Università di Parma
Dipartimento di Ingegneria e Architettura

Introduzione all'Intelligenza Artificiale

Big Data & Business Intelligence

A.A. 2022/2023

Corso di «Introduzione all'Intelligenza Artificiale»
Corso di «Big Data & Business Intelligence»

Agenti Intelligenti

Monica Mordonini (monica.mordonini@unipr.it)

PROBLEM SOLVING AGENTS

RICERCARE LE SOLUZIONI: STRATEGIE INFORMATE

Metodi di ricerca informati ed esplorazione

- I metodi precedenti si basano al massimo su una misura del costo delle sequenze di azioni trovate, ma non hanno nessun modo per valutare quanto queste possibili soluzioni parziali siano vicine ad un goal del problema.

⇨ *Metodi molto inefficienti*

- I metodi di ricerca informata o euristica

  usano una conoscenza specifica relativa al problema

⇨ *Soluzioni più efficienti*

Strategie informate

"*L'intelligenza di un sistema non è misurabile in termini di capacità di ricerca, ma nella capacità di utilizzare conoscenza sul problema per eliminare il pericolo dell'esplosione combinatoria*.

Se il sistema avesse un qualche controllo sull'ordine nel quale vengono generate le possibili soluzioni, sarebbe allora utile **disporre questo ordine in modo che le soluzioni vere e proprie abbiano un'alta possibilità di comparire prima**.

L'intelligenza, per un sistema con **capacità di elaborazione limitata** consiste nella saggia scelta di cosa fare dopo.....".

Newell-Simon

Strategie informate

- I metodi di ricerca precedenti in uno spazio di profondità d e fattore di ramificazione b risultano di complessità proporzionale a b^d per trovare un goal in una delle foglie.
- Questo è inaccettabile per problemi di una certa complessità.
- Invece di espandere i nodi in modo qualunque utilizziamo conoscenza euristica sul dominio (funzioni di valutazione) per decidere quale nodo espandere per primo.
- Le **funzioni di valutazione** danno una stima computazionale dello sforzo per raggiungere lo stato finale.

Ricerca lungo il Cammino Migliore (Best-First Search)

- E' l'approccio generale dei metodi di ricerca informati
- E' un'istanza dell'algoritmo generale ricerca su un albero o un grafo in cui il nodo da espandere viene scelto sulla base di una **funzione di valutazione**
- Il nome "best-search" è impreciso: se veramente sapessimo sempre quale è il nodo migliore non ci sarebbe ricerca ma solo una marcia diretta verso il goal
- Tutto quello che si può fare è espandere il nodo che sembra più promettente

Ricerca lungo il Cammino Migliore

- Possiamo definire un tale algoritmo come:

```
function Best-First-Search(problem, Enqueue-
   by-Eval-Fn) {
  return General-Search(problem, Enqueue-by-
   Eval-Fn);
}
```

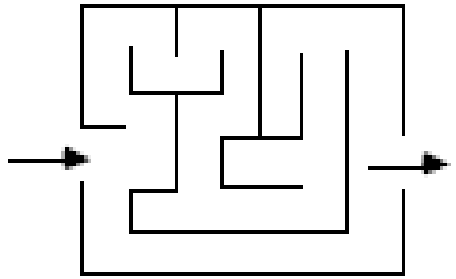
Best-First Search: note

- Il tempo speso per valutare mediante una funzione euristica il nodo da espandere deve corrispondere a una riduzione nella dimensione dello spazio esplorato.
- Trade-off fra tempo necessario nel risolvere il problema (livello base) e tempo speso nel decidere **come** risolvere il problema (meta-livello).
- Le ricerche non informate non hanno queste attività di meta-livello.

Best-First Search: note

- Come trovare le funzioni di valutazione corrette,cioè come si fa a valutare bene quale è il nodo più "promettente"?
- È difficile caratterizzare numericamente la conoscenza empirica sul problema.
- Non sempre la scelta più ovvia è la migliore.

- *Esempio*
- Scelta euristica: muoversi sempre per ridurre la distanza dall'uscita.
- Nel caso di questo labirinto sceglieremmo la strada più lunga.



Ricerca lungo il Cammino Migliore

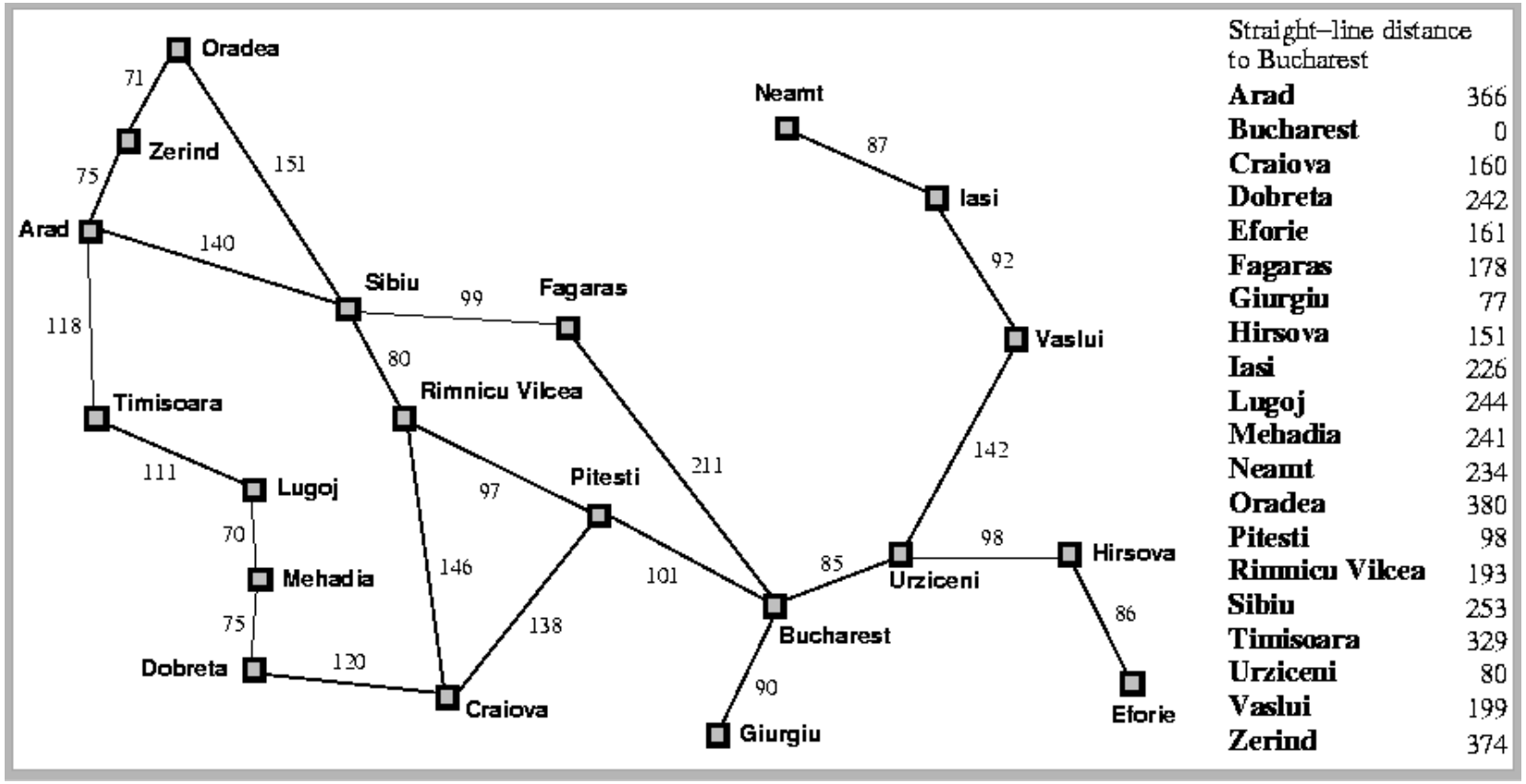
- Punto chiave di questo metodo è l'uso di una *funzione di valutazione* che calcola un numero che rappresenta la desiderabilità relativa all'espansione di nodo.

✓ **h(n)** = costo stimato del cammino più conveniente dal nodo n a un nodo obiettivo

Ricerca lungo il Cammino Migliore

- *Best-first* significa scegliere come nodo da espandere quello che sembra più desiderabile.
- *QueuingFn* = inserisce i successori in ordine decrescente di desiderabilità.
- Casi particolari:
  - *ricerca greedy (golosa)*
  - *ricerca A**

Ricerca lungo il Cammino Migliore



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

Esempio

Mappa della Romania con distanze stradali e distanze in linea d'aria da Bucarest

Ricerca lungo il Cammino più Vicino al Goal (Greedy search = ricerca golosa)

- Questo metodo cerca di minimizzare il costo per raggiungere il goal. Quindi espande il nodo che viene considerato più vicino al goal.
- In molti casi questo costo può essere solo stimato e non calcolato in modo deterministico.
- Si cerca di muoversi verso il massimo (minimo) di una funzione che "stima" il costo per raggiungere il goal

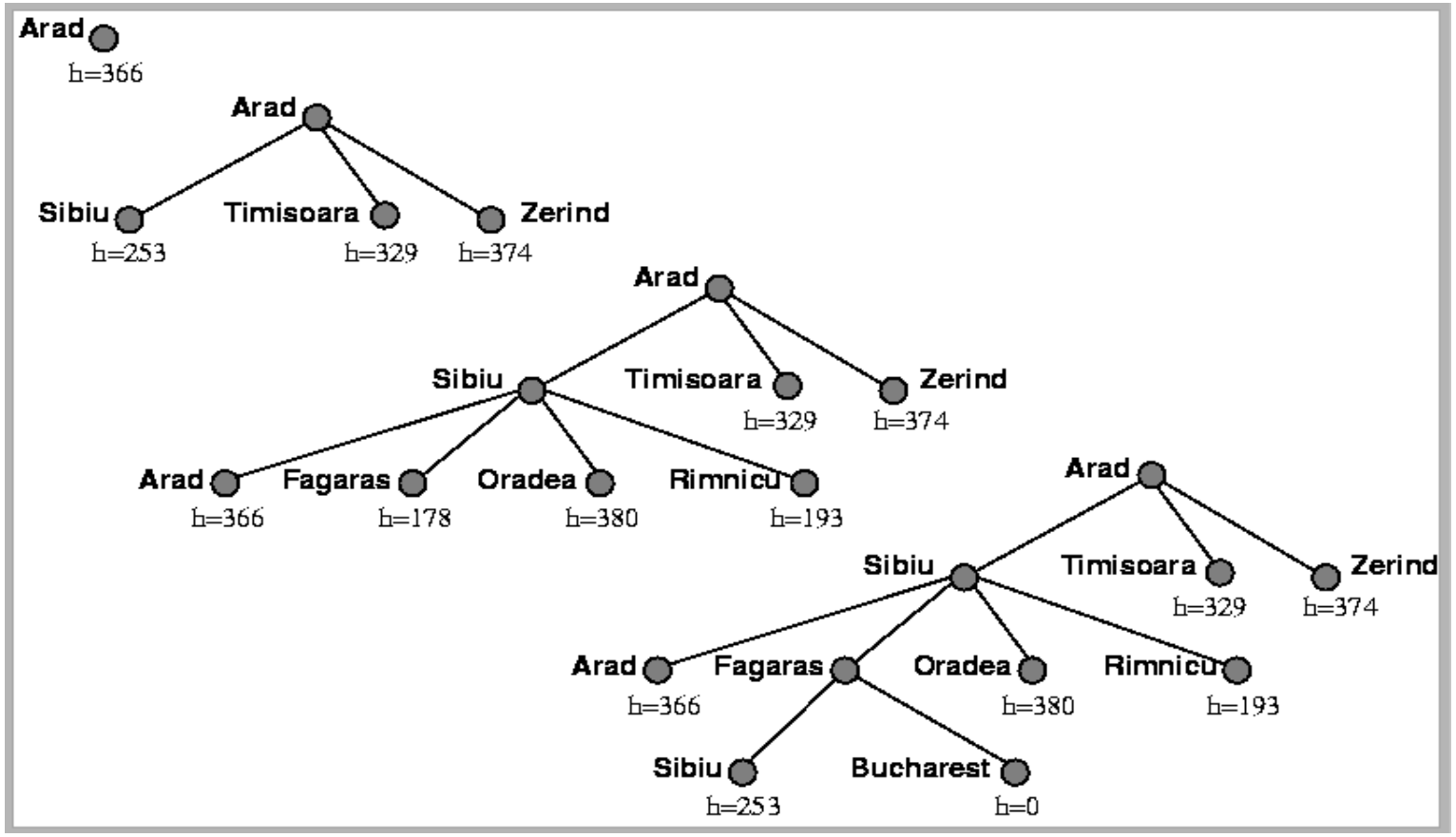
Ricerca lungo il Cammino più Vicino al Goal (Greedy search)

- La funzione che fa questa stima del costo è detta ***funzione euristica*** ed è indicata dalla lettera ***h***.
- h(n) stima il costo per andare dal nodo n al goal.

- Possiamo definire un tale algoritmo come:

```
function Greedy-Search(problem, Enqueue-by-h) {
  return General-Search(problem, Enqueue-by-h);
}
```

Ricerca lungo il Cammino più Vicino al Goal (Greedy search)



Stadi di una ricerca golosa per Bucarest usando come funzione di valutazione la distanza in linea d'aria. I nodi sono etichettati con i valori di ***h***

Ricerca lungo il Cammino più Vicino al Goal (Greedy search)

- Nel caso precedente la ricerca greedy trova una soluzione senza mai espandere alcun nodo che non sia sul cammino della soluzione
- Ne consegue che il costo della ricerca è minimo
- Tuttavia non è ottimo, altri cammini possono essere più brevi
- Inoltre se non si riescono a rilevare la presenza di stati ripetuti la soluzione non verrà mai trovata.

Ricerca lungo il Cammino più Vicino al Goal (Greedy search)

- Questo algoritmo ha molte caratteristiche che lo rendono simile alla ricerca in profondità.
- Quindi non è né ottimo né completo.
- Se b è il fattore di ramificazione e m è la massima profondità dello spazio di ricerca, allora nel peggior caso questo algoritmo ha una complessità spaziale e temporale di b^m.
- A differenza dell'algoritmo di ricerca in profondità, la complessità temporale e spaziale sono uguali perché tutti i nodi vengono mantenuti in memoria.
- ***L'efficienza della ricerca dipende dalla bontà dell'euristica.***

Ricerca lungo il Cammino più Vicino al Goal (Greedy search)

- Questa ricerca (detta anche) in salita non è detto che trovi la soluzione migliore, ovvero **il cammino migliore** per arrivare ad essa
- Questo perchè la tecnica greedy search cerca di trovare il più presto possibile un nodo con distanza 0 dal goal senza curarsi di trovare quel nodo che ha profondità più bassa.
- ***Invece di considerare solo la distanza dal goal, noi consideriamo anche il "costo" nel raggiungere il nodo N dalla radice.***

L'Algoritmo A*
(ricerca A-stella)

- Si propone di minimizzare il costo totale stimato della soluzione
- E' la forma di ricerca a costo uniforme e best-first più diffusa
- Combina i metodi di ricerca a costo uniforme e quello lungo il cammino più vicino al goal
- Si ottiene un algoritmo, detto A*, che offre i vantaggi di entrambi

L'Algoritmo A* (ricerca A-stella)

- L'algoritmo lungo il cammino più vicino al goal (Greedy Search) minimizza il costo, h(n), per raggiungere il goal e spesso permette di ridurre considerevolmente il costo della ricerca, ma non è completo e neppure ottimo.
- L'algoritmo di ricerca a costo uniforme minimizza il costo, g(n), del percorso per arrivare al nodo ed è completo e ottimo, ma può essere molto inefficiente.

- Per integrare entrmbi, l'algoritmo A* somma le due funzioni di valutazione: $f(n) = g(n) + h(n)$.

Ricerca A*: Minimizzare il costo totale stimato della soluzione

Evita di espandere quei cammini che sono già costosi.

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

$g(n)$ = costo effettivo dalla radice al nodo n

$h(n)$ = costo stimato dal nodo n al nodo obiettivo (goal)

$f(n)$ = costo totale stimato di un cammino che arriva al goal passando per n

Ricerca A*: Minimizzare il costo totale stimato della soluzione

La ricerca ***A**** usa una euristica ammissibile cioè:

h(n) **<=** ***h*(n)*** dove ***h*(n)*** è il vero costo da ***n*** al goal.

(Nel nostro esempio la distanza in linea d'aria non sovrastima mai l'effettiva distanza stradale)

Teorema: la ricerca A* è ottimale (e completa)

NB le euristiche ammissibili sono ottimiste per natura perché pensano che il costo della soluzione del problema sia inferiore a quello reale

L'Algoritmo A*
(ricerca A-stella)

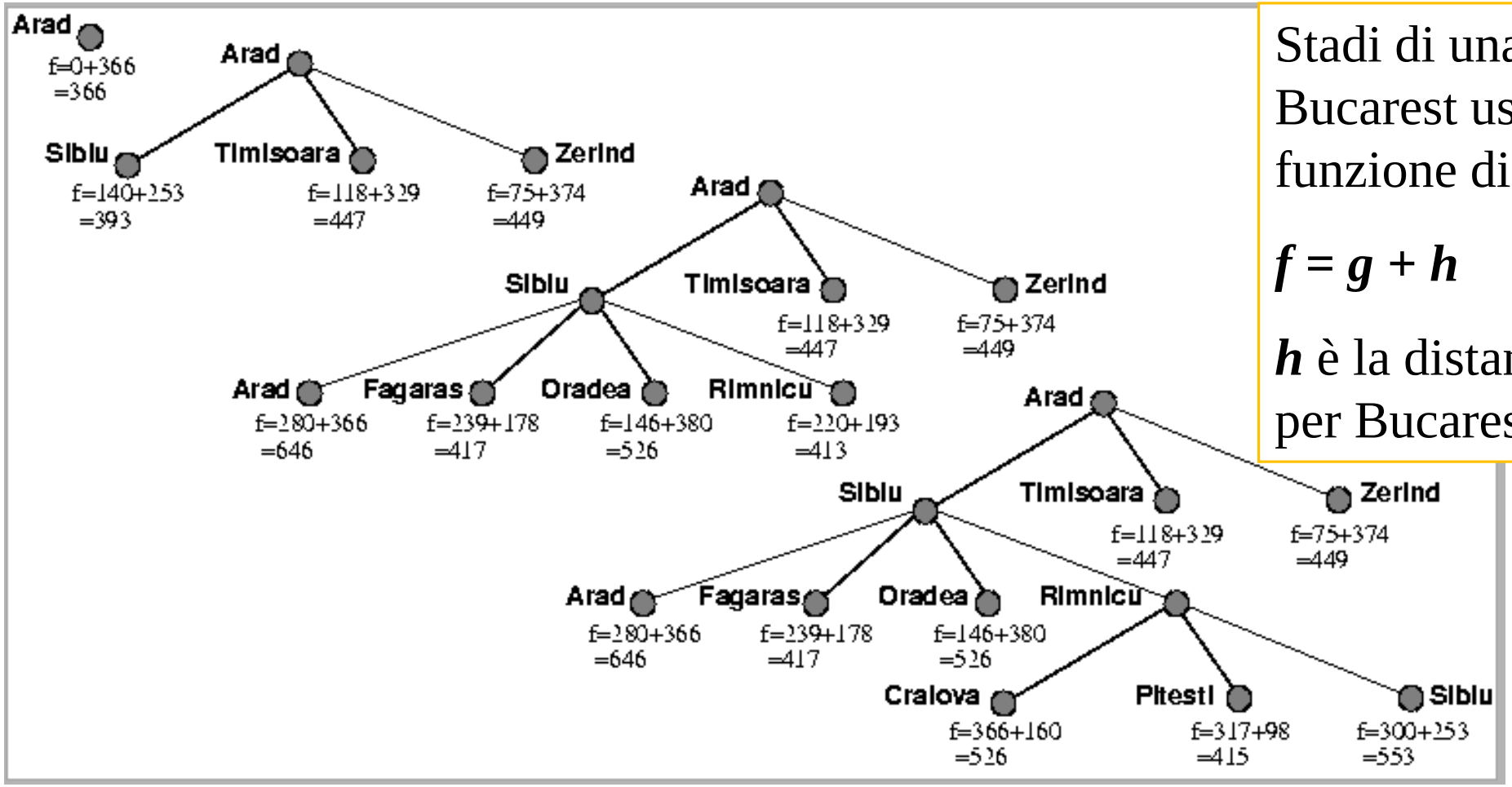
Possiamo definire un l'algoritmo A* come:

```
function A*-Search(problem, Enqueue-by-g+h) {
  return General-Search(problem, Enqueue-by-g+h);
}
```

Cioè:

- 1. Sia L una lista dei nodi iniziali del problema.
- 2. Sia n il nodo di L **per cui g(n) + h'(n) è minima**. Se L è vuota fallisci;
- 3. Se n è il goal fermati e ritorna esso più la strada percorsa per raggiungerlo
- 4. Altrimenti rimuovi n da L e aggiungi a L tutti i nodi figli di n con label la strada percorsa partendo dal nodo iniziale. Ritorna al passo 2

L'Algoritmo A*
(ricerca A-stella)



Stadi di una ricerca ***A**** per Bucarest usando come funzione di valutazione

$f = g + h$

h è la distanza in linea d'aria per Bucarest

L'Algoritmo A*
(ricerca A-stella)

- L'algoritmo A* è ottimamente efficiente (optimally efficient) per ogni funzione euristica, cioè non esiste alcun altro algoritmo ottimo che espande un numero di nodi minore dell'algoritmo A*.
- L'algoritmo non garantisce la soluzione ottima generale
  - Dipende dalla funzione euristica
- *Si può provare che l'algoritmo A* è ottimo e completo se h è una euristica ammissibile, cioè se non sovrastima mai il costo per raggiungere il goal.*

L'Algoritmo A*
(ricerca A-stella)

- Si può dimostrare che l'algoritmo A* è ottimo se h(n) e' ammissibile.
- Supponiamo che sulla frontiera appaia un nodo obiettivo subottimo G2 e sia C* il costo della soluzione ottima.
- Allora G2 è subottimo e h(G2)=0 (nodo obiettivo) si ha che

  (1) f(G2)=g(G2)+h(G2)=g(G2)>C*
- Ora sia n un nodo frontiera che si trova sul cammino della soluzione ottima , sara'

  (2) f(n)=g(n)+h(n)<=C*
- Così da 1e 2 => f(n)<=C*<f(G2)
- cosicché G2 non sarà espanso e l'algoritmo A* dovrà restituire la soluzione ottima

L'Algoritmo A* sui grafi

- Abbiamo assunto, in sostanza, fin qui che lo spazio di ricerca sia un albero e non un grafo. Non è quindi possibile raggiungere lo stesso nodo da strade diverse.

- Come si estendono gli algoritmi precedenti per trattare con i grafi?
- Facendo due liste:
  - Nodi espansi e rimossi dalla lista per evitare di esaminarli nuovamente (nodi chiusi);
  - Nodi ancora da esaminare (nodi aperti).

L'Algoritmo A* per grafi

- 1. Sia La una lista aperta dei nodi iniziali del problema.
- 2. Sia n il nodo di La **per cui g(n) + h'(n) è minima**. Se La è vuota fallisci;
- 3. Se n è il goal fermati e ritorna esso più la strada percorsa per raggiungerlo
- 4. Altrimenti rimuovi n da La, inseriscilo nella lista dei nodi chiusi Lc e aggiungi a La tutti i nodi figli di n con label la strada percorsa partendo dal nodo iniziale. Se un nodo figlio è già in La, non riaggiungerlo, ma aggiornalo con la strada migliore che lo connette al nodo iniziale. Se un nodo figlio è già in Lc, non aggiungerlo a La, ma, se il suo costo è migliore, aggiorna il suo costo e i costi dei nodi già espansi che da lui dipendono. --Ritorno al punto 2.

L'Algoritmo A*

- Si può dimostrare che se invece di un albero ricerchiamo in un grafo l'algoritmo A* potrebbe restituire delle soluzioni subottime

- Infatti l'algoritmo scarta un cammino appena scoperto se questo porta ad un nodo già espanso e può succedere che venga scartato un cammino migliore di quello trovato in precedenza, e quindi che si perda una soluzione ottima.

- Per evitare questo dobbiamo imporre un requisito supplementare su h(n)

L'Algoritmo A*

- Requisito di consistenza o monotocità
- *Un euristica è consistente se, per ogni nodo n e ogni successore n' di n generato da un'azione a, il costo stimato per raggiungere l'obiettivo partendo da n non è superiore al costo di passo per arrivare a n' sommato al costo stimato per andare da li' all'obiettivo*

$h(n) <= c(n,a,n') + h(n')$

Questa è una forma di disuguaglianza triangolare:
Ogni lato del triangolo non può mai essere più lungo della somma degli altri due.
In questo caso il triangolo è formato da n, n' e l'obiettivo

L'Algoritmo A*

- L'algoritmo A* è completo: questo algoritmo espande nodi nell'ordine dato dai rispettivi valori di f. Quindi se un goal ha valore f*, l'algoritmo A* è completo se non ci sono infiniti nodi con f < f*.
- Anche se l'algoritmo A* è l'algoritmo di ricerca ottimo che espande il minor numero di nodi, in molti casi la complessità è esponenziale in funzione della lunghezza della soluzione.
- Si è provato che il numero di nodi espansi cresce esponenzialmente a meno che l'errore della stima h del vero costo h* non cresca più lentamente del logaritmo del costo reale del percorso.

  Cioè $|h(n) - h^*(n)| <= O(\log h^*(n))$

A* Iterativo in Profondità (Iterative Deepening A* - IDA*)

- L'algoritmo A* nei casi peggiori richiede molta memoria.
  Si può usare la tecnica della ricerca iterativa in profondità per limitare la memoria necessaria.
- In questo caso il limite non è la profondità del nodo, ma il costo f:
  ad ogni ciclo vengono espansi solo i nodi con costo minore del limite e viene aggiornato il limite per il prossimo ciclo al minimo valore f dei nuovi nodi generati.
- Come A* è completo ed ottimo e dato che si basa su una ricerca in profondità ha una complessità spaziale di *bd*.

A* Iterativo in Profondità (Iterative Deepening A* - IDA*)

- L'algoritmo A* nei casi peggiori richiede molta memoria. Si può usare la tecnica della ricerca iterativa in profondità per limitare la memoria necessaria.
- In questo caso il limite non è la profondità del nodo, ma il costo f: ad ogni ciclo vengono espansi solo i nodi con costo minore del limite e viene aggiornato il limite per il prossimo ciclo al minimo valore f dei nuovi nodi generati.
- Come A* è completo ed ottimo e dato che si basa su una ricerca in profondità ha una complessità spaziale di *bd*.

A* Iterativo in Profondità - IDA*

- La complessità temporale dipende dal numero di valori che la funzione euristica può assumere dato che questo numero determina il numero di iterazioni.
- In alcuni casi, ad esempio la Manhattan distance per il problema dell'8-puzzle, la funzione euristica assume pochi valori.
- In altri casi, ad esempio, la distanza euclidea per il problema del commesso viaggiatore, la funzione euristica assume molti valori con limite massimo il numero di nodi N:
  - $1 + 2 + ... + N = O(N^2)$

A* Iterativo in Profondità - IDA*

- Un modo per superare questo limite è incrementare il limite di costo di ε rendendo il numero di cicli proporzionale a $1/\varepsilon$.
- Questo algoritmo è detto ε-ammissibile.
- L'algoritmo non è ottimo perché può ritornare una soluzione che può essere peggiore di quella ottima, ma al massimo di ε.

A* Iterativo in Profondità - IDA*

- Possiamo definire l'algoritmo IDA* come segue:

```
function IDA*(problem) {
  root = Make-Node(Initial-State(problem));
  f-limit = f-Cost(root)
  while true {
    solution, f-limit = DFS-Contour(root, f-limit);
    if solution return solution;
    if (f-limit = = ∞) return failure;
}
```

A* Iterativo in Profondità - IDA*

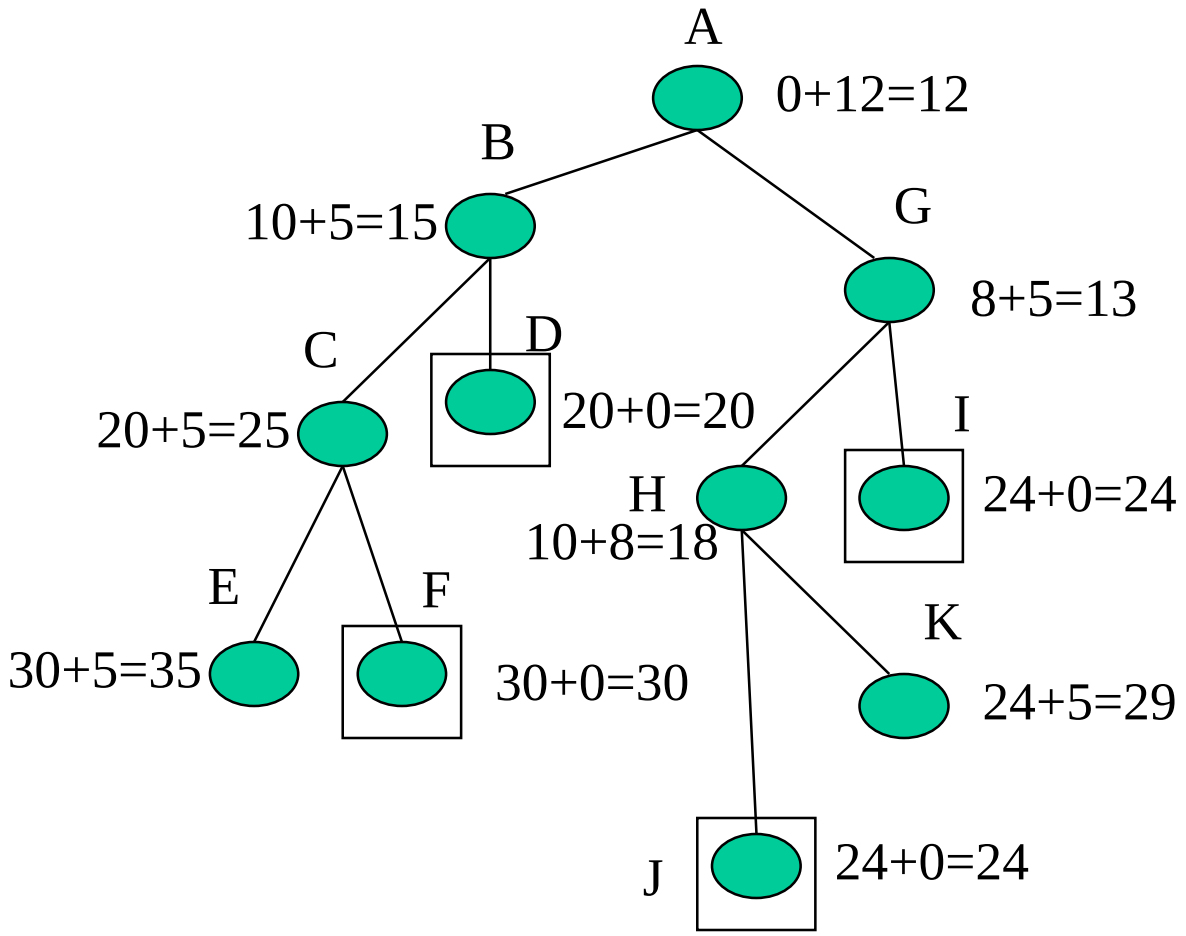
- Dove la funzione DFS-Contour è definita come segue:

```
function DFS-Contour(problem, node, f-limit) {
  next-f = ∞;
  if (f-Cost(node) > f-limit) return null, f-limit;
  if Goal(State(node), problem) return node, f-
   limit;
  for each son in Expand(node, problem) {
    solution, new-f = DFS-Contour(son, f-limit);
    if solution return solution, f-limit;
    next-f = Min(next-f, new-f);
  }
  return null, next-f;
}
```

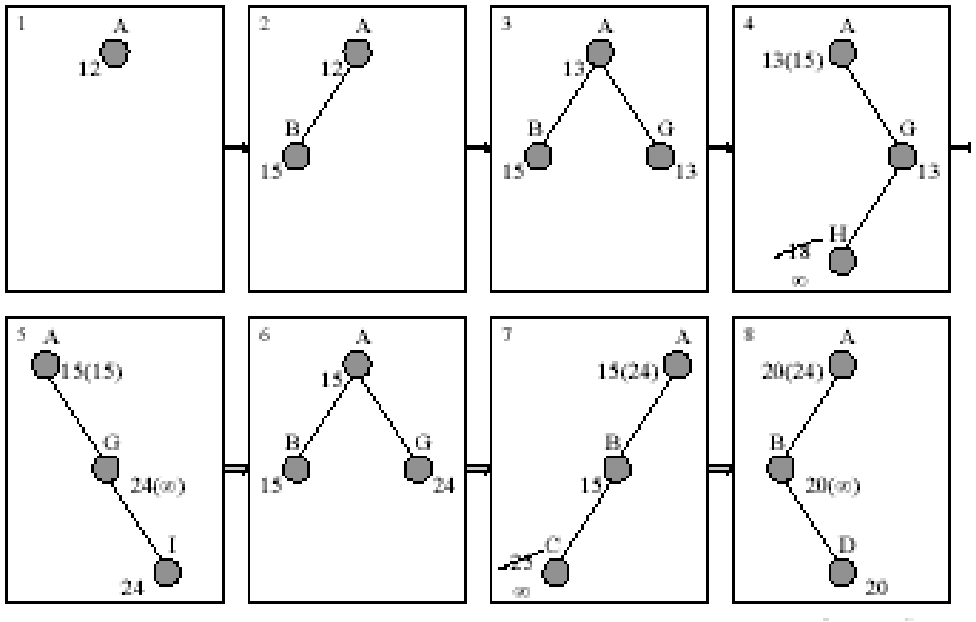
SMA*
(Simplified Memory-Bounded A*)

- SMA* espande sempre la foglia migliore finché la memoria è piena
- A questo punto deve cancellare un nodo in memoria
- SMA* cancella il peggiore quello con f-valore più elevato
- Però memorizza nel nodo padre il valore del nodo rimosso
- In questo modo SMA* può rigenerare il sottoalbero dimenticato quando tutti gli altri cammini sono falliti

SMA*


A
0+12=12
B
10+5=15
G
8+5=13
C
D
20+5=25
20+0=20
I
H
24+0=24
10+8=18
E
F
K
30+5=35
30+0=30
24+5=29
J
24+0=24

SMA*



- Si aggiunge B ad A, poi G ad A . Quello a costo minimo è G, allora scarto B ma mantengo in memoria A eppoi proseguo espandendo G.
- Vado ad espandere poi H ma non posso proseguire
- Espando I.
- I nodo obiettivo ma non è detto che sia il migliore
- Rigenero a partire da A il nodo B…

SMA*

- I nodi rimossi sono rigenerati solo nel caso in cui tutti i percorsi attivi assumano un valore peggiore del valore di un percorso stimato rimosso.
- L'algoritmo SMA* ha le seguenti proprietà:
  - Utilizza la memoria a disposizione.
  - Evita di rigenerare nodi nei limiti della memoria a disposizione.
  - È completo se la memoria è sufficiente a contenere il percorso che comprende la soluzione.
  - È ottimo se la memoria è sufficiente a contenere il percorso della soluzione, altrimenti ritorna la migliore soluzione che può raggiungere con la memoria a disposizione.
  - È ottimamente efficiente se può memorizzare tutto l'albero di ricerca.

SMA*

- Se la memoria è sufficiente l'algoritmo SMA* risolve dei problemi più difficili di quelli risolti da A*.
- Tuttavia, con problemi molto difficili, risolvibili da A* se avesse a disposizione una memoria illimitata, la necessità di ripetere più volte la generazione di nodi rende il problema intrattabile dal punto di vista temporale.

Learning to search better

- Sebbene non ci sia una teoria che formalizzi il problema del compromesso tra tempo e memoria, non sembra che esista una soluzione efficace se non quella di rinunciare al requisito dell'ottimalità
- Ed insegnare alla macchina ad imparare meglio aggiungendo uno spazio di metalivello al dominio del problema.
- Per problemi più difficili, ci saranno molti di questi passi falsi e un algoritmo di apprendimento ad un metalivello può imparare da queste esperienze per evitare di esplorare assemblaggi di sottoalberi poco promettenti, queste tecniche sono fondamentali per il successo
- e cercando di minimizzare il costo totale della soluzione del problema dato dal costo del cammino ma anche dal costo computazionale che serve per ottenerlo

Ricerca informata

SCELTA DELL'EURISTICA

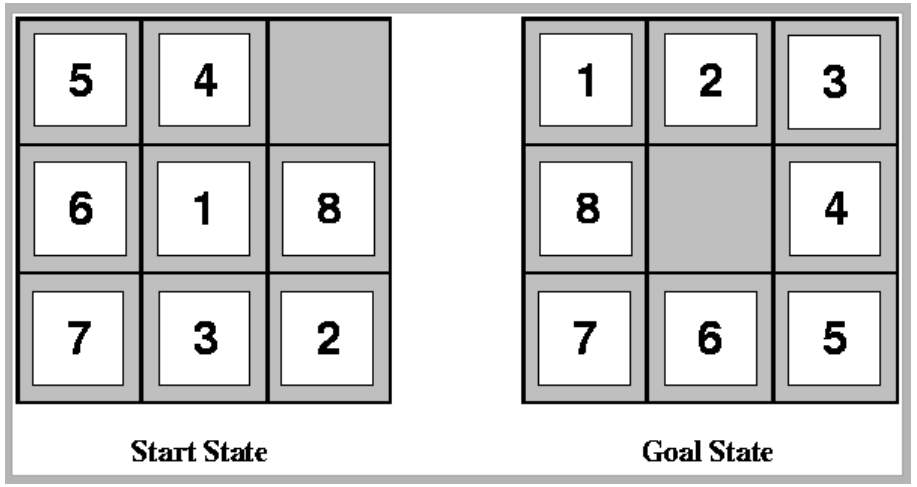
Scelta dell'Euristica

- Consideriamo un problema molto semplice come 8-puzzle. Una soluzione tipica ha 20 mosse.
- Dato che il fattore di ramificazione è circa 3, allora una ricerca in profondità esaustiva visiterebbe $3^{20} = 3.5 \times 10^9$ nodi.
- Il numero di nodi visitati si può limitare eliminando i nodi già visitati, dato che gli stati possibili sono 9! = 362880.
- Tuttavia, sono valori eccessivi per poter risolvere il problema in modo esaustivo. Occorre quindi cercare di limitare la lunghezza della ricerca con delle buone euristiche.

Scelta delle euristiche

Esempio 8-puzzle

E' possibile definire differenti funzioni euristiche



Ad esempio:

h_1= numero di tessere che sono fuori posto (h_1= 7)

h_2= la somma delle distanze dalle posizioni che le tessere devono assumere nella configurazione obiettivo.

La distanza è una somma delle distanze orizzontali e verticali (**distanza di Manhattan**).

Le tessere da 1 a 8 nello stato iniziale danno una distanza h_2= 2+3+3+2+4+2+0+2 = 18

Scelta dell'Euristica

- Si è visto che due euristiche candidate per l'8-puzzle sono:
  - h_1 = numero di quadrati fuori posizione.
  - h_2 = Manhattan distance = somma delle distanze dei quadrati dalla loro posizione finale.

- Da risultati sperimentali si ricava che h_2 è migliore.

- Questo è ovvio dato che:
  - Entrambe le euristiche non sovrastimano il costo per raggiungere il goal, cioè sono ammissibili.
  - Per ogni configurazione $h_2 >= h_1$

- Quindi h_2 è una stima migliore del costo per raggiungere il goal e domina sull'altra.

Scelta dell'Euristica

- Ecco un confronto dei costi di IDS (iterative-deepening search) e A* con h_1 e h_2 (numero medio di nodi espansi in 100 esperimenti):

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Figure 3.29 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.

Scelta dell'Euristica

- La dominazione si traduce direttamente in efficienza: A∗ usando h2 non espanderà mai più nodi di A∗ usando h1 (tranne forse per alcuni nodi con f(n)=C∗).
- Il motivo è semplice:
- ogni nodo con f(n) < C∗ sarà sicuramente espanso. Questo equivale a dire che ogni nodo con h(n) < C∗ − g(n) sarà sicuramente espanso.
- Ma poiché h2 è grande almeno quanto h1 per tutti i nodi, ogni nodo che è sicuramente espanso dalla ricerca di A∗ con h2 sarà sicuramente espanso anche con h1, e h1 potrebbe far espandere anche altri nodi.
- Quindi, è generalmente meglio usare una funzione euristica con valori più alti, a condizione che sia coerente e che il tempo di calcolo per l'euristica non sia troppo lungo.

Scelta dell'Euristica

- Come si può scegliere una euristica? Può un programma compiere questa scelta in modo automatico ?

- Le due euristiche esaminate per l'8-puzzle sono le funzioni di valutazione **esatte** della distanza dal goal per due versioni semplificate del problema.
- Quindi si può dire che le soluzioni di un problema P_r, ottenuto rilassando (togliendo restrizioni alle) le regole di un problema P, sono delle buone euristiche per P.

Scelta dell'Euristica

Generating admissible heuristics from relaxed problems

- Il costo di una soluzione ottima di un problema rilassato è un'euristica ammissibile per il problema originale perché la soluzione ottima del problema originale è per definizione anche una soluzione del problema rilassato, quindi deve avere un costo almeno pari alla soluzione ottima di quest'ultimo.

- Dato che l'euristica derivata è un costo esatto del problema rilassato, deve rispettare la disuguaglianza triangolare ed è quindi consistente.

Scelta dell'Euristica

- Se la definizione di un problema è scritta in un linguaggio formale, è possibile costruire automaticamente i suoi rilassamenti.

- *Es: un tassello si può muovere da A a B se A e B sono adiacenti orizzontalmente o verticalmente AND B è vuota*
- Possiamo generare 3 problemi rilassati rimuovendo 1 o entrambe le condizioni
  A. Un tassello può muoversi da A a B se A e B sono adiacenti
  B. Un tassello può muoversi da A a B se B è vuota
  C. Un tassello può muoversi da A a B

Scelta dell'Euristica

A. Un tassello può muoversi da A a B se A e B sono adiacenti

   Da quì deriviamo la distanza di Manhattan

   h2 sarebbe la distanza esatta dalla soluzione se potessimo muovere I tasselli uno dopo l'altro fino alla loro destinazione

B. Un tassello può muoversi da A a B se B è vuota

C. Un tassello può muoversi da A a B

   Da qui' deriviamo h1 che rappresenterebbe la distanza esatta se i tasselli potessero essere spostati nella loro destinazione in un solo passo

Scelta dell'Euristica

Notice that it iscrucial that the relaxed problems generated by th is technique can be solved essentially without search, because the relaxed rules allow the problem to be decomposed into eight independent subproblems.

If the relaxed problem is hard to solve, then the values of the corresponding heuristic will be expensive to obtain

Scelta dell'Euristica

Se la definizione di un problema è scritta in un linguaggio formale, è possibile quindi far generare ad una macchina euristiche a partire dalla definizione dei problemi.

- Il programma ABSOLVER (1993) è riuscito a trovare la prima euristica utile per il cubo di Rubik

One problem with generating new heuristic functions is that one often fails to get a single "clearly best" heuristic.

If a collection of admissible heuristics h1 . . .hm is available for a problem and none of them dominates any of the others, which should we choose?

As it turns out, we need not make a choice.

Scelta dell'Euristica

We can have the best of all worlds, by defining
$h(n) = \max\{h1(n), \ldots, hm(n)\}$

- This composite heuristic uses whichever function is most accurate on the node in question.
- Because the component heuristics are admissible, h is admissible; it is also easy to prove that h is consistent. Furthermore, h dominates all of its component heuristics

Scelta dell'Euristica

Se la definizione di un problema è scritta in un linguaggio formale, è possibile quindi far generare ad una macchina euristiche a partire dalla definizione dei problemi.

- Il programma ABSOLVER (1993) è riuscito a trovare la prima euristica utile per il cubo di Rubik

- Un altro modo per generare delle buone euristiche è usare delle informazioni statistiche.

- Un'altra possibilità è apprendere dall'esperienza
  - Reti neurali, alberi di decisione,….