# Intro al MidTerm 1: Soluzioni

Corso di Intelligenza Artificiale, a.a. 2024-25

R. Basili

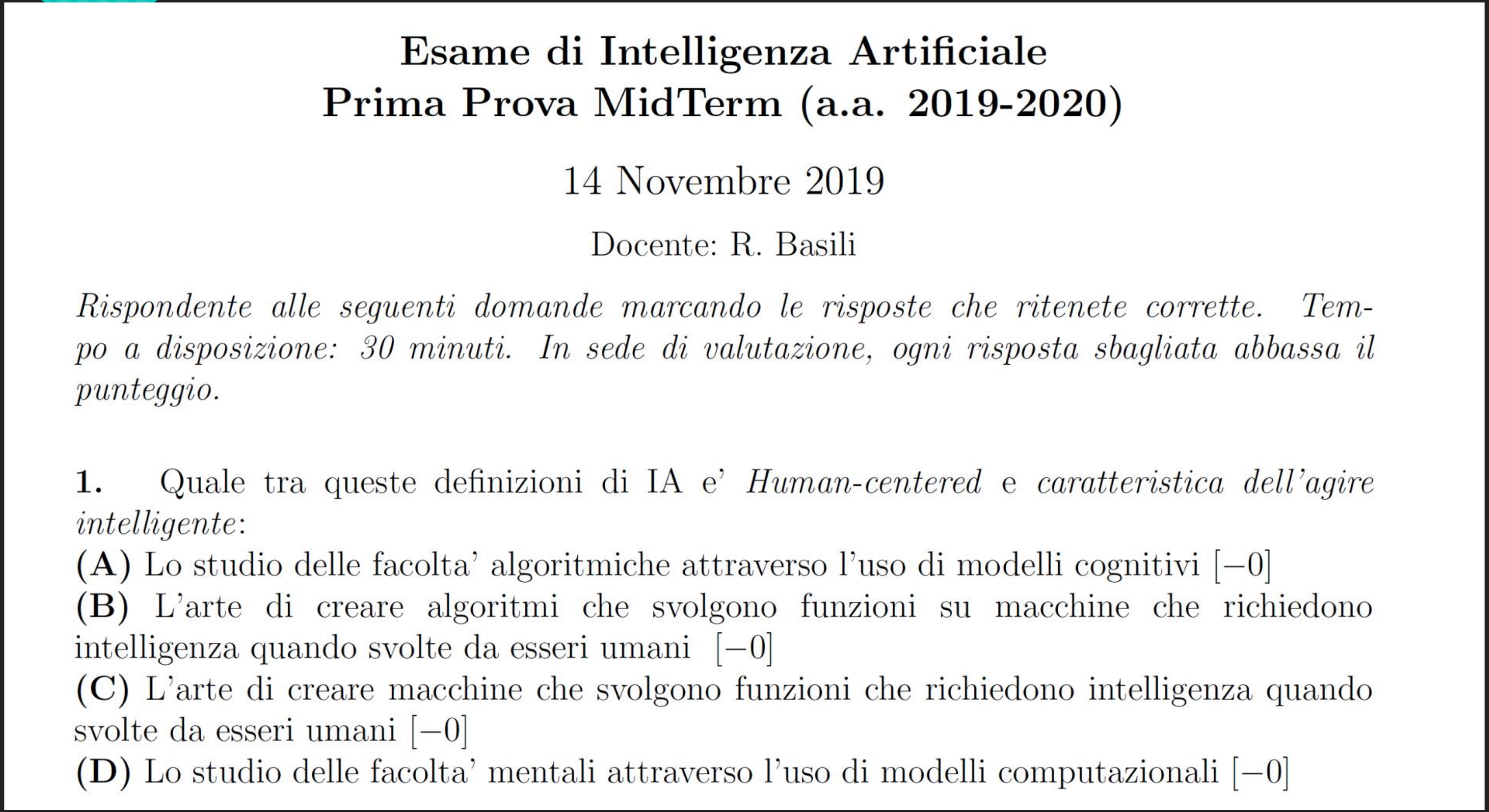
## Outline

- Programma del MidTerm
- Esempi di domande a risp. multiple
  - Introduzione all'AI
  - Agenti
    - Scopi e metodologie
    - Ambiente
    - Conoscenza e Modelli
  - Problem Solving: Ricerca Semplice
  - Problem Solving: Ricerca Informata
  - Tecniche avanzate di ricerca
  - Logica: Sintassi e Modelli
- Esempi domande aperte
- Altri Esercizi libro
- Progettazione di Agenti: *Vacuum Agent World*

## Programma

- Il MidTerm insistera’ sulle seguenti tematiche (vedi AIMA book)
  - Introduzione all’AI (Chapt. 1)
  - Agenti  (Chapt 2)
    - Scopi e metodologie
    - Ambiente
    - Conoscenza e Modelli
  - Problem Solving: Ricerca Semplice (Chapt. 3, 3.1-3.4)
  - Problem Solving: Ricerca Informata (Capt. 3, 3.5-3.6)
  - Tecniche avanzate di ricerca (Capt. 4, 4.1, 4.4; Chapt. 5, 5.1-5.3)
  - Logica Proposizionale: Sintassi e Semantica

# Domane a Risposte Chiuse



**Esame di Intelligenza Artificiale**
**Prima Prova MidTerm (a.a. 2019-2020)**

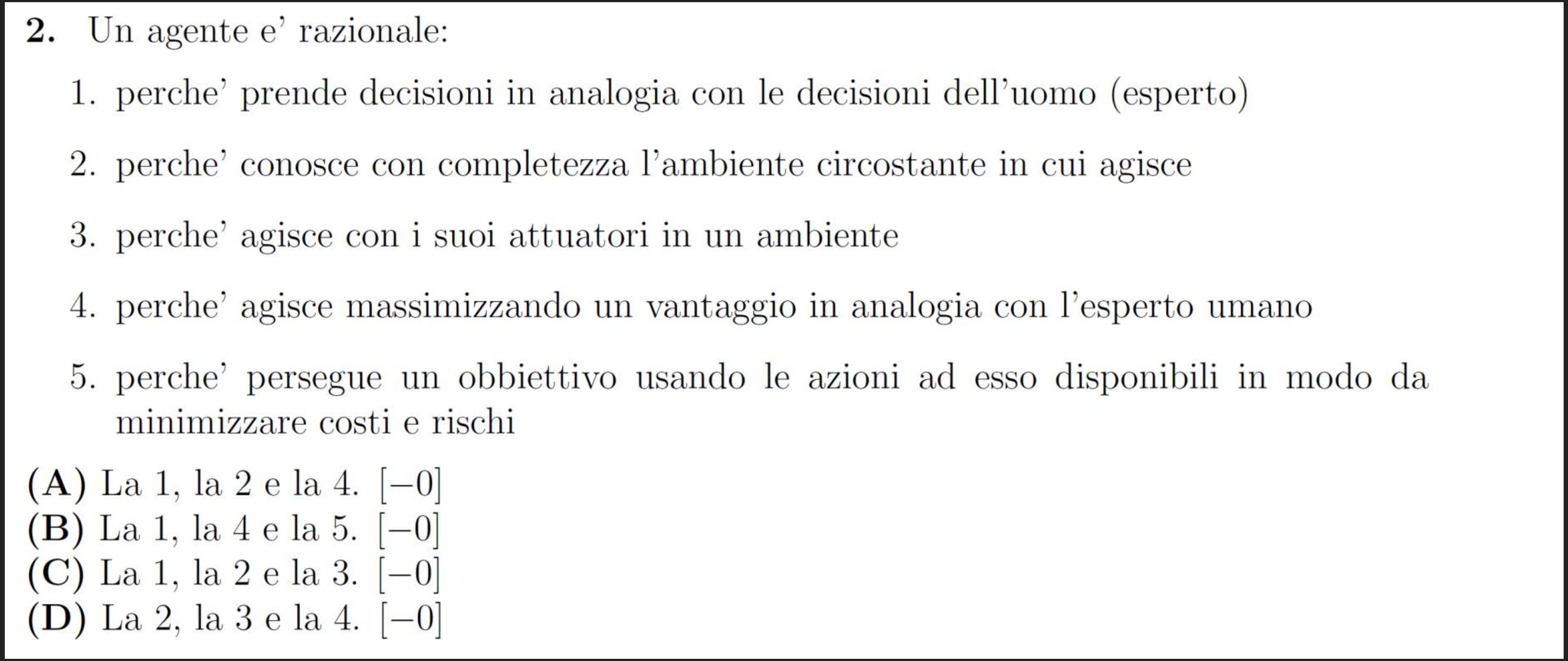
14 Novembre 2019

Docente: R. Basili

*Rispondente alle seguenti domande marcando le risposte che ritenete corrette. Tempo a disposizione: 30 minuti. In sede di valutazione, ogni risposta sbagliata abbassa il punteggio.*

**1.** Quale tra queste definizioni di IA e' *Human-centered* e *caratteristica dell'agire intelligente*:
**(A)** Lo studio delle facolta' algoritmiche attraverso l'uso di modelli cognitivi [−0]
**(B)** L'arte di creare algoritmi che svolgono funzioni su macchine che richiedono intelligenza quando svolte da esseri umani [−0]
**(C)** L'arte di creare macchine che svolgono funzioni che richiedono intelligenza quando svolte da esseri umani [−0]
**(D)** Lo studio delle facolta' mentali attraverso l'uso di modelli computazionali [−0]

# Domande a Risposte Chiuse (2)

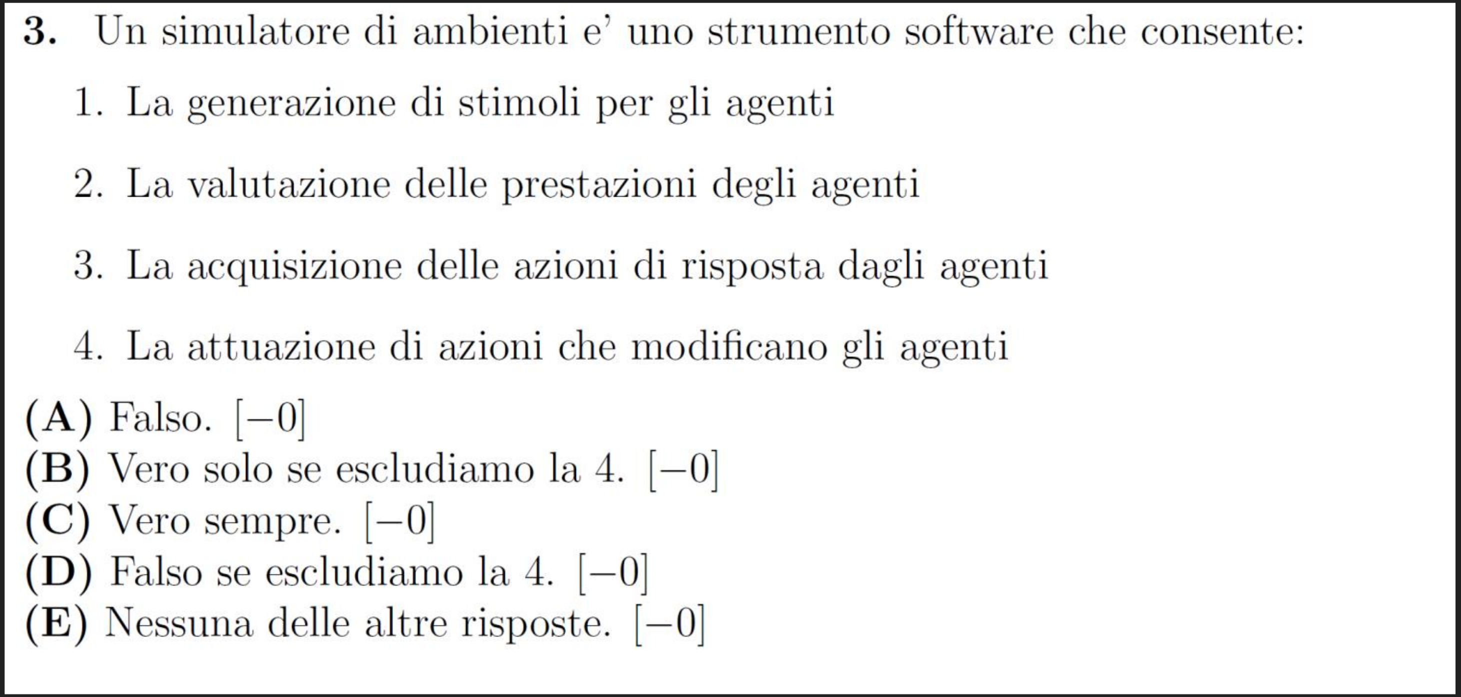


**2.** Un agente e' razionale:

1. perche' prende decisioni in analogia con le decisioni dell'uomo (esperto)
2. perche' conosce con completezza l'ambiente circostante in cui agisce
3. perche' agisce con i suoi attuatori in un ambiente
4. perche' agisce massimizzando un vantaggio in analogia con l'esperto umano
5. perche' persegue un obbiettivo usando le azioni ad esso disponibili in modo da minimizzare costi e rischi

**(A)** La 1, la 2 e la 4. [−0]
**(B)** La 1, la 4 e la 5. [−0]
**(C)** La 1, la 2 e la 3. [−0]
**(D)** La 2, la 3 e la 4. [−0]

# Domande a Risposte Chiuse (3)



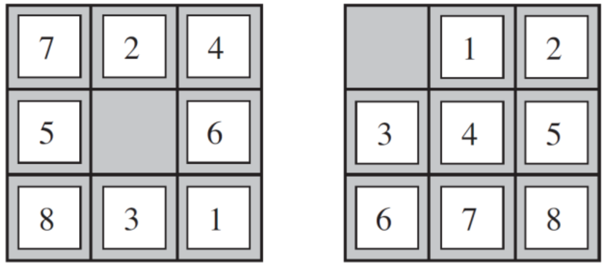
**3.** Un simulatore di ambienti e' uno strumento software che consente:

1. La generazione di stimoli per gli agenti
2. La valutazione delle prestazioni degli agenti
3. La acquisizione delle azioni di risposta dagli agenti
4. La attuazione di azioni che modificano gli agenti

**(A)** Falso. $[-0]$
**(B)** Vero solo se escludiamo la 4. $[-0]$
**(C)** Vero sempre. $[-0]$
**(D)** Falso se escludiamo la 4. $[-0]$
**(E)** Nessuna delle altre risposte. $[-0]$

## Domande a Risposte chiuse (4)

**4.** Date le due configurazione del gioco dell'8 puzzle qui sotto rappresentate



Start State Goal State

si determini la affermazione corretta:

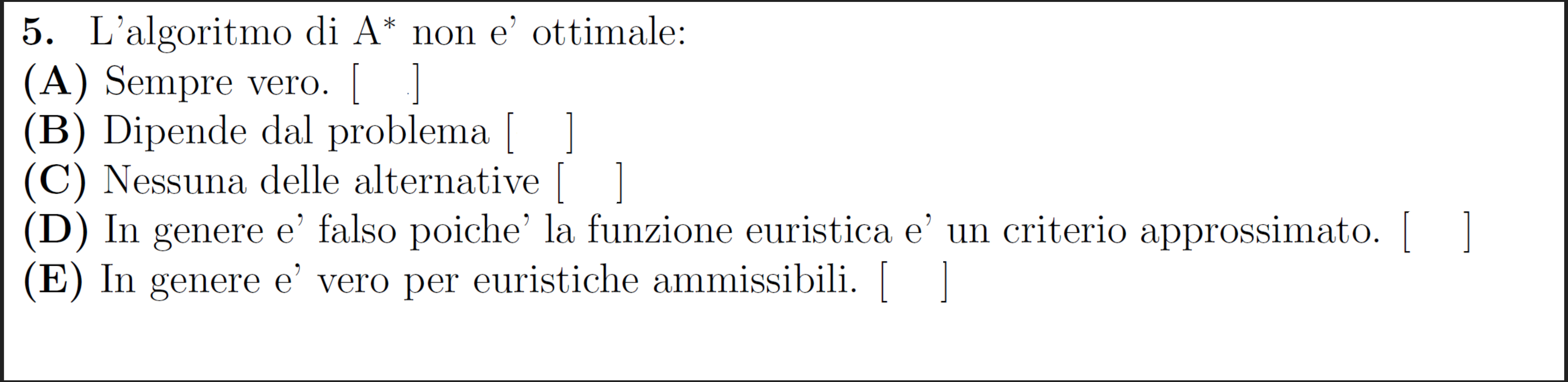
**(A)** La distanza euclidea tra le due configurazione conta il quadrato del numero di mosse che porta la prima nella seconda [−0]

**(B)** Una euristica possibile considera il numero di caselle che sono ordinate nello stato corrente [−0]

**(C)** Una euristica possibile (Manhattan distance) conta il numero di spostamenti (orizzonatli e verticali) da applicare alle diverse caselle dello Start State per condurle tutte nella posizione ad esse assegnata nel Goal State finale. [−0]

**(D)** La combinazione lineare di euristiche diverse non Ã¨ una euristica valida [−0]

## Domande a Risposte Chiuse (5)


**5.** L'algoritmo di A* non e' ottimale:

**(A)** Sempre vero. [ ]

**(B)** Dipende dal problema [ ]

**(C)** Nessuna delle alternative [ ]

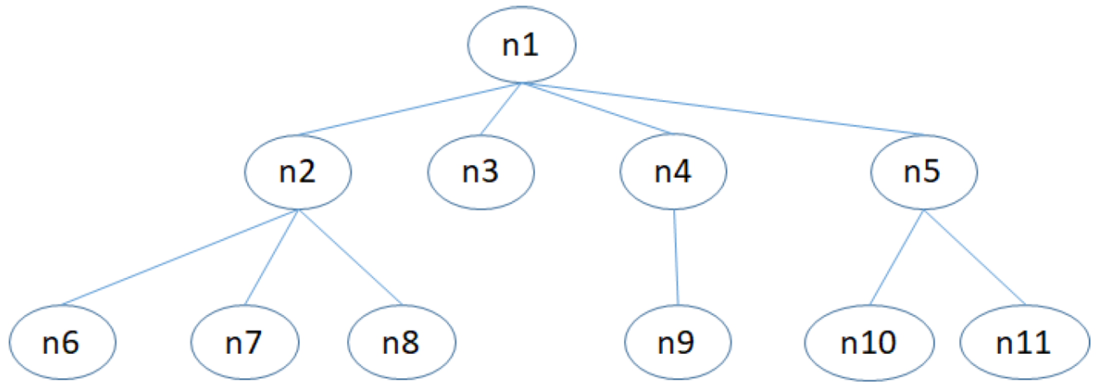
**(D)** In genere e' falso poiche' la funzione euristica e' un criterio approssimato. [ ]

**(E)** In genere e' vero per euristiche ammissibili. [ ]

## Domande a Risposte Chiuse (6): Algoritmi di Ricerca

# Albero di Ricerca Uninformed

Dato il seguente spazio di stati: determinare tra le seguenti la sequenza generata



dall'algoritmo di *depth-first* nella ricerca del nodo $n3$:

**(A)** n1,n2,n3

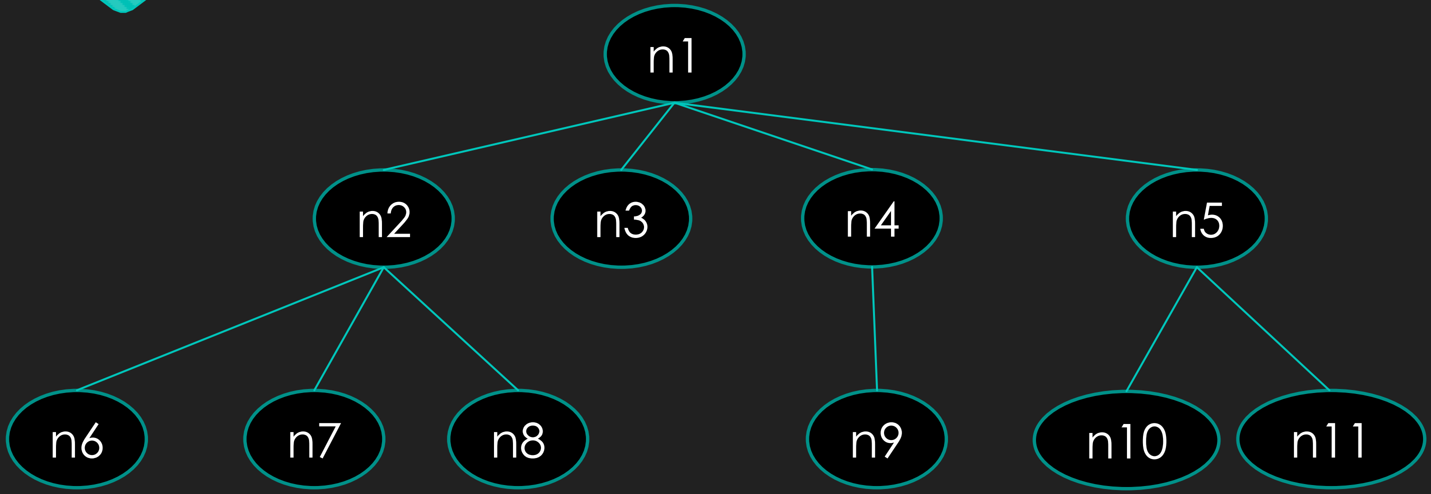
**(B)** n1,n2,n6,n7,n8,n4,n9,n5,n10,n11,n3.

**(C)** Non è possibile stabilirlo perche' manca la euristica o il costo utilizzati.

**(D)** n1,n2,n6,n7,n8,n3.

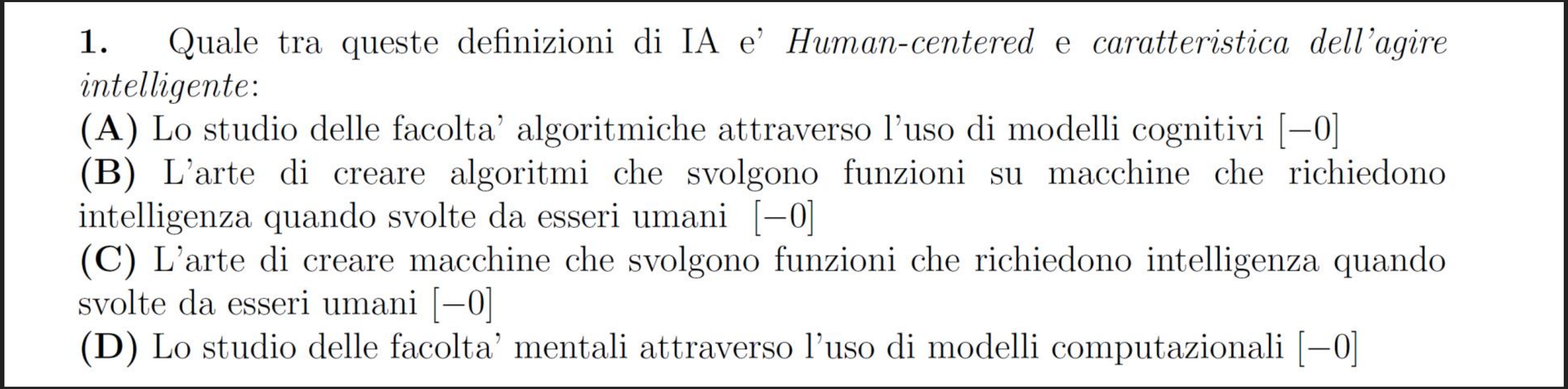
**(E)** Nessuna delle altre risposte.

## Algoritmi di Ricerca con Euristica



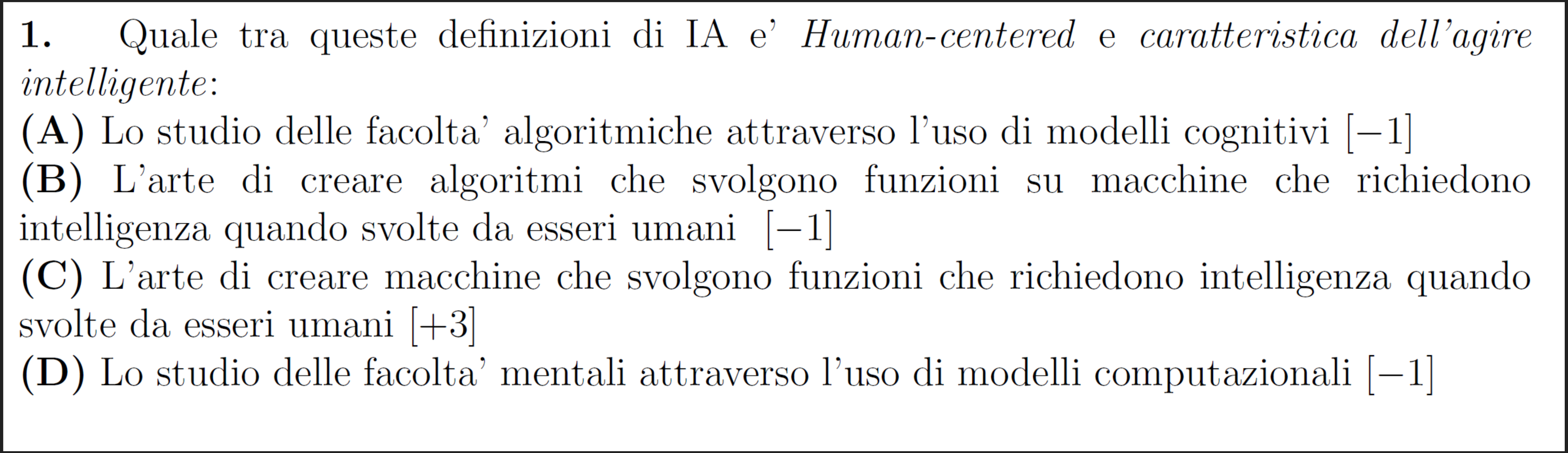
## Domande a Risposte Chiuse Soluzioni

## Domande a Risposte Chiuse



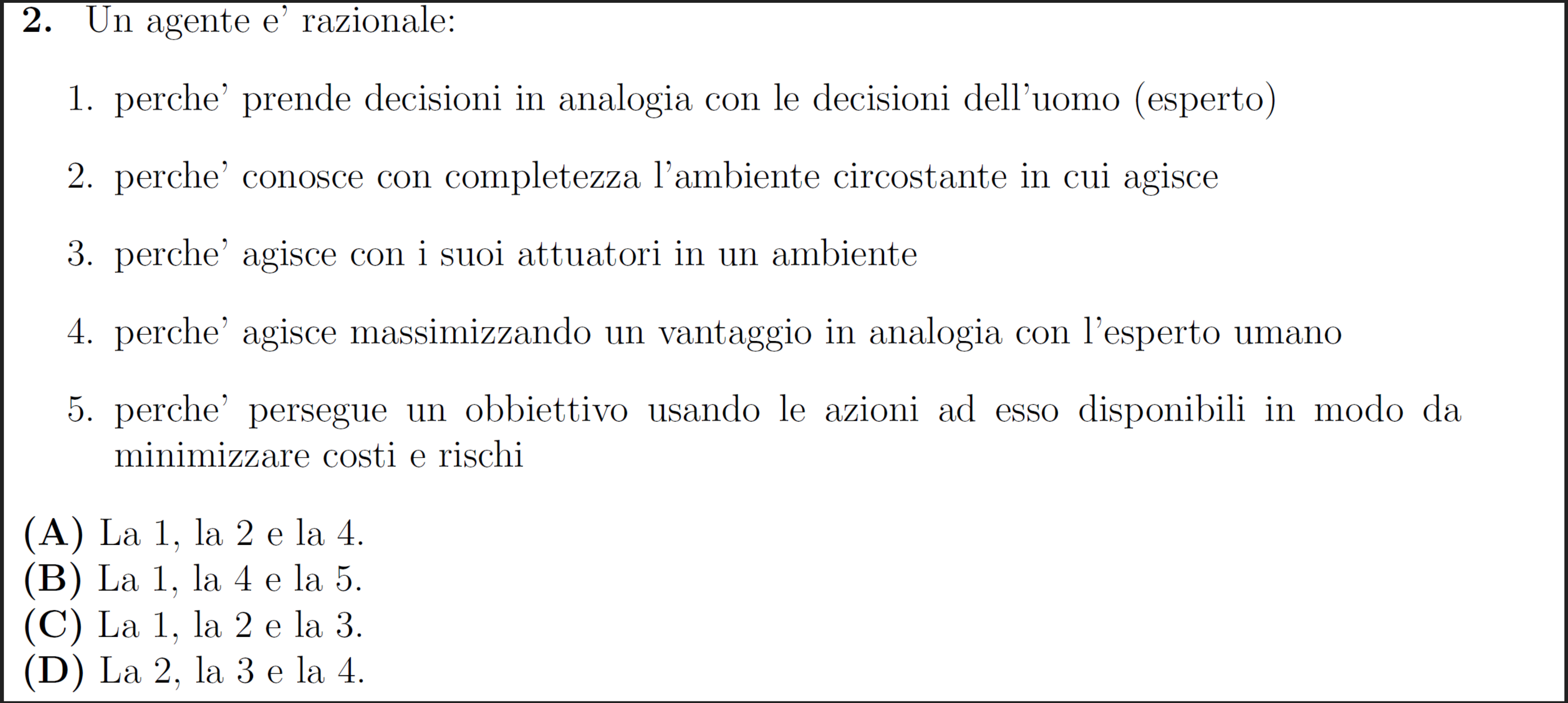
**1.** Quale tra queste definizioni di IA e' *Human-centered* e *caratteristica dell'agire intelligente*:
**(A)** Lo studio delle facolta' algoritmiche attraverso l'uso di modelli cognitivi [−0]
**(B)** L'arte di creare algoritmi che svolgono funzioni su macchine che richiedono intelligenza quando svolte da esseri umani [−0]
**(C)** L'arte di creare macchine che svolgono funzioni che richiedono intelligenza quando svolte da esseri umani [−0]
**(D)** Lo studio delle facolta' mentali attraverso l'uso di modelli computazionali [−0]

## R a domanda 1



**1.** Quale tra queste definizioni di IA e' *Human-centered* e *caratteristica dell'agire intelligente*:
**(A)** Lo studio delle facolta' algoritmiche attraverso l'uso di modelli cognitivi [−1]
**(B)** L'arte di creare algoritmi che svolgono funzioni su macchine che richiedono intelligenza quando svolte da esseri umani [−1]
**(C)** L'arte di creare macchine che svolgono funzioni che richiedono intelligenza quando svolte da esseri umani [+3]
**(D)** Lo studio delle facolta' mentali attraverso l'uso di modelli computazionali [−1]

# R a domanda 2

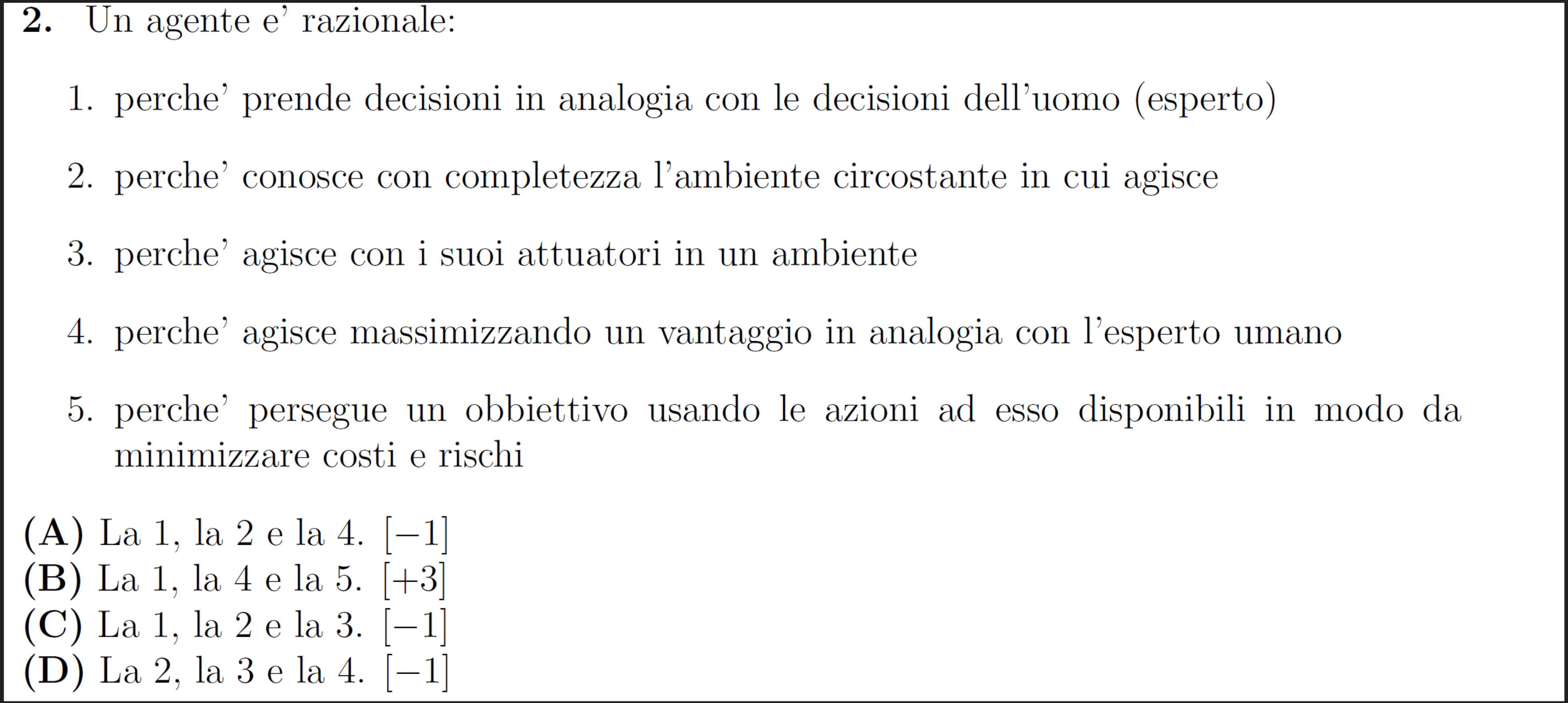


**2.** Un agente e' razionale:

1. perche' prende decisioni in analogia con le decisioni dell'uomo (esperto)
2. perche' conosce con completezza l'ambiente circostante in cui agisce
3. perche' agisce con i suoi attuatori in un ambiente
4. perche' agisce massimizzando un vantaggio in analogia con l'esperto umano
5. perche' persegue un obbiettivo usando le azioni ad esso disponibili in modo da minimizzare costi e rischi

**(A)** La 1, la 2 e la 4.
**(B)** La 1, la 4 e la 5.
**(C)** La 1, la 2 e la 3.
**(D)** La 2, la 3 e la 4.

# R a domanda 2

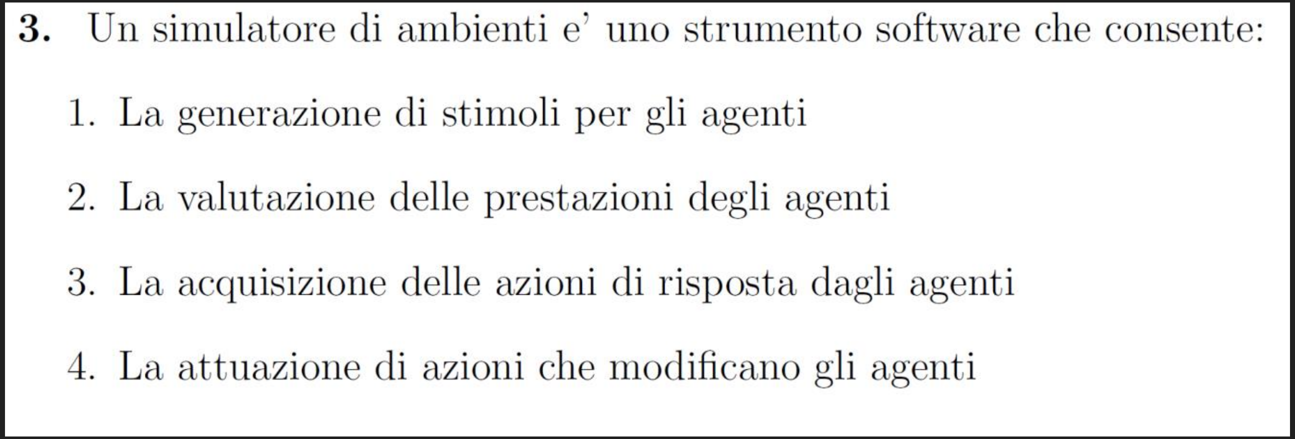


**2.** Un agente e' razionale:

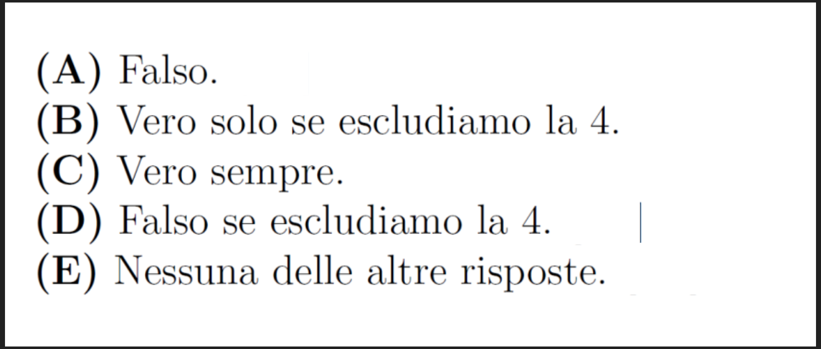
1. perche' prende decisioni in analogia con le decisioni dell'uomo (esperto)
2. perche' conosce con completezza l'ambiente circostante in cui agisce
3. perche' agisce con i suoi attuatori in un ambiente
4. perche' agisce massimizzando un vantaggio in analogia con l'esperto umano
5. perche' persegue un obbiettivo usando le azioni ad esso disponibili in modo da minimizzare costi e rischi

**(A)** La 1, la 2 e la 4. [−1]
**(B)** La 1, la 4 e la 5. [+3]
**(C)** La 1, la 2 e la 3. [−1]
**(D)** La 2, la 3 e la 4. [−1]

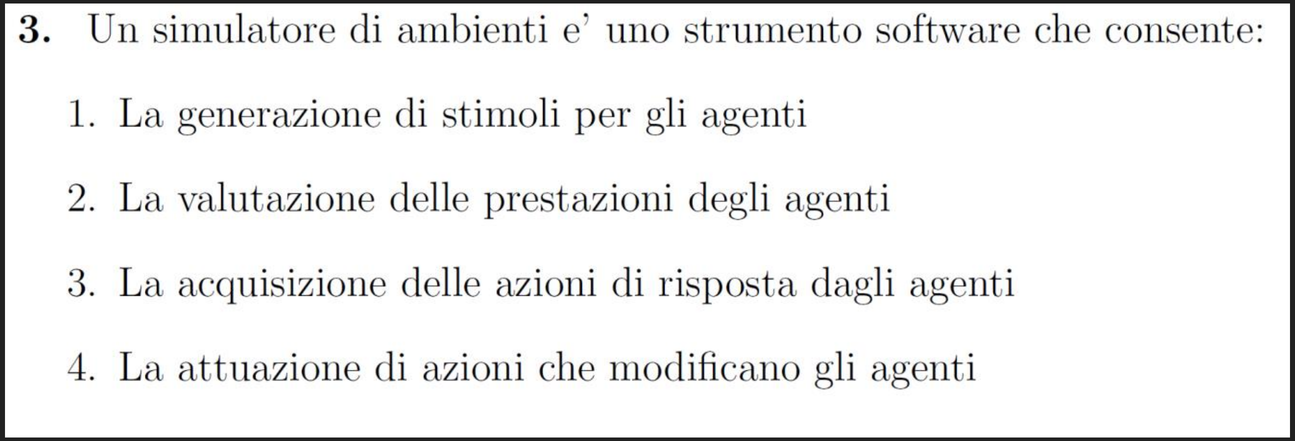
# R a domanda 3


**3.** Un simulatore di ambienti e' uno strumento software che consente:

1. La generazione di stimoli per gli agenti
2. La valutazione delle prestazioni degli agenti
3. La acquisizione delle azioni di risposta dagli agenti
4. La attuazione di azioni che modificano gli agenti

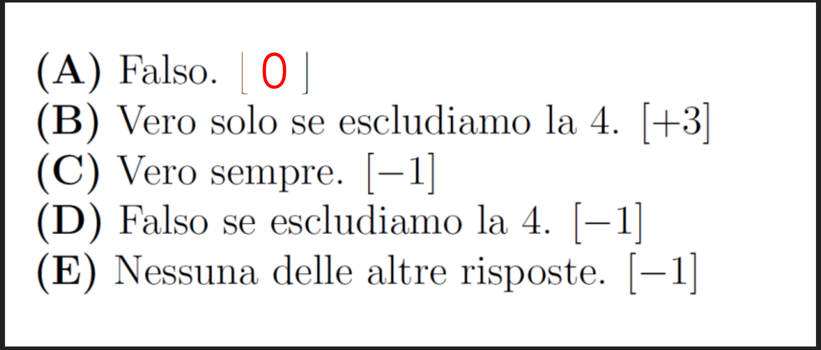

**(A)** Falso.
**(B)** Vero solo se escludiamo la 4.
**(C)** Vero sempre.
**(D)** Falso se escludiamo la 4.
**(E)** Nessuna delle altre risposte.

## R a domanda 3



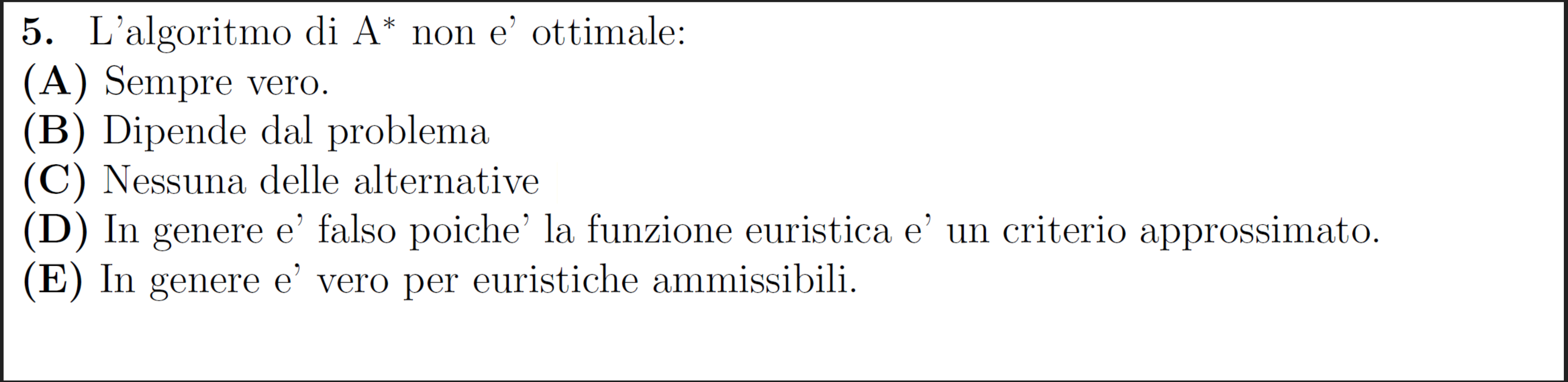
**3.** Un simulatore di ambienti e' uno strumento software che consente:

1. La generazione di stimoli per gli agenti
2. La valutazione delle prestazioni degli agenti
3. La acquisizione delle azioni di risposta dagli agenti
4. La attuazione di azioni che modificano gli agenti

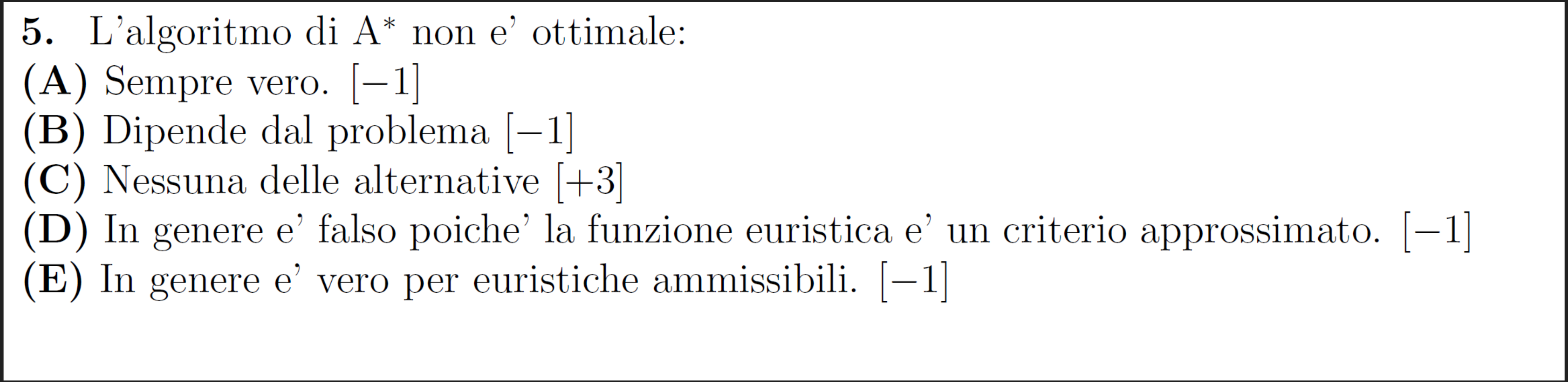


**(A)** Falso. [0]
**(B)** Vero solo se escludiamo la 4. [+3]
**(C)** Vero sempre. [−1]
**(D)** Falso se escludiamo la 4. [−1]
**(E)** Nessuna delle altre risposte. [−1]

## R a domanda 5


**5.** L'algoritmo di A* non e' ottimale:
**(A)** Sempre vero.
**(B)** Dipende dal problema
**(C)** Nessuna delle alternative
**(D)** In genere e' falso poiche' la funzione euristica e' un criterio approssimato.
**(E)** In genere e' vero per euristiche ammissibili.

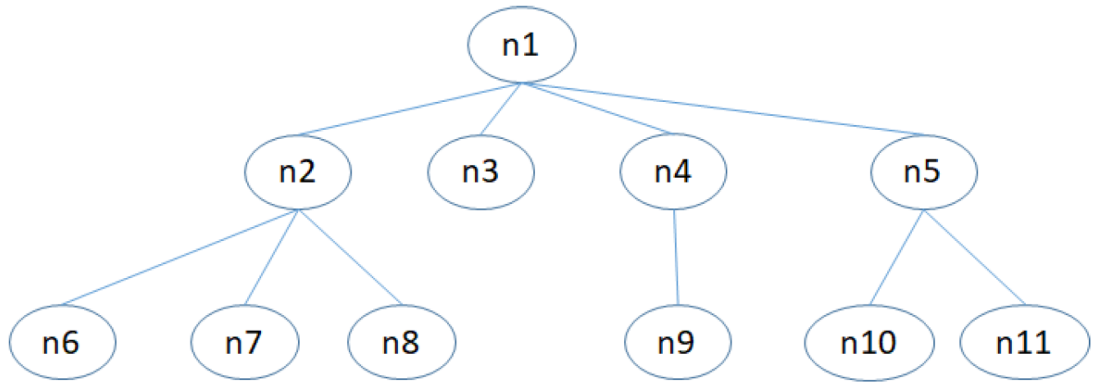
# R a domanda 5



**5.** L'algoritmo di A$^*$ non e' ottimale:
**(A)** Sempre vero. [−1]
**(B)** Dipende dal problema [−1]
**(C)** Nessuna delle alternative [+3]
**(D)** In genere e' falso poiche' la funzione euristica e' un criterio approssimato. [−1]
**(E)** In genere e' vero per euristiche ammissibili. [−1]

# R. Algoritmi di Ricerca non informata

Dato il seguente spazio di stati: determinare tra le seguenti la sequenza generata



dall'algoritmo di *depth-first* nella ricerca del nodo $n3$:

**(A)** n1,n2,n3

**(B)** n1,n2,n6,n7,n8,n4,n9,n5,n10,n11,n3.

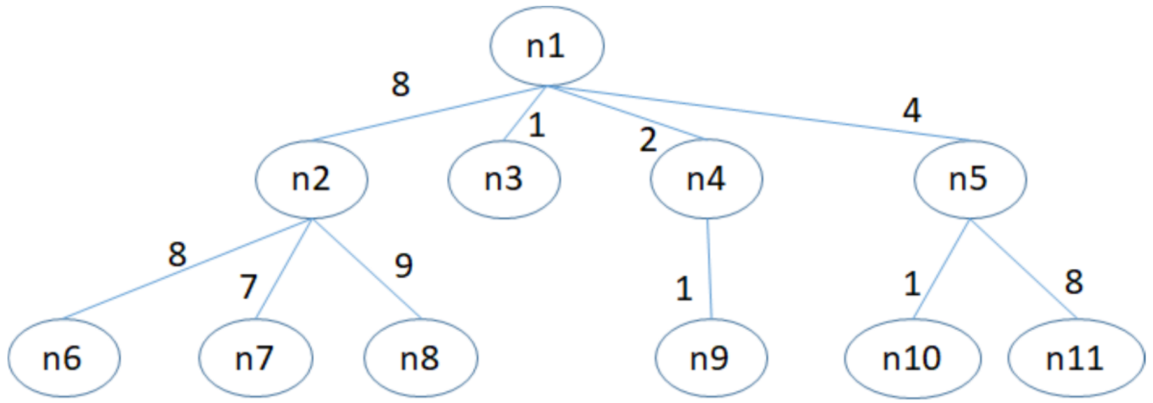
**(C)** Non è possibile stabilirlo perche' manca la euristica o il costo utilizzati.

**(D)** n1,n2,n6,n7,n8,n3.

**(E)** Nessuna delle altre risposte.

# R. Algoritmi di Ricerca Informata

Dato il seguente spazio di stati: determinare tra le seguenti la sequenza generata



dall'algoritmo di *uniform cost* per la ricerca del nodo $n7$, con la funzione $f(n) = g(n)$:

**(A)** n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11 [−1]
**(B)** n1, n2, n6, n7, n8, n4, n9, n5, n10, n11, n3. [−1]
**(C)** Non è possibile stabilirlo perche' è definita la euristica utilizzata. [−1]
**(D)** n1, n3, n4, n9, n5, n10, n2, n11, n7. [+3]
**(E)** Nessuna delle altre risposte. [−1]

## Domande Aperte

- Discutere la **relazione tra la nozione di agente e di ambiente in AI**. Si esplicitino esempi di applicazioni di AI in cui distinguere le due nozioni. Si facciano  esempi di applicazioni che caratterizzano i diversi tipi di ambiente e di agente.

- Data un puzzle da *n* celle (ad es. 9), definire su tale gioco **la nozione di agente e di ambiente; di percezione e di azione; si determini lo spazio degli stati (dell'ambiente). Si determinino le azioni dell'agente i caso di agente semplice dotati di riflessi o di agente a comportamento randomizzato**. Si definisca un insieme di euristiche possibili. Si discuta infine facoltativamente un algoritmo di ricerca euristica per la soluzione del gioco.

# Domande Aperte (2)

- Si discuta la nozione di **algoritmi di Ricerca non Informati** per il Problem Solving ed in particolare si discuta le proprietà di **completezza, e ottimalità** per tali algoritmi. Si confrontino tra loro almeno due algoritmi.

- Si discuta **l'algoritmo di A*** e se ne esemplifichi la applicazione ad **almeno due problemi diversi** (es. labirinto/mappa e gioco dell'8).

- Si confrontino tra loro gli algoritmi di **Breadth-first, Depth-first e di Greedy Best First**: si utilizzi un esempio di applicazione ad un problema (ad es. le 8 regine)

- Si discuta la nozione di **ricerca on-line** con **due esempi di algoritmi** utili alla ricerca di una soluzione possibile.

# Altri esempi di modellazione e domande chiuse:
# Problemi ed Agenti

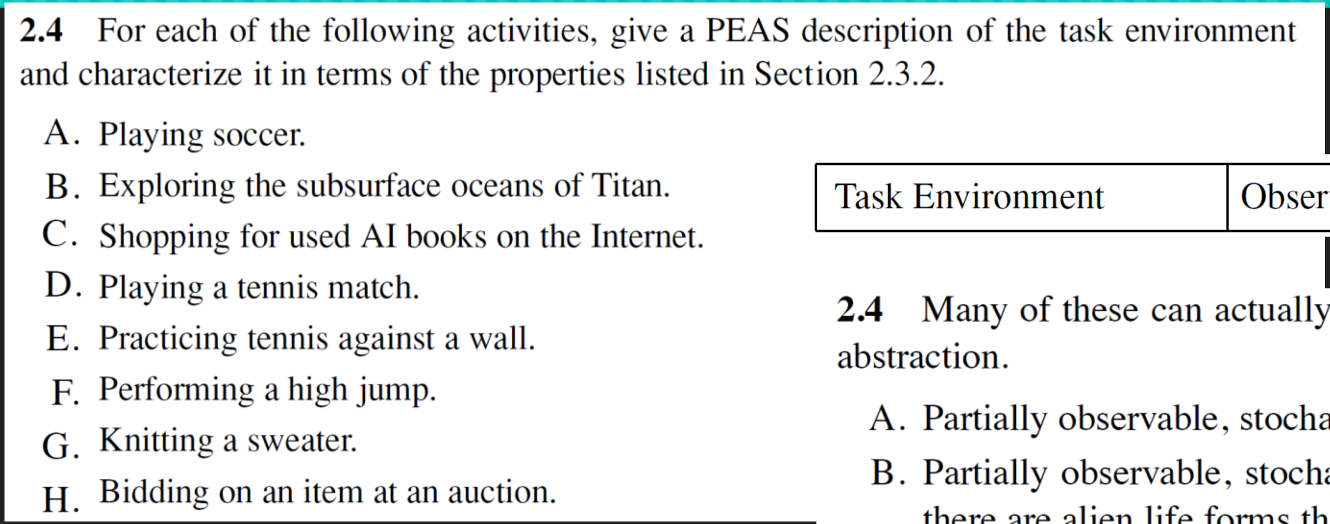
- Riempire e giustificare la seguente tabella

| Agente | Prestazione | Ambiente | Attuatori | Sensori |
|---|---|---|---|---|
| TAXI Driver | Arrivare alla destinazione, sicuro, veloce, ligio alla legge, viaggio confortevole, minimo consumo di benzina, profitti massimi | Strada, altri veicoli, pedoni, clienti | Sterzo, acceleratore, freni, frecce, clacson, schermo di interfaccia o sintesi vocale | Telecamere, sensori a infrarossi e sonar, tachimetro, GPS, contachilometri, acelerometro, sensori sullo stato del motore, tastiera o microfono |
| Giocatore Sudoku | | | | |
| Robot cameriere in una sala ristorante | | | | |
| Motore di ricerca su Facebook (ricerca post e altri utenti) | | | | |
| Speech recognizer (e.g. SIRI) | | | | |

# Agenti: formalizzazione del PEAS model

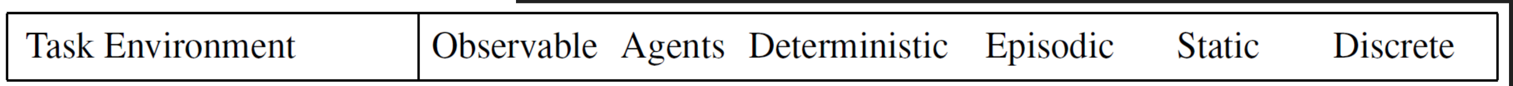
| Agente | Prestazione | Ambiente | Attuatori | Sensori |
|---|---|---|---|---|
| 8-puzzle | Ordinare il puzzle con il minimo tempo t (o numero di mosse, n) | Il puzzle da 81 celle | Spostamento celle | Osservazione dell'intero stato del puzzle |
| 8-puzzle | Dato lo stato *S* di partenza, applica la sequenza di azioni P=($A_1$, $A_2$, …, $A_n$) per raggiungere il Goal *G* in modo che: $n=\min_P (A_n \circ(A_{n-1} \ldots A_2\circ(A_1(S))\ldots)=G$ | Una matrice *M* 3x3 con valori in $m_{ij} \in\{1, \ldots, 8, blank\}$ dove $\forall(i,j)\neq(k,l) \quad m_{ij} \neq m_{kl}$ | Due opzioni: <br>• Muovi un qualsiasi $m_{ij}$ in *blank* (se possible) <br>• Blank-left, …, Blank-down | Tutta la matrice *M* |
| Vacuum World Cleaner con due stanze | | | | |
| Ricerca di un percorso Start-Goal in un a mappa connessa di N città | | | | |
| | | | | |

# Caratterizzazione ambienti di agenti razionali

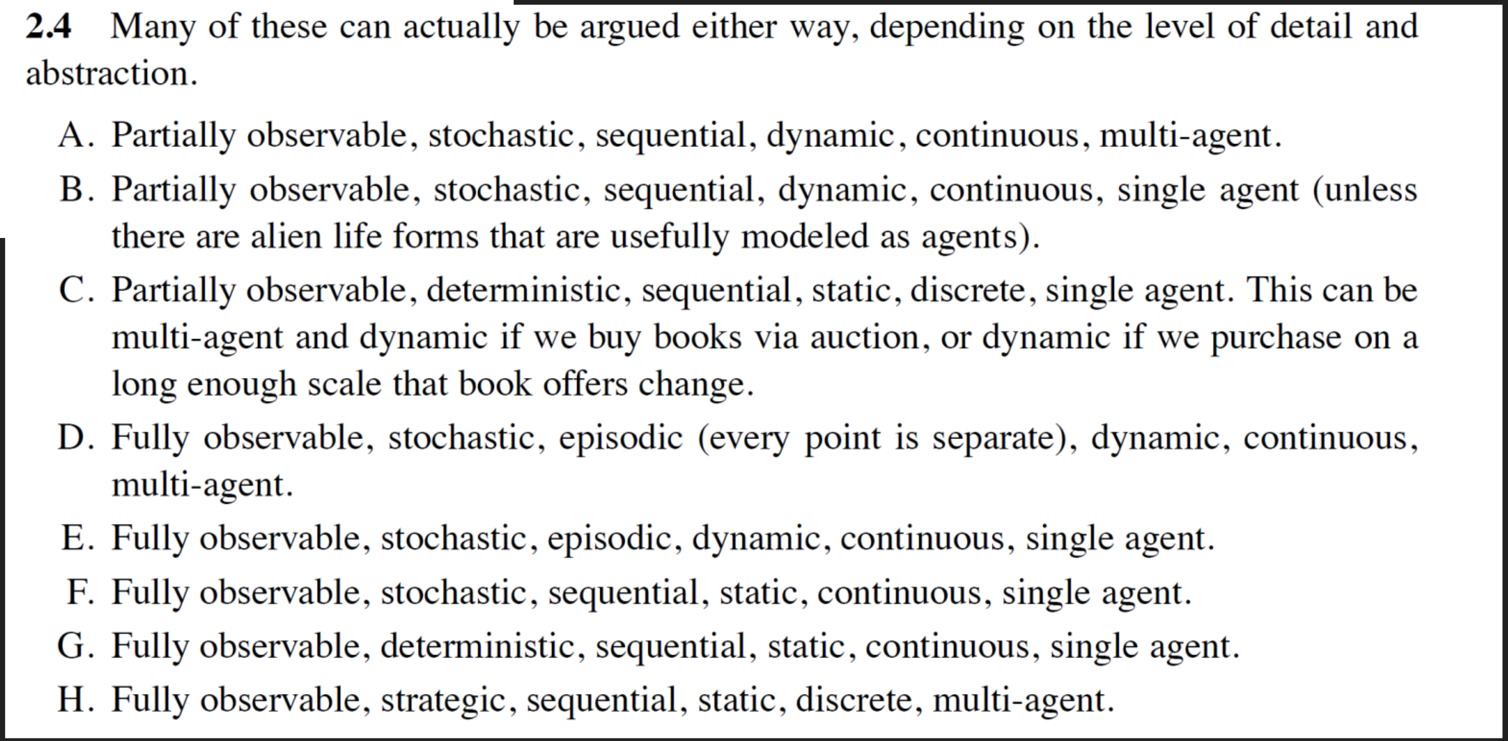


**2.4** For each of the following activities, give a PEAS description of the task environment and characterize it in terms of the properties listed in Section 2.3.2.

A. Playing soccer.
B. Exploring the subsurface oceans of Titan.
C. Shopping for used AI books on the Internet.
D. Playing a tennis match.
E. Practicing tennis against a wall.
F. Performing a high jump.
G. Knitting a sweater.
H. Bidding on an item at an auction.



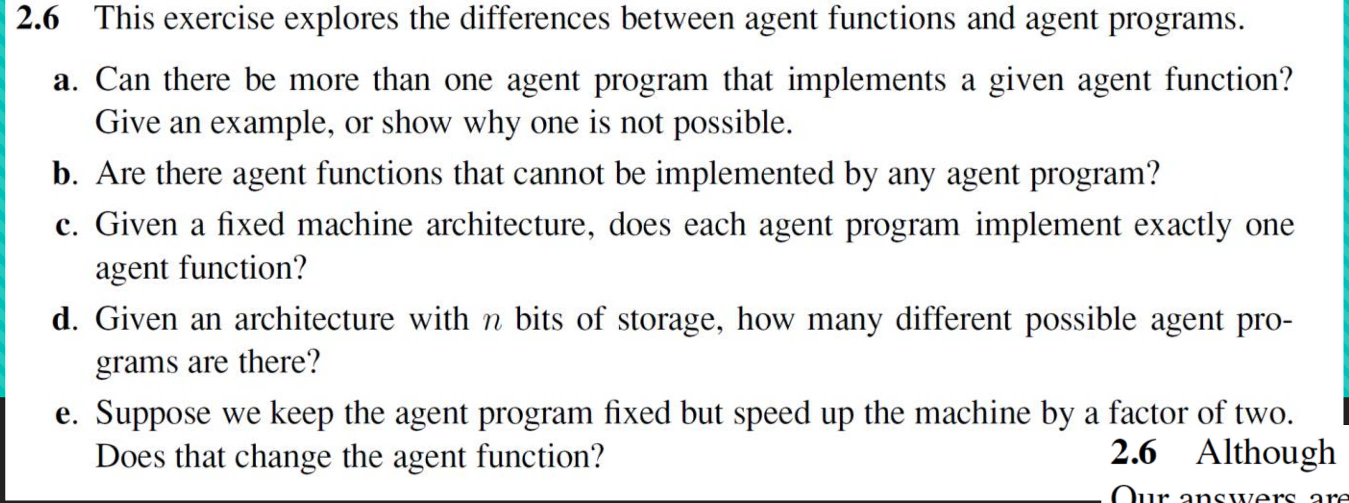
| Task Environment | Observable | Agents | Deterministic | Episodic | Static | Discrete |
|---|---|---|---|---|---|---|



**2.4** Many of these can actually be argued either way, depending on the level of detail and abstraction.

A. Partially observable, stochastic, sequential, dynamic, continuous, multi-agent.
B. Partially observable, stochastic, sequential, dynamic, continuous, single agent (unless there are alien life forms that are usefully modeled as agents).
C. Partially observable, deterministic, sequential, static, discrete, single agent. This can be multi-agent and dynamic if we buy books via auction, or dynamic if we purchase on a long enough scale that book offers change.
D. Fully observable, stochastic, episodic (every point is separate), dynamic, continuous, multi-agent.
E. Fully observable, stochastic, episodic, dynamic, continuous, single agent.
F. Fully observable, stochastic, sequential, static, continuous, single agent.
G. Fully observable, deterministic, sequential, static, continuous, single agent.
H. Fully observable, strategic, sequential, static, discrete, multi-agent.

## Agenti e complessità



**2.6** This exercise explores the differences between agent functions and agent programs.

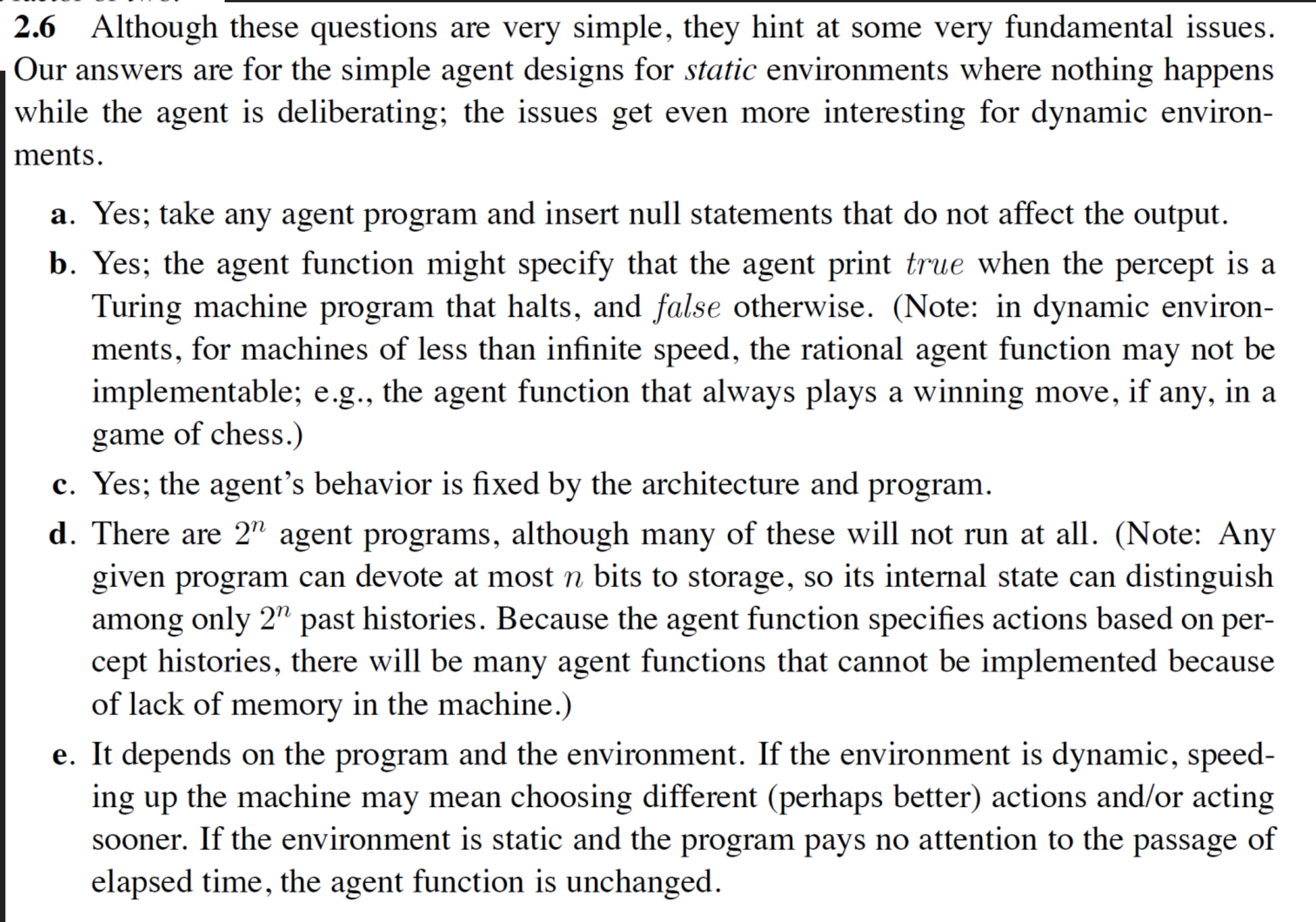
**a.** Can there be more than one agent program that implements a given agent function? Give an example, or show why one is not possible.

**b.** Are there agent functions that cannot be implemented by any agent program?

**c.** Given a fixed machine architecture, does each agent program implement exactly one agent function?

**d.** Given an architecture with $n$ bits of storage, how many different possible agent programs are there?

**e.** Suppose we keep the agent program fixed but speed up the machine by a factor of two. Does that change the agent function?



**2.6** Although these questions are very simple, they hint at some very fundamental issues. Our answers are for the simple agent designs for *static* environments where nothing happens while the agent is deliberating; the issues get even more interesting for dynamic environments.

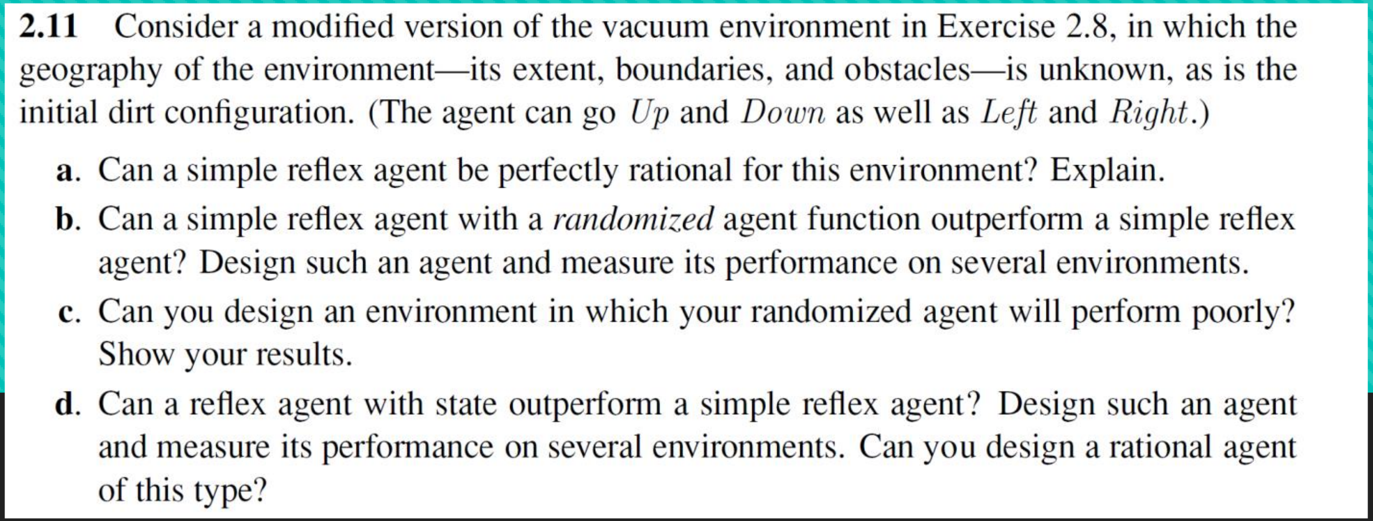
**a.** Yes; take any agent program and insert null statements that do not affect the output.

**b.** Yes; the agent function might specify that the agent print $true$ when the percept is a Turing machine program that halts, and $false$ otherwise. (Note: in dynamic environments, for machines of less than infinite speed, the rational agent function may not be implementable; e.g., the agent function that always plays a winning move, if any, in a game of chess.)

**c.** Yes; the agent's behavior is fixed by the architecture and program.

**d.** There are $2^n$ agent programs, although many of these will not run at all. (Note: Any given program can devote at most $n$ bits to storage, so its internal state can distinguish among only $2^n$ past histories. Because the agent function specifies actions based on percept histories, there will be many agent functions that cannot be implemented because of lack of memory in the machine.)

**e.** It depends on the program and the environment. If the environment is dynamic, speeding up the machine may mean choosing different (perhaps better) actions and/or acting sooner. If the environment is static and the program pays no attention to the passage of elapsed time, the agent function is unchanged.



**2.11** Consider a modified version of the vacuum environment in Exercise 2.8, in which the geography of the environment—its extent, boundaries, and obstacles—is unknown, as is the initial dirt configuration. (The agent can go *Up* and *Down* as well as *Left* and *Right*.)

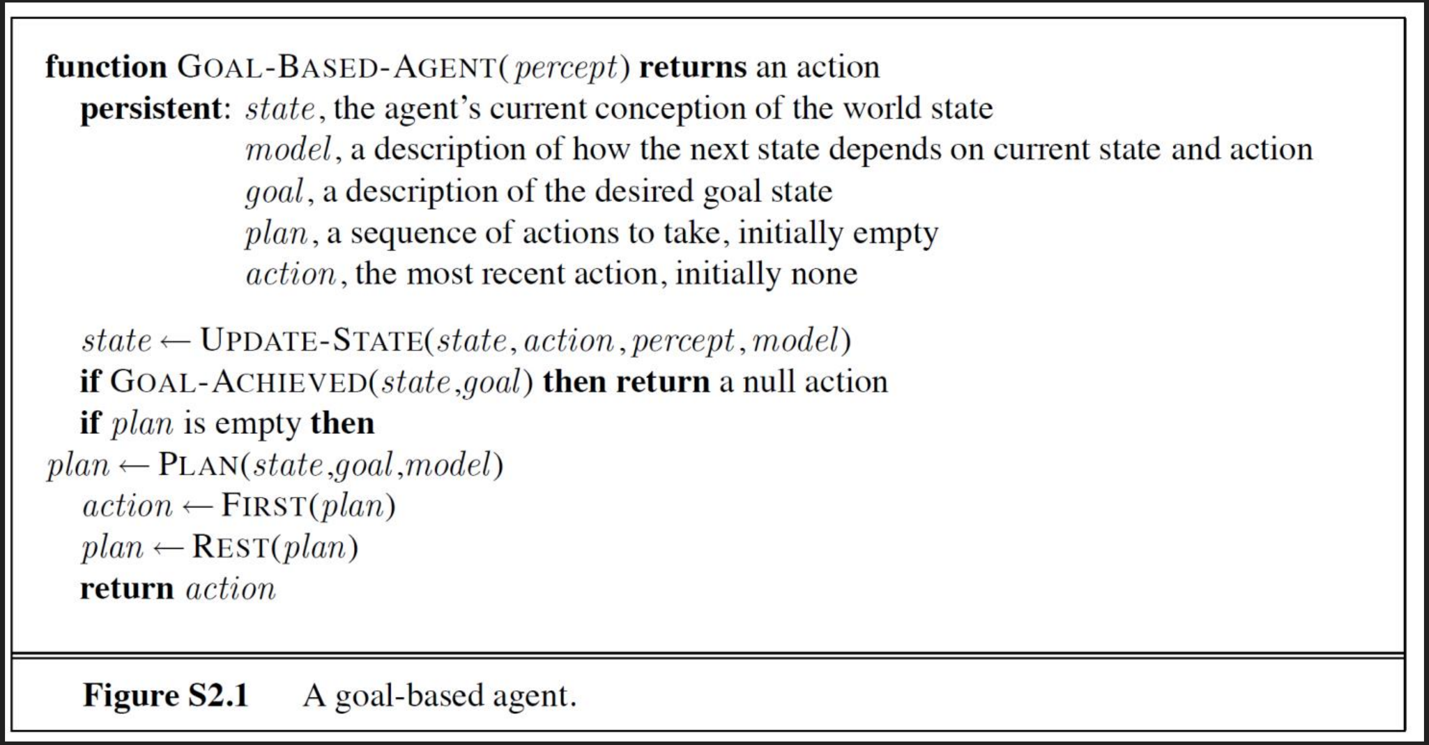
**a.** Can a simple reflex agent be perfectly rational for this environment? Explain.

**b.** Can a simple reflex agent with a *randomized* agent function outperform a simple reflex agent? Design such an agent and measure its performance on several environments.

**c.** Can you design an environment in which your randomized agent will perform poorly? Show your results.

**d.** Can a reflex agent with state outperform a simple reflex agent? Design such an agent and measure its performance on several environments. Can you design a rational agent of this type?

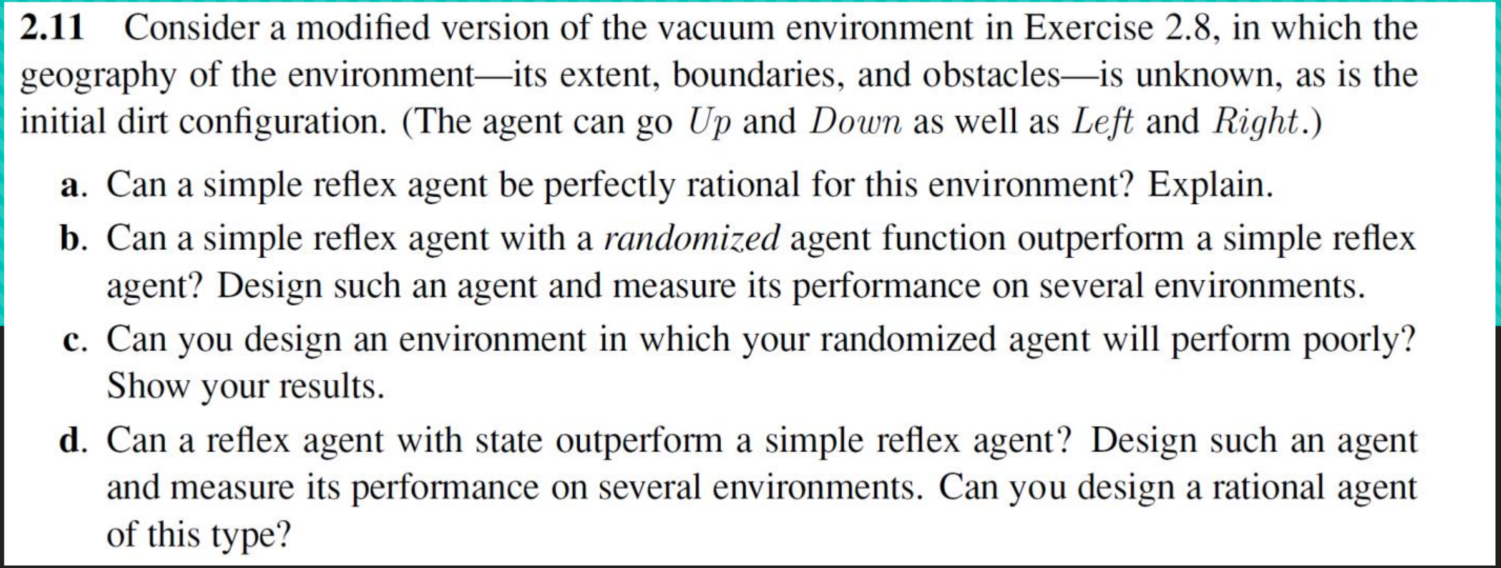
## Vacuum Cleaner



```
function GOAL-BASED-AGENT(percept) returns an action
   persistent: state, the agent's current conception of the world state
               model, a description of how the next state depends on current state and action
               goal, a description of the desired goal state
               plan, a sequence of actions to take, initially empty
               action, the most recent action, initially none

   state ← UPDATE-STATE(state, action, percept, model)
   if GOAL-ACHIEVED(state, goal) then return a null action
   if plan is empty then
plan ← PLAN(state, goal, model)
   action ← FIRST(plan)
   plan ← REST(plan)
   return action
```

**Figure S2.1** A goal-based agent.

# Vacuum Cleaner



**2.11** Consider a modified version of the vacuum environment in Exercise 2.8, in which the geography of the environment—its extent, boundaries, and obstacles—is unknown, as is the initial dirt configuration. (The agent can go *Up* and *Down* as well as *Left* and *Right*.)

**a.** Can a simple reflex agent be perfectly rational for this environment? Explain.

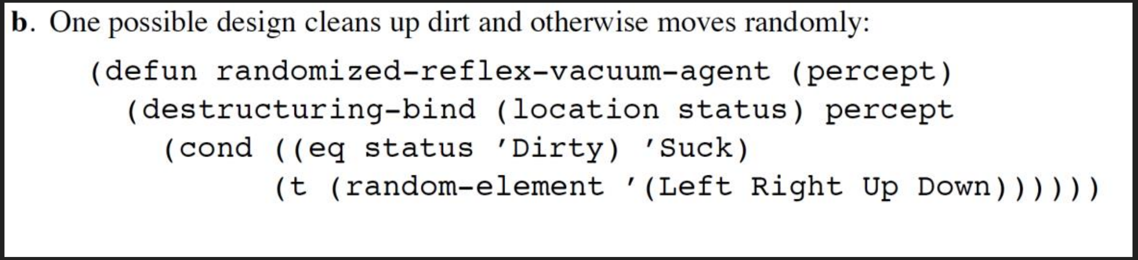
**b.** Can a simple reflex agent with a *randomized* agent function outperform a simple reflex agent? Design such an agent and measure its performance on several environments.

**c.** Can you design an environment in which your randomized agent will perform poorly? Show your results.

**d.** Can a reflex agent with state outperform a simple reflex agent? Design such an agent and measure its performance on several environments. Can you design a rational agent of this type?

- (a) and (b)

- **a.** No. Geography matters, and no rational behaviour is possible in  blind situation.



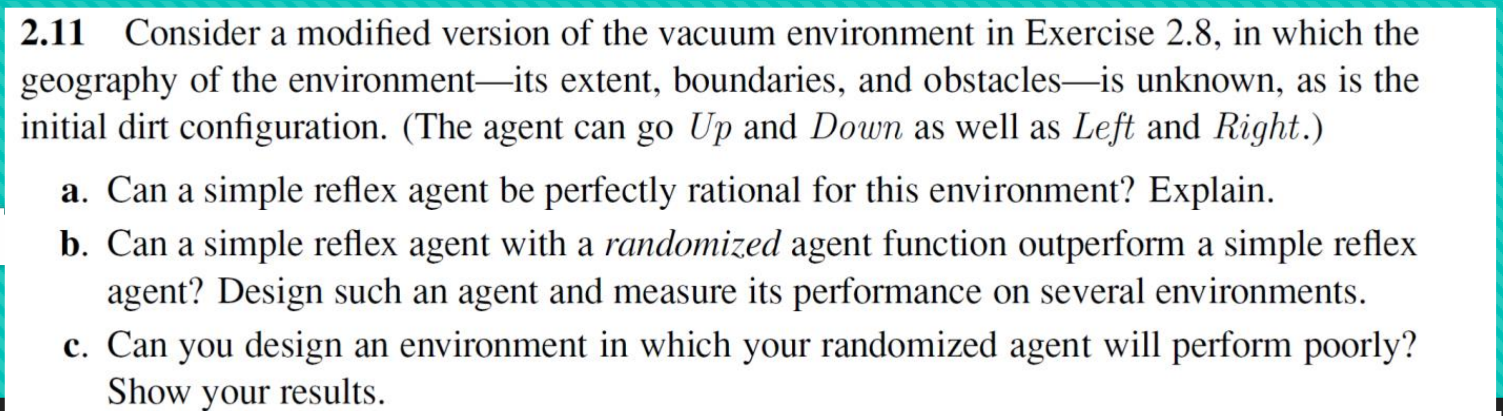
**b**. One possible design cleans up dirt and otherwise moves randomly:

```
(defun randomized-reflex-vacuum-agent (percept)
  (destructuring-bind (location status) percept
    (cond ((eq status 'Dirty) 'Suck)
          (t (random-element '(Left Right Up Down))))))
```

```
Function randomized_reflex_vacuum(percept)
 (location, status)=percept
 if( status=‘Dirty’)
   return(‘Suck’)
 else
   return(rand_choice([‘Left’, ‘Right’, ‘Up’, ‘Down’]))
```

# Vacuum Cleane

- A complex environment requested in c.



**2.11** Consider a modified version of the vacuum environment in Exercise 2.8, in which the geography of the environment—its extent, boundaries, and obstacles—is unknown, as is the initial dirt configuration. (The agent can go *Up* and *Down* as well as *Left* and *Right*.)

**a.** Can a simple reflex agent be perfectly rational for this environment? Explain.

**b.** Can a simple reflex agent with a *randomized* agent function outperform a simple reflex agent? Design such an agent and measure its performance on several environments.

**c.** Can you design an environment in which your randomized agent will perform poorly? Show your results.

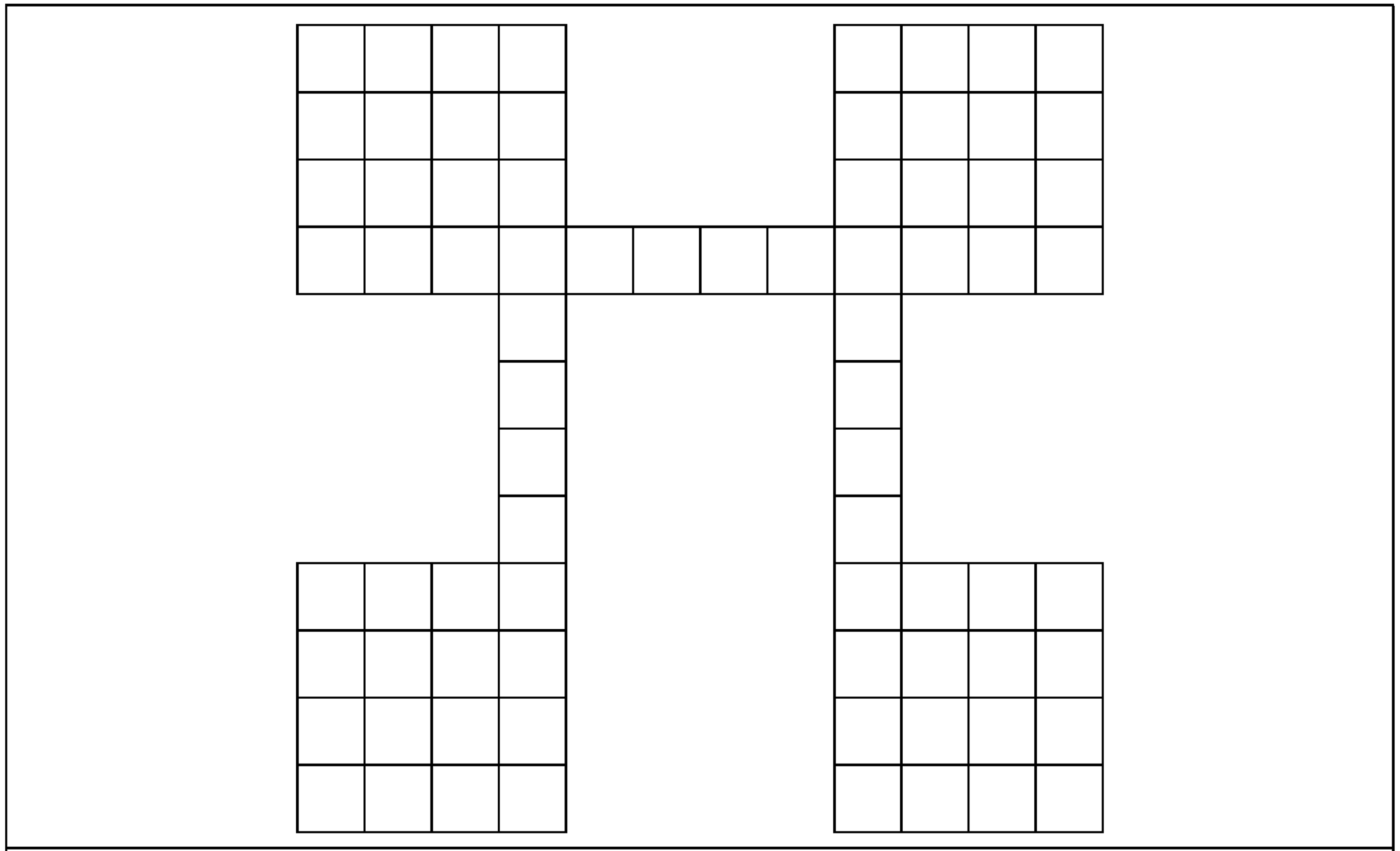
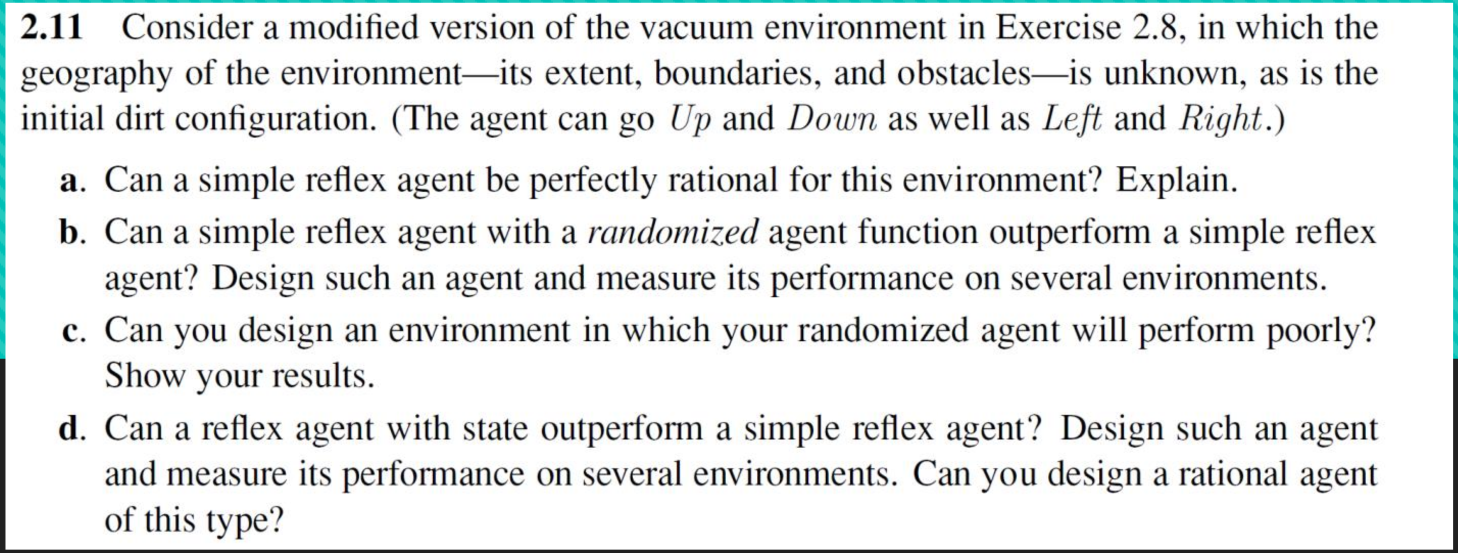


**Figure S2.3** An environment in which random motion will take a long time to cover all the squares.

## Vacuum Cleaner



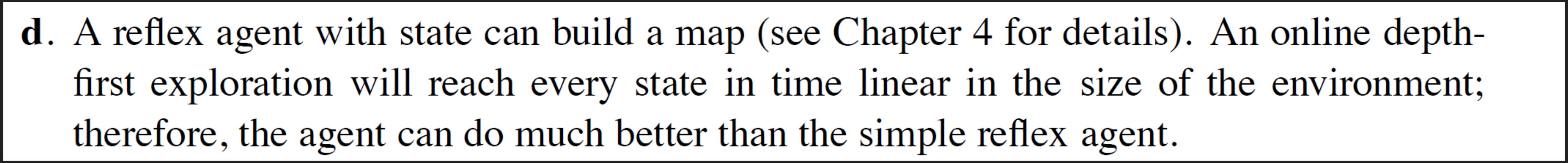
**2.11** Consider a modified version of the vacuum environment in Exercise 2.8, in which the geography of the environment—its extent, boundaries, and obstacles—is unknown, as is the initial dirt configuration. (The agent can go *Up* and *Down* as well as *Left* and *Right*.)

**a.** Can a simple reflex agent be perfectly rational for this environment? Explain.

**b.** Can a simple reflex agent with a *randomized* agent function outperform a simple reflex agent? Design such an agent and measure its performance on several environments.

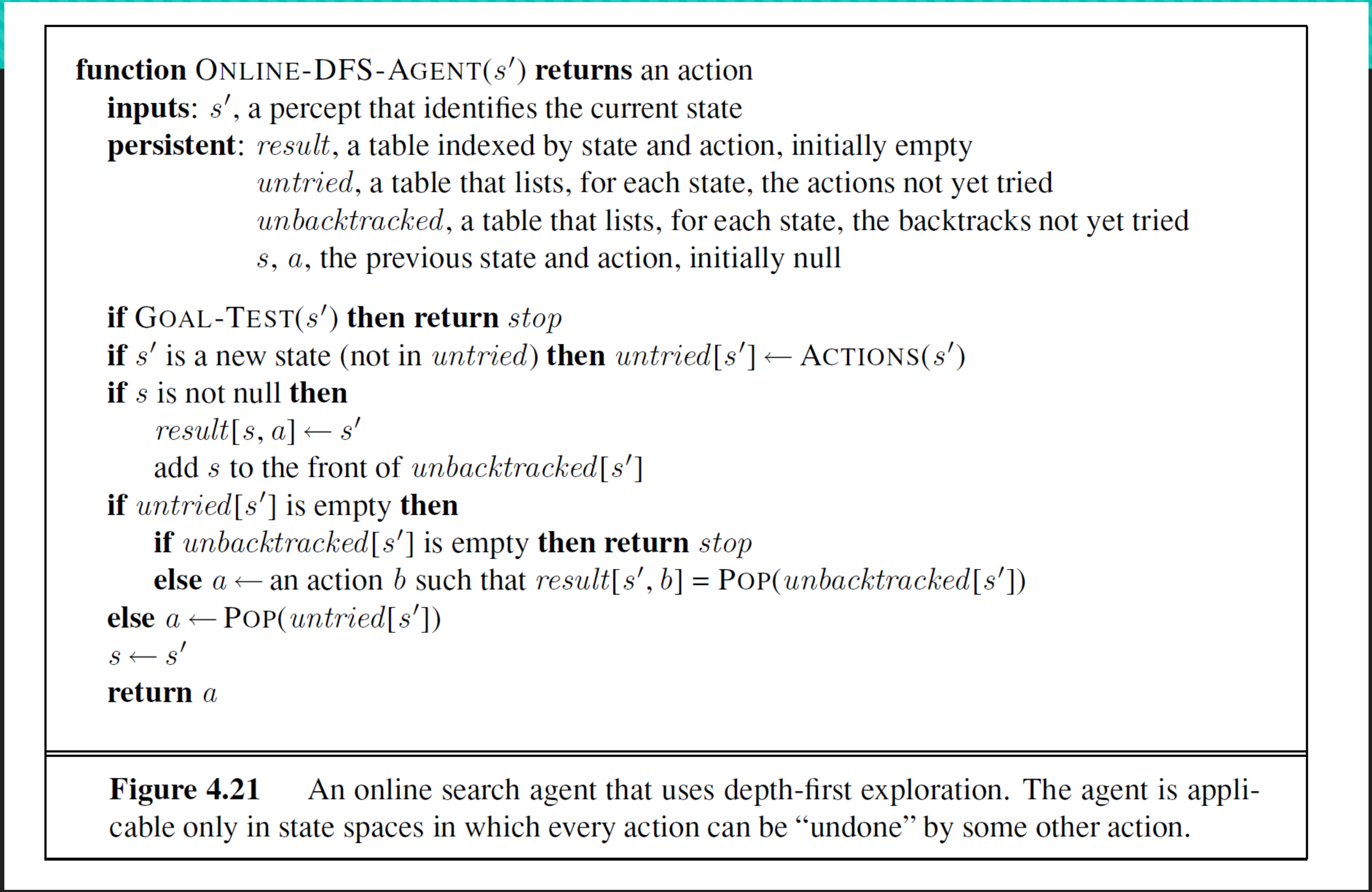
**c.** Can you design an environment in which your randomized agent will perform poorly? Show your results.

**d.** Can a reflex agent with state outperform a simple reflex agent? Design such an agent and measure its performance on several environments. Can you design a rational agent of this type?



**d.** A reflex agent with state can build a map (see Chapter 4 for details). An online depth-first exploration will reach every state in time linear in the size of the environment; therefore, the agent can do much better than the simple reflex agent.

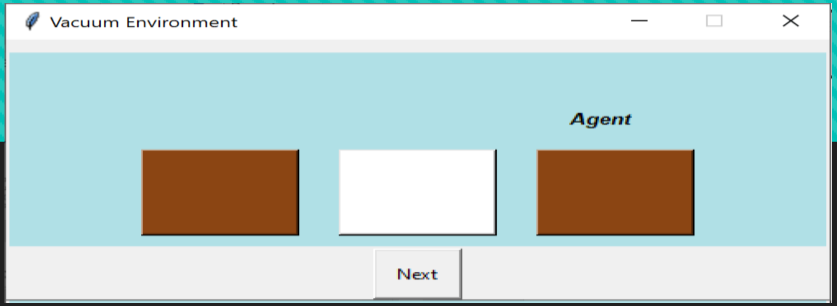
# Vacuum Cleaner: DFS

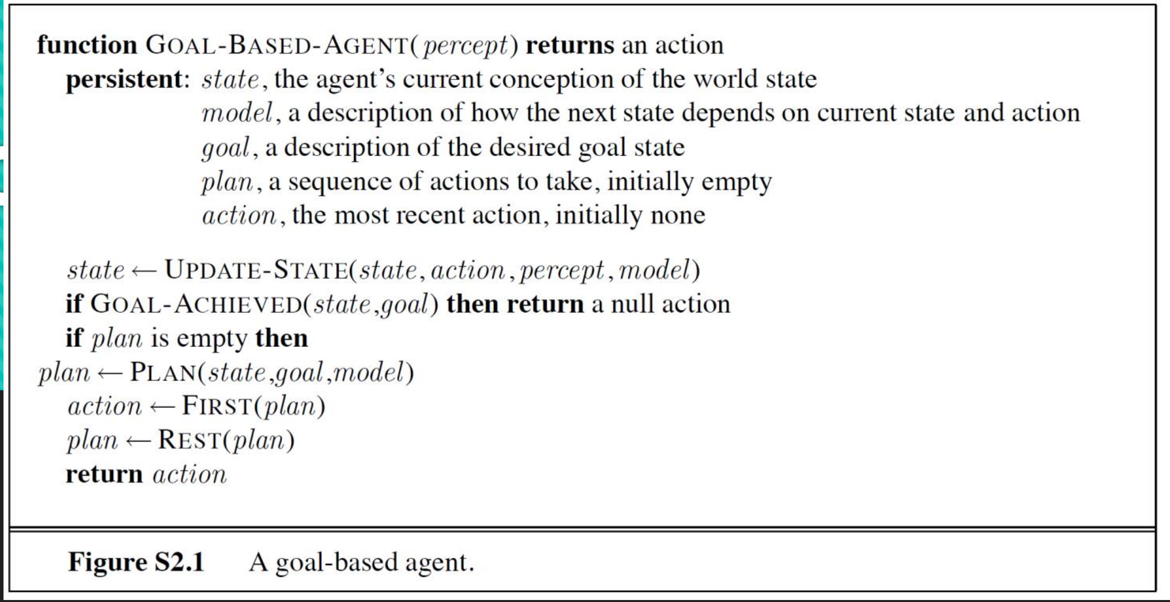


```
function ONLINE-DFS-AGENT($s'$) returns an action
  inputs: $s'$, a percept that identifies the current state
  persistent: $result$, a table indexed by state and action, initially empty
              $untried$, a table that lists, for each state, the actions not yet tried
              $unbacktracked$, a table that lists, for each 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 (not in $untried$) then $untried[s'] \leftarrow$ 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 $stop$
      else $a \leftarrow$ an action $b$ such that $result[s', b]$ = POP($unbacktracked[s']$)
  else $a \leftarrow$ POP($untried[s']$)
  $s \leftarrow s'$
  return $a$
```

**Figure 4.21** An online search agent that uses depth-first exploration. The agent is applicable only in state spaces in which every action can be "undone" by some other action.

# Vacuum Cleaner: eserc





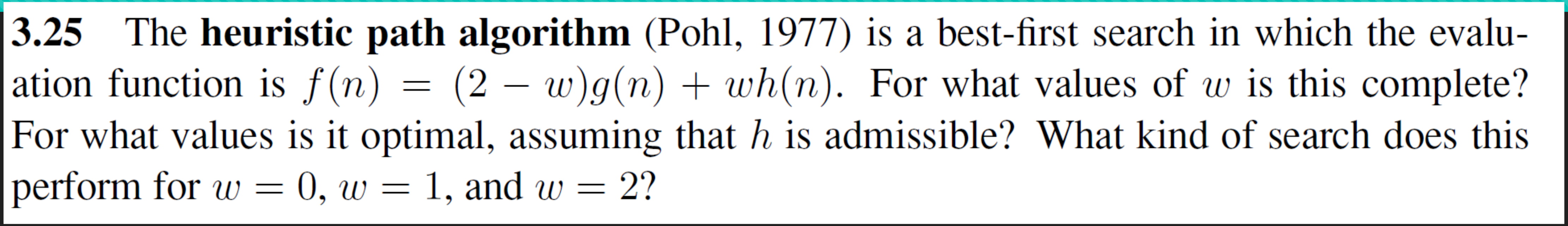
```
function GOAL-BASED-AGENT(percept) returns an action
   persistent: state, the agent's current conception of the world state
               model, a description of how the next state depends on current state and action
               goal, a description of the desired goal state
               plan, a sequence of actions to take, initially empty
               action, the most recent action, initially none

   state ← UPDATE-STATE(state, action, percept, model)
   if GOAL-ACHIEVED(state,goal) then return a null action
   if plan is empty then
plan ← PLAN(state,goal,model)
   action ← FIRST(plan)
   plan ← REST(plan)
   return action
```

**Figure S2.1** A goal-based agent.

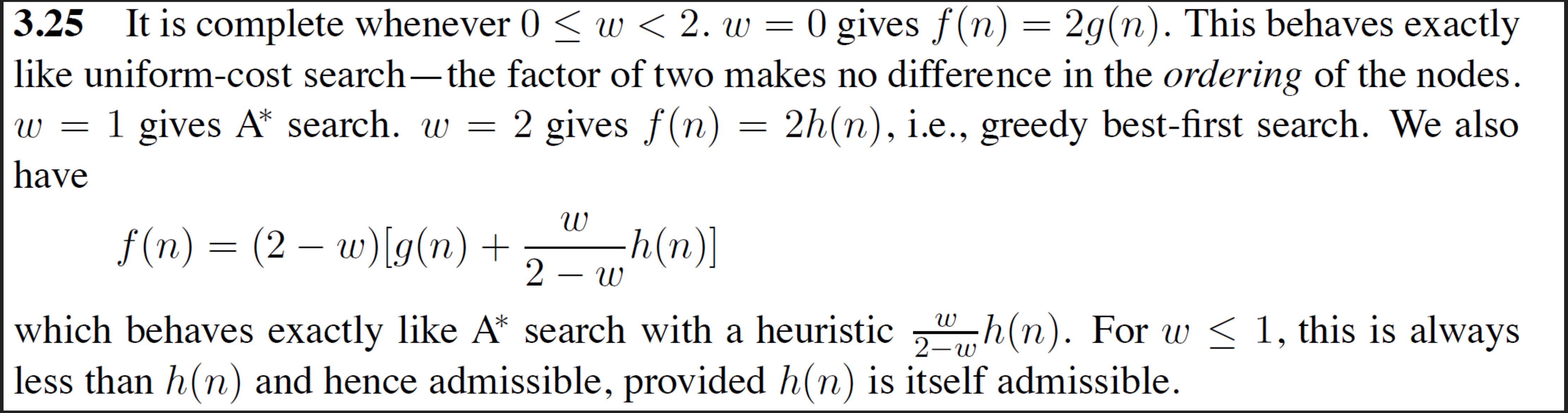
- Adattare il programma tabellare dell'agente alla situazione in cui:
  - Esistono 3 stanze (non due sole)
    - Monodirezionale ma lungo tre e non più due stanze
    - Inizializzazione randomica dello stato delle stanze
  - L'agente ha una percezione «completa» dell'ambiente, cioè l'intero insieme di stanze
  - L'agente è GoalBased e quindi può
    - Elaborare un piano rispetto alla situazione iniziale
    - Eseguire il piano ad ogni passo
  - Si confrontino il comportamento di
    - Breadth-first Search
    - A* (basati su una euristica, ad es. numero di stanze sporche)

## Ulteriori domande dal libro AIMA

## Exercise 3.25



**3.25** The **heuristic path algorithm** (Pohl, 1977) is a best-first search in which the evaluation function is $f(n) = (2 - w)g(n) + wh(n)$. For what values of $w$ is this complete? For what values is it optimal, assuming that $h$ is admissible? What kind of search does this perform for $w = 0$, $w = 1$, and $w = 2$?

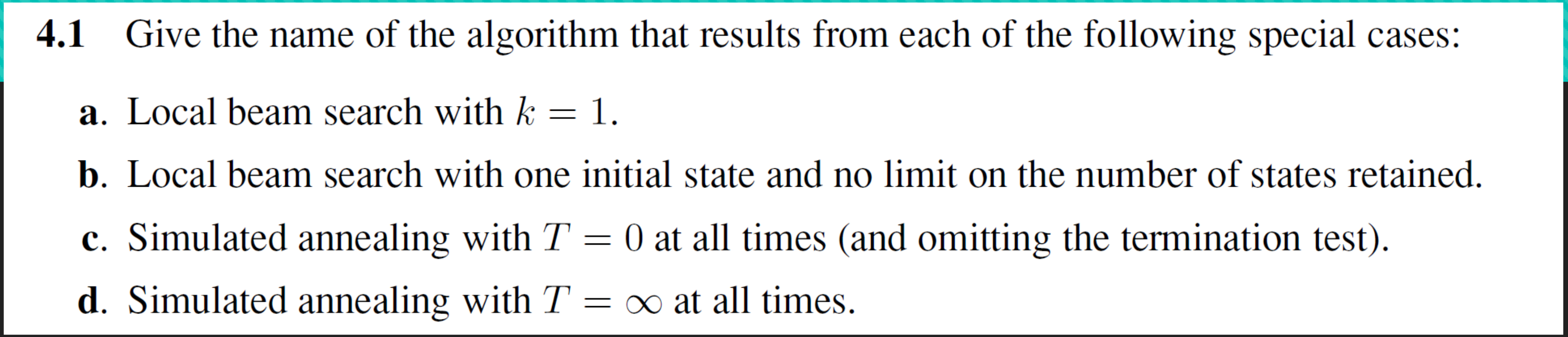


**3.25** It is complete whenever $0 \leq w < 2$. $w = 0$ gives $f(n) = 2g(n)$. This behaves exactly like uniform-cost search—the factor of two makes no difference in the *ordering* of the nodes. $w = 1$ gives A* search. $w = 2$ gives $f(n) = 2h(n)$, i.e., greedy best-first search. We also have

$$f(n) = (2 - w)[g(n) + \frac{w}{2 - w}h(n)]$$

which behaves exactly like A* search with a heuristic $\frac{w}{2-w}h(n)$. For $w \leq 1$, this is always less than $h(n)$ and hence admissible, provided $h(n)$ is itself admissible.

## Exercise 4.1



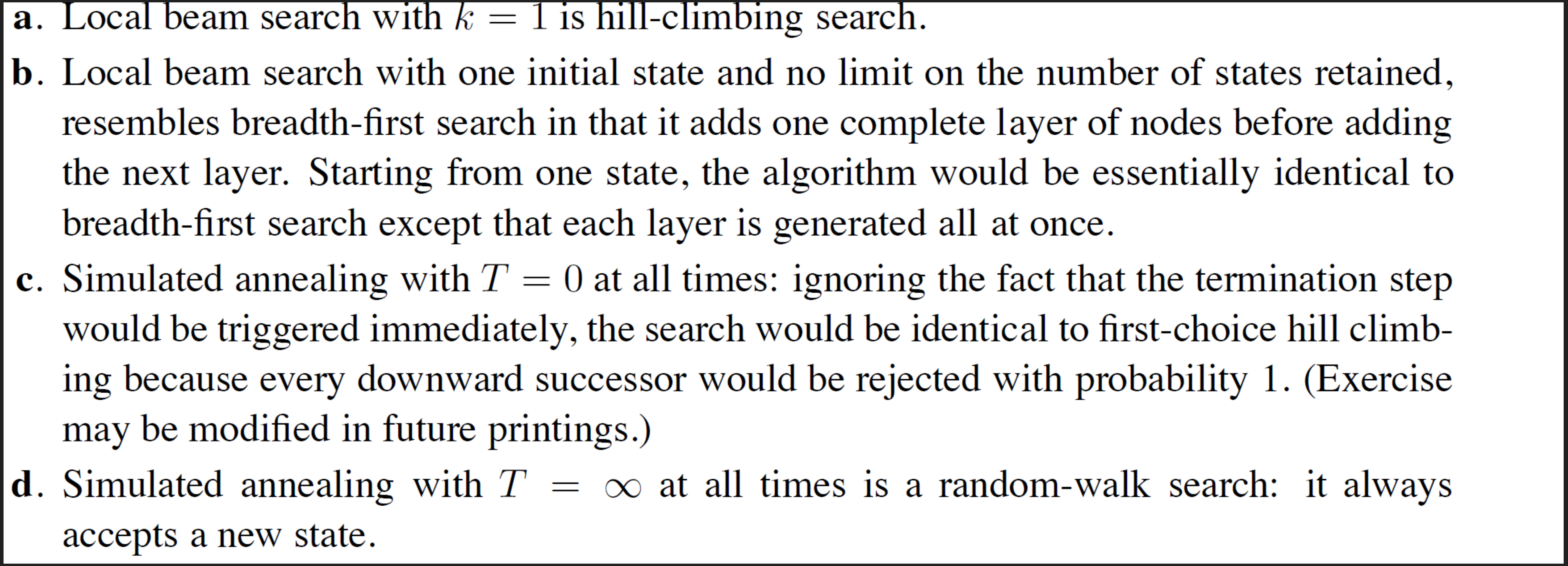
**4.1** Give the name of the algorithm that results from each of the following special cases:

**a.** Local beam search with $k = 1$.

**b.** Local beam search with one initial state and no limit on the number of states retained.

**c.** Simulated annealing with $T = 0$ at all times (and omitting the termination test).

**d.** Simulated annealing with $T = \infty$ at all times.



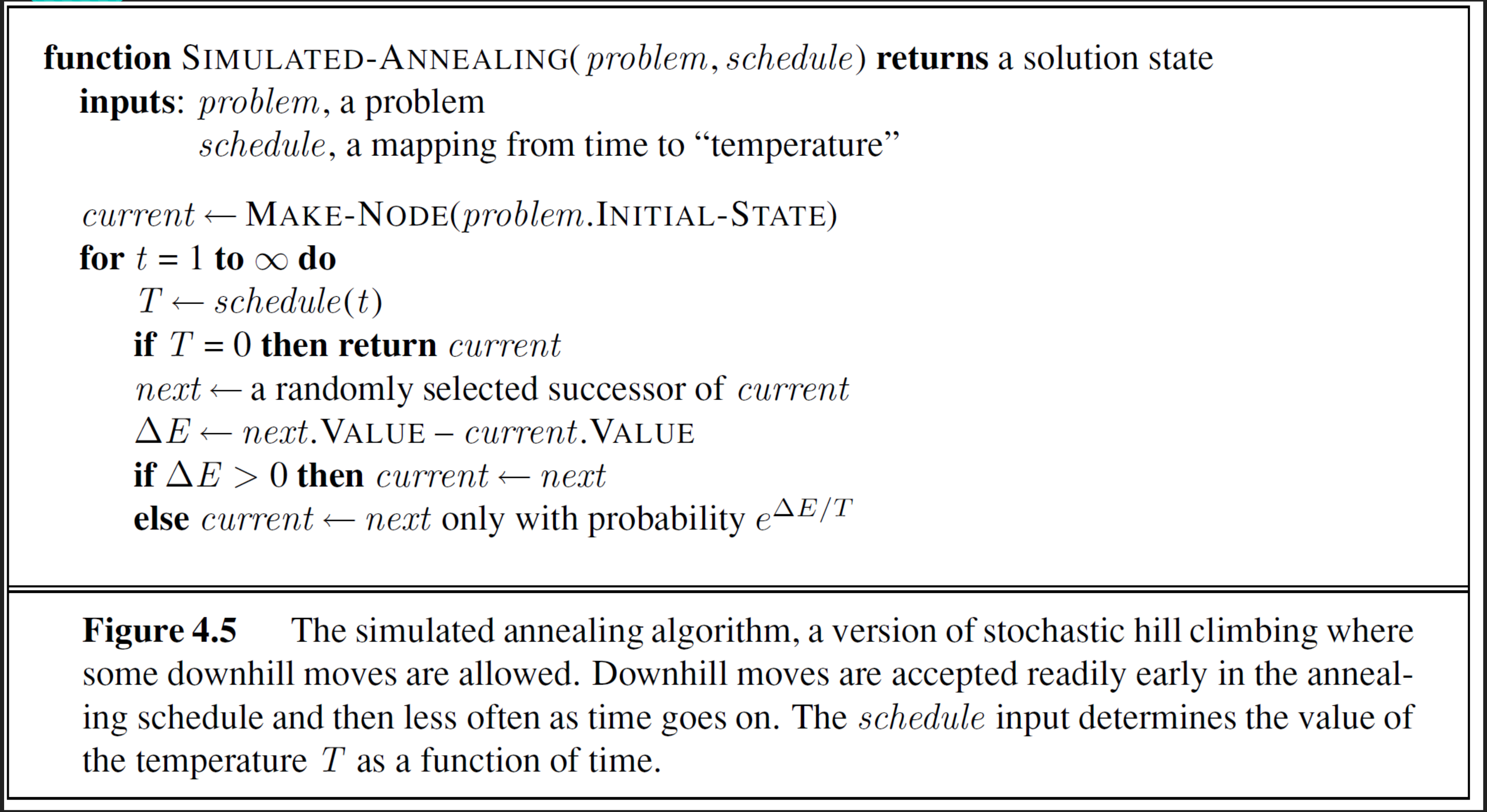
**a.** Local beam search with $k = 1$ is hill-climbing search.

**b.** Local beam search with one initial state and no limit on the number of states retained, resembles breadth-first search in that it adds one complete layer of nodes before adding the next layer. Starting from one state, the algorithm would be essentially identical to breadth-first search except that each layer is generated all at once.

**c.** Simulated annealing with $T = 0$ at all times: ignoring the fact that the termination step would be triggered immediately, the search would be identical to first-choice hill climbing because every downward successor would be rejected with probability 1. (Exercise may be modified in future printings.)

**d.** Simulated annealing with $T = \infty$ at all times is a random-walk search: it always accepts a new state.

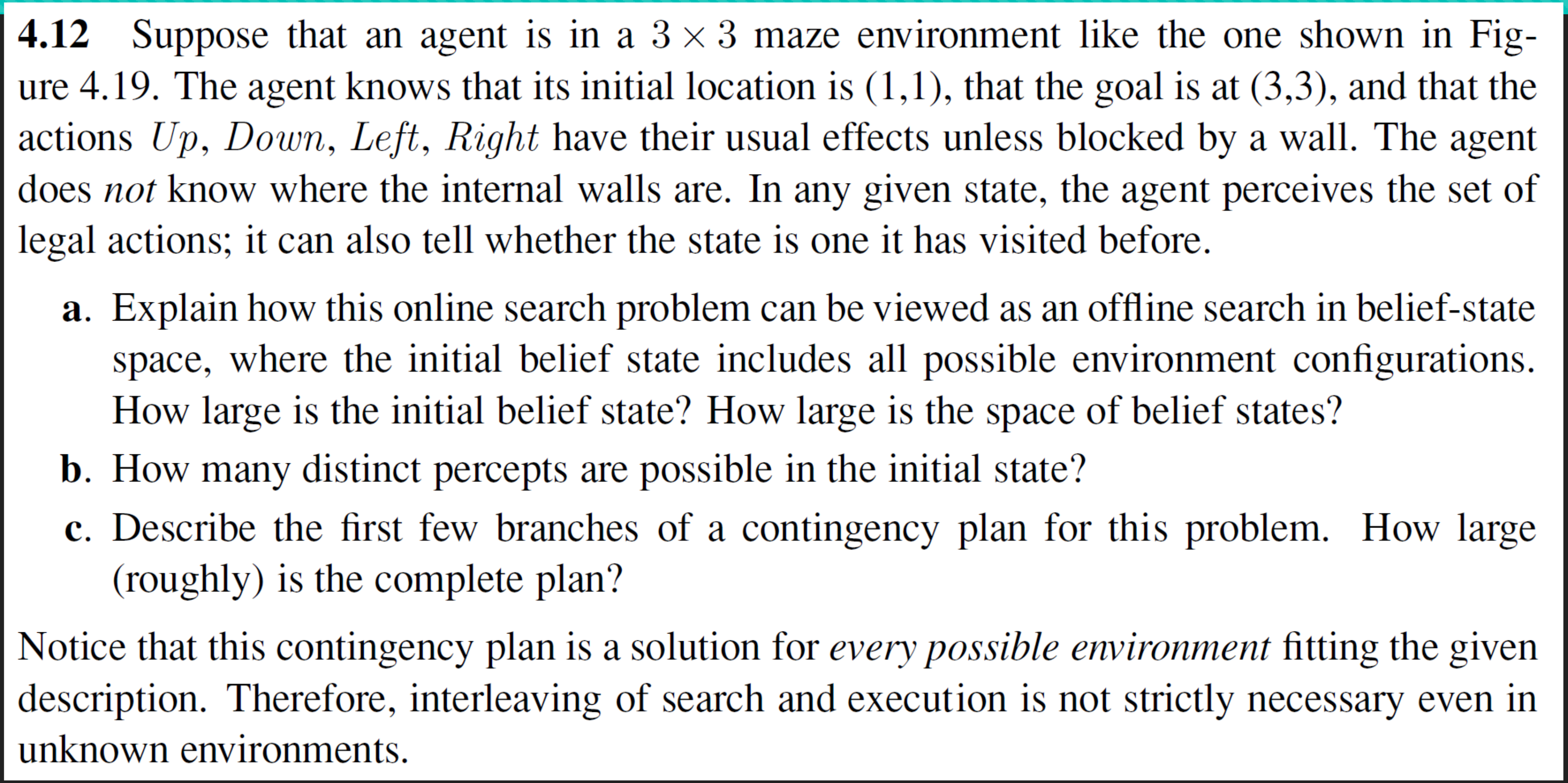
# Exercise 4.1: recap Simulated Annealing



```
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 $t$ = 1 to $\infty$ do
      $T \leftarrow schedule(t)$
      if $T = 0$ then return current
      next ← a randomly selected successor of current
      $\Delta E \leftarrow next.\text{VALUE} - current.\text{VALUE}$
      if $\Delta E > 0$ then current ← next
      else current ← next only with probability $e^{\Delta E/T}$
```

**Figure 4.5** The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of the temperature $T$ as a function of time.

## Exercise 4.12



**4.12** Suppose that an agent is in a $3 \times 3$ maze environment like the one shown in Figure 4.19. The agent knows that its initial location is (1,1), that the goal is at (3,3), and that the actions $Up$, $Down$, $Left$, $Right$ have their usual effects unless blocked by a wall. The agent does *not* know where the internal walls are. In any given state, the agent perceives the set of legal actions; it can also tell whether the state is one it has visited before.

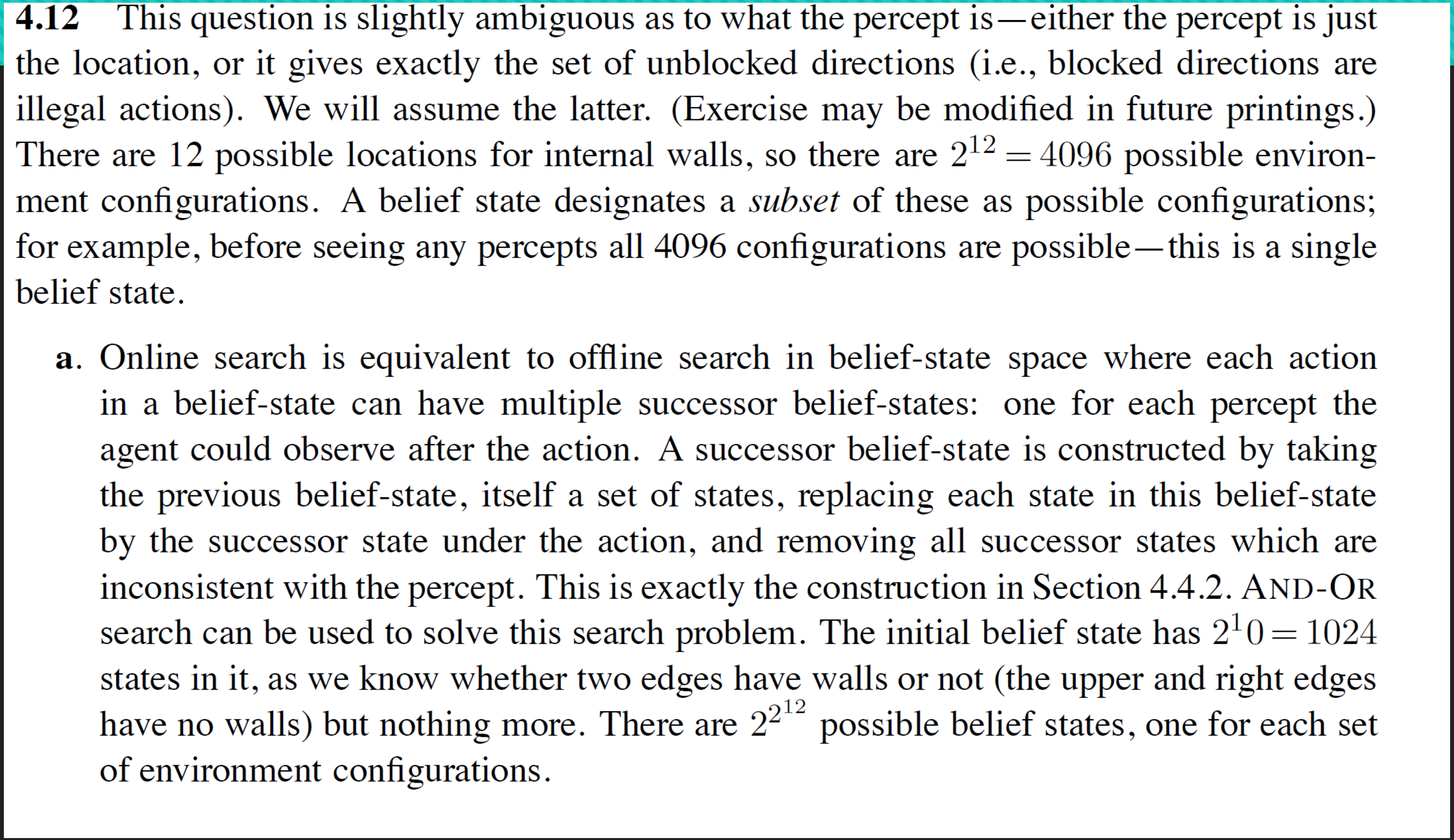
**a**. Explain how this online search problem can be viewed as an offline search in belief-state space, where the initial belief state includes all possible environment configurations. How large is the initial belief state? How large is the space of belief states?

**b**. How many distinct percepts are possible in the initial state?

**c**. Describe the first few branches of a contingency plan for this problem. How large (roughly) is the complete plan?

Notice that this contingency plan is a solution for *every possible environment* fitting the given description. Therefore, interleaving of search and execution is not strictly necessary even in unknown environments.

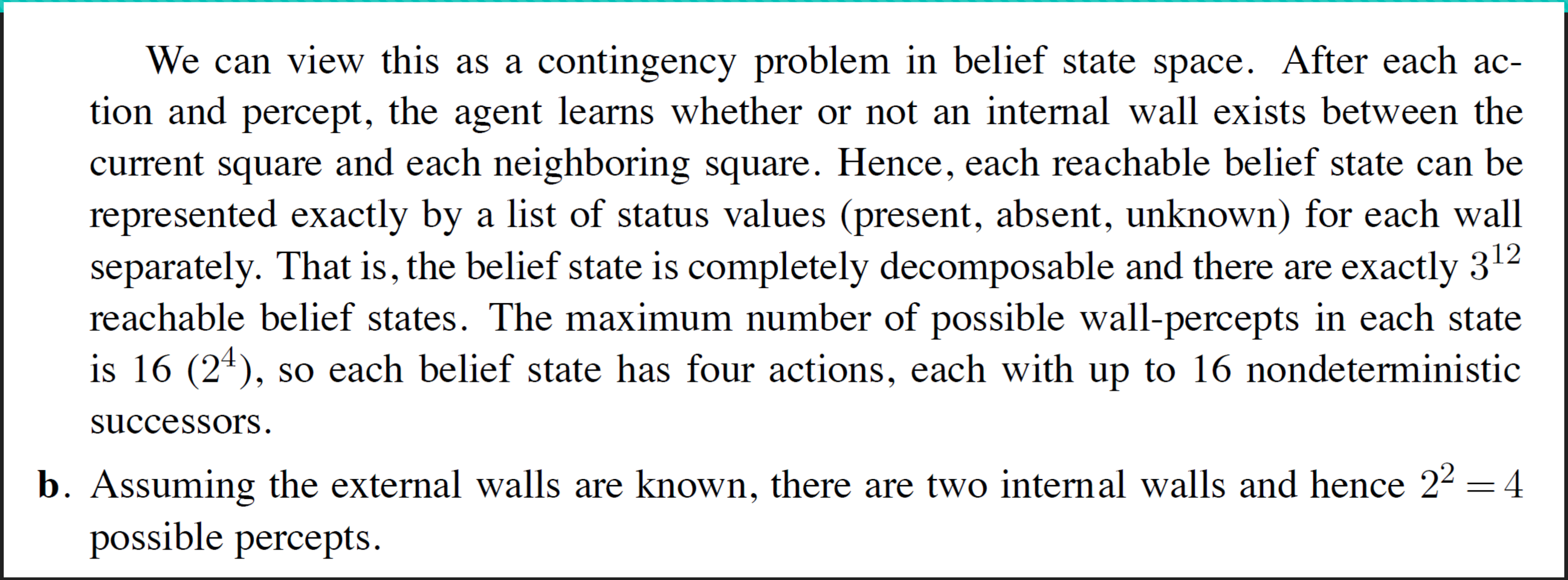
# Exercise 4.12 (a …)



**4.12** This question is slightly ambiguous as to what the percept is—either the percept is just the location, or it gives exactly the set of unblocked directions (i.e., blocked directions are illegal actions). We will assume the latter. (Exercise may be modified in future printings.) There are 12 possible locations for internal walls, so there are $2^{12} = 4096$ possible environment configurations. A belief state designates a *subset* of these as possible configurations; for example, before seeing any percepts all 4096 configurations are possible—this is a single belief state.

**a**. Online search is equivalent to offline search in belief-state space where each action in a belief-state can have multiple successor belief-states: one for each percept the agent could observe after the action. A successor belief-state is constructed by taking the previous belief-state, itself a set of states, replacing each state in this belief-state by the successor state under the action, and removing all successor states which are inconsistent with the percept. This is exactly the construction in Section 4.4.2. AND-OR search can be used to solve this search problem. The initial belief state has $2^1 0 = 1024$ states in it, as we know whether two edges have walls or not (the upper and right edges have no walls) but nothing more. There are $2^{2^{12}}$ possible belief states, one for each set of environment configurations.

## Exercise 4.12 (… a - b)



We can view this as a contingency problem in belief state space. After each action and percept, the agent learns whether or not an internal wall exists between the current square and each neighboring square. Hence, each reachable belief state can be represented exactly by a list of status values (present, absent, unknown) for each wall separately. That is, the belief state is completely decomposable and there are exactly $3^{12}$ reachable belief states. The maximum number of possible wall-percepts in each state is 16 ($2^4$), so each belief state has four actions, each with up to 16 nondeterministic successors.

**b**. Assuming the external walls are known, there are two internal walls and hence $2^2 = 4$ possible percepts.

# Exercise 4.12 (c)

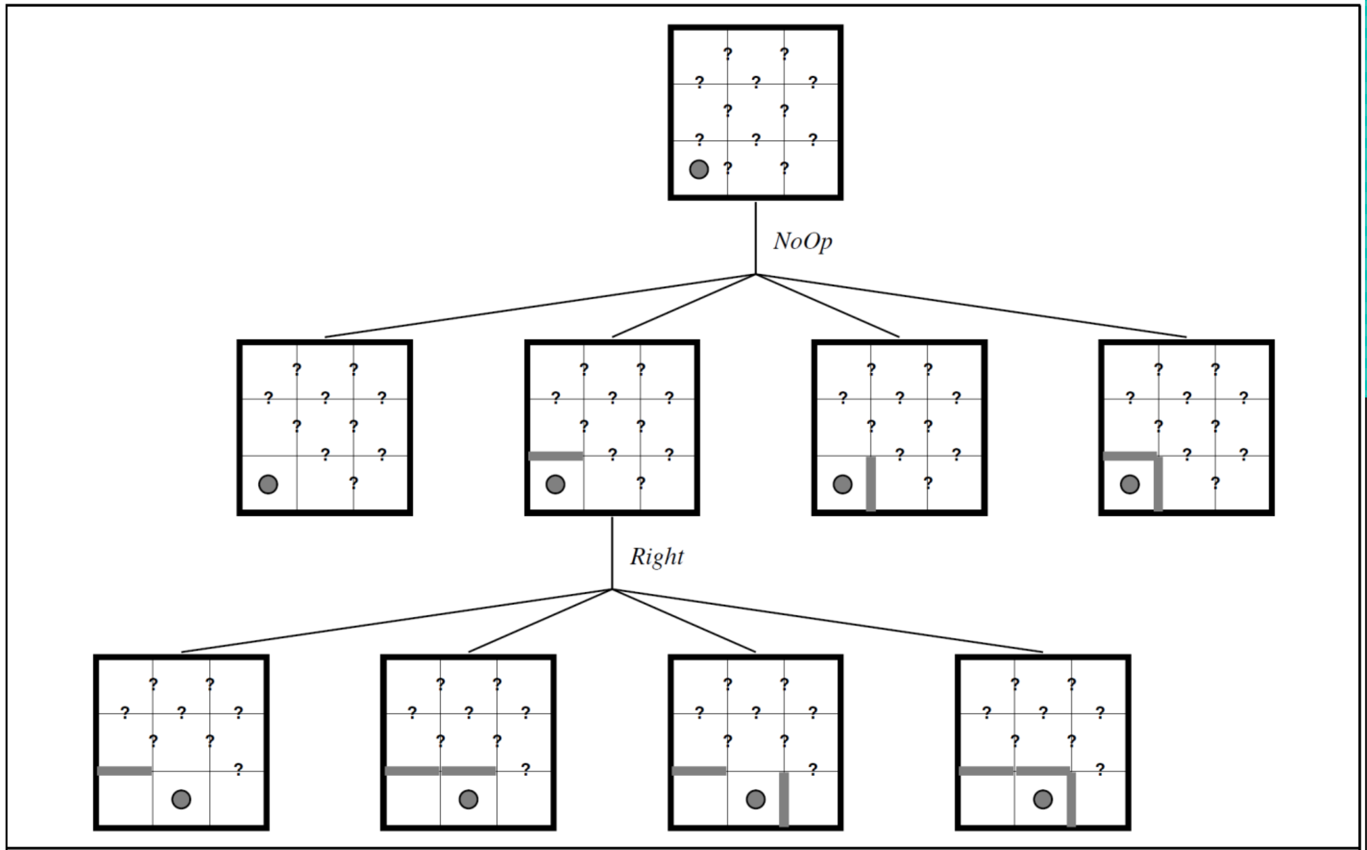


**Figure S4.4** The $3 \times 3$ maze exploration problem: the initial state, first percept, and one selected action with its perceptual outcomes.

**c.** The initial null action leads to four possible belief states, as shown in Figure S4.4. From each belief state, the agent chooses a single action which can lead to up to 8 belief states (on entering the middle square). Given the possibility of having to retrace its steps at a dead end, the agent can explore the entire maze in no more than 18 steps, so the complete plan (expressed as a tree) has no more than $8^{18}$ nodes. On the other hand, there are just $3^{12}$ reachable belief states, so the plan could be expressed more concisely as a table of actions indexed by belief state (a **policy** in the terminology of Chapter 17).

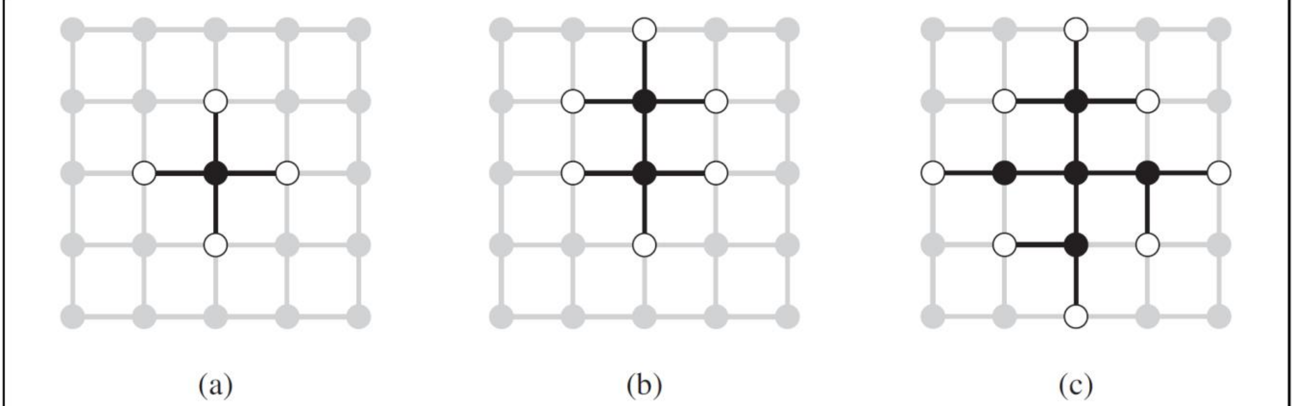


**Figure 3.9** The separation property of GRAPH-SEARCH, illustrated on a rectangular-grid problem. The frontier (white nodes) always separates the explored region of the state space (black nodes) from the unexplored region (gray nodes). In (a), just the root has been expanded. In (b), one leaf node has been expanded. In (c), the remaining successors of the root have been expanded in clockwise order.

**3.26** Consider the unbounded version of the regular 2D grid shown in Figure 3.9. The start state is at the origin, (0,0), and the goal state is at $(x, y)$.

**a.** What is the branching factor $b$ in this state space?

**b.** How many distinct states are there at depth $k$ (for $k > 0$)?

**c.** What is the maximum number of nodes expanded by breadth-first tree search?

**d.** What is the maximum number of nodes expanded by breadth-first graph search?

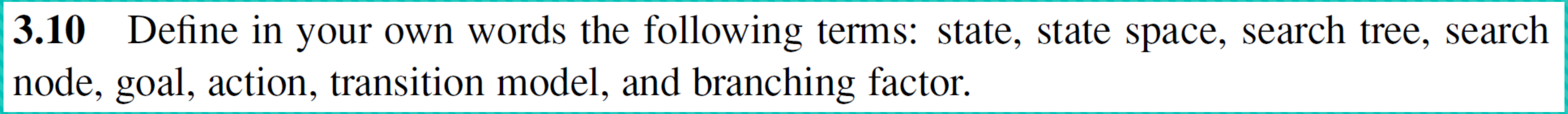
**e.** Is $h = |u - x| + |v - y|$ an admissible heuristic for a state at $(u, v)$? Explain.

**f.** How many nodes are expanded by A* graph search using $h$?

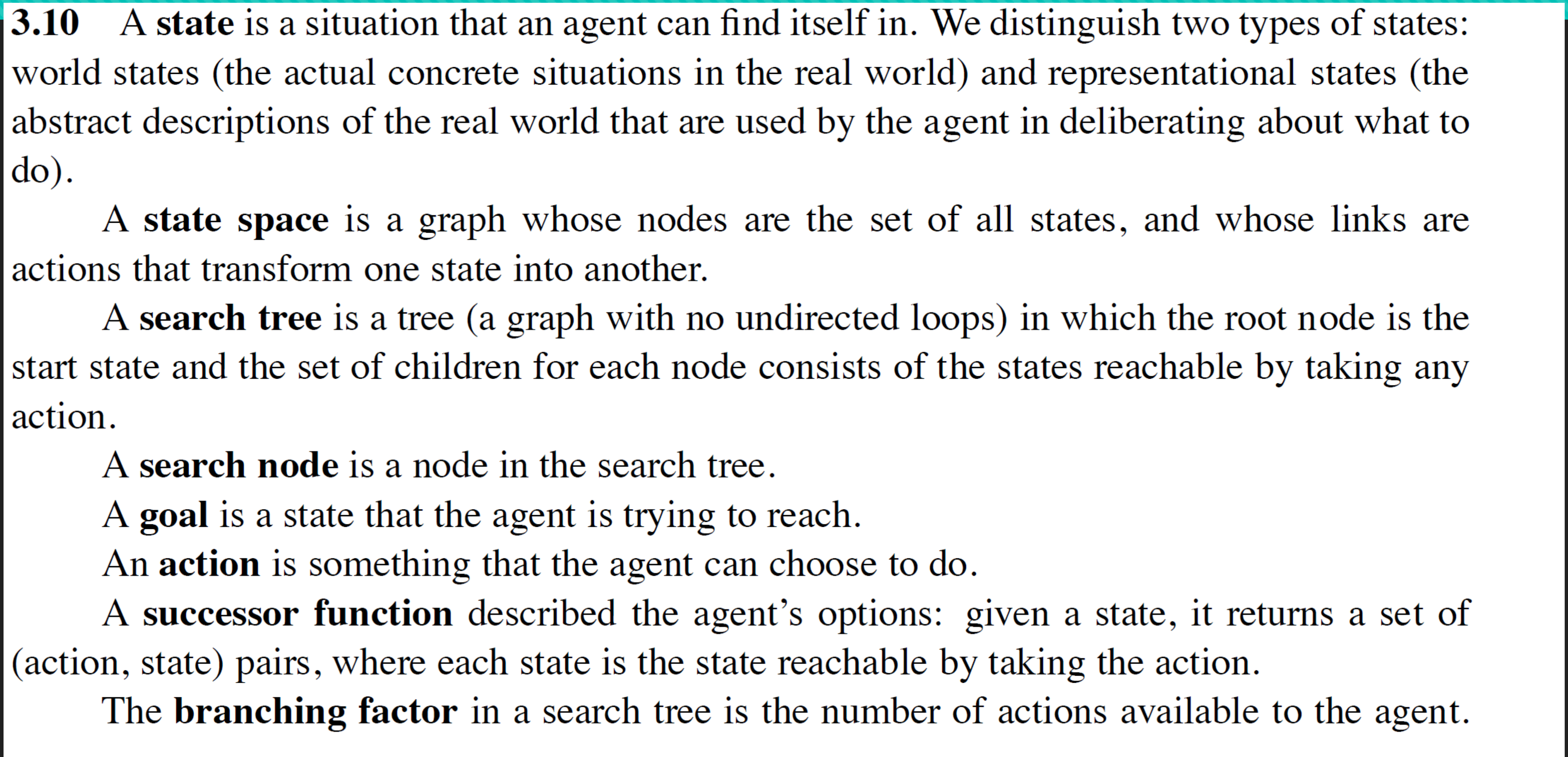
**g.** Does $h$ remain admissible if some links are removed?

**h.** Does $h$ remain admissible if some links are added between nonadjacent states?

# Risposte



**3.10** Define in your own words the following terms: state, state space, search tree, search node, goal, action, transition model, and branching factor.



**3.10** A **state** is a situation that an agent can find itself in. We distinguish two types of states: world states (the actual concrete situations in the real world) and representational states (the abstract descriptions of the real world that are used by the agent in deliberating about what to do).

A **state space** is a graph whose nodes are the set of all states, and whose links are actions that transform one state into another.

A **search tree** is a tree (a graph with no undirected loops) in which the root node is the start state and the set of children for each node consists of the states reachable by taking any action.

A **search node** is a node in the search tree.

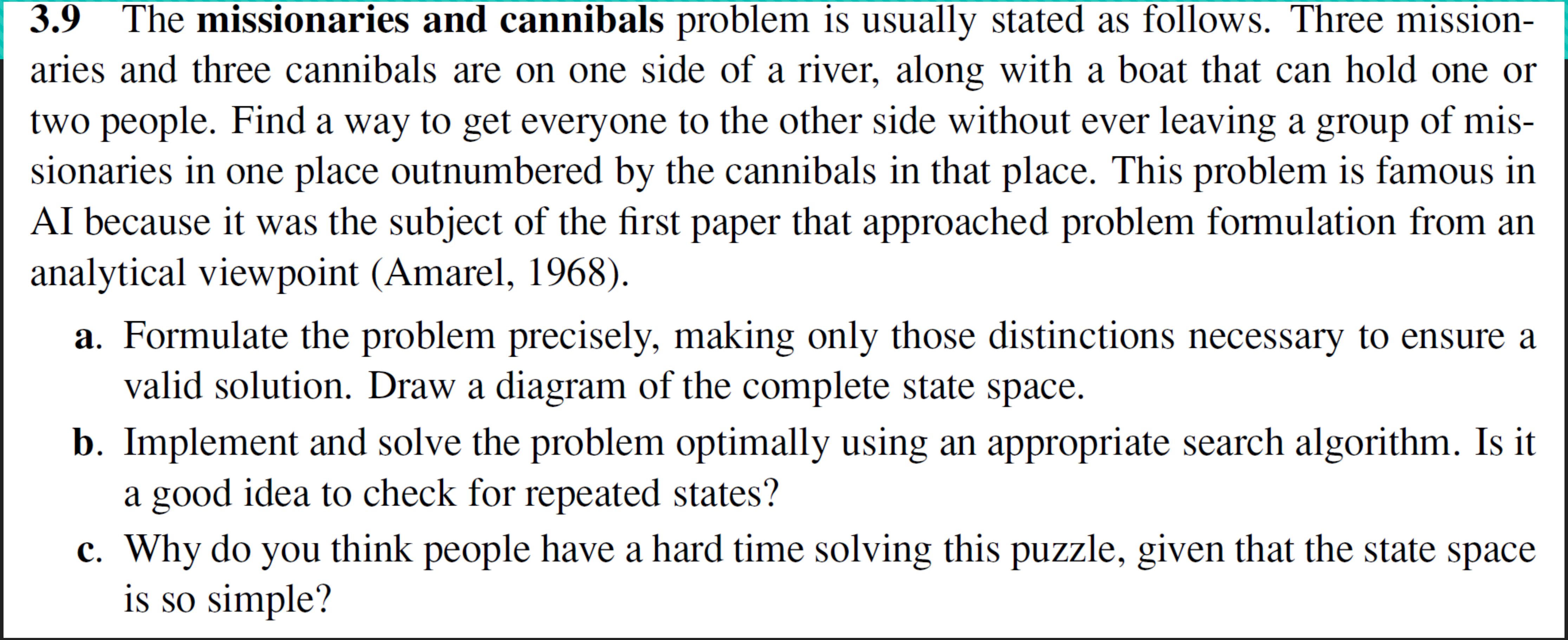
A **goal** is a state that the agent is trying to reach.

An **action** is something that the agent can choose to do.

A **successor function** described the agent's options: given a state, it returns a set of (action, state) pairs, where each state is the state reachable by taking the action.

The **branching factor** in a search tree is the number of actions available to the agent.

## Risposte



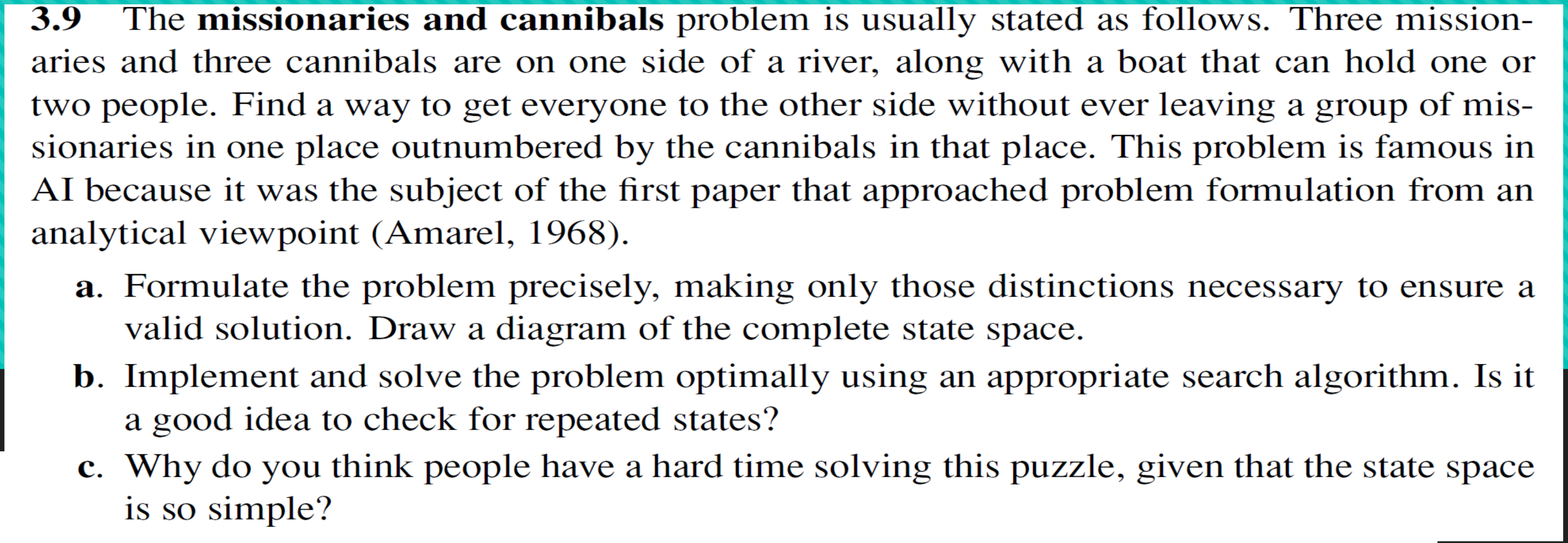
**3.9** The **missionaries and cannibals** problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).

**a**. Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution. Draw a diagram of the complete state space.

**b**. Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?

**c**. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?

# Risposte

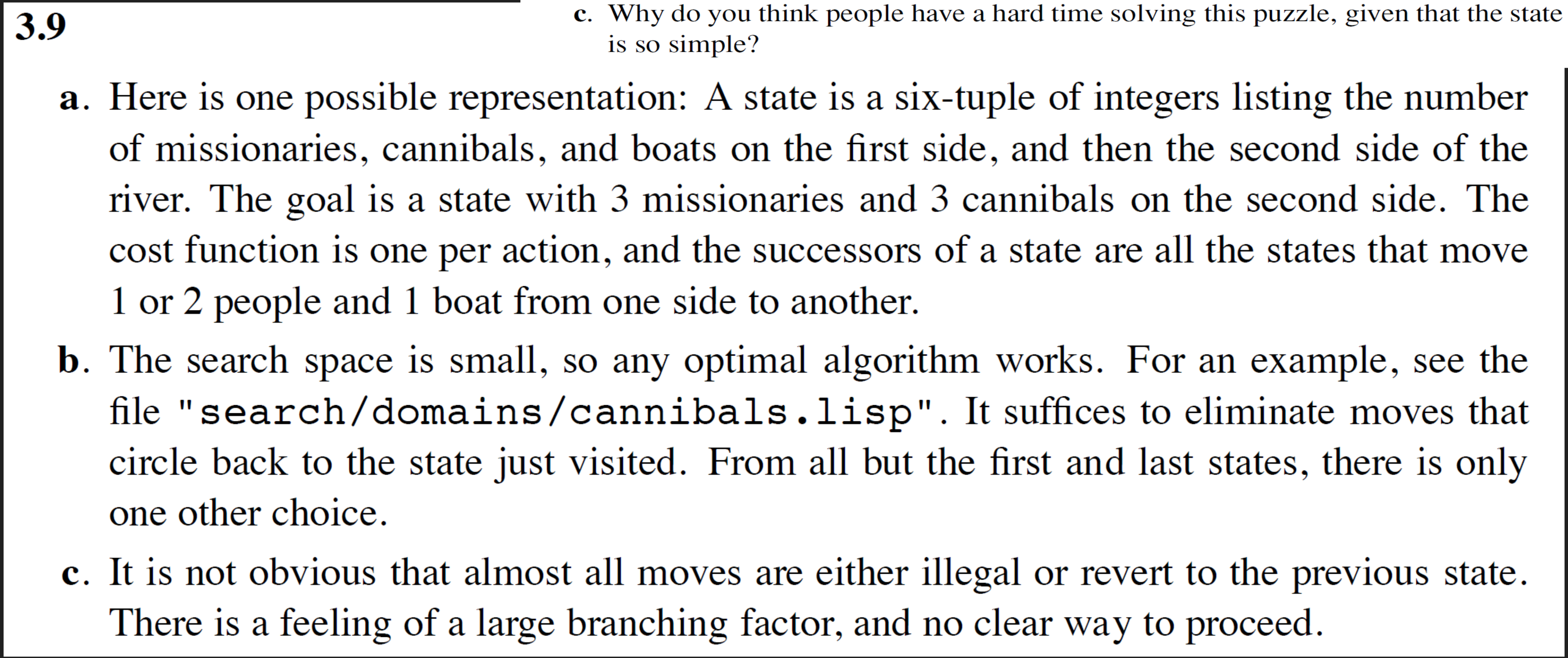


**3.9** The **missionaries and cannibals** problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).

**a.** Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution. Draw a diagram of the complete state space.

**b.** Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?

**c.** Why do you think people have a hard time solving this puzzle, given that the state space is so simple?



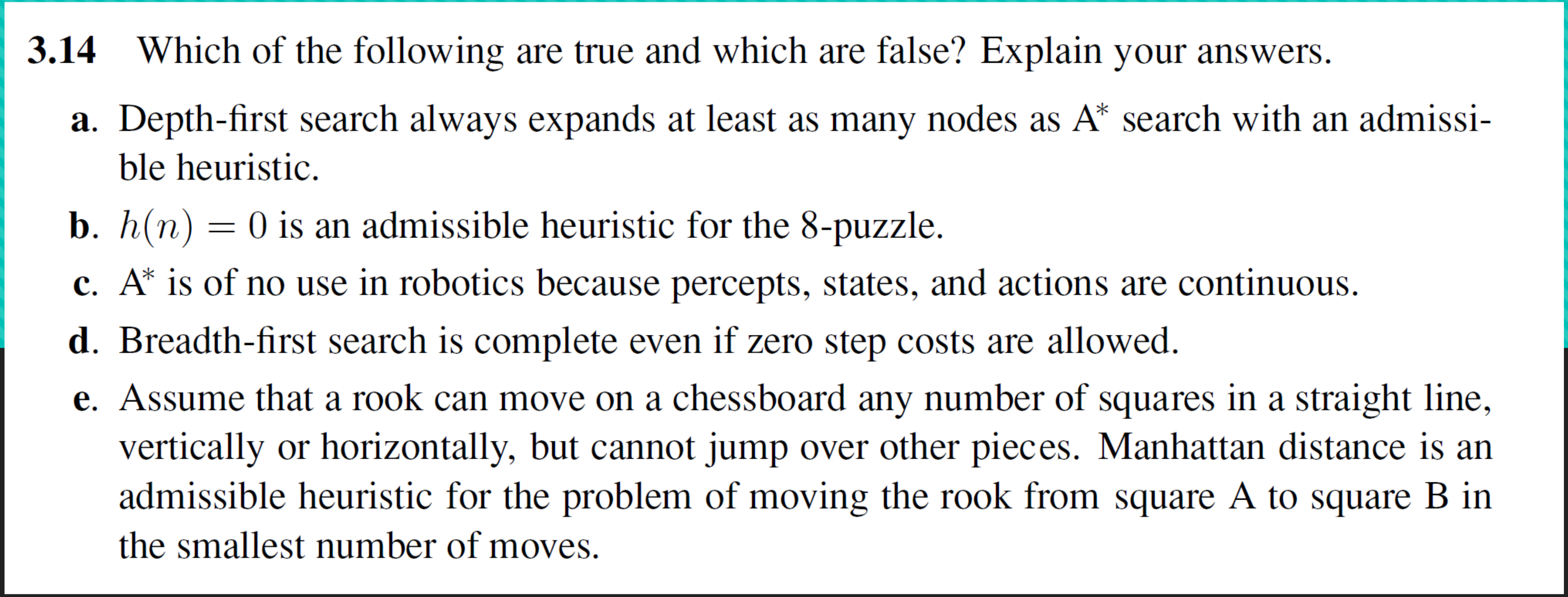
**3.9**

**a.** Here is one possible representation: A state is a six-tuple of integers listing the number of missionaries, cannibals, and boats on the first side, and then the second side of the river. The goal is a state with 3 missionaries and 3 cannibals on the second side. The cost function is one per action, and the successors of a state are all the states that move 1 or 2 people and 1 boat from one side to another.

**b.** The search space is small, so any optimal algorithm works. For an example, see the file `"search/domains/cannibals.lisp"`. It suffices to eliminate moves that circle back to the state just visited. From all but the first and last states, there is only one other choice.

**c.** It is not obvious that almost all moves are either illegal or revert to the previous state. There is a feeling of a large branching factor, and no clear way to proceed.

## Risposte



**3.14** Which of the following are true and which are false? Explain your answers.

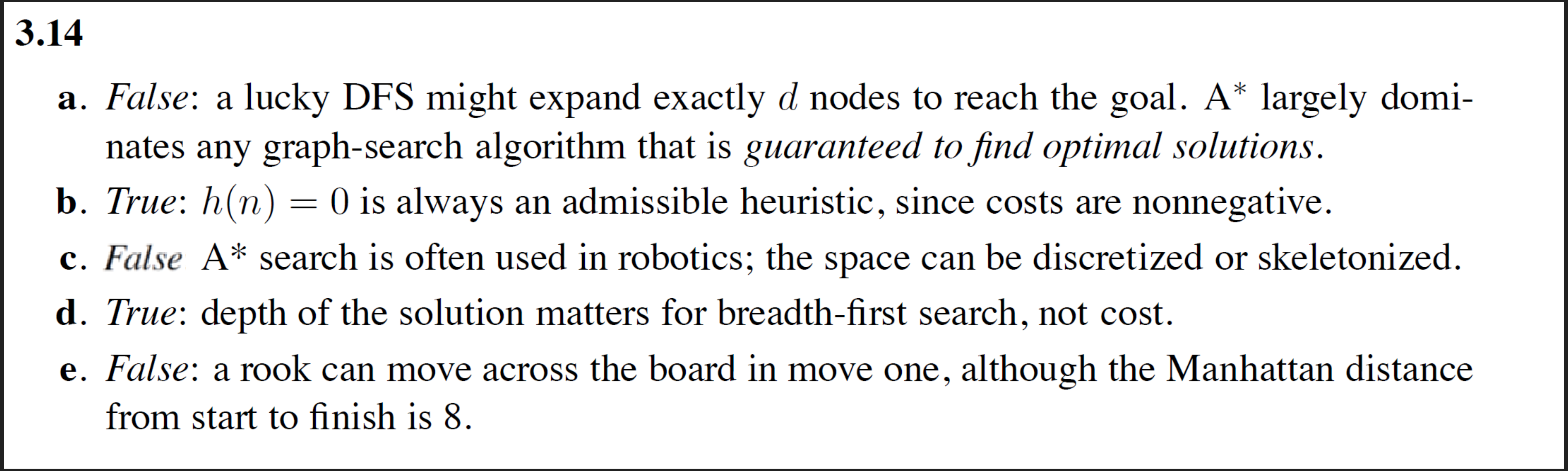
**a.** Depth-first search always expands at least as many nodes as A$^*$ search with an admissible heuristic.

**b.** $h(n) = 0$ is an admissible heuristic for the 8-puzzle.

**c.** A$^*$ is of no use in robotics because percepts, states, and actions are continuous.

**d.** Breadth-first search is complete even if zero step costs are allowed.

**e.** Assume that a rook can move on a chessboard any number of squares in a straight line, vertically or horizontally, but cannot jump over other pieces. Manhattan distance is an admissible heuristic for the problem of moving the rook from square A to square B in the smallest number of moves.



**3.14**

**a.** *False*: a lucky DFS might expand exactly $d$ nodes to reach the goal. A$^*$ largely dominates any graph-search algorithm that is *guaranteed to find optimal solutions*.

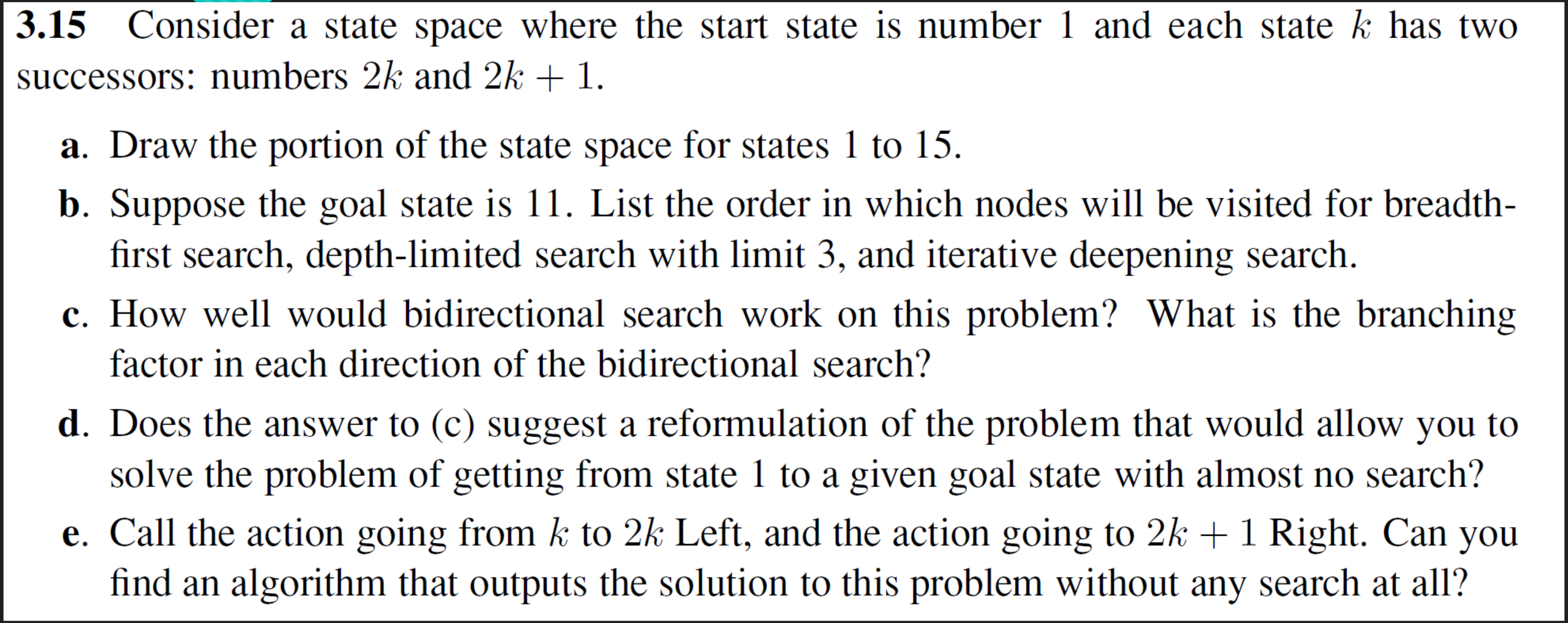
**b.** *True*: $h(n) = 0$ is always an admissible heuristic, since costs are nonnegative.

**c.** *False* A* search is often used in robotics; the space can be discretized or skeletonized.

**d.** *True*: depth of the solution matters for breadth-first search, not cost.

**e.** *False*: a rook can move across the board in move one, although the Manhattan distance from start to finish is 8.

## Risposte



**3.15** Consider a state space where the start state is number 1 and each state $k$ has two successors: numbers $2k$ and $2k + 1$.

**a.** Draw the portion of the state space for states 1 to 15.

**b.** Suppose the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with limit 3, and iterative deepening search.

**c.** How well would bidirectional search work on this problem? What is the branching factor in each direction of the bidirectional search?

**d.** Does the answer to (c) suggest a reformulation of the problem that would allow you to solve the problem of getting from state 1 to a given goal state with almost no search?

**e.** Call the action going from $k$ to $2k$ Left, and the action going to $2k + 1$ Right. Can you find an algorithm that outputs the solution to this problem without any search at all?

**3.15**

numbers $2k$ and $2k + 1$.

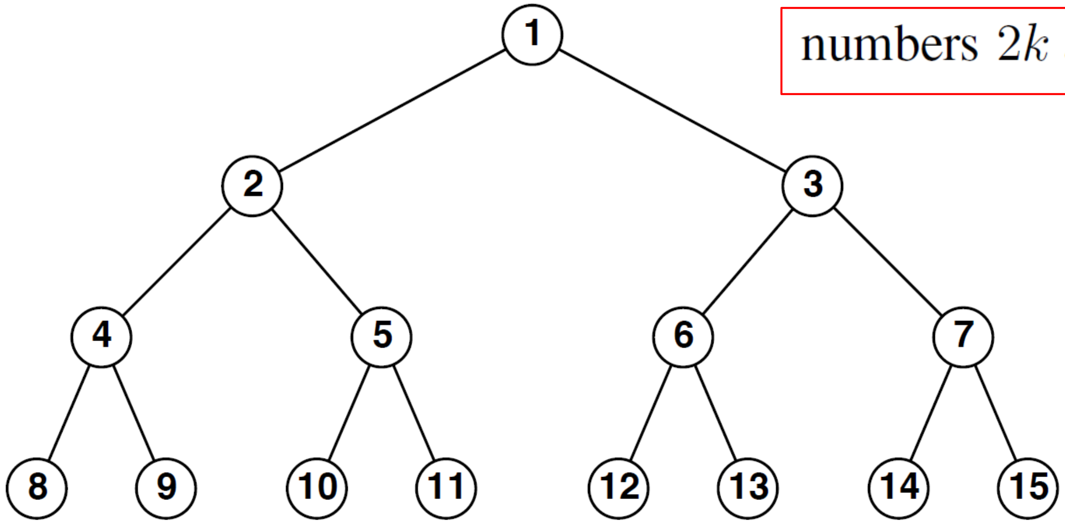


**Figure S3.1** The state space for the problem defined in Ex. 3.15.

**a**. See Figure S3.1.

**b**. Breadth-first: 1 2 3 4 5 6 7 8 9 10 11
Depth-limited: 1 2 4 8 9 5 10 11
Iterative deepening: 1; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11

**c**. Bidirectional search is very useful, because the only successor of $n$ in the reverse direction is $\lfloor(n/2)\rfloor$. This helps focus the search. The branching factor is 2 in the forward direction; 1 in the reverse direction.

**3.15** Consider a state space where the start state is number 1 and each state $k$ has two successors: numbers $2k$ and $2k+1$.

**a.** Draw the portion of the state space for states 1 to 15.

**b.** Suppose the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with limit 3, and iterative deepening search.

**c.** How well would bidirectional search work on this problem? What is the branching factor in each direction of the bidirectional search?

**d.** Does the answer to (c) suggest a reformulation of the problem that would allow you to solve the problem of getting from state 1 to a given goal state with almost no search?

**e.** Call the action going from $k$ to $2k$ Left, and the action going to $2k+1$ Right. Can you find an algorithm that outputs the solution to this problem without any search at all?

**3.15**

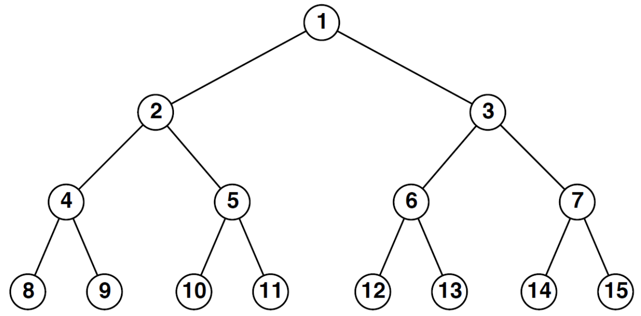


**Figure S3.1** The state space for the problem defined in Ex. 3.15.

**a.** See Figure S3.1.

**b.** Breadth-first: 1 2 3 4 5 6 7 8 9 10 11
Depth-limited: 1 2 4 8 9 5 10 11
Iterative deepening: 1; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11

**c.** Bidirectional search is very useful, because the only successor of $n$ in the reverse direction is $\lfloor (n/2) \rfloor$. This helps focus the search. The branching factor is 2 in the forward direction; 1 in the reverse direction.

**d.** Yes; start at the goal, and apply the single reverse successor action until you reach 1.

**e.** The solution can be read off the binary numeral for the goal number. Write the goal number in binary. Since we can only reach positive integers, this binary expansion beings with a 1. From most- to least- significant bit, skipping the initial 1, go Left to the node $2n$ if this bit is 0 and go Right to node $2n+1$ if it is 1. For example, suppose the goal is 11, which is 1011 in binary. The solution is therefore Left, Right, Right.

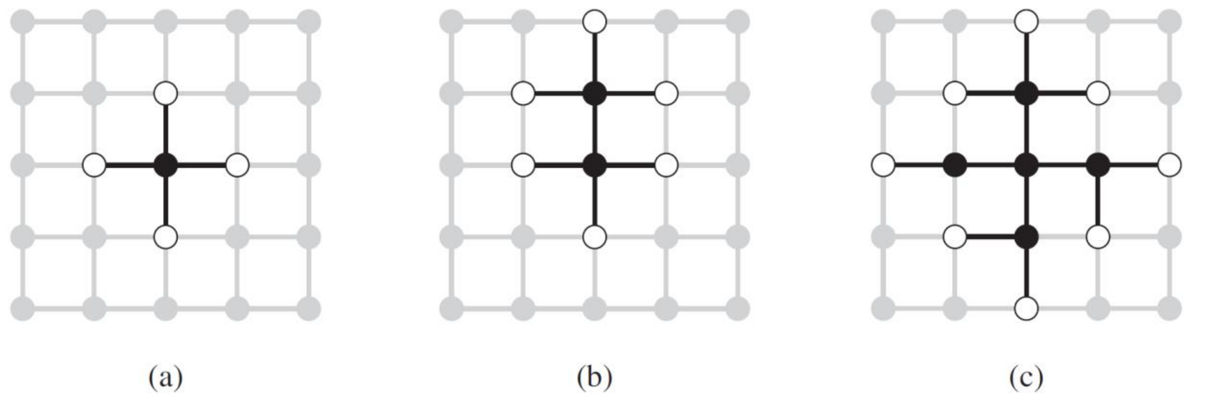


**Figure 3.9** The separation property of GRAPH-SEARCH, illustrated on a rectangular-grid problem. The frontier (white nodes) always separates the explored region of the state space (black nodes) from the unexplored region (gray nodes). In (a), just the root has been expanded. In (b), one leaf node has been expanded. In (c), the remaining successors of the root have been expanded in clockwise order.

**3.26** Consider the unbounded version of the regular 2D grid shown in Figure 3.9. The start state is at the origin, (0,0), and the goal state is at $(x, y)$.

**a.** What is the branching factor $b$ in this state space?

**b.** How many distinct states are there at depth $k$ (for $k > 0$)?

**c.** What is the maximum number of nodes expanded by breadth-first tree search?

**d.** What is the maximum number of nodes expanded by breadth-first graph search?

**e.** Is $h = |u - x| + |v - y|$ an admissible heuristic for a state at $(u, v)$? Explain.

**f.** How many nodes are expanded by A* graph search using $h$?

**g.** Does $h$ remain admissible if some links are removed?

**h.** Does $h$ remain admissible if some links are added between nonadjacent states?

## Hill Climbing

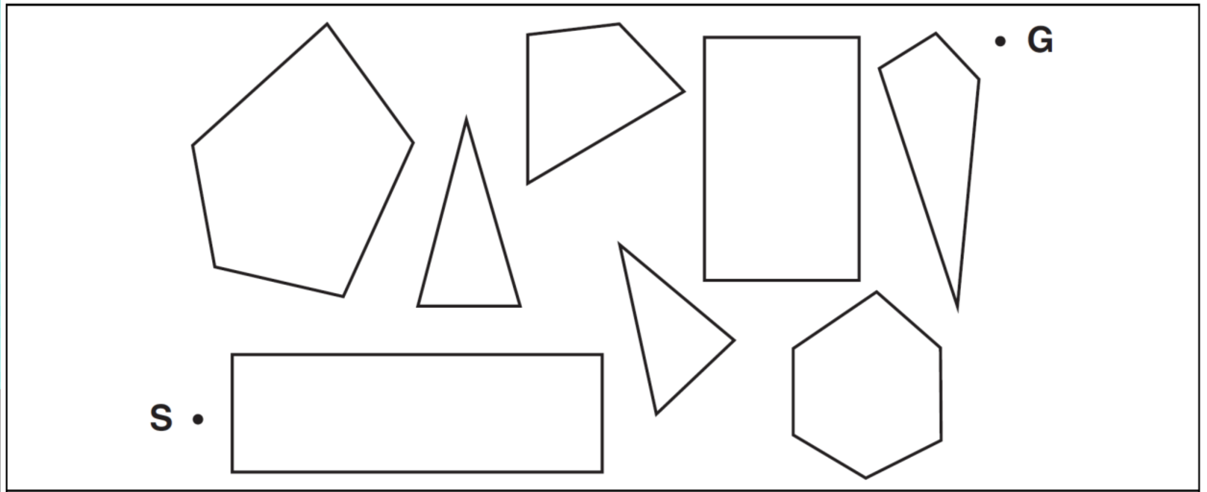


**Figure 3.31** A scene with polygonal obstacles. $S$ and $G$ are the start and goal states.

**4.13** In this exercise, we examine hill climbing in the context of robot navigation, using the environment in Figure 3.31 as an example.

**a.** Repeat Exercise 4.11 using hill climbing. Does your agent ever get stuck in a local minimum? Is it *possible* for it to get stuck with convex obstacles?

**b.** Construct a nonconvex polygonal environment in which the agent gets stuck.

**c.** Modify the hill-climbing algorithm so that, instead of doing a depth-1 search to decide where to go next, it does a depth-$k$ search. It should find the best $k$-step path and do one step along it, and then repeat the process.

**d.** Is there some $k$ for which the new algorithm is guaranteed to escape from local minima?

**e.** Explain how LRTA* enables the agent to escape from local minima in this case.

# Hill Climbing

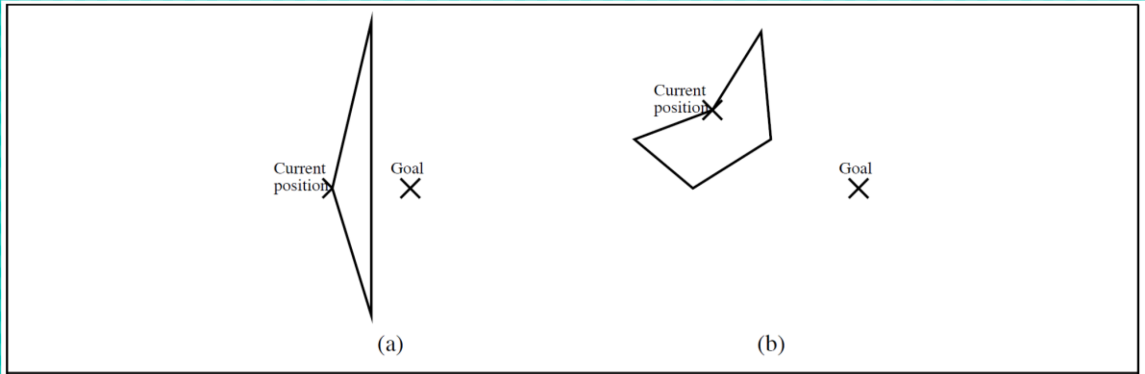


**Figure S4.5** (a) Getting stuck with a convex obstacle. (b) Getting stuck with a nonconvex obstacle.

**4.13** Hillclimbing is surprisingly effective at finding reasonable if not optimal paths for very little computational cost, and seldom fails in two dimensions.

**a.** It is possible (see Figure S4.5(a)) but very unlikely—the obstacle has to have an unusual shape and be positioned correctly with respect to the goal.

**b.** With nonconvex obstacles, getting stuck is much more likely to be a problem (see Figure S4.5(b)).

**c.** Notice that this is just depth-limited search, where you choose a step along the best path even if it is not a solution.

**d.** Set $k$ to the maximum number of sides of any polygon and you can always escape.

**e.** LRTA* always makes a move, but may move back if the old state looks better than the new state. But then the old state is penalized for the cost of the trip, so eventually the local minimum fills up and the agent escapes.