

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

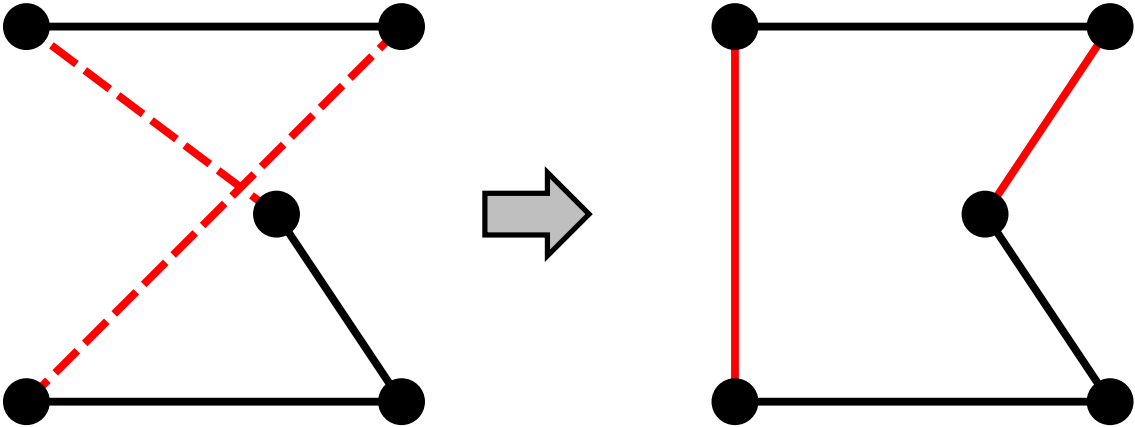
o, trovare una configurazione che soddisfi dei vincoli, es.: orario

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

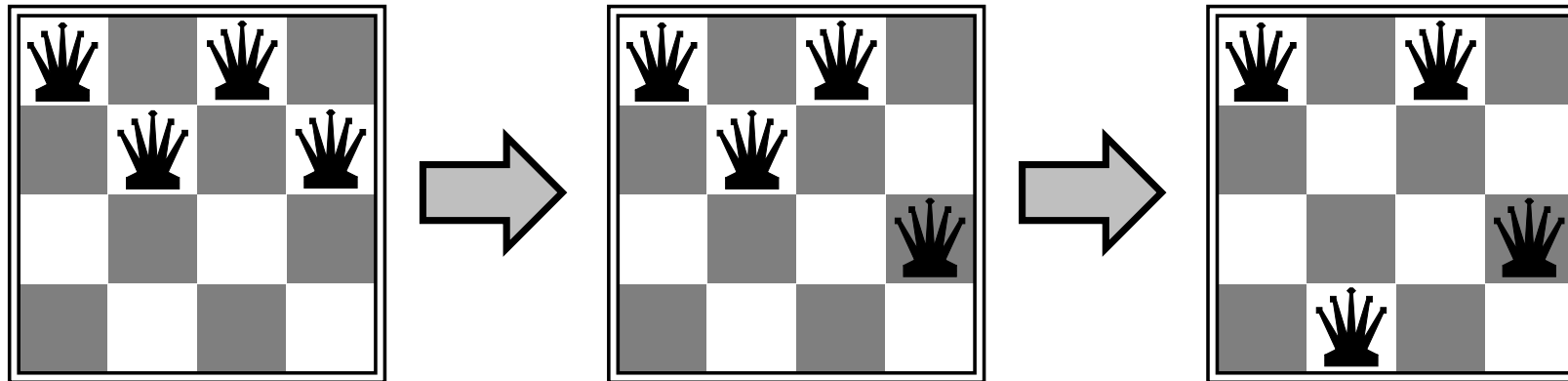
Si parte con un percorso qualsiasi, e si eseguono scambi a coppie



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(*problema*) **returns** uno stato che è un massimo locale

inputs: *problema*, un problema

variabili locali: *nodo_corrente*, un nodo
vicino, un nodo

nodo_corrente ← CREA-NODO(STATO-INIZIALE[*problema*])

loop do

vicino ← il successore di *nodo_corrente* di valore più alto

if VALORE[*vicino*] ≤ VALORE[*nodo_corrente*] **then return** STATO[*nodo_corrente*]

nodo_corrente ← *vicino*

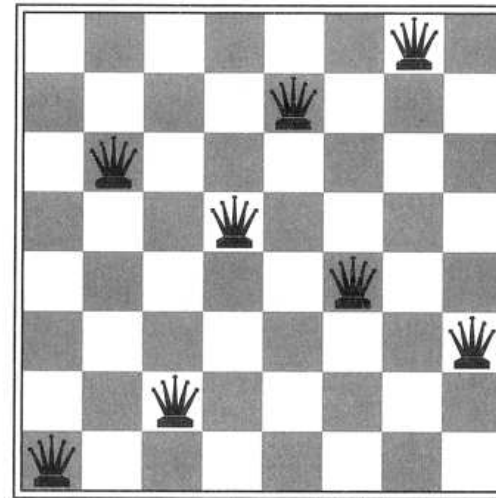
Hill-climbing: esempio delle 8 regine

$h(s)$ = numero di coppie di regine che si attaccano a vicenda

Quanti successori per ogni stato ? 8×7

(# regine \times caselle libere su colonna/riga [deve esserci 1 regina per colonna/riga])

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♚	13	16	13	16
♚	14	17	15	♚	14	16	16
17	♚	16	18	15	♚	15	♚
18	14	♚	15	15	14	♚	16
14	14	13	17	12	14	12	18



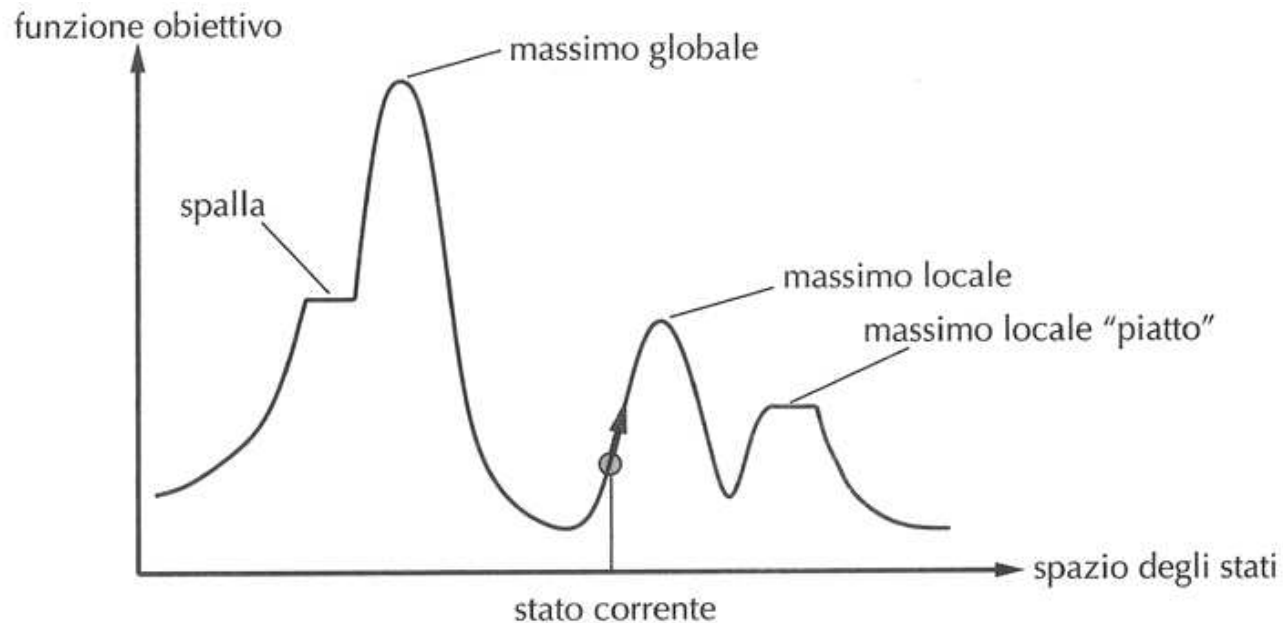
Uno stato con $h = 17$ e valori di h per ogni successore indicati

Un minimo locale con $h = 1$. Ogni successore di questo stato ha $h > 1$.

Lo stato a destra “dista” solo 5 passi da quello a sinistra

Hill-climbing: massimi locali ed altri problemi

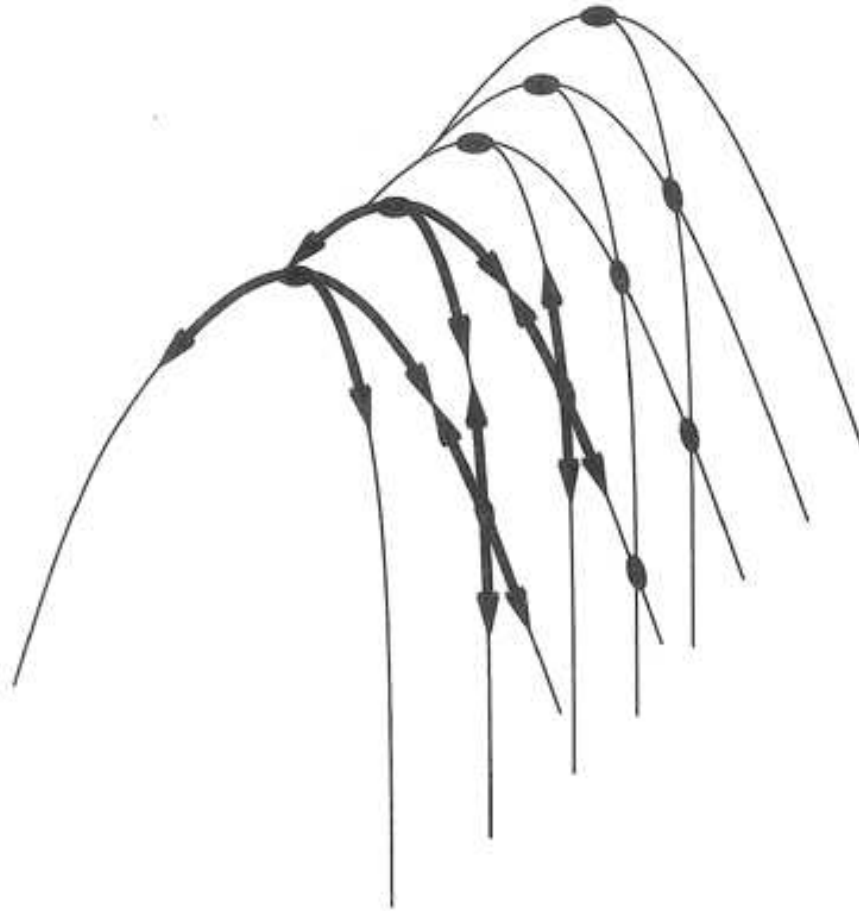
Problemi: a seconda dello stato iniziale, può fermarsi su dei massimi locali e/o "spalle"



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

Hill-climbing: cresta (*ridge*)

Sequenza di massimi locali (a valore crescente muovendosi verso l' "interno" della figura) molto difficili da esplorare per Hill-climbing



Hill-climbing: alcune soluzioni

Possibili soluzioni:

- **plateau (h piatta)**: “mossa laterale”, cioè ci si sposta in uno stato con identico valore di h
 - bisogna stare attenti ad evitare cicli, specialmente nel caso di massimi (minimi) locali piatti
 - tipica soluzione: porre un limite massimo al numero consecutivo di mosse laterali
- **massimi (minimi) locali**: eseguire scelte stocastiche e/o più ricerche da stati iniziali diversi
 - **Hill-climbing stocastico**: scegliere a caso fra tutte le mosse che migliorano h , eventualmente usando una probabilità di selezione che è proporzionale al miglioramento (convergenza + lenta, ma spesso soluzioni migliori)
 - **Hill-climbing con riavvio casuale**: esegue più ricerche a partire da stati iniziali diversi (scelti a caso). Se p è la probabilità di trovare una soluzione ottima per una singola ricerca, il numero di ricerche atteso prima di trovare una soluzione ottima è $1/p$

Numero ricerche prima di trovare una soluzione

Perché il numero di ricerche atteso prima di trovare una soluzione ottima è $1/p$??

Numero ricerche prima di trovare una soluzione

Perché il numero di ricerche atteso prima di trovare una soluzione ottima è $1/p$??

Consideriamo le variabili aleatorie binarie \mathbf{x}_i tali che

$$\mathbf{x}_i = \begin{cases} 0 & \text{se la } i\text{-esima ricerca non trova una soluzione ottima} \\ 1 & \text{se la } i\text{-esima ricerca trova una soluzione ottima} \end{cases}$$

Sappiamo che $\forall i, P(\mathbf{x}_i = 1) = p$, e $P(\mathbf{x}_i = 0) = 1 - p$

Inoltre le variabili \mathbf{x}_i sono mutuamente indipendenti:

il risultato di una ricerca non influenza il risultato di un'altra ricerca

Abbiamo quindi un **Processo di Bernoulli** per cui:

il numero di ricerche prima di trovare una soluzione ottima rispetta una **distribuzione Geometrica**

Numero ricerche prima di trovare una soluzione

Infatti, la probabilità che la k -esima ricerca (su k) sia la prima a trovare una soluzione ottima è

$$P\left(\left(\sum_{j=1}^{k-1} \mathbf{x}_j = 0\right) \wedge (\mathbf{x}_k = 1)\right) = (1 - p)^{k-1}p$$

Il valore atteso della quantità che a noi interessa è quindi

$$\begin{aligned} \sum_{k=1}^{\infty} k(1 - p)^{k-1}p &= p \sum_{k=1}^{\infty} k(1 - p)^{k-1} \\ &= p \sum_{k=1}^{\infty} \frac{d(1 - p)^k}{dp} \\ &= -p \frac{d}{dp} \left(\sum_{k=0}^{\infty} (1 - p)^k - 1 \right) = -p \frac{d}{dp} \left(\frac{1}{p} - 1 \right) \\ &= p \frac{1}{p^2} = \frac{1}{p} \end{aligned}$$

da cui deriva il risultato

Hill-climbing e le 8 regine

Numero stati: 8^8 (circa 17 milioni)

- **Hill-climbing standard**

- soluzione (ottima) trovata il 14% delle volte
- in media circa 4 passi per trovare una soluzione, altrimenti circa 3 passi in caso di soluzione subottima

- **Hill-climbing con mosse laterali (non più di 100 consecutive)**

- soluzione (ottima) trovata il 94% delle volte
- in media circa 21 passi per trovare una soluzione, altrimenti circa 64 passi in caso di soluzione subottima

- **Hill-climbing con riavvio casuale**

- soluzione (ottima) trovata con probabilità $p = 0,14$, quindi circa 7 ricerche per trovare una soluzione ottima ($(1 - p)/p = 6,14$ fallimenti + 1 successo). Numero di passi complessivo atteso: $3(1 - p)/p + 4 = 22,43$
- **con mosse laterali**: soluzione (ottima) trovata con probabilità $p = 0,94$, quindi circa 1,06 ricerche per trovare una soluzione ottima ($(1 - p)/p = 0,06$ fallimenti + 1 successo). Numero di passi complessivo atteso: $0,06(1 - p)/p + 21 = 25,08$

Simulated annealing

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

```
function SIMULATED-ANNEALING(problema, raffreddamento) returns uno stato soluzione
  inputs: problema, un problema
         velocità_raffreddamento, una corrispondenza dal tempo alla “temperatura”
  variabili locali: nodo_corrente, un nodo
                  successivo, un nodo
                  T, una “temperatura” che controlla la probabilità
                    di compiere passi verso il basso

  nodo_corrente ← CREA-NODO(STATO-INIZIALE[problema])
  for t ← 1 to ∞ do
    T ← velocità_raffreddamento[t]
    if T = 0 then return nodo_corrente
    successivo ← un successore scelto a caso di nodo_corrente
     $\Delta E$  ← VALORE[successivo] – VALORE[nodo_corrente]
    if  $\Delta E > 0$  then nodo_corrente ← successivo
    else nodo_corrente ← successivo solo con probabilità  $e^{\Delta E/T}$ 
```

Proprietà del simulated annealing

A “temperatura” fissata T , la probabilità di occupazione degli stati raggiunge la distribuzione di Boltzmann

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

T diminuito abbastanza lentamente \implies 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

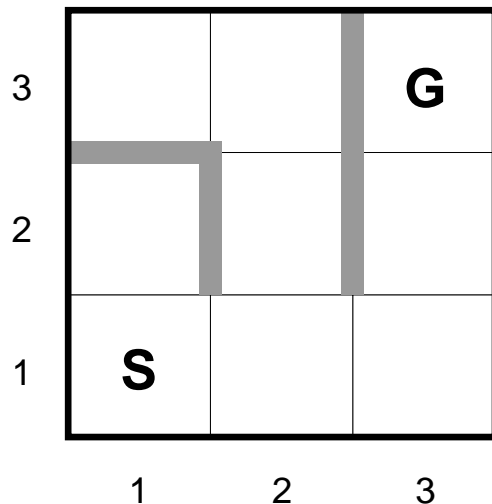
Ricerca online

Quando il problema non è totalmente osservabile o è dinamico (non-stazionario), bisogna interagire con l'ambiente per estrarre informazione:

- non è possibile pianificare a tavolino tutte le possibili azioni
- bisogna che l'agente alterni **percezione** con **azione**

In questo caso si parla di **ricerca online**

La ricerca online è particolarmente adatta a **problemi di esplorazione**



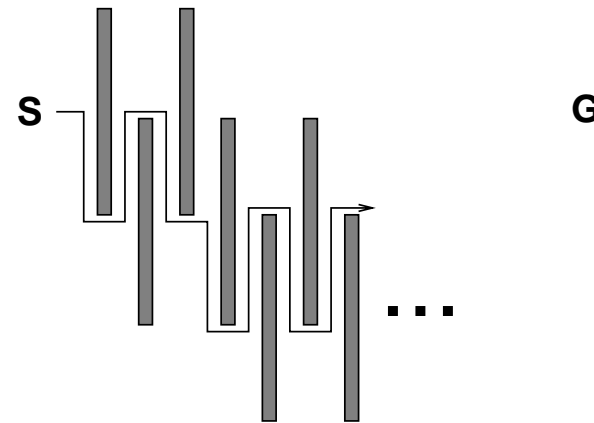
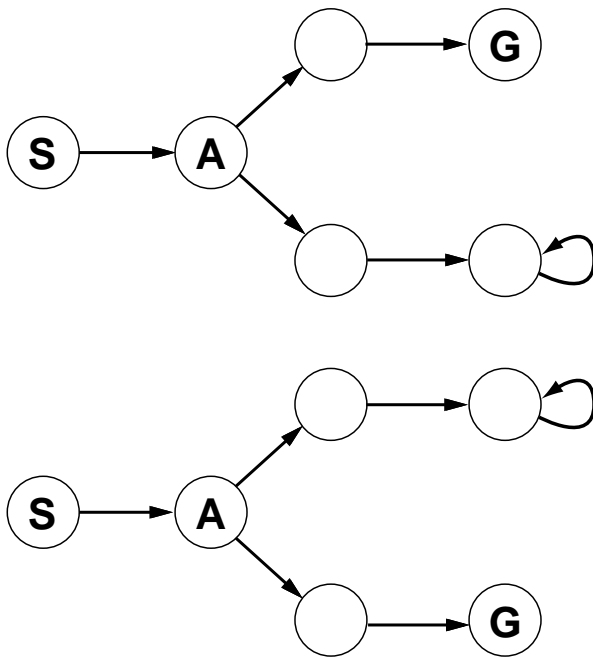
Si assume che l'agente conosca solo quanto segue:

- **AZIONI**(s), che restituisce una lista di azioni permesse nello stato s
- il costo $c(s, a, s')$ della azione a per andare dallo stato s allo stato s'
- **TEST-OBIETTIVO**(s)

Problemi della ricerca online

Poiché l'agente ha informazione parziale, rischia di finire in **vicoli ciechi**

In generale, non è possibile evitare questo problema



Essendo la ricerca online inerentemente locale, si può usare una ricerca DFS

Ricerca in profondità online

```
function ONLINE-DFS-AGENT( $s'$ ) returns an action
  inputs:  $s'$ , a percept that identifies the current state
  static:  $result$ , a table, indexed by action and state, initially empty
            $unexplored$ , a table that lists, for each visited state, the actions not yet tried
            $unbacktracked$ , a table that lists, for each visited state, the backtracks not yet tried
            $s, a$ , the previous state and action, initially null

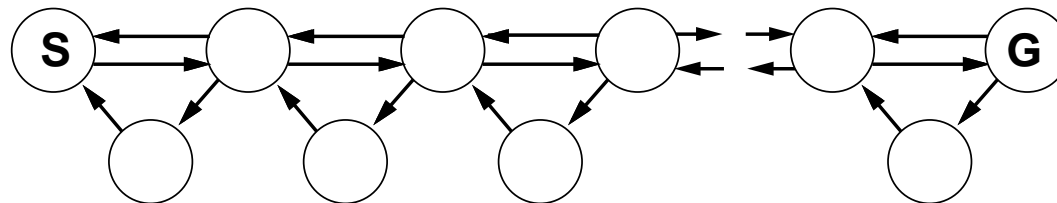
  if GOAL-TEST( $s'$ ) then return  $stop$ 
  if  $s'$  is a new state then  $unexplored[s'] \leftarrow$  ACTIONS( $s'$ )
  if  $s$  is not null then do
     $result[a, s] \leftarrow s'$ 
    add  $s$  to the front of  $unbacktracked[s']$ 
  if  $unexplored[s']$  is empty then
    if  $unbacktracked[s']$  is empty then return  $stop$ 
    else  $a \leftarrow$  an action  $b$  such that  $result[b, s'] =$  POP( $unbacktracked[s']$ )
  else  $a \leftarrow$  POP( $unexplored[s']$ )
   $s \leftarrow s'$ 
  return  $a$ 
```

Ricerca casuale

Una alternativa è data da una ricerca che opera una scelta casuale sulle azioni da intraprendere (**random walk**):

- si possono favorire le azioni non utilizzate fino a quel momento
- si può dimostrare che **prima o poi** la ricerca ha successo (in spazi finiti)

La ricerca però può essere **MOLTO** lenta, come nel seguente caso



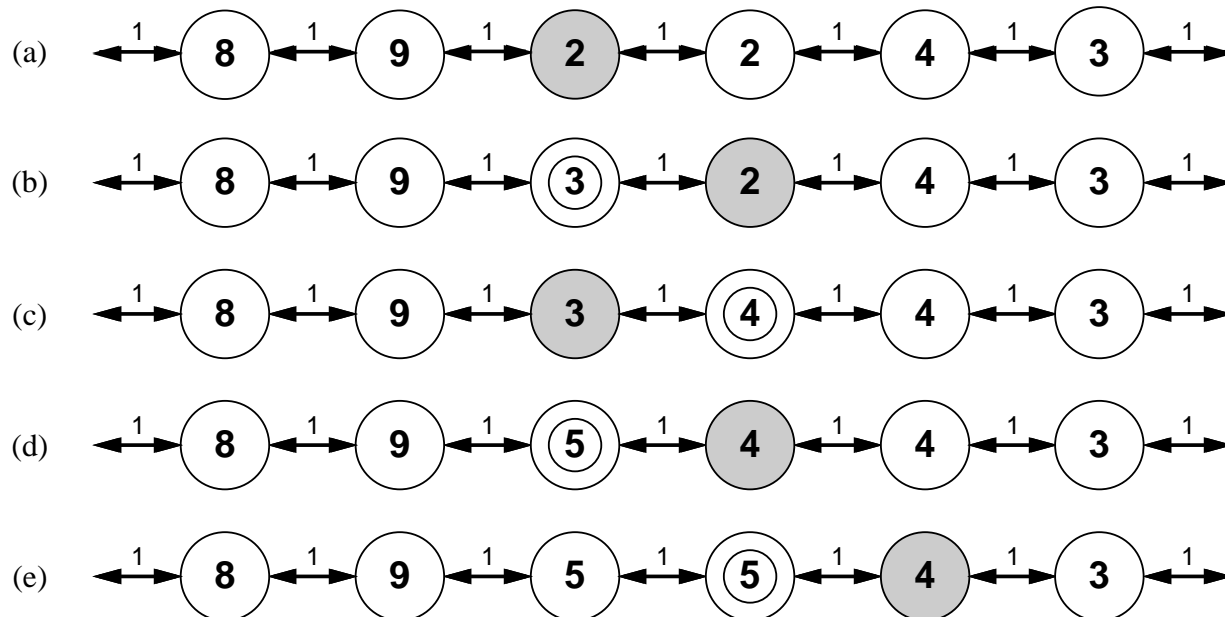
dove la probabilità di tornare indietro è doppia rispetto a quella di procedere verso il goal

Si può fare di meglio se si utilizza Hill-climbing con **memoria** e fornita di una strategia per superare gli ottimi locali: **Learning Real-Time A* (LRTA*)**

Ricerca LRTA*

L'idea di base consiste:

- nel memorizzare la **miglior stima corrente** $H(s)$ del costo per raggiungere il goal da ogni stato visitato
- $H(s)$ inizialmente coincide con $h(s)$, ma successivamente è **aggiornata con l'esperienza**



Ricerca LRTA*

function LRTA*-AGENT(s') **returns** an action

inputs: s' , a percept that identifies the current state

static: *result*, a table, indexed by action and state, initially empty

H , a table of cost estimates indexed by state, initially empty

s , a , the previous state and action, initially null

if GOAL-TEST(s') **then return** *stop*

if s' is a new state (not in H) **then** $H[s'] \leftarrow h(s')$

unless s is null

$result[a, s] \leftarrow s'$

$H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)} \text{LRTA}^*\text{-COST}(s, b, result[b, s], H)$

$a \leftarrow$ an action b in $\text{ACTIONS}(s')$ minimizing $\text{LRTA}^*\text{-COST}(s', b, result[b, s'], H)$

$s \leftarrow s'$

return a

function LRTA*-COST(s, a, s', H) **returns** a cost estimate

if s' is undefined **then return** $h(s)$

else return $c(s, a, s') + H[s']$