

Purpose: To introduce the concept of search and methods for uninformed search

Materials

1. Projectable example of a child's maze problem
2. Projectable of farmer, wolf, goat, cabbage problem (text 658, 659) plus partial state space (p 660)
3. Projectable of example 8-puzzle state space (p. 90)
4. Projectable of BFS for example 8-puzzle problem (p. 103)
5. Projectable of DFS with depth cutoff of 5 for example 8-puzzle problem (p. 105)
6. Demonstration program: 8-puzzle - with files Luger, 3movesdfs, 3movesdfs-2
7. Handout on search methods

I. Introduction

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

ASK

Knowledge representation and search

B. Today, we turn our attention to the second of these. As the text notes, many AI problems involve some sort of search through a state space in order to find an acceptable solution to a problem - or perhaps the best solution among many possible ones.

Examples:

1. Puzzles and games. At any point, we are faced with a choice among several possible moves; generally, the "correct" move is not obvious.
2. Planning a series of actions to accomplish some goal. At any point, we are faced with a choice among several possible actions.
3. Exploring alternative interpretations of a sentence in natural language processing. At various points, we may be confronted with two or more possible ways of interpreting a word or phrase.
4. Expert systems. Given a specific problem, the expert system must single out the relevant knowledge to apply.

C. 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.
- 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 most of the techniques we will consider can, in principle, be applied either way, we will develop our examples using forward chaining from the given data to the goal.
 - 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: chess - the goal is any state in which the opponent's king is in checkmate).
5. We will say more about this issue when we consider expert systems later in the course.

D. There are a considerable variety of search techniques available, each of which may be good for some problems but bad for others.

1. Considerable thought should go into selecting a search technique if the space to be searched is large, since search is an inherently exponential problem, but efficiencies of various techniques can vary widely.
2. There is no one best search technique - only techniques that are best for specific problems.
3. However, though a knowledge of search techniques is important, it is even more important to try to avoid search when possible. Often, good use of knowledge can avoid or at least minimize search.

It is at this point that we meet an important distinction between what are sometimes called *weak methods* and *strong methods*.

a) A weak method is one that uses a search approach that does not make use of knowledge about the problem.

(1) A good example of such a method is the one used by an early AI system called General Problem Solver (GPS)

(2) As the name implied, GPS attempted to be able to solve any problem, by using a search technique called “means ends analysis” that was independent of the specific problem being solved.

(3) GPS turned out to be a failure for most problems.

b) A strong method is one that makes use of heuristics based on the specific problem. We will look at some examples of this in the next lecture on heuristic search.

II. The States Plus Operators Model of Search

A. As a preliminary point, we note that many AI problems can be described in terms of a model based on STATES and OPERATORS, with search being used to select the next operator to apply at each step along the way.

1. We will illustrate this first with the problem of the farmer, wolf, goat, and cabbage used in the text.

PROJECT excerpts from pages 658 and 659 of text

- a) The STATE captures the situation at a particular point in time during the course of moving toward a solution to the problem.

Ex: In the farmer, wolf, goat, and cabbage problem the state would be some representation of the current position of the 4 entities the problem is concerned with - for example, something like state(farmer-position, wolf-position, goat-position, cabbage-position) - where each position is either e or w for east bank and west bank respectively. (We don't need to model the boat position explicitly, since it must be where the farmer is)

Clearly, there are 16 possible states since each entity can be on either of the two banks of of the river. (However, 6 of the 16 states are ruled out by the requirement that the goat cannot be left with either the wolf or the cabbage if the farmer is not present)

- b) OPERATORS are possible actions that may be taken to alter the state.

Ex: In the farmer, wolf, goat, and cabbage problem the following operators are available (though not all are applicable in a given state if the farmer and the thing taken with him are not on the same side of the river):

- move farmer across river
- move farmer plus wolf across river
- move farmer plus goat across river
- move farmer plus cabbage across river

(Note how the operators implicitly capture one of the rules of the problem - namely the limit on the capacity of the boat.

- c) The STATE SPACE of the problem is the set of all possible states. In general, the state space is a directed graph, with the nodes of the graph corresponding to the possible states and the arcs leaving a given state corresponding to the operators that are legal in that state.

Ex: The book gives a partial state space for the farmer, wolf, goat and cabbage problem

PROJECT page 660

Observe that this diagram is not complete - though it is close - and includes some states that violate one of the problem's safety rules. (E.g. state(e, e, w, w) has the goat and cabbage left alone on the west bank.) One of the states at the bottom of the graph is, in fact, just two moves away from a solution to the problem.

ASK class to identify and complete it

Note, however, that this is an unusually small problem. Usually, it is neither possible nor desirable to explicitly construct the entire state graph. (Consider, for example, the size of the total state space for a game like chess. For any given game, only a portion of this would be applicable.)

- d) When a problem's state space is modeled as a graph, we can think of the task of solving the problem as that of finding a path through the graph from a specified START STATE to a GOAL STATE which is either a specific, specified state or some state which satisfies a certain criterion.

Ex: In the farmer, wolf, cabbage and goat problem there is a single goal state which satisfies the problem's requirements. However, if we modeled a chess game as a state space we would accept as a goal state any state in which the opponent's king is in checkmate.

- e) In general, finding a path which solves the problem may involve a SEARCH process in which we must make choices along the way as to which direction to pursue.

Ex: The farmer, wolf, goat, and cabbage problem does not illustrate this very well; assuming that we adopt the rule of never going back to a state we have just been in, there is only one place where we have a choice between two possible moves that do not lead to an unsafe state, and it turns out that either choice leads inexorably to success.

Such is not generally the case, of course - hence our need to study search techniques.

- f) Of course, for most interesting problems we will not have even a partial state space available to us ahead of time. Rather, we will construct the state space as we conduct the search, adding to it each time the states that are reachable from the current state.

I used the farmer, wolf, goat, and cabbage problem for an initial example because it would be easily possible to exhibit the complete search space - however, the fact that this is possible means that this isn't an interesting problem!

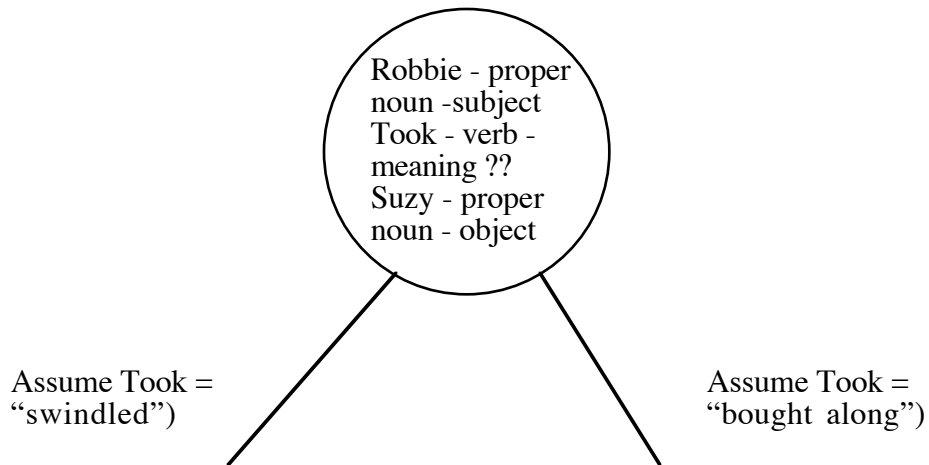
2. In general, the states + operators model is directly applicable to the broad category of "planning" problems where the goal is to derive a set of steps to transform the existing state of a system into a new state. Such problems arise frequently in connection with robotics, as well as being the focus of various puzzles. We will see this model again and again.
3. The states + operators model can also be applied to "interpretation" problems, of the sort which arise in natural language processing or interpreting visual data. Here, the states might be partial solutions to the interpretation problem, with the operators being the fixing of another aspect of the interpretation.

Ex: Suppose we were trying to interpret an English text which included the following sentence: "Robbie took Suzy". assume it is already determined that Robbie is a proper noun, the subject of the sentence, and that Suzy is a proper noun, the direct object of took. "Took" can be interpreted in at least two ways:

- took as in swindled
- took as in brought along

(e.g. if the next sentence were "Suzy sued Robby" the first interpretation is likely, while if the previous sentence were "The ball was last Saturday night", then the second is more likely.)

However, it might be necessary to tentatively assume one alternative before moving, in which case we would have the following node in our state graph



4. For our next examples, we will use a planning problem - the eight puzzle. As you read in the book, 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

```

- a) Here, the states are possible arrangements of the tiles.
- b) The operators are the valid moves in a given state. If the blank tile is at the center of the grid, then four moves are possible - there are four applicable operators; if it is at the middle of one of the sides, three moves are possible; if it is at a corner, two moves are possible.

PROJECT Example of state space for a specific problem

B. All search techniques can be regarded as traversing a graph composed of nodes and edges.

1. The graph will be a state space in which the nodes are states and the edges are operators, as we have seen.
2. Since most search spaces are large, the entire graph will not generally be constructed. Instead, we will start out with an initial node, a rule for generating possible successor nodes, and a criterion for success.

- a) If the successors of a node have not been generated, we call it an open node.
- b) Once the successors of a node have been generated, we call it closed.
- c) We call the process of generating the successors of a node expanding the node.

3. Generally, when we expand a node we add its successors to a list we call the "open list" in some order depending on the search method.

Example: suppose we set out to solve the following eight puzzle:

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

This has a three-move solution. What is it?

ASK

Our initial open list is

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

When we expand this node, we get the following three successors:

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

At this point, our original node is taken off the open list, and we get a new open list consisting of its three successors **IN SOME ORDER**. (The order shown above is the one that we get if we examine possible moves in the order move blank left, up, right, down, which is the order used for the examples in the book, though not the order in the diagram we looked at earlier! Note that the "move blank down" operator is not applicable in this case.) (The way we order the successors turns out to be a key distinction between search methods).

4. In many cases, one issue we must deal with is that our search space is a graph, not a tree - that is, we must deal with the possibility that expanding a node may generate a successor that has also been generated by a previous expansion.

a) Example: Given our initial eight puzzle, the following sequence of moves can be repeated any number of times, each time bringing us back to our starting state:

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

b) In such cases, we may check each newly generated node to be sure it has not been seen already, lest we waste effort or (if the graph is cyclic) go into an infinite loop. In this way, we effectively convert our state space to a tree.

C. The general pattern for a search is then:

```

create an open list containing only the start state
(optional) create an empty closed list
while the open list is not empty
    select next open node from the list
    (optional) add it to the list of closed nodes
    if it is the goal, then
        exit successfully
    else
        generate its successors
        add the successors (or those not already seen) to the open
        list in some method-specific way
exit in failure

```

What distinguishes various search strategies is largely how we go about adding the successors of the node being expanded to the open list.

III. Classification of Search Techniques

A. Search techniques fall into three broad categories, based on the desired outcome of the search.

1. Techniques which yield A solution - without any promise of optimality. (In this case, we could exit as soon as a goal is encountered in the “generate its successors” step - we don't have to wait until it makes it to the front of the open list)
2. Techniques which yield THE BEST solution of all possible solutions - where the definition of "best" may be the smallest number of moves, or the shortest overall path, or some other more complex criterion.

Note that we must distinguish the cost of PERFORMING THE SEARCH and the cost of CARRYING OUT THE SOLUTION. A search that finds a “best solution” (one that is lowest cost to carry out) may entail more effort in searching.

3. Techniques which are specialized for situations like two-player games in which there is an adversary working against one's goal.
4. We consider only the first category of searches today; the other two will come later.

B. Search techniques can also be classified as UNINFORMED (BLIND) or INFORMED

1. An uninformed search does not rely on knowledge about the problem domain to control the search. As a result, such searches tend to be very expensive.
2. An informed search uses heuristic knowledge (“rules of thumb”) to control the search order. When suitable heuristic knowledge about the problem is available (not always the case), the use of heuristic search can make the difference between a process that is impractical to carry out and one that is practical.
3. We will consider uninformed searches today, and heuristic searches later.

IV. Basic Blind Search Techniques

A. Breadth First Search

1. BFS expands all the nodes on one level before expanding any on the next level down. This is achieved by making the open list a FIFO queue. Nodes are removed from the front of the queue and expanded, and newly generated nodes are inserted at the rear of the queue. Initially, it is loaded with the start node; expanding it causes the queue to contain the expansion of the level one start node - i.e. all the level two nodes. Each of these is expanded in turn, filling the queue with all the level three nodes etc.

Example: BFS applied to an 8-puzzle problem

PROJECT tree from page 103 in book

DEMO: Run with computer simulation - file Luger

Note that simulation keeps expanding nodes until a goal comes to the front of the queue - so goal is 46th node considered after taking out discards (though it was in the queue earlier than this). [The program reports expanding 45 nodes because the goal does not need to be expanded!]

2. DISTRIBUTE HANDOUT ON SEARCH METHODS

DISCUSS ALGORITHM

3. Observe that BFS is guaranteed to always produce the best possible solution (in terms of minimum number of moves).
4. However, BFS is an UNINFORMED method. In general, it tends to do a lot of unnecessary work. If the average branching factor at a node is b , and the goal is found at a depth d , then BFS requires:
 $O(b^d)$ time
 $O(b^d)$ space.
5. In a planning problem where the expense of computing a plan may be much less than that of carrying it out, doing the extra work to get the best solution may be worthwhile. (However, if "goodness" of a solution is measured by some criterion other than fewest total steps, this may not help.)

B. Depth-first search

1. Basic principle: Expand the current node, then select an arbitrary successor to become the current node (saving the rest for possible backtracking later). That is, keep moving forward as fast as possible.
2. If the current node has no successors that we haven't already been expanded, then backtrack to the most recently expanded node that still has open successors, and choose one of these to become the current node.

Example: DFS applied to our example eight puzzle - same assumption as BFS. Note that it fails to find a solution.

Another example - using a simpler puzzle (file 3movesdfs) - SHOW

3. This can easily be achieved by making the open list a stack - i.e. newly generated nodes are placed at the front so that they are selected for expansion first.
4. Though DFS can yield a solution quickly, pure DFS is rarely a good procedure - especially if the search space is such that an early mistake can lead into a large but fruitless subtree.

To see this, consider what would happen if we used DFS to solve the following puzzle, which is similar to the one we've been using, with the same order of expansion:

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

a) This also has a three move solution (ask).

b) DEMO with program file 3movesdfs-2

Show solution it found

5. One improvement that can be made to DFS is to impose a limit on the depth to which a search down a subtree can go before it is cut off. (This supposes that one has an up front idea of the length of the path being sought.) For example, if it is known (or strongly conjectured) that a given problem can be solved in at most 6 steps, then whenever a node that is 6 levels down from the root is reached without finding a solution the search can backtrack instead of expanding the node.

DEMO: rerun with depth limit of 3 (in preferences)

DEMO: Show transparency of problem from Luger using depth limit of 5; rerun program with depth limit of 5

6. DFS takes

$O(b^d)$ time (with a constant of proportionality somewhat less than that for BFS)
only $O(b*d)$ space

However, its performance is much more unpredictable than that of BFS. While in some problems it can arrive at a solution very quickly, in other problems it can miss a good solution and discover a poor one instead, or discover no solution at all if the graph is infinitely deep.

7. DISCUSS algorithm in handout

C. DFS with Iterative Deepening

1. An attractive variant of DFS that builds on the notion of a depth limit is DFS with Iterative Deepening. It allows the use of a depth limit even though nothing may be known about how long the path to a solution will actually be.
 - a) The idea is to first conduct a DFS with a small depth limit - say 3 or 4.
 - b) If this search fails to find a solution, then start the entire search over, but increase the depth limit by 1. Keep repeating this process (increasing the depth limit by 1 each time) until a solution is found.
 - c) At first glance, this strategy may appear very wasteful, since all of the work of the previous search is repeated on each new search. But this turns out to not be a major problem, because with exponential growth level, so the repeated work forms only a small fraction of the total effort of each new search.

Example: For the eight puzzle, each state has 2-4 successors (with at least one being a duplicate of its parent.) If we assume an average of 2 valid successors per state, then on each new search about 50% of the effort is a repeat of work previously done.

DEMO: Rerun 3movedfs-2 (which found a solution requiring 29 moves) with iterative deepening

Rerun Luger problem with iterative deepening. (Compare nodes expanded with previous demo where depth limit was set to 5 to begin with)

2. Like BFS and pure DFS, this search takes $O(b^d)$ time - though the constant of proportionality is greater. Like pure DFS, it takes only $O(bd)$ space. Further, like BFS it is guaranteed to find the solution with the fewest number of steps, and cannot go into an infinite loop (as long as some goal state exists).

3. DISCUSS algorithm in handout

D. Another variant is called iterative broadening, in which we keep only a fraction of the children of each node, and increase the fraction if the search fails to find a solution. (This can be used with either DFS or BFS). This method can be very beneficial in cases where there are many possible solutions, but where it is also possible to make a bad decision early in the search that dooms a branch to failure.

E. In our next lecture, we will discuss search algorithms that make use of heuristics to either find a solution more quickly, or to find a better solution, or both.

V. And-Or Trees

A. Sometimes, the search space has a structure called an and-or graph.

An and-or graph consists of two kinds of nodes - and nodes and or nodes.

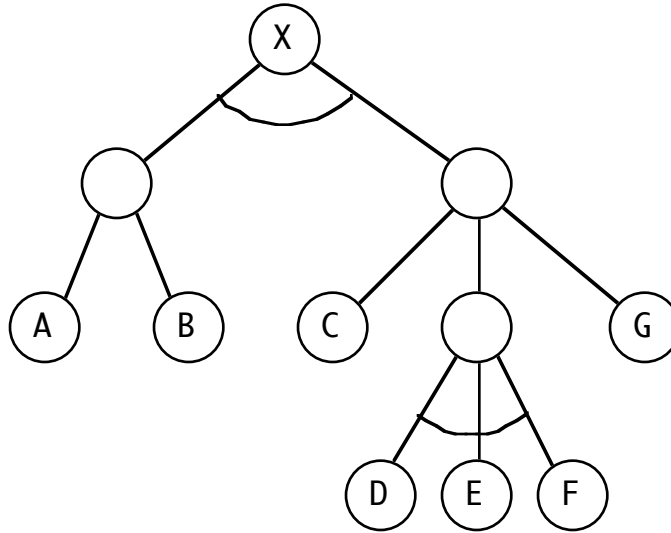
1. An or-node represents a situation where finding a solution down any path emanating from the node will suffice.
2. An and-node represents a situation where one must find a solution down all paths emanating from the node. (Or sometimes a subset of the paths)
3. The nodes in the graphs we have been considering thus far have all been or nodes.

B. A complex propositional or predicate calculus formulas can be represented by an and-or graph.

Example: Suppose we have the following WFF:

$$(A \cup B) \cap (C \cup (D \cap E \cap F) \cup G) \supset X$$

The “proof graph” (tree in this case) for the goal X looks like this:



where the goals with arcs are and nodes and require that all subtrees be proved, while the goals without arcs are or nodes and require that any subtree be proved. (Thus, the proof search procedure can consider these as alternatives in some order based on the search algorithm.) [Actually, it is possible to have an and node where one is required to prove some subset of the total number of branches - e.g. 2 out of 3. We haven't considered this possibility here.]

Note that the search algorithms we have considered have all assumed that the proof graph is composed entirely of or nodes.

C. We will look at and-or graphs (actually trees) again in conjunction with search specialized for game playing.