A Historical Example

- One of the most famous problems in graph theory is the bridges of Konigsberg

Can you cross every bridge exactly once and come back to the start?

- Here is an abstraction of the problem as a graph and its representation in the predicate logic

- The answer is “no”, which everyone realized, but Euler proved this result in general for any graph structure where any vertex has an odd degree

The Real Koningsberg

- Koningsberg was part of Germany at the time of Euler
- You won’t find Koningsberg on a map today because it is no longer in Germany or have a German name
- What country do you think this city belongs to?
Some Definitions

**Definition**

**Graph**

A graph consists of:

A set of nodes $N_1, N_2, N_3, \ldots$, which need not be finite.

A set of arcs that connect pairs of nodes.

(Arcs are often described as an ordered pair of nodes, i.e., the arc $(N_2, N_3)$ connects node $N_2$ to node $N_3$.)

A directed graph has an indicated direction for traversing each arc. For example, a directed graph might have $(N_1, N_2)$ as an arc but not $(N_2, N_3)$. This would indicate that a path through the graph could go from node $N_2$ to $N_3$ but not from $N_3$ to $N_2$.

If a directed arc connects $N_j$ and $N_k$, then $N_j$ is called the parent of $N_k$ and $N_k$ the child of $N_j$. If the graph also contains an arc $(N_i, N_j)$, then $N_k$ and $N_l$ are called siblings.

A rooted graph has a unique node $N_r$ from which all paths in the graph originate. That is, the root has no parent in the graph.

A tip or leaf node is a node that has no children.

An ordered sequence of nodes $[N_1, N_2, N_3, \ldots, N_n]$, where each $N_i, N_j$, in the sequence represents an arc $(N_i, N_j)$, is called a path of length $n$ in the graph.

On a path in a rooted graph, a state is said to be an ancestor of all states positioned after it (to its right) and a descendant of all states before it (to its left).

A path that contains any state more than once (some $N_j$ in the definition of path above is repeated) is said to contain a cycle or loop.

A tree is a graph in which there is a unique path between every pair of nodes. (The paths in a tree, therefore, contain no cycles.)

The edges in a rooted tree are directed away from the root. Each node in a rooted tree has a unique parent.

Two states in a graph are said to be connected if a path exists that includes them both.

The Farmer, Wolf, Goat, and Cabbage

A farmer with his wolf, goat, and cabbage come to the edge of a river they wish to cross. There is a boat at the river’s edge, but, of course, only the farmer can row. The boat also can carry only two things (including the rower) at a time. If the wolf is ever left alone with the goat, the wolf will eat the goat; similarly, if the goat is left alone with the cabbage, the goat will eat the cabbage. Devise a sequence of crossings of the river so that all four characters arrive safely on the other side of the river.

- Using predicate logic we can express the current state of the problem as:
  \text{state(farmer-side,wolf-side,goat-side,cabbage-side)}

- If the goal is to go from the west side to the east side, then the starting position is state($w,w,w,w$)

- The goal is state($e,e,e,e$)

- Unsafe states must be avoided, one unsafe state is state($e,e,w,w$)

What are some other unsafe states?
Some Possible Safe Trips

The Search Space

Notice that this diagram includes unsafe states; where are they?

Find a solution to the FWGC problem.
Another Search Problem

- There are no markings on the jugs
- How would you represent the current state of the problem? Try to make this generalize to any sizes for jugs.
- What are the initial and final states?
- What would be the legal moves from one state to the next?
- Find a solution to this problem

Water jugs problem: We have one 3 litre jug, one 5 litre jug and an unlimited supply of water. The goal is to get exactly one litre of water into either jug. Either jug can be emptied or filled, or poured into the other.

Yet Another Search Problem

- Three missionaries and three cannibals come to a river. A rowboat that seats two is available. If the cannibals ever outnumber the missionaries on either bank of the river, the missionaries will be eaten. How shall they cross the river safely?
- How would you represent the current state of the problem?
- What are the initial and final states?
- What would be the legal moves from one state to the next?
- Find a solution to this problem

Three missionaries and three cannibals come to a river. A rowboat that seats two is available. If the cannibals ever outnumber the missionaries on either bank of the river, the missionaries will be eaten. How shall they cross the river safely? How would you represent the current state of the problem? What are the initial and final states? What would be the legal moves from one state to the next? Find a solution to this problem.
State Space

• We have already seen several examples
  – Moving the farmer, wolf, goat and cabbage from one side of a river to another
  – Start with empty water jugs and end up with a specified amount of water in one of the jugs
  – Moving missionaries and cannibals across a river without being eaten
• “Brute force” search techniques include depth first search and breadth first search

State Space - Example 1

• Tic-tac-toe is a two person game that starts with a blank 3x3 board
• Player X marks first, removing symmetry there are three choices
• Player 0 is next, there are 5 choices in two cases and only two choices in the third case
• The state space is small due to the simplicity of this game

**Definition**

**State Space Search**

A state space is represented by a four-tuple \([N,A,S,GD]\), where:

- \(N\) is the set of nodes or states of the graph. These correspond to the states in a problem-solving process.
- \(A\) is the set of arcs (or links) between nodes. These correspond to the steps in a problem-solving process.
- \(S\), a nonempty subset of \(N\), contains the start state(s) of the problem.
- \(GD\), a nonempty subset of \(N\), contains the goal state(s) of the problem. The states in \(GD\) are described using either:
  1. A measurable property of the states encountered in the search.
  2. A property of the path developed in the search.

A solution path is a path through this graph from a node in \(S\) to a node in \(GD\).
How Many Children

- How many children are there for the following state eliminating symmetric states? What are these states?

X

O

State Space - Example 2

- The eight puzzle involves a 3x3 arrangement of tiles where one tile is missing; this allows the other tiles to move.
- You start in some random state and must reach a particular goal pattern.
- Moves are labeled in terms of the “blank piece moving”, but this is really just another piece moving into the blank position.
How Many Legal Moves

• How many legal moves can be made from each position?

• How would this number be changed if we eliminated the previous state as a duplicate?

State Space - Example 3

• In the traveling salesman problem, you want to visit each city exactly once and return home

• If home is A, then there are four choices for the next city: B, C, D, or E; from each of these cities there are three choices, etc.

• The problem is to find the minimum cost route
• This problem has \((n-1)!\) complexity
Data Driven Search

• Given $P \Rightarrow Q$, start with $P$ and derive all possible $Q$ until the goal is found
• Assuming transitivity of inference (what is this?), this is called forward chaining
• Example - “I am a descendant of Thomas Jefferson”, assuming ten generations and three children per generation, the search space is $3^{10}$
• Applications
  – most data is provided in the initial problem statement (PROSPECTOR)
  – the number of realistic goals can be quickly pruned (DENDRAL)
  – it is difficult to formulate the goal ahead of time

Goal Driven Search

• Backward chaining: change $P \Rightarrow Q$ into $\neg Q \Rightarrow \neg P$, find all $\neg P$ until no more can be found, thus proving $Q$
• Example - “I am a descendant of Thomas Jefferson”, assuming ten generations and each person has two parents, the search space is $2^{10}$
• Applications
  – if the goal is easily formulated, as in MYCIN
  – if the search tree expands too quickly
  – if data is not known initially, as in medical diagnosis, backtracking can be useful in determining the data that must be collected
**A Graphical Comparison**

- Goal directed search starts at the goal and works backward to the data pruning extraneous search paths.

  ![Graphical Comparison](image1)

- Data directed search prunes irrelevant data and possible consequences until a goal is reached.

  ![Graphical Comparison](image2)

**Group Work**

- For each of the problem domains described, decide if a goal-driven or data-driven search is best. Be prepared to defend your choice as there are no right or wrong answers for this problem.

  a. Diagnosing mechanical problems in an automobile.
  b. You have met a person who claims to be your distant cousin, with a common ancestor named John Doe. You would like to verify her claim.
  c. Another person claims to be your distant cousin. He does not know the common ancestor’s name but knows that it was no more than eight generations back. You would like to either find this ancestor or determine that she did not exist.
  d. A theorem prover for plane geometry.
  e. A program for examining some readings and interpreting them, such as telling a large submarine from a small submarine from a whale from a school of fish.
  f. An expert system that will help a human classify plants by species, genus, etc.
Backtracking Graph Search

- Since there is no “oracle” to guide us directly to the goal, search strategies involve backtracking when the current path does not succeed
- Common backtracking strategies include depth-first search, breadth-first search, and heuristic search
- Some notation
  - SL: state list gives states on the current path that have been evaluated
  - NSL: new states awaiting evaluation
  - DE: dead ends, states whose descendents do not contain the goal
  - newly generated states must be checked for membership on these lists
  - CS: current state, just added to SL

Backtracking Algorithm

function backtrack:
begin
SL := [Start]: NSL := [Start]: DE := []: CS := Start: \% initialize:
while NSL ≠ [] \%while there are states to be tried
do begin
  if CS = goal (or meets goal description) then return(SL): \% on success, return list of states in path.
  if CS has no children (excluding nodes already on DE, SL, and NSL) then begin
    while SL is not empty and CS = the first element of SL
      do begin
        add CS to DE: \% record state as dead end
        remove first element from SL: \% backtrack
        remove first element from NSL:
        CS := first element of NSL:
        end;
        add CS to SL:
      end
  else begin
    place children of CS (except nodes already on DE, SL, or NSL) on NSL:
    CS := first element of NSL:
    add CS to SL
    end;
  end;
return FAIL: \% state space is exhausted.
end.
data driven, depth first

Breadth-first Search

procedure breadth_first_search;
begin
    open := [Start];
    closed := [];
    while open ≠ []
    begin
        remove leftmost state from open, call it X;
        if X is a goal then return(success)
        else begin
            generate children of X;
            put X on closed;
            eliminate children of X on open or closed;
            put remaining children on right end of open
        end
        return(failure)
        % goal found
        % loop check
        % queue
        % no states left
        end:

• We need to use a queue, a FIFO (first-in, first-out) structure, to implement a breadth-first search
• If the search space is organized level-by-level, then the nodes at the first level are visited, then the second level, then the third level, and so forth.

For a goal driven search let the goal be the root and work backward searching for a start state
Example - BFS

BFS for the Eight Puzzle

1. open = [A]; closed = [ ]
2. open = [B,C,D]; closed = [A]
3. open = [C,D,E,F]; closed = [B,A]
4. open = [D,E,F,G,H]; closed = [C,B,A]
5. open = [E,F,G,H,I,J ]; closed = [ D,C,B,A]
7. open = [G,H,I,J,K,L,M] (as L is already on open); closed = [F,E,D,C,B,A]
9. and so on until either U is found or open = [ ]
Your Own BFS

- Given the following starting state for the eight puzzle, generate the next eight states using BFS. Assume the order of moves is up, left, down, right for the “blank”; discard illegal moves or repeat states.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

- Start

Depth-first Search

procedure depth_first_search;
begin
open := [Start];
closed := [];
while open ≠ [] do
begin
   remove leftmost state from open, call it X;
   if X is a goal then return(success)
   else begin
      generate children of X;
      put X on closed;
      eliminate children of X on open or closed;
      put remaining children on left end of open
   end
end
end; return(failure)

- We need to use a stack, a LIFO (last-in, first-out) structure, to implement a depth-first search
- The search will proceed as deeply as allowed until backtracking causes the algorithm to go back up the graph to try an alternative route
Example - DFS

1. open = [A]; closed = []
2. open = [B,C,D]; closed = [A]
3. open = [E,F,C,D]; closed = [B,A]
4. open = [K,L,F,C,D]; closed = [E,B,A]
5. open = [S,L,F,C,D]; closed = [K,E,B,A]
6. open = [L,F,C,D]; closed = [S,K,E,B,A]
8. open = [F,C,D]; closed = [T,L,S,K,E,B,A]
9. open = [M,C,D], as L is already on closed; closed = [F,T,L,S,K,E,B,A]

DFS for the Eight Puzzle
Your Own DFS

• Given the following starting state for the eight puzzle, generate the next eight states using DFS. Assume the order of moves is up, left, down, right for the blank; discard illegal moves or repeat states.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

• Start

Comparison of BFS and DFS

• Breadth First Search
  – Finds a shortest path to the goal
  – Good for search spaces where it is known a simple (shallow) solution exists
  – But BFS generates a large number of open states with high branching factors
  – and BFS is not very efficient

• Depth First Search
  – Gets quickly into a deep search space
  – More efficient than BFS due to smaller set of open states
  – But DFS can easily get lost exploring a useless path
  – and DFS does not produce an optimal result
DFS with Iterative Deepening

- This strategy combines the benefits of DFS and BFS
  - The search strategy is DFS but not going any deeper than a specified depth
  - The specified depth is increased by one level for each search
  - This strategy will find an optimal solution but does not have the inefficiency associated with BFS
  - The major drawback of this strategy is that the same computation at shallow levels may be repeated since no information about the state space is retained between iterations

Duplicate Work

- Assume a search space has a uniform branching factor of four
- Fill in the following table up to level 5

<table>
<thead>
<tr>
<th>Level</th>
<th># previously visited</th>
<th># visited for the first time</th>
<th>% Duplicate Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>16</td>
<td>0.238</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- What do you think to maximum percent of duplicate work will be?
- Can you prove this mathematically?
- Is the amount of duplicate work worth the benefits of iterative deepening?
Which Search is Best?

- For each of the problem domains described, decide if a BFS or a DFS search is best. Be prepared to defend your choice as there are no right of wrong answers.

a. Diagnosing mechanical problems in an automobile.

b. You have met a person who claims to be your distant cousin, with a common ancestor named John Doe. You would like to verify her claim.

c. Another person claims to be your distant cousin. He does not know the common ancestor's name but knows that it was no more than eight generations back. You would like to either find this ancestor or determine that she did not exist.

d. A theorem prover for plane geometry.

e. A program for examining sonar readings and interpreting them, such as telling a large submarine from a small submarine from a whale from a school of fish.

f. An expert system that will help a human classify plants by species, genus, etc.

Propositional Calculus Graphs

- It is possible to express a problem in the propositional calculus using a graph

\[
q \Rightarrow p \\
r \Rightarrow p \\
v \Rightarrow q \\
s \Rightarrow r \\
t \Rightarrow r \\
s \Rightarrow u \\
s \Rightarrow t
\]

- s and t are given as true
- s and s \Rightarrow r results in r being true
- r and r \Rightarrow p results in p being true
- In general, to prove a particular proposition is true you must find a path from a start node (a "true" node) to the desired node
- p could also be proved starting from t; but q cannot be proved true
Implementation in Prolog

Script started on Thu Feb  8 13:59:30 2001
> more pg107.pl
p:-q.
p:-r.
q:-v.
r:-s.
r:-t.
u:-s.
s.
t.
v:-fail.

>pl
Welcome to SWI-Prolog (Version 3.4.2)
?- [pg107].
% pg107 compiled 0.00 sec, 2,992 bytes
Yes
?- q.
No
?- r.
Yes
?- s.
Yes
?- r.
Yes
?- v.
No
?- halt.
>
script done on Thu Feb  8 14:00:28 2001

And/Or Graph

- The graph structure shown previously represents an “or” condition, only one path had to reach p for it to be true
- An “and” condition is represented by an arc across all the components that comprise the and condition
- A hyperarc in a hypergraph have ordered pairs to represent arcs: there is a single node as the first element but a set of nodes for the second element
- \( p \land q \Rightarrow r \) is represented by
- An ordinary graph is also a hypergraph when the second set of nodes only contains one element
And/Or Graph Example

- This example includes and and or relations

```
  a
  b
  c
  a ∧ b ⇒ d
  a ∧ c ⇒ e
  b ∧ d ⇒ f
  f ⇒ g
  a ∧ e ⇒ h
```

- In this problem, everything is provable
- Suppose b were false, what other items would become false?
- A goal directed search for g would have to show f is true, which requires b to be true, it also requires d to be true, so a has to be true
- A data directed search starts with a and b, proves d, then f, and finally g

Implementation in Prolog

```
Script started on Thu Feb 8 14:04:08 2001
> more pg110.pl
a.
b.
c.
d:-a,b.
e:-a,c.
f:-b,d.
g:-f.
h:-a,e.
> pl
Welcome to SWI-Prolog (Version 3.4.2)
?- [pg110].
% pg110 compiled 0.02 sec, 3,168 bytes
Yes
?- a.
Yes
?- b.
Yes
?- c.
Yes
?- d.
Yes
?- e.
Yes
?- f.
Yes
?- h.
Yes
?- halt.
>
```

script done on Thu Feb 8 14:05:14 2001
**Modified Implementation**

Script started on Thu Feb 8 14:06:14 2001
> more pg110.pl
 a.
b:-fail.
c.
d:-a,b.
e:-a,c.
f:-b,d.
g:-f.
h:-a,e.

> pl
Welcome to SWI-Prolog (Version 3.4.2)
?- [pg110].
% pg110 compiled 0.02 sec, 3,184 bytes
Yes
?- a.
  Yes
?- b.
  No
?- c.
  Yes
?- d.
  No
?- e.
  Yes
?- f.
  No
?- g.
  No
?- h.
  Yes
?- halt.
>
script done on Thu Feb 8 14:06:58 2001

**MACSYMA**

- MACSYMA is designed to perform complex mathematical operations, such as the integration problem shown here
The Predicate Calculus

- Translation from English into the predicate calculus

1. Fred is a collie.
   \[ \text{collie}(\text{fred}). \]
2. Sam is Fred's master.
   \[ \text{master}(\text{fred}, \text{sam}). \]
3. It is Saturday.
   \[ \text{day}(\text{sat}) \]
4. It is cold on Saturday.
   \[ \neg (\text{warm}(\text{sat})) \]
5. Fred is a trained dog.
   \[ \text{trained}(\text{fred}). \]
6. Spaniels or collies that are trained are good dogs.
   \[ \forall X \left( \text{spaniel}(X) \lor \text{collie}(X) \land \text{trained}(X) \rightarrow \text{gooddog}(X) \right) \]
7. If a dog is a good dog and has a master then he will be with his master.
   \[ \forall (X,Y,Z) \left( \text{gooddog}(X) \land \text{master}(X,Y) \land \text{location}(Y,Z) \rightarrow \text{location}(X,Z) \right) \]
8. If it is Saturday and warm, then Sam is at the park.
   \[ \text{day}(\text{sat}) \land \text{warm}(\text{sat}) \Rightarrow \text{location}(\text{sam}, \text{park}) \]
9. If it is Saturday and not warm, then Sam is at the museum.
   \[ \text{day}(\text{sat}) \land \neg \text{warm}(\text{sat}) \Rightarrow \text{location}(\text{sam}, \text{museum}) \]

- We want to find out where Fred is
- Universal quantification is assumed and we don’t have any existentials we have to implement
- We use a goal directed search

Where is Fred?

- The substitutions are \{fred/X, sam/Y, museum/Z\}
- It ends up that Fred is at the museum with his master
Grammar Rules

• np is noun phrase, vp is verb phrase, n is noun, v is verb, and art is article
• The available words appear as leaves

Implementation in Prolog

sentence--> noun_phrase, verb_phrase.
noun_phrase --> noun.
noun_phrase --> article, noun.
verb_phrase--> verb.
verb_phrase --> verb, noun_phrase.
article --> [a].
article --> [the].
noun --> [man].
noun --> [dog].
verb --> [likes].
verb --> [bites].

• This is actual Prolog code that can be queried, as we will see shortly.
• This special form of Prolog rules is called “Definite Clause Grammars”
The dog bites the man

- A successful parse tree, if one exists, must be a subtree and the grammar rules
- We replace the terminals with nonterminals when possible; this is a data directed depth first parse
- It is also possible to generate sentences making appropriate substitutions to obtain the given sentence

Parsing in Prolog

- Determining whether a sentence is legal or not

```
> pl
Welcome to SWI-Prolog (Version 3.4.2)
?- [dcg].
% dcg compiled 0.00 sec, 3,968 bytes
Yes
?- sentence([the,dog,likes,the,man],[]).
Yes
?- sentence([the,man],[]).
No
?- sentence([the,dog,likes],[]).
Yes
?- sentence([dog,likes,dog],[]).
Yes
?- sentence([likes,the,dog],[]).
No
?- halt. >
```
Generation in Prolog

- Prolog will also generate all legal sentences for you (or at least until you get tired of looking at legal sentences)

```prolog
?- [dcg].
% dcg compiled 0.00 sec, 3,968 bytes
Yes
?- sentence(X,[]).
X = [man, likes] ;
X = [man, bites] ;
X = [man, likes, man] ;
X = [man, likes, dog] ;
X = [man, likes, a, man] ;
X = [man, likes, a, dog] ;
X = [man, likes, the, man]
Yes
```