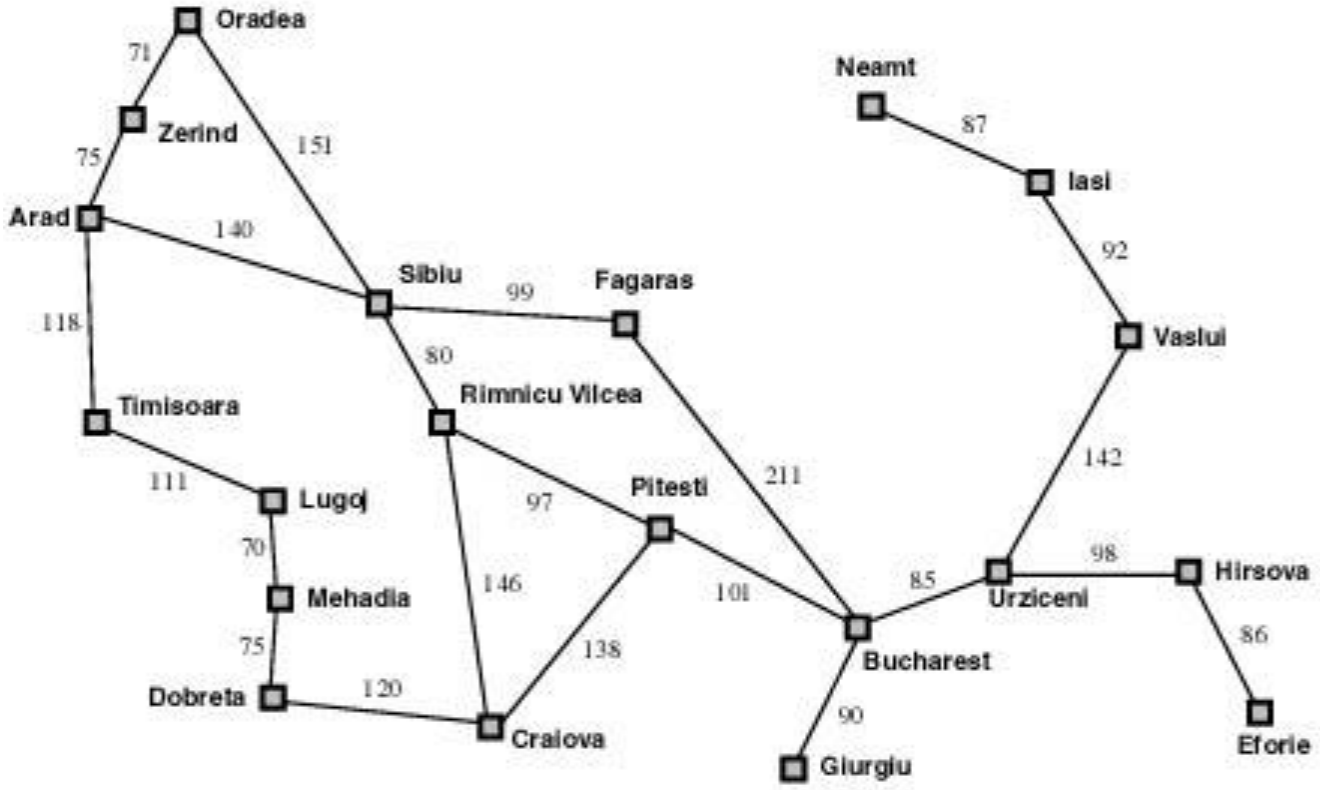
Ricerca greedy best-first

- Quando best-first è applicato ad una f(n) che consiste solo della stima euristica della distanza di *n* della soluzione, da ora in poi *h*(*n*) [*h*≥*0*], la ricerca prende il nome di **greedy best-first**.

- *Esempio*:
  - ricerca greedy per Route Finding che usi *h*(*n*) = *distanza in linea d'aria* tra lo stato *n* e il goal

  In generale l'algoritmo *non è completo*

Ricerca greedy: esempio

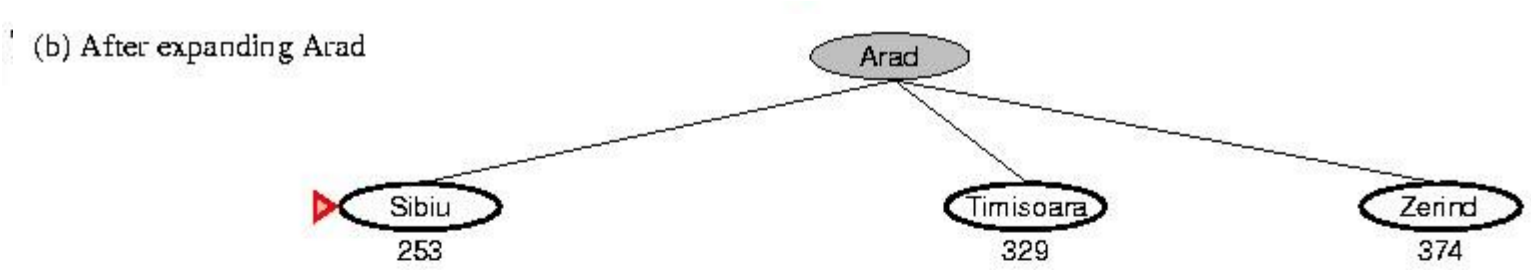


Straightline distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

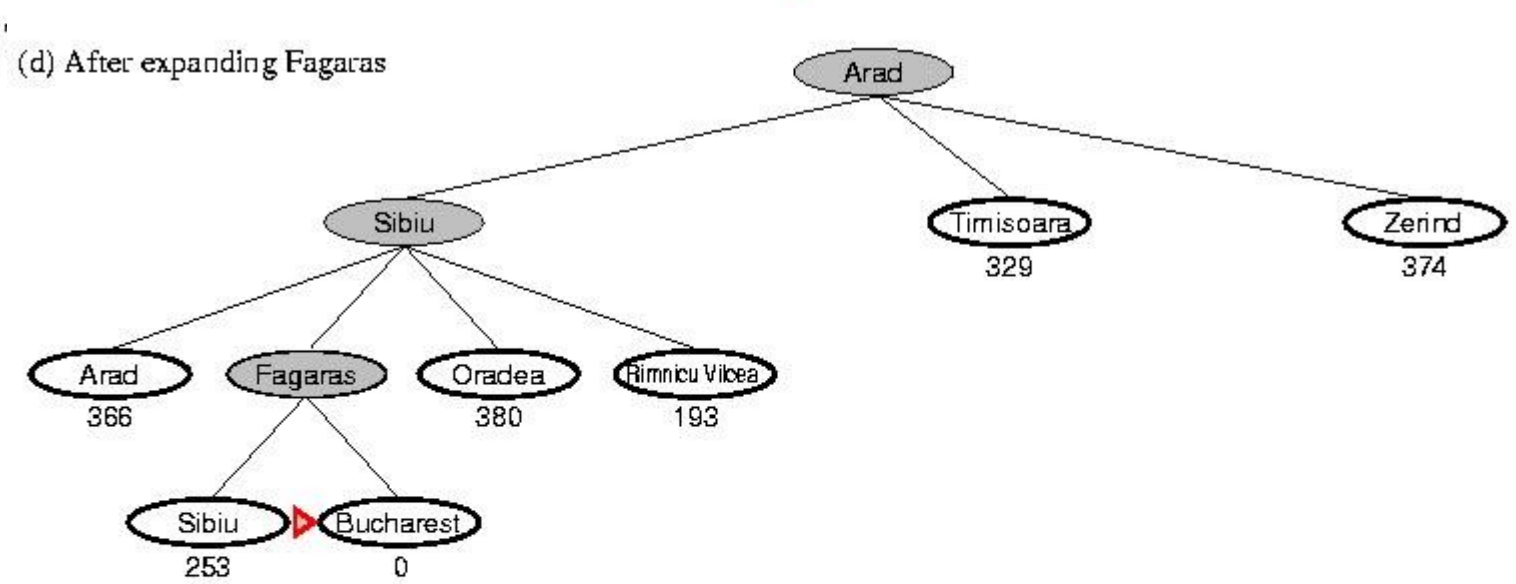
Itinerario con Greedy Best-First

The initial state

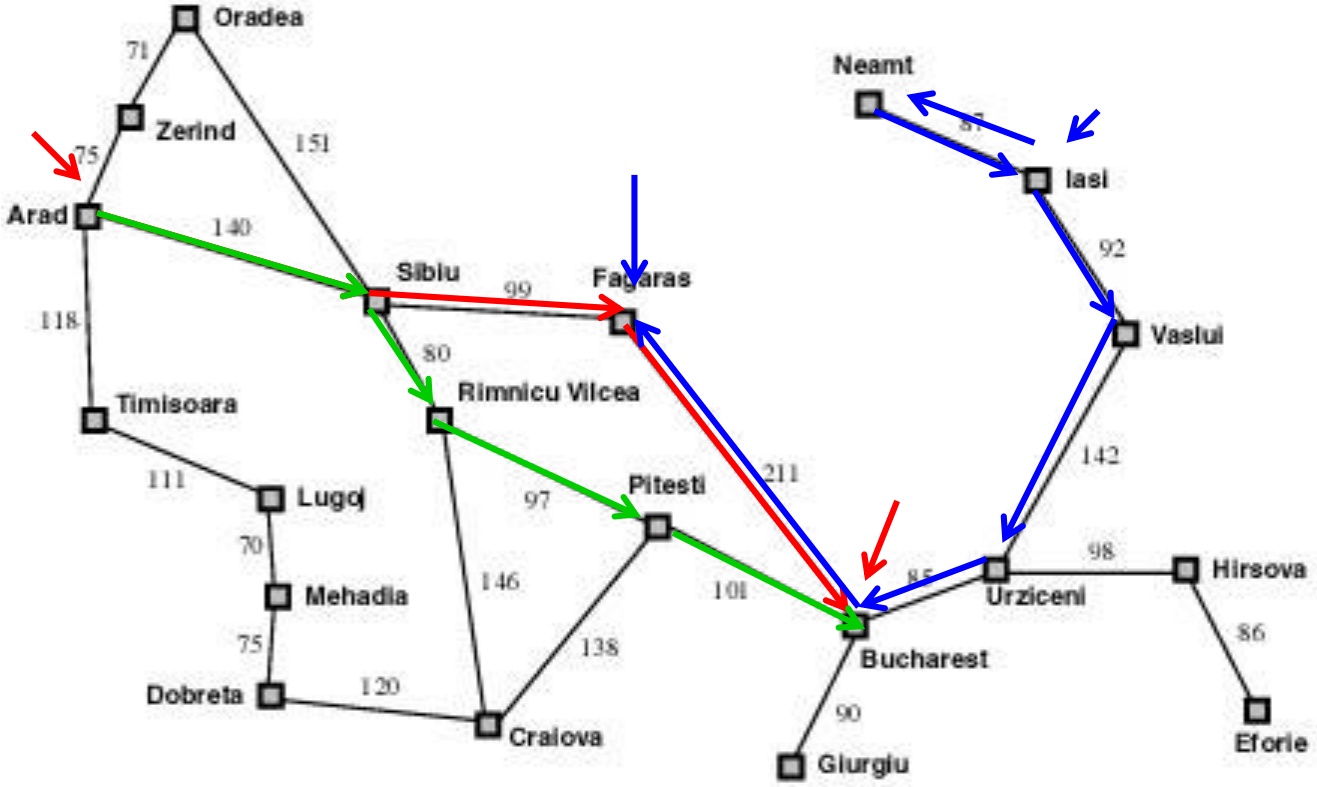
Itinerario con Greedy Best-First



Itinerario con Greedy Best-First



Ricerca greedy: esempio



Straightline 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

Da Arad a Bucarest …

Greedy: Arad, Sibiu, Fagaras, Bucharest (450)

Ottimo: Arad, Sibiu, Rimnicu, Pitesti, Bucarest (418)

Da Iasi a Fagaras: … falsa partenza … o cicla

A* search

- Idea: Evitare di espandere cammini che siano già costosi

- Funzione di Valutazione: $f(n) = g(n) + h(n)$

- $g(n)$ = costo incorso nel raggiungere n
- $h(n)$ = (stima del) del costo nel raggiungere il goal partendo da n
- $f(n)$ = (stima del) costo totale del cammino passante per n in grado di raggiungere il nostro goal

Algoritmo A: definizione

- Si può dire qualcosa di *f* per avere garanzie di completezza e ottimalità?
- Un algoritmo A è un algoritmo *Best First* con una funzione di valutazione dello stato del tipo:

$$f(n) = g(n) + h(n), \text{ con } h(n) \geq 0 \text{ e } h(goal)=0$$

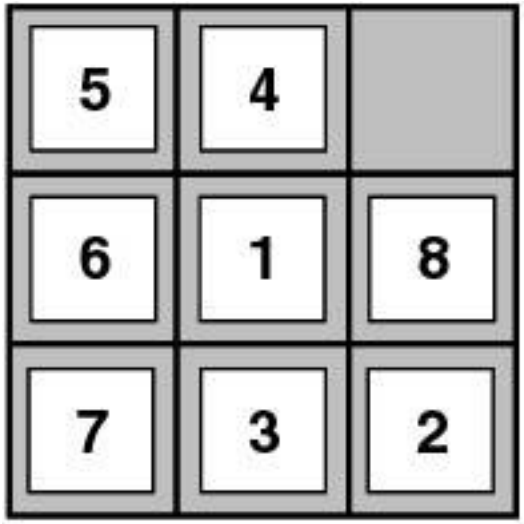
  - $g(n)$ è il costo del cammino percorso per raggiungere n
  - $h(n)$ una stima del costo per raggiungere da n un nodo goal

Casi particolari dell'algoritmo A:

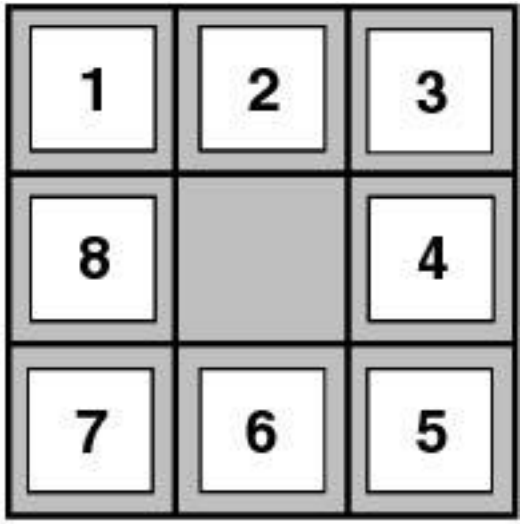
- Se $h(n) = 0$ [$f(n) = g(n)$] si ha Ricerca Uniform Cost
- Se $g(n) = 0$ [$f(n) = h(n)$] si ha Greedy Best First

Algoritmo A: esempio

Esempio nel gioco dell'otto



Start State

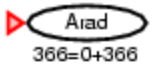


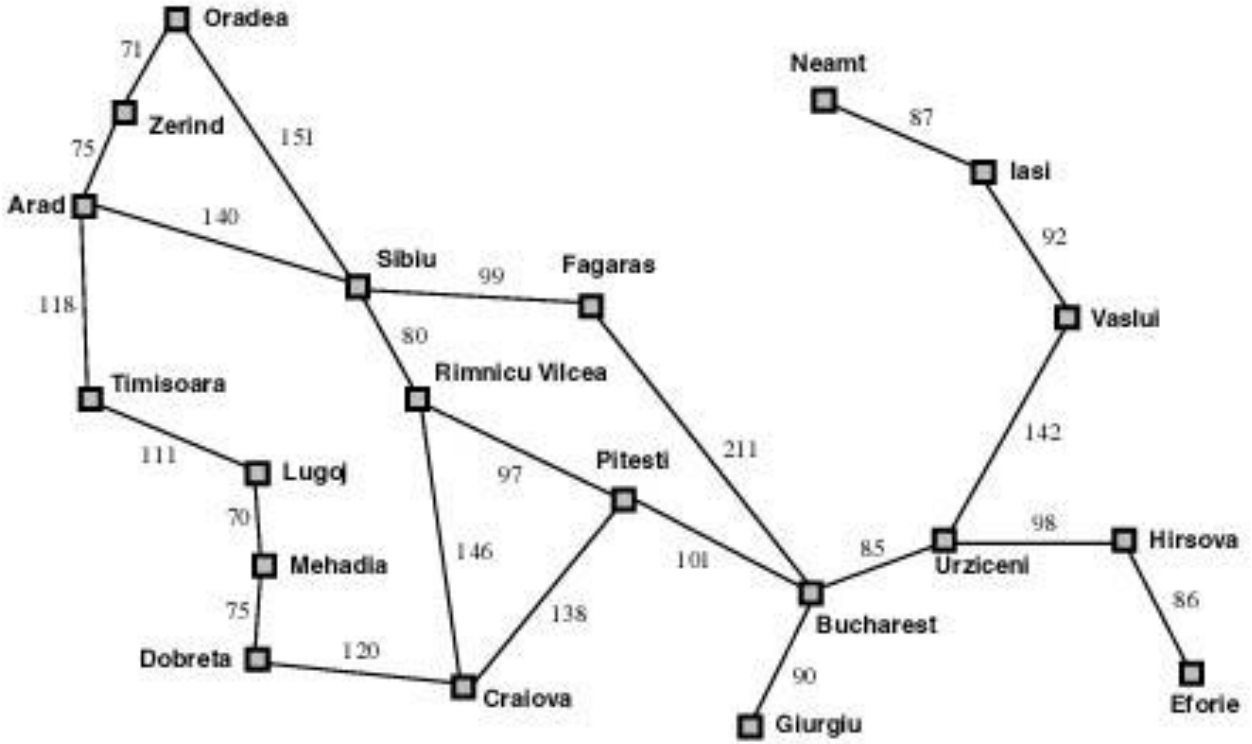
Goal State

$f(n)$ = #mosse fatte + #caselle-fuori-posto

f(Start) = 0 + 7 Dopo $\leftarrow, \downarrow, \uparrow, \rightarrow$ $f = 4 + 7$

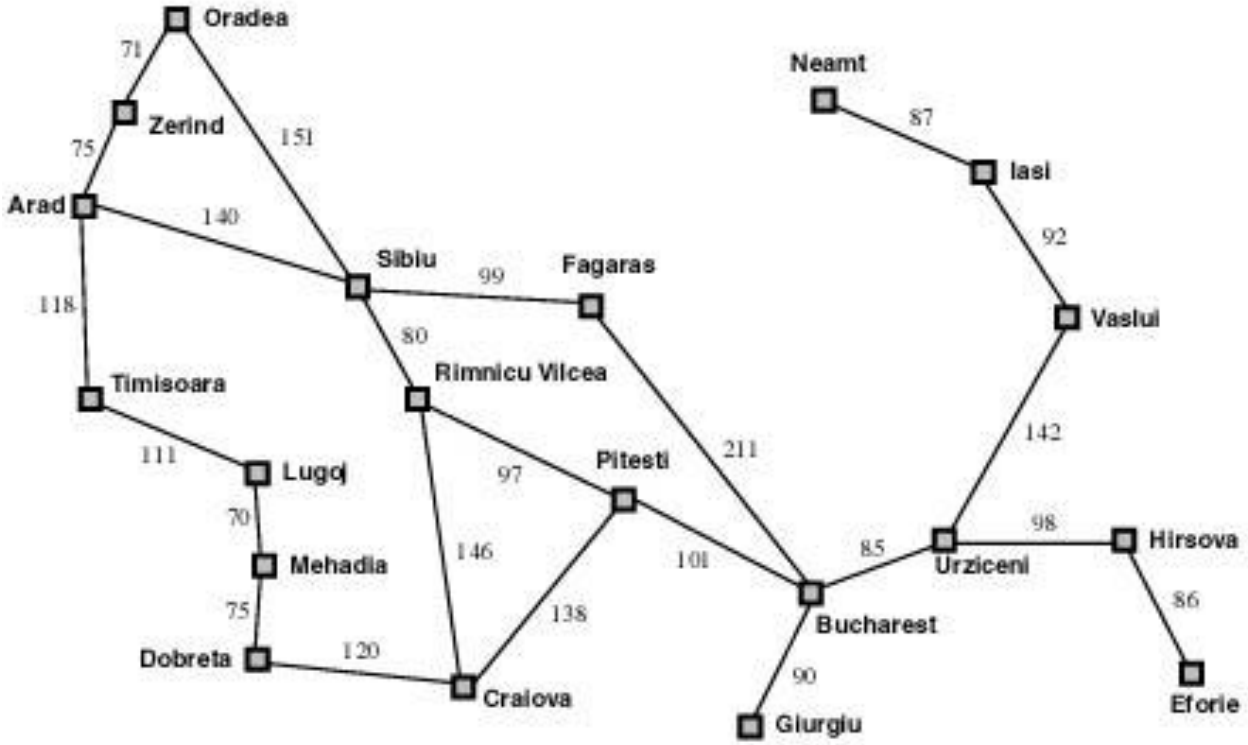
A* search example





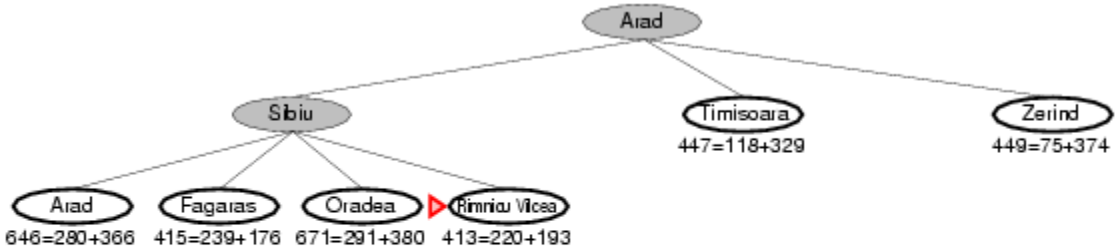
Straightline distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

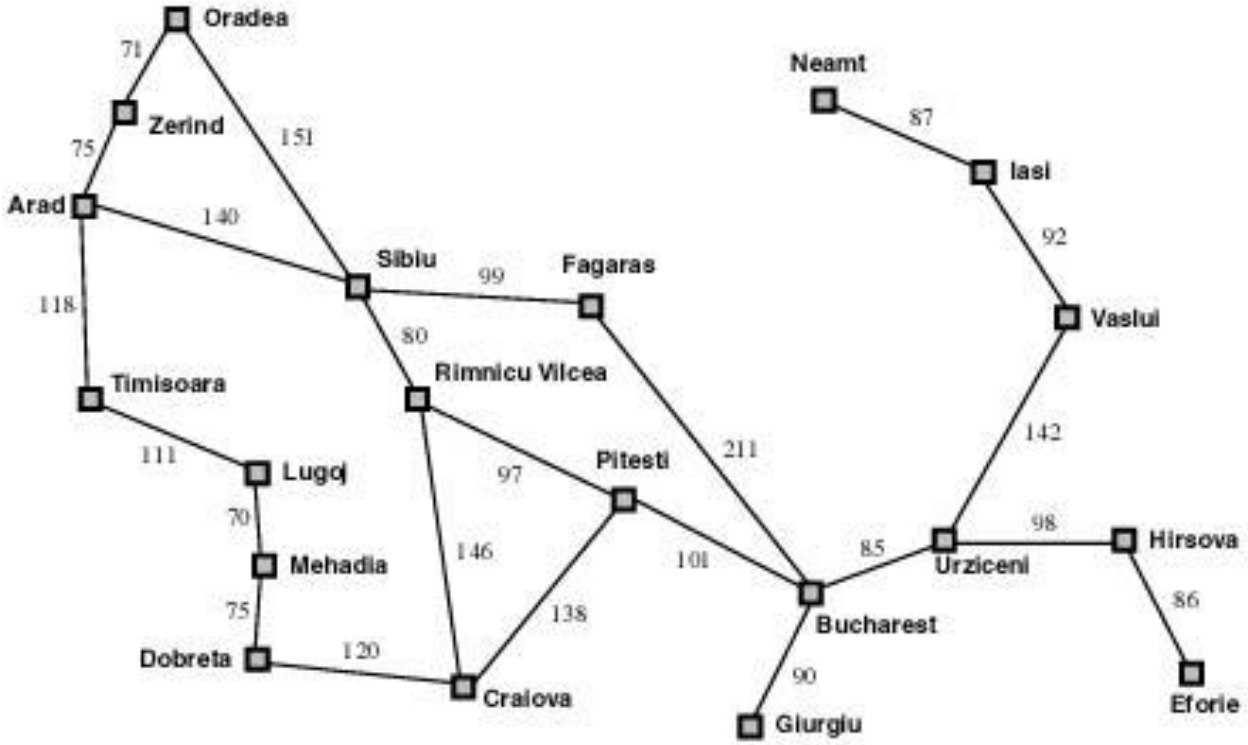
A* search example



Straightline 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

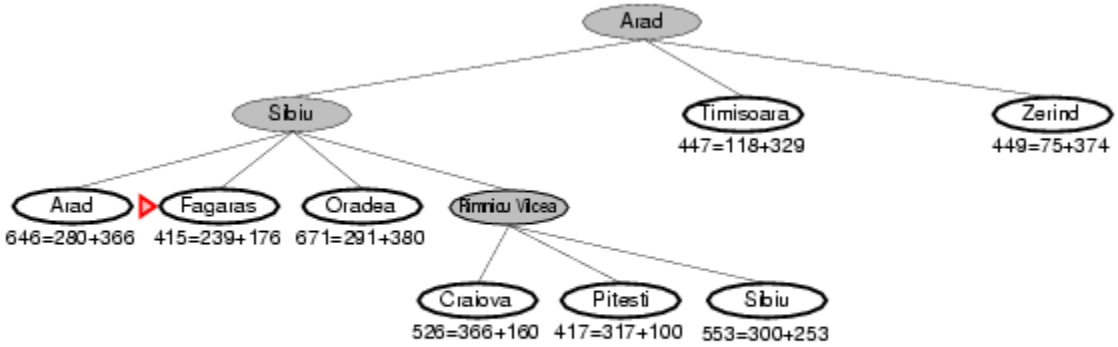
A* search example

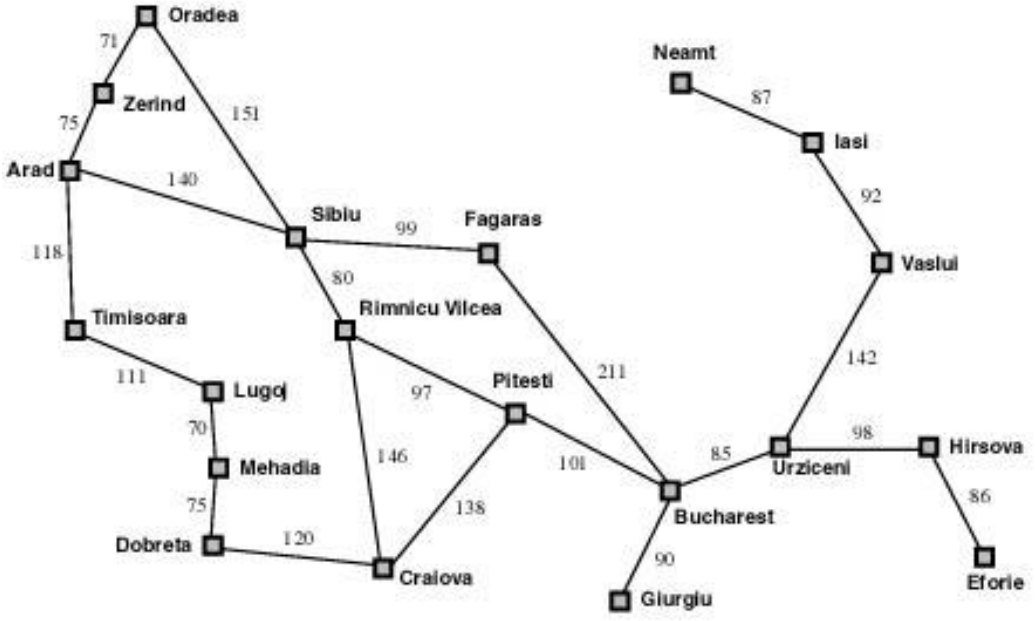




Straightline distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

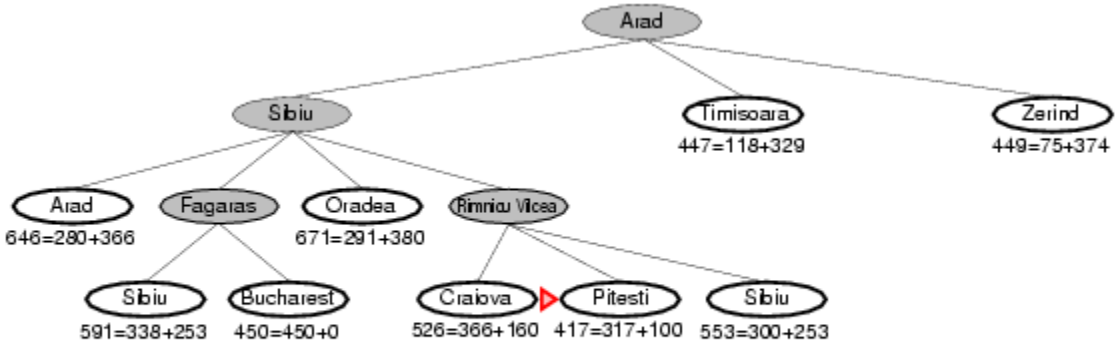
A* search example

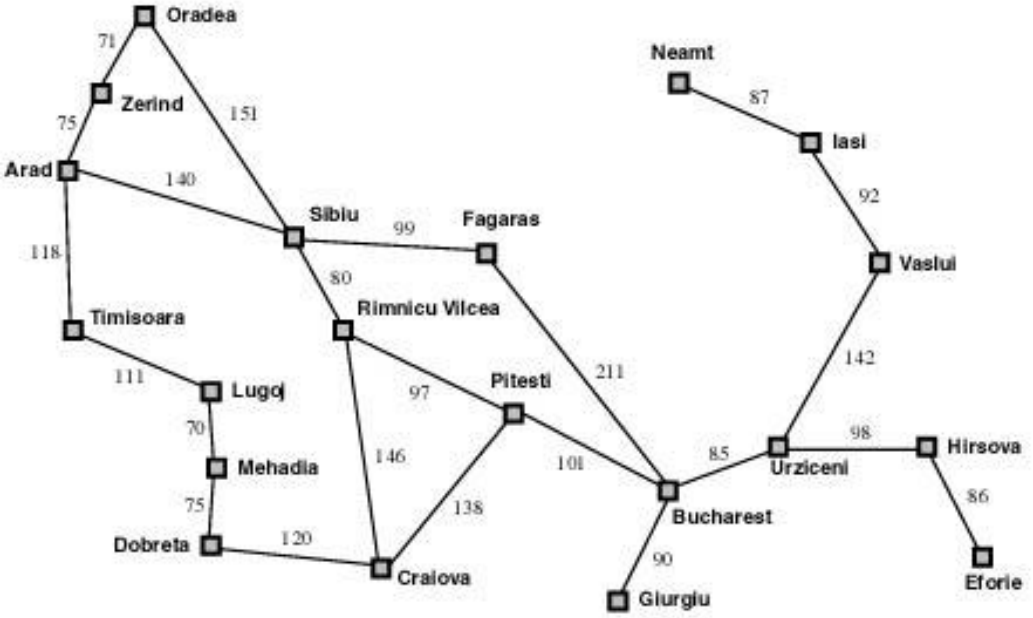




Straightline distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

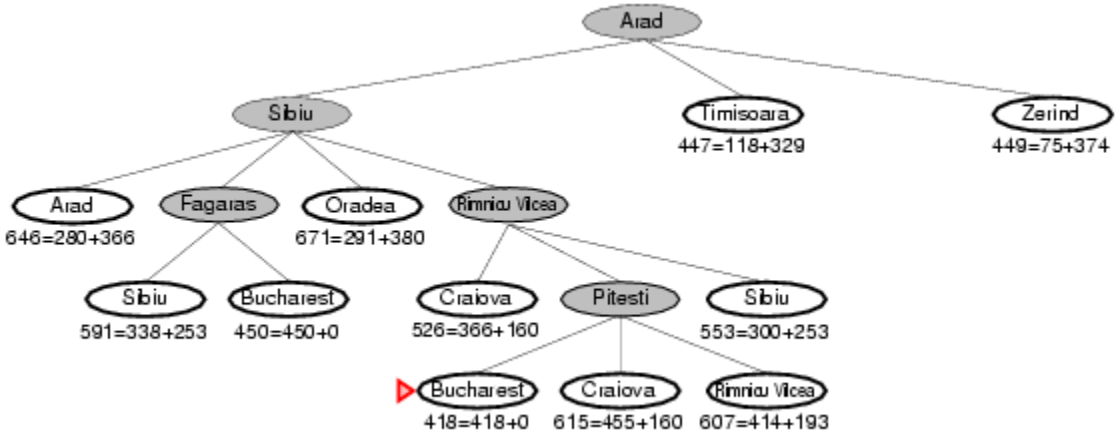
A* search example

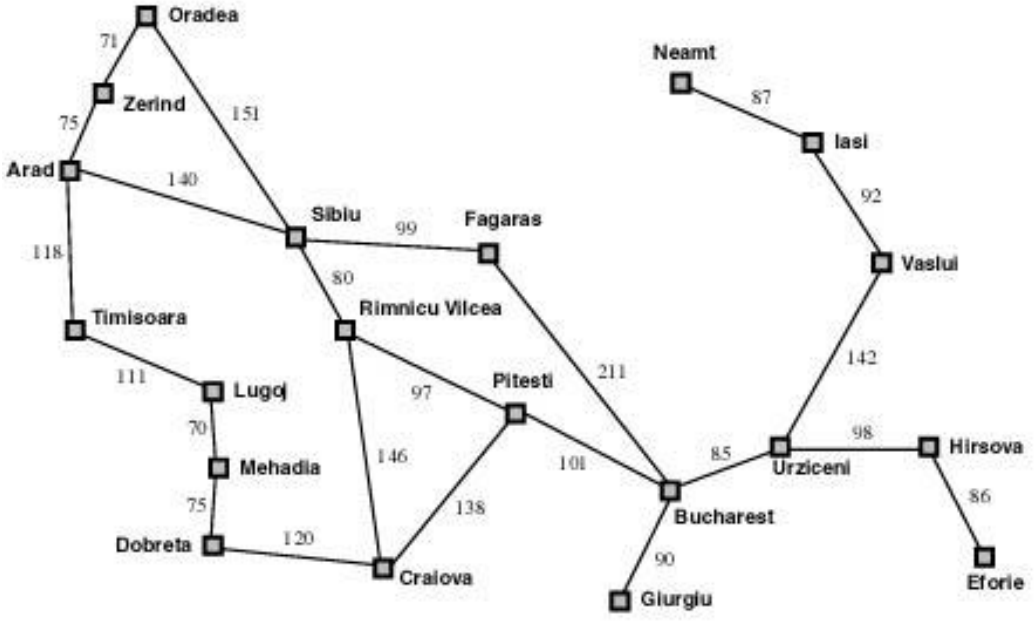




Straightline distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example





Straightline 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

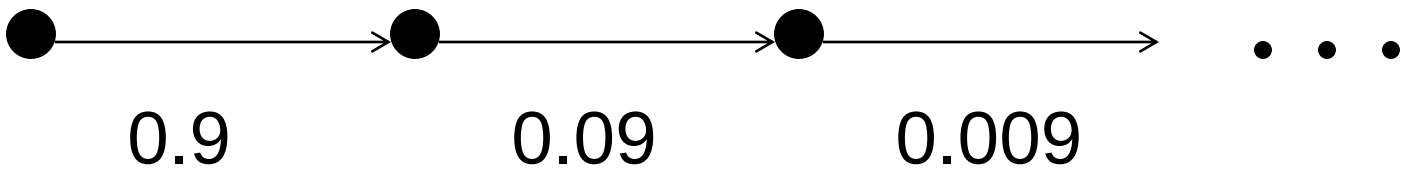
L' algoritmo A è completo

Teorema: L'algoritmo A con la condizione

$$g(n) \geq d(n) \cdot \varepsilon \qquad (\varepsilon > 0 \text{ costo minimo arco})$$

è *completo*.

Nota: la condizione ci garantisce che non si verifichino situazioni strane del tipo



e che il costo lungo un cammino non cresca "abbastanza".

Completezza di A: dimostrazione

- Sia [n_0 n_1 n_2 … n^*… n_k=*goal*] un cammino soluzione.
- Sia ***n**** un nodo della frontiera su un cammino soluzione: n^* prima o poi sarà espanso.
  - Infatti esistono solo un numero finito di nodi x che possono essere aggiunti alla frontiera con $f(x) \leq f(n^*)$;
- Quindi, se non si trova una soluzione prima, n^* verrà espanso e i suoi successori aggiunti alla frontiera. Tra questi anche il suo successore sul cammino soluzione.
- Il ragionamento si può ripetere fino a dimostrare che anche il nodo *goal* sarà selezionato per l'espansione

Algoritmo A*: la stima ideale

Funzione di valutazione ideale (*oracolo*):

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

$g^*(n)$ costo del cammino minimo da radice a n

$h^*(n)$ costo del cammino minimo da n a goal

$f^*(n)$ costo del cammino minimo da radice a goal, attraverso n

Normalmente:

$g(n) \geq g^*(n)$ e $h(n)$ è una stima di $h^*(n)$

Algoritmo A*: definizione

Definizione: euristica ammissibile

$\forall n \,.\, h(n) \leq h^*(n)$ *h è una sottostima*

Es. l'euristica della distanza in linea d'aria

Definizione: Algoritmo A*

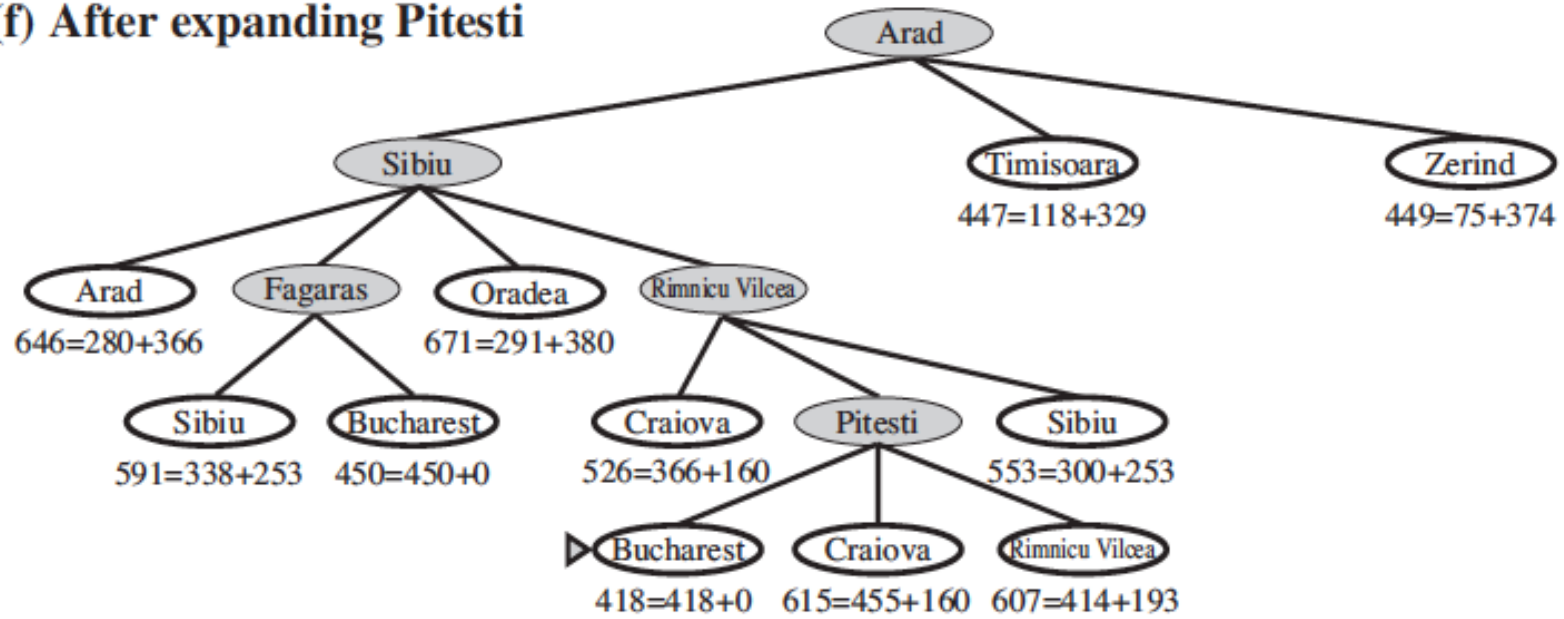
Un algoritmo A in cui *h* è una funzione euristica ammissibile.

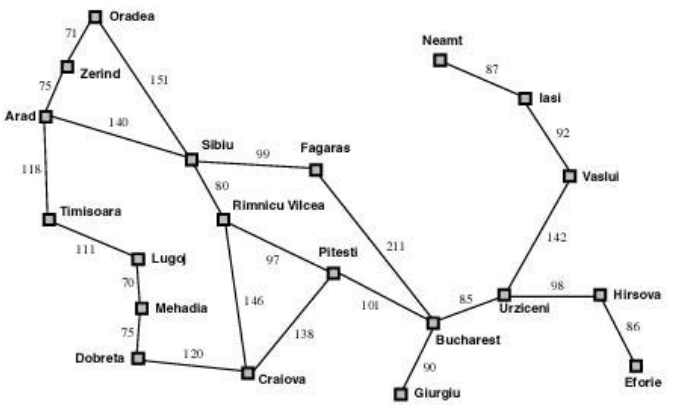
Teorema: gli algoritmi A* sono ottimali.

Corollario: BF e UC sono ottimali (*h(n)=0*)

Itinerario con A*

(f) After expanding Pitesti





Straightline 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

Proprietà di A*

- Completezza? Si (a meno che non vi sia un numero infinito di nodi con f ≤ *f(G)*)
-
- Tempo? Esponenziale
-
- Spazio? Mantiene tutti i nodi in memoria
-
- Ottimalità? Si
-

Osservazioni su A*

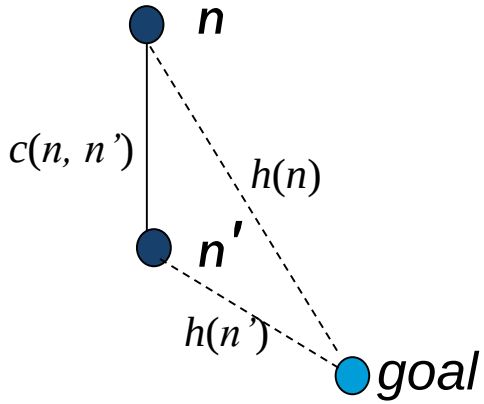
1. Una sottostima può farci compiere del lavoro inutile, però non ci fa perdere il cammino migliore
2. La componente *g* fa sì che si abbandonino cammini che vanno troppo in profondità
3. Una funzione che qualche volta sovrastima può farci perdere la soluzione ottimale

Ottimalità di A*

- Nel caso di ricerca su albero l'uso di un'euristica ammissibile è sufficiente a garantire l'ammissibilità.
- Nel caso di ricerca su grafo serve una proprietà più forte: la consistenza (detta anche monotonicità)

Euristica consistente o monotòna

- *Definizione*: euristica consistente
  - [h(goal) = 0]
  - $\forall n.\ h(n) \leq c(n, a, n') + h(n')$
    dove $n'=succ(n)$
  - Ne segue che $f(n) \leq f(n')$



- Nota: se *h* è consistente la *f* non decresce mai lungo i cammini, da cui il termine monotòna

Euristiche monotòne: proprietà

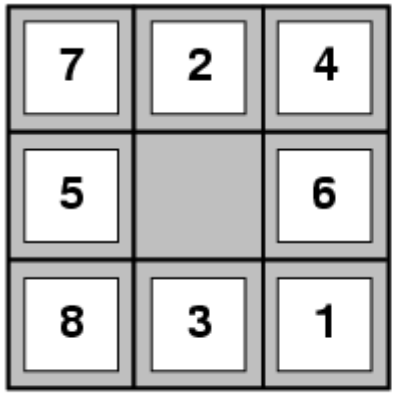
- *Teorema*: Un'euristica monotona è ammissibile
- Esistono euristiche ammissibili che non sono monotone, ma sono rare.
- Le euristiche monotone garantiscono che la soluzione meno costosa venga trovata per prima e quindi sono ottimali anche nel caso di *ricerca su grafo.*

Euristiche ammissibili: esempi

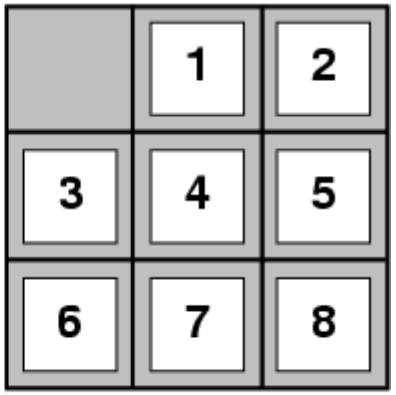
E.g., il gioco dell'8:

- $h_1(n)$ = numero di caselle fuori posto
- $h_2(n)$ = la Manhattan distance totale

(i.e., # caselle di distanza dalla posizione finale di ogni casella)



Start State



Goal State

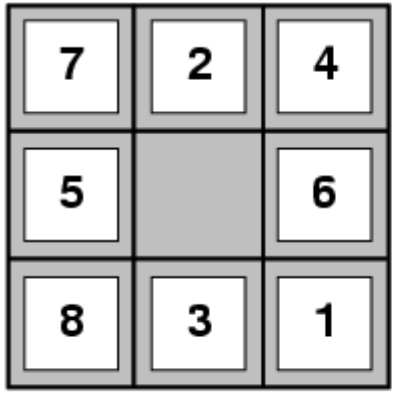
- $h_1(S) = ?$
- $h_2(S) = ?$

Euristiche ammissibili: esempi

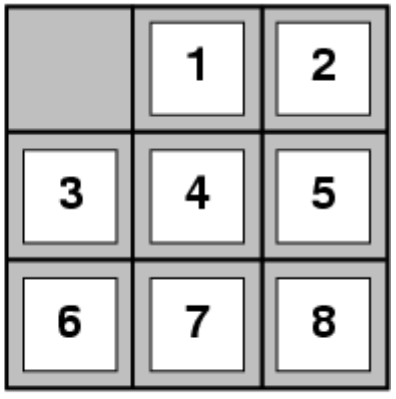
E.g., il gioco dell'8:

- $h_1(n)$ = numero di caselle fuori posto
- $h_2(n)$ = la Manhattan distance totale

(i.e., # caselle di distanza dalla posizione finale di ogni casella)

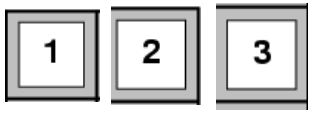


Start State



Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3 +1 +2 +2+2+3+3+2 = 18



Ottimalità di A*

1. Se $h(n)$ è consistente i valori di $f(n)$ lungo un cammino sono non decrescenti

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

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

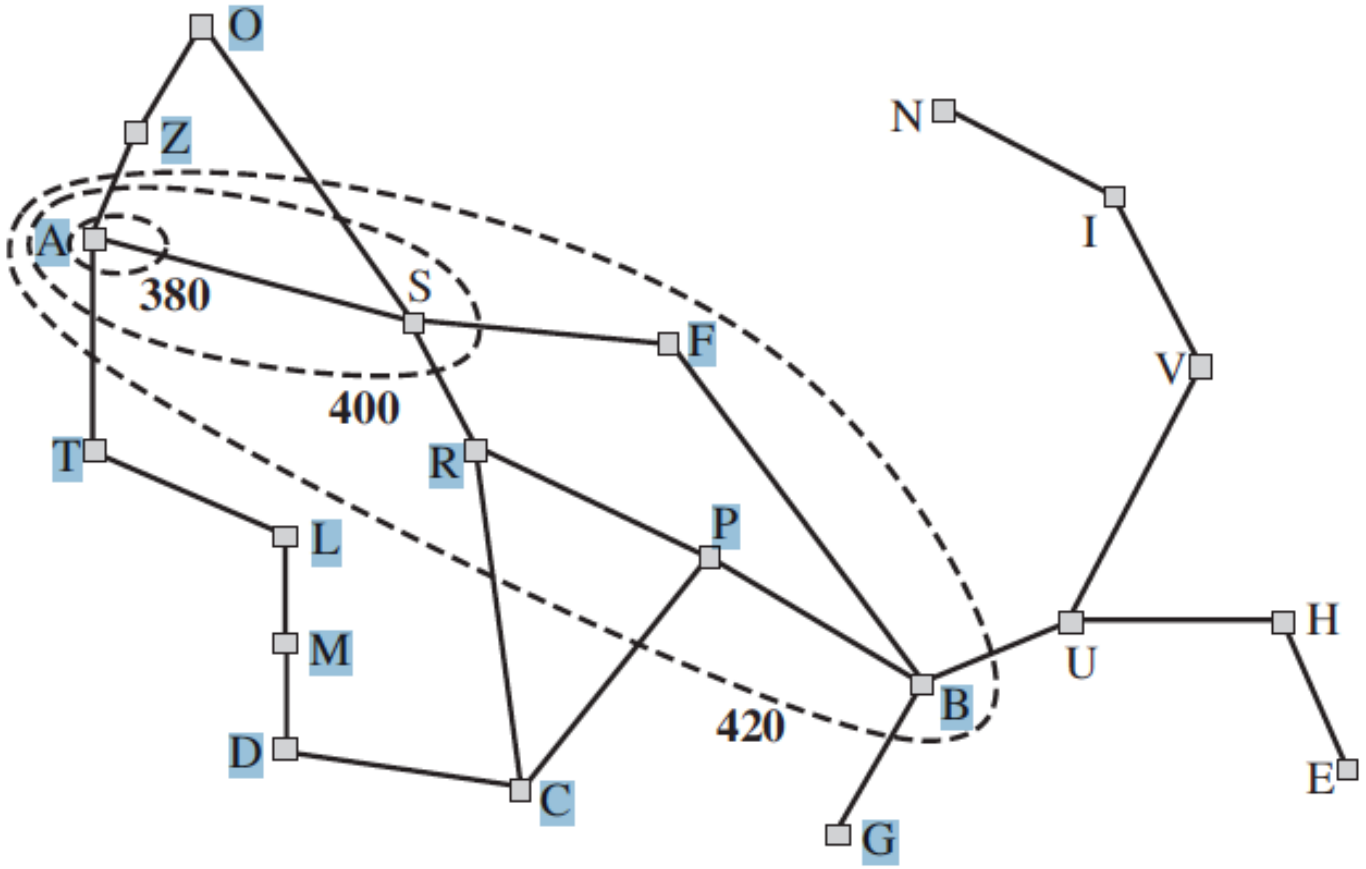
   ma siccome $g(n) + c(n, a, n') = g(n')$

   allora $f(n) \leq f(n')$

2. Ogni volta che A* seleziona un nodo per l'espansione, il cammino ottimo a tale nodo è stato trovato

   se così non fosse ci sarebbe un altro nodo n' sulla frontiera sul cammino ottimo con $f(n')$ minore; ma ciò non è possibile perché tale nodo sarebbe già stato espanso

I contorni nella ricerca A*



Bilancio su A*

- A* è completo: discende dalla completezza di A (A* è un algoritmo A particolare)
- A* con euristica monotona è ottimale
- A* è ottimamente efficiente: a parità di euristica nessun altro algoritmo espande meno nodi (senza rinunciare a ottimalità)
- Qual è il problema?
- ... ancora l'occupazione di memoria ($O(b^{d+1})$)

Migliorare l’occupazione di memoria

- Beam search
- A* con approfondimento iterativo (IDA*)
- Ricerca best-first ricorsiva (RBFS)
- A* con memoria limitata (MA*) in versione semplice (SMA*)

Beam search

- Nel Best First viene tenuta tutta la frontiera; se l'occupazione di memoria è eccessiva si può ricorrere ad una variante: la *Beam search*.
- La *Beam Search* tiene ad ogni passo solo *i k nodi più promettenti*, dove *k* è detto *l'ampiezza del raggio (beam).*
- La *Beam Search* non è completa.

Beam Search

- L'algoritmo (grado *w*)

ALGORITHM 12.5: *THE BEAM SEARCH ALGORITHM*

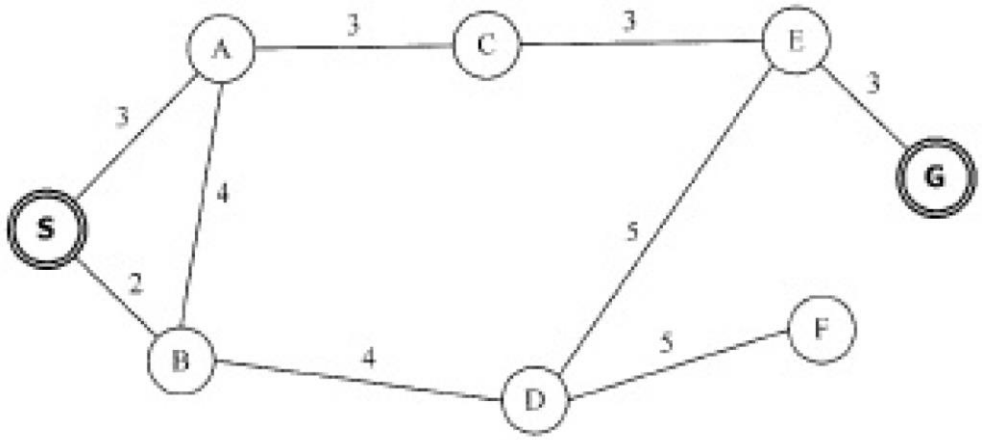
Step 1: Initialization: Put *S* in the *OPEN* list and create an initially empty *CLOSE* list.
Step 2: If the *OPEN* list is empty, exit and declare failure.
Step 3: $\forall N \in OPEN$ do
3a. Pop up node *N* in the *OPEN* list, remove it from the *OPEN* list and put it into the *CLOSE* list.
3b. If node *N* is a goal node, exit successfully with the solution obtained by tracing back the path along the pointers from *N* to *S*.
3c. Expand node *N* by applying a successor operator to generate the successor set *SS*(*N*) of node *N*. Be sure to eliminate the successors, which are ancestors of *N*, from *SS*(*N*).
3d. $\forall v \in SS(N)$ Create a pointer pointing to *N* and push it into *Beam-Candidate* list.
Step 4: Sort the *Beam-Candidate* list according to the heuristic function $f(N)$ so that the best *w* nodes can be pushed into the the *OPEN* list. Prune the rest of nodes in the *Beam-Candidate* list.
Step 5: Go to Step 2.

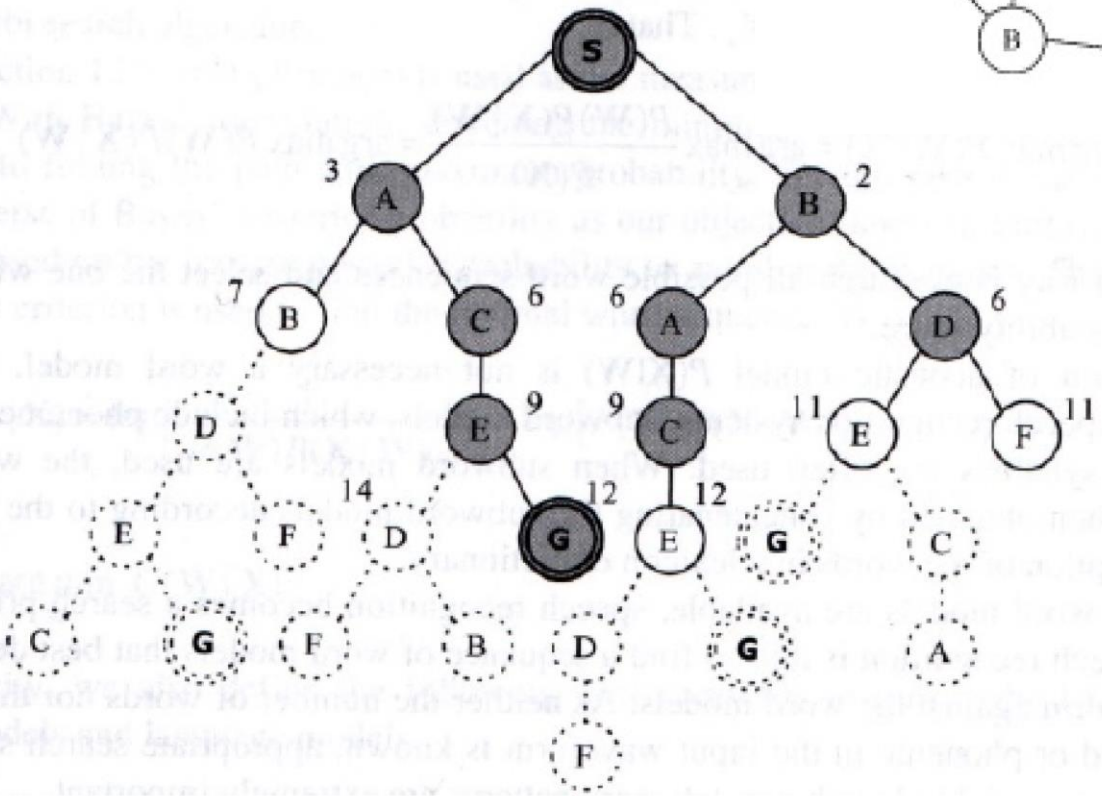
Beam search

Algorithm 1: Beam Search Algorithm

Data: Graph (G), start node (s), goal node (g), beam width (β)
Result: Path with lowest cost
Function beamSearch(G, s, g, β)
    $openList \leftarrow s$
    $closedList \leftarrow empy\ list$
    $path \leftarrow empy\ list$
    while *open list is not empty* **do**
        $b \leftarrow best\ node\ from\ openList$
        $openList.remove(b)$
        $closedList.add(b)$
        if b *is* g **then**
            $path.add(b)$
            return *path*
        end
        $N \leftarrow neighbors(b)$
        for n *in* N **do**
            if n *is in neither closedList nor openList* **then**
                $openList.add(n)$
            else if n *is in openList* **then**
                if *path with current parent* $\leq$ *path with old parent* **then**
                    Replace parents of n
                end
            else if n *is not in closedList* **then**
                $openList.add(n)$
            end
        end
        if *number of nodes in openList* $> \beta$ **then**
            $openList \leftarrow best\ \beta\ nodes\ in\ openList$
        end
    end
    return *path*
end

An example (*w*=2)





Applications: S2S modeling

- Sequence-to-sequence (S2S) tasks

Machine translation:

kare wa ringo wo tabeta → he ate an apple

Tagging:

he ate an apple → PRN VBD DET PP

Dialog:

he ate an apple → good, he needs to slim down

Speech Recognition:

 → he ate an apple

And just about anything...:

1010000111101 → 00011010001101

Figure 1: An example of sequence-to-sequence modeling tasks.

i am from pittsburgh .
i study at a university .
my mother is from utah .

$P(e_2=am \mid e_1=i) = c(e_1=i, e_2=am)/c(e_1=i) = 1 / 2 = 0.5$

$P(e_2=study \mid e_1=i) = c(e_1=i, e_2=study)/c(e_1=i) = 1 / 2 = 0.5$

Figure 3: An example of calculating probabilities using maximum likelihood estimation.

Application: Sequence decoding

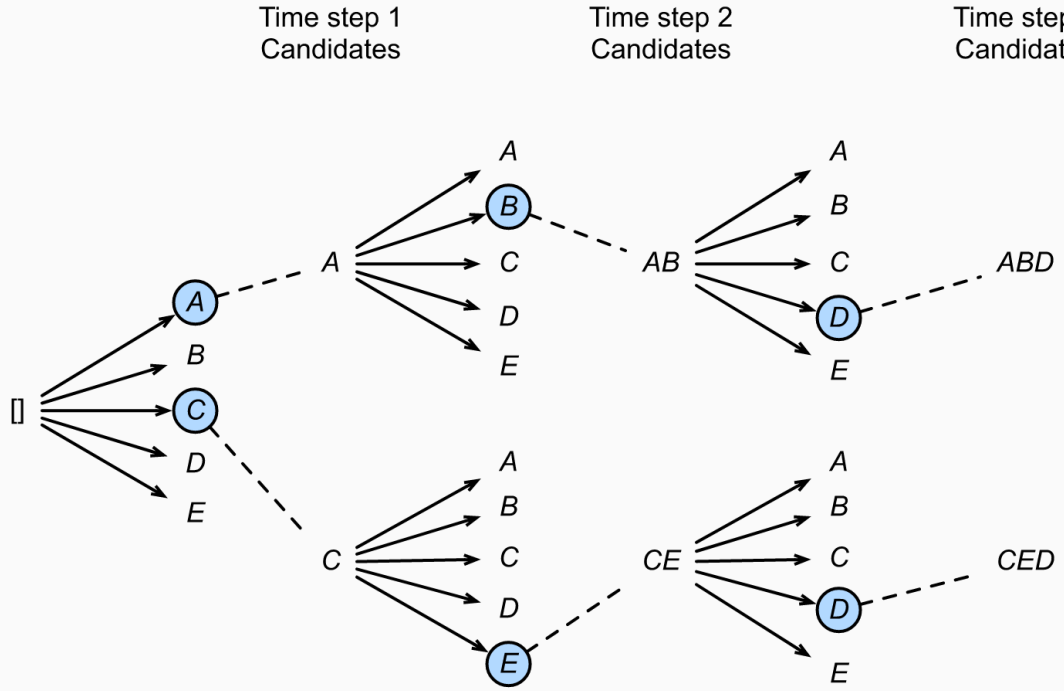


Fig. 9.8.3 The process of beam search (beam size: 2, maximum length of an output sequence: 3). The candidate output sequences are A, C, AB, CE, ABD, and CED.

Application: speech recognition o MT

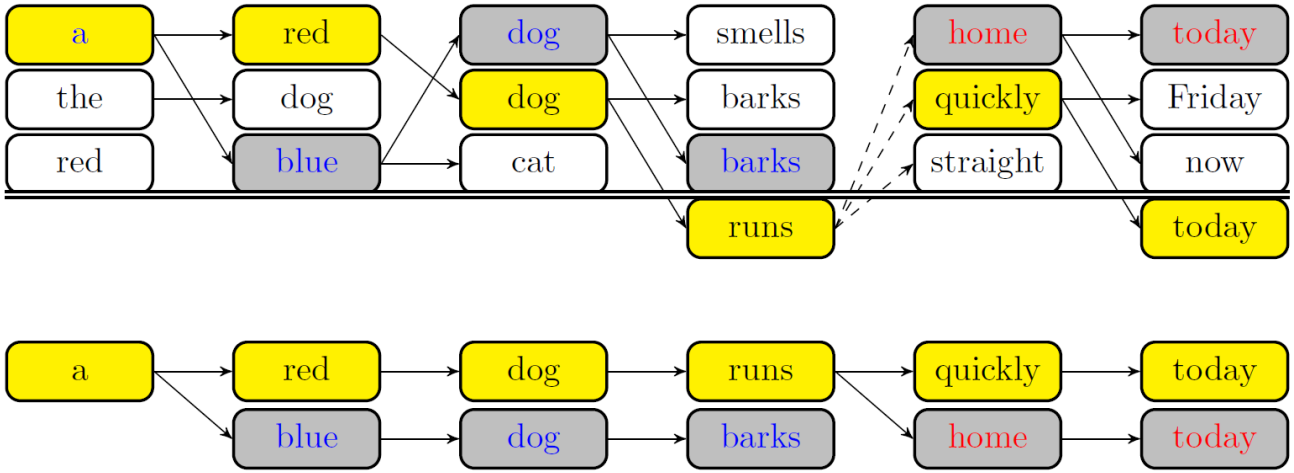
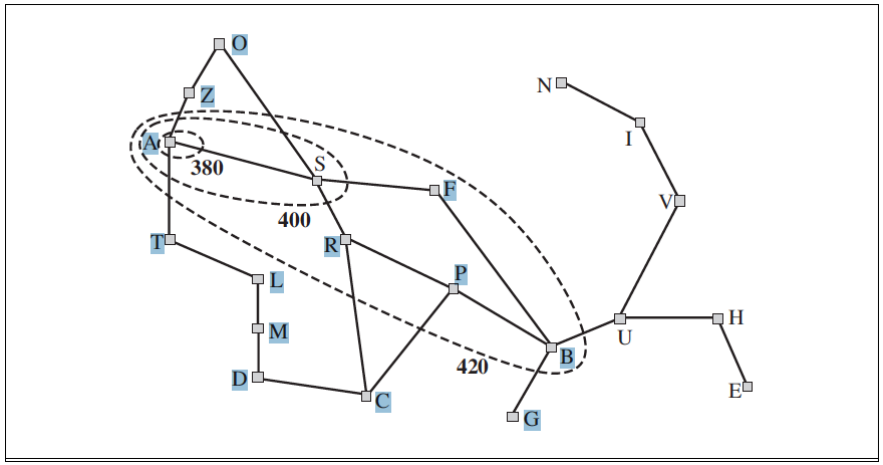


Figure 1: Top: possible $\hat{y}_{1:t}^{(k)}$ formed in training with a beam of size $K = 3$ and with gold sequence $y_{1:6}$ = "a red dog runs quickly today". The gold sequence is highlighted in yellow, and the predicted prefixes involved in margin violations (at $t = 4$ and $t = 6$) are in gray. Note that time-step $T = 6$ uses a different loss criterion. Bottom: prefixes that actually participate in the loss, arranged to illustrate the back-propagation process.

IDA*

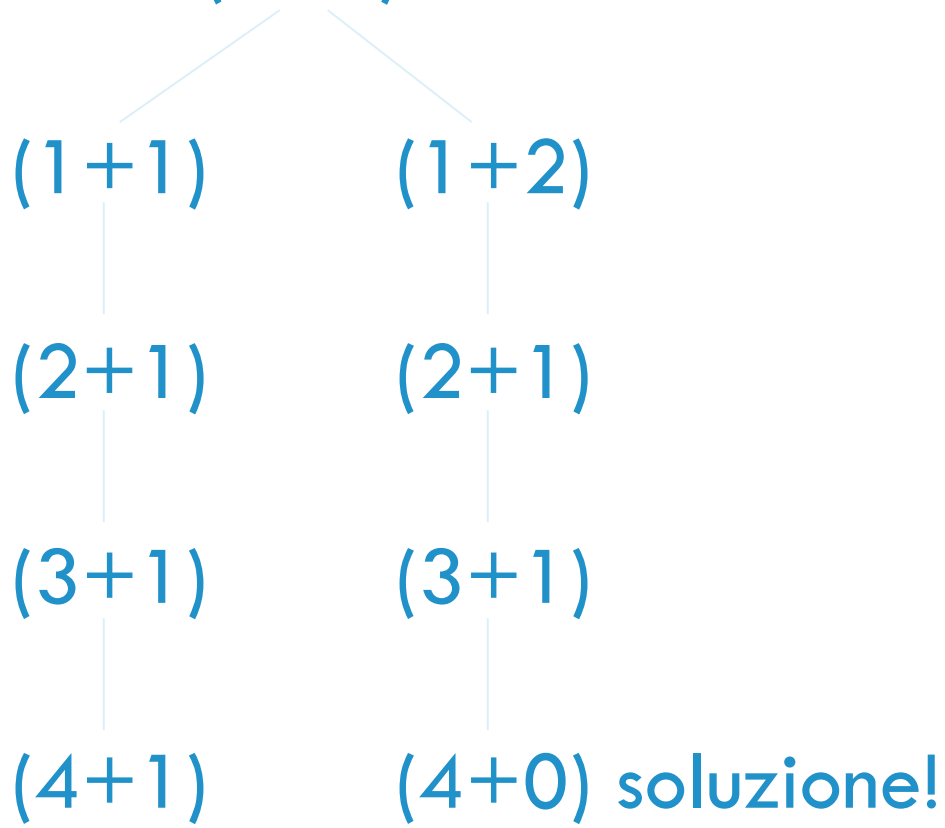
A con approfondimento iterativo*

- IDA* combina A* con ID: ad ogni iterazione si ricerca in profondità con un limite dato dal valore della funzione *f* (e non dalla profondità)
- il limite, detto *f-limit,* viene aumentato ad ogni iterazione, fino a trovare la soluzione.
- Ogni nodo all'interno di un perimetro ha valore di *f* minore o uguale a *f-limit*



Esempio

Iteraz. 4



f-limit=4

Quale incremento?

- Cruciale la scelta dell'incremento per garantire l’ottimalità
  - Nel caso di costo delle azioni fisso è chiaro: il limite viene incrementato del costo delle azioni.
  - Nel caso che i costi delle azioni siano variabili, si potrebbe ad ogni passo fissare il limite successivo al valore minimo delle *f* scartate (in quanto superavano il limite) all’iterazione precedente.
    - La più piccola *f* tra tutte quelle scartate

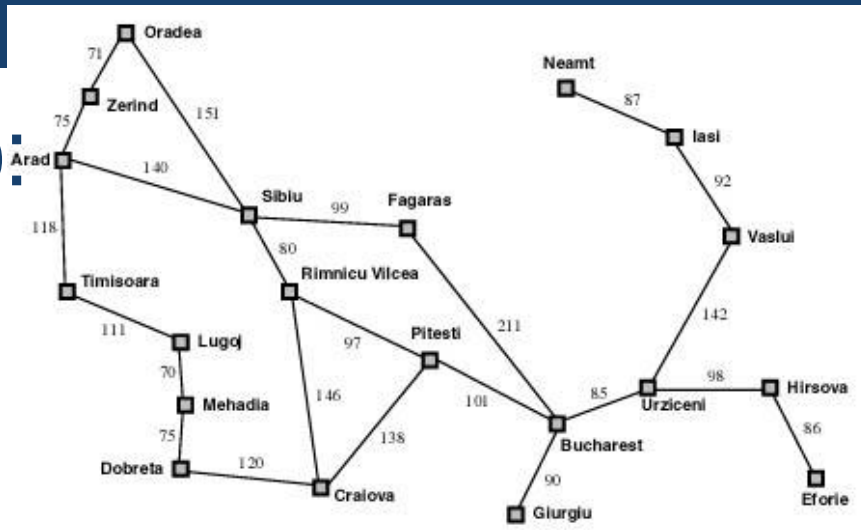
Analisi IDA*

- IDA* è completo e ottimale
  - Se le azioni hanno costo costante k (caso tipico 1) e *f-limit* viene incrementato di k
  - Se le azioni hanno costo variabile e l'incremento di *f-limit* è $\leq \varepsilon$ (minimo costo degli archi)
  - Se il nuovo *f-limit* = min. valore *f dei nodi generati ed* esclusi all'iterazione precedente
- Occupazione di memoria O(bd), come l'algoritmo DF, *//ogni volta tengo in OpenList solo i nodi di un cammino*

Best-first ricorsivo

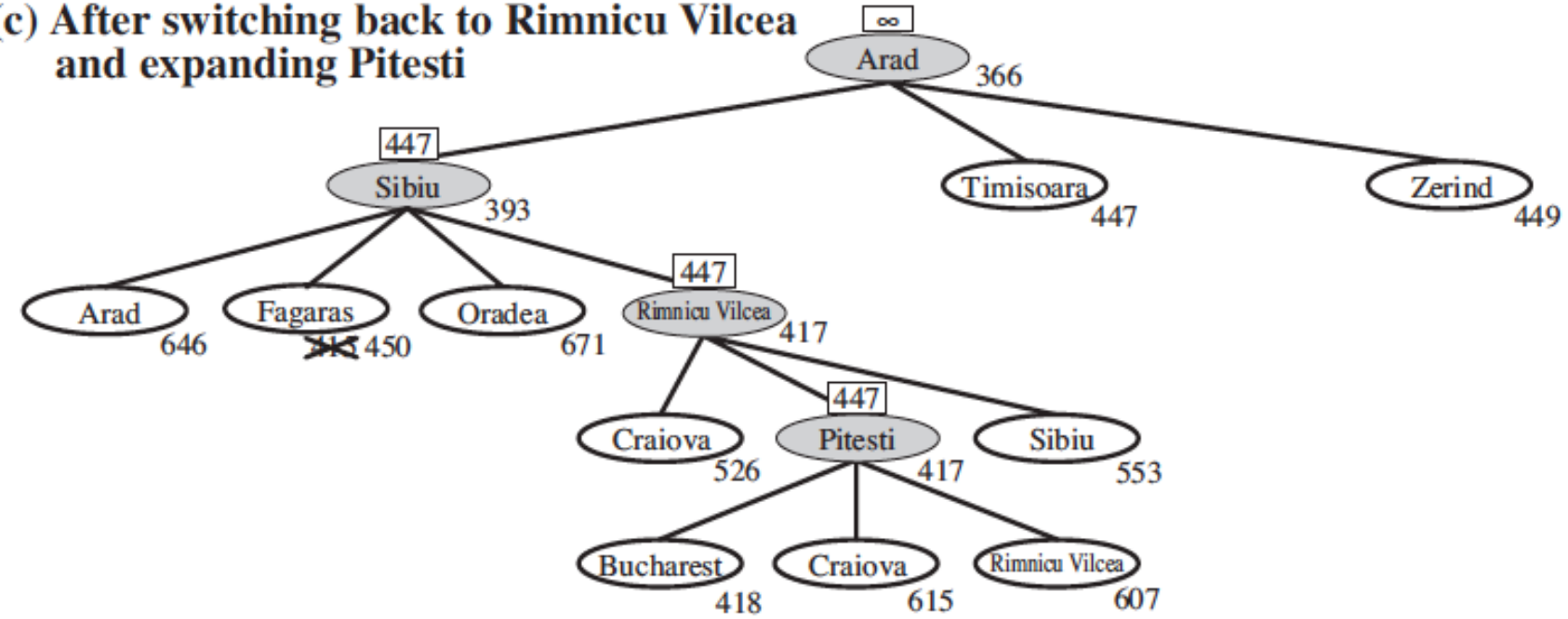
- Simile a DF ricorsivo: cerca di usare meno memoria, facendo del lavoro in più
- Tiene traccia ad ogni livello del migliore percorso alternativo
- Invece di fare backtracking in caso di fallimento interrompe l'esplorazione quando trova un nodo meno promettente (secondo *f*)
- Nel tornare indietro si ricorda il miglior nodo che ha trovato nel sottoalbero esplorato, per poterci eventualmente tornare

Best first ricorsivo: esempio



Straightline 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

(c) After switching back to Rimnicu Vilcea and expanding Pitesti



Best First ricorsivo: algoritmo

```
function Ricerca-Best-First-Ricorsiva(problema)
    returns soluzione oppure fallimento
    return RBFS(problema, CreaNodo(problema.Stato-iniziale), ∞)        // all'inizio f-limite è un valore molto grande

function RBFS (problema, nodo, f-limite)
    returns soluzione oppure fallimento e un nuovo limite all' f-costo      // restituisce due valori
    if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
    successori = [ ]
    for each azione in problema.Azioni(nodo.Stato) do
            aggiungi Nodo-Figlio(problema, nodo, azione) a successori     // genera i successori
    if successori è vuoto then return fallimento, ∞
    for each s in successori do        // valuta i successori
            s.f = max(s.g + s.h, nodo.f)
    loop do
            migliore = il nodo con f minimo tra i successori
            if migliore.f > f_limite then return fallimento, migliore.f
            alternativa = il secondo nodo con f minimo tra i successori
            risultato, migliore.f = RBFS(problema, migliore, min(f_limite, alternativa))
            if risultato ≠ fallimento then return risultato
```

A* con memoria limitata

Versione semplice

- L'idea è quella di utilizzare al meglio la memoria disponibile
- SMA* (Simplified Memory-bounded A*) procede come A* fino ad esaurimento della memoria disponibile
- A questo punto “dimentica” il nodo peggiore, dopo avere aggiornato il valore del padre.
- A parità di *f* si sceglie il nodo migliore più recente e si dimentica il nodo peggiore più vecchio.
- Ottimale se il cammino soluzione sta in memoria.

Considerazioni

- In algoritmi a memoria limitata (IDA* e SMA*) le limitazioni della memoria possono portare a compiere molto lavoro inutile
- Difficile stimare la complessità temporale effettiva
- *Le limitazioni di memoria possono rendere un problema intrattabile dal punto di vista computazionale*

Valutazione di funzioni euristiche

A parità di ammissibilità, una euristica può essere più efficiente di un'altra nel trovare il cammino soluzione migliore (visitare meno nodi): dipende da quanto *informata* è (o *dal grado di informazione posseduto)*

- $h(n)=0$ minimo di informazione (BF o UF)
- $h^*(n)$ massimo di informazione (oracolo)

In generale, per le euristiche ammissibili:

$$0 \leq h(n) \leq h^*(n)$$

Più informata, più efficiente

Teorema: Se $h_1 \leq h_2$, i nodi espansi da A* con h_2 sono un sottoinsieme di quelli espansi da A* con h_1.

Se $h_1 \leq h_2$, A* con h_2 è almeno efficiente quanto A* con h_1

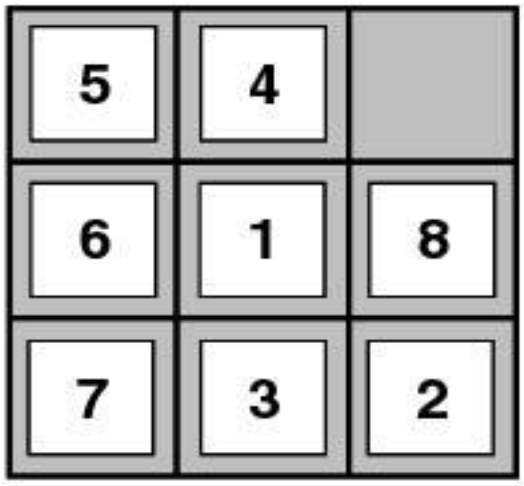
- Un'euristica più informata riduce lo spazio di ricerca (è più efficiente), ma è tipicamente più costosa da calcolare

Confronto di euristiche ammissibili

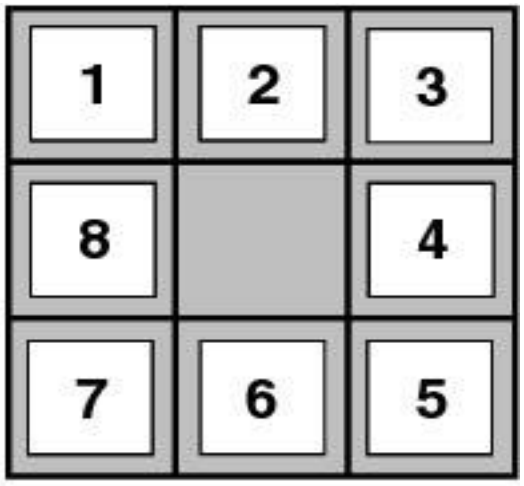
- Due euristiche ammissibili per il gioco dell'8

  h_1: conta il numero di caselle fuori posto

  h_2: somma delle distanze Manhattan delle caselle fuori posto dalla posizione finale
- *h_2 è più informata di h_1* infatti $\forall n \,.\, h_1(n) \leq h_2(n)$



Start State

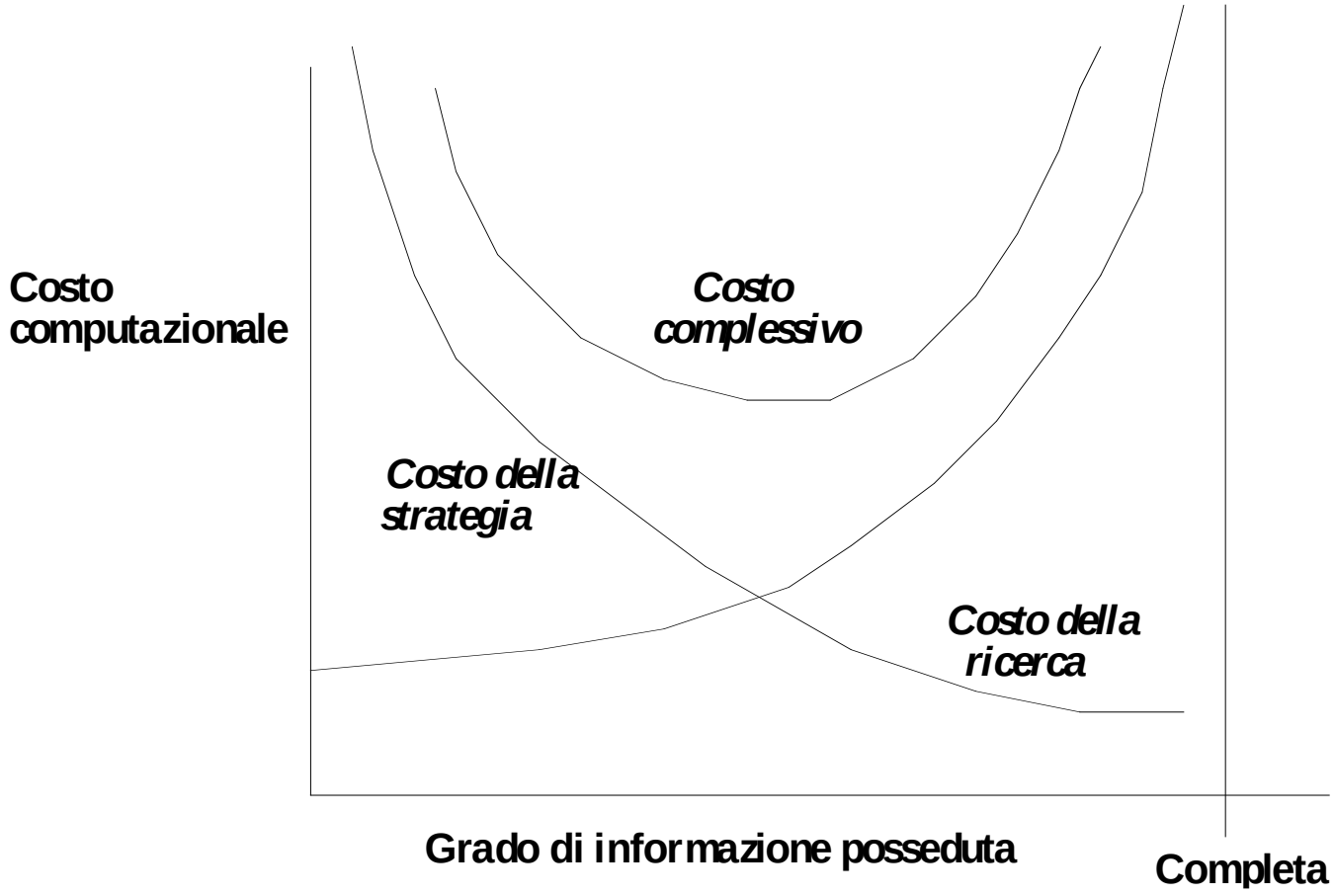


Goal State

$h1 = 7$

$h2 = 4+2+2+2+$
$+2+0+3+3 = 18$

Costo ricerca vs costo euristica



[figura da Nilsson 1980]

Misura del potere euristico

Come valutare gli algoritmi di ricerca euristica ...

*Fattore di diramazione effettivo b**

N: numero di nodi generati

d: profondità della soluzione

Esempio:

d=5; N= 52

b*= 1.92

*b** è il fattore di diramazione di un albero uniforme con N+1 nodi; soluzione dell'equazione

$$N + 1 = b^* + (b^*)^2 + \ldots + (b^*)^d$$

Sperimentalmente una buona euristica ha un *b** abbastanza vicino a 1 (< 1.5)

Esempio: dal gioco dell'otto

d	IDS	A*(h1)	A*(h2)
2	10 (2,43)	6 (1,79)	6 (1,79)
4	112 (2,87)	13 (1,48)	12 (1,45)
6	680 (2,73)	20 (1,34)	18 (1,30)
8	6384 (2,80)	39 (1,33)	25 (1,24)
10	47127 (2,79)	93 (1,38)	39 (1,22)
12	3644035 (2,78)	227 (1,42)	73 (1,24)
14	-	539 (1,44)	113 (1,23)
…	-	…	…

I dati sono mediati, per ogni d, su 100 istanze del problema [AIMA]

Nodi generati e fattore di diramazione effettivo

Capacità di esplorazione

Con b=2

d=6	N=100
d=12	N=10.000

ma con b=1.5

d=12	N=100
d=24	N=10.000

… migliorando di poco l'euristica si riesce, a parità di nodi espansi, a raggiungere una profondità doppia!

Quindi …

1. Tutti i problemi dell'IA (o quasi) sono di complessità esponenziale … ma c'è esponenziale e esponenziale!
2. L'euristica può migliorare di molto la capacità di esplorazione dello spazio degli stati rispetto alla ricerca cieca
3. Migliorando anche di poco l'euristica si riesce ad esplorare uno spazio molto più grande.

Come si inventa un'euristica?

- Alcune strategie per ottenere euristiche ammissibili:
  - Rilassamento del problema
  - Massimizzazione di euristiche
  - Database di pattern disgiunti
  - Combinazione lineare
  - Apprendere dall'esperienza

Rilassamento del problema

- Nel gioco dell'8 mossa da A a B possibile se ...
  1. B adiacente a A
  2. B libera
- h_1 e h_2 sono calcoli della *distanza esatta* della soluzione in versioni semplificate del puzzle:
  - h_1 (nessuna restrizione): sono sempre ammessi scambi a piacimento tra caselle $\rightarrow$ # caselle fuori posto
  - h_2 (solo restrizione 1): sono ammessi spostamenti anche su caselle occupate, purché adiacenti $\rightarrow$ somma delle distanze Manhattan

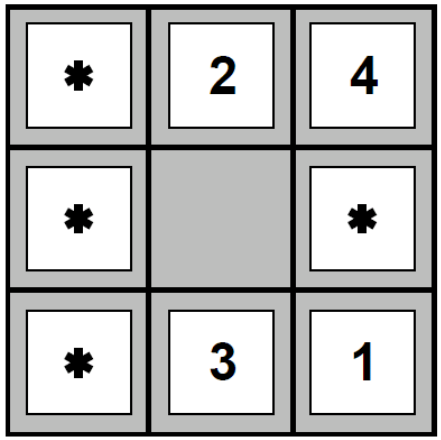
Massimizzazione di euristiche

- Se si hanno una serie di euristiche ammissibili h_1, h_2, … h_k senza dominazione tra queste allora conviene prendere il massimo dei loro valori:

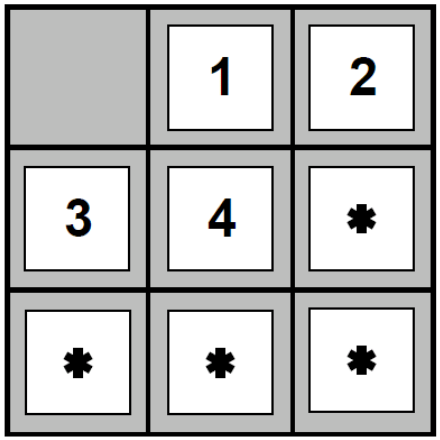
$$h(n)=\max(h_1(n), h_2(n), \ldots, h_k(n))$$

- Se le h_i sono ammissibili anche la h lo è
- La h domina tutte le altre.

Euristiche da sottoproblemi



Start State



Goal State

- Costo della soluzione ottima al sottoproblema (di sistemare 1,2,3,4) è una sottostima del costo per il problema nel suo complesso
- *Database di pattern*: memorizzare ogni istanza del sottoproblema con relativo costo
- Usare questo database per calcolare h_{DB}

Sottoproblemi multipli

- Potremmo poi fare la stessa cosa per altri sottoproblemi: 5-6-7-8, 2-4-6-8 … ottenendo altre euristiche ammissibili
- Poi prendere il valore massimo: ancora una euristica ammissibile
- Ma potremmo sommarle e ottenere un’euristica ancora più accurata?

Pattern disgiunti

- In generale no perchè le soluzioni ai sottoproblemi interferiscono e la somma delle euristiche in generale non è ammissibile
- Si deve eliminare il costo delle mosse che contribuiscono all'altro sottoproblema
- Database di pattern *disgiunti* consentono di sommare i costi (euristiche additive)
- Sono molto efficaci: gioco del 15 in pochi ms

Apprendere dall'esperienza

- Far girare il programma, raccogliere dati: coppie <*stato*, *h**>
- Usare i dati per apprendere a predire la *h* con algoritmi di apprendimento *induttivo*
- Gli algoritmi di apprendimento si concentrano su caratteristiche salienti dello stato (*feature*)

Combinazione di euristiche

- Quando diverse caratteristiche influenzano la bontà di uno stato, si può usare una combinazione lineare

$$h(n) = c_1\, h_1(n) + c_2\, h_2(n) + \ldots + c_k\, h_k(n)$$

Gioco dell'8:

$h(n) = c_1$ #fuori-posto + c_2 #coppie-scambiate

Scacchi:

$h(n) = c_1$ vant-pezzi + c_2 pezzi-attacc. + c3 regina + …

- Il peso dei coefficienti può essere aggiustato con l'esperienza, anche automaticamente

SummarAIzing (1): *uninformed*

- La qualità di un algoritmo di ricerca è misurata in base alla sua completezza, ottimalità, complessità spaziale e temporale (*b*, il branching factor nello spazio degli stati, and *d*, la profondità della prima soluzione trovata)
- Metodi di **Uninformed search**:
  - **Breadth-first search (**espande i nodi meno profondi prima); è completo, ottimo per costi unitary di passo, ma ha una complessità spaziale esponenziale.
  - **Uniform-cost search (**ottimo per il costo del passo generico)
  - **Depth-first search** espande i nodi più profondi prima (nè completo nè ottimo, ma ha una complessità spaz. lineare. (**Depth-limited search** aggiunge un bound alla profondità**)**.
  - **Iterative deepening search** è un Depth-limited search con profondità crescenti fino al goal: completo, ottimo per costi di step unitari, complessità simile a breadth-first search, complessità spaziale lineare.
  - **Bidirectional search** può ridurre enormemente la time complexity: non sempre applicabile poichè richiede molto spazio di memoria.

SummarAIzing (2)

I metodi di ricerca informata possono avere accesso a funzioni **euristiche** *h(n)* che stimano il costo delle soluzioni accessibili a partire da *n*.

- L'algoritmo **best-first** seleziona un nodo per l'espansione sulla base di una **funzione di valutazione**.
- La **Greedy best-first search** espande i nodi che hanno *h(n)* minima. Non è ottimale ma è spesso efficente.
- **A∗ search** espande i nodi che minimizzano *f(n)* = *g(n)* + *h(n)*. A∗ è complete e ottimo, se *h(n)* è ammissibile (per il TREE-SEARCH) o consistente (per il GRAPH-SEARCH). La complessità spaziale di A∗ è però ancora proibitiva.

SummarAIzing (3)

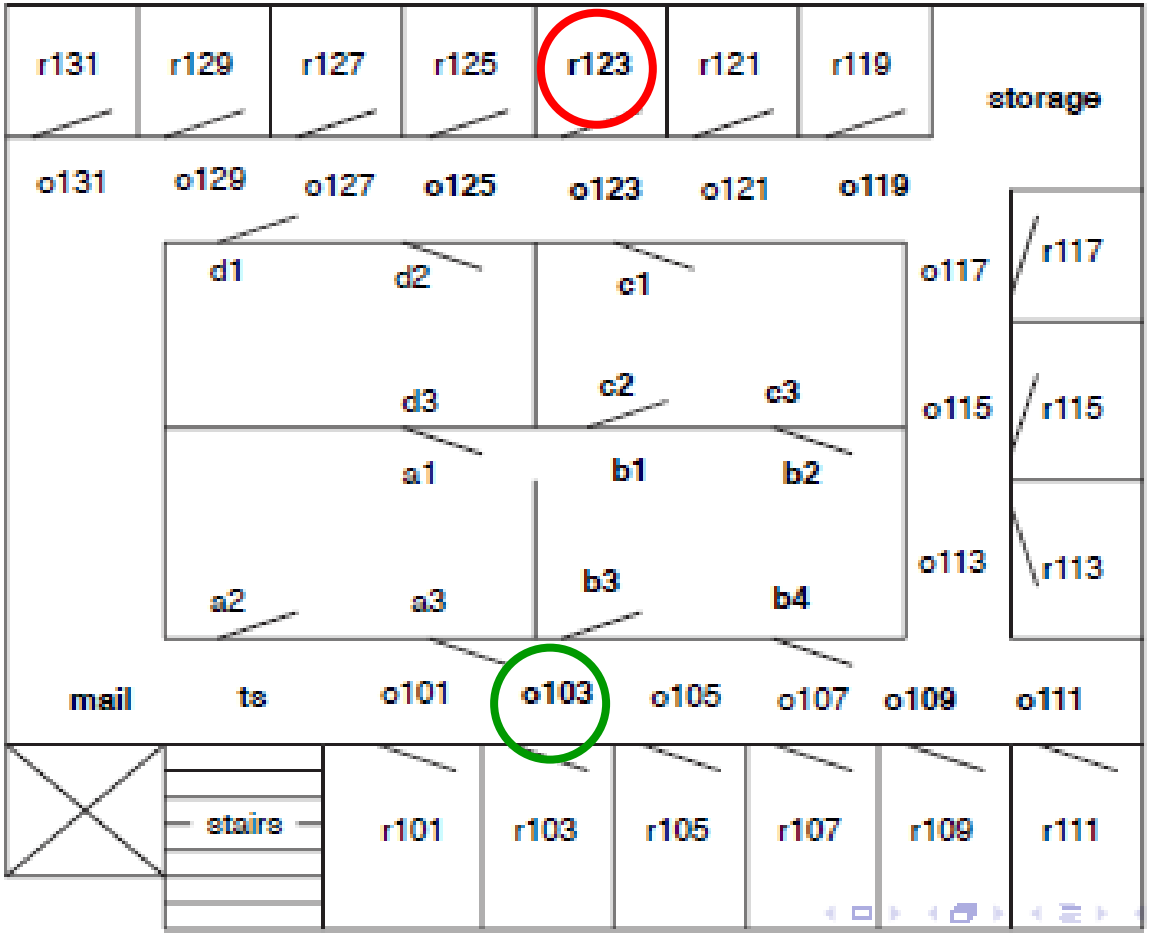
La prestazione degli algoritmi di ricera euristica dipende dalla **qualità della funzione euristica**.

Direzioni interessanti per la scelta di una funzione euristica:

- **rilassamento** della definizione del problema
- **memorizzazione di soluzioni a sottoproblemi** ricorrenti con il chaching dei loro costi precomputati
- **apprendimento dall'esperienza** nella stessa classe di problemi.

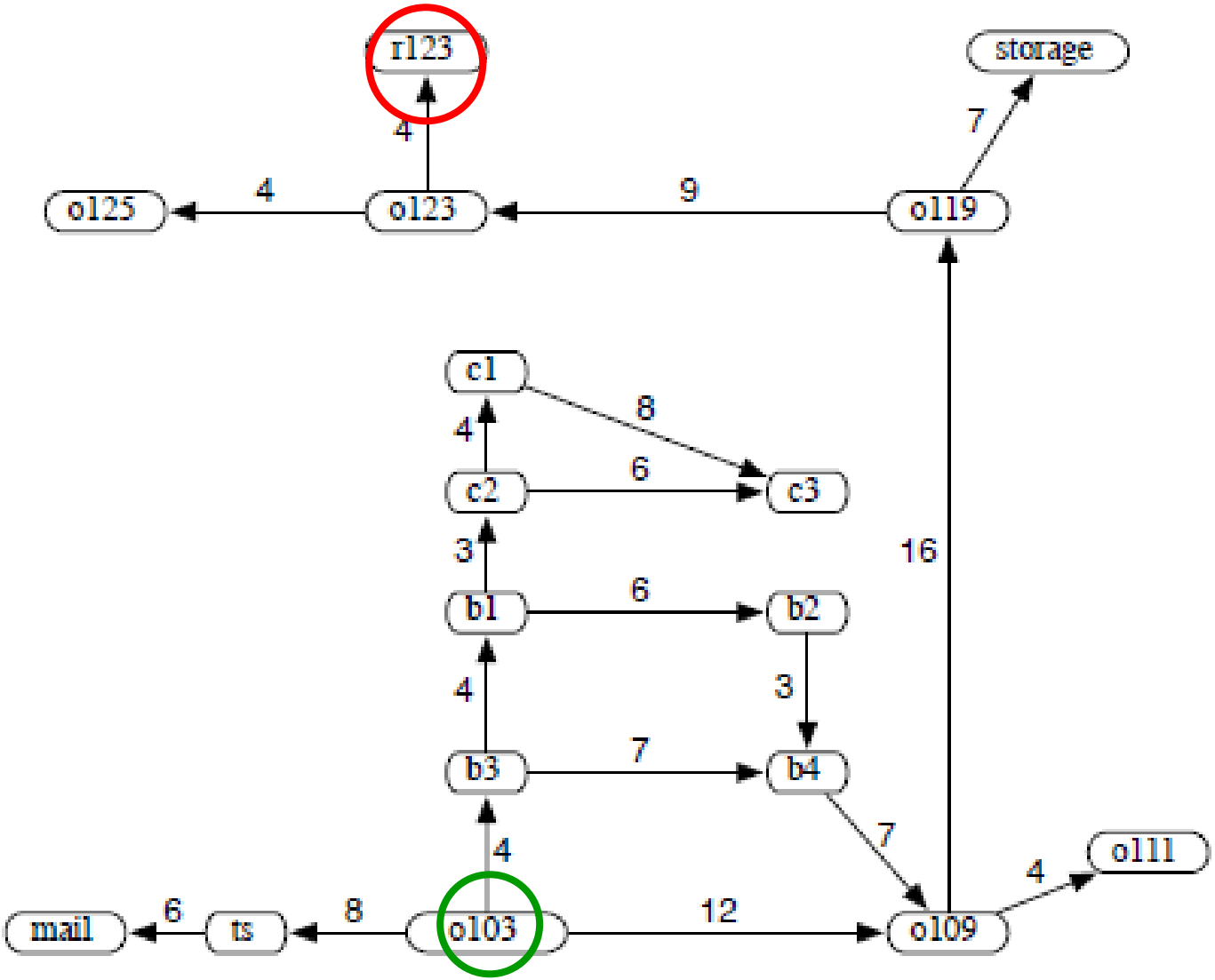
Un esempio ...robotico

Delivery Robot

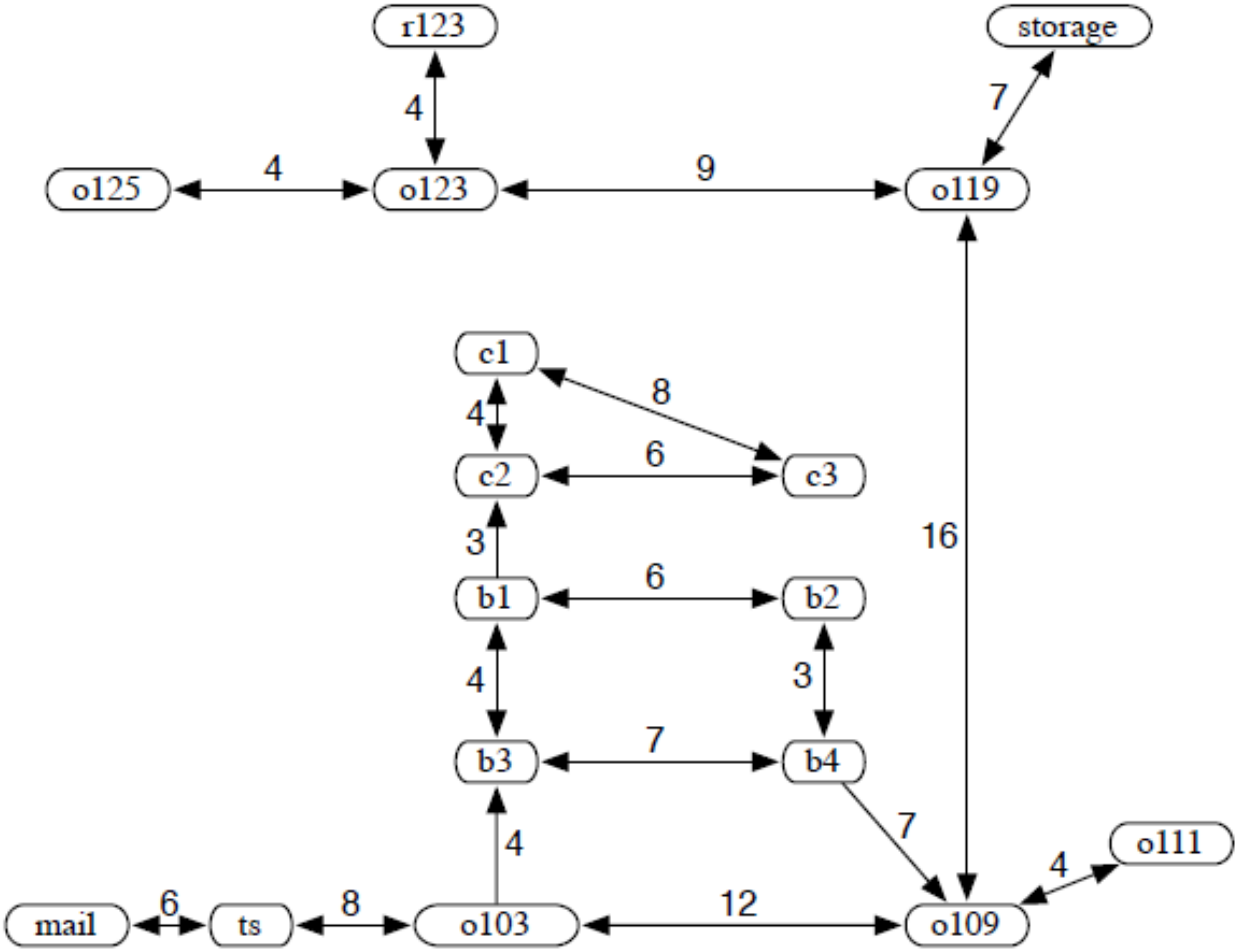


- Il robot vuole andare dall'esterno della stanza 103 all'interno di quella 123
- I nomi (e.g. r101) etichettano I luoghi di interesse
- Etichette in grassetto (e.g. **mail**) aiutano a tracciare la ricerca

Grafo di ricerca per il Delivery Robot



Il Grafo: Cicli



Edges of the form $X \longleftrightarrow Y$ means there is an arc from X to Y and an arc from Y to X. That is, $\langle X, Y \rangle \in A$ and $\langle Y, X \rangle \in A$.

Esempio di euristica ammisssibile: Distanza Euclidea da r123

$$
\begin{array}{rclrclrcl}
h(mail) & = & 26 & h(ts) & = & 23 & h(o103) & = & 21 \\
h(o109) & = & 24 & h(o111) & = & 27 & h(o119) & = & 11 \\
h(o123) & = & 4 & h(o125) & = & 6 & h(r123) & = & 0 \\
h(b1) & = & 13 & h(b2) & = & 15 & h(b3) & = & 17 \\
h(b4) & = & 18 & h(c1) & = & 6 & h(c2) & = & 10 \\
h(c3) & = & 12 & h(storage) & = & 12 & & &
\end{array}
$$