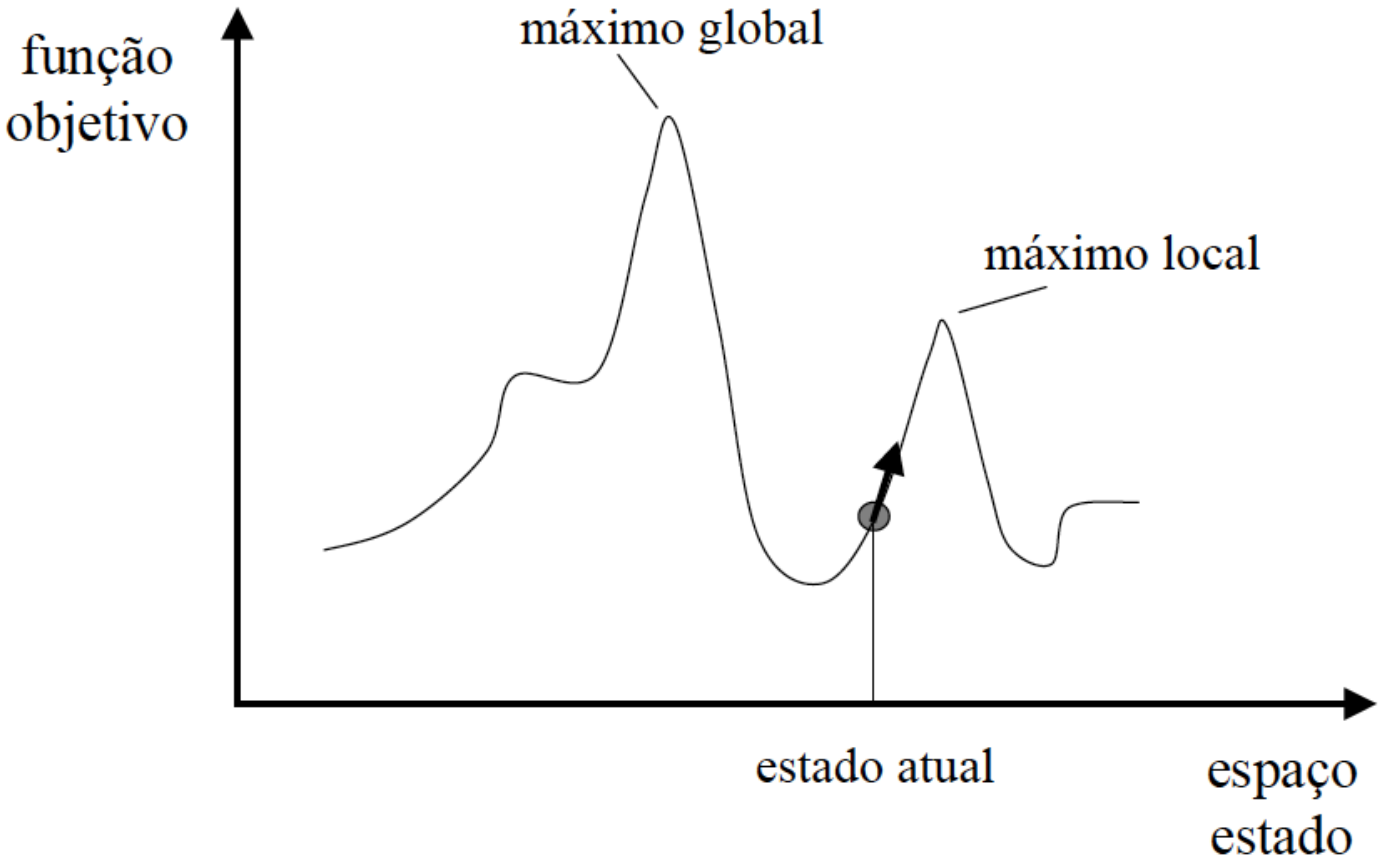
Engenharia de Computação
Inteligência Artificial
04 - Estruturas e Estratégias de Busca
Busca em Ambientes Complexos

Busca Local

- estado é o que interessa, não caminho
- busca inicia com um nó
- move para vizinhos do nó
- necessitam de pouca memória
- operam em espaços contínuos e discretos
- completo: se encontrar uma meta (se existir)
- ótimo: se encontrar um ótimo global

Topologia de espaço de estados



Algoritmo do gradiente (hill-climbing)

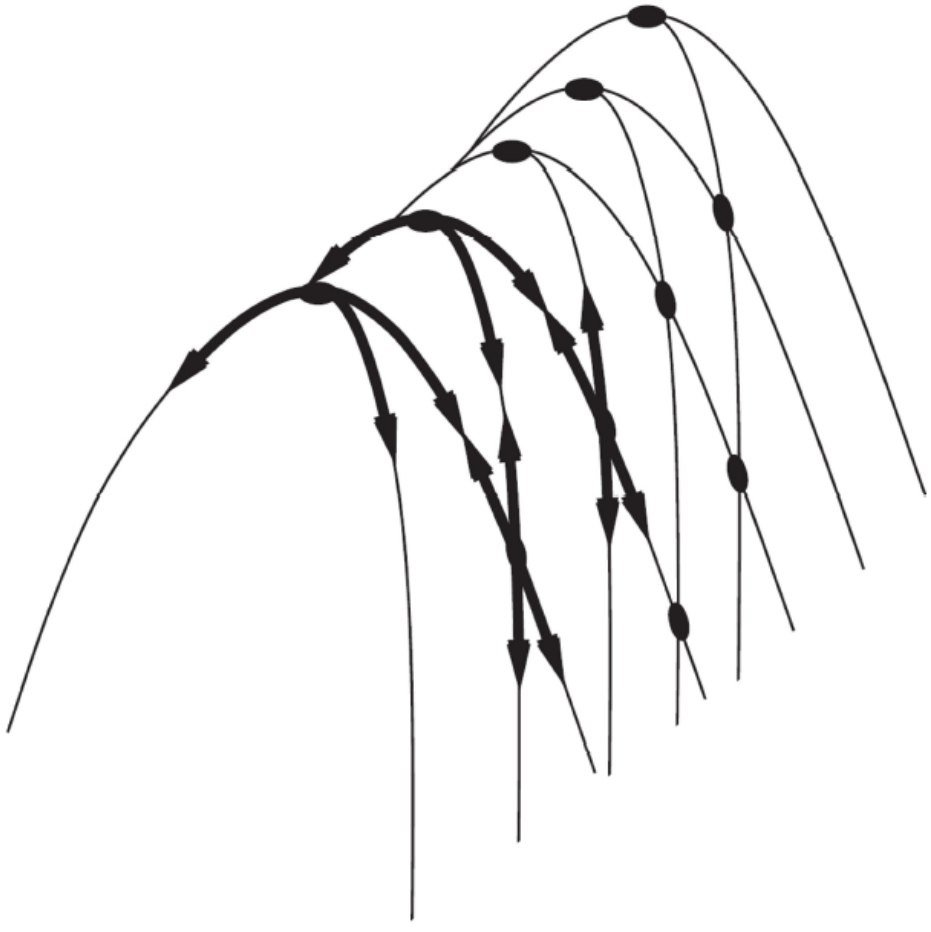
function HILL_CLIMBING (*problem*) **returns** a state that is local maximum

current ← MAKE-NODE (*problem*.INITIAL-STATE)
loop do
neighbor ← a highest-valued successor of *current*
if *neighbor*.VALUE < *current*.VALUE **then return** *current*.STATE
current ← *neighbor*

Características

- não mantém uma árvore de busca
- estrutura dados nó: estado e valor função objetivo
- move para vizinhos imediatos do nó
- busca local gulosa
- problemas: ótimos locais, platôs, ridges (cordilheiras)
- incompleto (ótimos locais)

Ótimos locais



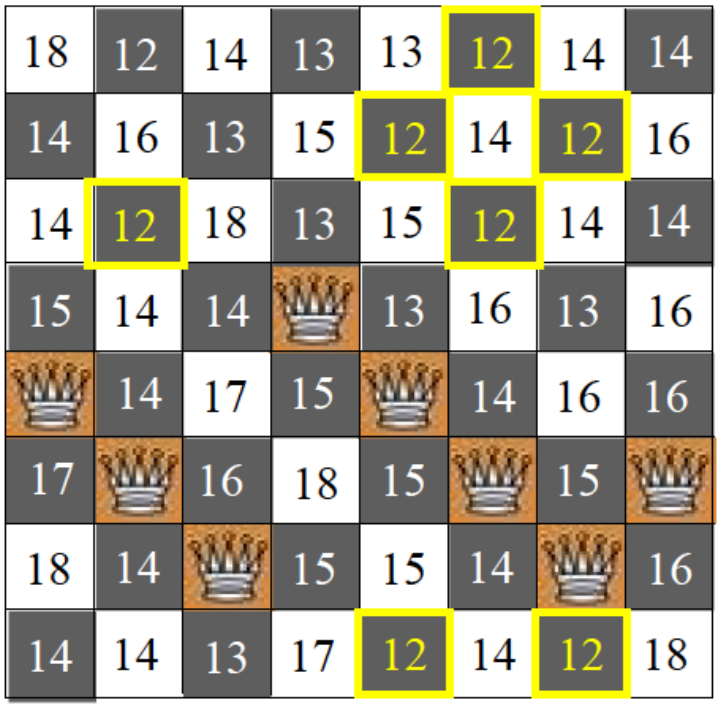
Propostas de melhorias

- Como melhorar o algoritmo hill-climbing?

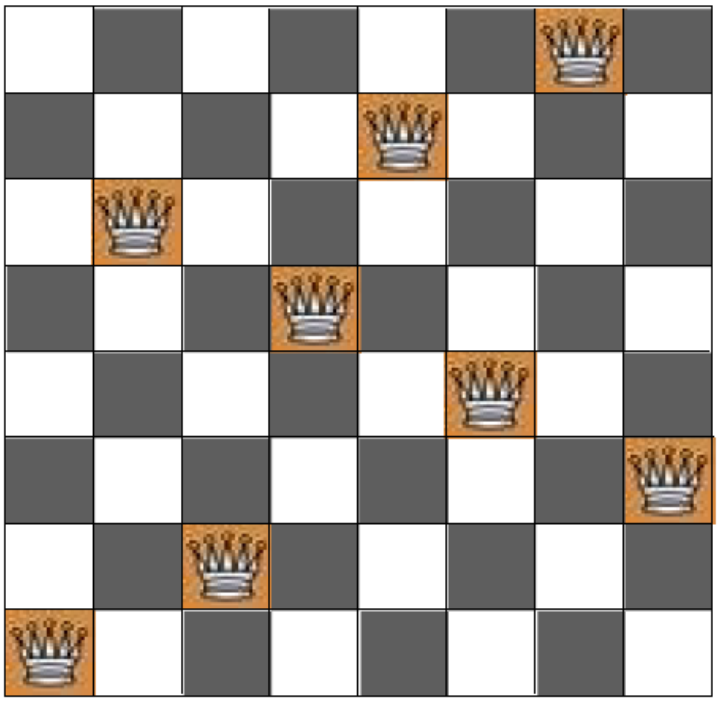
Propostas de melhorias

- Como melhorar o algoritmo hill-climbing?
  - Passos laterais
  - Subida de encosta estocástica
  - Subida de encosta estocástica pela primeira escolha
  - Subida de encosta com reinício aleatório

Exemplo: problema das 8 rainhas



$h = 17$



$h = 1$ (mínimo local)

$h =$ número de pares de rainhas que se atacam (direta ou indiretamente)
5 passos $h = 17$ para $h = 1$

8 rainhas com busca local

- estado inicial: gerado aleatoriamente
- 86% falha depois de 4 passos (média)
- 14% acha solução depois de 3 passos (média)
- espaço estado: $8^8 \approx 17$ milhões estados!
- busca em platô: limite no número de iterações (passos laterais)
  - 94% resolvidos com 21 passos (média)
  - 6% de falhas com 64 passos (média)
- reinicializações: problema com 3 milhões de rainhas em 3 min.!

Algoritmo simulated annealing

function SIMULATED_ANNEALING (*problem*, *schedule*) **returns** a solution state

inputs: *problem*, a problem
schedule, a mapping from time to "temperature"

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

Características

- probabilidade diminui exponencialmente se a qualidade piora
- probabilidade diminui quando a temperatura diminui
- schedule diminui probabilidade suavemente
- ótimo global com probabilidade $\rightarrow 1$

Local beam search - Busca em feixe local

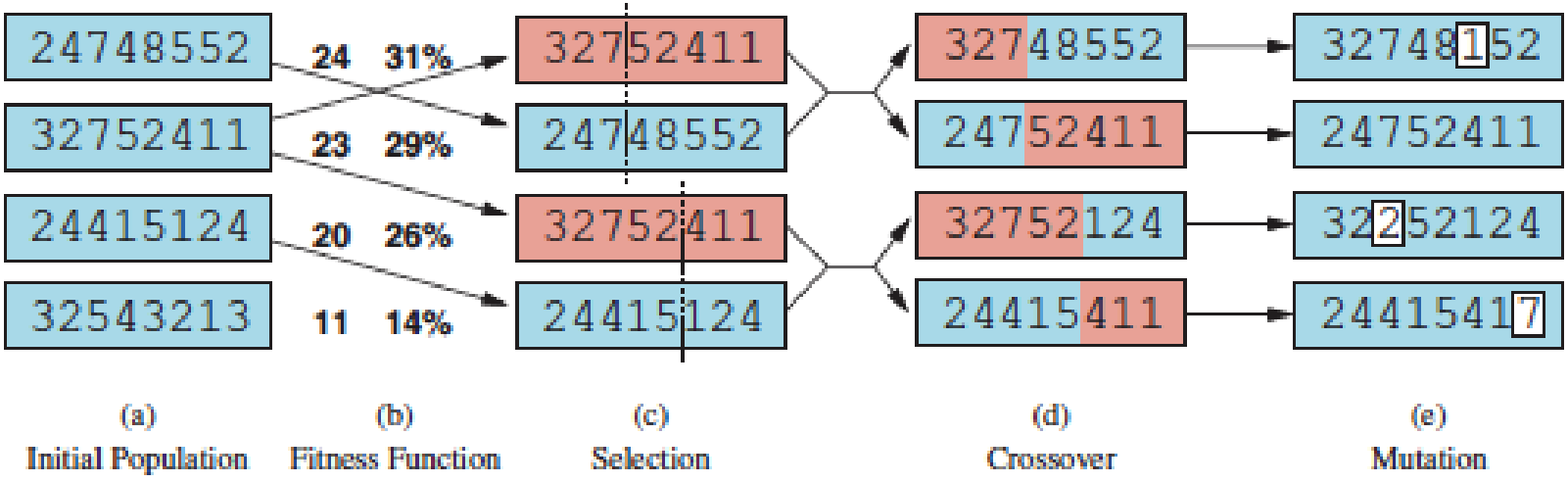
- mantém k nós (estados) ao invés de um único
- inicializado com k nós, gerados aleatoriamente
- gera todos os sucessores dos k nós
- se encontra meta: para
- senão escolhe os k melhores e continua
- difere do hill-climbing com reinicializações
- problema: diversidade das k soluções
  - aliviado escolhendo k nós aleatoriamente

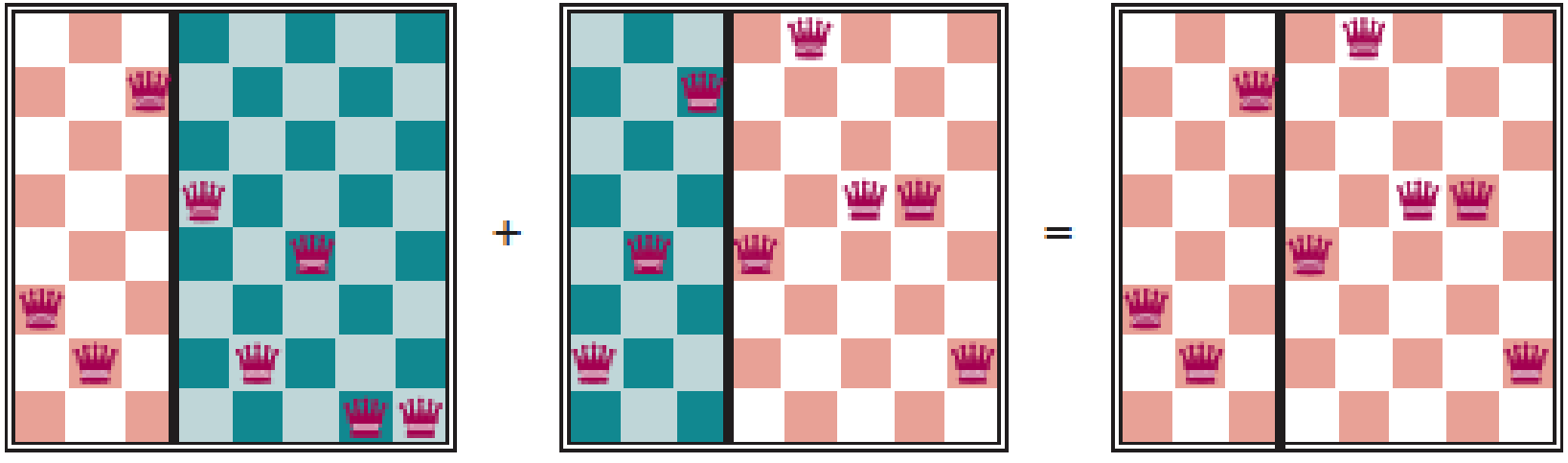
Algoritmos Genéticos

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← 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 ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

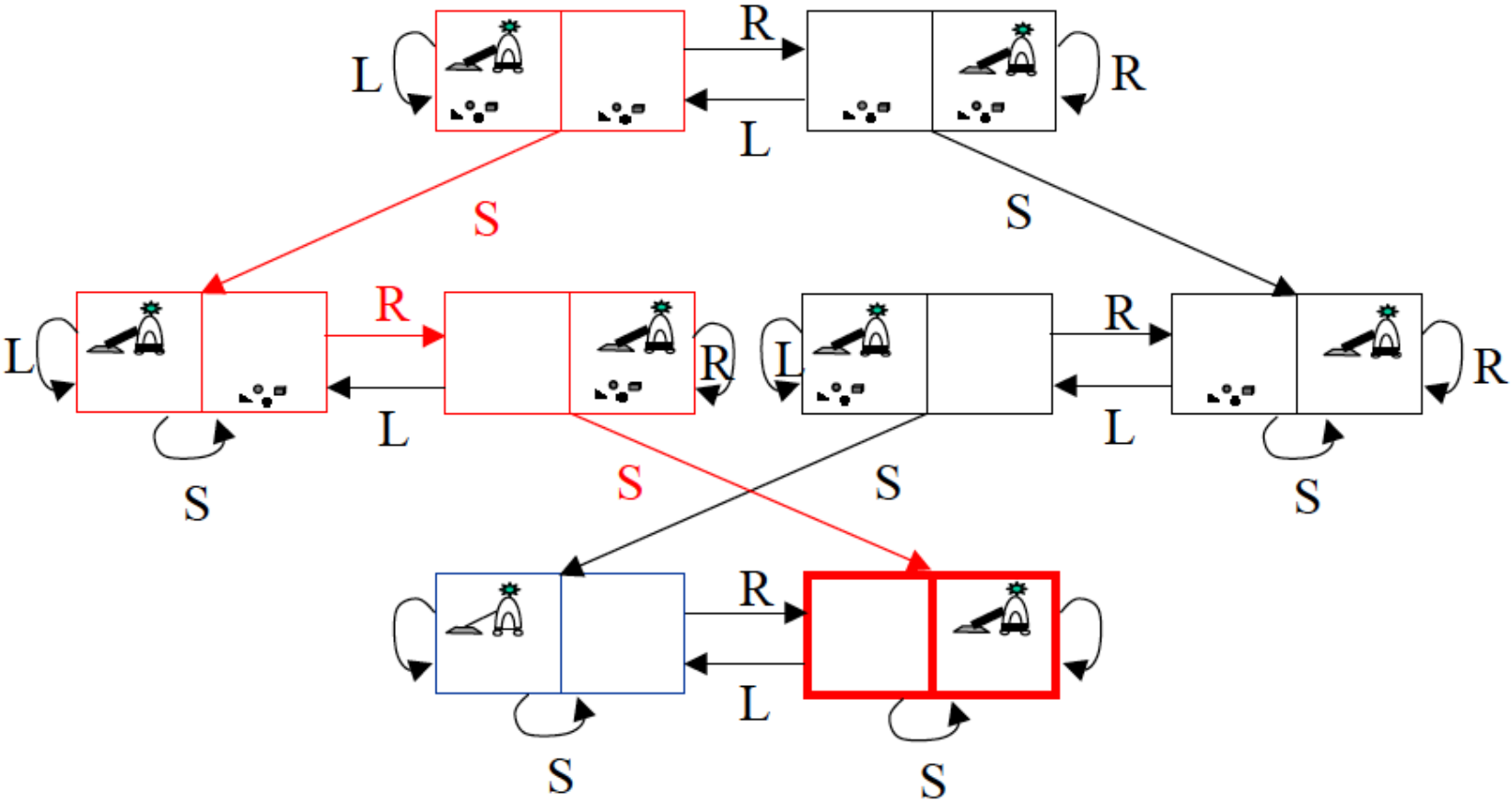
Algoritmos Genéticos





Ambientes de Busca

- Se
  - ambiente: observável, determinístico, completamente conhecido
  - agente conhece o estado onde está
  - efeito das ações são conhecidos
- Então
  - solução: sequência de ações
  - *percepts* são irrelevantes


L
R
L
R
S
S
L
R
L
R
S
L
R
L
R
S
S
S
L
R
L
S
S

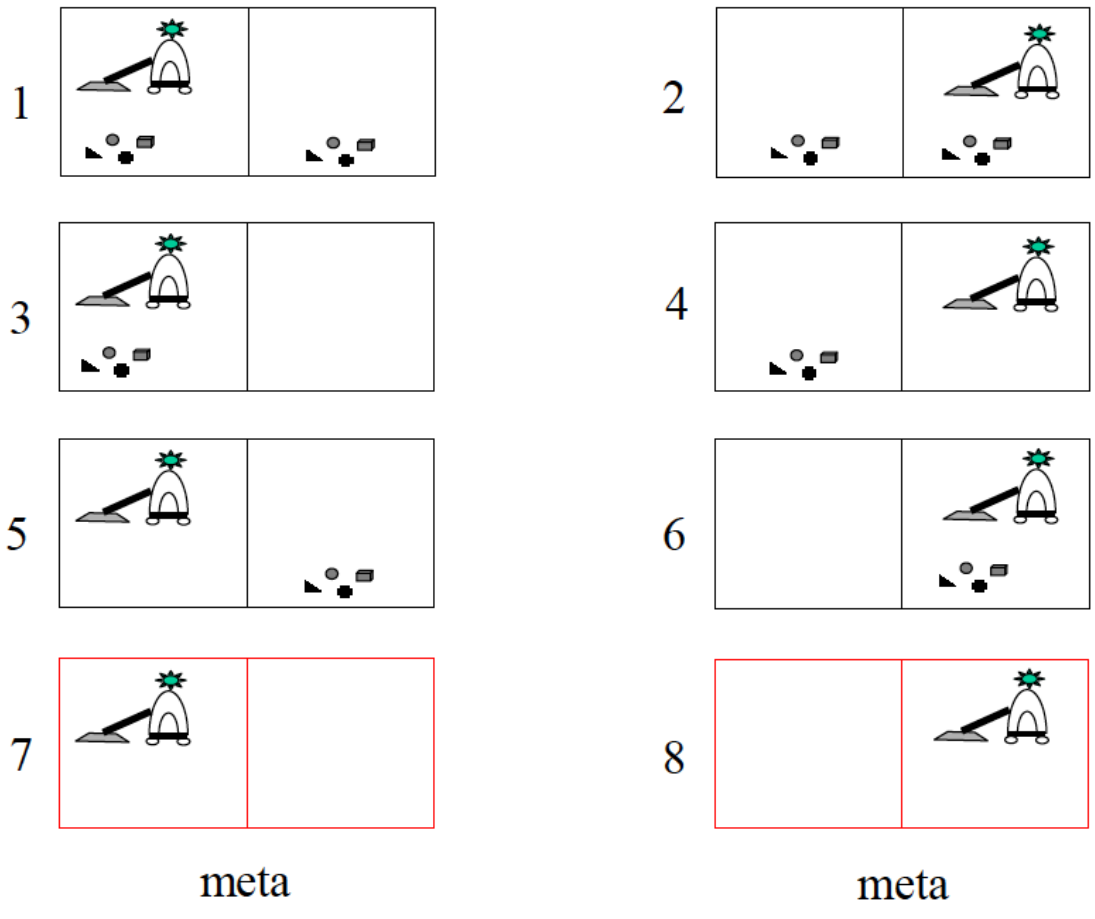
Busca com ações não determinísticas

- Ambiente
  - parcialmente observável
  - não determinístico
- Importância dos percepts
  - ajuda a focalizar a busca
  - resultados das ações
- Percepções futuras são desconhecidas
- Solução do problema: estratégia (plano de contingência)

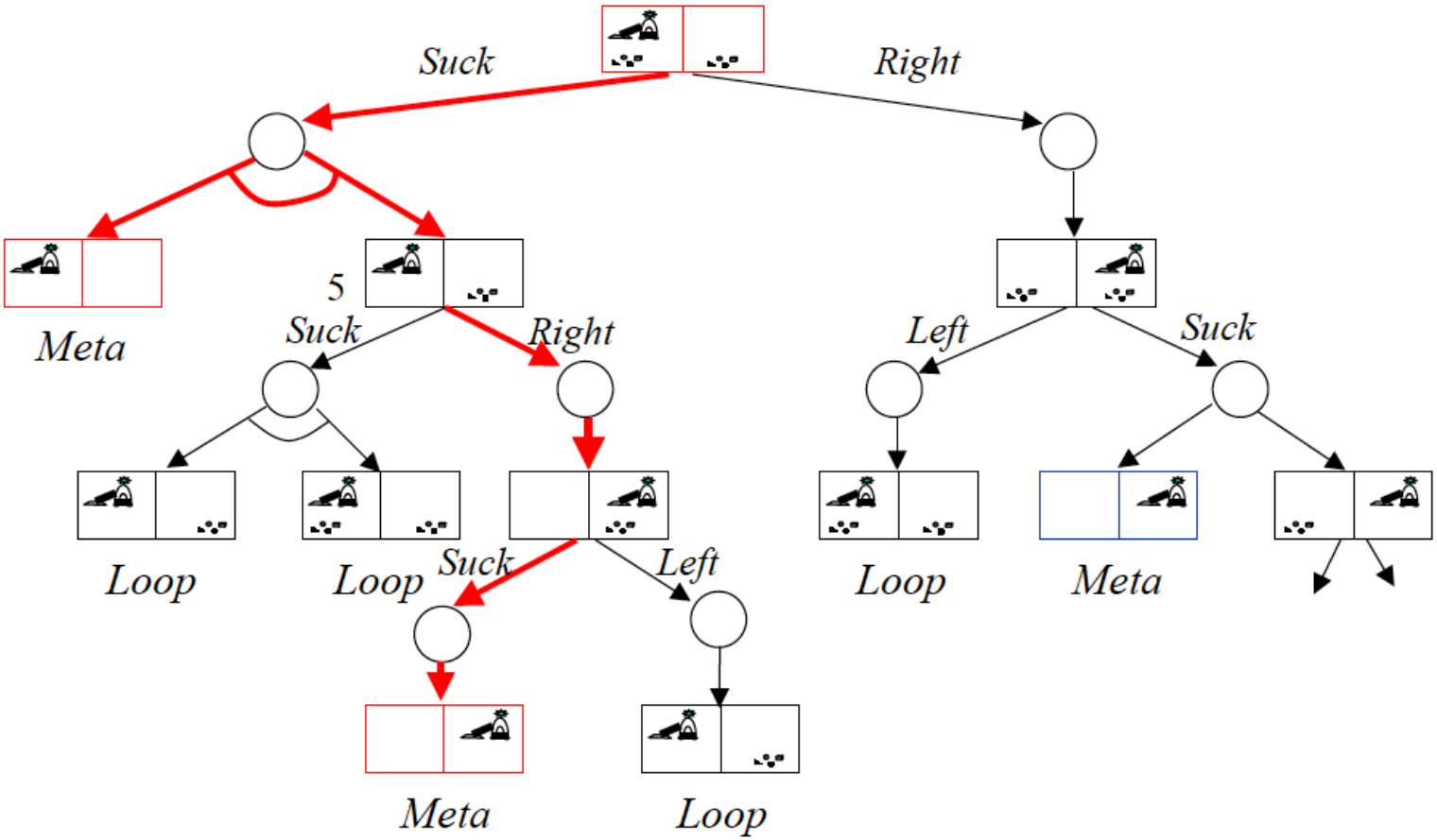
Exemplo: agente errático

- Ação Suck
  - posição com sujeira: limpa posição e eventualmente a adjacente
  - posição limpa: ação eventualmente deposita sujeira
- Modelo de transição
  - função RESULTS (ao invés de RESULT)
  - retorna um conjunto de estados
  - exemplo: $\{1\}Suck \rightarrow \{5, 7\}$
- Solução
  - plano de contingência: (estratégia)
  - [*Suck*, *if State* $=$ 5 *then* [*Right*, *Suck*] *else* [ ]]

Espaço de estados do problema



Árvore de busca And-Or



Solução em um grafo AND-OR: sub-árvore que

- tem uma meta em cada folha
- especifica uma ação em cada nó OR
- inclui todos descendentes em cada nó AND
- solução: é um plano do tipo

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

Algoritmo de busca em grafos AND-OR

function AND_OR_GRAPH_SEARCH (*problem*) **returns** a conditional plan, or failure
  OR_SEARCH (*problem*.INITIAL-STATE, *problem*, [])

function OR_SEARCH (*state*, *problem*, *path*) **returns** a conditional plan, or failure
  if *problem*.GOAL-TEST(*state*) **then return** the empty plan
  if *state* is on *path* **then return** *failure*
  for each *action* **in** *problem*.ACTIONS(*state*) **do**
    plan ← AND_SEARCH(RESULTS(*state*, *action*), *problem*, [*state*|*path*])
    if *plan* ≠ *failure* **then return** [*action*|*plan*]
  return *failure*

function AND_SEARCH (*states*, *problem*, *path*) **returns** a conditional plan, or failure
  for each s_i in *states* **do**
  $plan_i$ ← OR_SEARCH(s_i, *problem*, *path*)
  if $plan_i$ = *failure* **then return** *failure*
  return [**if** s_1 then $plan_1$ **else if** s_2 **then** $plan_2$ **else**....**if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$]

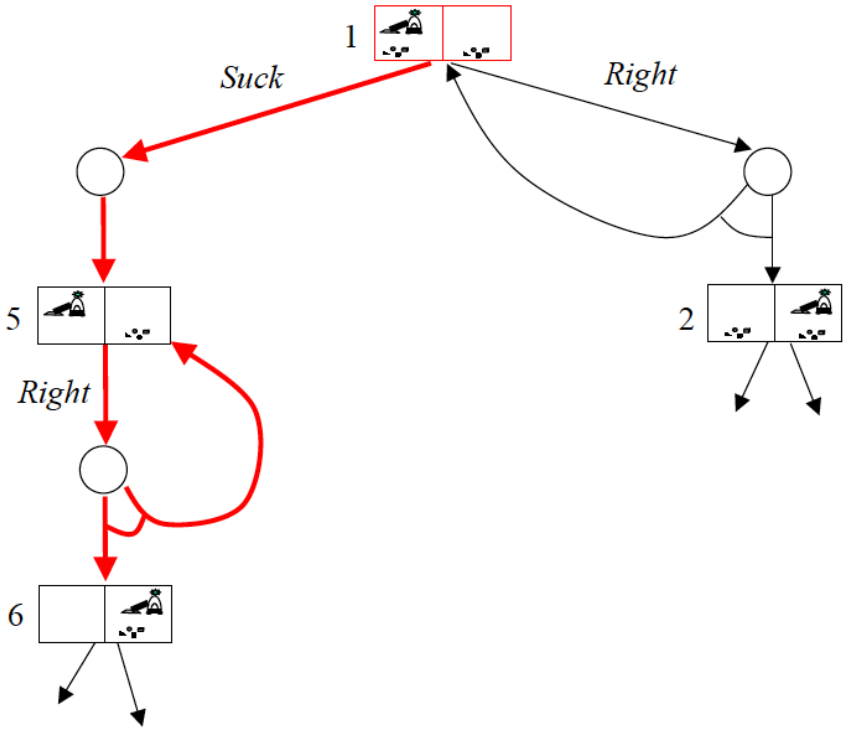
Exemplo 2: agente com falha no movimento

Ação R (Right) e L (Left) do agente

- falha eventualmente
- agente permanece no mesmo local
- exemplo: $\{1\}Right \rightarrow \{1, 2\}$

Solução cíclica

- AND_OR_GRAPH_SEARCH falha
- exemplo solução: aplicar Right até funcionar
- como? rotulando parte do plano



[*Suck*, S_1: *Righ*, **if** *State* = 5 **then** S_1 **else** *Suck*]

while *State* = 5 **do** *Right*

Busca com observações parciais

Conceito importante

- estado crença (belief state) - que representa a crença atual do agente sobre os possíveis estados físicos em que poderia estar, dada a sequência de ações e percepções até aquele ponto.

- conjunto de estados físicos possíveis (dados: sequência de ações e percepts)

Exemplo 1: busca sem observações

Agentes sem sensores

- percepts não fornecem nenhuma informação

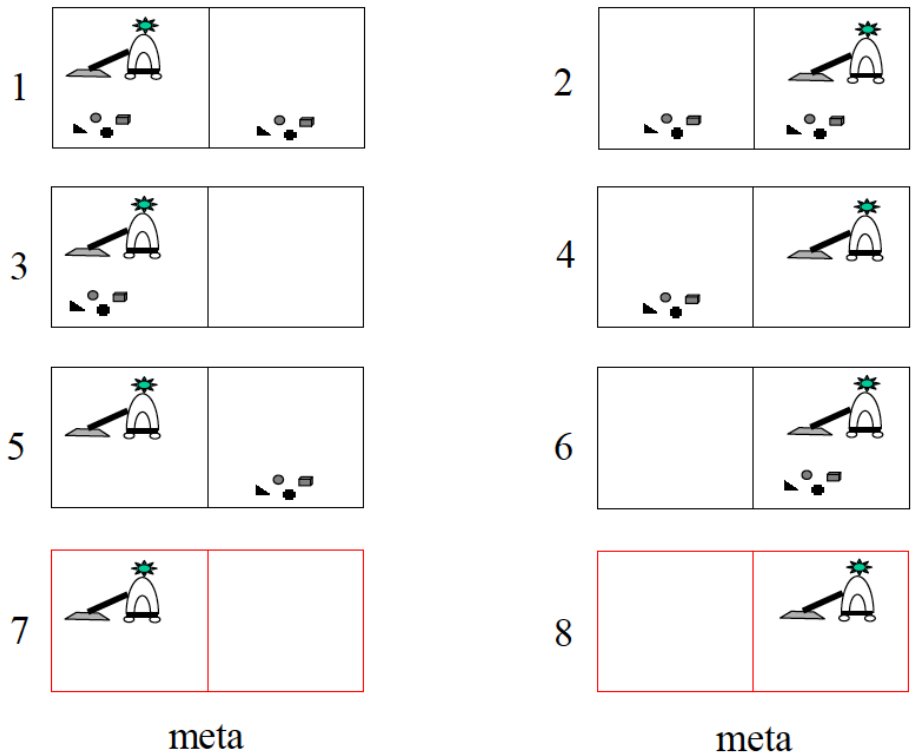
Hipótese: agente conhece geografia do seu mundo

- mas não conhece sua posição
- não conhece a distribuição de sujeira

Estado inicial: um elemento de $\{1, 2, 3, 4, 5, 6, 7, 8\}$

- $\mathit{Right} \rightarrow \{2, 4, 6, 8\}$
- $[\mathit{Right}, \mathit{Suck}] \rightarrow \{4, 8\}$
- $[\mathit{Right}, \mathit{Suck}, \mathit{Left}, \mathit{Suck}]$ sempre atingue $\{7\}$ (meta) para qualquer valor inicial

Exemplo 1: busca sem observações



- *Right* $\rightarrow \{2, 4, 6, 8\}$
- [*Right*, *Suck*] $\rightarrow \{4, 8\}$
- [*Right*, *Suck*, *Left*, *Suck*] sempre atingue $\{7\}$ (meta) para qualquer valor inicial

Busca sem observações

- Espaço de estados crença totalmente observável
- Percepts observados depois das ações são previsíveis!
- Não há contingências
- Formulação problemas busca com estados crença?

Formulação do problema

Problema subjacente P

$ACTIONS_p$, $RESULT_p$, $GOALTEST_p$, $STEPCOST_p$

1. espaço de estados crença
   - conjunto de todos os estados possíveis de P
   - se P tem N estados, então 2^N estados possíveis
   - nem todos estados são atingíveis
2. estado inicial
   - conjunto de todos estados de P

Formulação do problema

Problema subjacente P

$ACTIONS_p$, $RESULT_p$, $GOALTEST_p$, $STEPCOST_p$

3. Ações

- dado um estado crença $b = \{s_1, s_2\}$
- em geral $ACTIONS_p(s_1) \neq ACTIONS_p(s_2)$
- se todas ações são aplicáveis

  $ACTIONS(b) = \cup_{s \in b} ACTIONS_p(s)$
- senão

  $ACTIONS(b) = \cap_{s \in b} ACTIONS_p(s)$
- conjunto das ações aplicáveis em todos os estados

Formulação do problema

Problema subjacente P

$ACTIONS_p$, $RESULT_p$, $GOALTEST_p$, $STEPCOST_p$

4. Modelo de transição

- se ações são determinísticas

  $b' = RESULT(b, a) = \{s' : s' = RESULT_P(s, a) \text{ and } s \in b\}$

- se ações são não determinísticas

  $b' = RESULT(b, a) = \{s' : s' \in RESULTS_p(s, a) \text{ and } s \in b\}$

  $= \cup_{s \in b} RESULTS(s, a)$

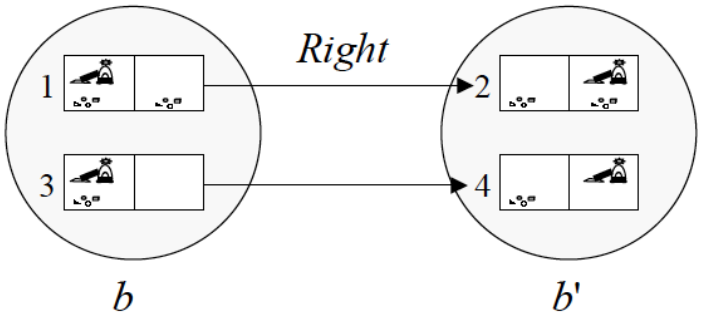
  previsão: $b' = PREDICT_p(b, a)$

Previsão: $b' = PREDICT_p(b, a)$

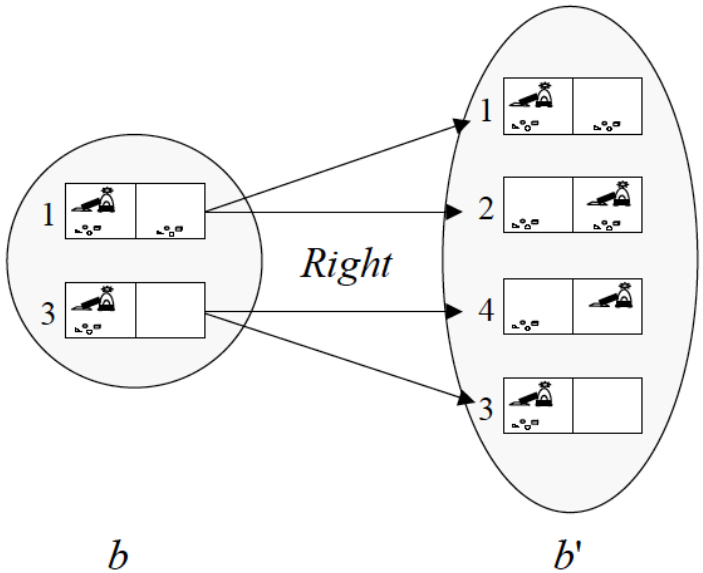
Problema subjacente P

$ACTIONS_p$, $RESULT_p$, $GOALTEST_p$, $STEPCOST_p$

Previsão: $b' = PREDICT_p(b, a)$



Ação determinística



Ação não determinística

Formulação do problema

Problema subjacente P

$ACTIONS_p$, $RESULT_p$, $GOALTEST_p$, $STEPCOST_p$

5. teste de meta

- agente: necessita de um plano que funcione, com certeza
- estado crença satisfaz meta se todos seus elementos satisfazem $GOAL - TEST_p$

6. custo de um caminho

- ações podem ter diferentes custos
- aqui assumimos custos são os mesmos

Formulação do problema

Problema subjacente P

$ACTIONS_p$, $RESULT_p$, $GOALTEST_p$, $STEPCOST_p$

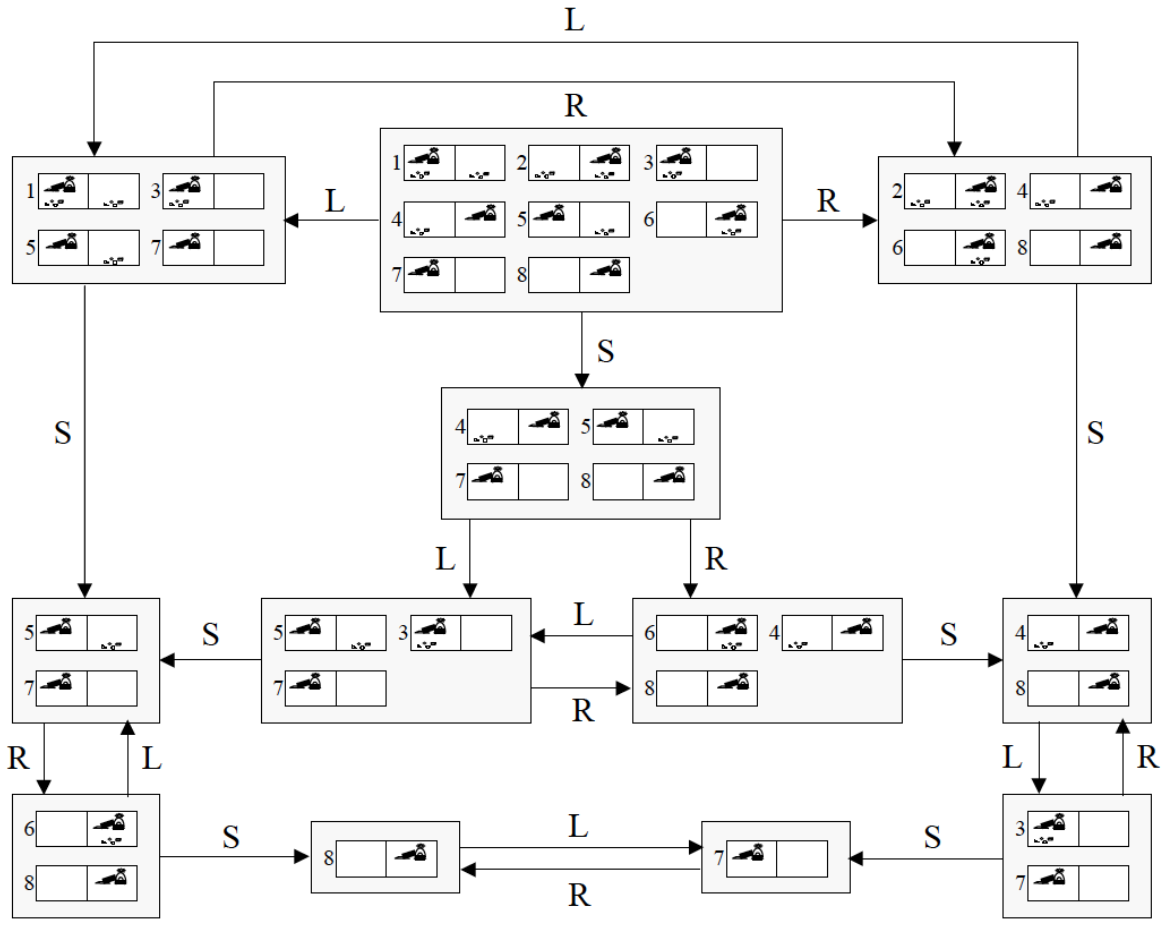
- 12 estados crença atingíveis entre os $2^8 = 256$ possíveis
- algoritmos de busca informados/não informados são aplicáveis se uma sequência de ações é uma solução para um estado crença b então ela é também solução para qualquer subconjunto de b

  exemplo: $[Suck, Left, Suck] \rightarrow \{5, 7\}$

  $[Left] \rightarrow \{1, 3, 5, 7\}$ superconjunto de $\{5, 7\}$

  podemos descartar $\{1, 3, 5, 7\}$ se $\{5, 7\}$ foi gerado

Previsão: $b' = PREDICT_p(b, a)$



Exemplo 2: busca com observações parciais

- Agente com sensores
  - ambiente gera *percepts*
  - sensores do agente fornece informações locais como posição e sujeita na posição (não em outras posições)
- Definição do problema requer função $PERCEPTS(s)$
  - determinístico: retorna o percept recebido em um estado s

    exemplo: PERCEPT({1})=[A,Dirty]
  - não determinístico: PERCEPTS retorna conjunto de *percepts*

    exemplo: observação (parcial)[A, $Dirty$] $\rightarrow$ estado crença - inicial associado $\{1, 3\}$

Formulação do Problema

ACTIONS, GOAL-TEST$_p$, STEP-COST$_P$

- idêntico ao caso do agente sem sensores

Modelo de transição

- transições: estado crença $\rightarrow$ ação $\rightarrow$ estado crença
- ocorrem em três estágios
  1. predição
  2. observação da predição
  3. atualização

Podemos pensar em transições de um estado de crença para o próximo de uma determinada ação como ocorrendo em três etapas.

Formulação do Problema

1. Predição

$\hat{b} = PREDICT(b, a)$

2. Predição da observação

   - determina o conjunto de observações o que poderia ser observado no estado crença previsto

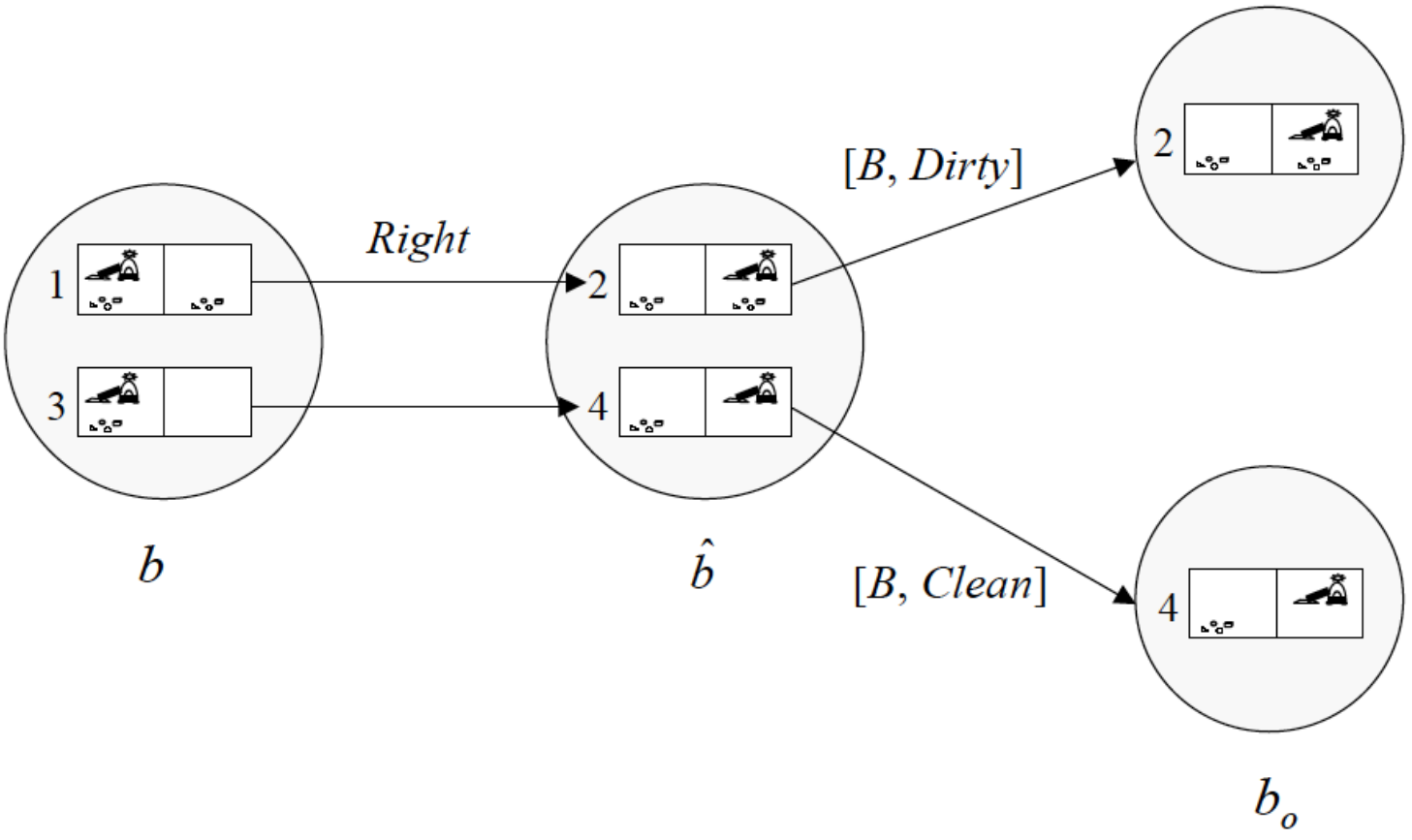
     $POSSIBLE - PERCEPTS(\hat{b}) = \{o : o = PERCEPT(s) \text{ and } s \in \hat{b}\}$

3. Atualização

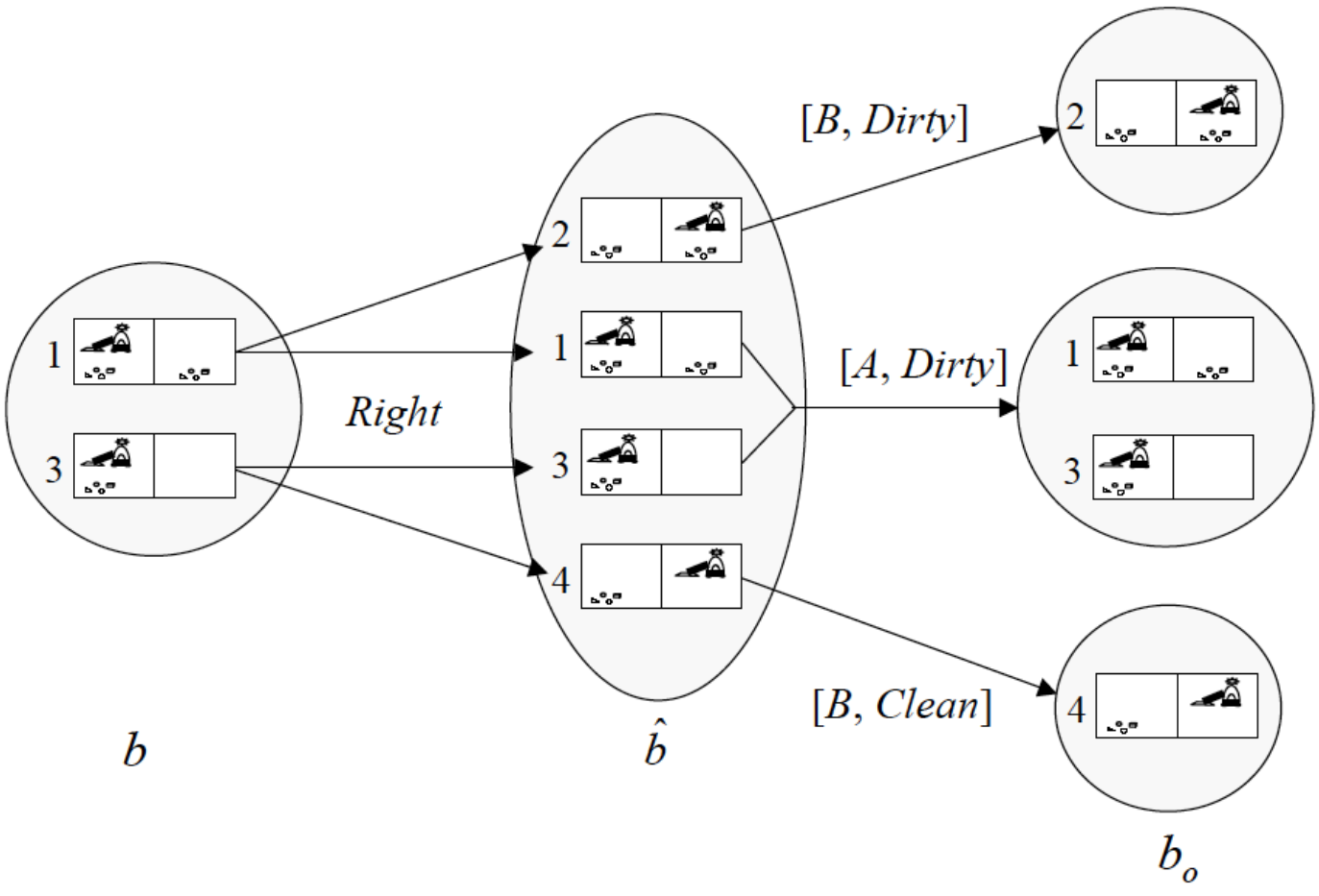
   - determina, para cada *percept* possível, estado crença resultante

     $b_o = UPDATE(\hat{b}, o) = \{s : o = PERCEPT(s) \text{ and } s \in \hat{b}\}$

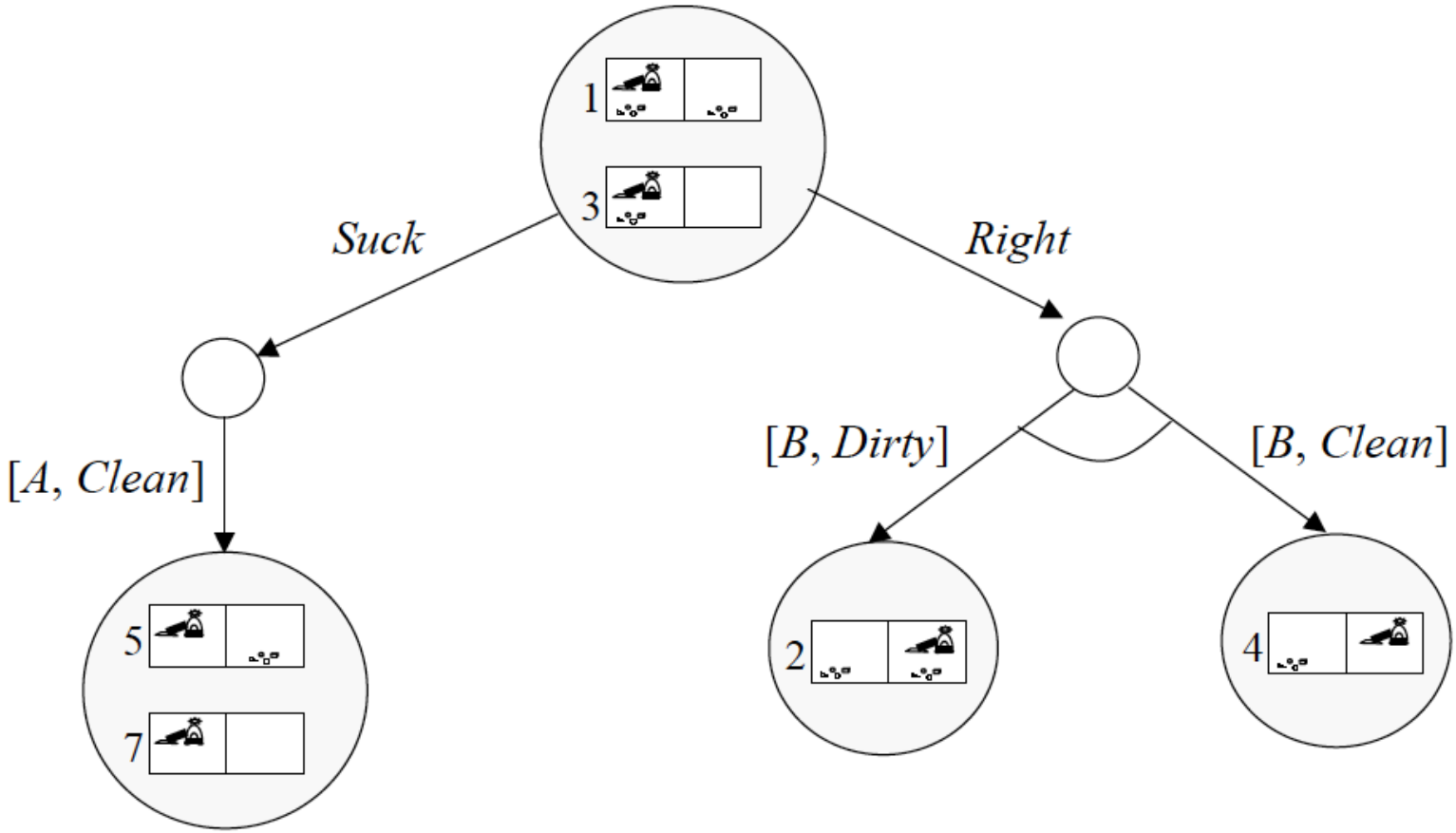
Transição de estado com ação determinística



Transição de estado com ação não determinística



Resolvendo problemas parcialmente observáveis



[*Suck*, *Right*, *if B State* $= \{6\}$ *then Suck else*[]]

Agente em ambientes parcialmente observáveis

Projeto do agente

- similar ao *SIMPLE_PROBLEM_SOLVING_AGENT*(*percept*)
  - formula o problema
  - chama algoritmo busca
  - executa ações

Agente em ambientes parcialmente observáveis

Projeto do agente

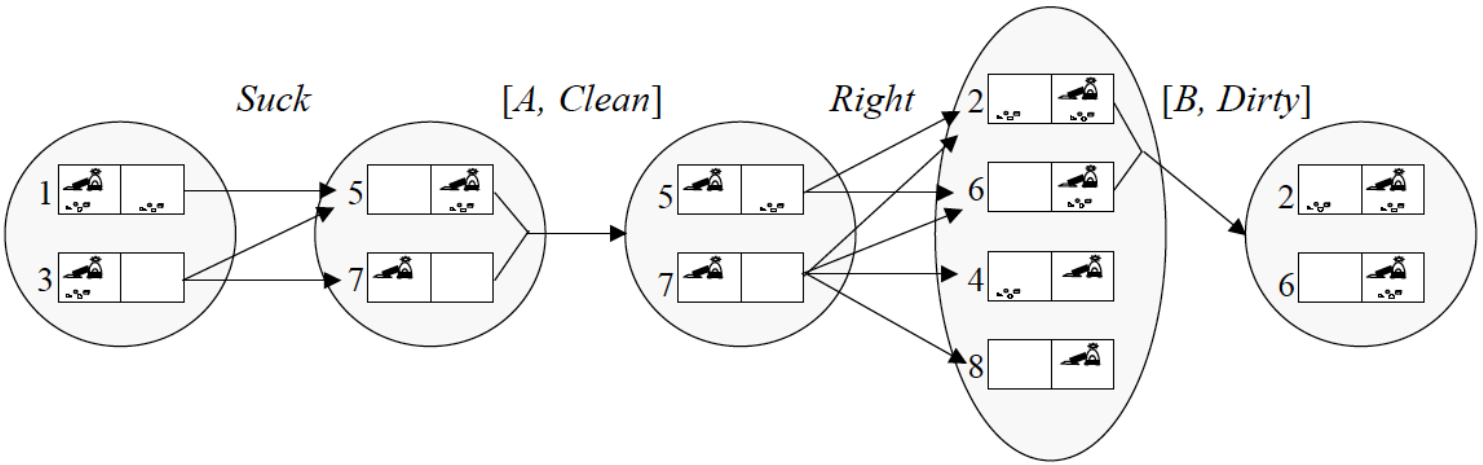
- Em ambientes parcialmente observáveis
  - solução é plano condicional
  - agente tem que manter o estado crença
  - estado crença é mais fácil de ser estimado
  - se primeiro passo é if-then-else: verificar if e executar then ou else
  - atualiza estado crença ao executar ações e receber *percepts*
  - atualização estado crença mais simples porque

    agente não estima/calcula o *percept*

    agente usa o percept *o* fornecido pelo ambiente

    $b' = UPDATE(PREDICT(b, a), o)$

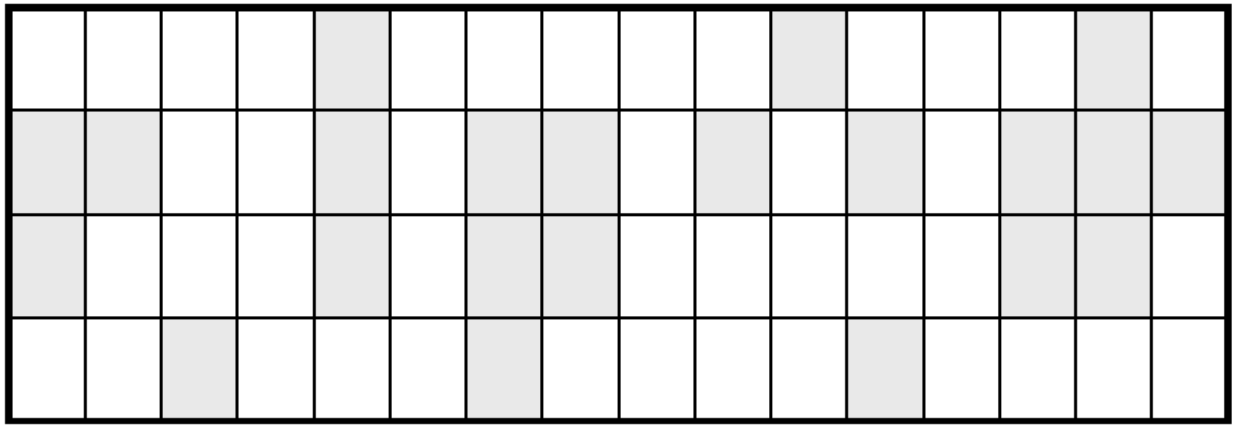
Ciclo predição-atualização do estado crença



$b' = UPDATE(PREDICT(b, a), o)$

kindergarten world com sensor local: qualquer local pode ficar sujo, em qualquer instante, a menos que o agente esteja aspirando o ambiente.

Exemplo: localização posição

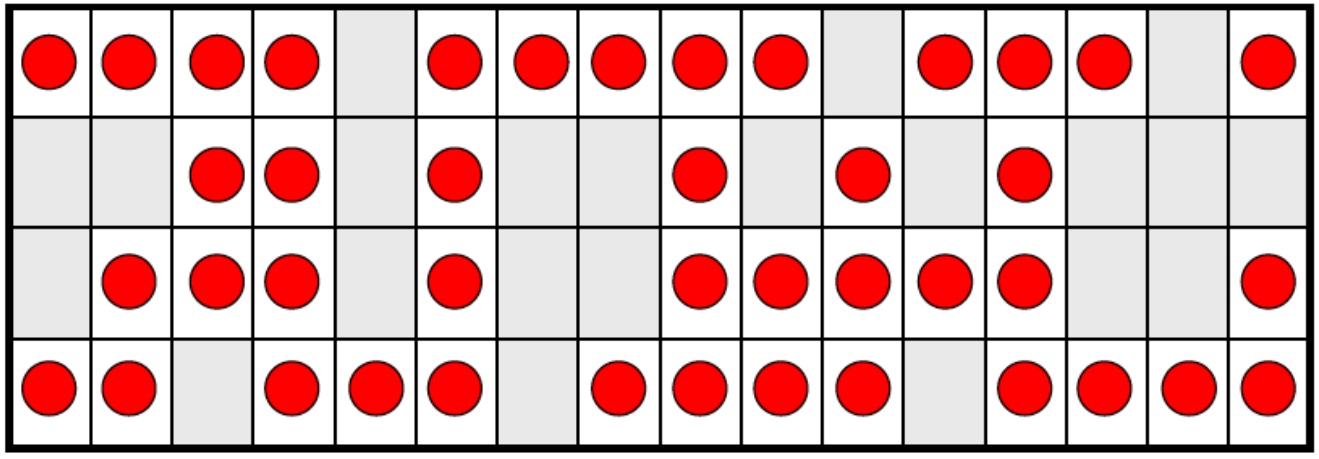


robô

obstáculos

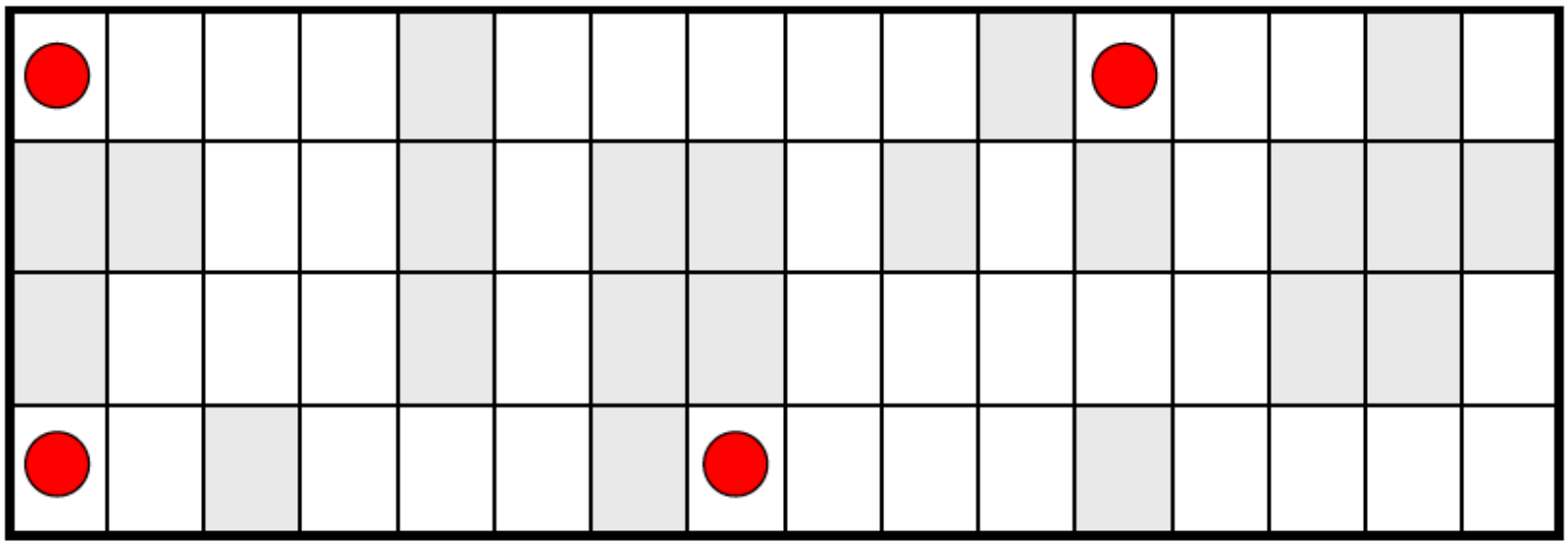
sensores: 4 sonares
detectam obstáculos
direção obstáculo: *N*, *S*, *W* ou *E*
ação: *Move* (posição adjacente aleatória)

Posição atual do robô ?

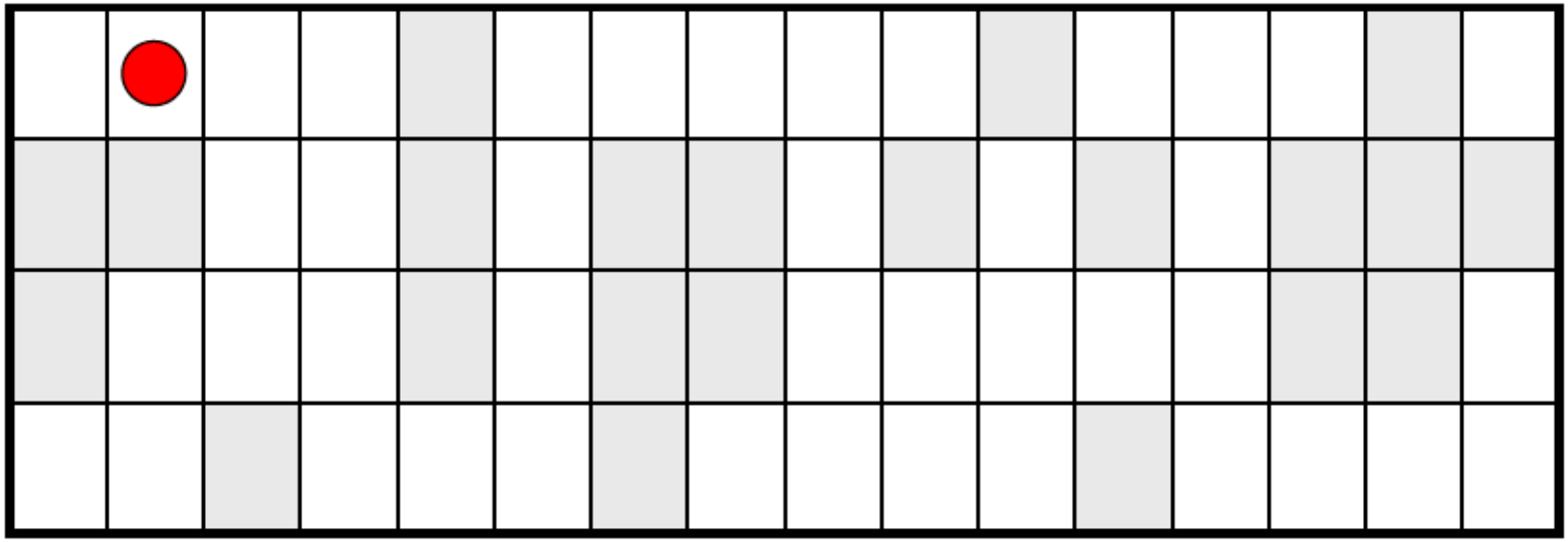


estado inicial: b

$o = NSW$



$b_o = \text{UPDATE}(b, o)$



$$b_a = PREDICT(b_o, Move)$$

$$b_n = UPDATE(b_a, o)$$

$$b_n = UPDATE(PREDICT(UPDATE(b, NSW), Move), NS)$$

Agente com busca online

Agente online

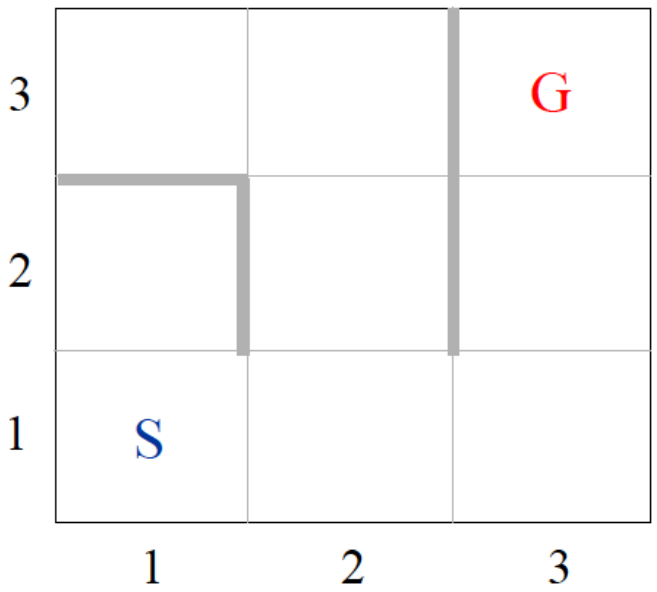
- executa ciclos: ação-observação-ação-...
- importante em ambientes dinâmicos e não determinísticos
- necessário em ambientes desconhecidos
  - não conhece estados
  - não sabe que ações executar
  - ações como experimentos de aprendizagem

Conhecimento do Agente

Ambiente determinístico e inteiramente observável

- $ACTIONS(s)$: retorna lista das ações permitidas no estado s
- $c(s, a, s')$: custo de um passo, só pode ser calculado quando agente sabe que está em s'
- $GOALTEST(s)$
- $RESULT(s, a)$: determinado somente quando agente está em s e executa ação a
- Função heurística $h(n)$
- Objetivo: atingir meta com menor custo, explorar ambiente

Exemplo



S estado inicial

G meta

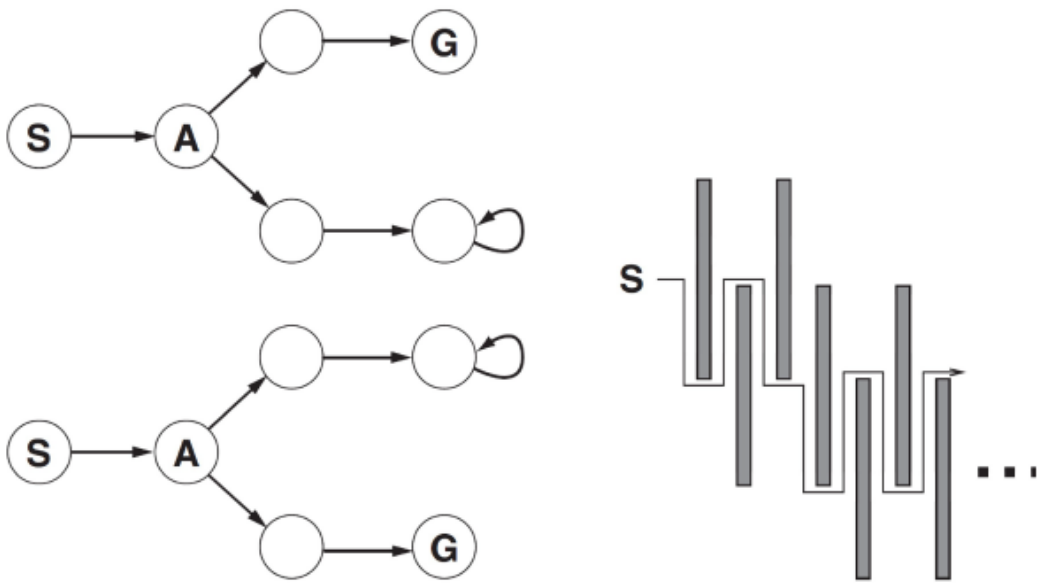
Ações: Subir, Descer, Direita, Esquerda

$h(n)$: distância Manhattan

Agente não sabe que da posição (1,1) e ação UP $\rightarrow$ nova posição (1,2)

Exemplo

- razão competitiva
- pode ser infinita se ações são irreversíveis

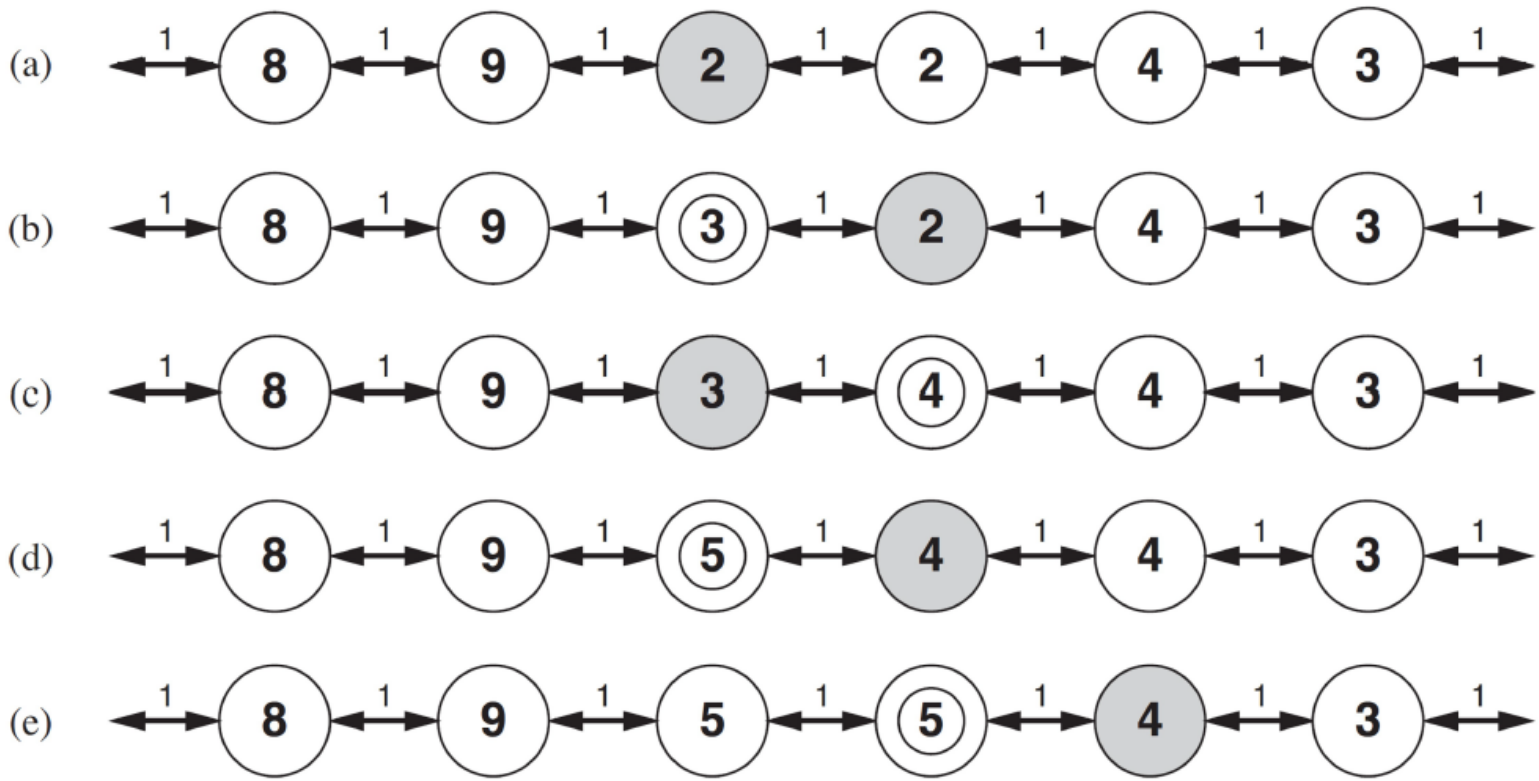


Busca em profundidade Online

function ON_LINE_DFS_AGENT (*s'*) **returns** an action
  inputs: *s'*, a percept that identifies the current state
  persistent: *result*, table indexed by state and action, initially empty
    untried, table that lists, for each state, actions not yet tried
    unbacktracked, table that lists, for each state, backtracks not yet tried

  if GOAL-TEST(*s'*) **then return** *stop*
  if *s'* is a new state (not in *untried*) **then** *untried* [*s'*] ← ACTIONS(*s'*)
  if *s* is not null **then**
    result [*s*,*a*] ← *s'*
    add *s* to the front of *unbacktracked* [*s'*]
  if *untried* [*s'*] is empty **then**
    if *unbacktracked* [*s'*] is empty **then return** *stop*
    else *a* ← an action *b* such that *result* [*s'*,*b*] = POP(*unbacktracked* [*s'*])
  else *a* ← POP(*untried* [*s'*])
  s ← *s'*
  return *a*

Busca online local e aprendizagem



function LRTA*_AGENT (s') **returns** an action

inputs: s', a percept that identifies the current state
static: *result*, table indexed by state and action, initially empty
H, a table of costs 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** $H[s'] \leftarrow h(s')$
if s is not null
$result[s,a] \leftarrow s'$
$H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)}$ LRTA*_COST($s,b,result[s,b],H$)
$a \leftarrow$ action b in ACTIONS(s') that minimizes LRTA*_COST($s',b,result[s',b],H$)
$s \leftarrow s'$
return a

function LRTA*_COST (s, a, s', H) **returns** a cost estimate
if s' is undefined **then return** $h(s)$
else return $c(s, a, s') + H[s']$

Dúvidas

Dúvida é o começo da sabedoria.