

Procura em Ambientes Complexos

Capítulo 4

Motivação

- Capítulo 3: ambiente completamente observável, determinístico, estático, conhecido
 - Solução = caminho = sequência de ações
- Capítulo 4: o caminho é irrelevante
 - 4.1 e 4.2: estados discretos e contínuos
 - 4.3: relaxar determinismo
 - 4.4: relaxar observabilidade
 - 4.5: ambientes desconhecidos

Resumo

- 4.1 Procura local em ambientes de otimização
 - Hill-climbing
 - Simulated annealing
 - Local beam
 - Genetic algorithms
- 4.2 Procura em espaços contínuos
- 4.3 Procura com ações não determinísticas
- 4.4 Procura em ambientes parcialmente observáveis
- 4.5 Agentes de procura online e ambientes desconhecidos

Procura Local

- Em muitos problemas de otimização, o **caminho** que leva ao objetivo é **irrelevante**; o próprio estado objetivo é a solução (e.g., n-rainhas)
- Nestes casos, podemos usar **procura local**

Procura Local

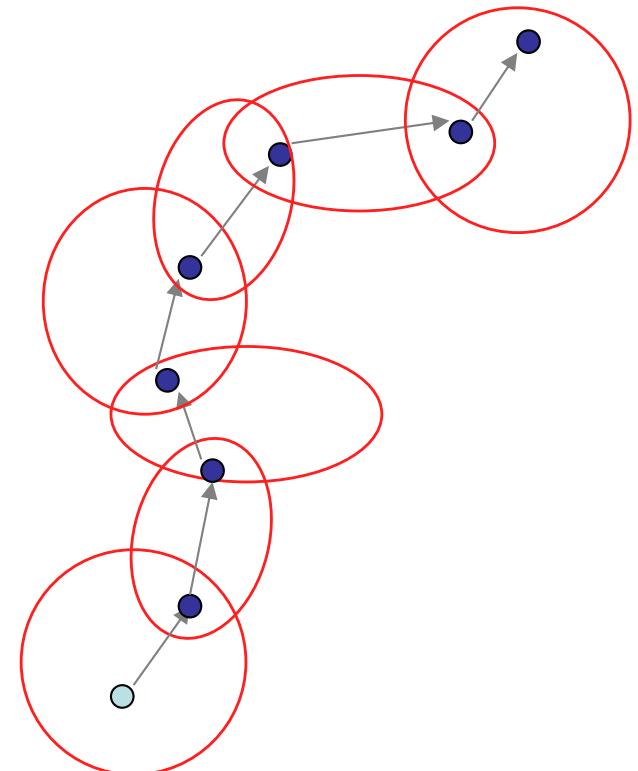
- Começa com um **estado inicial**
- Em cada iteração atualiza para **estado “vizinho”**
 - Na expectativa de melhorar estado
- Não mantém **registro do caminho** ou de estados já visitados
- Significado de **“melhorar” o estado atual**
 - Dependendo da função de avaliação usada podemos escolher o menor ou o maior valor
 - Se usarmos uma função de qualidade escolhemos o valor mais elevado
 - Se usarmos uma função de custo / heurística escolhemos o menor valor

Procura local vs sistemática

- Prós
 - Menos requisitos de memória
- Contras
 - Pode nunca encontrar uma solução (e nunca prova que não há solução)
- Na prática: encontra muitas vezes solução quando procura sistemática não consegue
- Adequada para resolver problemas de otimização

Procura Local

- Mantém um único “estado atual”; caminhos não são memorizados
 - Em cada iteração procura “melhorar” o estado atual; útil em otimização
 - Tipicamente, um estado transita para estados “vizinhos”



Exemplo: *N*-raínhas

- Problema: Colocar as N raínhas numa matriz $n \times n$ de modo que nenhuma esteja em posição de atacar as outras



- Estado inicial gerado **aleatoriamente**
- Novos estados gerados a partir de **movimentos para estados vizinhos**

Hill-climbing (trepa colinas) ou procura local gananciosa

- É um simples ciclo que se **move continuamente** na direção de um **valor melhor**. Termina quando nenhum sucessor tem valores melhores
- Podemos considerar **dois tipos de valores**, dependendo do problema
 - Mais elevado melhor (mais frequente)
 - Procura pelo maior valor
 - Menos elevado melhor
 - Procura pelo menor valor

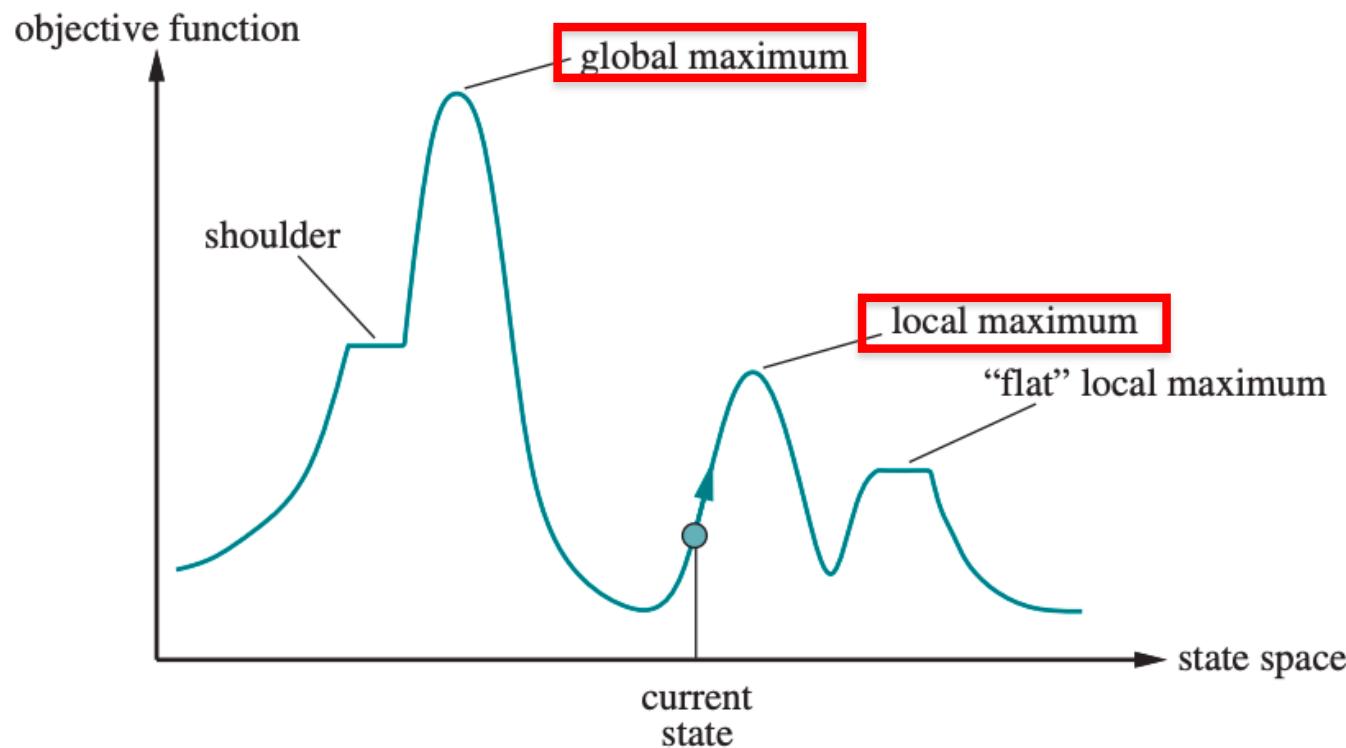
Hill-climbing

- “É como subir o Evereste com nevoeiro cerrado e amnésia”

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current  $\leftarrow$  problem.INITIAL
  while true do
    neighbor  $\leftarrow$  a highest-valued successor state of current
    if VALUE(neighbor)  $\leq$  VALUE(current) then return current
    current  $\leftarrow$  neighbor
```

Hill-climbing

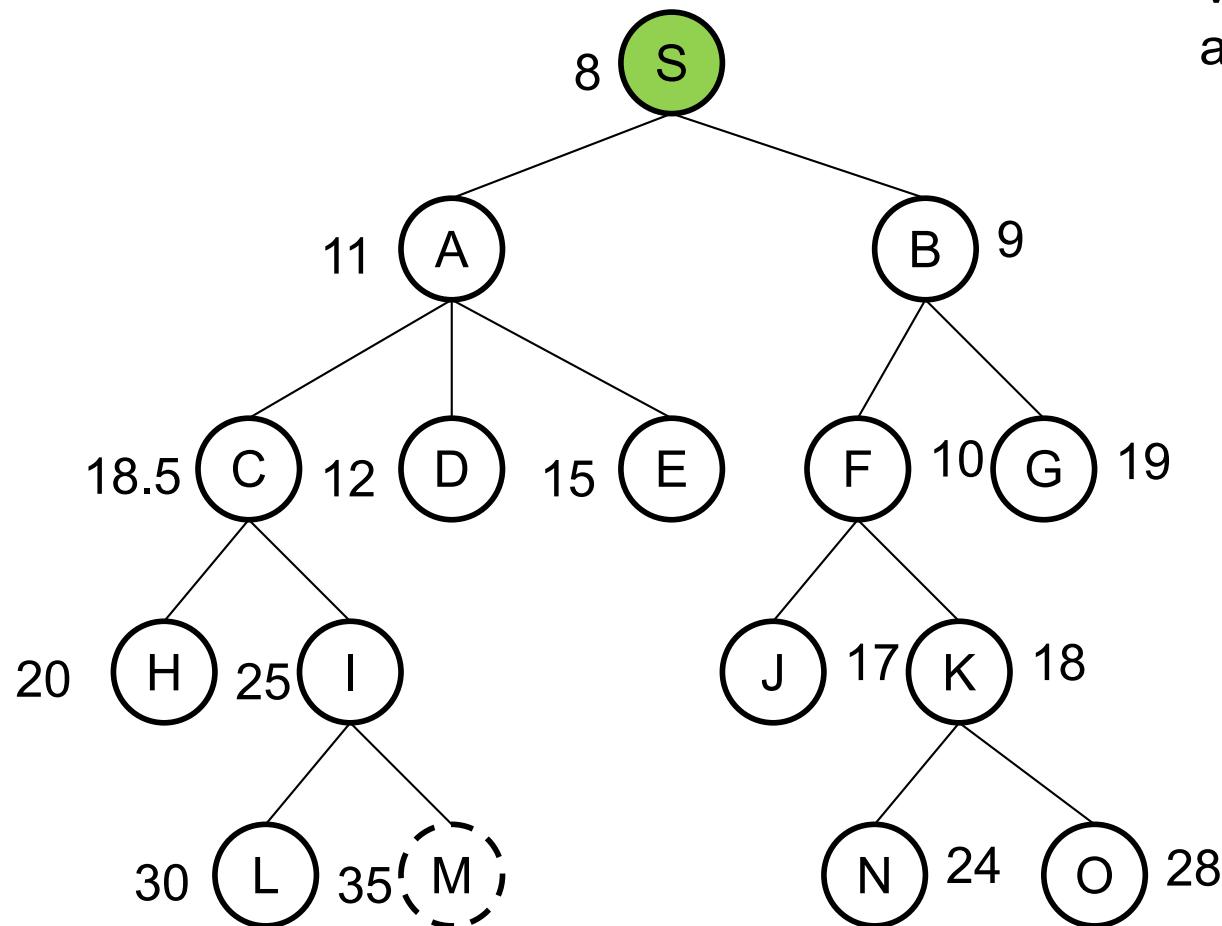
- Problema: dependendo do estado inicial, pode ficar preso a um máximo local



Hill-Climbing: exemplo

Objetivo: encontrar caminho de **S** para **M**

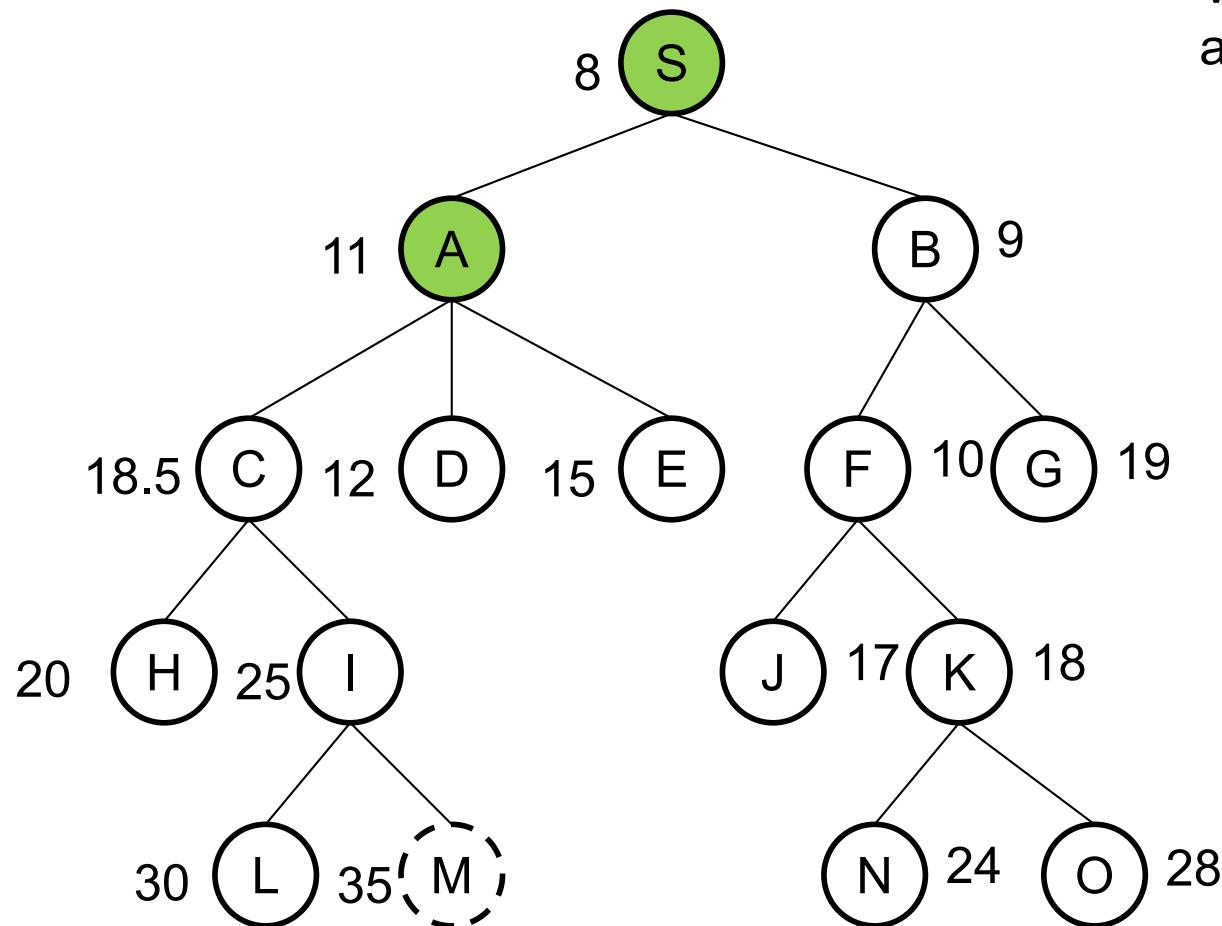
Valor associado
a cada nó



Hill-Climbing: exemplo

Objetivo: encontrar caminho de **S** para **M**

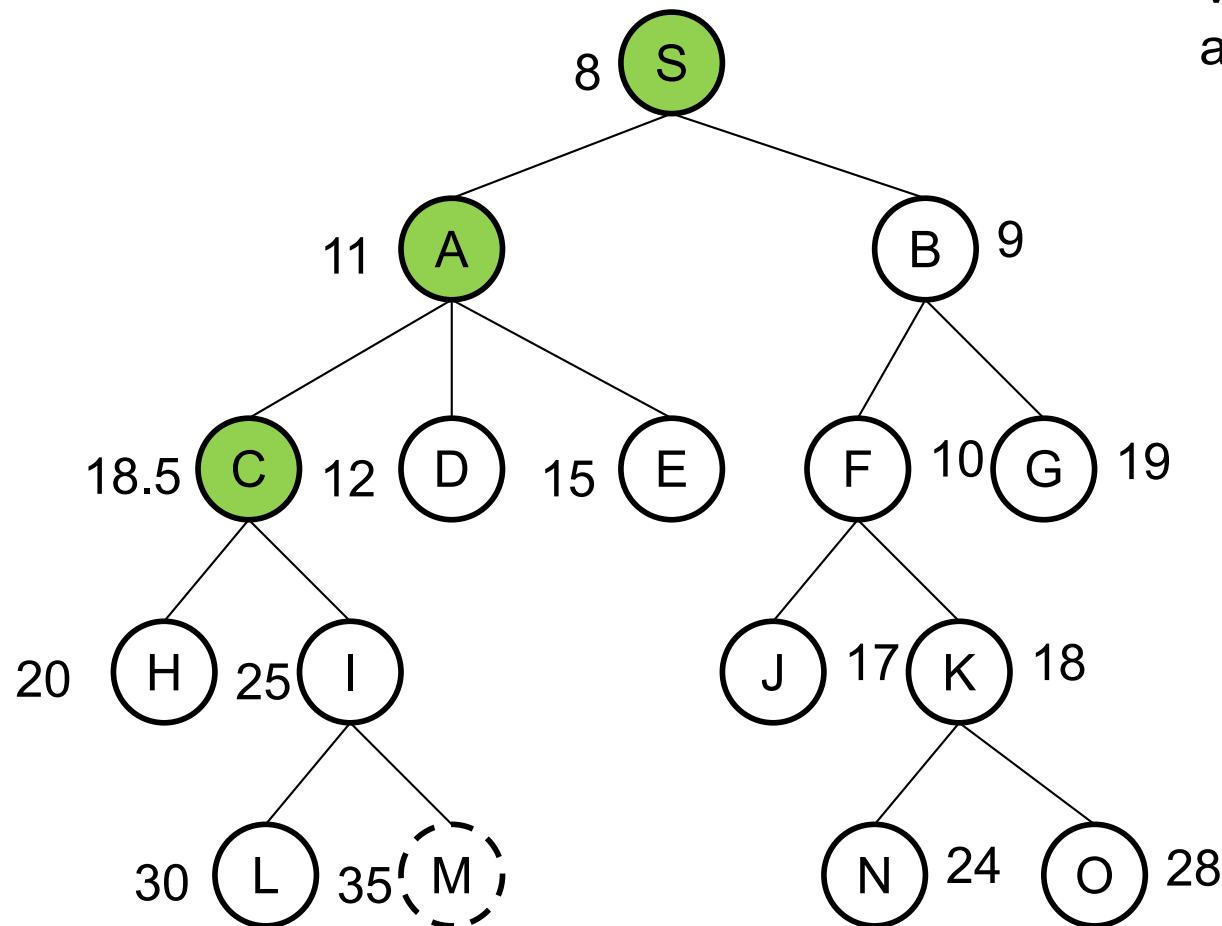
Valor associado
a cada nó



Hill-Climbing: exemplo

Objetivo: encontrar caminho de **S** para **M**

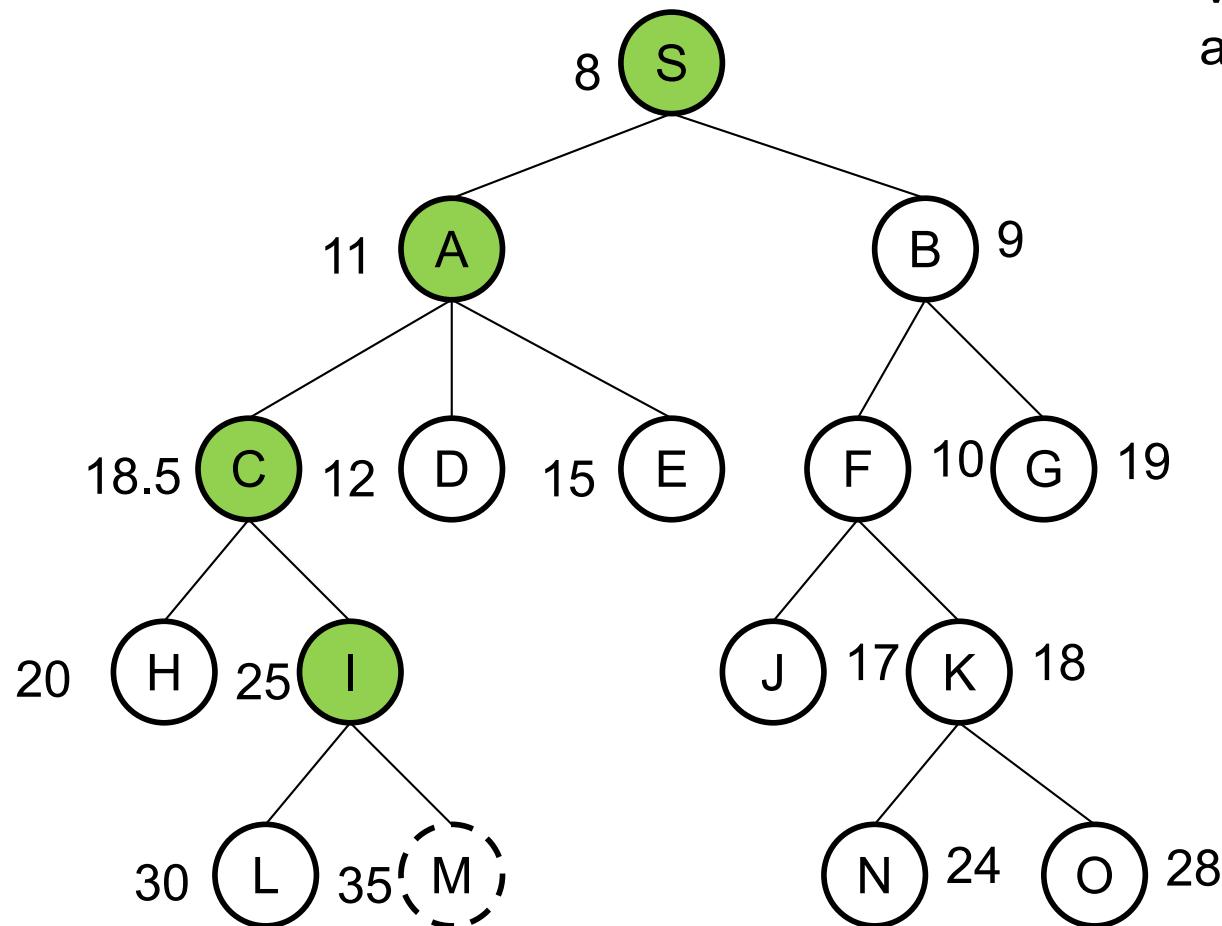
Valor associado
a cada nó



Hill-Climbing: exemplo

Objetivo: encontrar caminho de **S** para **M**

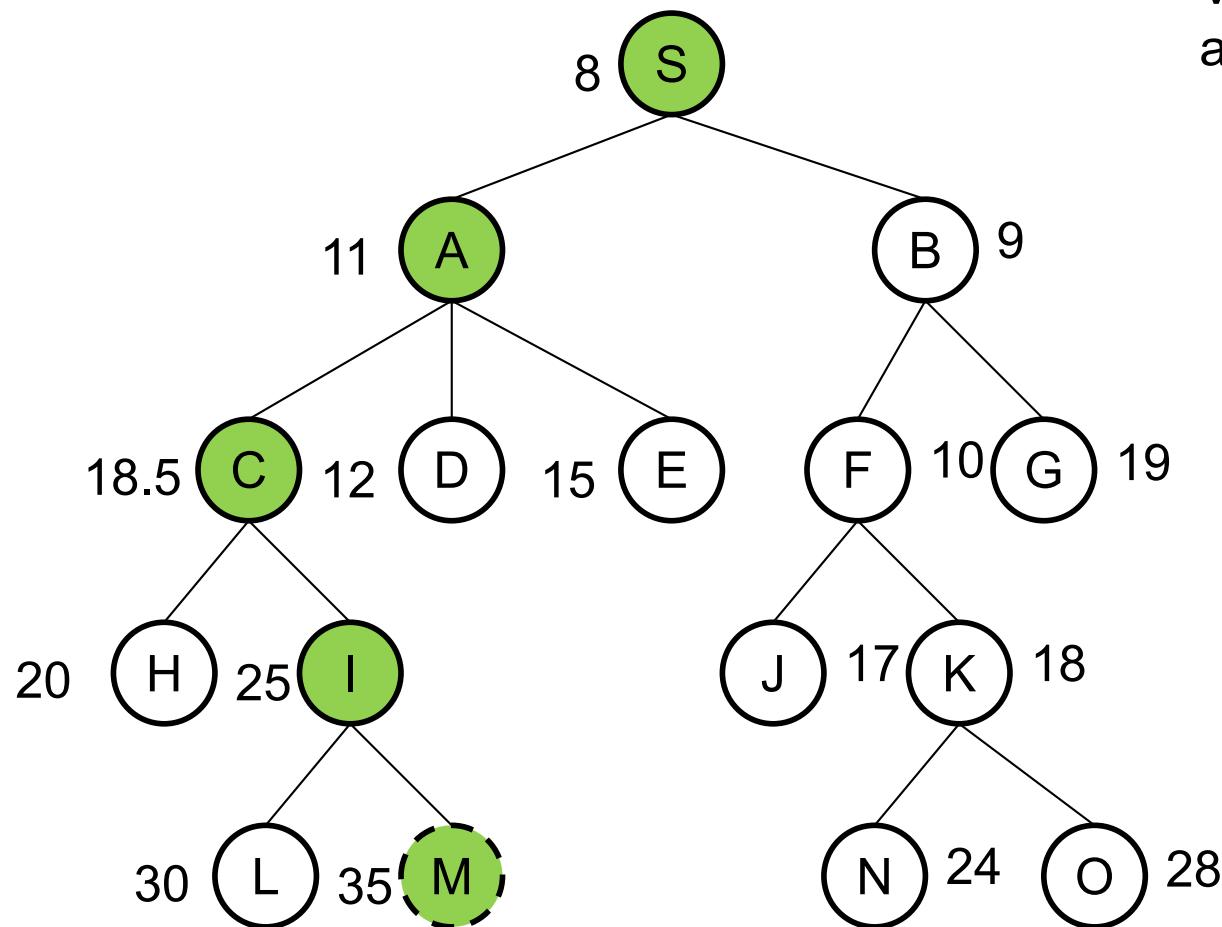
Valor associado
a cada nó



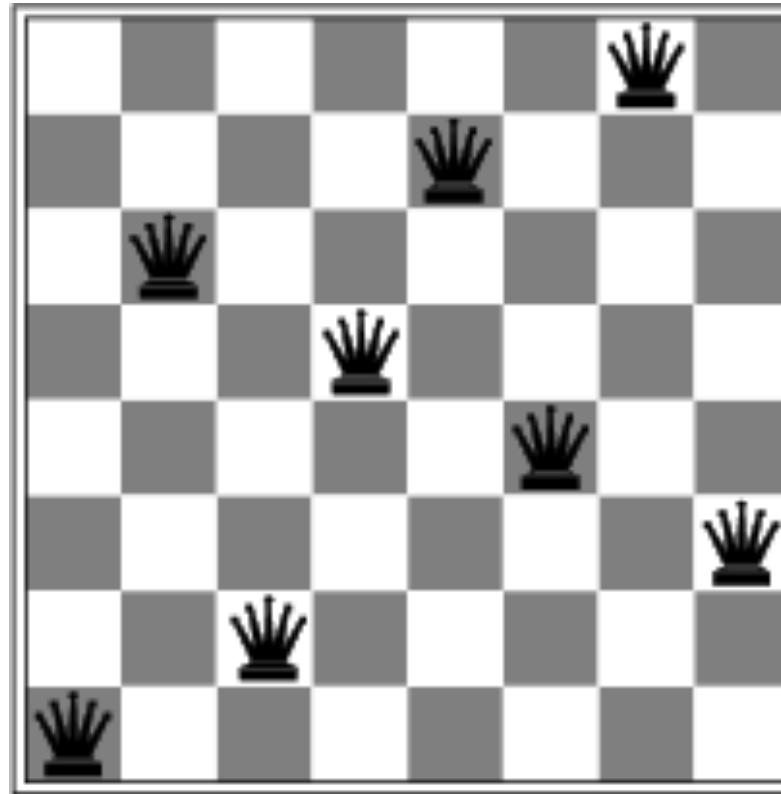
Hill-Climbing: exemplo

Objetivo: encontrar caminho de **S** para **M**

Valor associado
a cada nó

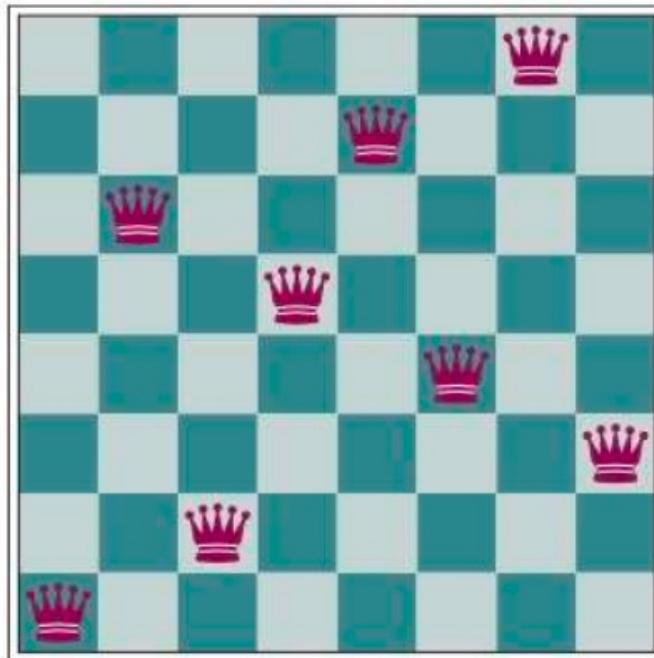


Procura com o Hill-climbing no problema das 8 rainhas



- Função sucessor: mexer uma rainha para outra posição na mesma coluna.

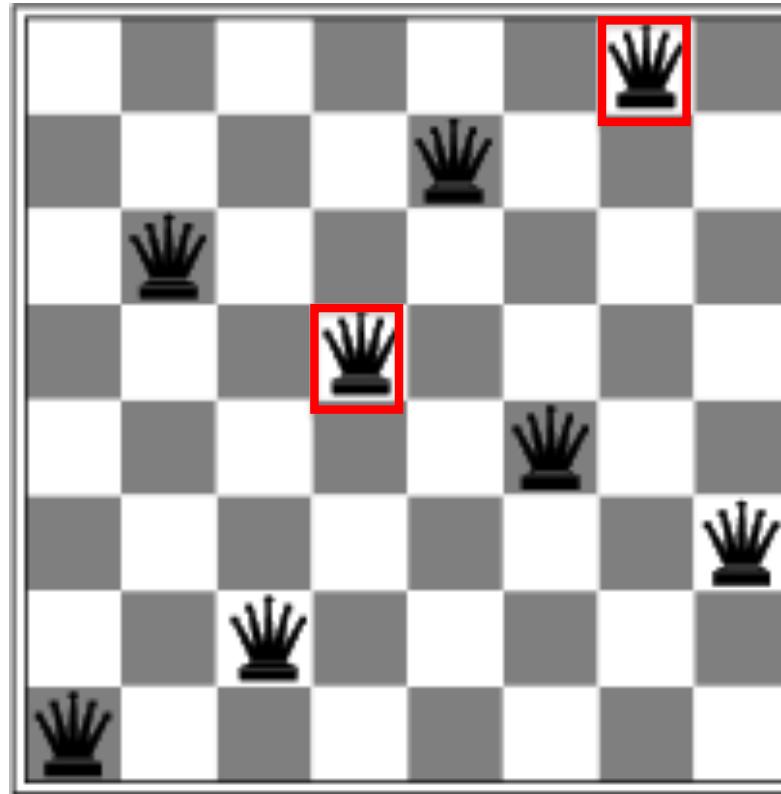
Procura com o Hill-climbing no problema das 8 rainhas



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

- $h = \text{nº de pares de rainhas que se estão a atacar}$
 - Tabuleiro esquerda: $h = 1$
 - Tabuleiro direita: $h = 17$
 - Tabuleiro objectivo: $h = 0$
- Tabuleiro direita: valor de h para sucessores resultantes de mover rainha na respectiva coluna (≥ 12)

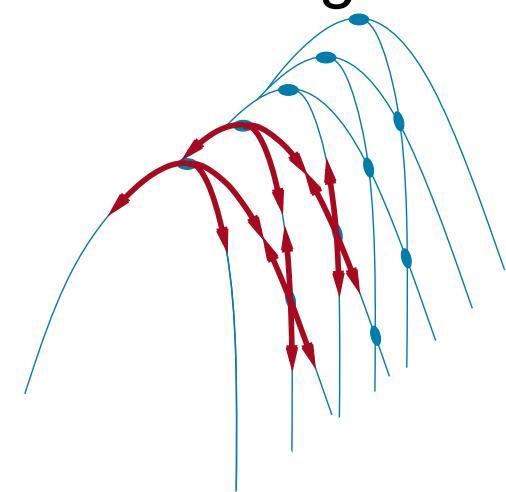
Procura com o Hill-climbing no problema das 8 rainhas



- Mínimo local com $h = 1$
- Qualquer sucessor tem valor de h superior
Não encontrou solução e não evoluiu para outro estado...

Hill Climbing: problemas

- Máximos locais
 - Melhores que vizinhos mas piores que máximo global
- Ridges (cumes)
 - Sequência de máximos locais
- Plateaus (planaltos)
 - Superfície plana como *shoulder* (pode melhorar) ou *flat local maximum* (não pode melhorar)
 - Permitir (nº limitado de) *sideway moves* para sair de *shoulder*



Hill Climbing

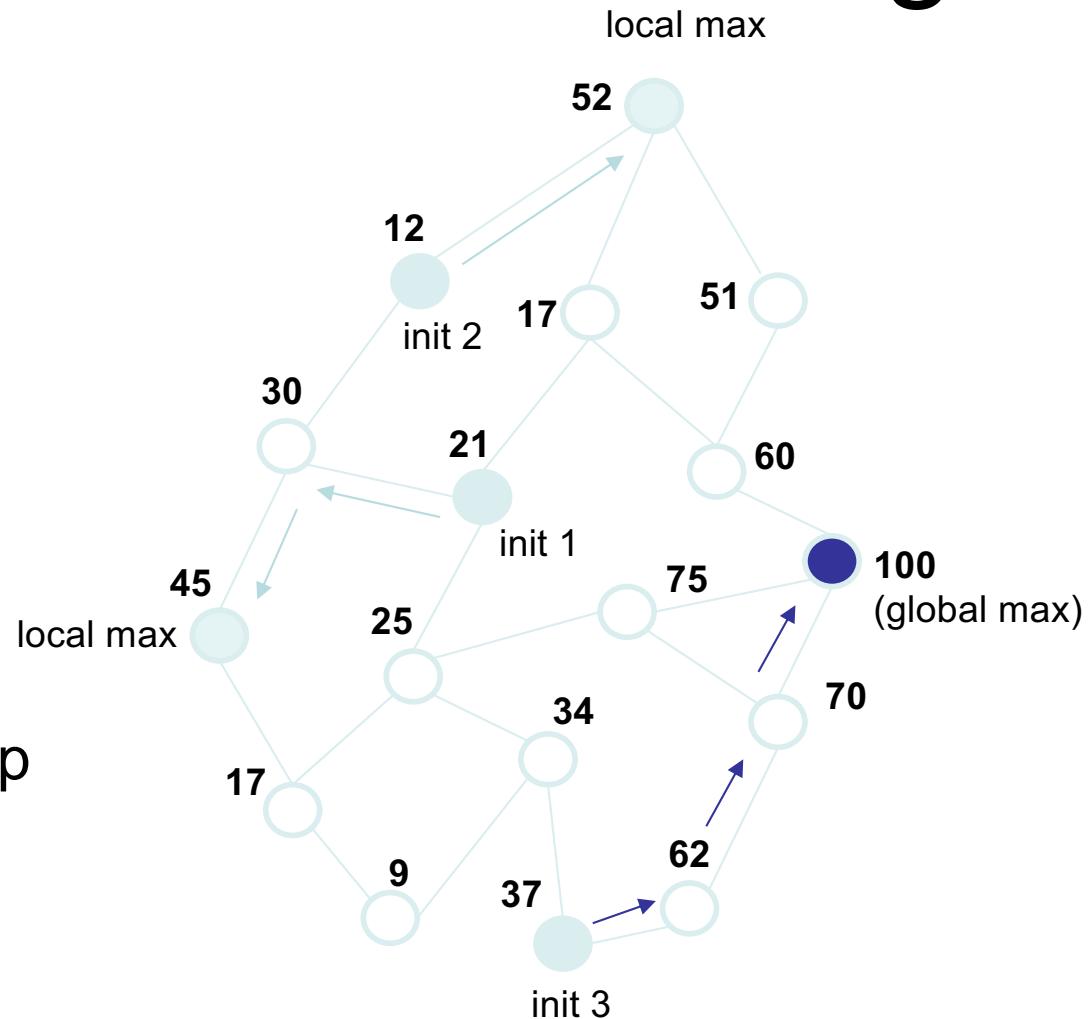
- No caso das 8-rainhas
 - Começando num estado aleatório...
 - Só resolve 14% dos casos (necessita em média de 4 iterações)
 - Nos restantes casos fica “parado” ao fim de 3 iterações (em média)

Variantes do Hill Climbing

- **Stochastic Hill climbing**: escolhe aleatoriamente de entre sucessores melhores que estado atual
- **First-choice Hill Climbing**: gera os sucessores aleatoriamente até encontrar o primeiro com valor melhor que o estado atual que é escolhido (conveniente se um estado tiver milhares de possíveis sucessores)
- **Random-restart Hill Climbing**: conduz uma série de procura a partir de diferentes estados iniciais, gerados aleatoriamente; pára quando se encontra o objectivo

Random Restart Hill Climbing

- Se uma procura falha, reiniciar com um novo estado inicial
 - Repetir até encontrar máximo global (100)
- Se hill-climbing tem probabilidade p de sucesso, então o nº esperado de restarts é $1/p$
 - 7 iterações para as 8-rainhas!



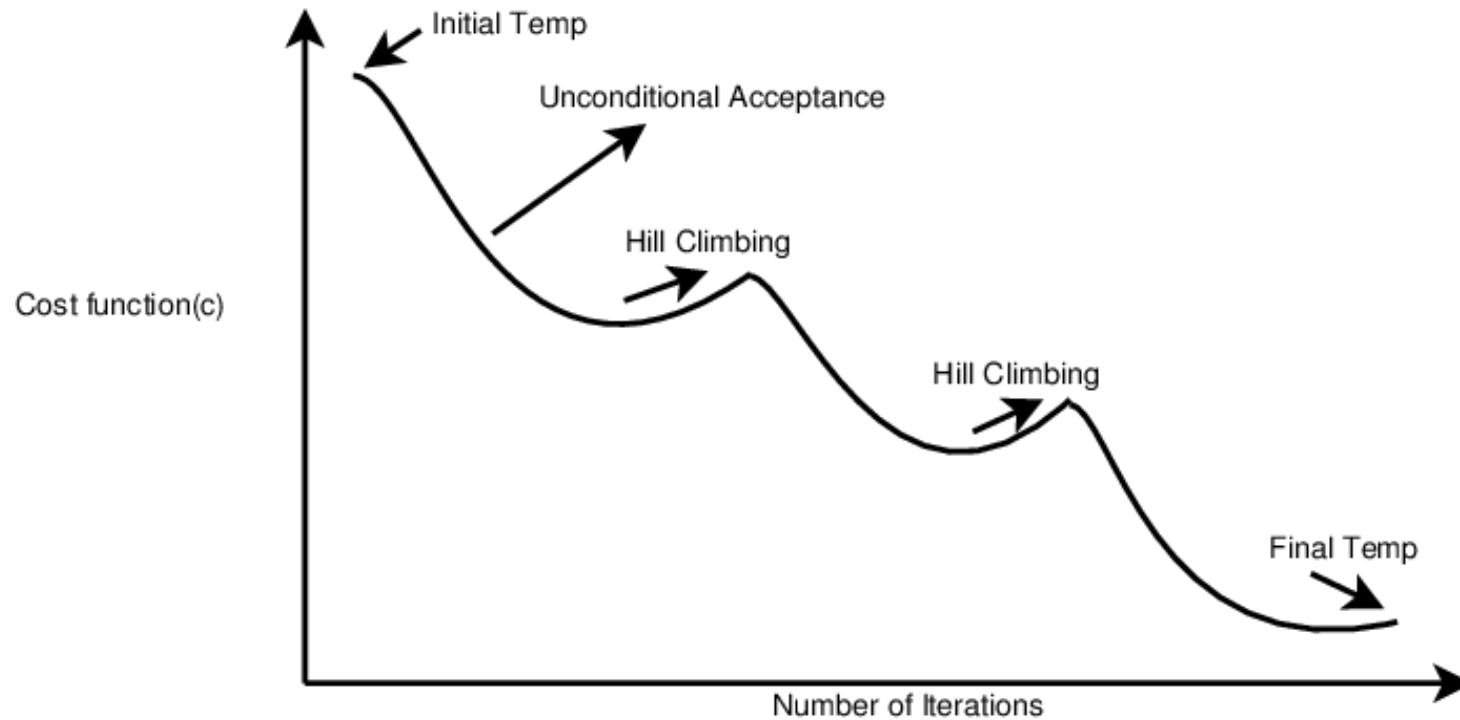
Hill Climbing

- Apesar de tudo:
 - Converge (ou não) rapidamente
 - Por exemplo, o Random-restart Hill Climbing consegue encontrar uma solução para as n-rainhas, em menos de um minuto, mesmo para 3 milhões de rainhas

Simulated Annealing

- Assumimos função de custo
 - i.e. mínimos locais / mínimo global
- Ideia: **escapar aos mínimos locais** permitindo que se façam movimentos “maus”, mas vai gradualmente decrementando a sua frequência
 - Em vez de escolher o melhor sucessor, escolhe um sucessor aleatoriamente que *tipicamente* é “aceite” se melhorar a situação
 - Em Português: têmpora simulada

Simulated Annealing: exemplo



Exemplo assume mínimo local (vs máximo local)

Simulated Annealing

- Consegue-se provar que se temperatura T diminuir suficientemente devagar (em função do *schedule*), então a procura simulated annealing vai encontrar um mínimo global com probabilidade próxima do 1

Simulated Annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  current  $\leftarrow$  problem.INITIAL
  for t = 1 to  $\infty$  do
    T  $\leftarrow$  schedule(t)
    if T = 0 then return current
    next  $\leftarrow$  a randomly selected successor of current
     $\Delta E \leftarrow \text{VALUE}(\textit{current}) - \text{VALUE}(\textit{next})$ 
    if  $\Delta E > 0$  then current  $\leftarrow$  next
    else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$ 
```

Probabilidade de aceitar “mau” sucessor diminui exponencialmente
com a falta de qualidade do sucessor ($\Delta E < 0$)
com a diminuição de T

Simulated Annealing

- Metáfora: imaginar a tarefa de pôr uma bola de ping-pong no buraco mais profundo de uma superfície cheia de buracos
- Uma solução é deixar a bola ir parar a um mínimo local e depois abanar a superfície de modo a tirá-la do mínimo local
- Simulated annealing começa por “abanar” muito no início e depois vai abanando cada vez menos
 - Na expectativa de não tirar a bola do máximo global

Local Beam (procura em banda)

- Guarda a **referência a k estados**, em vez de 1
 - Começa com k estados gerados aleatoriamente
- Em cada iteração, todos os sucessores dos k estados são gerados
- Se algum é um estado objetivo, pára; caso contrário **escolhe os k melhores sucessores** e repete

Procura Local Beam

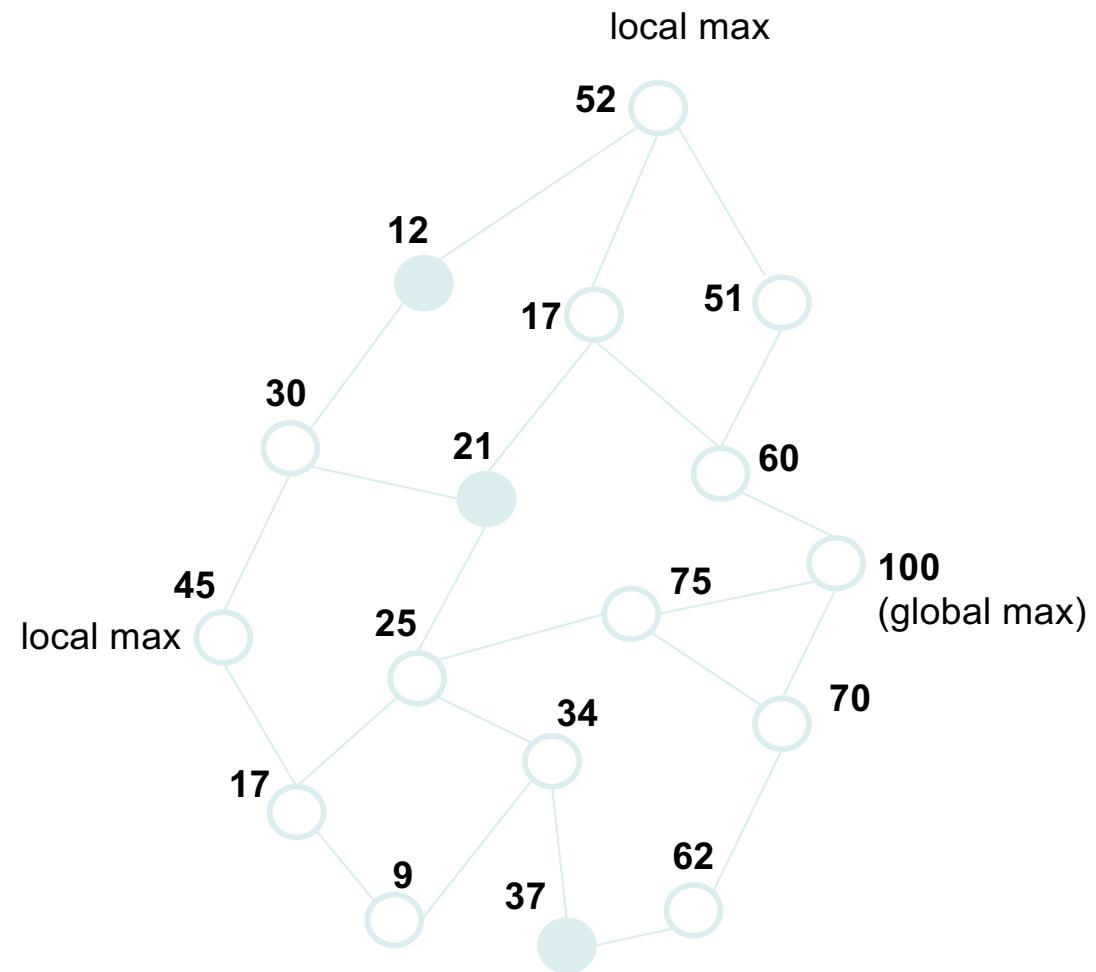
```
function BEAM-SEARCH(problem, k) returns a solution state
    start with k randomly generated states
    loop
        generate all successors of all k states
        if any of them is a solution then return it
        else select the k best successors
```

Procura Local Beam

- Atenção que este algoritmo é mais do que correr k Random-restart Hill Climbs em paralelo!!!
 - Não têm de ser escolhidos sucessores de todos os estados
 - Se um estado gera vários bons sucessores e os outros $k-1$ estados não, os estados menos promissores são abandonados

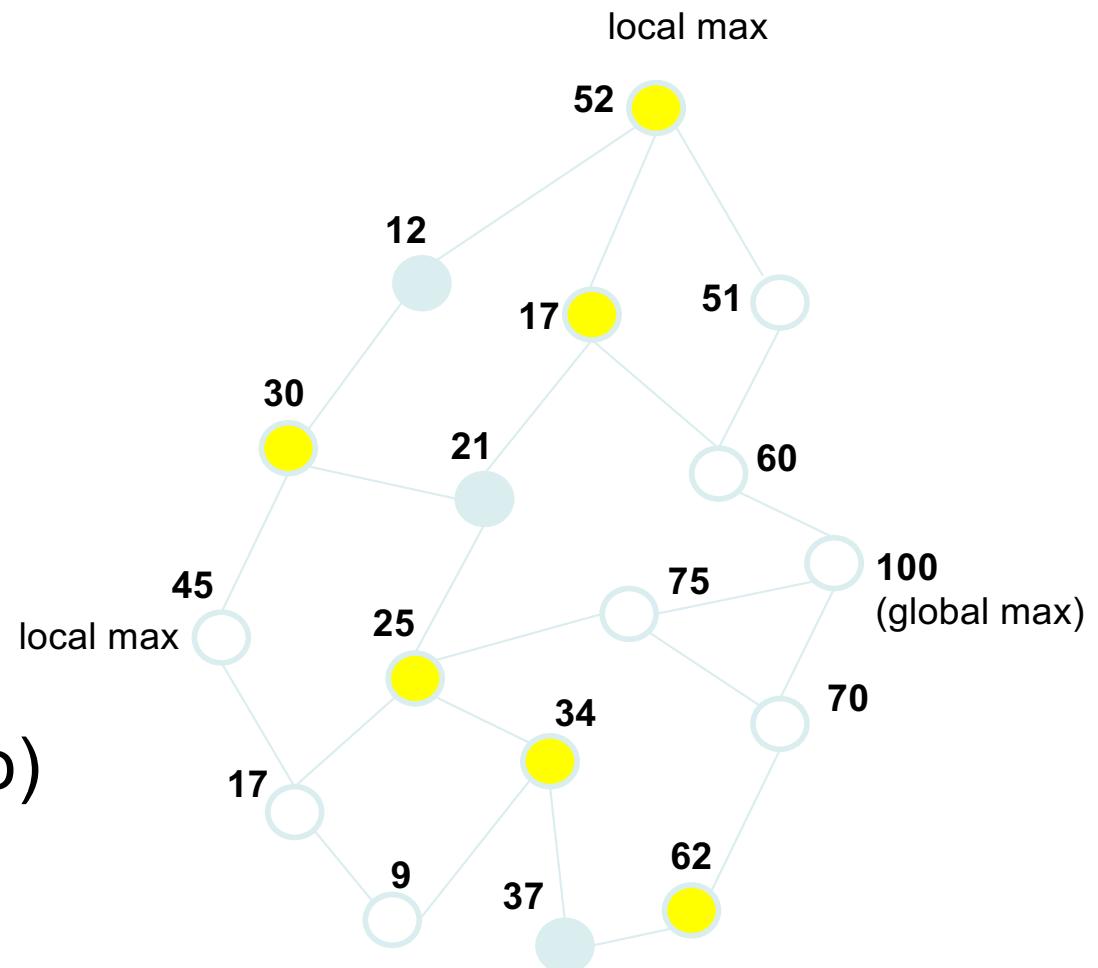
Procura Local Beam

- Começar com k estados gerados aleatoriamente
- Gerar todos os sucessores destes k estados
- Se o objetivo não é encontrado, selecionar os k melhores sucessores e repetir
- Para este exº usar $k = 3$



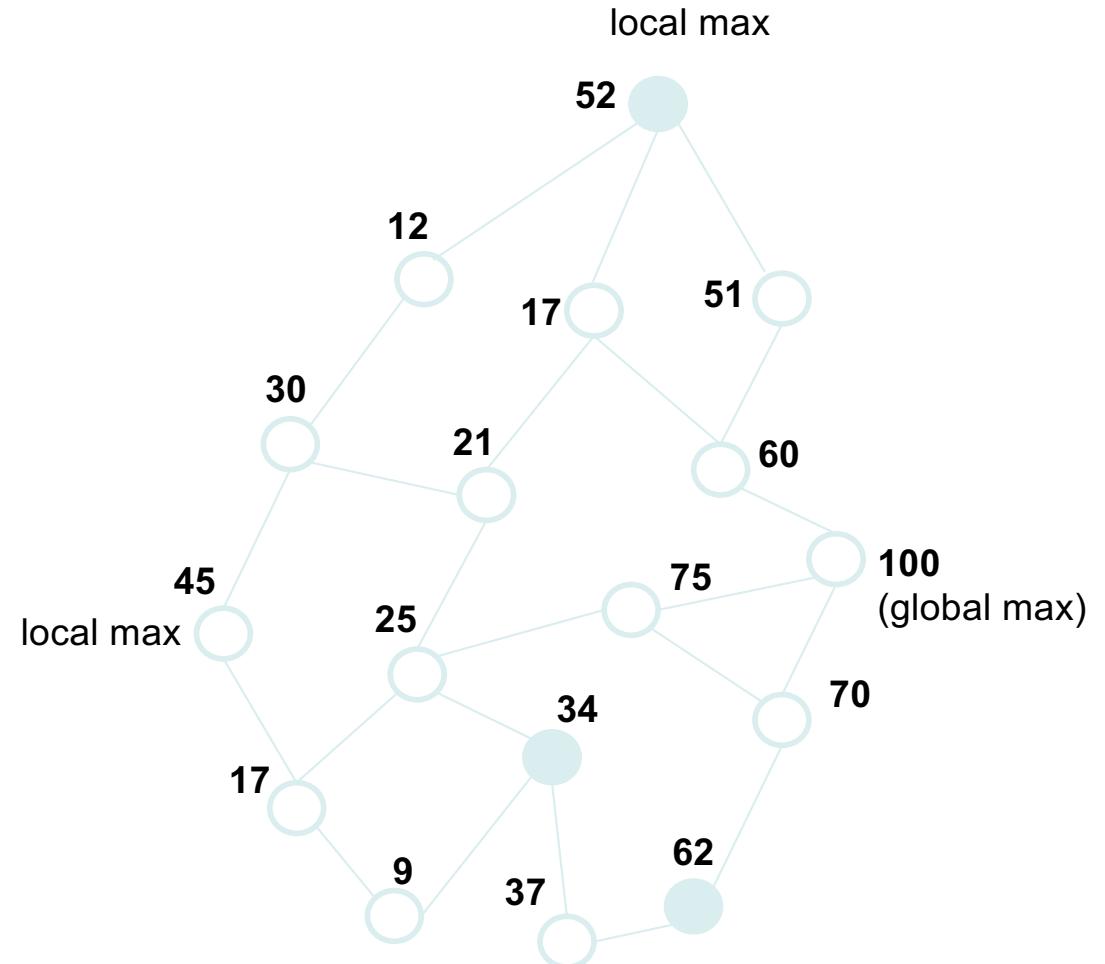
Procura Local Beam

- Gerar todos os sucessores dos k estados
- Entre estes sucessores (amarelo) escolher top 3



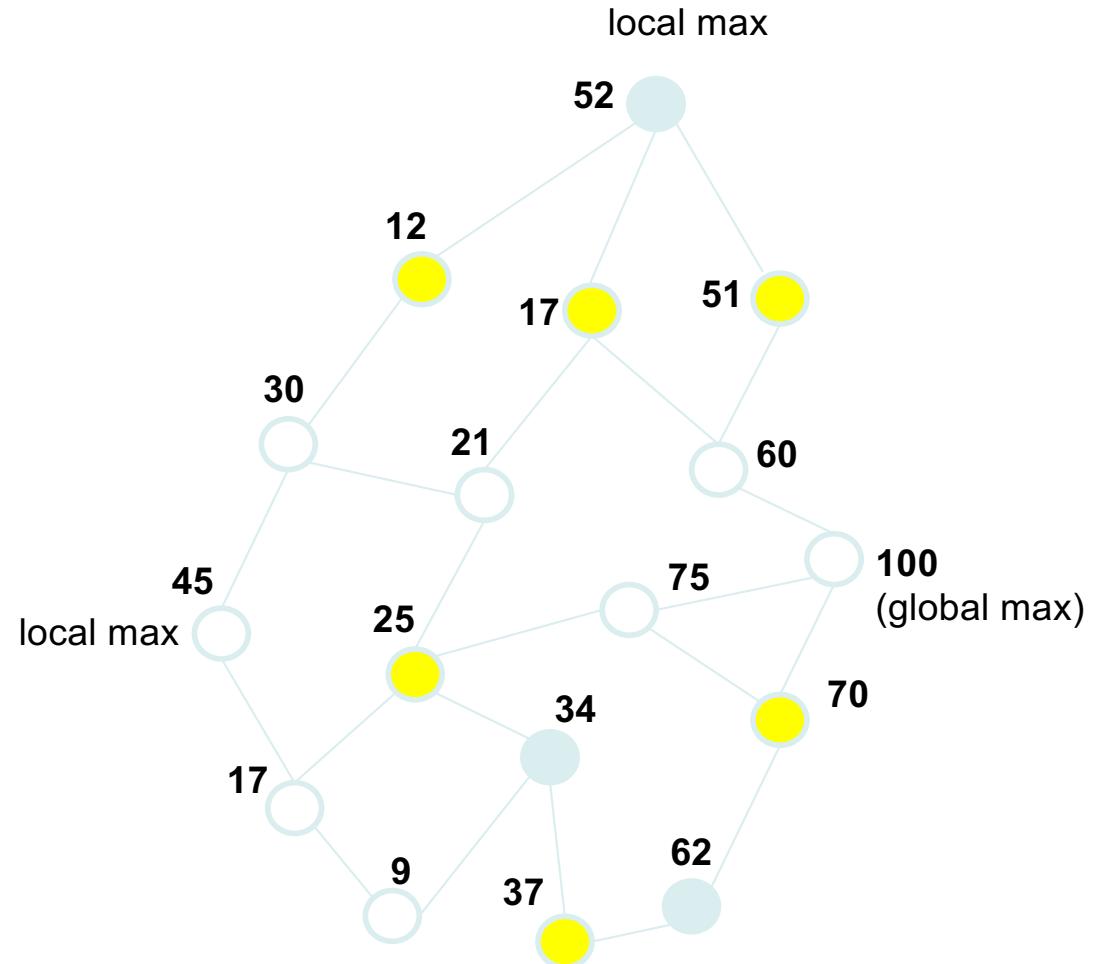
Procura Local Beam

- Para estes 3, repetir o mesmo procedimento até encontrar máximo global



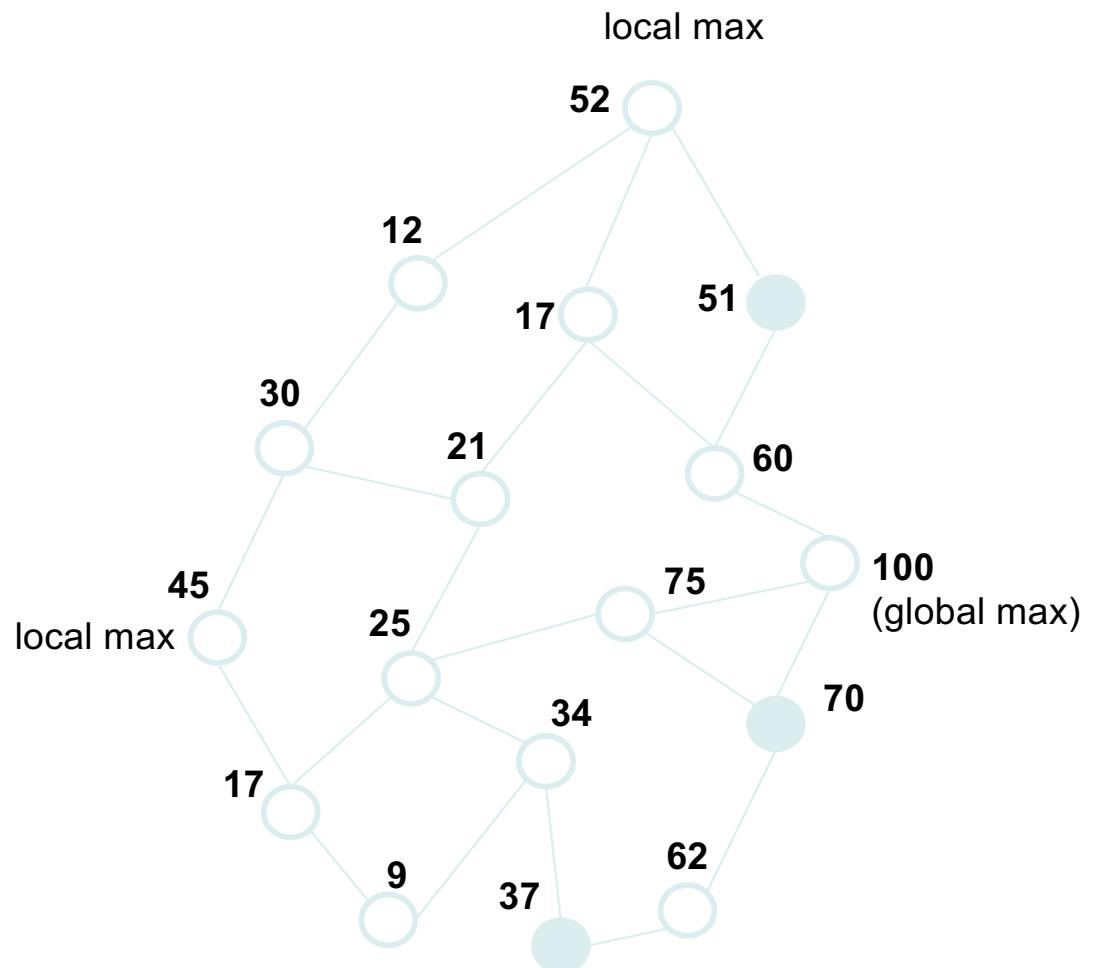
Procura Local Beam

- Novos sucessores
são gerados



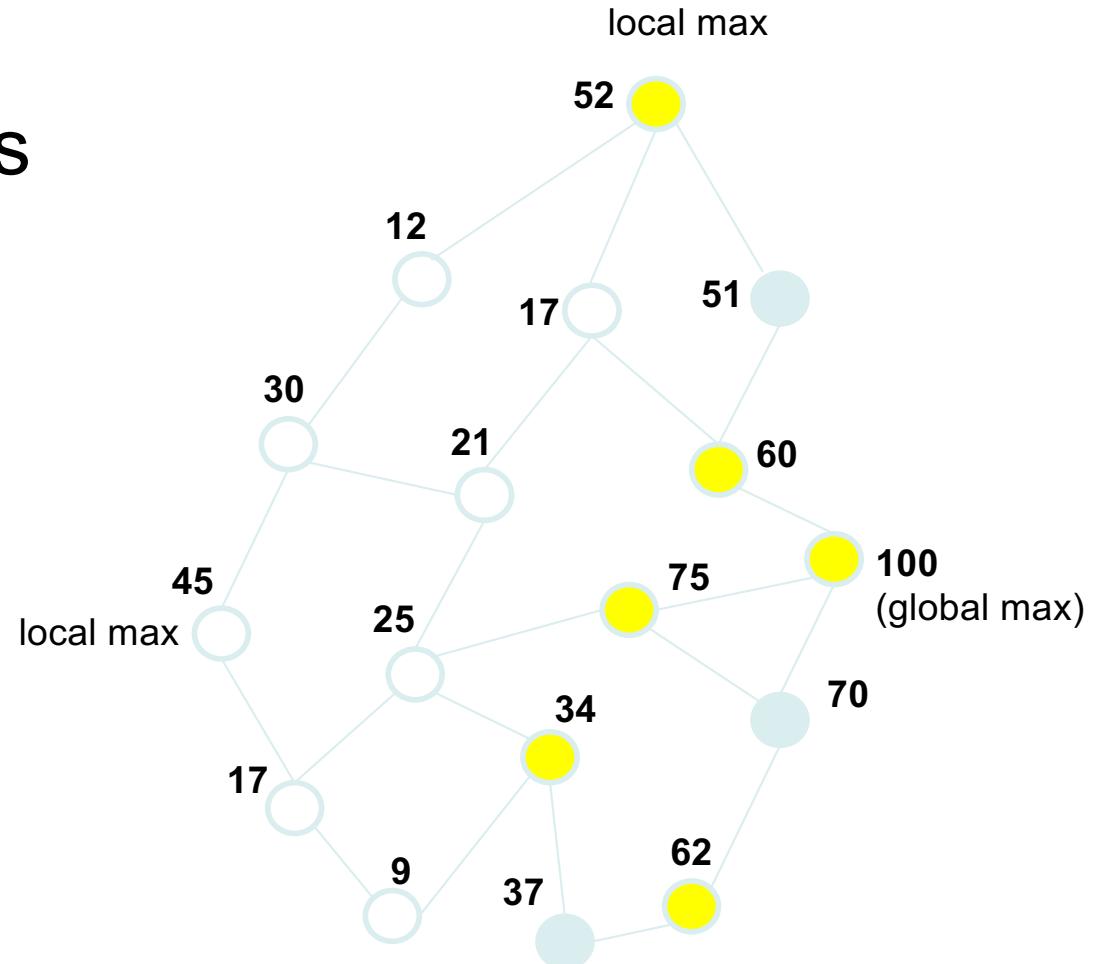
Procura Local Beam

- Os 3 melhores são selecionados



Procura Local Beam

- Um dos sucessores é o objetivo (máximo global) !



Procura Local Beam

- No entanto, também pode ter problemas: pode haver pouca diversidade nos k estados...
 - **Stochastic Beam Search**: k sucessores são escolhidos aleatoriamente

Algoritmos Genéticos/Evolutivos

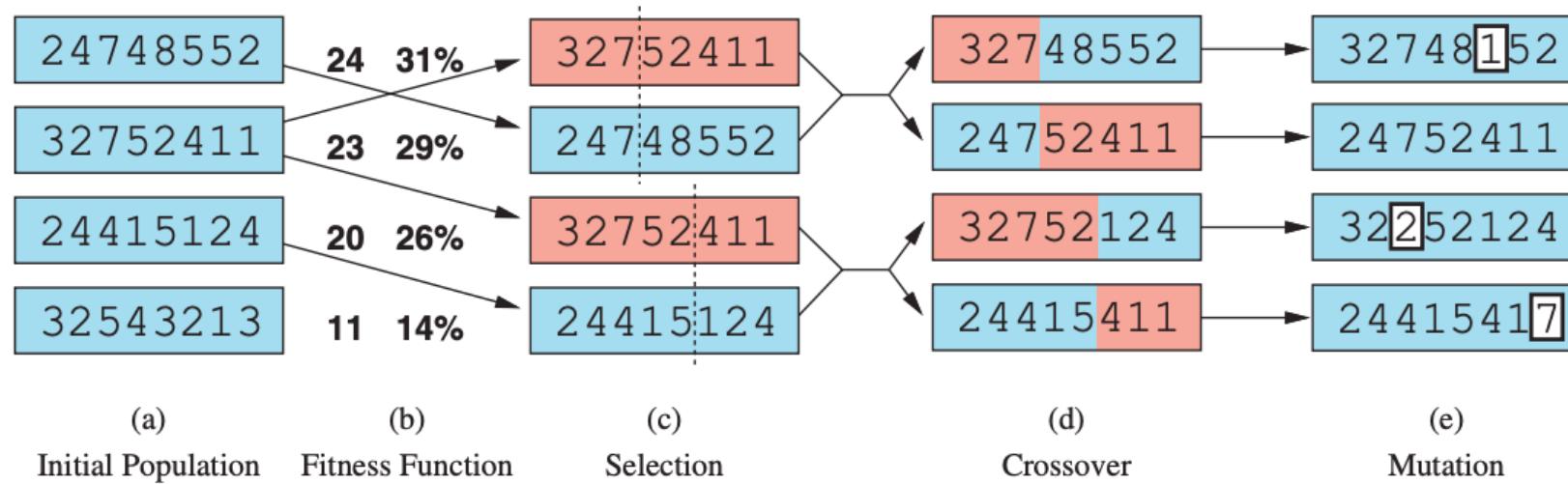
- Variante da *stochastic beam search*
- Começa com k estados gerados aleatoriamente (**população**) tal como procura em banda
 - Um estado é representado como uma string sobre um alfabeto finito (geralmente {0,1})
- O estado sucessor é gerado através da combinação de dois estados (**pais**)
 - Produz a próxima geração de estados por seleção, cruzamento e mutação
 - A função de avaliação (**fitness function**) dá valores mais altos aos melhores estados

Algoritmos Genéticos

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights  $\leftarrow$  WEIGHTED-BY(population, fitness)
    population2  $\leftarrow$  empty list
    for i = 1 to SIZE(population) do
      parent1, parent2  $\leftarrow$  WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child  $\leftarrow$  REPRODUCE(parent1, parent2)
      if (small random probability) then child  $\leftarrow$  MUTATE(child)
      add child to population2
    population  $\leftarrow$  population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness
```

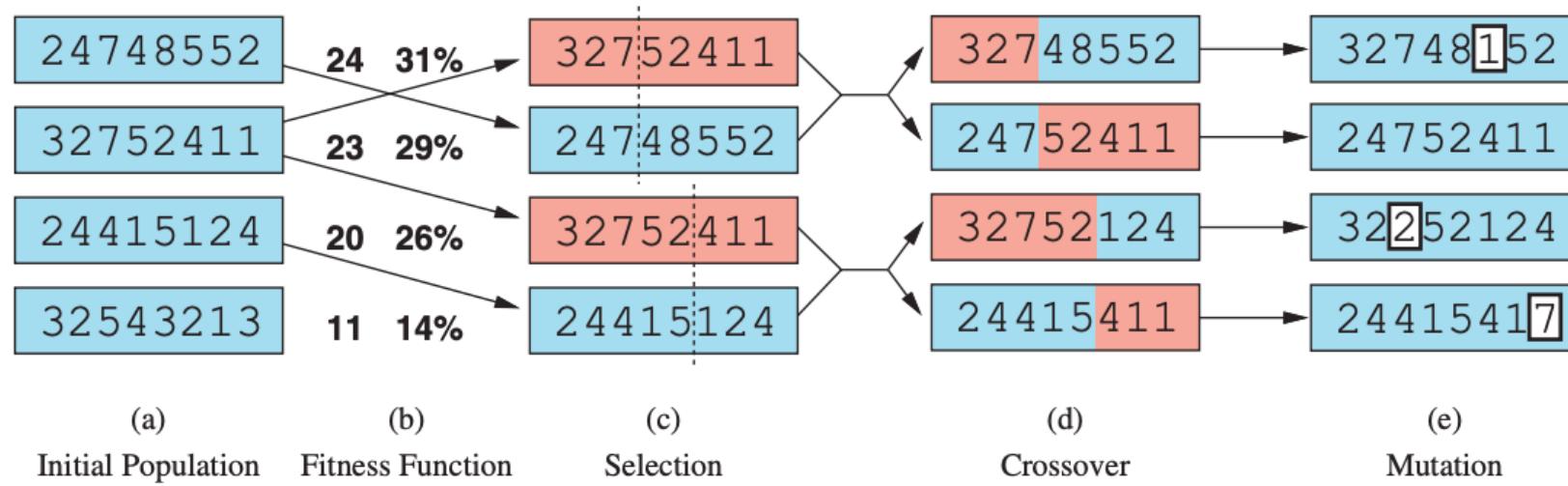
```
function REPRODUCE(parent1, parent2) returns an individual
  n  $\leftarrow$  LENGTH(parent1)
  c  $\leftarrow$  random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Algoritmos Genéticos



- Função de Fitness (b): nº de pares de rainhas **não atacantes** ($\min = 0$, $\max = (8 \times 7)/2 = 28$)
 - Probabilidade de seleção (c) dependente da função de fitness
 - Exº $24/(24+23+20+11) = 31\%$

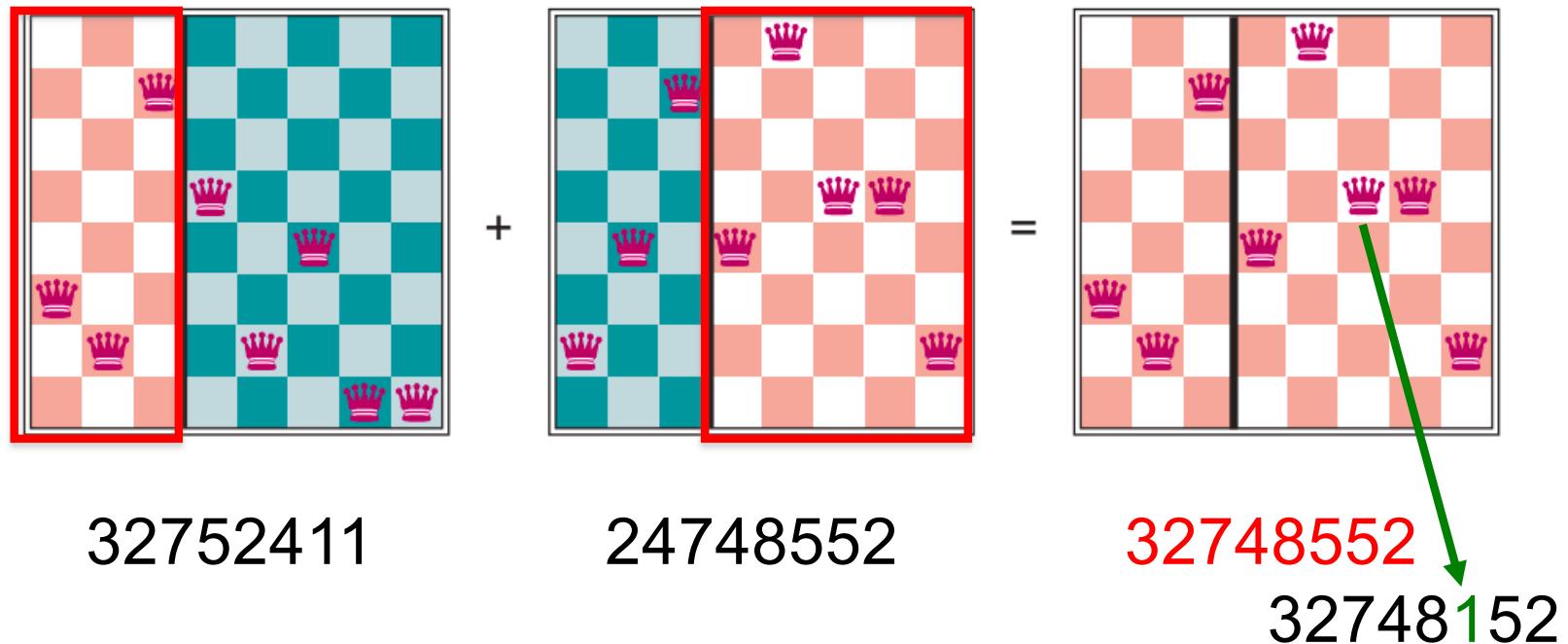
Algoritmos Genéticos



- O ponto de cruzamento é escolhido aleatoriamente
- São criados os filhos (d)
- Cada estado sofre mutações aleatórias (e)

Algoritmos Genéticos

exº cruzamento e mutação



Algoritmos Genéticos

- Há ainda muito trabalho a fazer de modo a perceber em que condições e com que parâmetros é que os algoritmos genéticos se comportam bem

Resumo

- 4.1 Procura local em ambientes de otimização
 - Hill-climbing
 - Simulated annealing
 - Local beam
 - Genetic algorithms
- 4.2 Procura em espaços contínuos
- 4.3 Procura com ações não determinísticas
- 4.4 Procura em ambientes parcialmente observáveis
- 4.5 Agentes de procura online e ambientes desconhecidos

Procura em Espaços Contínuos

- Espaço contínuo tem fator de ramificação infinito
 - Somente *first choice hill climbing* e *simulated annealing* são adequados

Espaços contínuos: exemplo

- Encontrar localização (x_i, y_i) de 3 novos aeroportos na Roménia
 - t.q. a soma das distâncias quadradas em linha reta de cada cidade $c \in C_i$ no mapa até ao aeroporto mais próximo (x_i, y_i) é minimizada

$$f(\mathbf{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^3 \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

Espaços contínuos: exemplo

$$f(\mathbf{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^3 \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

- Problema: C_i é conjunto dinâmico
- Solução: discretização?
 - Limitar (x_i, y_i) a pontos fixos numa grelha retangular com espaçamento de tamanho δ
 - Cada estado tem apenas $3 \times 4 = 12$ sucessores

Espaços contínuos: exemplo

$$f(\mathbf{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^3 \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

- Problema: C_i é conjunto dinâmico
- Solução: métodos de gradiente empírico
 - medem o progresso pela mudança no valor da função objetivo entre dois pontos próximos
 - gradiente da função objetivo (solução $\nabla f=0$)

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

– gradiente local $\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_1 - x_c)$

Espaços contínuos

- Limitações com máximos locais, cumes e planaltos
- Otimização com restrições: restrições *hard* têm de ser satisfeitas (por exº aeroporto localizado em terreno seco)
- Programação linear
- Otimização convexa

Resumo

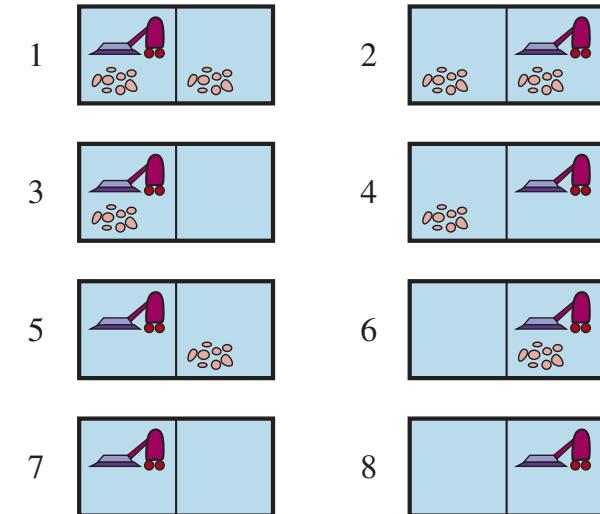
- 4.1 Procura local em ambientes de optimização
 - Hill-climbing
 - Simulated annealing
 - Local beam
 - Genetic algorithms
- 4.2 Procura em espaços contínuos
- 4.3 Procura com ações não determinísticas
- 4.4 Procura em ambientes parcialmente observáveis
- 4.5 Agentes de procura online e ambientes desconhecidos

Ações não determinísticas

- Agente **desconhece para que estado transita** depois de realizar uma ação
 - Agente acredita que vai estar num estado correspondente a um subconjunto de estados físicos (crenças)
 - Solução é um plano condicional
- Árvore de procura com **nós diferenciados**
 - Nós OR: decisões do agente
 - Nós AND: output do ambiente

Ações não determinísticas: ex⁰

- Estados possíveis
 - 7 e 8 objetivos



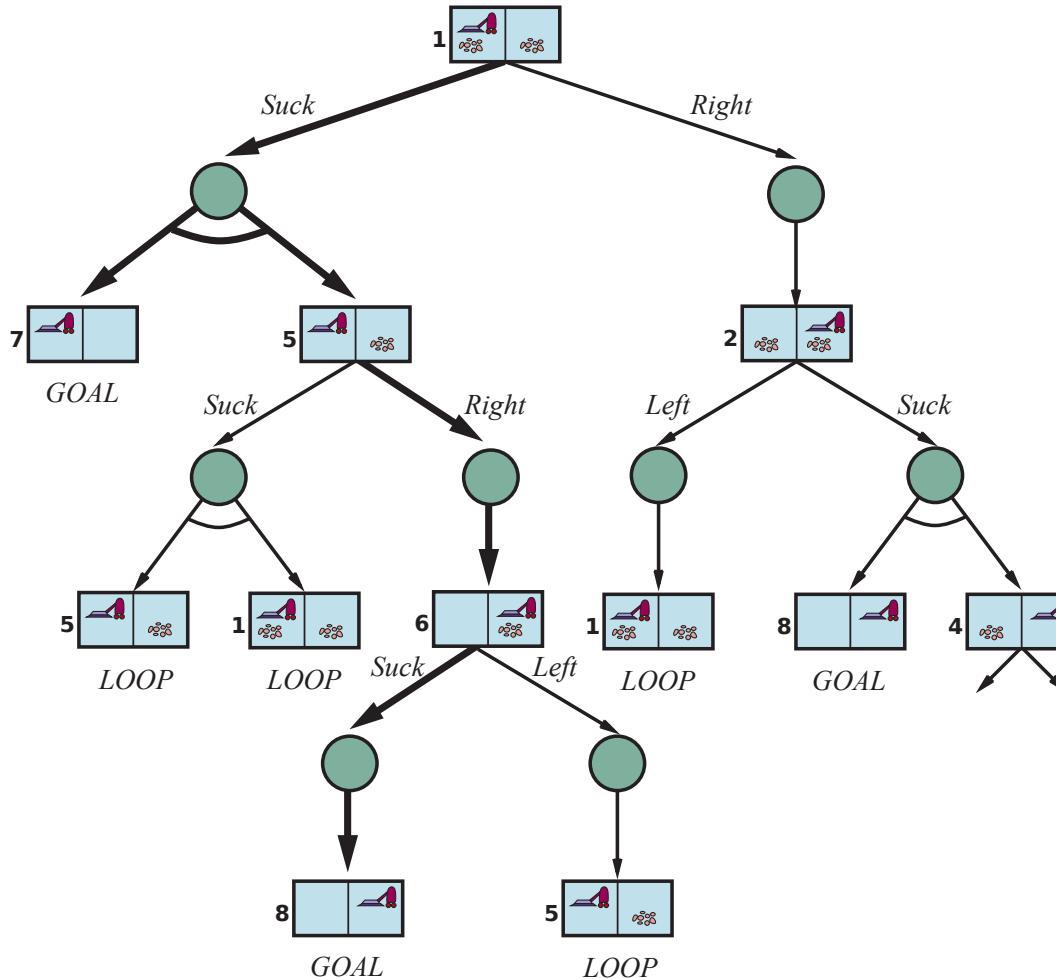
- Resultado de ação errática

$$\text{RESULTS}(1, \text{Suck}) = \{5, 7\}$$

- Plano condicional para estado inicial 1

$[\text{Suck}, \text{if } \text{State} = 5 \text{ then } [\text{Right}, \text{Suck}] \text{ else } []]$.

Árvore AND-OR: exemplo



[Suck, if State = 5 then [Right, Suck] else []].

Árvore AND-OR

- Solução = sub-árvore
 - Tem um objetivo em cada folha
 - Especifica uma ação em cada nó OR
 - Inclui cada ramo de cada nó AND

Procura AND-OR

function AND-OR-SEARCH(*problem*) **returns** a conditional plan, or *failure*
return OR-SEARCH(*problem*, *problem.INITIAL*, [])

function OR-SEARCH(*problem*, *state*, *path*) **returns** a conditional plan, or *failure*
if *problem.IS-GOAL(state)* **then return** the empty plan
if IS-CYCLE(*state*, *path*) **then return** *failure*
for each *action* **in** *problem.ACTIONS(state)* **do**
 plan \leftarrow AND-SEARCH(*problem*, RESULTS(*state*, *action*), [*state*] + [*path*])
 if *plan* \neq *failure* **then return** [*action*] + [*plan*]
return *failure*

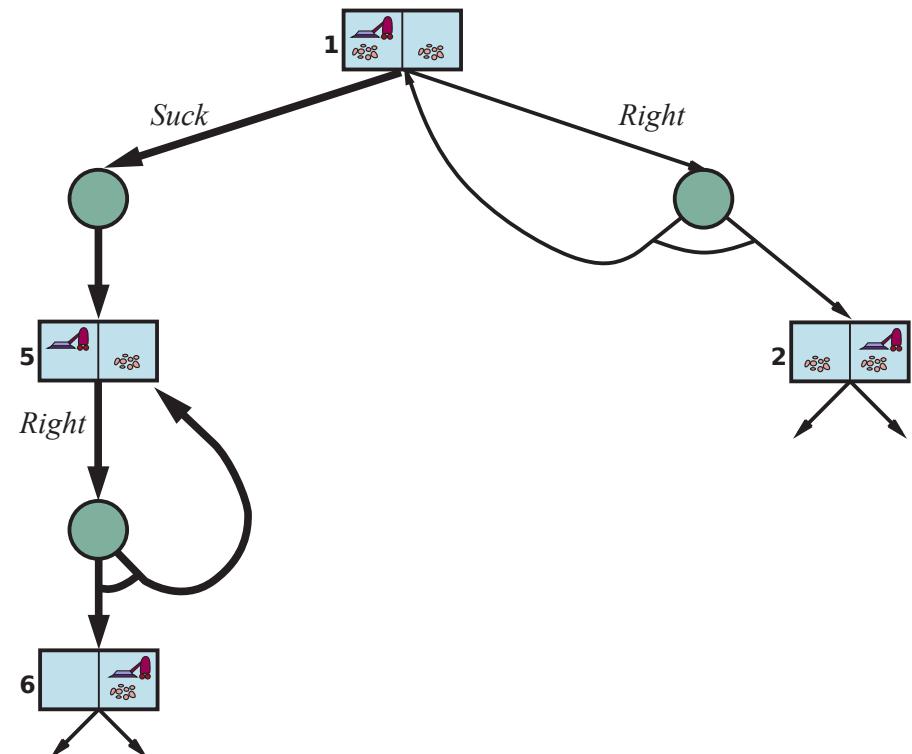
function AND-SEARCH(*problem*, *states*, *path*) **returns** a conditional plan, or *failure*
for each *s_i* **in** *states* **do**
 plan_i \leftarrow OR-SEARCH(*problem*, *s_i*, *path*)
 if *plan_i* = *failure* **then return** *failure*
return [**if** *s₁* **then** *plan₁* **else if** *s₂* **then** *plan₂* **else** ... **if** *s_{n-1}* **then** *plan_{n-1}* **else** *plan_n*]

Procura AND-OR: ciclos

- Identificados ao longo do mesmo caminho
 - **Return failure**
- Podem surgir estados repetidos em caminhos diferentes
 - Não são eliminados

Solução cíclica: try, try again

- Todas as soluções são cíclicas
- Solução cíclica consiste em tentar e tentar... plano cíclico!
[*Suck, while State = 5 do Right, Suck*]
- Plano cíclico é solução?
Depende da causa de não determinismo...
 - Se ação é aleatória e independente, ok!
 - Caso contrário, ko



Resumo

- 4.1 Procura local em ambientes de otimização
 - Hill-climbing
 - Simulated annealing
 - Local beam
 - Genetic algorithms
- 4.2 Procura em espaços contínuos
- 4.3 Procura com ações não determinísticas
- 4.4 Procura em ambientes parcialmente observáveis
 - Procura sem observações
 - Procura em ambientes parcialmente observáveis
 - Resolução de problemas parcialmente observáveis
 - Um agente para ambientes parcialmente observáveis
- 4.5 Agentes de procura online e ambientes desconhecidos

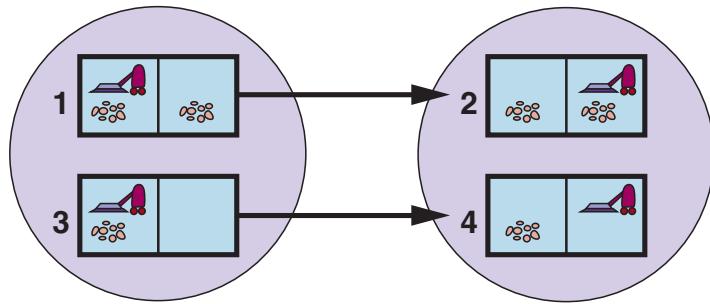
Procura sem observações

- Perceções sem qualquer informação!
- Problema *sensorless / conformant*
 - Vantagem de não depender do correto funcionamento de sensores!
 - Analogia com antibióticos de largo espetro

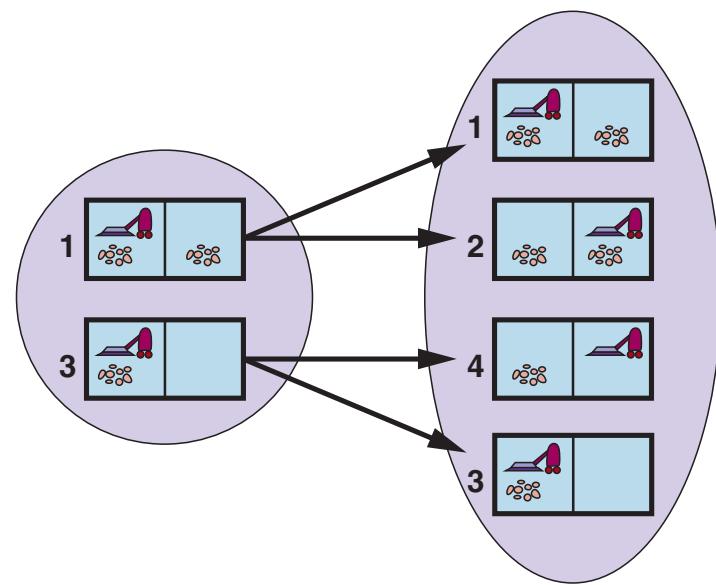
Caraterização do problema

- Estados: espaço de crenças contém todos os subconjuntos de estados físicos de P (problema original)
- Estado inicial: crença com todos os estados
- Ações para crença b $\text{ACTIONS}(b) = \bigcup_{s \in b} \text{ACTIONS}_P(s)$
- Modelo de transição
 - Com determinismo $b' = \text{RESULT}(b, a) = \{s' : s' = \text{RESULT}_P(s, a) \text{ and } s \in b\}$
 - Sem determinismo $b' = \text{RESULT}(b, a) = \{s' : s' \in \text{RESULTS}_P(s, a) \text{ and } s \in b\}$ $= \bigcup_{s \in b} \text{RESULTS}_P(s, a),$
- Teste objetivo
 - *Possivelmente* alcançado se estado de crença satisfaz teste objetivo
 - *Necessariamente* alcançado se todos os estados de crença satisfazem objetivo
- Custo de uma ação: o mesmo para todos os estados

Modelo de transição: exemplo

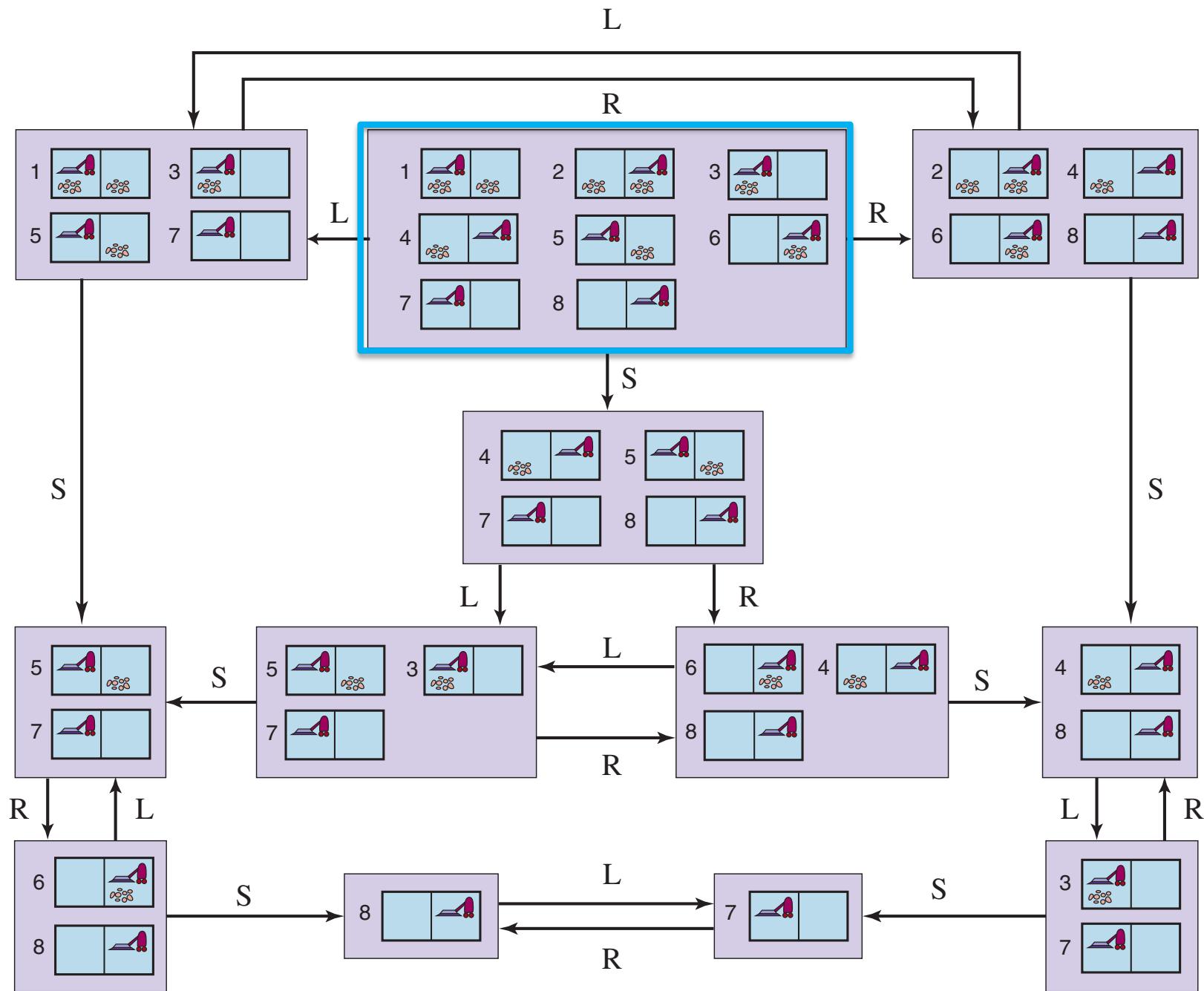


(a)



(b)

Figure 4.13 (a) Predicting the next belief state for the sensorless vacuum world with the deterministic action, *Right*. (b) Prediction for the same belief state and action in the slippery version of the sensorless vacuum world.



Como procurar?

- Usando os algoritmos de procura tradicionais
- Identificação de estados repetidos é estendida...
 - Se uma crença é *superset* de outra podemos descartá-la
 - Se uma crença é *subset* de outra que pertence a uma solução, então a crença em causa também pertence a uma solução
- Otimização: procura incremental com estado inicial para cada estado da crença inicial

Ambientes parcialmente observáveis

- 3 etapas no modelo de transição
 - Predição: calcula crenças resultantes de ação
$$\hat{b} = \text{RESULT}(b, a)$$
 - Percepções possíveis: calcula conjunto de percepções que podem ser observadas a partir de crença resultante de ação
$$\text{POSSIBLE-PERCEPTS}(\hat{b}) = \{o : o = \text{PERCEP}(s) \text{ and } s \in \hat{b}\}$$
 - Atualiza: calcula crença resultante de cada percepção possível
$$b_o = \text{UPDATE}(\hat{b}, o) = \{s : o = \text{PERCEP}(s) \text{ and } s \in \hat{b}\}$$
- Junção $\text{RESULTS}(b, a) = \{b_o : b_o = \text{UPDATE}(\text{PREDICT}(b, a), o) \text{ and } o \in \text{POSSIBLE-PERCEPTS}(\text{PREDICT}(b, a))\}$

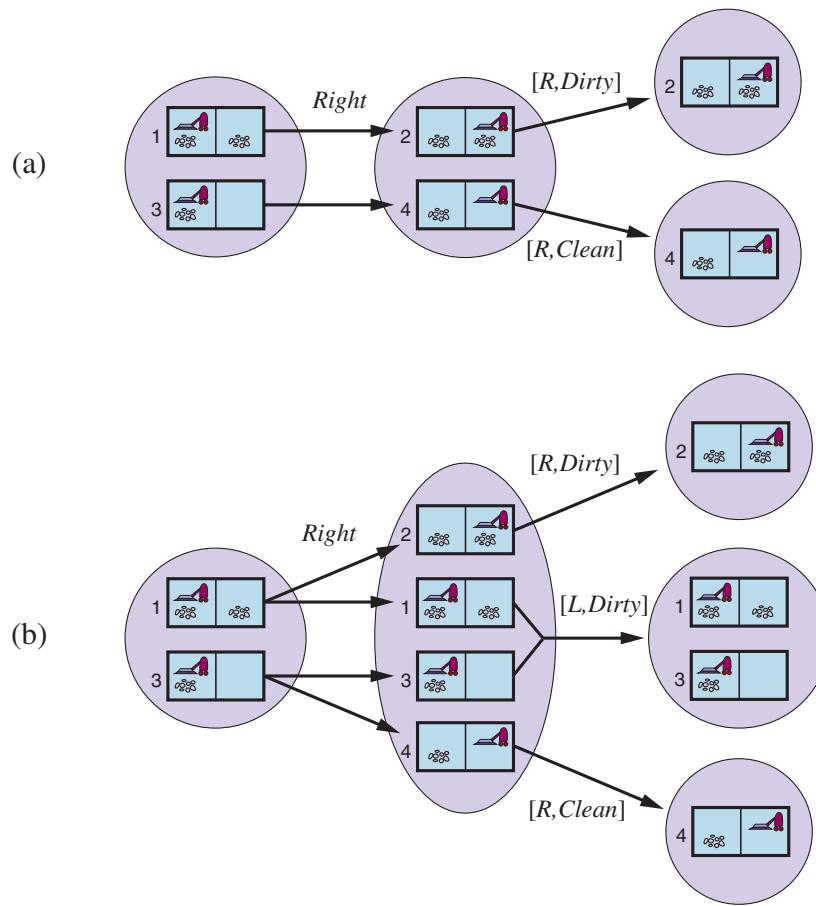
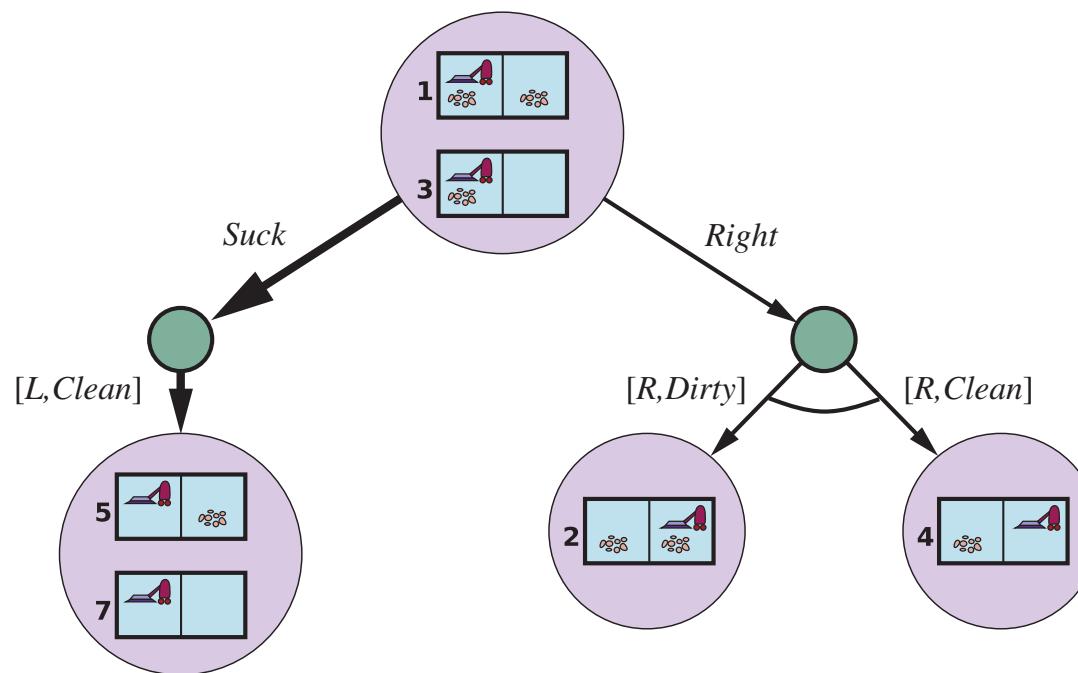


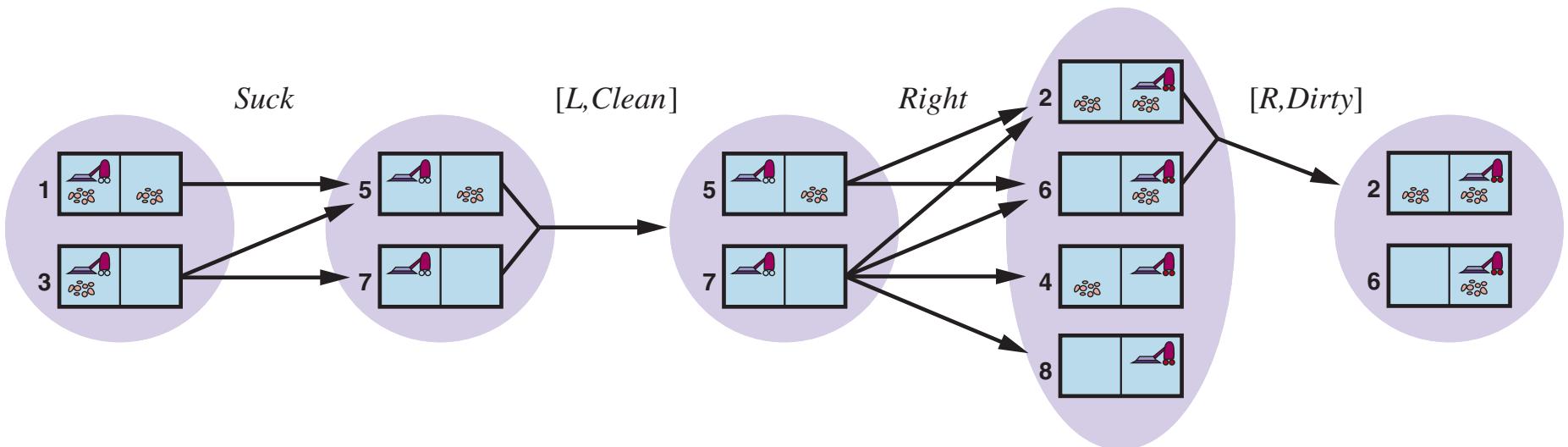
Figure 4.15 Two examples of transitions in local-sensing vacuum worlds. (a) In the deterministic world, *Right* is applied in the initial belief state, resulting in a new predicted belief state with two possible physical states; for those states, the possible percepts are [*R*, *Dirty*] and [*R*, *Clean*], leading to two belief states, each of which is a singleton. (b) In the slippery world, *Right* is applied in the initial belief state, giving a new belief state with four physical states; for those states, the possible percepts are [*L*, *Dirty*], [*R*, *Dirty*], and [*R*, *Clean*], leading to three belief states as shown.

Procura AND-OR (incompleta)



Solução: plano condicional **[Suck, Right, if Rstate = {6} then Suck else []]**

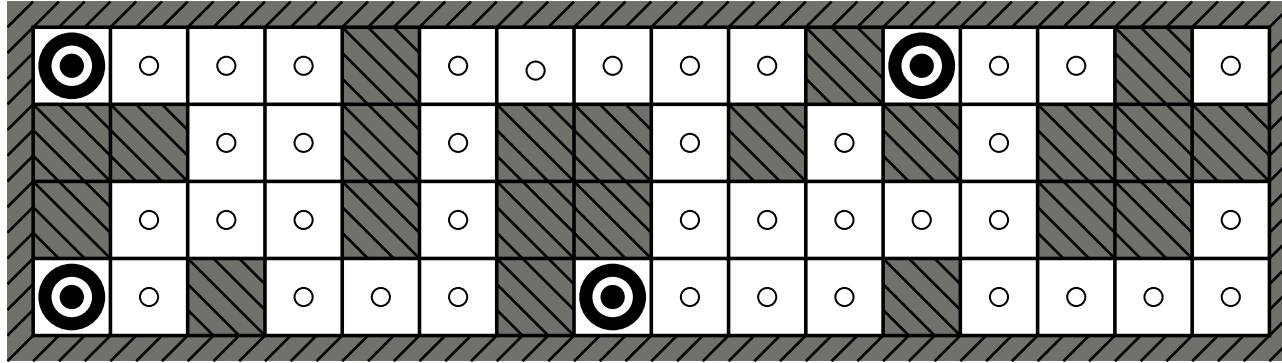
Aspirador em jardim de infância



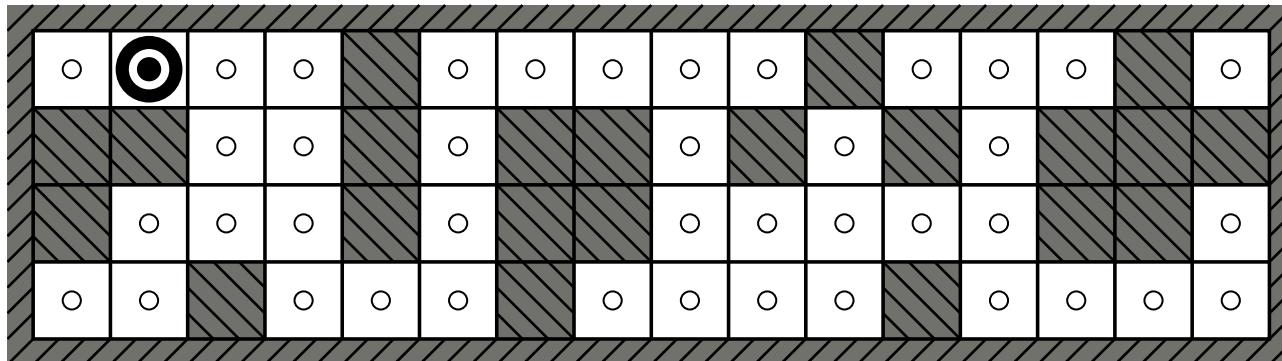
Solução: *[Suck, Right, if Rstate = {6} then Suck else []]*

Robot com tarefa localização

Onde está
robot, dado
mapa e
perceções?



(a) Possible locations of robot after $E_1 = 1011$



(b) Possible locations of robot after $E_1 = 1011, E_2 = 1010$

Dígitos de E
denotam
barreiras a
norte, este,
sul e oeste

Figure 4.18 Possible positions of the robot, \odot , (a) after one observation, $E_1 = 1011$, and (b) after moving one square and making a second observation, $E_2 = 1010$. When sensors are noiseless and the transition model is accurate, there is only one possible location for the robot consistent with this sequence of two observations.

Resumo

- 4.1 Procura local em ambientes de otimização
 - Hill-climbing
 - Simulated annealing
 - Local beam
 - Genetic algorithms
- 4.2 Procura em espaços contínuos
- 4.3 Procura com ações não determinísticas
- 4.4 Procura em ambientes parcialmente observáveis
- 4.5 Agentes de procura online e ambientes desconhecidos
 - Problemas de procura online
 - Agentes de procura online
 - Procura local online
 - Aprendizagem em procura online

Procura *online*

- Procura *offline*: primeiro procura e só depois aplica ações
- Procura *online*: intervala procura e ação
 - Executa ação, observa ambiente, calcula próxima ação
 - Adequada para ambiente dinâmicos ou semi-dinâmicos
 - Exº problemas de mapeamento – exploração espacial

Procura online

- Agente sabe apenas...
 - Actions(s): ações possíveis em estado s
 - $c(s,a,s')$: custo de aplicar ação a no estado s para alcançar s'
 - Is-Goal(s): teste objetivo
- Agente não pode determinar Result(s,a) exceto se estiver em s e fizer a
 - Ações são determinísticas mas desconhecidas
- Vulnerabilidade a *dead-ends*: estados a partir dos quais não se alcança objetivo
- Caracterização do desempenho em função de
 - Espaço de estados
 - Profundidade da solução

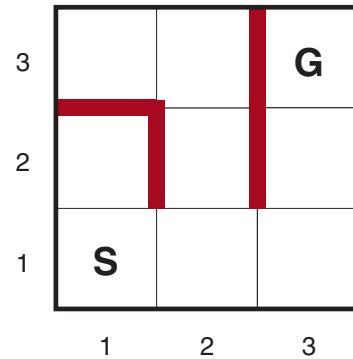


Figure 4.19 A simple maze problem. The agent starts at S and must reach G but knows nothing of the environment.

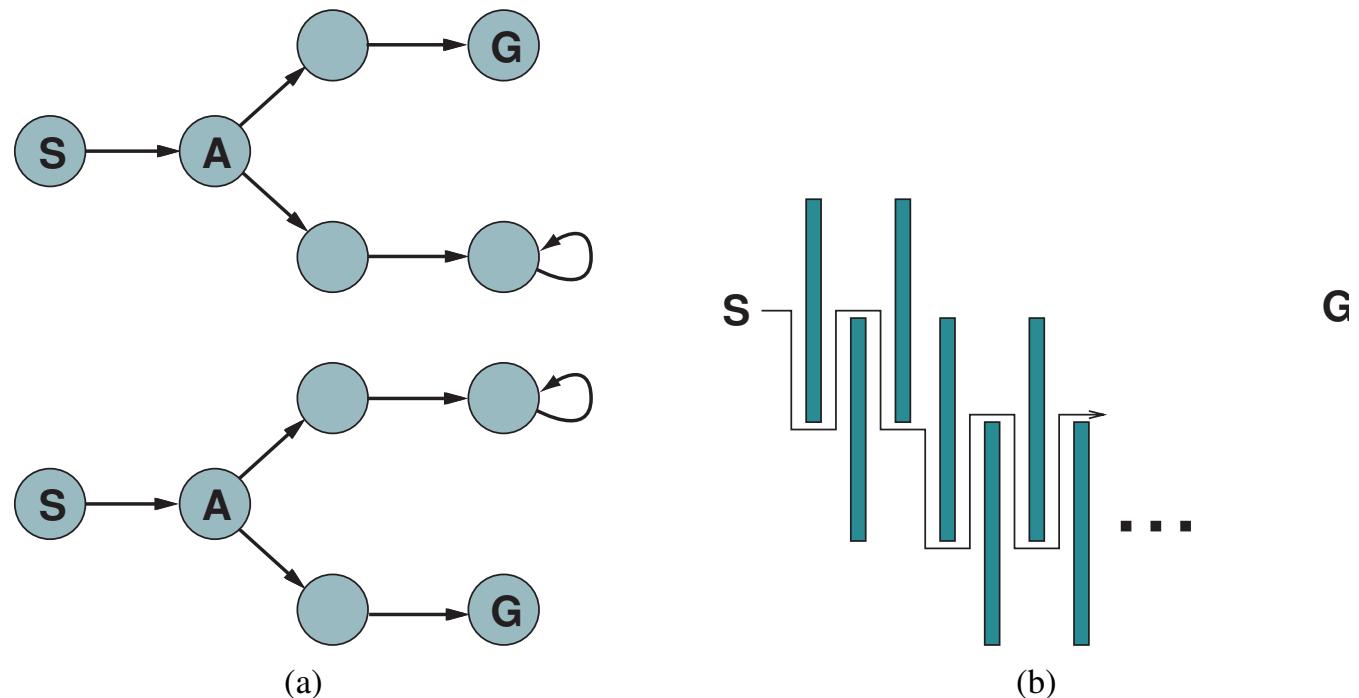


Figure 4.20 (a) Two state spaces that might lead an online search agent into a dead end. Any given agent will fail in at least one of these spaces. (b) A two-dimensional environment that can cause an online search agent to follow an arbitrarily inefficient route to the goal. Whichever choice the agent makes, the adversary blocks that route with another long, thin wall, so that the path followed is much longer than the best possible path.

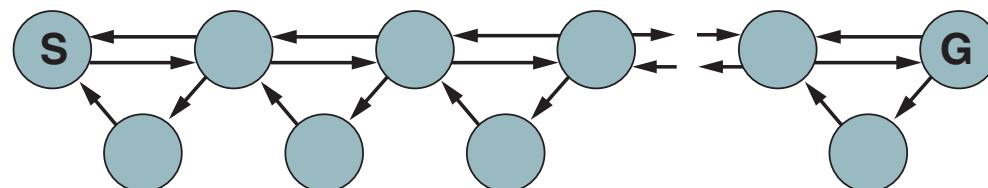
Agente *online* DFS

```
function ONLINE-DFS-AGENT(problem,  $s'$ ) returns an action
     $s, a$ , the previous state and action, initially null
    result, a table mapping  $(s, a)$  to  $s'$ , initially empty
    untried, a table mapping  $s$  to a list of untried actions
    unbacktracked, a table mapping  $s$  to a list of states never backtracked to

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

Procura local *online*

- Hill-climbing
 - Vantagem: um só estado de cada vez
 - Desvantagem: pára em máximos locais
- Random walk
 - Seleciona *aleatoriamente* ação a partir de estado
 - *Eventualmente* encontra solução ou explora todo o espaço se espaço for finito
 - Exemplo que requer número exponencial de iterações



Procura local online

- Melhor: usar hill-climbing com memória 😊
 - Guardar a melhor estimativa do custo para cada estado $H(s)$
 - $H(s)$ atualizado com custo real

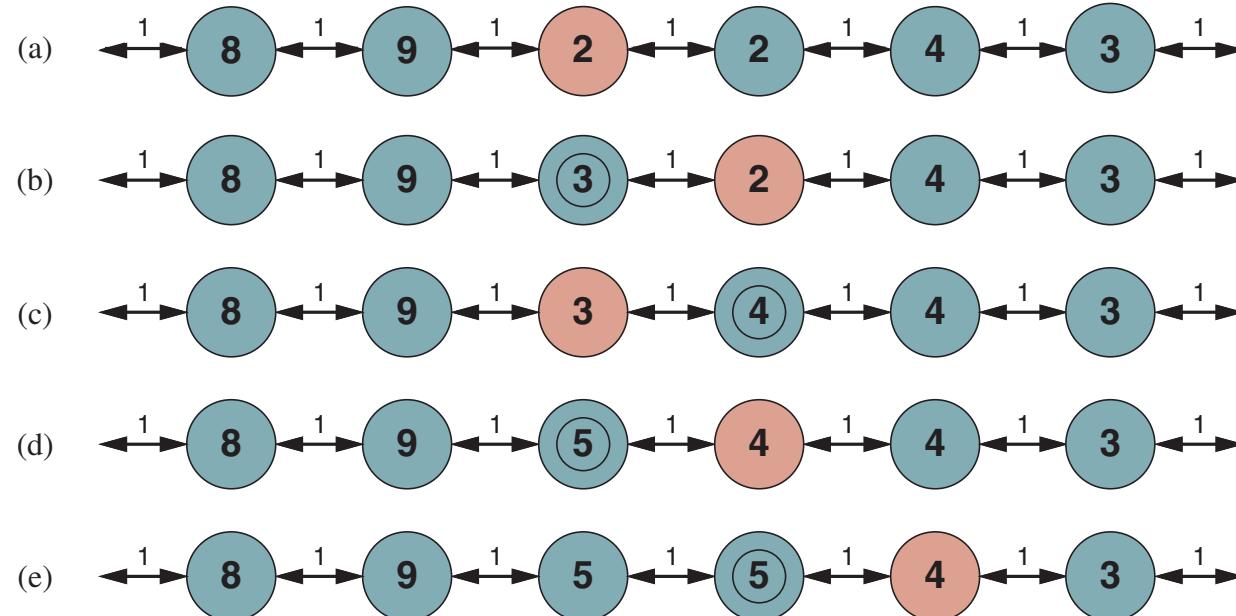


Figure 4.23 Five iterations of LRTA* on a one-dimensional state space. Each state is labeled with $H(s)$, the current cost estimate to reach a goal, and every link has an action cost of 1. The red state marks the location of the agent, and the updated cost estimates at each iteration have a double circle.

Learning real-time A* (LRTA*)

```
function LRTA*-AGENT(problem,  $s'$ ,  $h$ ) returns an action
     $s, a$ , the previous state and action, initially null
     $result$ , a table mapping  $(s, a)$  to  $s'$ , initially empty
     $H$ , a table mapping  $s$  to a cost estimate, initially empty

    if IS-GOAL( $s'$ ) then return stop
    if  $s'$  is a new state (not in  $H$ ) then  $H[s'] \leftarrow h(s')$ 
    if  $s$  is not null then
         $result[s, a] \leftarrow s'$ 
         $H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)} \text{LRTA}^*-\text{COST}(\text{problem}, s, b, result[s, b], H)$ 
     $a \leftarrow \operatorname{argmin}_{b \in \text{ACTIONS}(s)} \text{LRTA}^*-\text{COST}(\text{problem}, s', b, result[s', b], H)$ 
     $s \leftarrow s'$ 
    return  $a$ 
```

```
function LRTA*-COST(problem,  $s, a, s', H$ ) returns a cost estimate
    if  $s'$  is undefined then return  $h(s)$ 
    else return problem.ACTION-COST( $s, a, s'$ ) +  $H[s']$ 
```