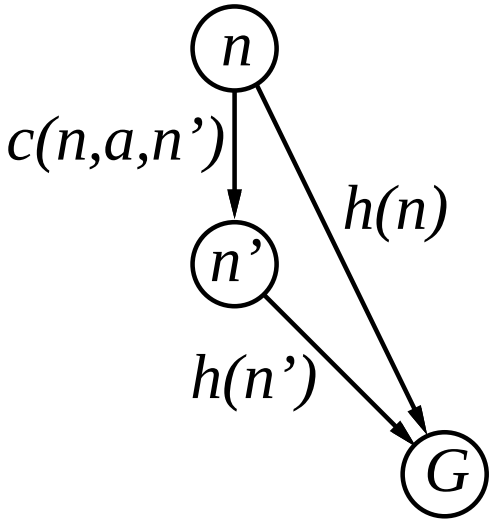
Ricerca Informata

Capitolo 4

Consistenza

Una euristica è *consistente* se

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



Se h è consistente, si ha

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

cioè, $f(n)$ è non decrescente lungo un cammino

Altra proprietà di A*

A* non è solo ottimo:

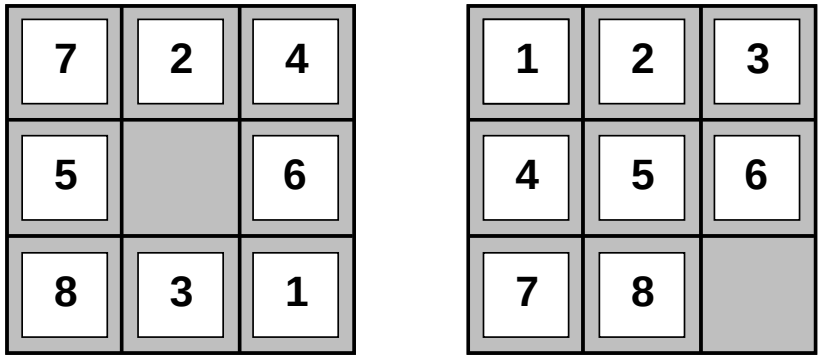
non esiste altro algoritmo ottimo che sicuramente espande meno nodi di A* (se non per risolvere "patte" sui nodi con f uguale al valore ottimo)

Questo è vero perché se così non fosse, allora tale algoritmo rischierebbe di non esplorare nodi ottimi, e quindi non sarebbe un algoritmo ottimo.

Euristiche ammissibili

Per esempio, consideriamo 8-puzzle:

$h_1(n) =$ numero di tasselli in posizione errata
$h_2(n) =$ distanza di Manhattan totale
(cioè numero di "quadrati" dalla posizione desiderata per ogni tassello)

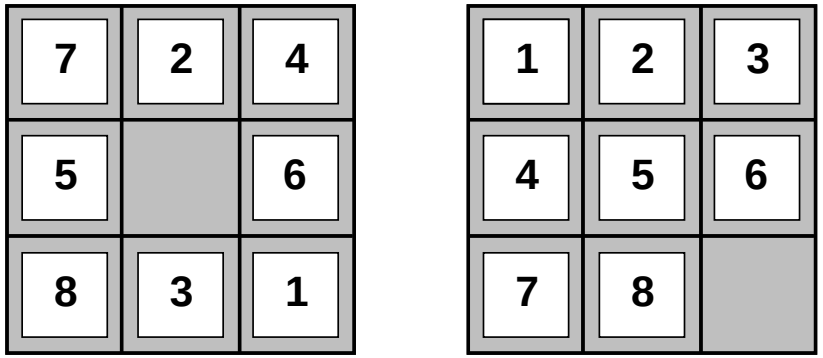


$h_1(S) =$??
$h_2(S) =$??

Euristiche ammissibili

Per esempio, consideriamo 8-puzzle:

$h_1(n) =$ numero di tasselli in posizione errata
$h_2(n) =$ distanza di Manhattan totale
(cioè numero di "quadrati" dalla posizione desiderata per ogni tassello)



$h_1(S) =$?? 7
$h_2(S) =$?? 4+0+3+3+1+0+2+1 = 14

Dominanza

Se $h_2(n) \geq h_1(n)$ per tutti gli n (entrambe ammissibili)
allora h_2 *domina* h_1 ed è migliore per la ricerca

Tipici costi di ricerca:

$d = 14$ IDS $=$ 3,473,941 nodi
A$^*(h_1) =$ 539 nodi
A$^*(h_2) =$ 113 nodi
$d = 24$ IDS $\approx$ 54,000,000,000 nodi
A$^*(h_1) =$ 39,135 nodi
A$^*(h_2) =$ 1,641 nodi

Problemi rilassati

Euristiche ammissibili possono essere derivate dal costo
esatto di una soluzione di una versione *rilassata* del problema

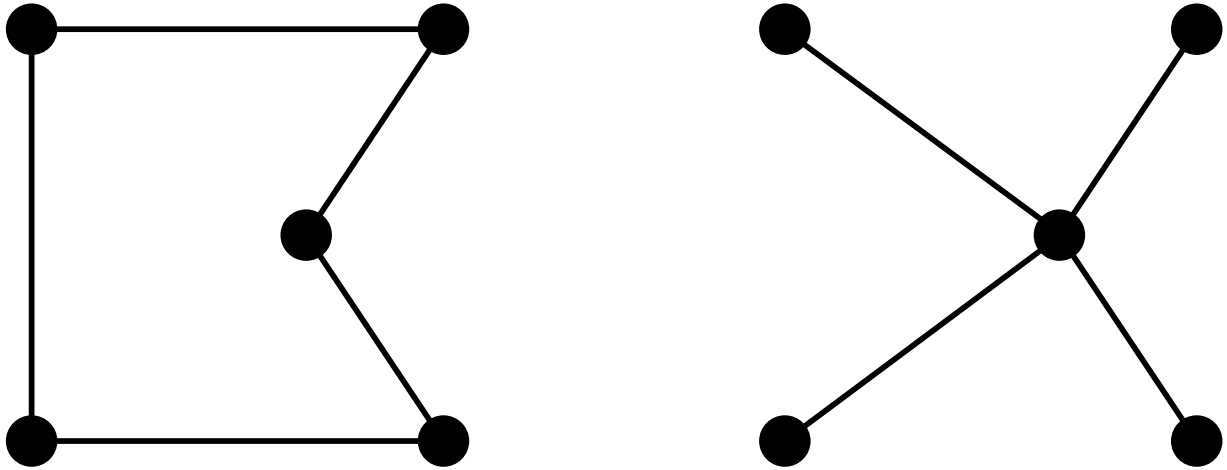
Se le regole di 8-puzzle sono rilassate così che un tassello può muoversi *ovunque*, allora $h_1(n)$ corrisponde alla soluzione sul cammino più breve

Se le regole di 8-puzzle sono rilassate così che un tassello può muoversi *ad ogni "quadrato" adiacente*, allora $h_2(n)$ corrisponde alla soluzione sul cammino più breve

Punto chiave: il costo della soluzione ottima di un problema rilassato
non è più grande del costo della soluzione ottima del problema originario

Problemi rilassati

Esempi noti: Problema del commesso viaggiatore (TSP)
Trovare il percorso più corto che visiti tutte le città esattamente una volta



Albero di copertura minimo (Minimum spanning tree) può essere calcolato in $O(n^2)$ e fornisce un limite inferiore al percorso (aperto) più breve

Algoritmi di miglioramento iterativo

In molti problemi di ottimizzazione, *il cammino* è irrilevante;
la soluzione è costituita dallo stato goal stesso

Quindi lo spazio degli stati è dato dall'insieme delle
configurazioni "complete";

trovare una configurazione *ottima*, es.: TSP

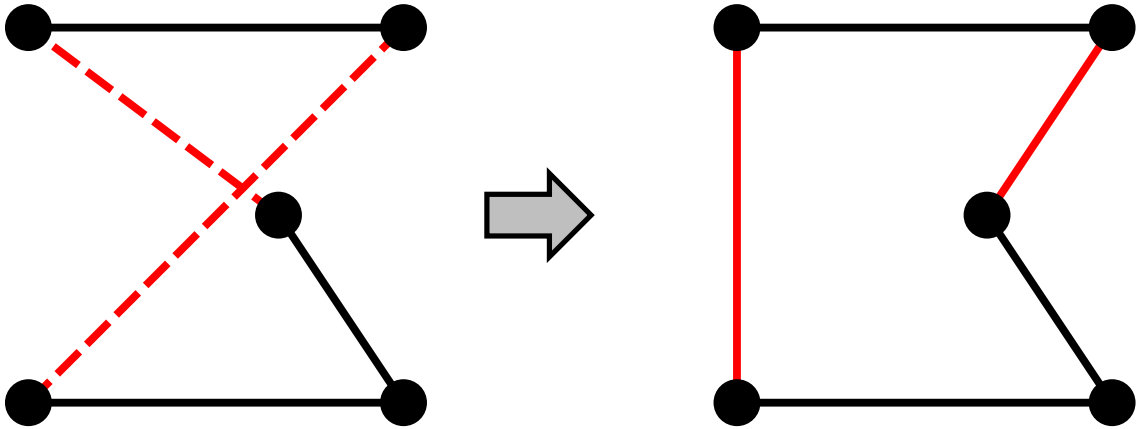
o, trovare una configurazione che soddisfi dei vincoli, es.: orario

In tali casi, si possono usare gli algoritmi di *miglioramento iterativo*;
Mantengono un singolo stato "corrente", e tentano di migliorarlo

Impiegano spazio costante, e sono quindi adatti sia per ricerca online che per ricerca offline

Esempio: Problema del commesso viaggiatore

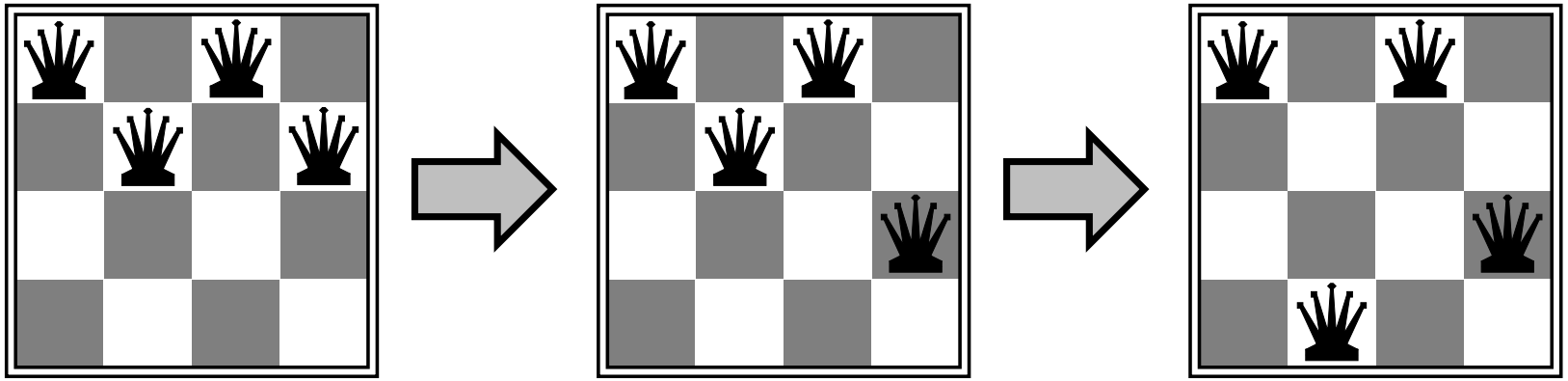
Si parte con un percorso qualsiasi, e si eseguono scambi a coppie



Esempio: n-regine

Disporre n regine su una scacchiera $n \times n$ senza che si minaccino
(non ci devono essere due regine sulla stessa riga, colonna, o diagonale)

Muovere una regina in modo da minimizzare il numero di minacce



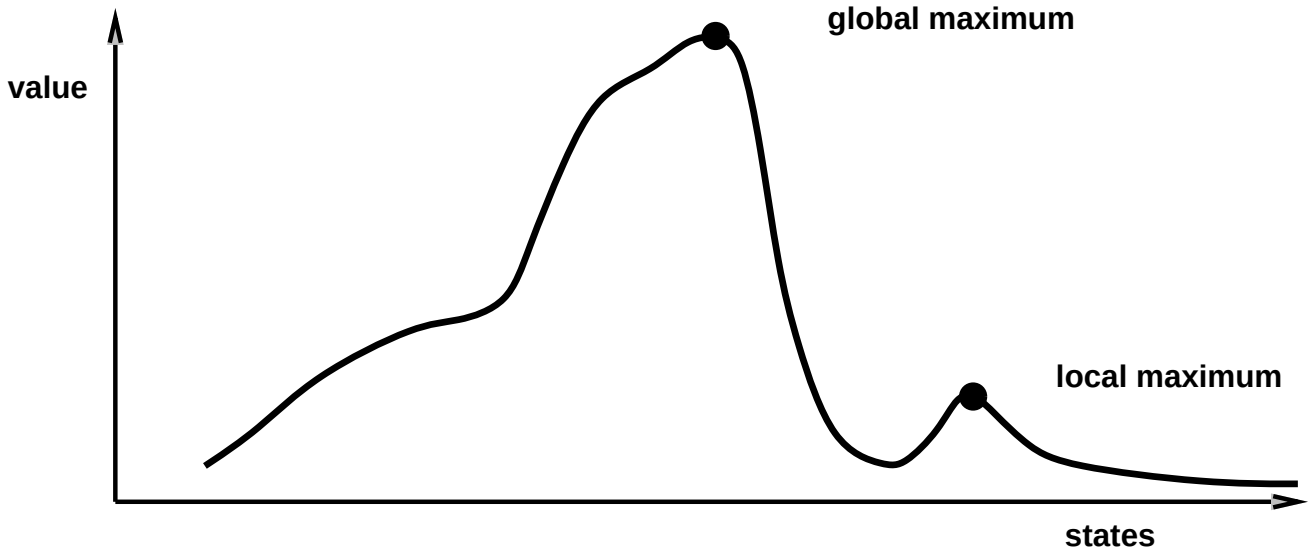
Hill-climbing (o discesa/ascesa di gradiente)

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] < VALUE[current] then return STATE[current]
      current ← neighbor
   end
```

Hill-climbing

Problema: a seconda dello stato iniziale, può fermarsi su dei massimi locali



Nel caso di spazi continui è difficile scegliere la dimensione del passo di progressione ed inoltre si può avere una convergenza molto lenta

Simulated annealing

Idea: evitare i massimi locali permettendo delle mosse "cattive"
ma gradualmente decrementare la loro grandezza e frequenza

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

  current $\leftarrow$ MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[t]
    if $T = 0$ **then return** *current*
    next $\leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* $\leftarrow$ *next*
    else *current* $\leftarrow$ *next* only with probability $e^{\Delta\ E/T}$

Proprietà del simulated annealing

A "temperatura" fissata T, la probabilità di occupazione degli stati raggiunge la distribuzione di Boltzman

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T diminuito abbastanza lentamente $\Longrightarrow$ si raggiunge sempre lo stato migliore (Metropolis et al., 1953, per problemi di modellazione di processi fisici)

Ampiamente usato in applicazioni pratiche, come la progettazione di circuiti VLSI e la definizione degli orari dei voli delle linee aeree

Alcune considerazioni finali su A*

A* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Alcune considerazioni finali su A*

A* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Come si può ridurre l'occupazione di memoria??

Alcune considerazioni finali su A*

A* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Come si può ridurre l'occupazione di memoria??

La soluzione più semplice è quella di adattare l'idea dell'iterative deepening al contesto delle euristiche

Come??

Alcune considerazioni finali su A*

A* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Come si può ridurre l'occupazione di memoria??

La soluzione più semplice è quella di adattare l'idea dell'iterative deepening al contesto delle euristiche

Come??

Algoritmo Iterative Deepening A* (IDA*):

come A*, però:

- non si inseriscono nella coda nodi con valore di f maggiori del valore di $cutoff$
- il valore di $cutoff$ alla iterazione successiva si pone uguale al minimo valore di f dei nodi non inseriti nella coda

Alcune considerazioni finali su A*

A* ha occupazione esponenziale in spazio, e quindi rischia di non essere applicabile a molti problemi interessanti

Come si può ridurre l'occupazione di memoria??

La soluzione più semplice è quella di adattare l'idea dell'iterative deepening al contesto delle euristiche

Come??

Esistono però soluzioni migliori, come RBFS, MA*, SMA*