

Bidirectional versions of A^* have been investigated; a combination of bidirectional A^* and known landmarks was used to efficiently find driving routes for Microsoft's online map service (Goldberg *et al.*, 2006). After caching a set of paths between landmarks, the algorithm can find an optimal path between any pair of points in a 24 million point graph of the United States, searching less than 0.1% of the graph. Others approaches to bidirectional search include a breadth-first search backward from the goal up to a fixed depth, followed by a forward IDA^* search (Dillenburg and Nelson, 1994; Manzini, 1995).

A^* and other state-space search algorithms are closely related to the *branch-and-bound* techniques that are widely used in operations research (Lawler and Wood, 1966). The relationships between state-space search and branch-and-bound have been investigated in depth (Kumar and Kanal, 1983; Nau *et al.*, 1984; Kumar *et al.*, 1988). Martelli and Montanari (1978) demonstrate a connection between dynamic programming (see [Chapter 17](#)) and certain types of state-space search. Kumar and Kanal (1988) attempt a “grand unification” of heuristic search, dynamic programming, and branch-and-bound techniques under the name of CDP—the “composite decision process.”

Because computers in the late 1950s and early 1960s had at most a few thousand words of main memory, memory-bounded heuristic search was an early research topic. The Graph Traverser (Doran and Michie, 1966), one of the earliest search programs, commits to an operator after searching best-first up to the memory limit. IDA^* (Korf, 1985a, 1985b) was the first widely used optimal, memory-bounded heuristic search algorithm, and a large number of variants have been developed. An analysis of the efficiency of IDA^* and of its difficulties with real-valued heuristics appears in Patrick *et al.* (1992).

RBFS (Korf, 1993) is actually somewhat more complicated than the algorithm shown in [Figure 3.26](#), which is closer to an independently developed algorithm called **iterative expansion** (Russell, 1992). RBFS uses a lower bound as well as the upper bound; the two algorithms behave identically with admissible heuristics, but RBFS expands nodes in best-first order even with an inadmissible heuristic. The idea of keeping track of the best alternative path appeared earlier in Bratko's (1986) elegant Prolog implementation of A^* and in the DTA^* algorithm (Russell and Wefald, 1991). The latter work also discusses metalevel state spaces and metalevel learning.

ITERATIVE EXPANSION

The MA^* algorithm appeared in Chakrabarti *et al.* (1989). SMA^* , or Simplified MA^* , emerged from an attempt to implement MA^* as a comparison algorithm for IE (Russell, 1992). Kaindl and Khorsand (1994) have applied SMA^* to produce a bidirectional search algorithm that is substantially faster than previous algorithms. Korf and Zhang (2000) describe a divide-and-conquer approach, and Zhou and Hansen (2002) introduce memory-bounded A^* graph search and a strategy for switching to breadth-first search to increase memory-efficiency (Zhou and Hansen, 2006). Korf (1995) surveys memory-bounded search techniques.

The idea that admissible heuristics can be derived by problem relaxation appears in the seminal paper by Held and Karp (1970), who used the minimum-spanning-tree heuristic to solve the TSP. (See Exercise 3.30.)

The automation of the relaxation process was implemented successfully by Prieditis (1993), building on earlier work with Mostow (Mostow and Prieditis, 1989). Holte and Hernadvolgyi (2001) describe more recent steps towards automating the process. The use of pattern databases to derive admissible heuristics is due to Gasser (1995) and Culberson and Schaeffer (1996, 1998); disjoint pattern databases are described by Korf and Felner (2002); a similar method using symbolic patterns is due to Edelkamp (2009). Felner *et al.* (2007) show how to compress pattern databases to save space. The probabilistic interpretation of heuristics was investigated in depth by Pearl (1984) and Hansson and Mayer (1989).

By far the most comprehensive source on heuristics and heuristic search algorithms is Pearl's (1984) *Heuristics* text. This book provides especially good coverage of the wide variety of offshoots and variations of A^* , including rigorous proofs of their formal properties. Kanal and Kumar (1988) present an anthology of important articles on heuristic search, and Rayward-Smith *et al.* (1996) cover approaches from Operations Research. Papers about new search algorithms—which, remarkably, continue to be discovered—appear in journals such as *Artificial Intelligence* and *Journal of the ACM*.

The topic of **parallel search** algorithms was not covered in the chapter, partly because it requires a lengthy discussion of parallel computer architectures. Parallel search became a popular topic in the 1990s in both AI and theoretical computer science (Mahanti and Daniels, 1993; Grama and Kumar, 1995; Crauser *et al.*, 1998) and is making a comeback in the era of new multicore and cluster architectures (Ralphs *et al.*, 2004; Korf and Schultze, 2005). Also of increasing importance are search algorithms for very large graphs that require disk storage (Korf, 2008).

PARALLEL SEARCH

EXERCISES

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) $2 \cdot D(i, j)$; (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 $\sqrt[3]{n!}$ 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.

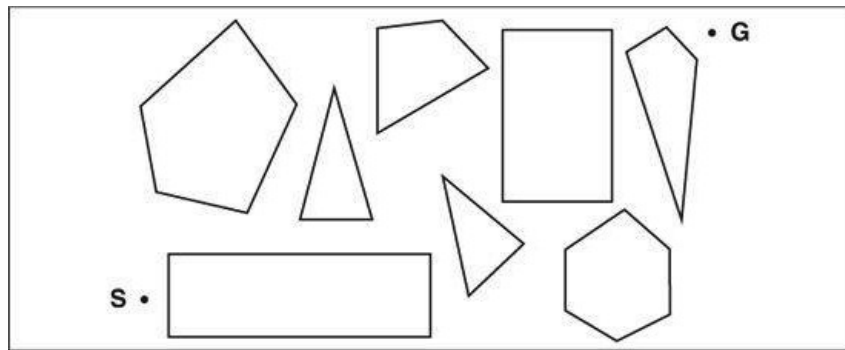


Figure 3.31 A scene with polygonal obstacles. S and G are the start and goal states.

- 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.



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.

- Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal algorithm to explore the entire state space.
- 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.
- 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?
- 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.
- 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).

- Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution. Draw a diagram of the complete state space.
- Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?
- 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 super-composite 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.

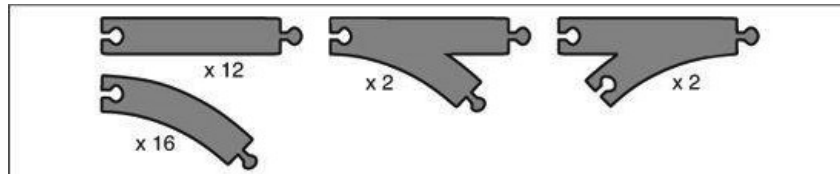


Figure 3.32 The track pieces in a wooden railway set; each is labeled with the number of copies in the set. Note that curved pieces and “fork” pieces (“switches” or “points”) can be flipped over so they can curve in either direction. Each curve subtends 45 degrees.

- Which of the following are true and which are false? Explain your answers.
 - Depth-first search always expands at least as many nodes as A^* search with an admissible heuristic.
 - $h(n) = 0$ is an admissible heuristic for the 8-puzzle.
 - A^* is of no use in robotics because percepts, states, and actions are continuous.
 - Breadth-first search is complete even if zero step costs are allowed.
 - 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.
- Consider a state space where the start state is number 1 and each state k has two successors: numbers $2k$ and $2k + 1$.
 - Draw the portion of the state space for states 1 to 15.
 - 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.
 - How well would bidirectional search work on this problem? What is the branching factor in each direction of the bidirectional search?
 - 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?
 - 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?
- 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.

- Suppose that the pieces fit together *exactly* with no slack. Give a precise formulation of the task as a search problem.
- Identify a suitable uninformed search algorithm for this task and explain your choice.
- Explain why removing any one of the “fork” pieces makes the problem unsolvable.
- 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.

- Show that this algorithm is optimal for general path costs.
- Consider a uniform tree with branching factor b , solution depth d , and unit step costs. How many iterations will iterative lengthening require?
- Now consider step costs drawn from the continuous range $[\epsilon, 1]$, where $0 < \epsilon < 1$. How many iterations are required in the worst case?
- 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.

- Which of the algorithms defined in this chapter would be appropriate for this problem? Should the algorithm use tree search or graph search?
- 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.
- 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.
- Compare your best search agent with a simple randomized reflex agent that sucks if there is dirt and otherwise moves randomly.
- Consider what would happen if the world were enlarged to $n \times n$. 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:

- Breadth-first search is a special case of uniform-cost search.
- Depth-first search is a special case of best-first tree search.
- 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, 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 $w = 2$?

HEURISTIC PATH ALGORITHM

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) .

- What is the branching factor b in this state space?
- How many distinct states are there at depth k (for $k > 0$)?
- What is the maximum number of nodes expanded by breadth-first tree search?
- What is the maximum number of nodes expanded by breadth-first graph search?
- Is $h = |u - x| + |v - y|$ an admissible heuristic for a state at (u, v) ? Explain.
- How many nodes are expanded by A^* graph search using h ?
- 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 $n \times n$ 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.

- Calculate the size of the state space as a function of n .
- Calculate the branching factor as a function of n .
- Suppose that vehicle i is at (x_i, y_i) ; write a nontrivial admissible heuristic h_i 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.
- Which of the following heuristics are admissible for the problem of moving all n vehicles to their destinations? Explain.
 - $\sum_{i=1}^n h_i$.
 - $\max\{h_1, \dots, h_n\}$.
 - $\min\{h_1, \dots, h_n\}$.

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-spanning-tree (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.

- Show how this heuristic can be derived from a relaxed version of the TSP.
- Show that the MST heuristic dominates straight-line distance.
- Write a problem generator for instances of the TSP where cities are represented by random points in the unit square.
- 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 h_1 (misplaced tiles), and show cases where it is more accurate than both h_1 and h_2 (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.

