

Module 1

Introduction – Definitions – AI application areas – Example problems- Problems and problem spaces - Problem characteristics – Problem solving by searching, Searching strategies – Breadth first search, Uniform cost search, DFS, Depth – Limited search, Bi-directional search – Constraint satisfaction search.

Artificial Intelligence

(ref: AI by Rich & Knight)

Artificial intelligence is the study of how to make computers do things which, at the moment, people do better.

Much of the early work in AI focused on formal tasks, such as game playing and theorem proving.

Some of the game playing programs are

Checkers playing program,

Chess.

Some of the theorem proving programs are

Logic theorist,

Galernter's theorem prover.

Another area of AI is common sense reasoning. It includes reasoning about physical objects and their relationships to each other as well as reasoning about actions and their consequences.

Eg.. General problem solver (GPS)

As AI research progressed, new tasks were solved such as perception (vision and speech), natural language understanding and problem solving in domains such as medical diagnosis and chemical analysis.

Perception

Perceptual tasks are very difficult because they involve analog signals.

Natural language understanding

The problem of understanding spoken language is a perceptual problem and is hard to solve. But if we try to simplify the problem by restricting it to written language, then also it is extremely difficult. This problem is referred to as natural language understanding.

In addition to these, many people can perform specialized tasks in which expertise is necessary. Eg.

Engineering design,
Scientific discovery,
Medical diagnosis,
Financial planning.

These expert tasks require knowledge that many of us do not have, they often require much less knowledge than do the mundane tasks. This knowledge is easier to represent and deal with inside programs.

As a result, the problem areas where AI is now flourishing most as a practical discipline are primarily the domains that require only specialized expertise without the assistance of commonsense knowledge. There are now thousands of programs called expert systems in day to day operation throughout all areas of industry and government.

Mundane tasks

- Perception
 - Vision
 - Speech
- Natural language
 - Understanding
 - Generation
 - Translation
- Commonsense reasoning
- Robot control

Formal tasks

- Games
 - Chess
 - Backgammon
 - Checkers
 - Go
- Mathematics
 - Geometry
 - Logic
 - Integral calculus
 - Proving properties of programs

Expert tasks

- Engineering
 - Design
 - Fault finding
 - Manufacturing planning
- Scientific analysis
- Medical diagnosis
- Financial analysis

Definitions

(AI by Russel and Norvig)

Definitions of artificial intelligence according to 8 text books are given below.

Artificial intelligence is a system that thinks like human beings.

1. **AI is an exciting effort to make computers think... machines with minds, in the full and literal sense.**
2. **AI is the automation of activities that we associate with human thinking, activities such as decision making, problem solving, learning..**

Artificial intelligence is a system that thinks rationally.

3. **AI is the study of mental faculties through the use of computational models.**
4. **AI is the study of computations that make it possible to perceive, reason and act.**

Artificial intelligence is a system that acts like human beings.

5. **AI is the art of creating machines that perform functions that require intelligence when performed by people.**
6. **AI is the study of how to make computers do things at which , at the moment , people do better.**

Artificial intelligence is a system that acts rationally.

7. **AI is the study of the design of intelligent agents.**
8. **AI is concerned with intelligent behavior in artifacts.**

AI is a system that acts like human beings

For this, a computer would need to possess the following capabilities.

Natural language processing

To enable it to communicate successfully in English.

Knowledge representation

To store what it knows or hears.

Automated reasoning

To use the stored information to answer questions and to draw new conclusions.

Machine learning

To adapt to new circumstances and to detect and extrapolate patterns.

Computer vision

To perceive objects.

Robotics

To manipulate objects and move about.

AI is a system that thinks like human beings.

First we must have some way of determining how humans think. We need to get inside the workings of the human minds. Once we have a sufficiently precise theory of the mind, it becomes possible to express that theory using a computer program.

The field of cognitive science brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind.

AI is a system that thinks rationally

For a given set of correct premises, it is possible to yield new conclusions. For eg. “Socrates is a man; all men are mortal; therefore, Socrates is mortal.” These laws of thought were supposed to govern the operation of the mind. This resulted in a field called logic.

A precise notation for the statements about all kinds of things in the world and about relations among them are developed. Programs exist that could in principle solve any solvable problem described in logical notation.

There are 2 main obstacles to this approach. First it is not easy to take informal knowledge and state it in the formal terms required by logical notation. Second, there is a big difference between being able to solve a problem “in principle” and doing so in practice.

AI is a system that acts rationally

An agent is something that acts. A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

We need the ability to represent knowledge and reason with it because this enables us to reach good decisions in a wide variety of situations. We need to be able to generate comprehensive sentences in natural language because saying those sentences helps us get by in a complex society. We need learning because having a better idea of how the world works enables us to generate more effective strategies for dealing with it. We need visual perception to get a better idea of what an action might achieve.

AI application areas

(AI by Luger)

The 2 most fundamental concerns of AI researchers are knowledge representation and search.

Knowledge representation

It addresses the problem of capturing the full range of knowledge required for intelligent behavior in a formal language, i.e. One suitable for computer manipulation.

Eg. predicate calculus,

LISP,

Prolog

Search

It is a problem solving technique that systematically explores a space of problem states, ie, successive and alternative stages in the problem solving process.

The following explains the major application areas of AI.

Game playing

Much of the early research in AI was done using common board games such as checkers, chess and the 15 puzzle. Board games have certain properties that made them ideal for AI research. Most games are played using a well defined set of rules. This makes it easy to generate the search space. The board configuration used in playing these games can be easily represented on a computer. As games can be easily played, testing a game playing program presents no financial or ethical burden.

Heuristics

Games can generate extremely large search spaces. So we use powerful techniques called heuristics to explore the problem space. A heuristic is a useful but potentially fallible problem strategy, such as checking to make sure that an unresponsive appliance is plugged in before assuming that it is broken.

Since most of us have some experience with these simple games, we do not need to find and consult an expert. For these reasons games provide a rich domain for the study of heuristic search.

Automated reasoning and theorem proving

Examples for automatic theorem provers are Newell and Simon's Logic Theorist, General Problem Solver (GPS).

Theorem proving research is responsible for the development of languages such as predicate calculus and prolog.

The attraction of automated theorem proving lies in the rigor and generality of logic. A wide variety of problems can be attacked by representing the problem description as logical axioms and treating problem instances as theorems to be proved.

Reasoning based on formal mathematical logic is also attractive. Many important problems such as design and verification of logic circuits, verification of the correctness of computer programs and control of complex systems come in this category.

Expert systems

Here comes the importance of domain specific knowledge. A doctor, for example, is effective at diagnosing illness because she possesses some innate general problem solving skill; she is effective because she knows a lot about medicine. A geologist is effective at discovering mineral deposits.

Expert knowledge is a combination of theoretical understanding of the problem and a collection of heuristic problem solving rules that experience has shown to be effective in the domain. Expert systems are constructed by obtaining this knowledge from a human expert and coding it into a form that a computer may apply to similar problems.

To develop such a system, we must obtain knowledge from a human domain expert. Examples for domain experts are doctor, chemist, geologist, engineer etc.. The domain expert provides the necessary knowledge of the problem domain. The AI specialist is responsible for implementing this knowledge in a program. Once such a program has been written, it is necessary to refine its expertise through a process of giving it example problems to solve and making any required changes or modifications to the program's knowledge.

Dendral is an expert system designed to infer the structure of organic molecules from their chemical formulas and mass spectrographic information about the chemical bonds present in the molecules.

Mycin is an expert system which uses expert medical knowledge to diagnose and prescribe treatment for spinal meningitis and bacterial infections of the blood.

Prospector is an expert system for determining the probable location and type of ore deposits based on geological information about a site.

Internist is an expert system for performing diagnosis in the area of internal medicine.

The dipmeter advisor is an expert system for interpreting the results of oil well drilling logs.

Xcon is an expert system for configuring VAX computers.

Natural language understanding and semantic modeling

One goal of AI is the creation of programs that are capable of understanding and generating human language. Systems that can use natural language with the flexibility and generality that characterize human speech are beyond current methodologies.

Understanding natural language involves much more than parsing sentences into their individual parts of speech and looking those words up in a dictionary. Real understanding depends on extensive background knowledge.

Consider for example, the difficulties in carrying out a conversation about baseball with an individual who understands English but knows nothing about the rules of the game. This person will not be able to understand the meaning of the sentence. “With none down in the top of the ninth and the go ahead run at second, the manager called his relief from the bull pen”. Even though half of the words in the sentence may be individually understood, this sentence would be difficult to even the most intelligent non base ball fan.

The task of collecting and organizing this background knowledge in such a way that it may be applied to language comprehension forms the major problem in automating natural language understanding.

Modeling human performance

We saw that human intelligence is a reference point in considering artificial intelligence. It does not mean that programs should pattern themselves after the organization of the human mind. Programs that take non human approaches to solving problems are often more successful than their human counterparts. Still, the design of systems that explicitly model some aspect of human performance has been a fertile area of research in both AI and psychology.

Planning and robotics

Research in planning began as an effort to design robots that could perform their tasks with some degree of flexibility and responsiveness to outside world. Planning assumes a robot that is capable of performing certain atomic actions.

Planning is a difficult problem because of the size of the space of possible sequences of moves. Even an extremely simple robot is capable of generating a vast number of potential move sequences.

One method that human beings use in planning is hierarchical problem decomposition. If we are planning a trip to London, we will generally treat the problems of arranging a flight, getting to the air port, making airline connections and finding ground transportation in London separately. Each of these may be further decomposed into smaller sub problems.

Creating a computer program that can do the same is a difficult challenge.

A robot that blindly performs a sequence of actions without responding to changes in its environment cannot be considered intelligent. Often, a robot will have to formulate a plan based on the incomplete information and correct its behavior. A robot may not have adequate sensors to locate all obstacles in the way of a projected path. Organizing plans in a fashion that allows response to changing environmental conditions is a major problem for planning.

Languages and environments for AI

Programming environments include knowledge structuring techniques such as object oriented programming and expert systems frameworks. High level languages such as Lisp and Prolog support modular development.

Many AI algorithms are also now built in more traditional computing languages such as C++ and Java.

Machine learning

An expert system may perform extensive and costly computations to solve a problem. But if it is given the same or similar problem a second time, it usually does not remember the solution. It performs the same sequence of computations again. This is not the behavior of an intelligent problem solver.

The programs must learn on their own. Learning is a difficult area. But there are several programs that suggest that it is possible.

One program is AM, the automated mathematician which was designed to discover mathematical laws. Initially given the concepts and axioms of set theory, AM was able to induce important mathematical concepts such as cardinality, integer arithmetic and many of the results of number theory. AM conjectured new theorems by modifying its current knowledge base.

Early work includes Winston's research on the induction of structural concepts such as "arch" from a set of examples in the blocks world.

The ID3 algorithm has proved successful in learning general patterns from examples.

Meta dendral learns rules for interpreting mass spectrographic data in organic chemistry from examples of data on compounds of known structure.

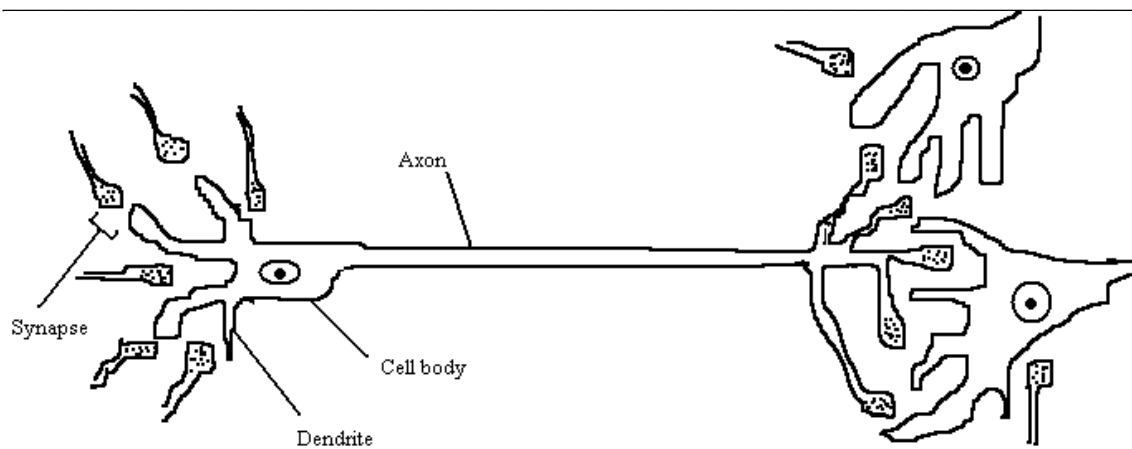
Teiresias, an intelligent front end for expert systems, converts high level advice into new rules for its knowledge base.

There are also now many important biological and sociological models of learning.

Neural nets and genetic algorithms

An approach to build intelligent programs is to use models that parallel the structure of neurons in the human brain.

A neuron consists of a cell body that has a number of branched protrusions called dendrites and a single branch called the axon. Dendrites receive signals from other neurons. When these combined impulses exceed a certain threshold, the neuron fires and an impulse or spike passes down the axon.



This description of the neuron captures features that are relevant to neural models of computation. Each computational unit computes some function of its inputs and passes the result along to connected units in the network; the final results are produced by the parallel and distributed processing of this network of neural connection and threshold weights.

Example problems

(AI by Russel)

Problems can be classified as toy problems and real world problems.

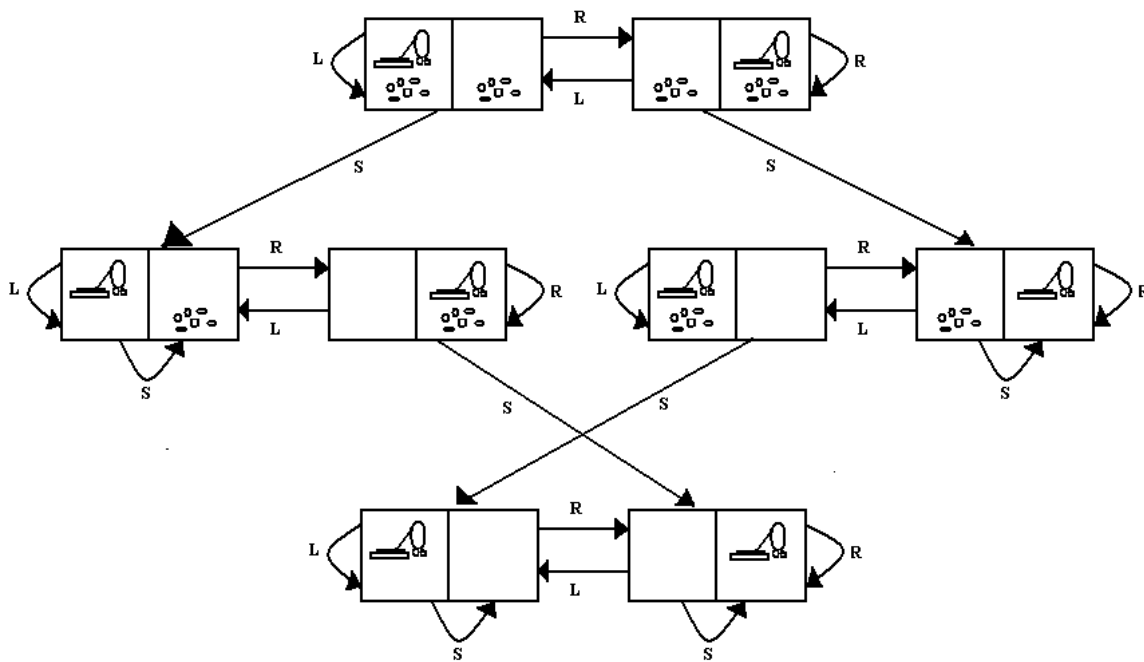
A toy problem is intended to illustrate or exercise various problem solving methods. It can be used by different researchers to compare the performance of algorithms.

A real world problem is one whose solutions people actually care about.

Toy problems

The first example is the

vacuum world



The agent is in one of the 2 locations, each of which might or might not contain dirt.

Any state can be designated as the initial state.

After trying these actions (Left, Right, Suck), we get another state.

The goal test checks whether all squares are clean.

8 – puzzle problem

it consists of a 3 X 3 board with 8 numbered tiles and a blank space as shown below.

7	2	4
5		6
8	3	1

Start state

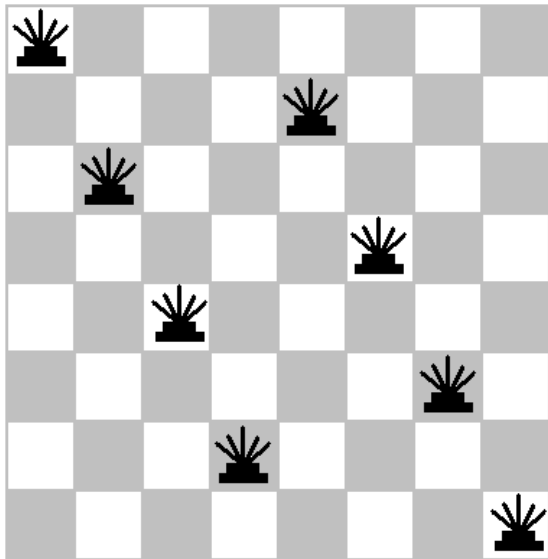
	1	2
3	4	5
6	7	8

Goal state

A tile adjacent to the blank space can slide into the space. The aim is to reach a specified goal state, such as the one shown on the right of the figure.

8 – Queens problem

The goal of 8- queens' problem is to place 8 queens on a chess board such that no queen attacks any other.



(a queen attacks any piece in the same row, column or diagonal). Figure shows an attempted solution that fails: the queen in the rightmost column is attacked by the queen at the top left.

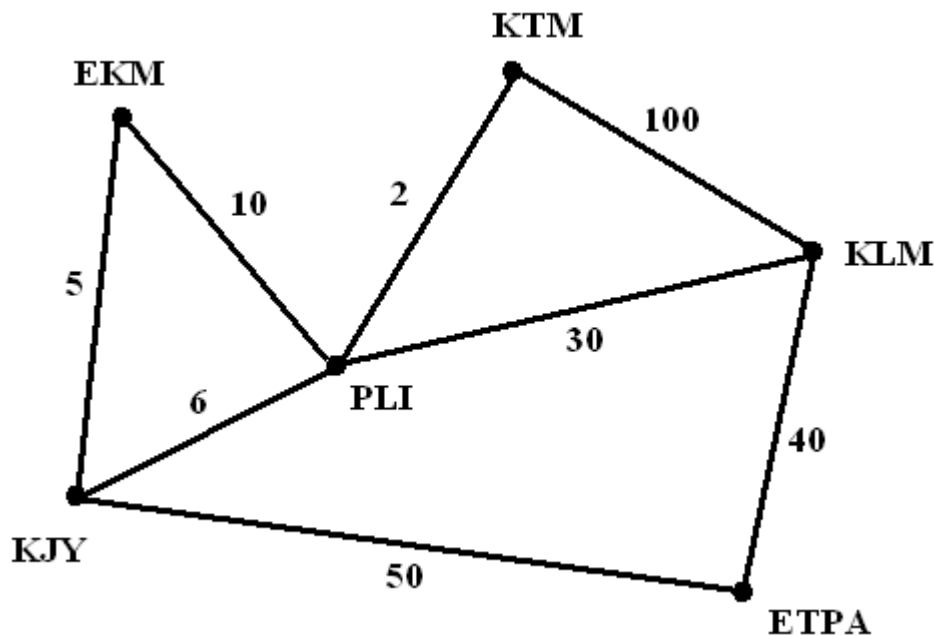
Real world problems

Route finding problem

Route finding algorithms are used in a variety of applications, such as routing in computer networks, military operations planning and air line travel planning systems.

Touring problems

They are related to route finding problems. For example, consider the figure.



Consider the problem. ‘Visit every city in the figure at least once starting and ending in Palai’. Each state must include not just the current location but also the set of cities the agent has visited.

Traveling salesperson problem (TSP)

Is a touring problem in which each city must be visited exactly once. The aim is to find the shortest tour.

VLSI layout problem

It requires positioning of millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances, and maximize manufacturing yield.

Robot navigation

It is a generalization of the route finding problem. Rather than a discrete set of routes, a robot can move in a continuous space with an infinite set of possible actions and states.

Automatic assembly sequencing of complex objects by a robot

The assembly of intricate objects such as electric motors is economically feasible. In assembly problems, the aim is to find an order in which to assemble the parts of some object. If the wrong order is chosen, there will be no way to add some part later in the sequence without undoing some of the work already done.

Protein design

Here the aim is to find a sequence of amino acids that will fold into a three dimensional protein with the right properties to cure some disease.

Internet searching

It means looking for answers to questions for related information or for shopping details. Software robots are being developed for performing this internet searching.

Problems

(AI by Ritchie & Knight)

We have seen different kinds of problems with which AI is typically concerned. To build a system to solve a particular problem, we need to do 4 things.

1. Define the problem precisely. This includes the initial state as well as the final goal state.
2. Analyze the problem.
3. Isolate and represent the knowledge that is needed to solve the problem.
4. Choose the best problem solving technique and apply it to the particular problem.

































Problem spaces

(AI by Ritchie & Knight)

Suppose we are given a problem statement “play chess”. This now stands as a very incomplete statement of the problem we want solved.

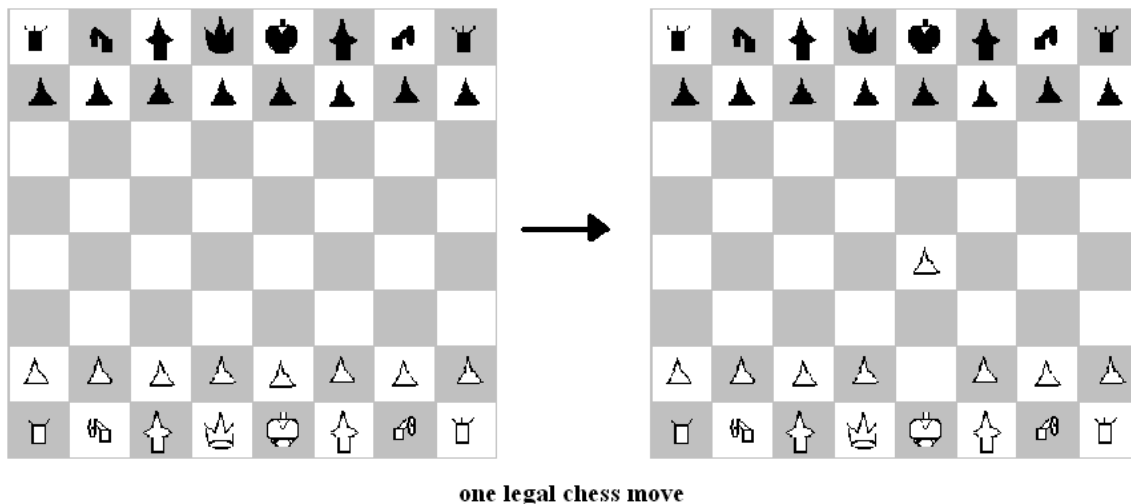
To build a program that could “play chess”, we could first have to specify the starting position of the chess board, the rules that define the legal moves and the board positions that represent a win for one side or the other.

For this problem “play chess”, it is easy to provide a formal and complete problem description. The starting position can be described as an 8 by 8 array as follows.

We can define as our goal any board position in which the opponent does not have a legal move and his or her king is under attack.

The legal moves provide the way of getting the initial state to a goal state. They can be described easily as a set of rules consisting of 2 parts: a left side that serves as a pattern to be matched against the current board position and a right side that describes the change to be made to the board position to reflect the move.



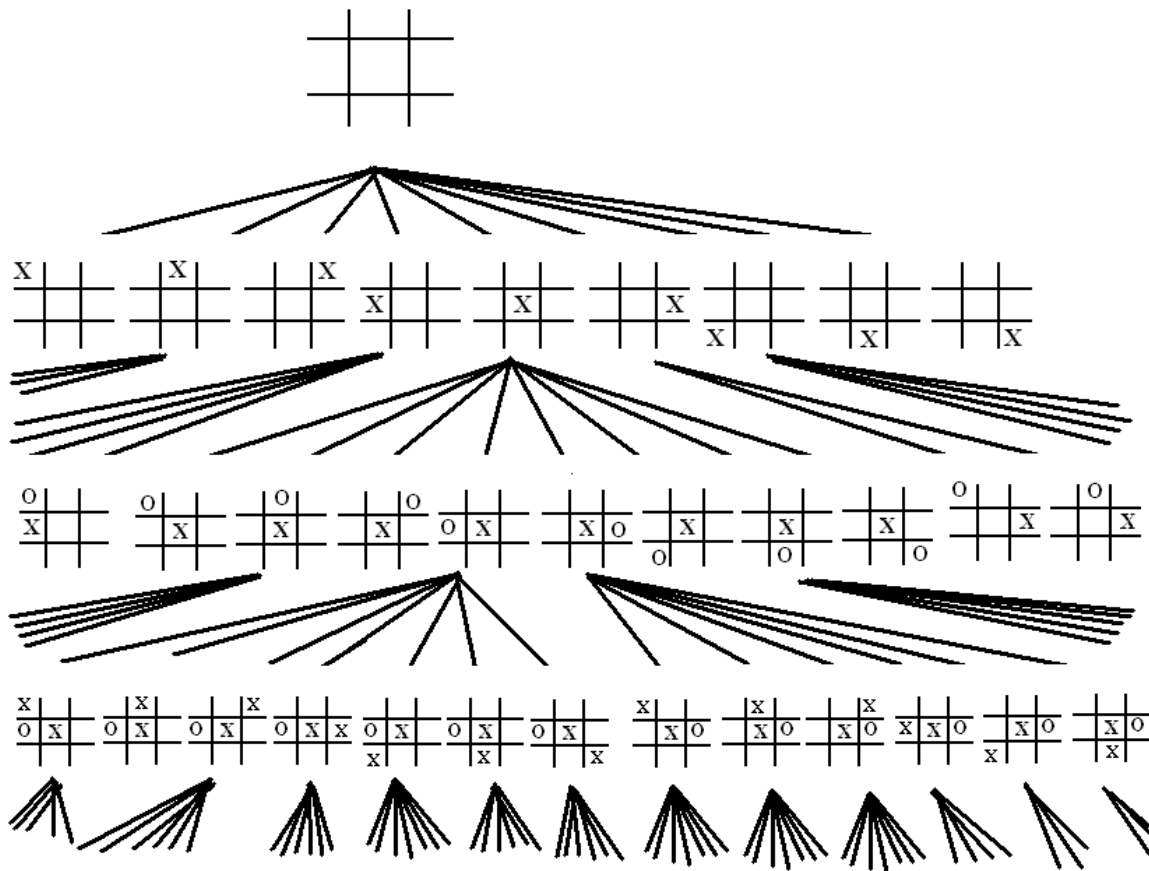
We have defined the problem of playing chess as a problem of moving around in a state space, where each state corresponds to a legal position of the board. We can then play chess by starting at an initial state, using a set of rules to move from one state to another, and attempting to end up in one of a set of final states.

This state space representation is useful for naturally occurring, less well structured problems. This state space representation forms the basis of most of the AI methods.

Example 2

The game of tic-tac-toe

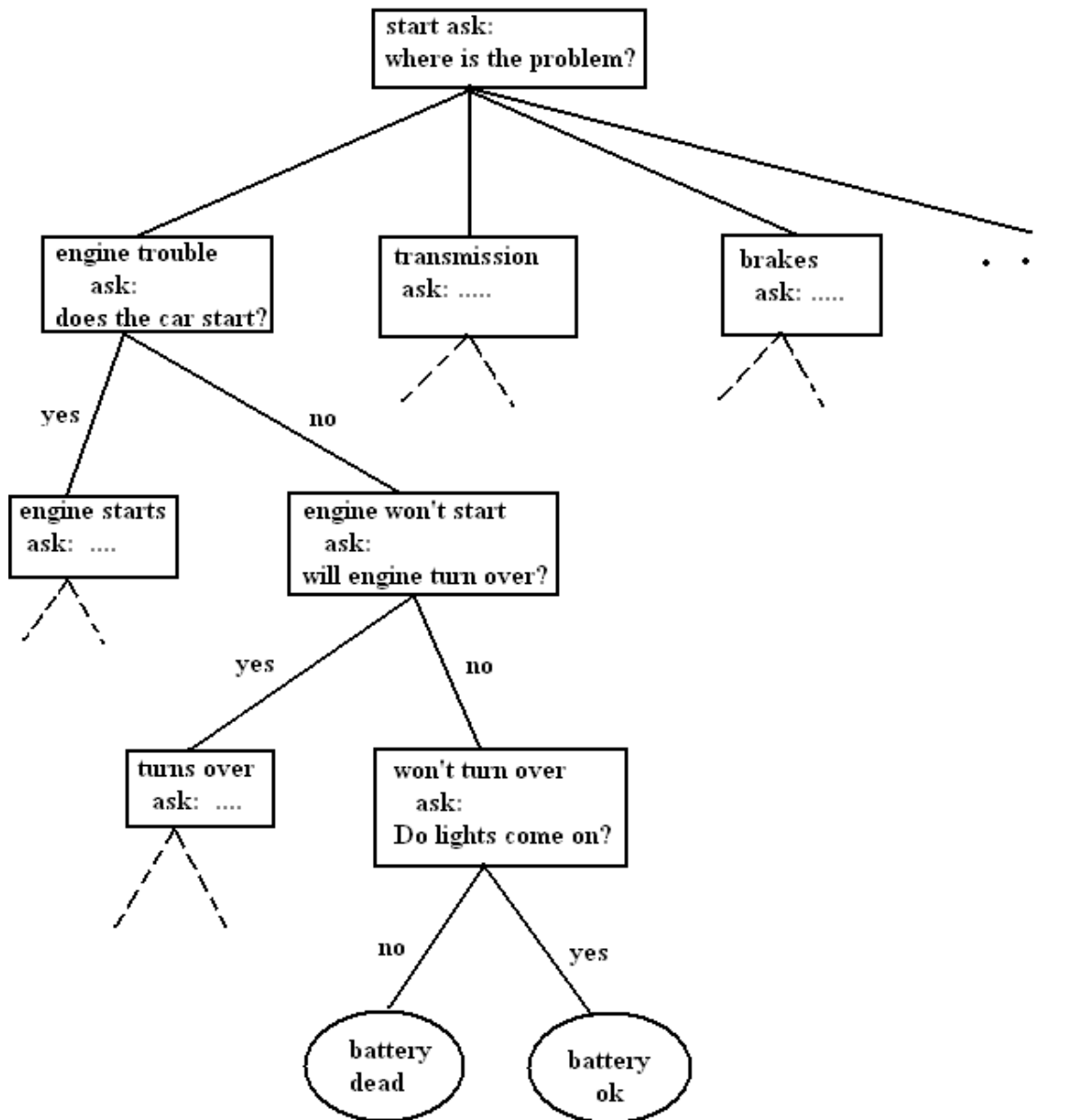
Starting with an empty board, the first player may place an X in any one of nine places. Each of these moves yields a different board that will allow the opponent 8 possible responses and so on. We can represent this collection of possible moves and responses by regarding each board configuration as a node in a graph. The links of the graph represent legal moves from one board configuration to another. These nodes correspond to different states of the game board. The resulting structure is called a state space graph.



Example 3

Diagnosing a mechanical fault in an automobile

Here each node of the state space graph represent a partial knowledge about the automobile's mechanical problems.



The starting node of the graph is empty, indicating that nothing is known about the cause of the problem. Each of the states in the graph has arcs that lead to states representing further accumulation of knowledge in the diagnostic process.

For example the engine trouble node has arcs to nodes labeled 'engine starts' and 'engine won't start'. From the 'won't start' node, we may move to nodes labeled 'turns over' and 'won't turn over'. The 'won't turn over' node has arcs to nodes labeled 'battery ok'.

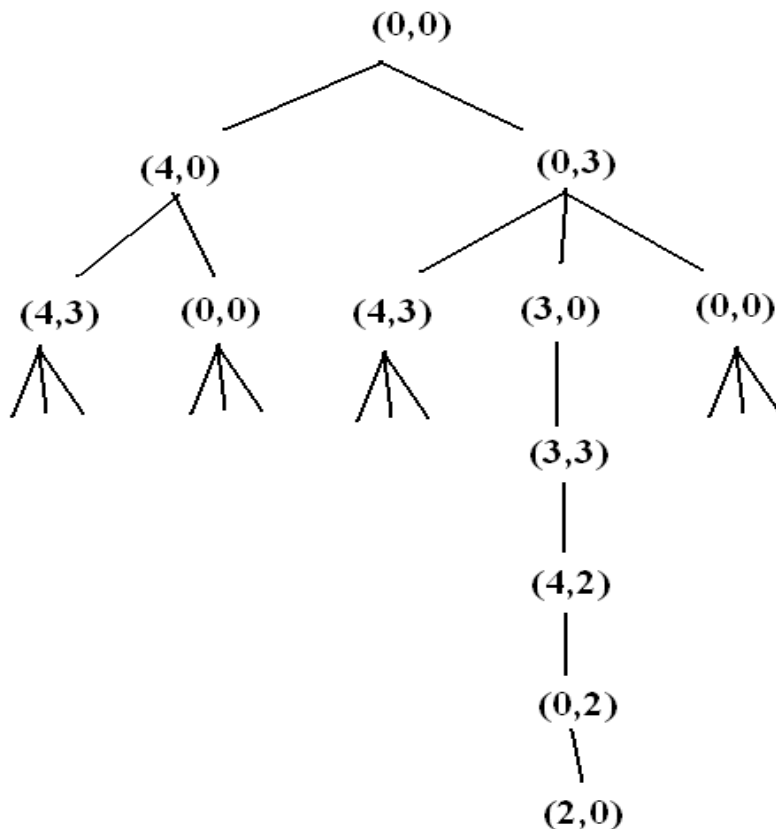
Example 3

Water jug problem

We are given 2 jugs, a 4 gallon and a 3 gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. We need to get exactly 2 gallons of water in to the 4 gallon jug.

The state space for this problem can be described as the set of ordered pairs of integers (x,y) , such that $x=0,1,2,3$ or 4 and $y=0,1,2$ or 3 ; x represents the number of gallons of water in the 4 gallon jug, and y represents the quantity of water in the 3 gallon jug.

The start state is $(0,0)$. The goal state is $(2,n)$ for any value of n .



In order to provide a formal description of a problem, we must do the following.

1. Define a state space that contains all the possible configurations of the relevant objects.
2. Specify one or more states within that space that describe possible situations from which the problem solving process may start. These states are called the initial states.
3. Specify one or more states that would be acceptable as solutions to the problem. These states are called goal states.
4. Specify a set of rules that describe the actions available.

Problem characteristics

(AI by Ritchie & Knight)

In order to choose the most appropriate method for a particular problem, it is necessary to analyze the problem along several dimensions.

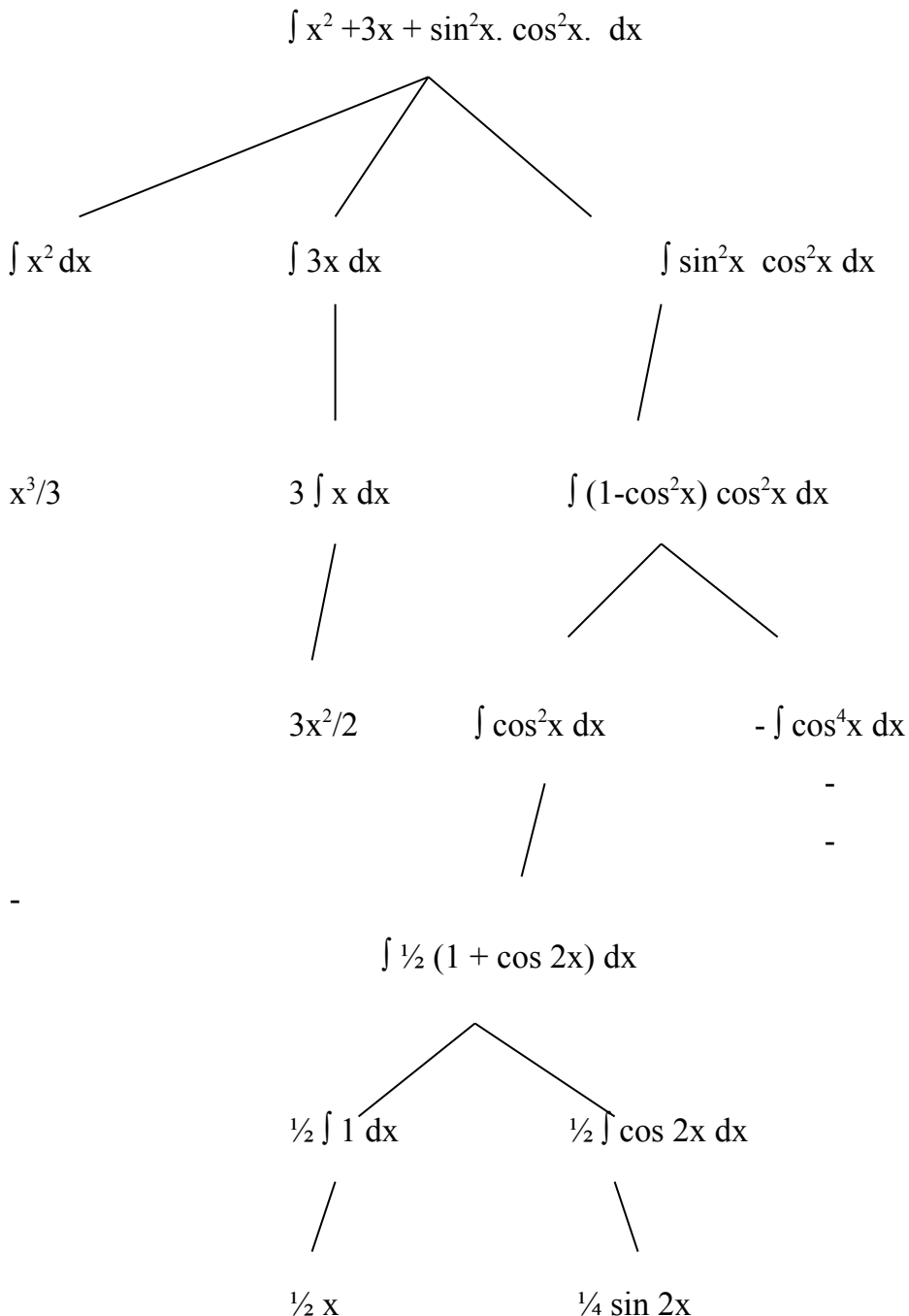
- Is the problem decomposable into a set of independent smaller or easier sub problems?
- Can solution steps be ignored or at least undone if they prove unwise?
- Is the problem's universe predictable?
- Is a good solution to the problem obvious without comparison to all other possible solutions?
- Is the desired solution a state of the world or a path to a state?
- Is a large amount of knowledge absolutely required to solve the problem, or is knowledge important only to constrain the search?
- Can a computer that is simply given the problem return the solution, or will the solution of the problem require interaction between the computer and a person?

Is the problem decomposable?

Suppose we want to solve the problem of computing the expression

$$\int (x^2 + 3x + \sin^2 x \cdot \cos^2 x) dx$$

We can solve this problem by breaking it down into 3 smaller problems, each of which can then solve by using a small collection of specific rules.



Can solution steps be ignored or undone?

Here we can divide problems into 3 classes.

Ignorable, in which solution steps can be ignored.

Recoverable, in which solution steps can be undone.

Irrecoverable, in which solution steps cannot be undone.

Ignorable problems (eg. Theorem proving)

Here solution steps can be ignored.

Suppose we are trying to prove a mathematical theorem. We proceed by first proving a lemma that we think will be useful. Eventually we realize that the lemma is no help at all. Here the different steps in proving the theorem can be ignored. Then we can start from another rule. The former can be ignored.

Recoverable problems (eg. 8 puzzle problem)

Consider the 8 puzzle problem.

7	2	4
5		6
8	3	1

Start state

	1	2
3	4	5
6	7	8

Goal state

The goal is to transform the starting position into the goal position by sliding the tiles around.

In an attempt to solve the 8- puzzle, we might make a stupid move. For example, in the game shown above, we might start by sliding tile 5 into the empty space. Having done that, we cannot

change our mind and immediately slide tile 6 into the empty space since the empty space will essentially have moved. But we can backtrack and undo the 1st move, sliding tile 5 back to where it was. Then we can move tile 6. here mistakes can be recovered.

Irrecoverable problems (eg. Chess)

Consider the problem of playing chess. Suppose a chess playing program makes a stupid move and realizes it a couple of moves later. It cannot simply play as though it had never made the stupid move. Nor can it simply back up and start the game over from that point. All it can do is to try to make the best of the current situation and go from there.

The recoverability of a problem plays an important role in determining the complexity of the control structure necessary for the problem's solution. Ignorable problems can be solved using a simple control structure. Recoverable problems can be solved by a slightly more complicated control strategy that does sometimes makes mistakes. Irrecoverable problems will need to be solved by a system that expends a great deal of effort making each decision since the decision must be final.

Is the universe predictable?

Certain outcome problems (eg. 8 puzzle)

Suppose we are playing with the 8 puzzle problem. Every time we make a move, we know exactly what will happen. This means that it is possible to plan an entire sequence of moves and be confident that we know what the resulting state will be.

Uncertain outcome problems (eg. Bridge)

However, in games such as bridge, this planning may not be possible. One of the decisions we will have to make is which card to play on the first trick. What we would like to do is to plan the entire hand before making that first play. But now it is not possible to do such planning with certainty since we cannot know exactly where all the cards are or what the other players will do on their turns.

One of the hardest types of problems to solve is the irrecoverable, uncertain outcome. Examples of such problems are
Playing bridge,
Controlling a robot arm,
Helping a lawyer decide how to defend his client against a murder charge.

Is a good solution absolute or relative?

Any path problems

Consider the problem of answering questions based on a database of simple facts, such as the following.

- 1. Marcus was a man.
- 2. Marcus was a Pompean.
- 3. Marcus was born in 40 A. D.
- 4. all men are mortal.
- 5. All pompeans died when the volcano erupted in 79 A. D.
- 6. No mortal lives longer than 150 years.
- 7. It is now 1991 A. D.

Suppose we ask the question. “Is Marcus alive?”. By representing each of these facts in a formal language, such as predicate logic, and then using formal inference methods, we can fairly easily derive an answer to the question. The following shows 2 ways of deciding that Marcus is dead.

	axioms
1. Marcus was a man.	1
4. All men are mortal.	4
8. Marcus is mortal.	1,4
3. Marcus was born in 40 A.D.	3
7. It is now 1991 A. D.	7
9. Marcus’ age is 1951 years.	3,7
6. no mortal lives longer than 150 years.	6
10. Marcus is dead.	8,6,9

 OR

- | | |
|----------------------------------|------|
| 7. It is now 1991 A.D. | 7 |
| 5. All Pompeans died in 79 A. D. | 5 |
| 11. All Pompeans are dead now. | 7,5 |
| 2. Marcus was a Pompean. | 2 |
| 12. Marcus is dead. | 11,2 |

Since all we are interested in is the answer to the question, it does not matter which path we follow.

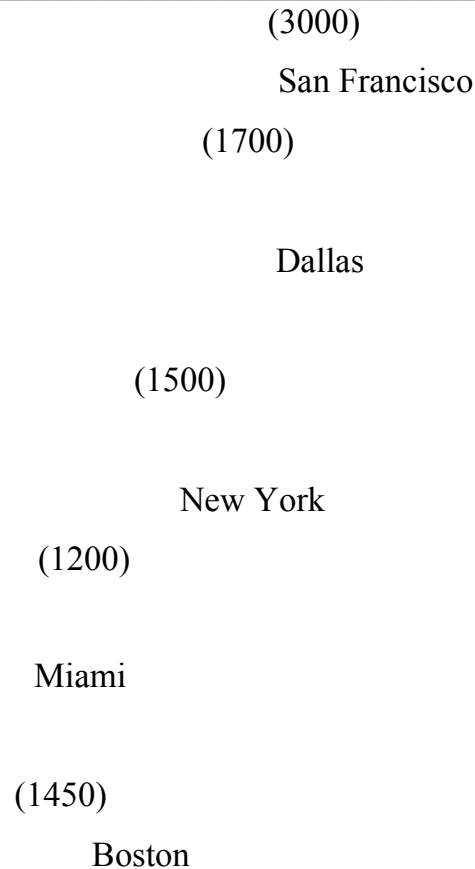
Best path problems (eg. Traveling salesman problem)

Consider the traveling salesman problem. Our goal is to find the shortest route that visits each city exactly once. Suppose the cities to be visited and the distances between them are shown below.

	Boston	NY	Miami	Dallas	SF
Boston		250	1450	1700	3000
NY	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
SF	3000	2900	3300	1700	

One place the salesman could start is Boston. In that case, one path that might be followed is the one shown below which is 8850 miles long.

Boston



But is this the solution to the problem? The answer is that we cannot be sure unless we also try all other paths to make sure that none of them is shorter.

Best path problems are computationally harder than any path problems.

Is the solution a state or a path?

Problems whose solution is a state of the world.

eg. Natural language understanding

Consider the problem of finding a consistent interpretation for the sentence,

‘ The bank president ate a dish of pasta salad with the fork’.

There are several components of this sentence, each of which, in isolation, may have more than one interpretation. Some of the sources of ambiguity in this sentence are the following.

The word ‘bank’ may refer either to a financial institution or to a side of a river.

The word dish is the object of the verb 'eat'. It is possible that a dish was eaten. But it is more likely that the pasta salad in the dish was eaten.

Pasta salad is a salad containing pasta. But there are other ways interpretations can be formed from pairs of nouns. For example, dog food does not normally contain dogs.

The phrase 'with the fork' could modify several parts of the sentence. In this case, it modifies the verb 'eat'. But, if the phrase had been 'with vegetables', then the modification structure would be different.

Because of the interaction among the interpretations of the constituents of this sentence, some search may be required to find a complete interpretation for the sentence. But to solve the problem of finding the interpretation, we need to produce only the interpretation itself. No record of the processing by which the interpretation was found is necessary.

Problems whose solution is a path to a state?

Eg. Water jug problem

In water jug problem, it is not sufficient to report that we have solved the problem and that the final state is (2,0). For this kind of problem, what we really must report is not the final state, but the path that we found to that state.

What is the role of knowledge?

Problems for which a lot of knowledge is important only to constrain the search for a solution.

Eg. Chess

Consider the problem of playing chess. How much knowledge would be required by a perfect chess playing program? Just the rules for determining the legal moves and some simple control mechanism that implements an appropriate search procedure.

Problems for which a lot of knowledge is required even to be able to recognize a solution.

Eg. News paper story understanding

Consider the problem of scanning daily newspapers to decide which are supporting democrats and which are supporting the republicans in some upcoming election. How much knowledge would be required by a computer trying to solve this problem? Here a great deal of knowledge is necessary.

Does the task require interaction with a person?

Solitary problems

Here the computer is given a problem description and produces an answer with no intermediate communication and with no demand for an explanation for the reasoning process.

Consider the problem of proving mathematical theorems. If

1. All we want is to know that there is a proof.
2. The program is capable of finding a proof by itself.

Then it does not matter what strategy the program takes to find the proof.

Conversational problems

In which there is intermediate communication between a person and the computer, either to provide additional assistance to the computer or to provide additional information to the user. Eg. Suppose we are trying to prove some new, very difficult theorem. Then the program may not know where to start. At the moment, people are still better at doing the high level strategy required for a proof. So the computer might like to be able to ask for advice. To exploit such advice, the computer's reasoning must be analogous to that of its human advisor, at least on a few levels.

Problem solving by searching

We have seen problems and problem spaces.

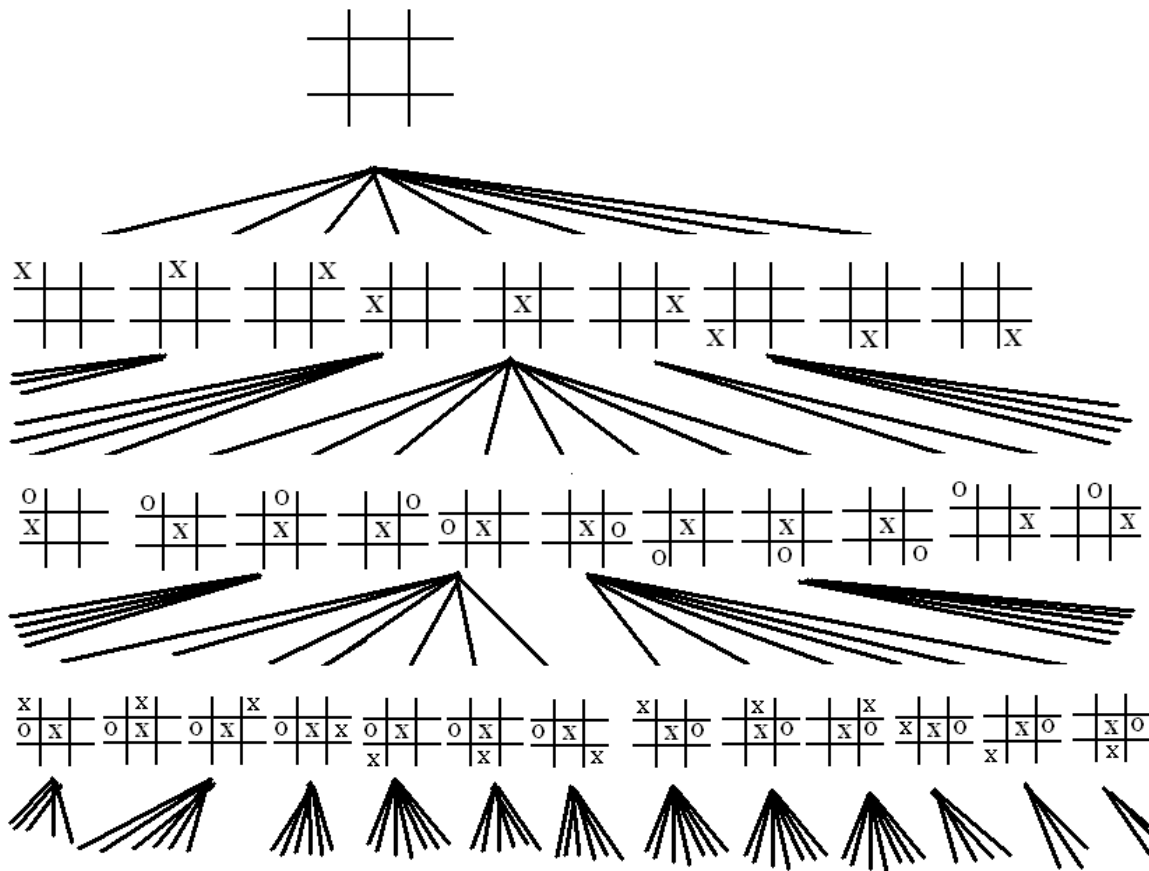
Search is a problem solving technique that systematically explores a space of problem states. That is to move through the problem space until a path from an initial state to a goal state is found.

Examples of problem states might include the different board configurations in a game or intermediate steps in a reasoning process. This space of alternate solutions is then searched to find a final answer.

We can contrast this with human problem solving. Humans generally consider a number of alternative strategies on their way to solving a problem. A chess player typically considers a number of alternative moves, selecting the best according to such criteria as the opponent's possible responses. A mathematician will choose from a different but equally complex set of strategies to find a proof for a difficult theorem, a physician may systematically evaluate a number of possible diagnoses and so on.

Eg. Game of tic-tac-toe

Given any board situation, there is only a finite set of moves that a player can make. Starting with an empty board, the first player may place an X in any one of 9 places. Each of these moves yields a different board that will allow the opponent 8 possible responses, and so on. We can represent this collection of possible moves and responses as a state space graph.



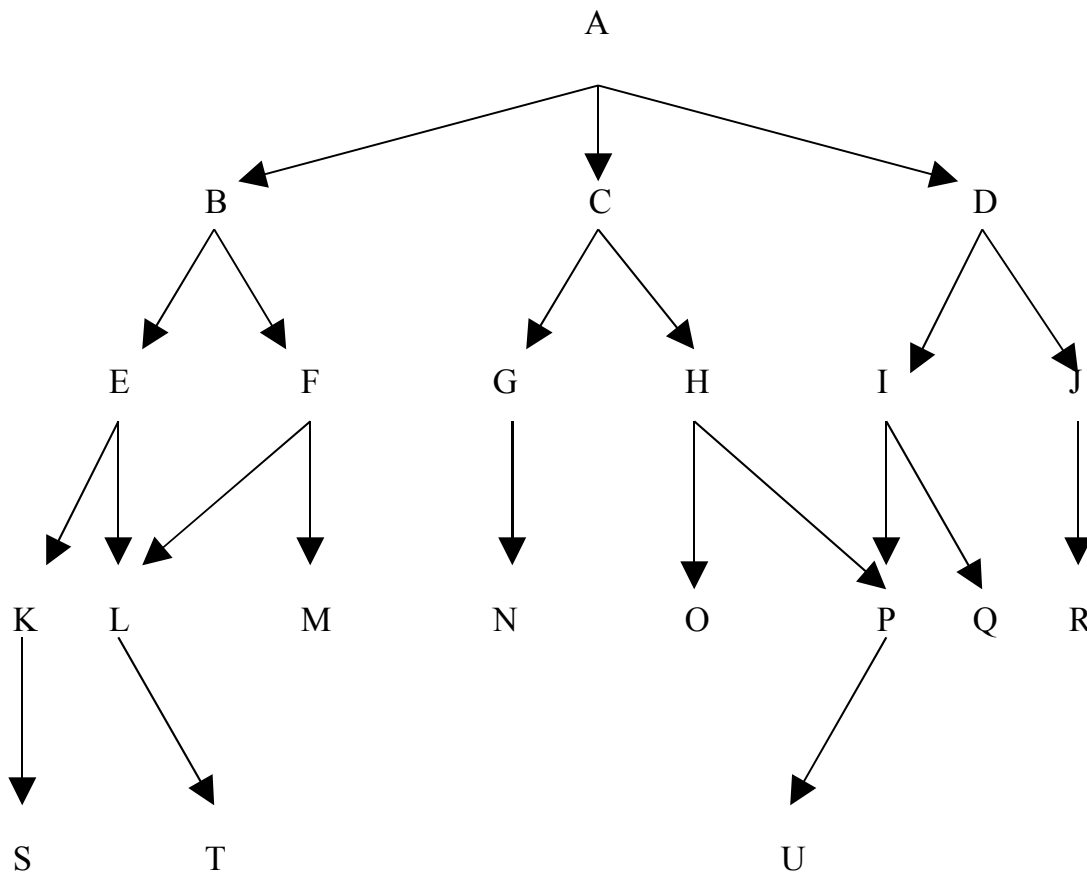
Given this representation, an effective game strategy will search through the graph for the paths that lead to the most wins and fewest losses and play in a way that always tries to force the game along one of the optimal paths. This searching strategy is an effective one and also it is straightforward to implement on a computer.

Searching strategies

Breadth first search

(AI by Luger)

Consider the graph shown below.



States are labeled (A, B, C....).

Breadth first search explores the space in a level by level fashion. Only when there are no more states to be explored at a given level does the algorithm move on to the next level. A breadth first search of the above graph considers the states in the order A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U.

We implement breadth first search using lists, open and closed, to keep track of progress through the state space. 'open' lists states that have been generated but whose children have not been

examined. The order in which states are removed from open determines the order of the search. 'closed' records states that have already been examined.

```
void breadth_first_search ( )
{
    open = [ start ];
    closed = [ ];
    while ( open not empty )
    {
        Remove the leftmost state from open, call it X;
        if X is a goal,
            then return SUCCESS;
        else
        {
            Generate children of X;
            Put X on closed;
            Discard children of x, if already on open or closed;
            Put remaining children on right end of open;
        }
    }
    return FAIL;
}
```

Child states are generated by inference rules, legal moves of a game or other state transition operators. Each iteration produces all children of the state x and adds them to open.

Note that open is maintained as a queue (FIFO) data structure. States are added to the right of the list and removed from the left.

A trace of the breadth first search on the graph appears below.

Open	closed
A	empty
BCD	A
CDEF	BA
DEFGH	CBA
EFGHIJ	DCBA
FGHIJKL	EDCBA
GHIJKLM	FEDCBA
HIJKLMN	GFEDCBA
.	
.	
.	
.	

And so on until open = [].

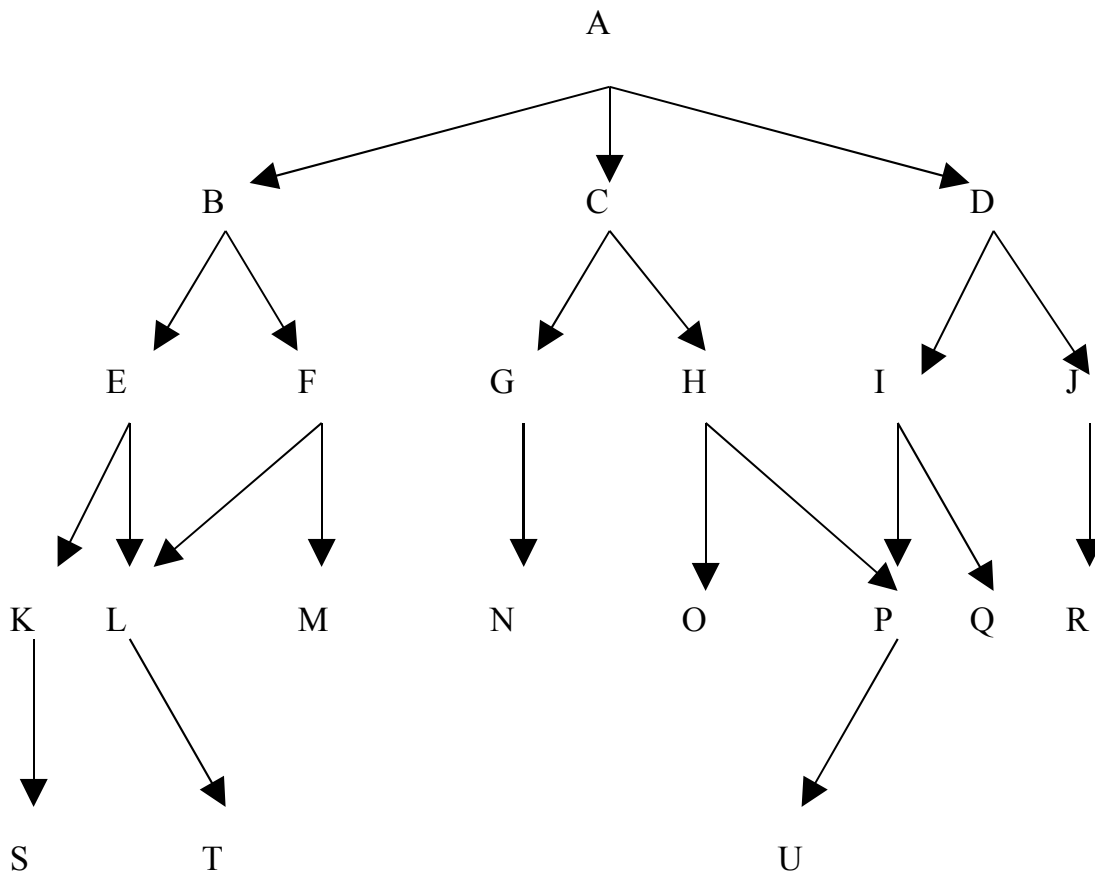
Because breadth first search considers every node at each level of the graph before going deeper in to the space, all states are first reached along the shortest path from the start state. Breadth first search is therefore guaranteed to find the shortest path from the start state to goal.

Depth first search

(AI by Luger)

Depth first search goes deeper in to the search space whenever this is possible.

Consider the graph



Depth first search examines the states in the graph in the order A, B, E, K, S, L, T, F, M, C, G, N, H, O, P, U, D, I, Q, J, R.

In depth first search, the descendent states are added and removed from the left end of open. That is, open is maintained as a stack (LIFO) data structure.

```

void depth_first_search ()
{
    open = [start];
    closed = [ ];
    while (open not empty )
    {
        remove leftmost state from open, call it X;
        if X is a goal state
            then return SUCCESS;
        else
        {
            generate children of X;
            put X on closed;
            discard children of X, if already on open or closed;
            put remaining children on left end of open;
        }
    }
    return FAIL;
}

```

A trace of depth first search on the above graph is shown below.

open	closed
A	empty
B C D	A
E F C D	B A
K L F C D	E B A
S L F C D	K E B A
L F C D	S K E B A
T F C D	L S K E B A

F C D	T L S K E B A
M C D	F T L S K E B A
C D	M F T L S K E B A
G H D	C M F T L S K E B A

And so on until open = [];

‘open’ records all states that are discovered and ‘closed’ contains all states that are already considered.

DFS is not guaranteed to find the shortest path to a state the first time that state is encountered.

Depth limited search

(AI by Russel)

to limit the depth of DFS, we can supply a predetermined depth limit, l to DFS. That is nodes at depth l are treated as if they have no successors. This approach is called depth limited search.

Depth first search can be viewed as a special case of depth limited search with $l = \infty$.

Sometimes depth limit can be based on knowledge of the problem. For example, on the map of Romania, there are 20 cities. Therefore, we know that if there is a solution, it must be of length 19 at the longest, so $l = 19$ is a possible choice. But in fact if we studied the map carefully, we would discover that any city can be reached from any other city in at most 9 steps. This number, known as the diameter of the state space, gives us a better depth limit, which leads to a more efficient depth limited search.

Uniform cost search

(AI by Russell)

Breadth first search that we have learned assumes that all path costs from a state to a successor is same. It expands the shallowest unexpanded node.

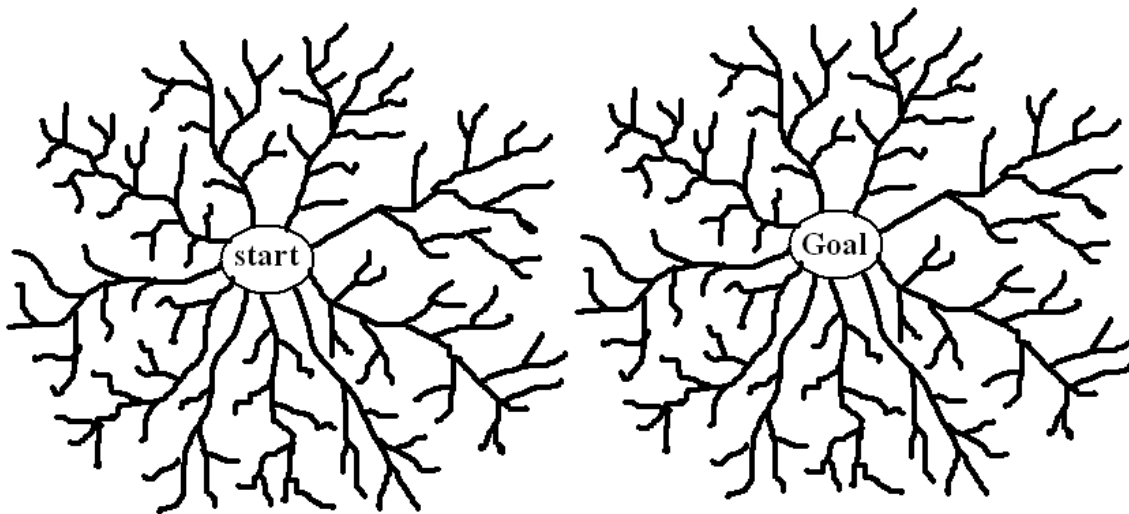
But uniform costs search expands the node n with the lowest path cost, instead of expanding the shallowest node. Note that if all step costs are equal, this is identical to breadth first search.

Uniform cost search does not care about the number of steps a path has, but only about their total cost. Therefore, it will get stuck in an infinite loop if it ever expands a node that has a zero cost action leading back to the same state. We can guarantee completeness provided the cost of every step is greater than or equal to some small positive constant. The algorithm expands nodes in order of increasing path cost.

Bidirectional search

(AI by Russell)

The idea behind bidirectional search is to run two simultaneous searches- one forward from the initial state and other backward from the goal, stopping when the 2 searches meet in the middle.



A schematic view of a bidirectional search that is about to succeed , when a branch from the start node meets a branch from the goal node.

Bidirectional search is implemented by having one or both of the searches check each node before it is expanded to see if it is in the fringe of the other search tree; if so, a solution has been found.

What do we mean by “the goal” in searching “backward from the goal”. For the 8-puzzle and for finding a route in Romania, there is just one goal state, so the backward search is very much like the forward search. If there are several explicitly listed goal states, then we can construct a new dummy goal state whose immediate predecessors are all the actual goal states.

The most difficult case for bidirectional search is when the goal test gives only an implicit description of some possibly large set of goal states. For example, all the states satisfying the “check mate” goal test in chess.

Constraint satisfaction search

(AI by Ritchie and Knight)

Many problems in AI are called to be the problems of constraint satisfaction. In these problems, the goal is to discover some problem state that satisfies a given set of constraints.

Examples are cryptarithmic puzzles, map coloring.

Crypt arithmetic problem

Here letters must be assigned particular numbers as their values. A constraint satisfaction approach to solving this problem avoids making guesses on particular assignments of numbers to letters until it has to. Instead, the initial set of constraints, which says that each number may correspond to only one letter and that the sums of the digits must be as they are given in the problem, is first augmented to include restrictions that can be inferred from the rules of arithmetic.

Constraint satisfaction is a search procedure that operates in a space of constraint sets. The initial state contains the constraints that are originally given in the problem description. A goal state is any state that has been constrained “enough”. For example, for cryptarithmic, enough means that each letter has been assigned a unique numeric value.

Constraint satisfaction is a 2 step process. First constraints are discovered and propagated as far as possible throughout the system. Second, if there is still not a solution, search begins. A guess about something is made and added as a new constraint. Propagation can then occur with this new constraint, and so forth.

The first step, propagation, arises from the fact that there are usually dependencies among the constraints. So for example, assume we start with one constraint, $N = E + 1$. Then, if we added the constraint $N = 3$, we could propagate that to get a stronger constraint on E , namely that $E = 2$.

Constraint propagation terminates for one of two reasons. First, a contradiction may be detected. If this happens, then there is no solution consistent with all the known constraints.

The second possible reason for termination is that the propagation has run out of steam and there are no further changes that can be made on the basis of current knowledge. If this happens, search is necessary to get the process moving again.

At this point, the second step begins. Some hypothesis about a way to strengthen the constraints must be made. In the case of the cryptarithmic problem, for example, this usually means guessing a particular value for some letter. Once this has been done, constraint propagation can begin again from this new state. If a solution is found, it can be reported. If still more guesses are required, they can be made. If a contradiction is detected, then backtracking can be used to try a different guess and proceed with it.

Constraint satisfaction algorithm

1. propagate available constraints. To do this, first set OPEN to the set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until OPEN is empty.
 - a. Select an object OB from OPEN. Strengthen as much as possible the set of constraints that apply to OB.
 - b. If this set is different from the set that was assigned the last time OB was examined or if this is the first time OB has been examined, then add to OPEN all objects that share any constraints with OB.
 - c. Remove OB from OPEN.

2. If the union of the constraints discovered above defines a solution, then quit and report the solution.
3. If the union of the constraints discovered above defines a contradiction, then return failure.
4. If the neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated.
 - a. Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.
 - b. Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.

The following example describes the working of the above algorithm.

Consider the cryptarithmic problem below.

Problem

```
  S  E  N  D
+ M  O  R  E
-----
M  O  N  E  Y
```

Initial state:

No two letters have the same value.

The sums of the digits must be as shown in the problem.

The goal state is a problem state in which all letters have been assigned a digit in such a way that all the initial constraints are satisfied.

The solution proceeds in cycles. At each cycle, 2 significant things are done.

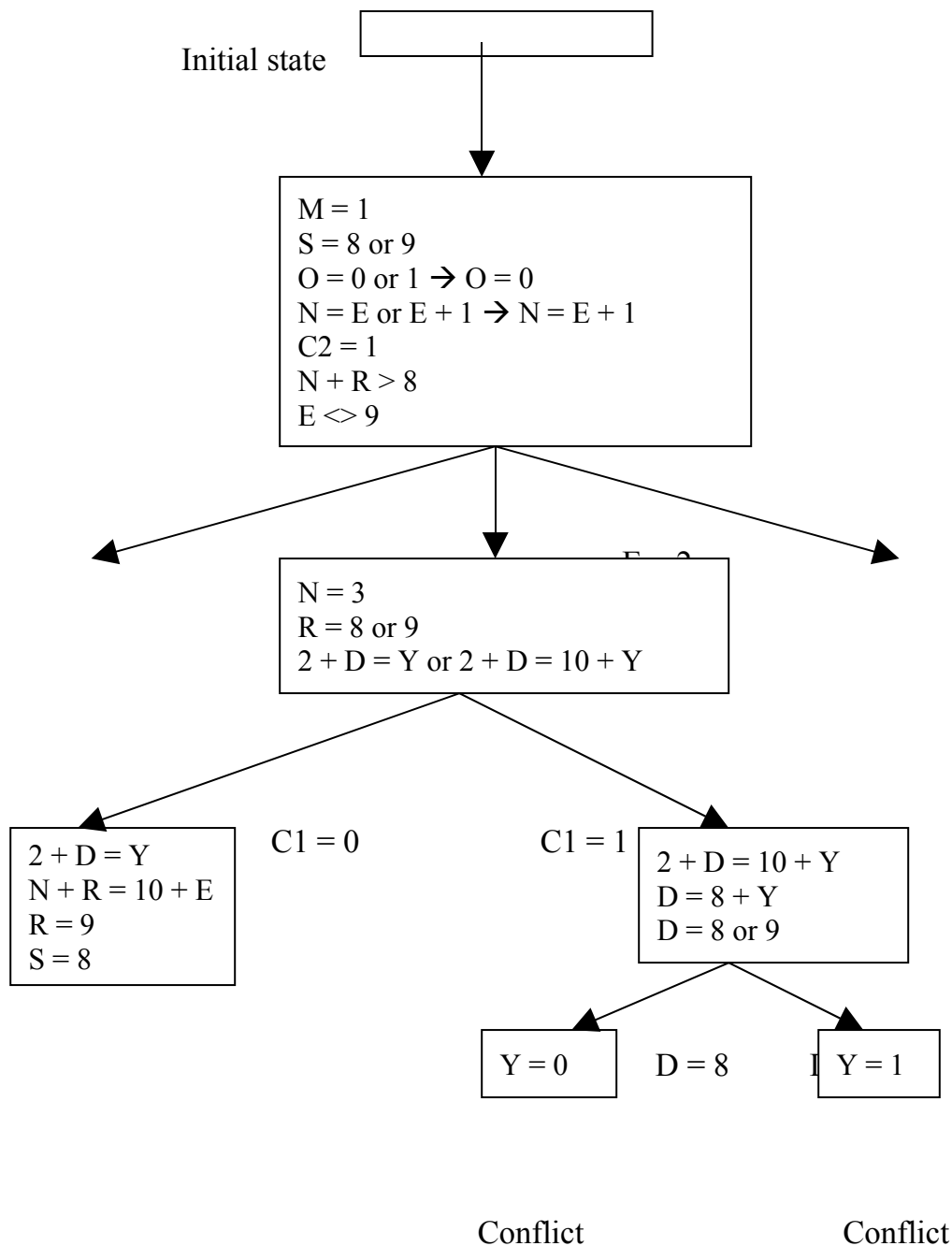
1. Constraints are propagated by using rules that correspond to the properties of arithmetic.
2. A value is guessed for some letter whose value is not yet determined.

In the first step, it does not usually matter a great deal what order the propagation is done in, since all available propagations will be performed before the step ends. In the second step, though the order in which guesses are tried may have a substantial impact on the degree of search that is necessary. A few useful heuristics can help to select the best guess to try first.

For example, if there is a letter that has only 2 possible values and another with 6 possible values, there is a better chance of guessing right on the first than on the second. Another useful heuristic is that if there is a letter that participates in many constraints then it is a good idea to prefer it to a letter that participates in a few. A guess on such a highly constrained letter will usually lead quickly either to a contradiction or to the generation of many additional constraints.

The result of the first few cycles of processing this example is shown below.

```
S  E  N  D
+ M  O  R  E
-----
M  O  N  E  Y
```



Initially, rules for propagating constraints generate the following additional constraints.

$M = 1$,

Since 2 single digit numbers plus a carry cannot total more than 19.

$S = 8 \text{ or } 9$,

Since $S + M + C3 > 9$ (to generate the carry) and $M=1$, $S + 1 + C3 > 9$, so $S + C3 > 8$ and $C3$ is at most 1.

$O = 0$,

Since $S + M(1) + C3 (\leq 1)$ must be at least 10 to generate a carry and it can be at most 1.

But M is already 1, so O must be 0.

$N = E$ or $E + 1$,

Depending on the value of $C2$. But N cannot have the same value as E . So $N = E + 1$ and $C2$ is 1.

In order for $C2$ to be 1, the sum of $N + R + C1$ must be greater than 9, so $N + R$ must be greater than 8.

$N + R$ cannot be greater than 18, even with a carry in, so E cannot be 9.

At this point, let us assume that no more constraints can be generated. Then, to make progress from here, we must guess. Suppose E is assigned the value

Now the next cycle begins. The constraint propagator now observes that

$N = 3$, since $N = E + 1$.

$R = 8$ or 9 , since $R + N(3) + C1(1 \text{ or } 0) = 2$ or 12 .

But since N is already 3, the sum of these nonnegative numbers cannot be less than 3. thus $R + 3 + (0 \text{ or } 1) = 12$ and $R = 8$ or 9 .

$2 + D = Y$ or $2 + D = 10 + Y$, from the sum in the rightmost column.

Again, assuming no further constraints can be generated, a guess is required. Suppose $C1$ is chosen to guess a value for. If we try the value 1, then we eventually reach dead ends as shown in the figure. When this happens, the process will backtrack and try $C1 = 0$.