AI Exercise questions

Chapter 1:

1.1 Define in your own words: (a) intelligence, (b) artificial intelligence, (c) agent, (d)rationality, (e) logical reasoning.

1.2 Read Turing's original paper on AI (Turing, 1950). In the paper, he discusses several objections to his proposed enterprise and his test for intelligence. Which objections still carry weight? Are his refutations valid? Can you think of new objections arising from developments since he wrote the paper? In the paper, he predicts that, by the year 2000, a computer will have a 30% chance of passing a five-minute Turing Test with an unskilled interrogator. What chance do you think a computer would have today? In another 50 years?

1.3 Are reflex actions (such as flinching from a hot stove) rational? Are they intelligent?

1.4 Suppose we extend Evans's ANALOGY program so that it can score 200 on a standard IQ test. Would we then have a program more intelligent than a human? Explain.

1.5 The neural structure of the sea slug *Aplysia* has been widely studied (first by Nobel Laureate Eric Kandel) because it has only about 20,000 neurons, most of them large and easily manipulated. If the cycle time for an *Aplysia* neuron is roughly the same as for a human neuron, how does the computational power, in terms of memory updates per second, compare with the high-end computer described in Figure 1.3?

1.6 How could introspection—reporting on one's inner thoughts—be inaccurate? Could I be wrong about what I'm thinking? Discuss.

1.7 To what extent are the following computer systems instances of artificial intelligence:
- Supermarket bar code scanners.
- Web search engines.
- Voice-activated telephone menus.
- Internet routing algorithms that respond dynamically to the state of the network.

1.8 Many of the computational models of cognitive activities that have been proposed involve quite complex mathematical operations, such as convolving an image with a Gaussian or finding a minimum of the entropy function. Most humans (and certainly all animals) never learn this kind of mathematics at all, almost no one learns it before college, and almost no one can compute the convolution of a function with a Gaussian in their head. What sense does it make to say that the "vision system" is doing this kind of mathematics, whereas the actual person has no idea how to do it?

1.9 Why would evolution tend to result in systems that act rationally? What goals are such systems designed to achieve?

1.10 Is AI a science, or is it engineering? Or neither or both? Explain.

1.11 "Surely computers cannot be intelligent—they can do only what their programmers tell them." Is the latter statement true, and does it imply the former?

1.12 “Surely animals cannot be intelligent—they can do only what their genes tell them.” Is the latter statement true, and does it imply the former?

1.13 “Surely animals, humans, and computers cannot be intelligent—they can do only what their constituent atoms are told to do by the laws of physics.” Is the latter statement true, and does it imply the former?

1.14 Examine the AI literature to discover whether the following tasks can currently be solved by computers:
a. Playing a decent game of table tennis (Ping-Pong).
b. Driving in the center of Cairo, Egypt.
c. Driving in Victorville, California.
d. Buying a week’s worth of groceries at the market.
e. Buying a week’s worth of groceries on the Web.
f. Playing a decent game of bridge at a competitive level.
g. Discovering and proving new mathematical theorems.
h. Writing an intentionally funny story.
i. Giving competent legal advice in a specialized area of law.
j. Translating spoken English into spoken Swedish in real time.
k. Performing a complex surgical operation.
For the currently infeasible tasks, try to find out what the difficulties are and predict when, if ever, they will be overcome.

1.15 Various subfields of AI have held contests by defining a standard task and inviting researchers to do their best. Examples include the DARPA Grand Challenge for robotic cars, The International Planning Competition, the Robocup robotic soccer league, the TREC information retrieval event, and contests in machine translation, speech recognition. Investigate five of these contests, and describe the progress made over the years. To what degree have the contests advanced toe state of the art in AI? Do what degree do they hurt the field by drawing energy away from new ideas?

Chapter 2:

2.1 Suppose that the performance measure is concerned with just the first T time steps of the environment and ignores everything thereafter. Show that a rational agent's action may depend not just on the state of the environment but also on the time step it has reached.

2.2 Let us examine the rationality of various vacuum-cleaner agent functions.
a. Show that the simple vacuum-cleaner agent function described in Figure 2.3 is indeed rational under the assumptions listed on page 38.
b. Describe a rational agent function for the case in which each movement costs one point. Does the corresponding agent program require internal state?
c. Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense for the agent to learn from its experience in these cases? If so, what should it learn? If not, why not?
2.3 For each of the following assertions, say whether it is true or false and support your answer with examples or counterexamples where appropriate.
a. An agent that senses only partial information about the state cannot be perfectly rational.

b. There exist task environments in which no pure reflex agent can behave rationally.

c. There exists a task environment in which every agent is rational.

d. The input to an agent program is the same as the input to the agent function.

e. Every agent function is implementable by some program/machine combination.

f. Suppose an agent selects its action uniformly at random from the set of possible actions. There exists a deterministic task environment in which this agent is rational.

g. It is possible for a given agent to be perfectly rational in two distinct task environments.

h. Every agent is rational in an unobservable environment.

i. A perfectly rational poker-playing agent never loses.

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.
- Playing soccer.
- Exploring the subsurface oceans of Titan.
- Shopping for used AI books on the Internet.
- Playing a tennis match.
- Practicing tennis against a wall.
- Performing a high jump.
- Knitting a sweater.
- Bidding on an item at an auction.

2.5 Define in your own words the following terms: agent, agent function, agent program, rationality, autonomy, reflex agent, model-based agent, goal-based agent, utility-based agent, learning agent.

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?

2.7 Write pseudocode agent programs for the goal-based and utility-based agents.
The following exercises all concern the implementation of environments and agents for the vacuum-cleaner world.

2.8 Implement a performance-measuring environment simulator for the vacuum-cleaner world depicted in Figure 2.2 and specified on page 38. Your implementation should be modular so that the sensors, actuators, and environment characteristics (size, shape, dirt placement, etc.) can be changed easily. (*Note:* for some choices of programming language and operating system there are already implementations in the online code repository.)

2.9 Implement a simple reflex agent for the vacuum environment in Exercise 2.8. Run the environment with this agent for all possible initial dirt configurations and agent locations. Record the performance score for each configuration and the overall average score.

2.10 Consider a modified version of the vacuum environment in Exercise 2.8, in which the agent is penalized one point for each movement.
a. Can a simple reflex agent be perfectly rational for this environment? Explain.
b. What about a reflex agent with state? Design such an agent.
c. How do your answers to **a** and **b** change if the agent's percepts give it the clean/dirty status of every square in the environment?

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?

2.12 Repeat Exercise 2.11 for the case in which the location sensor is replaced with a "bump" sensor that detects the agent's attempts to move into an obstacle or to cross the boundaries of the environment. Suppose the bump sensor stops working; how should the

agent behave?

2.13 The vacuum environments in the preceding exercises have all been deterministic. Discuss possible agent programs for each of the following stochastic versions:
a. Murphy's law: twenty-five percent of the time, the Suck action fails to clean the floor if it is dirty and deposits dirt onto the floor if the floor is clean. How is your agent program affected if the dirt sensor gives the wrong answer 10% of the time?
b. Small children: At each time step, each clean square has a 10% chance of becoming dirty. Can you come up with a rational agent design for this case?

Chapter 3:

3.1 Explain why problem formulation must follow goal formulation.

3.2 Your goal is to navigate a robot out of a maze. The robot starts in the center of the maze facing north. You can turn the robot to face north, east, south, or west. You can direct the robot to move forward a certain distance, although it will stop before hitting a wall.
a. Formulate this problem. How large is the state space?
b. In navigating a maze, the only place we need to turn is at the intersection of two or more corridors. Reformulate this problem using this observation. How large is the state space now?
c. From each point in the maze, we can move in any of the four directions until we reach a turning point, and this is the only action we need to do. Reformulate the problem using these actions. Do we need to keep track of the robot's orientation now?
d. In our initial description of the problem we already abstracted from the real world, restricting actions and removing details. List three such simplifications we made.

3.3 Suppose two friends live in different cities on a map, such as the Romania map shown in Figure 3.2. On every turn, we can simultaneously move each friend to a neighboring city on the map. The amount of time needed to move from city i to neighbor j is equal to the road distance d(i, j) between the cities, but on each turn the friend that arrives first must wait until the other one arrives (and calls the first on his/her cell phone) before the next turn can begin. We want the two friends to meet as quickly as possible.
a. Write a detailed formulation for this search problem. (You will find it helpful to define some formal notation here.)
b. Let D(i, j) be the straight-line distance between cities i and j. Which of the following heuristic functions are admissible? (i) D(i, j); (ii) 2 · D(i, j); (iii) D(i, j)/2.
c. Are there completely connected maps for which no solution exists?
d. Are there maps in which all solutions require one friend to visit the same city twice?

3.4 Show that the 8-puzzle states are divided into two disjoint sets, such that any state is reachable from any other state in the same set, while no state is reachable from any state in the other set. (*Hint:* See Berlekamp *et al.* (1982).) Devise a procedure to decide which set a given state is in, and explain why this is useful for generating random states.

3.5 Consider the n-queens problem using the "efficient" incremental formulation given on page 72. Explain why the state space has at least 3 √
n! states and estimate the largest n for which exhaustive exploration is feasible. (*Hint*: Derive a lower bound on the branching factor by considering the maximum number of squares that a queen can attack in any column.)

3.6 Give a complete problem formulation for each of the following. Choose a formulation that is precise enough to be implemented.
a. Using only four colors, you have to color a planar map in such a way that no two adjacent regions have the same color.
b. A 3-foot-tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. He would like to get the bananas. The room contains two stackable, movable,

climbable 3-foot-high crates.
c. You have a program that outputs the message “illegal input record” when fed a certain file of input records. You know that processing of each record is independent of the other records. You want to discover what record is illegal.
d. You have three jugs, measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from one to another or onto the ground. You need to measure out exactly one gallon.

3.7 Consider the problem of finding the shortest path between two points on a plane that has convex polygonal obstacles as shown in Figure 3.31. This is an idealization of the problem that a robot has to solve to navigate in a crowded environment.
a. Suppose the state space consists of all positions (x, y) in the plane. How many states are there? How many paths are there to the goal?
b. Explain briefly why the shortest path from one polygon vertex to any other in the scene must consist of straight-line segments joining some of the vertices of the polygons. Define a good state space now. How large is this state space?
c. Define the necessary functions to implement the search problem, including an ACTIONS function that takes a vertex as input and returns a set of vectors, each of which maps the current vertex to one of the vertices that can be reached in a straight line. (Do not forget the neighbors on the same polygon.) Use the straight-line distance for the heuristic function.
d. Apply one or more of the algorithms in this chapter to solve a range of problems in the domain, and comment on their performance.

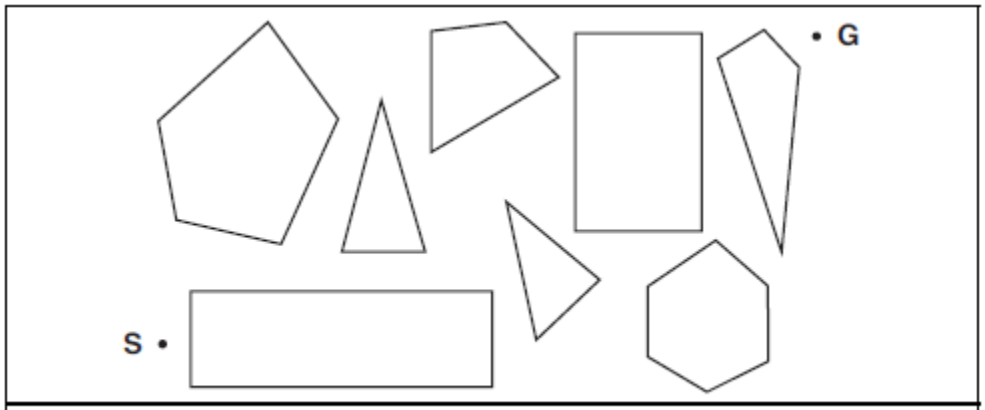

Figure 3.31 A scene with polygonal obstacles. S and G are the start and goal states.

3.8 On page 68, we said that we would not consider problems with negative path costs. In this exercise, we explore this decision in more depth.
a. Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal algorithm to explore the entire state space.
b. Does it help if we insist that step costs must be greater than or equal to some negative constant c? Consider both trees and graphs.
c. Suppose that a set of actions forms a loop in the state space such that executing the set in some order results in no net change to the state. If all of these actions have negative cost, what does this imply about the optimal behavior for an agent in such an environment?
d. One can easily imagine actions with high negative cost, even in domains such as route finding. For example, some stretches of road might have such beautiful scenery as to

far outweigh the normal costs in terms of time and fuel. Explain, in precise terms, within the context of state-space search, why humans do not drive around scenic loops indefinitely, and explain how to define the state space and actions for route finding so that artificial agents can also avoid looping.
e. Can you think of a real domain in which step costs are such as to cause looping?

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?

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

3.11 What's the difference between a world state, a state description, and a search node? Why is this distinction useful?

3.12 An action such as *Go(Sibiu)* really consists of a long sequence of finer-grained actions: turn on the car, release the brake, accelerate forward, etc. Having composite actions of this kind reduces the number of steps in a solution sequence, thereby reducing the search time. Suppose we take this to the logical extreme, by making super-composite actions out of every possible sequence of *Go* actions. Then every problem instance is solved by a single supercomposite action, such as *Go(Sibiu)Go(Rimnicu Vilcea)Go(Pitesti)Go(Bucharest)*. Explain how search would work in this formulation. Is this a practical approach for speeding up problem solving?

3.13 Prove that GRAPH-SEARCH satisfies the graph separation property illustrated in Figure 3.9. (*Hint*: Begin by showing that the property holds at the start, then show that if it holds before an iteration of the algorithm, it holds afterwards.) Describe a search algorithm that violates the property.

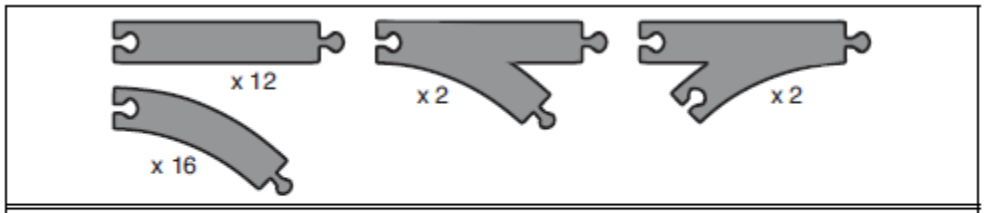

Figure 3.32 The track pieces in a wooden railway set; each is labeled with the number of copies in the set. Note that curved pieces and "fork" pieces ("switches" or "points") can be flipped over so they can curve in either direction. Each curve subtends 45 degrees.

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.

3.15 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 breadthfirst 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?
3.16 A basic wooden railway set contains the pieces shown in Figure 3.32. The task is to connect these pieces into a railway that has no overlapping tracks and no loose ends where a train could run off onto the floor.
a. Suppose that the pieces fit together *exactly* with no slack. Give a precise formulation of the task as a search problem.
b. Identify a suitable uninformed search algorithm for this task and explain your choice.
c. Explain why removing any one of the "fork" pieces makes the problem unsolvable.
d. Give an upper bound on the total size of the state space defined by your formulation. (*Hint*: think about the maximum branching factor for the construction process and the maximum depth, ignoring the problem of overlapping pieces and loose ends. Begin by pretending that every piece is unique.)

3.17 On page 90, we mentioned **iterative lengthening search**, an iterative analog of uniform cost search. The idea is to use increasing limits on path cost. If a node is generated whose path cost exceeds the current limit, it is immediately discarded. For each new iteration, the limit is set to the lowest path cost of any node discarded in the previous iteration.
a. Show that this algorithm is optimal for general path costs.
b. Consider a uniform tree with branching factor b, solution depth d, and unit step costs. How many iterations will iterative lengthening require?
c. Now consider step costs drawn from the continuous range [_, 1], where 0 < _ < 1. How many iterations are required in the worst case?
d. Implement the algorithm and apply it to instances of the 8-puzzle and traveling salesperson problems. Compare the algorithm's performance to that of uniform-cost search, and comment on your results.

3.18 Describe a state space in which iterative deepening search performs much worse than

depth-first search (for example, O(n
2) vs. O(n)).

3.19 Write a program that will take as input two Web page URLs and find a path of links from one to the other. What is an appropriate search strategy? Is bidirectional search a good idea? Could a search engine be used to implement a predecessor function?

3.20 Consider the vacuum-world problem defined in Figure 2.2.
a. Which of the algorithms defined in this chapter would be appropriate for this problem? Should the algorithm use tree search or graph search?
b. Apply your chosen algorithm to compute an optimal sequence of actions for a 3×3 world whose initial state has dirt in the three top squares and the agent in the center.
c. Construct a search agent for the vacuum world, and evaluate its performance in a set of 3×3 worlds with probability 0.2 of dirt in each square. Include the search cost as well as path cost in the performance measure, using a reasonable exchange rate.
d. Compare your best search agent with a simple randomized reflex agent that sucks if there is dirt and otherwise moves randomly.
e. Consider what would happen if the world were enlarged to n × n. How does the performance of the search agent and of the reflex agent vary with n?

3.21 Prove each of the following statements, or give a counterexample:
a. Breadth-first search is a special case of uniform-cost search.
b. Depth-first search is a special case of best-first tree search.
c. Uniform-cost search is a special case of A∗ search.

3.22 Compare the performance of A∗ and RBFS on a set of randomly generated problems in the 8-puzzle (with Manhattan distance) and TSP (with MST—see Exercise 3.30) domains. Discuss your results. What happens to the performance of RBFS when a small random number is added to the heuristic values in the 8-puzzle domain?

3.23 Trace the operation of A∗ search applied to the problem of getting to Bucharest from Lugoj using the straight-line distance heuristic. That is, show the sequence of nodes that the algorithm will consider and the f, g, and h score for each node.

3.24 Devise a state space in which A∗ using GRAPH-SEARCH returns a suboptimal solution with an h(n) function that is admissible but inconsistent.

3.25 The **heuristic path algorithm** (Pohl, HEURISTIC PATH 1977) is a best-first search in which the evalu-
ALGORITHM
ation 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?

3.26 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?

3.27 n vehicles occupy squares (1, 1) through (n, 1) (i.e., the bottom row) of an n×n grid. The vehicles must be moved to the top row but in reverse order; so the vehicle i that starts in (i, 1) must end up in (n−i+1, n). On each time step, every one of the n vehicles can move one square up, down, left, or right, or stay put; but if a vehicle stays put, one other adjacent vehicle (but not more than one) can hop over it. Two vehicles cannot occupy the same square.
a. Calculate the size of the state space as a function of n.
b. Calculate the branching factor as a function of n.
c. Suppose that vehicle i is at (xi, yi); write a nontrivial admissible heuristic hi for the number of moves it will require to get to its goal location (n − i + 1, n), assuming no other vehicles are on the grid.
d. Which of the following heuristics are admissible for the problem of moving all n vehicles to their destinations? Explain.
(i) _n
i=1
hi.
(ii) max{h1, . . . , hn}.
(iii) min{h1, . . . , hn}.

3.28 Invent a heuristic function for the 8-puzzle that sometimes overestimates, and show how it can lead to a suboptimal solution on a particular problem. (You can use a computer to help if you want.) Prove that if h never overestimates by more than c, A∗ using h returns a solution whose cost exceeds that of the optimal solution by no more than c.

3.29 Prove that if a heuristic is consistent, it must be admissible. Construct an admissible heuristic that is not consistent.

3.30 The traveling salesperson problem (TSP) can be solved with the minimum-spanningtree (MST) heuristic, which estimates the cost of completing a tour, given that a partial tour has already been constructed. The MST cost of a set of cities is the smallest sum of the link costs of any tree that connects all the cities.
a. Show how this heuristic can be derived from a relaxed version of the TSP.
b. Show that the MST heuristic dominates straight-line distance.
c. Write a problem generator for instances of the TSP where cities are represented by random points in the unit square.
d. Find an efficient algorithm in the literature for constructing the MST, and use it with A∗ graph search to solve instances of the TSP.

3.31 On page 105, we defined the relaxation of the 8-puzzle in which a tile can move from square A to square B if B is blank. The exact solution of this problem defines **Gaschnig's heuristic** (Gaschnig, 1979). Explain why Gaschnig's heuristic is at least as accurate as h1 (misplaced tiles), and show cases where it is more accurate than both h1 and h2 (Manhattan distance). Explain how to calculate Gaschnig's heuristic efficiently.

3.32 We gave two simple heuristics for the 8-puzzle: Manhattan distance and misplaced tiles. Several heuristics in the literature purport to improve on this—see, for example, Nilsson (1971), Mostow and Prieditis (1989), and Hansson *et al.* (1992). Test these claims by implementing the heuristics and comparing the performance of the resulting algorithms

Chapter 4:

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 = ∞at all times.
e. Genetic algorithm with population size N = 1.

4.2 Exercise 3.16 considers the problem of building railway tracks under the assumption that pieces fit exactly with no slack. Now consider the real problem, in which pieces don't fit exactly but allow for up to 10 degrees of rotation to either side of the "proper" alignment. Explain how to formulate the problem so it could be solved by simulated annealing.

4.3 In this exercise, we explore the use of local search methods to solve TSPs of the type defined in Exercise 3.30.
a. Implement and test a hill-climbing method to solve TSPs. Compare the results with optimal solutions obtained from the A∗ algorithm with the MST heuristic (Exercise 3.30).
b. Repeat part (a) using a genetic algorithm instead of hill climbing. You may want to consult Larra˜naga *et al.* (1999) for some suggestions for representations.

4.4 Generate a large number of 8-puzzle and 8-queens instances and solve them (where possible) by hill climbing (steepest-ascent and first-choice variants), hill climbing with random restart, and simulated annealing. Measure the search cost and percentage of solved problems and graph these against the optimal solution cost. Comment on your results.

4.5 The AND-OR-GRAPH-SEARCH algorithm in Figure 4.11 checks for repeated states only on the path from the root to the current state. Suppose that, in addition, the algorithm were to store *every* visited state and check against that list. (See BREADTH-FIRST-SEARCH in Figure 3.11 for an example.) Determine the information that should be stored and how the algorithm should use that information when a repeated state is found. (*Hint*: You will need to distinguish at least between states for which a successful subplan was constructed previously and states for which no subplan could be found.) Explain how to use labels, as defined in Section 4.3.3, to avoid having multiple copies of subplans.

4.6 Explain precisely how to modify the AND-OR-GRAPH-SEARCH algorithm to generate a cyclic plan if no acyclic plan exists. You will need to deal with three issues: labeling the plan steps so that a cyclic plan can point back to an earlier part of the plan, modifying OR-SEARCH so that it continues to look for acyclic plans after finding a cyclic plan, and augmenting the plan representation to indicate whether a plan is cyclic. Show how your algorithm works on (a) the slippery vacuum world, and (b) the slippery, erratic vacuum world. You might wish to use a computer implementation to check your results.

4.7 In Section 4.4.1 we introduced belief states to solve sensorless search problems. A sequence of actions solves a sensorless problem if it maps every physical state in the initial belief state b to a goal state. Suppose the agent knows $h^*(s)$, the true optimal cost of solving the physical state s in the fully observable problem, for every state s in b. Find an admissible heuristic h(b) for the sensorless problem in terms of these costs, and prove its admissibilty. Comment on the accuracy of this heuristic on the sensorless vacuum problem of Figure 4.14. How well does A^* perform?

4.8 This exercise explores subset–superset relations between belief states in sensorless or partially observable environments.
a. Prove that if an action sequence is a solution for a belief state b, it is also a solution for any subset of b. Can anything be said about supersets of b?
b. Explain in detail how to modify graph search for sensorless problems to take advantage of your answers in (a).
c. Explain in detail how to modify AND–OR search for partially observable problems, beyond the modifications you describe in (b).

4.9 On page 139 it was assumed that a given action would have the same cost when executed in any physical state within a given belief state. (This leads to a belief-state search problem with well-defined step costs.) Now consider what happens when the assumption does not hold. Does the notion of optimality still make sense in this context, or does it require modification? Consider also various possible definitions of the "cost" of executing an action in a belief state; for example, we could use the *minimum* of the physical costs; or the *maximum*; or a cost *interval* with the lower bound being the minimum cost and the upper bound being the maximum; or just keep the set of all possible costs for that action. For each of these, explore whether A^* (with modifications if necessary) can return optimal solutions.

4.10 Consider the sensorless version of the erratic vacuum world. Draw the belief-state space reachable from the initial belief state {1, 2, 3, 4, 5, 6, 7, 8}, and explain why the problem is unsolvable.

4.11 We can turn the navigation problem in Exercise 3.7 into an environment as follows:
• The percept will be a list of the positions, *relative to the agent*, of the visible vertices. The percept does *not* include the position of the robot! The robot must learn its own position from the map; for now, you can assume that each location has a different "view."
• Each action will be a vector describing a straight-line path to follow. If the path is unobstructed, the action succeeds; otherwise, the robot stops at the point where its path first intersects an obstacle. If the agent returns a zero motion vector and is at the goal (which is fixed and known), then the environment teleports the agent to a *random location* (not inside an obstacle).
• The performance measure charges the agent 1 point for each unit of distance traversed and awards 1000 points each time the goal is reached.
a. Implement this environment and a problem-solving agent for it. After each teleportation, the agent will need to formulate a new problem, which will involve discovering its current location.
b. Document your agent's performance (by having the agent generate suitable commentary as it moves around) and report its performance over 100 episodes.
c. Modify the environment so that 30% of the time the agent ends up at an unintended destination (chosen randomly from the other visible vertices if any; otherwise, no move at all). This is a crude model of the motion errors of a real robot. Modify the agent so that when such an error is detected, it finds out where it is and then constructs a plan to get back to where it was and resume the old plan. Remember that sometimes getting back to where it was might also fail! Show an example of the agent successfully overcoming two successive motion errors and still reaching the goal.
d. Now try two different recovery schemes after an error: (1) head for the closest vertex on the original route; and (2) replan a route to the goal from the new location. Compare the performance of the three recovery schemes. Would the inclusion of search costs affect the comparison?
e. Now suppose that there are locations from which the view is identical. (For example, suppose the world is a grid with square obstacles.) What kind of problem does the agent now face? What do solutions look like?

4.12 Suppose that an agent is in a 3×3 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.

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. Isthere somek 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.

4.14 Like DFS, online DFS is incomplete for reversible state spaces with infinite paths. For example, suppose that states are points on the infinite two-dimensional grid and actions are unit vectors (1, 0), (0, 1), (−1, 0), (0,−1), tried in that order. Show that online DFS starting at (0, 0) will not reach (1,−1). Suppose the agent can observe, in addition to its current state, all successor states and the actions that would lead to them. Write an algorithm that is complete even for bidirected state spaces with infinite paths. What states does it visit in reaching (1,−1)?

Chapter 5:

5.1 Suppose you have an oracle, OM(s), that correctly predicts the opponent's move in any state. Using this, formulate the definition of a game as a (single-agent) search problem. Describe an algorithm for finding the optimal move.

5.2 Consider the problem of solving two 8-puzzles.
a. Give a complete problem formulation in the style of Chapter 3.
b. How large is the reachable state space? Give an exact numerical expression.
c. Suppose we make the problem adversarial as follows: the two players take turns moving; a coin is flipped to determine the puzzle on which to make a move in that turn; and the winner is the first to solve one puzzle. Which algorithm can be used to choose a move in this setting?
d. Give an informal proof that someone will eventually win if both play perfectly.

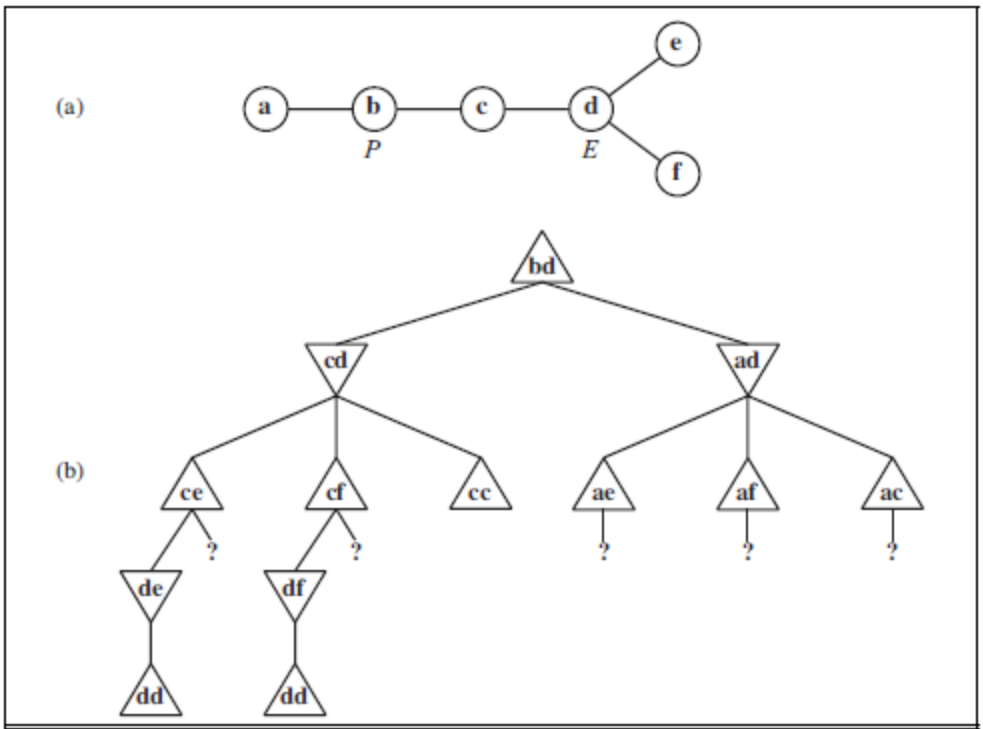

Figure 5.16 (a) A map where the cost of every edge is 1. Initially the pursuer P is at node **b** and the evader E is at node **d**. (b) A partial game tree for this map. Each node is labeled with the P, E positions. P moves first. Branches marked "?" have yet to be explored.

5.3 Imagine that, in Exercise 3.3, one of the friends wants to avoid the other. The problem then becomes a two-player **pursuit–evasion** game. We assume now that the players take turns moving. The game ends only when the players are on the same node; the terminal payoff to the pursuer is minus the total time taken. (The evader "wins" by never losing.) An example is shown in Figure 5.16.

a. Copy the game tree and mark the values of the terminal nodes.

b. Next to each internal node, write the strongest fact you can infer about its value (a number, one or more inequalities such as "≥ 14", or a "?").

c. Beneath each question mark, write the name of the node reached by that branch.

d. Explain how a bound on the value of the nodes in (c) can be derived from consideration of shortest-path lengths on the map, and derive such bounds for these nodes. Remember the cost to get to each leaf as well as the cost to solve it.

e. Now suppose that the tree as given, with the leaf bounds from (d), is evaluated from left to right. Circle those "?" nodes that would *not* need to be expanded further, given the bounds from part (d), and cross out those that need not be considered at all.

f. Can you prove anything in general about who wins the game on a map that is a tree?

5.4 Describe and implement state descriptions, move generators, terminal tests, utility functions, and evaluation functions for one or more of the following stochastic games: Monopoly, Scrabble, bridge play with a given contract, or Texas hold'em poker.

5.5 Describe and implement a *real-time*, *multiplayer* game-playing environment, where time is part of the environment state and players are given fixed time allocations.

5.6 Discuss how well the standard approach to game playing would apply to games such as tennis, pool, and croquet, which take place in a continuous physical state space.

5.7 Prove the following assertion: For every game tree, the utility obtained by MAX using minimax decisions against a suboptimal MIN will be never be lower than the utility obtained playing against an optimal MIN. Can you come up with a game tree in which MAX can do still better using a *suboptimal* strategy against a suboptimal MIN?

5.8 Consider the two-player game described in Figure 5.17.
a. Draw the complete game tree, using the following conventions:
• Write each state as (s_A, s_B), where s_A and s_B denote the token locations.
• Put each terminal state in a square box and write its game value in a circle.
• Put *loop states* (states that already appear on the path to the root) in double square boxes. Since their value is unclear, annotate each with a "?" in a circle.
b. Now mark each node with its backed-up minimax value (also in a circle). Explain how you handled the "?" values and why.
c. Explain why the standard minimax algorithm would fail on this game tree and briefly sketch how you might fix it, drawing on your answer to (b). Does your modified algorithm give optimal decisions for all games with loops?
d. This 4-square game can be generalized to n squares for any $n > 2$. Prove that A wins if n is even and loses if n is odd.

5.9 This problem exercises the basic concepts of game playing, using tic-tac-toe (noughts and crosses) as an example. We define X_n as the number of rows, columns, or diagonals with exactly n X's and no O's. Similarly, O_n is the number of rows, columns, or diagonals with just n O's. The utility function assigns +1 to any position with $X_3 = 1$ and −1 to any position with $O_3 = 1$. All other terminal positions have utility 0. For nonterminal positions, we use a linear evaluation function defined as $Eval(s) = 3X_2(s)+X_1(s)-(3O_2(s)+O_1(s))$.
a. Approximately how many possible games of tic-tac-toe are there?
b. Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account.
c. Mark on your tree the evaluations of all the positions at depth 2.
d. Using the minimax algorithm, mark on your tree the backed-up values for the positions at depths 1 and 0, and use those values to choose the best starting move.
e. Circle the nodes at depth 2 that would *not* be evaluated if alpha–beta pruning were applied, assuming the nodes are generated in the optimal order for alpha–beta pruning.

5.10 Consider the family of generalized tic-tac-toe games, defined as follows. Each particular game is specified by a set S of *squares* and a collection W of *winning positions.* Each winning position is a subset of S. For example, in standard tic-tac-toe, S is a set of 9 squares and W is a collection of 8 subsets ofW: the three rows, the three columns, and the two diagonals. In other respects, the game is identical to standard tic-tac-toe. Starting from an empty board, players alternate placing their marks on an empty square. A player who marks every square in a winning position wins the game. It is a tie if all squares are marked and neither player has won.
a. Let $N = |S|$, the number of squares. Give an upper bound on the number of nodes in the complete game tree for generalized tic-tac-toe as a function of N.

b. Give a lower bound on the size of the game tree for the worst case, where W = {}.
c. Propose a plausible evaluation function that can be used for any instance of generalized tic-tac-toe. The function may depend on S and W.
d. Assume that it is possible to generate a new board and check whether it is a winning position in 100N machine instructions and assume a 2 gigahertz processor. Ignore memory limitations. Using your estimate in (a), roughly how large a game tree can be completely solved by alpha–beta in a second of CPU time? a minute? an hour?

5.11 Develop a general game-playing program, capable of playing a variety of games.
a. Implement move generators and evaluation functions for one or more of the following games: Kalah, Othello, checkers, and chess.
b. Construct a general alpha–beta game-playing agent.
c. Compare the effect of increasing search depth, improving move ordering, and improving the evaluation function. How close does your effective branching factor come to the ideal case of perfect move ordering?
d. Implement a selective search algorithm, such as B* (Berliner, 1979), conspiracy number search (McAllester, 1988), or MGSS* (Russell and Wefald, 1989) and compare its performance to A*.

5.12 Describe how the minimax and alpha–beta algorithms change for two-player, nonzero-sum games in which each player has a distinct utility function and both utility functions are known to both players. If there are no constraints on the two terminal utilities, is it possible for any node to be pruned by alpha–beta? What if the player's utility functions on any state differ by at most a constant k, making the game almost cooperative?

5.13 Develop a formal proof of correctness for alpha–beta pruning. To do this, consider the situation shown in Figure 5.18. The question is whether to prune node n_j, which is a maxnode and a descendant of node n_1. The basic idea is to prune it if and only if the minimax value of n_1 can be shown to be independent of the value of n_j.
a. Mode n_1 takes on the minimum value among its children: $n_1 = \min(n_2, n_{21}, \ldots, n_{2b_2})$.
Find a similar expression for n_2 and hence an expression for n_1 in terms of n_j.
b. Let l_i be the minimum (or maximum) value of the nodes to the *left* of node n_i at depth i, whose minimax value is already known. Similarly, let r_i be the minimum (or maximum) value of the unexplored nodes to the right of n_i at depth i. Rewrite your expression for n_1 in terms of the l_i and r_i values.
c. Now reformulate the expression to show that in order to affect n_1, n_j must not exceed a certain bound derived from the l_i values.
d. Repeat the process for the case where n_j is a min-node.

5.14 Prove that alpha–beta pruning takes time $O(2^{m/2})$ with optimal move ordering, where m is the maximum depth of the game tree.

5.15 Suppose you have a chess program that can evaluate 10 million nodes per second. Decide on a compact representation of a game state for storage in a transposition table. About how many entries can you fit in a 2-gigabyte in-memory table? Will that be enough for the three minutes of search allocated for one move? How many table lookups can you do in the

time it would take to do one evaluation? Now suppose the transposition table is stored on disk. About how many evaluations could you do in the time it takes to do one disk seek with standard disk hardware?

5.16 This question considers pruning in games with chance nodes. Figure 5.19 shows the complete game tree for a trivial game. Assume that the leaf nodes are to be evaluated in left-to-right order, and that before a leaf node is evaluated, we know nothing about its value—the range of possible values is $-\infty$ to ∞.

a. Copy the figure, mark the value of all the internal nodes, and indicate the best move at the root with an arrow.

b. Given the values of the first six leaves, do we need to evaluate the seventh and eighth leaves? Given the values of the first seven leaves, do we need to evaluate the eighth leaf? Explain your answers.

c. Suppose the leaf node values are known to lie between –2 and 2 inclusive. After the first two leaves are evaluated, what is the value range for the left-hand chance node?

d. Circle all the leaves that need not be evaluated under the assumption in (c).

5.17 Implement the expectiminimax algorithm and the *-alpha–beta algorithm, which is described by Ballard (1983), for pruning game trees with chance nodes. Try them on a game such as backgammon and measure the pruning effectiveness of *-alpha–beta.

5.18 Prove that with a positive linear transformation of leaf values (i.e., transforming a value x to ax + b where a > 0), the choice of move remains unchanged in a game tree, even when there are chance nodes.

5.19 Consider the following procedure for choosing moves in games with chance nodes:

- Generate some dice-roll sequences (say, 50) down to a suitable depth (say, 8).
- With known dice rolls, the game tree becomes deterministic. For each dice-roll sequence, solve the resulting deterministic game tree using alpha–beta.
- Use the results to estimate the value of each move and to choose the best.

Will this procedure work well? Why (or why not)?

5.20 In the following, a “max” tree consists only of max nodes, whereas an “expectimax” tree consists of a max node at the root with alternating layers of chance and max nodes. At chance nodes, all outcome probabilities are nonzero. The goal is to *find the value of the root* with a bounded-depth search. For each of (a)–(f), either give an example or explain why this is impossible.

a. Assuming that leaf values are finite but unbounded, is pruning (as in alpha–beta) ever possible in a max tree?

b. Is pruning ever possible in an expectimax tree under the same conditions?

c. If leaf values are all nonnegative, is pruning ever possible in a max tree? Give an example, or explain why not.

d. If leaf values are all nonnegative, is pruning ever possible in an expectimax tree? Give an example, or explain why not.

e. If leaf values are all in the range [0, 1], is pruning ever possible in a max tree? Give an

example, or explain why not.
f. If leaf values are all in the range [0, 1], is pruning ever possible in an expectimax tree?
g. Consider the outcomes of a chance node in an expectimax tree. Which of the following evaluation orders is most likely to yield pruning opportunities?
(i) Lowest probability first
(ii) Highest probability first
(iii) Doesn't make any difference

5.21 Which of the following are true and which are false? Give brief explanations.
a. In a fully observable, turn-taking, zero-sum game between two perfectly rational players, it does not help the first player to know what strategy the second player is using—that is, what move the second player will make, given the first player's move.
b. In a partially observable, turn-taking, zero-sum game between two perfectly rational players, it does not help the first player to know what move the second player will make, given the first player's move.
c. A perfectly rational backgammon agent never loses.

5.22 Consider carefully the interplay of chance events and partial information in each of the games in Exercise 5.4.
a. For which is the standard expectiminimax model appropriate? Implement the algorithm and run it in your game-playing agent, with appropriate modifications to the gameplaying environment.
b. For which would the scheme described in Exercise 5.19 be appropriate?
c. Discuss how you might deal with the fact that in some of the games, the players do not have the same knowledge of the current state.

Chapter 6:

6.1 How many solutions are there for the map-coloring problem in Figure 6.1? How many solutions if four colors are allowed? Two colors?

6.2 Consider the problem of placing k knights on an n×n chessboard such that no two knights are attacking each other, where k is given and k ≤ n2.
a. Choose a CSP formulation. In your formulation, what are the variables?
b. What are the possible values of each variable?
c. What sets of variables are constrained, and how?
d. Now consider the problem of putting *as many knights as possible* on the board without any attacks. Explain how to solve this with local search by defining appropriate ACTIONS and RESULT functions and a sensible objective function.

6.3 Consider the problem of constructing (not solving) crossword puzzles:5 fitting words into a rectangular grid. The grid, which is given as part of the problem, specifies which squares are blank and which are shaded. Assume that a list of words (i.e., a dictionary) is provided and that the task is to fill in the blank squares by using any subset of the list. Formulate this problem precisely in two ways:
a. As a general search problem. Choose an appropriate search algorithm and specify a heuristic function. Is it better to fill in blanks one letter at a time or one word at a time?
b. As a constraint satisfaction problem. Should the variables be words or letters?
Which formulation do you think will be better? Why?

6.4 Give precise formulations for each of the following as constraint satisfaction problems:
a. Rectilinear floor-planning: find non-overlapping places in a large rectangle for a number of smaller rectangles.
b. Class scheduling: There is a fixed number of professors and classrooms, a list of classes to be offered, and a list of possible time slots for classes. Each professor has a set of classes that he or she can teach.
c. Hamiltonian tour: given a network of cities connected by roads, choose an order to visit all cities in a country without repeating any.

6.5 Solve the cryptarithmetic problem in Figure 6.2 by hand, using the strategy of backtracking with forward checking and the MRV and least-constraining-value heuristics.

6.6 Show how a single ternary constraint such as "A + B = C" can be turned into three binary constraints by using an auxiliary variable. You may assume finite domains. (*Hint:* Consider a new variable that takes on values that are pairs of other values, and consider

constraints such as “X is the first element of the pair Y .”) Next, show how constraints with more than three variables can be treated similarly. Finally, show how unary constraints can be eliminated by altering the domains of variables. This completes the demonstration that any CSP can be transformed into a CSP with only binary constraints.

6.7 Consider the following logic puzzle: In five houses, each with a different color, live five persons of different nationalities, each of whom prefers a different brand of candy, a different drink, and a different pet. Given the following facts, the questions to answer are “Where does the zebra live, and in which house do they drink water?”

The Englishman lives in the red house.
The Spaniard owns the dog.
The Norwegian lives in the first house on the left.
The green house is immediately to the right of the ivory house.
The man who eats Hershey bars lives in the house next to the man with the fox.
Kit Kats are eaten in the yellow house.
The Norwegian lives next to the blue house.
The Smarties eater owns snails.
The Snickers eater drinks orange juice.
The Ukrainian drinks tea.
The Japanese eats Milky Ways.
Kit Kats are eaten in a house next to the house where the horse is kept.
Coffee is drunk in the green house.
Milk is drunk in the middle house.
Discuss different representations of this problem as a CSP. Why would one prefer one representation over another?

6.8 Consider the graph with 8 nodes A_1, A_2, A_3, A_4, H, T, F_1, F_2. A_i is connected to A_{i+1} for all i, each A_i is connected to H, H is connected to T, and T is connected to each F_i. Find a 3-coloring of this graph by hand using the following strategy: backtracking with conflict-directed backjumping, the variable order A_1, H, A_4, F_1, A_2, F_2, A_3, T, and the value order R, G, B.

6.9 Explain why it is a good heuristic to choose the variable that is *most* constrained but the value that is *least* constraining in a CSP search.

6.10 Generate random instances of map-coloring problems as follows: scatter n points on the unit square; select a point X at random, connect X by a straight line to the nearest point Y such that X is not already connected to Y and the line crosses no other line; repeat the previous step until no more connections are possible. The points represent regions on the map and the lines connect neighbors. Now try to find k-colorings of each map, for both k=3 and k =4, using min-conflicts, backtracking, backtracking with forward checking, and backtracking with MAC. Construct a table of average run times for each algorithm for values of n up to the largest you can manage. Comment on your results.

6.11 Use the AC-3 algorithm to show that arc consistency can detect the inconsistency of the partial assignment {WA=green, V =red} for the problem shown in Figure 6.1.

6.12 What is the worst-case complexity of running AC-3 on a tree-structured CSP?

6.13 AC-3 puts back on the queue *every* arc (X_k,X_i) whenever *any* value is deleted from the domain of X_i, even if each value of X_k is consistent with several remaining values of X_i. Suppose that, for every arc (X_k,X_i), we keep track of the number of remaining values of X_i that are consistent with each value of X_k. Explain how to update these numbers efficiently and hence show that arc consistency can be enforced in total time O(n2,d2)

6.14 The TREE-CSP-SOLVER (Figure 6.10) makes arcs consistent starting at the leaves and working backwards towards the root. Why does it do that? What would happen if it went in the opposite direction?

6.15 We introduced Sudoku as a CSP to be solved by search over partial assignments because that is the way people generally undertake solving Sudoku problems. It is also possible, of course, to attack these problems with local search over complete assignments. How well would a local solver using the min-conflicts heuristic do on Sudoku problems?

6.16 Define in your own words the terms constraint, backtracking search, arc consistency, backjumping, min-conflicts, and cycle cutset.

6.17 Suppose that a graph is known to have a cycle cutset of no more than k nodes. Describe a simple algorithm for finding a minimal cycle cutset whose run time is not much more than $O(n^k)$ for a CSP with n variables. Search the literature for methods for finding approximately minimal cycle cutsets in time that is polynomial in the size of the cutset. Does the existence of such algorithms make the cycle cutset method practical?