CPS331 Lecture: Search

last revised January 29, 2014

Objectives:

1. To introduce the concept of a state space
2. To discuss uninformed search strategies
3. To introduce the use of heuristics in searches
4. To introduce some standard heuristic algorithms
5. To introduce criteria for evaluating heuristics

Materials

1. Projectable of beginning of water jug state space
2. Projectable of Figure 4.11 in Cawsey
3. Eight puzzle toy
4. Projectable of search tree for an 8-puzzle
5. Demonstration program: 8-puzzle - with files moderate, 3moves1.8p, 3moves2.8p, hard.8p
6. Demonstration program: Maze - with file lost-freshman
7. Projectable example of a child's maze problem
8. Projectable: solve method of garden puzzle
9. Projectable: chessboard and dominoes problem
10. Projectable: 8-puzzle BFS animation,
11. Projectable: 8-puzzle DFS animation
12. Projectable: maze Branch and Bound animation
13. Projectable Exercise 2 on page 96 as class exercise;
14. Projectable of solution to the above
15. Projectable of Figure 4-6 from Winston 2nd ed
16. 8 puzzle demo program with file hard.8p
17. Maze program demo with lost_freshman
18. Animation of A*

I. **Introduction to State Spaces**

A. One of the hallmarks of intelligence is the ability to solve problems.

Example: the velociraptors in *Jurrasic Park*

In many cases, problem solving can be conceptualized in terms of searching a <u>state space.</u>

B. We will use a problem from the book to illustrate some of the fundamental concepts.

"You are given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a tap that can be used to fill the jugs with water. How do you get exactly 2 gallons of water into the 4-gallon jug?"

1. This is an instance of what is sometimes called a <u>toy problem</u> - a problem that is not of particular interest in its own right, but serves as a vehicle for discussing and/or testing basic concepts. We are using it here to understand some of the basic ideas involved in search. Of course, "real" problems typically are much more complex and involve matters other than just search, but for now we want to focus our attention on search.

2. To solve the problem, we have various <u>operations</u> available to us:

   a) Fill the 4-gallon jug from the tap

   b) Fill the 3-gallon jug from the tap

   c) Pour as much water as possible from the 4-gallon jug into the 3-gallon jug. (This combines two alternatives given in Cawsey)

d) Pour as much water as possible from the 3-gallon jug into the 4-gallon jug (This combines two alternatives given in Cawsey)

e) Empty the 4-gallon jug on the ground

f) Empty the 3-gallon jug on the ground

3. In the course of solving this problem, we pass through a series of <u>states</u>, each of which can be represented by two numbers - the amount of water in the 4-gallon jug and the amount in the 3-gallon jug.

   a) The <u>initial state</u> is { 0, 0}.

   b) The <u>goal state</u> is any state in which the first number is 2. We can represent this as { 2, _ } (_ standing for "don't care").

   c) Of course, in any given state only a subset of the operations are available to us. We say that each operation has certain <u>preconditions</u> that must be satisfied for it to be potentially useful in a particular state.

      (1) For example, the operation "Fill the 4-gallon jug with water from the tap" has the precondition "the 4-gallon jug is not totally full".

      (Note: <u>not</u> "the jug is empty" - if it is partially full, we can still fill it the rest of the way.)

      (2) Preconditions for the other operations?

      ASK

   d) As we analyze the possibilities, we can develop a structure known as a <u>state space</u>. The state space is not given to us at the start - we develop it as we solve the problem.

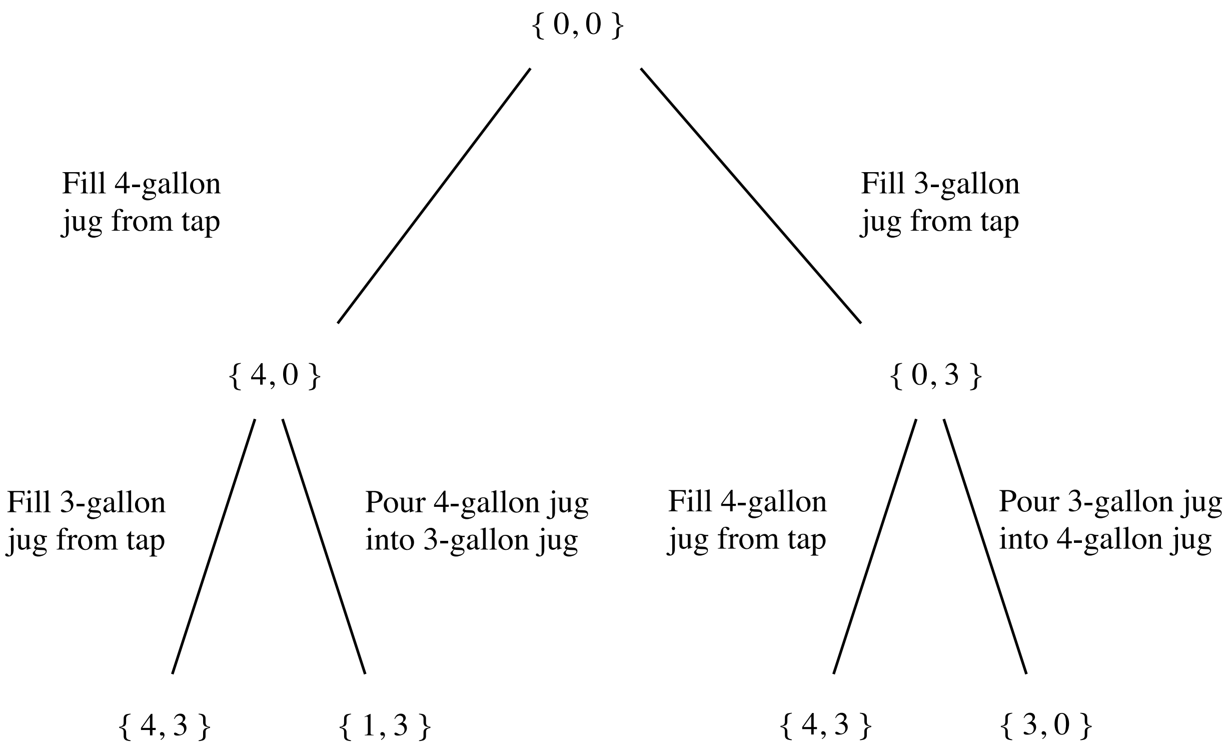
Example: State space for water jug problem after applying two operations: (PROJECT)

Some things to be aware of about a state space

(1) The states are sometimes called "nodes" or "vertices".

(2) The lines connecting states are sometimes called "edges". In a state space, an edge is implicitly one-directional going down the page. (We could label it with an arrow in this direction if we wanted to, but usually don't bother.)

(3) We sometimes speak of a "parent-child" or "ancestor-descendant" relationship between states - e.g. { 0, 0 } is the parent of { 4, 0} and { 0, 3 } and an ancestor of { 4, 3 }, { 1, 3 } etc.


{ 0, 0 }
Fill 4-gallon jug from tap
Fill 3-gallon jug from tap
{ 4, 0 }
{ 0, 3 }
Fill 3-gallon jug from tap
Pour 4-gallon jug into 3-gallon jug
Fill 4-gallon jug from tap
Pour 3-gallon jug into 4-gallon jug
{ 4, 3 }
{ 1, 3 }
{ 4, 3 }
{ 3, 0 }

(4) In some states, there is an available operation that takes us back to the parent or an ancestor state

Example: in the { 4, 0 } state, "Empty the 4-gallon jug on the ground brings us back to the parent { 0 , 0 }.

(5) In developing a state space, we always omit operations that take us back to the immediate parent, and generally omit operations that take us back to an ancestor. (The latter may require us to keep track of the full sequence of ancestors for any given state).

(6) Sometimes, there are two different sequences of operations which take us to the same state.

Example: from { 0, 0 } we can get to { 4, 3 } by filling the two jugs in either order.

Some problem solving strategies choose to discard a state that duplicates one found down another path - though others do not, as we shall see.

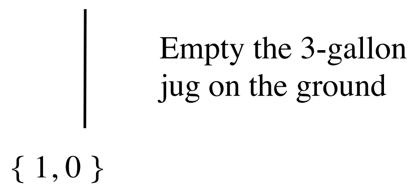
e) In talking about a state space, we use a number of additional terms:

(1) Open states are states at the bottom of the state space -

In the above: { 4, 3 }, { 1, 3 }, { 4, 3 }, { 3, 0 }

(2) We call adding the children of an open state to the state space expanding the state.

Example; if we expanded { 1, 3 }, we would add the following to the state space.



(Where we did not include the possibility "Pour the 3-gallon jug into the 4-gallon jug because this would bring us back to the parent { 4, 0 })

(3) When an open state is expanded, we say that the new children are generated. (For example, in the above we generated {1, 0}).

(4) When an open state has been expanded, we say it is now closed.

In the original diagram: { 0, 0 }, { 4, 0 }, { 0, 3 }

(5) A sequence of states from the initial state to some open state is called a path. A path that leads to a goal state is, of course, a solution to the problem.

If there are multiple solutions possible, often we are interested in finding the solution involving the fewest operations.

PROJECT - figure 4.11 from Cawsey, which shows a solution (and omits other possible paths to save space!)

C. Many problems can also be conceptualized in terms of a states and operations in a similar way.

1. Example: searching for a route from one place to another (as is done by mapquest or a GPS)

a) States?

ASK

b) Operations?

ASK

c) A unique feature of this sort of problem is that there is not only a single initial state (which is always the case), but also a single goal state.

d) Another unique feature of this sort of problem (which we will discuss later) is that there is generally some measure of the “cost” of a solution distinct from the number of operations. (In fact, sometimes there are multiple measures of cost - e.g. in planning a car route, the shortest route in terms of distance may take more time than some other longer route.)

2. Example: playing a 2-player game like chess (or tic-tac-toe for that matter)

   a) States?

      ASK

   b) Operations?

      ASK

   c) A distinctive feature is that the goal is defined not by a specific state or set of states, but by a <u>test</u> - a goal state is one in which the opponent’s King is in checkmate.

   d) A unique feature of this sort of problem (which we will discuss later) is that the one who is analyzing the state space is only in control of half the operations - thus, the analysis must consider <u>all</u> possibilities from states where it is the other player’s turn to move.

3. Example: solving a Sudoku problem

   a) States: partially filled in boards. We are given the initial state - the goal is a state in which every square is filled in without conflicts

   b) Operations: putting a particular number in a particular square. (Which means that, initially, there are scads of them!) The precondition for an operation is that it not conflict with values already filled in.

c) Again, the goal state is defined not by a specific state, but by a test; indeed, the object of solving a puzzle is to actually find out what the goals state is!

d) A unique feature of this sort of problem (which we will discuss later) is that there is a strategy that generally avoids exploring any unnecessary nodes.

II. **Introduction to Search**

A. Earlier, we saw that there are two key issues in symbolic AI. What are they?

ASK

Knowledge representation and search

B. Today, we turn our attention to the second of these: search.

C. The term "search" has a broad range of meanings.

When you hear the word "search", what do you think of?

ASK

1. Sometimes we use the term "search" to refer to finding something. (Example: searching for a lost book, CD, key, or even person.)

2. Sometimes., we use the term "search" to refer to finding information about some topic (Example: Google).

3. Sometimes, we use the term "search" to mean finding a <u>process</u> (a series of steps) that gets us to a goal. (Example: many mapquest searches.)

4. It is this final sense that we use the term in AI.

D. In terms of the model of state spaces we just developed, search is a systematic process of expanding a state space to find a solution (a path to a goal state from the initial state.)

   Search processes can have different goals, though

   1. Sometimes, we are happy to find any solution to the problem.

      Example: Sudoku - though there is only one goal state, there are many different solutions in the AI sense (orders of filling in the cells) that get us there - but we are happy to just find one.

   2. Other times, we want to find the "best" solution in terms of requiring the fewest operations.

      Example: in solving the water jug problem, we probably want to solve the problem in the most direct way possible.

   3. Other times, we want to find the "best" solution in terms of some other cost criterion

      Example: searching for a route between two locations

   4. Other times, we want to find a solution with minimal effort expended in actually doing the search

      Example: often, in going from one place to another, we take a familiar route, rather than expending the effort to figure out whether there might be a shorter or faster route.

E. In developing the concept of search, we will need to use two examples that are a bit more complex than the water jug problem.

1. The eight puzzle.

   (SHOW Toy)

   a) An eight puzzle consists of 8 tiles (numbered 1-8) in a 3 x 3 grid, with one cell vacant. The goal is to rearrange the tiles into numerical order around the outside of the grid - i.e.

      ```
      1 2 3
      8   4
      7 6 5
      ```

      PROJECT: Example search tree for a simple 8-puzzle

      (1) This is a very simple puzzle - requiring only 3 moves for a solution

      (2) The operations can be described in terms of which way the blank space moves - e.g. the first child of the initial state is called “left”

      (3) Any operation which leads back to any ancestor is omitted.

   b) It turns out that there are a variety of different search strategies that can be used to solve a puzzle like this, which we will discuss in this and subsequent lectures

      DEMO: 8-puzzle program with initial state “moderate” - solve using various strategies and comment on cost metrics (number of operations, number of nodes expanded)

      (1) Number of moves required for solution ranged from 5 (best possible) to 997.

      (2) Number of states expanded ranged from 6 to 2075.

(3) While it might be tempting to draw conclusions about the various strategies based on this example, one has to be careful. For different puzzles, very different patterns of results may ensue.

2. A maze

   DEMO with lost-freshman.

   This represents the following scenario: a freshman is searching for a way to get from MacDonald (colored green) to Drew (colored red), following only sidewalks or paths across the quad the freshman has seen other students following. (The numbers shown along the paths are distances measured in smoots. There is a fascinating discussion of the smoot as a unit of measurement which on Wikipedia - just look up smoot!)

   Solve using various strategies and comment on cost metrics (number of operations, distance, number of nodes expanded)

   Observe that, in this case, the solutions with the fewest operations (2) were longer than solutions involving 3 operations (430 vs 170)

III. **Basic Issues Pertaining to Search**

A. Before looking in detail at various search strategies, we need to look at a few common issues.

B. For a given problem, one often has a choice of conducting the search in either of two directions:

   1. One may reason forward, from the data to a solution. This is sometimes called data-directed search.

2. One may reason backward, from the problem at hand to the relevant data. This is some times called goal-directed search.

3. As an example, consider a typical maze problem of the sort you might have played with as a child:

   PROJECT simple maze problem

   a) To use forward chaining, we start with the start square and explore alternatives leading out from it.

   b) To use backward chaining, we start with the goal square and explore alternative paths leading into it.

4. For a given problem, one direction of search may work much better than the other, or may even be the only alternative possible.

   a) For this particular maze example, backward chaining gives an answer much more easily. For other mazes, forward chaining may be better, or both methods may be equally difficult.

   b) Though the techniques we will consider can, in principle, be applied either way, all the examples we will use are based on forward-chaining.

   c) For many problems, backward chaining is ruled out in any case because we don't know exactly what the goal looks like, though we can recognize it when we see it. (Example: Sudoku - the goal is a state in which all the cells are filled in without conflict, but we certainly don’t know what this looks like when we start!)

C. A key problem in search is what is called “combinatorial explosion” - the number of alternatives to be explored may grow so fast as to render carrying the search out impossible.

1. As Newell and Simon pointed out almost any kind of problem solving can be thought of as a search process involving:

   a) A test to decide whether a proposed solution is valid.

   b) A generator that creates possible solutions.

2. For most problems, there is a naive but very impractical way to solve them: generate all possible solutions, and then test each to see if it is what we want, or to find the best among all possibilities. Of course, for any but the smallest problems, this strategy is terribly impractical!

   Example: Recall the "garden tools" problem we used as an example in our lectures on Prolog. Recall the basic approach we took to solving the problem

   PROJECT excerpt from solution to garden puzzle (solve method).

   The basic strategy here is to keep generating possible solutions until one is found that works.

   For this particular problem, how many solutions might we have to consider?

   ASK

   a) There are 5 x 4 x 3 x 2 = 120 ways to permute a list of 120 items. For each of the 120 permutations of the last names list, there are 120 permutations of the bought list.

   b) Therefore, we potentially have to consider 120 x 120 = 144,400 solutions. (Of course, since our solution is probably not the last one we consider, the actual number tried will be less.) Being very fast, this took barely noticeable time. (But try doing it by hand sometime!)

c) Now consider what would happen if we tried a similar problem with more people.

(1) For 10 people rather than 5, there would be 10 x 9 x 8 x 7 x 6 x 5 x 4 x 3 x 2 x 1 permutations of each list = 3,628,800, so there would 3628800^2 = 13,168,189,440,000 possible solutions. This is over 91 million times as many solutions. If the original problem took 1 second to solve, this would be equivalent to over 2 years.

(2) For 15 people, the estimated solution time would be about 25 times the age of the universe.

(3) The fact that the number of possible solutions to a problem tends to grow rapidly (actually exponentially) with the size of the problem is called "combinatorial explosion"

D. One of the hallmarks of intelligence is ability to solve problems requiring some form of search, without falling victim to combinatorial explosion.

"Intelligent" search typically relies on having an intelligent generator that produces proposed solutions to be tested in a way that holds down the number that need to be considered..

E. Search strategies fall into two broad categories

1. Uninformed or weak strategies, that don't make use of any knowledge of the specific problem - i.e. they are general strategies that can be used, in principle, for any problem.

2. Informed or strong strategies, that use knowledge of the problem to constrain the number of possibilities, in an effort to prevent combinatorial explosion.

3. As an illustration of the difference, let's look at the a simple example using the 8 -puzzle.

   a) Demo: 3moves1.8p

      Solve using BFS - an uninformed search

   b) Demo: hard.8p

      Attempt to solve using DFS - another uninformed search. (You will try BFS on the homework)

      Solve using A* - an informed search - using Count heuristic.

      (This heuristic says that we count the number of tiles that are out of their proper place in each state, and prefer moves that reduce this number)

      Demo: Step through solution found

   c) We will focus on weak or uninformed strategies first, and informed or strong strategies next.

F. Finally, it is important to keep in mind that, though a knowledge of search techniques is useful, it is even more important to try to avoid search when possible. Often, good use of knowledge can avoid search altogether.

1. Example: Consider the problem of covering a standard 8x8 checker board with 32 dominoes - where each step in the solution process involves placing one domino.

   PROJECT Problem, Trivial Solution

2. Now consider a harder problem - covering a board missing two corners with 31 dominoes.

   ASK class for ideas on how to solve by search.

It turns out that this problem has a trivial solution - it cannot be done! A domino always covers one black square and one red one, but this board has 32 black squares but only 30 red ones!

IV. **Uninformed Search**

A. A search strategy is a systematic process for developing a state space in quest of a solution to a problem. We will look at several example strategies. Each works with two lists - an open list, which is a list of states that have not yet been expanded, and a closed list, which is a list of states that have already been expanded.

1. Initially, the open list contains just the initial state, and the closed list is empty.

2. The following process is repeated until a suitable solution is found:

   - Remove the first state from the open list
   - Expand it
   - Put the expanded state on the closed list
   - Deal appropriately with any newly generated states that are identical to states already appearing on the open or closed lists
   - Add the newly-generated states on the open list

3. There are quite a number of search strategies, differing in three ways:

   a) Most important: <u>where</u> on the open-list they put the newly generated states.

   b) For strategies aimed at finding the best solution, two other considerations may come into play:

      (1) How they deal with newly-generated states that duplicate states already on the open or closed list.

(2) When to terminate the search process (when a goal state is generated, or when a goal state makes it to the front of the open list.)

B. The strategy known as Breadth-First Search (BFS).

This strategy adds newly-generated nodes to the <u>end</u> of the open list. It discards any generated states that duplicate states already on the open or closed list. It stops when a goal has been generated.

1. Let's work through an example:

   PROJECT Animation of BFS (using 3moves1.8p)

2. An important characteristic of BFS is that it <u>always</u> finds the solution requiring the fewest operations.

   ASK class why

C. The reading for today looked at another uninformed strategy known as Depth-First Search (DFS). It differs from BFS in just one way - it puts newly generated nodes at the <u>front</u> of the open list, rather than the end. Interestingly, that one change can make a huge difference in performance.

1. How would DFS handle the problem we just walked through for BFS?

   ASK

2. In this particular case, the performance of DFS is <u>much</u> worse than BFS.

   DEMO with 8-puzzle program, 3moves1.8p, both strategies

3. There are times, though, when DFS can do much better than BFS in terms of the effort involved in finding the solution.

   DEMO with 8 puzzle program, 3moves2.8p, both strategies
4. Let's work through this second case as an example

   PROJECT Animation of DFS (using 3moves2.8p)

   Both are systematic approaches to exploring all possibilities, but for some problems one is better than the other (and for many problems neither is particularly good!)

5. We can summarize the performance difference between the two searches this way: BFS is "safe" but can be slow; DFS is "risky", but can do very well.

D. Class Exercise: Problem 2 on page 96 of the book

PROJECT

1. Formulate problem as a search problem.

   a) We can represent a state by the bank where farmer, dog, rabbit, and lettuce are.

   b) The initial state is EEEE. The final state is WWWW

   c) The possible operators, with preconditions, are:

      (1) Farmer row across the river alone

      (2) Farmer rows across the river with the dog. Precondition: the farmer and the dog are on the same bank.

      (3) Farmer rows across the river with the rabbit. Precondition: the farmer and the rabbit are on the same bank.

(4) Farmer rows across the river with the lettuce. Precondition: the farmer and the lettuce are on the same bank.

2. Solve using DFS, discarding unsafe and cycle states (do as class exercise) (Solution on next page)

   PROJECT

E. While DFS and BFS, and variants of DFS may be appropriate for some problems like simple versions of the 8 puzzle, they are less useful for some other kinds of problems where the steps in the solution have different costs - e.g.. the maze example we looked at earlier.

1. Example: Run Maze, load lost-freshman.maze.

   Solve using BFS - the solution found is clearly the best possible in terms of the number of moves, but not in terms of the total distance

2. For situations like this, a variant of BFS can be used called Branch-and-bound. In brief, what this strategy does is to pursue the path that has the lowest costs so far, rather than the path involving the fewest moves.

3. The key is that Branch and Bound keeps the open list in a sorted state, so that the open state having the least cost so far is always at the front. (Ties can be broken by the order discovered, though this doesn't really matter.) Also, Branch and Bound doesn't stop until an a goal state is at the front of the open list.

   Demo Branch-and-Bound animation

V. **Heuristic Search**

A. The basic idea.

1. Thus far, we have considered two basic search techniques (DFS and BFS) and a variant that is essentially "blind": they explore alternatives in some fixed order without regard to their likelihood of success. We call such search techniques UNINFORMED.

2. Though such search techniques will always ultimately find the solution to a problem if one exists, they can be terribly inefficient for even moderate size problems. In most cases, we need to make use of an INFORMED search - one that relies on heuristic knowledge about the specific problem to focus the search on the "right" region of the search space.

    a) A heuristic is a "rule of thumb" that gives some sense of what alternative among those available is most likely to lead to success.

       Example: in trying to get to an unfamiliar destination, we often use the heuristic "drive in the general direction of the destination"

    b) An important characteristic of a heuristic is that it is not guaranteed to always identify the best alternative; rather, a heuristic gives a good answer most of the time.

3. The key idea behind a heuristic technique is to make use of knowledge about the problem to estimate how close a given state is to the final solution, and then prefer steps that minimize this

    a) For example, for the 8 puzzle there are quite a few heuristics one might use - some better than others.

(1) One heuristic measure of a state is the count of the number of tiles that are out of place. As we saw last time, using this heuristic makes it possible to solve puzzles that could not be solved (in practical time) using an uninformed search like BFS.

(2) A better one is the sum of distances out of place for the tiles.

```
Example:2 1 6
        4   8
        7 5 3
```

Count is 7 - the only tile in the correct place is 7
Sum of distances is 1 + 1 + 3 + 2 + 2 + 0 + 1 + 2 = 12

(This is the one we will use for our examples)

b) Note the distinction between the goodness of a SOLUTION and the goodness of a STATE.

(1) The goodness of a solution is related to the cost of actually carrying it out.

(2) The goodness of a state is related to the ESTIMATED cost of a solution from that state. (E.g., for the distance heuristic, any solution from a given state must each tile a number of times equal to the distance it is out of place - but usually quite a few more!)

(3) We will shortly see an example where preferring to go through a state that looks good (based on its heuristic estimate) may actually lead us to a much less than optimal solution.

B. We will consider three subtopics concerning heuristics.

   1. A heuristic search techniques for finding ANY solution to a given problem

   2. A heuristic technique that minimizes both search effort and execution effort.

   3. Criteria for measuring the "goodness" of a heuristic.

C. Best-first-search: an informed variant of BFS

   In best-first search, after each expansion of a node, we reorganize (sort) the open list, and choose to expand the best node of all the ones that are open.

   1. This means that bad estimated cost nodes are unlikely to be expanded, except as a last resort; but we do not ever totally reject a possible solution - even if it doesn't look good - the way the other two techniques do.

   2. Of course, the complete reorganization of the list after each expansion is more computationally-expensive than the other methods.

      DEMO on hard.8p hard problem, using distance heuristic.

      Note that the solution that is found now takes more work to find, but is much better, though still far from optimal).

D. There is a variant of DFS known as hill-climbing that the book discussed, but that we won't discuss here. (However, we will bump into a different form of this when we talk about neural networks)

E. In the version of Branch and Bound we developed in our discussion of uninformed methods, the only criterion for solution selection was actual accumulated cost so far. We could improve the search if we added an estimate of the remaining cost to obtain an estimated total cost (cost so far + estimate), using that as the basis for ordering.

   This combination of Branch and Bound and Best First yields an algorithm know as A*, which is a widely-used AI search algorithm.

   At each step we order the nodes based on total cost, not just cost so far. We keep going until a goal is at the front of the open list.

   Demo: Project Maze with lost-freshman. Assume we use the following estimates of remaining cost (essentially an “as the crow flies” heuristic) - considering, for simplicity, only the nodes that actually show up in our search

   Barrington 80
   Roosevelt 90
   Frost 240
   MacDonald 155
   Emery 110
   Jenks 40
   KOSC 220
   Chase 210

   Run through animation

F. Criteria for choosing a good heuristic

   1. Naturally, our estimate of remaining cost will not be exactly correct. To preserve the guarantee of an optimal solution when using A*, it is important that we use an under-estimate of the remaining cost. (Where by under-estimate we mean one that is certain to be <= the true cost.)

a) This is necessary and sufficient to guarantee we will still find the optimal solution.

b) To see why this must hold, consider our maze problem again:

   PROJECT maze with lost-freshman animation - second slide (just after expanding MacDonald)

   Suppose our estimate of remaining distance for Emery were 400 (which is obviously way too big). Then at some point, after expanding KOSC, Frost, and Phillips) we would have the following (among other) on the open list:

   Chase (estimated cost (210+210) 420)
   Emery (estimated cost (50+400) 450)

   At this point, we would expand Chase and find a suboptimal path to Drew

c) To show that it is sufficient, suppose that A* finds a suboptimal solution. This would mean that the a goal node representing a suboptimal solution got to the front of the open list, which in turn would mean that its total cost is <= the estimated total cost of any other state on the open list. But since we are using an underestimate, the estimated cost of any state on the open list must be <= the true cost of any solution passing through that state, which violates our assumption that the solution we found is suboptimal (assuming, of course, as would reasonably be the case, that all costs are positive)

d) For a given problem, a heuristic that always yields an estimate of remaining cost that is <= the true cost is called an <u>admissable heuristic</u>. For our example problem, the "as the crow flies" distance heuristic is admissible, since it is always <= true path length.

   Use of an admissible heuristic is necessary to preserve the guarantee of finding an optimal solution.

2. However, the less the heuristic underestimates the better.

   a) Example: a heuristic that is guaranteed to be an underestimate is to always use an estimate of 0. This, of course, degenerates to a totally uninformed search.

   b) The ideal heuristic would be one that gives the exact value of remaining cost. This would be a totally informed search - but is seldom possible.

   c) Often, we have several candidate heuristics which are neither totally uninformed nor totally uninformed. In this case, we want to choose the most informed of the candidates.

      We say that a heuristic h1 is more informed than a heuristic h2 if h1 >= h2 for all nodes.

      Example: For the 8 puzzle, we one heuristic is the raw count of the # of tiles that are out of position. This is clearly admissible, since each of these tiles must be moved at least once.

      We used a somewhat better heuristic: the sum of the distances tiles are out of place (the Manhattan distance). This is still admissible, and is more informed since it is always >= our raw count heuristic.

      DEMO: hard 8 puzzle using A* with Count and Distance heuristics

      (1) Both find an 18 move solution (since both are admissible).

      (2) But using Count involves expanding a lot more states

      (3) Actually, we can do even better using some other possibilities:

         - Try each - note that some are much better, but others are worse (including one worse than Count)

         (We will explore these on homework.)

3. Though only an admissible heuristic guarantees that our search will find an optimal solution, there are times when a non-admissible heuristic (i.e. one that sometimes overestimates) may yield close to optimal results while also helping to minimize search effort. The last heuristic we demonstrated (Seq x 3 + distance) is such a heuristic

   a) The sequence score (which is listed by itself as well) is arrived at by going around the outside, counting a 2 for every tile not followed by its proper successor and a 0 otherwise. (Note: if a tile is followed by a blank and then its proper successor - e.g. if the top row is 1 _ 2 - then count this as a 0; otherwise count it as a 2. (In effect, we skip over the blank and look at the very next tile.) Finally, if there is a piece in the center, count it as 1.

      Example: The Sequence score for

      ```
      2 8 3
      1 6 4
      7   5
      ```

      is 2 followed by 8 - count as 2;
      8 followed by 3 - count as 2;
      3 followed by 4 - count as 0:
      4 followed by 5 - count as 0
      5 followed by 7 - count as 2
      7 followed by 1 - count as 2
      1 followed by 2 - count as 0
      tile in middle - count as 1:

      total = 9

   b) The sum of distances out of place is something we are already familiar with. For example, for the puzzle we just looked at, the sum of distances is 1 + 2 + 0 + 1 + 1 + 0 + 0 + 0 = 5

c) So the total using sequence x 3 + distance is 9 x 3 + 5 = 32

d) However, this heuristic is not admissible. For example, the following puzzle has a 1 move solution (move 8 left one square):

```
1 2 3
  8 4
7 6 5
```

However, for this configuration the sequence score is 1 and the distance score is 1, yielding an estimate of 4, which is clearly an overestimate.

e)Nonetheless, the heuristic can produce good results, as we showed when using it with a hard puzzle.

VI.**Conclusion**

A. The use of a good heuristic frequently makes the difference between a problem that is not practically-solvable and one that is.

In practice, then, uninformed algorithms like BFS DFS and its variants, and Branch and Bound is usually useful only for problems for which a good heuristic does not exist. (In fact, we discussed them largely to set the stage for heuristic algorithms).

B. The heuristic algorithm that tends to be especially useful is A*. There are also some cases where the hill climbing heuristic is useful.

C. However, it appears that the intelligence really resides more in identifying the heuristic - rather than in the actual search process that uses it!

D. As we shall see in a couple of days, even better than using a good search algorithm is finding a way to solve the problem without needing to do search at all, or only sparingly. (Recall the defective checker board problem.)