

**AI Study Guide for Exam 2**  
**October 5, 2006**

See email for why Chapter 3 is here. References to Java are shown in this outline by superscript 1 (<sup>1</sup>). References to material at the course web site such as lecture notes or web links are shown by superscript 2 (<sup>2</sup>).

### **I. Chapter 3**

“Blind” searches through a state space (breadth-first, depth-first, **uniform-cost<sup>1</sup>**, **depth-limited<sup>1</sup>**, **iterative deepening<sup>1</sup>**, **bidirectional**). The first two review Data Structures and Algorithms. You should be able to defend the use of one over the other.

Introduces the basic terminology for searching: **state**, **initial state**, **goal state**, **successor function**, **path cost**, **goal test**. Introduces the difference between the state space and the search tree. Because of possible backtracking, even a finite state space could have an infinite search tree. A **node n** in a **search tree** is: **state** plus path to the node, depth of current node, and  $g(n)$ =path-cost to current node from start. [Sep. 20 lecture] But usually the n below need only be the state.

I won't test the term **belief states** [84] tomorrow. It will help you to understand how Chapter 7 relates to agents. If an agent maintains a belief about its environment in the form of a statement in statement calculus, then states à la Chapter 3 can be described by truth values. If the agent has limited knowledge of its environment, then there may be several possible worlds (models, or world states, or sets of truth values for state variables) that entail that statement. The agent must regard them as a single state, since the agent cannot distinguish among them.

### **II. Chapter 4**

Now we deal with **informed searches**. A search algorithm is **complete** if it finds a solution whenever there is one. It is **optimal** if it has a lowest-path cost.

**Best first** means minimizing some **evaluation function**  $f(n)$  to determine the next steps to take when you are at node  $n$  of the search tree. (If our problem involves reals, not integers, I'll warn you below.)

Let  $h(n)$  be a **heuristic function**, to estimate cheapest path from state  $n$  to goal state. It is **admissible** if it underestimates the true cost. [97] The better the heuristic, the smaller the underestimation. [106]

**A. Theorem:** an admissible heuristic is complete. It may not be optimal. I didn't discuss any conditions guaranteeing optimality, so you don't need to know the theorem: A monotone heuristic function is optimal. [99]

**B. Two important best-first informed searches.** The first does not remember its past; the second does: Greedy<sup>1</sup> and A-star<sup>1</sup>.

36           **Greedy:**        $f(n) = h(n)$   
37           **A\*:**            $f(n) = g(n) + h(n)$

38 Both are  $O(2^n)$  in the worst case. But greedy works and is fast and uses limited memory for  
39 some problems, even though it is in general neither complete nor optimal. A\* is both complete  
40 and optimal, but it is still  $O(2^n)$  in the worst case, and it's a space hog.

### 41 **C. Admissible heuristic functions that you've seen**

- 42       Distance function, in coloring a map of Australia.<sup>1</sup>
- 43       Manhattan distance function, in solving the 8-puzzle.<sup>1</sup>
- 44       Number of misplaced tiles, in solving the 8-puzzle.<sup>1</sup>
- 45       Number of queens under attack, in the 8 queens puzzle<sup>1</sup>, assuming one begins with all 8
- 46               queens on the board and moves them.
- 47       Number of persons still on the starting bank in the missionaries and cannibals problem.
- 48       Can you think of others?

### 49 **D. Continuous state spaces and local search**

50 If we only care about a goal, like placing 8 non-attacking queens without remembering the path  
51 to the solution, we can do a local search (which tracks only the current state, not the path). If  
52 states exist for real numbers, we have an infinite search space. We discussed four search  
53 methods that work well in contexts like these.

#### 54 **1. Hill climbing<sup>1</sup>**

55 Hill climbing uses calculus to convert the infinite problem to a finite search problem. (Look  
56 among the places where the derivative is 0.)

57 Even without an explicit derivative, we can move to the state among nearby states that improves  
58 our evaluation function until we can't improve it anymore. This is called "gradient  
59 approximation." You can see how it is like the greedy algorithm. (For the above definition of  
60 evaluation function, we should probably call this "valley seeking instead of "hill climbing." The  
61 terminology stuck, however.) If you are fortunate to know an equation for the function to  
62 minimize, then use the gradient. [121] If you have only had Calculus I but not Calculus III,  
63 substitute the word "derivative" for the word "gradient" in the above statement.

64 Random restart hill-climbing [114] prevents you from getting stuck in the foothills. If  
65 randomness is introduced in the size of the steps instead of in the starting states, we have Dan  
66 Edwards' algorithm, which is almost simulated annealing. See below for the difference.

#### 67 **2. Linear programming**

68 LP uses linear constraints together with George B. Dantzig's theorem that guarantees that we  
69 only have to check the vertices to find all maxima and minima.

#### 70 **3. Simulated Annealing<sup>1</sup>**

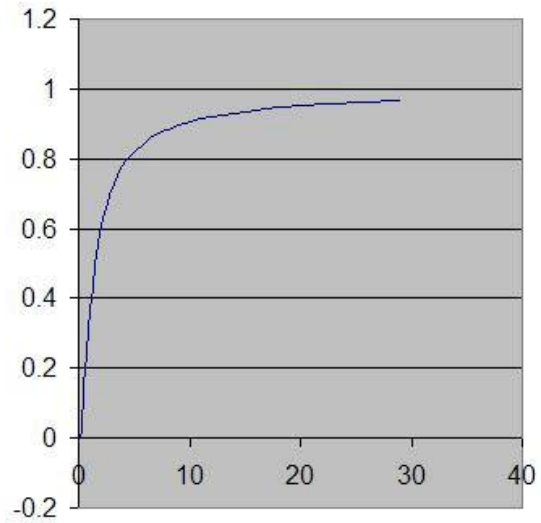
71 Like valley-seeking with gradient approximation, SA moves to a nearby improved point at each  
72 step. Unlike valley seeking, it generates that next move at random. Even if the next random

73 move is not an improvement, there is a certain probability that it will be accepted anyway.

74 As you can see by the algorithm [116], we think of  
75 altitude in the valley as being “energy.” Think of the  
76 “potential energy” of a ball rolling around in the  
77 valley. Call  $\Delta E$  the **decrease** of height. If  $\Delta E$  is a  
78 positive decrease, take the move. If  $\Delta E$  is a negative  
79 decrease (we’re moving uphill), take the move  
80 anyway but only with probability  $P = \exp(\Delta E/T)$ .

81 Look at the graph at the right of  $P = \exp(\Delta E/T)$  for  
82 varying values of  $T$ , assuming that  $\Delta E$  just stayed at  
83  $-1$ . Notice that as  $T$  gets smaller (from right to left),  $P$   
84  $= \exp(\Delta E/T)$  gets smaller too.

85 So<sup>2</sup> why not just use a simpler function of  $T$ , like  $T$   
86 itself for  $P$ , by restricting  $T$  to go from 1.0 down to  
87 0.0? Because the real physical annealing that this is  
88 simulating behaves like  $P = \exp(\Delta E/kT)$ , where  $k$  is a small fixed physical constant ( $\approx 1e-23$   
89 joules/°K), called “Boltzmann’s constant” in honor of the physicist who discovered the  
90 relationship. For our purposes,  $k$  can be 1. It serves only to allow energy/( $k \cdot$  temperature) to be  
91 dimensionless.



Graph of  $\exp(-1/T)$

92 Allowing the agent to move with high probability regardless of whether up or down for large  $T$   
93 allows SA to escape local minima by jumping to another valley. At first, those jumps are  
94 allowed to be big; then they are slowly made smaller and smaller. The rule for how  $T$  gets  
95 smaller is called the temperature **schedule** or cooling schedule. Often  $T$  is just reduced by a  
96 certain fraction each time.  $T_{\text{new}} = \alpha \cdot T_{\text{old}}$ . [This choice was not discussed in class; you can find  
97 it at the Wikipedia article entitled Simulated\_annealing.]

#### 98 4. Genetic Algorithms<sup>1</sup>

99 If the states of a problem can be described as strings, they can be thought of as chromosomes  
100 with the alphabet being genes. By mating and crossover and mutations, children can be formed.  
101 [Figure 4.17 on p. 119 says “child.” It means a pair of children.] By using the evaluation  
102 function (fitness function) on the states, we can keep the fit and discard the unfit, just like  
103 Darwinian evolution claims.

104 Here are my Christian philosophical comments about this process. I could tell by the surprise on  
105 your faces that you couldn’t imagine that the strings could be computer programs, and so genetic  
106 algorithms could be used to write programs, a process called genetic programming. Perhaps one  
107 of you would be willing to tackle that as a project for our course (about which more on Monday  
108 of next week). Why does all this power not give strong support for Darwinian evolution?  
109 Because Darwinian evolution doesn’t explain the design necessary to write  
110 GENETIC-ALGORITHM in the first place!

111 **E. Creativity**

112 Creativity comes in when you are trying to find a good **heuristic**.

113 Creativity comes in when we try to decide what counts as a **state**. That in turn is tied to our  
114 **ontology**. To take the problem of the monkey & banana, if we have many boxes of different  
115 heights, and so we distinguish between the boxes, then we have one problem; but if we treat all  
116 boxes as equivalent (even though they may be of different heights) we have another problem.  
117 Which we solve depends on whether we know that even the box of shortest height will do in  
118 getting the monkey to the banana.

119 Creativity comes in trying different cooling schedules for simulated annealing.

120 Creativity comes in casting a problem in the language of a solution algorithm that you would like  
121 to try out. For example, trying to make the states of a problem be strings that could be mated in a  
122 genetic algorithm, or making the states boolean so that a boolean constraint satisfaction  
123 algorithm can be applied.

124 **F. On-line searching** [§4.5]

125 Although you may omit this section, note that interleaving computation and action is exactly  
126 what Simple-Problem-Solving-Agent on page 61 of Chapter 3 does not do. On p. 61, although  
127 further percepts are received, they are not acted on until the entire sequence of actions calculated  
128 at the first call to the function is executed. They are remembered in a changed state, but the  
129 changed state doesn't affect the actions until the first goal is reached.

130 **G. Expected on exam.**

- 131 1. Know the above content.  
132 2. For a specific problem, be able to identify states in general, a goal state, a heuristic function,  
133 and to give some idea about how the algorithms would work in that problem. I promise easy  
134 problems.

135 **III. Chapter 7.1–4**

136 Because I only got to the line about the Deduction Theorem on my lecture notes<sup>2</sup> for this chapter,  
137 you are not responsible for the terms soundness and completeness on p. 203, nor the symbol  $\vdash$  on  
138 that page, which is not  $\models$ . Nor are you responsible for pp. 208–211. The Wumpus World is a  
139 nice example for me to use in tonight's review session, but any examples on the exam will be  
140 much simpler. For example, a world of colored circles and squares, such as I used at the  
141 blackboard. In that world, no agent is moving about.

---

142 <sup>1</sup> All the search demos that I ran for you in class from relate to the material marked (<sup>1</sup>)  
143 above. You can run them all at once from the Java directory via

144 `c:>java aima.search.demos.SearchDemo.`

145 You can run them separately by running instead NQueensDemo, EightPuzzleDemo, or  
146 CSPDemo in that same package. The corresponding Java source code is found in that Java  
147 directory at src.aima.search.