Ricerca euristica

Maria Simi
a.a. 2012/2013

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

Funzioni di valutazione euristica

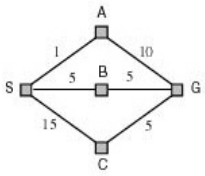
Conoscenza del problema data tramite una *funzione di valutazione* dello stato, detta funzione di valutazione euristica:

$f : n \rightarrow R$

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

Esempi di euristica

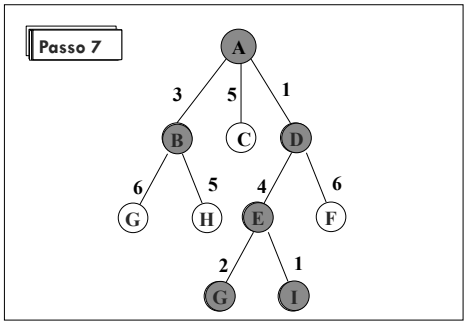
- La città più vicina (o la città più vicina alla mèta in linea d'aria) nel problema dell'itinerario
- Il numero delle caselle fuori posto nel gioco dell'otto
- Il vantaggio in pezzi nella dama o negli scacchi



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.

Strategia best-first: esempio

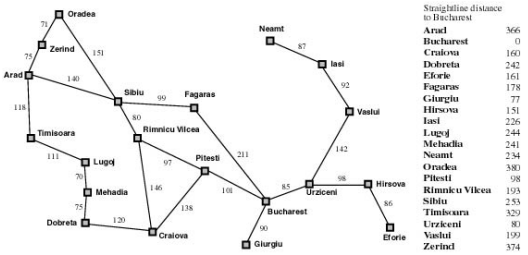


La Best First non è in generale completa, né ottimale

Ricerca greedy best-first

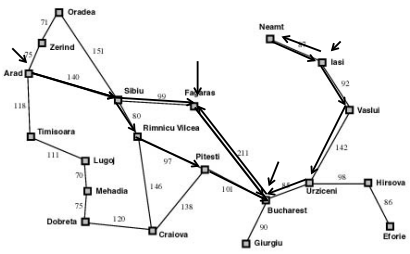
- Si usa come euristica una stima della distanza della soluzione, da ora in poi $h(n)$ $[h \geq 0]$
- *Esempio*: ricerca greedy per Route Finding
  $h(n)$ = distanza in linea d'aria tra lo stato di n e la destinazione
  In generale l'algoritmo *non* è *completo*

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

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

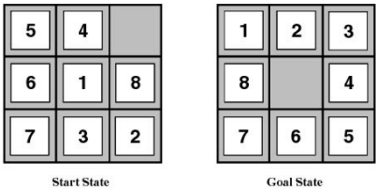
Algoritmo A: definizione

- 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:
  $f(n) = g(n) + h(n)$, con $h(n) \geq 0$ 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 Uniforme
- Se $g(n) = 0$ $[f(n) = h(n)]$ si ha Greedy Best First

Algoritmo A: esempio

Esempio nel gioco dell'otto

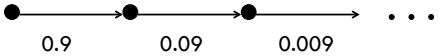


$f(n)$ = #mosse fatte + #caselle-fuori-posto
f(Start) = 0 + 7 Dopo ←,↓, ↑,→ f = 4 + 7

L' algoritmo A è completo

Teorema: L'algoritmo A con la condizione
$g(n) \geq d(n) \cdot \varepsilon$ ($\varepsilon > 0$ 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 $[n_0\ n_1\ n_2\ \ldots\ n' \ldots\ n_k{=}goal]$ 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

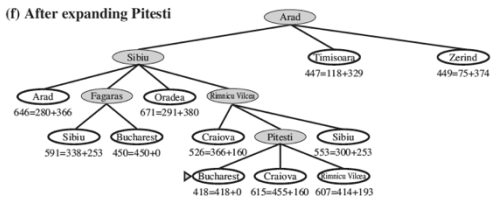
Definizione: Algoritmo A*

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*



Osservazioni su A*

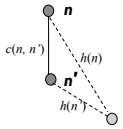
1. Una sottostima può farci compiere del lavoro inutile, però non ci fa perdere il cammino migliore
2. La componente g fa sì che si abbandonino cammini che vanno troppo in profondità
3. Una funzione che qualche volta sovrastima può farci perdere la soluzione ottimale

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

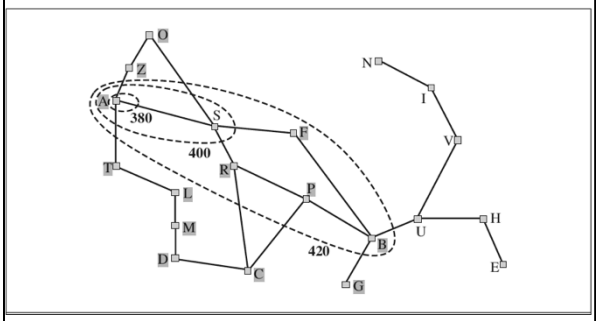
Euristiche monotòne: proprietà

- *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*.

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

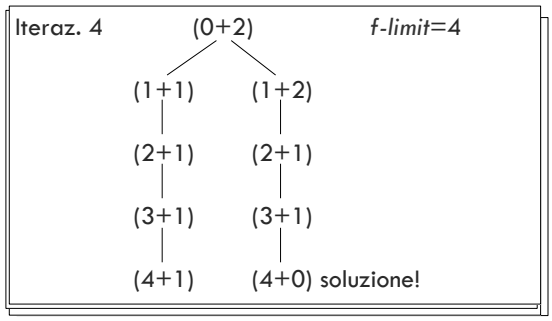
- Nel Best First viene tenuta tutta la frontiera; se l'occupazione di memoria è eccessiva si può ricorrere ad una variante: la *Beam search*.
- La *Beam Search* tiene ad ogni passo solo *i k nodi più promettenti*, dove *k* è detto *l'ampiezza del raggio (beam)*.
- La *Beam Search* non è completa.

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 *f-limit* viene aumentato ad ogni iterazione, fino a trovare la soluzione.

Esempio



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.

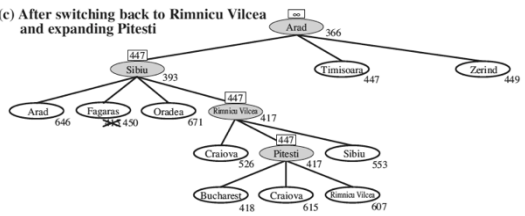
Analisi IDA*

- IDA* 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)

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



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* 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:

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

Più informata, più efficiente

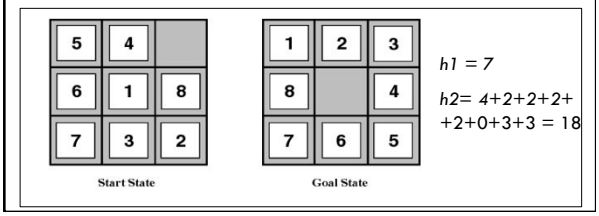
Teorema: Se $h_1 \le h_2$, i nodi espansi da A* con h_2 sono un sottoinsieme di quelli espansi da A* con h_1.

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

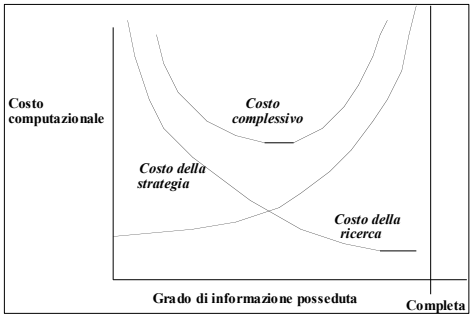
- Un'euristica più informata riduce lo spazio di ricerca (è più efficiente), ma è tipicamente più costosa da calcolare

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 $\forall n \,.\, h_1(n) \le h_2(n)$



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

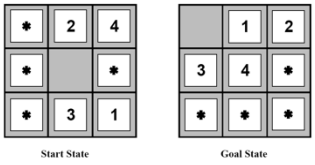
- Nel gioco dell'8 mossa da A a B possibile se ...
  1. B adiacente a A
  2. B libera
- h_1 e h_2 sono calcoli della *distanza esatta* della soluzione in versioni semplificate del puzzle:
  - h_1 (nessuna restrizione): sono sempre ammessi scambi a piacimento tra caselle → # caselle fuori posto
  - h_2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

Massimizzazione di euristiche

- Se si hanno una serie di euristiche ammissibili $h_1, h_2, \ldots h_k$ 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.

Eurstiche da sottoproblemi



- 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