Examples of problems which are solved by search

Setting the scene: your Romanian holiday search problem

How to be systematic in your search

Uninformed Search

Comparing Uninformed Search Algorithms

Informed Search

Assumptions made about the environment

Goal and Problem formulation

(Material taken from Russel and Norvig chapters 3 and 4, Callan chapter 2 and John Bullinaria's slides from last year)
Examples of problems which are solved by search

Route planning
   routing in computer networks,
   military operations planning,
   airline travel planning

Touring problems
   Travelling salesman problem

VLSI layout

Robot Navigation

Automatic assembly sequencing

Internet searching

(Russel & Norvig 67-68)
Setting the scene: your Romanian holiday search problem

You are on holiday in Romania – in a town called Arad

It is 1am in the morning and you have a car, but no map, no-one to ask advice, and you have just realised that your flight leaves from Bucharest Airport in the afternoon.

Luckily you have a satellite navigation gadget.

Unluckily it is partly broken, and will only give you limited information.

If you enter the name of any Romanian town it will give you only the details of the nearest neighbouring towns to the location you entered. You cannot get the gadget to give you details of the whole Arad to Bucharest route in one go.
Another way of viewing the Romanian holiday search problem

This problem is similar to how a software component within a satellite navigation gadget or route planning internet application might build up longer travel routes

(see http://www.theaa.com/travelwatch/planner_main.jsp for an example)

A route planner is given a starting point and an end point and works from one to the other.
First results from the sat-nav

The sat-nav gives you the names of the three towns that are immediately neighbouring Arad.

Arad

- Sibiu
- Timisoara
- Zerind

What kind of data structure are we starting to form?

What kind of data structure might the sat-nav be accessing?
State spaces and search trees

All the major towns in Romania form our state space.

The towns that we have expanded from Arad form our search tree:

- Arad
  - Sibiu
  - Timisoara
  - Zerind

The root of this search tree is the search node initial state.

It gives us three potential search paths to follow.
State spaces

In general, the state space is simply the space of all possible states, or configurations, that our system may be in. We prefer to work with some convenient representation of that state space.

There are two components to the representation of state space:

1. Static states, such as the Romanian town you are currently in.

2. Transitions between states, such as moving from a town to a neighbouring town that is connected by a road.

If states spaces are not too large they can be represented as graphs, which possess nodes and the transitions between them.

Our aim is to search for a route from initial state to goal state.
You expand your search tree – by entering Sibiu in the sat-nav

You get four new nodes in your search tree

Arad

Sibiu

Timisoara

Zerind

Arad  Fagaras  Oradea  Rimnicu Vilcea

Why does Arad appear twice in the search tree?

It is important to distinguish between state space and search tree:

- 20 states in state space
- infinite number of paths therefore infinite number of nodes in the search tree

If you were to automate this search, how would you deal with this?
What are your options for continuing this search systematically?

You get four new nodes in your search tree:

- **Arad**
  - Sibiu
  - Timisoara
  - Zerind

If you expand Fagaras, or one of the other children of Sibiu, then you might continue and undertake a **depth first search** by following a single path until it reaches the goal state or terminate.
What are your options for continuing this search systematically?

You get four new nodes in your search tree:

- Sibiu
- Timisoara
- Zerind

If you expand Timisoara, and then all the other children of Arad, then you might continue and undertake a breadth first search.
Will you stop searching when you find the first time route to Bucharest?

You might continue searching until you uncover the entire map (graph) of major towns in Romania.

Why might you want to know all possible routes to Bucharest?

If the state space you were searching was finite (in this case it is 20 towns) and you attempted a complete search, would it make any difference if you undertook depth or breadth first search?

If you didn't have time for a complete search (perhaps the sat-nav was very slow) AND the time for your actual journey really mattered, how might you change your search strategy?
Measuring problem solving performance

Completeness: Is the algorithm guaranteed to find a solution when there is one?

Optimality: Does the strategy find the optimal solution? (that is the lowest path cost)

Time complexity: how long does it take to find a solution?

Space complexity: how much memory is needed to perform the search?
You choose to search with depth first search. Will you stop searching if your path is too long?

In this problem there are only twenty towns in the state space (including Arad and Bucharest).

How long a search path should you construct before you conclude that the search path is not going to be optimal?

**Depth limited search** allows you to set a limit to the depth of the search tree that you will descend before back-tracking.
You want to combine benefits from depth and breadth first search

Depth first search possesses the benefit of lower space complexity.

Breadth first search is guaranteed to find an optimal path.

Can these two strategies be combined?
You want to combine benefits from depth and breadth first search

Depth first search possesses the benefit of lower space complexity.

Breadth first search is guaranteed to find an optimal path.

Can these two strategies be combined?

YES

**Iterative Deepening Search**

Can you guess what this does?
What benefits does it give?
Iterative Deepening Search

Iterative deepening search involves trying all possible depths in turn and stopping once a goal state is reached.

The benefits are that it is complete and optimal like Breadth First Search, but has the modest memory requirements of depth first search.

What is the obvious drawback of Iterative Deepening Search?

Under what conditions is this drawback not too serious?
Bi-directional search involves both simultaneously searching forward from the initial state and back from the goal state.
A graph of the whole state space: comparing Uninformed search strategies
### Comparing Uninformed Search Strategies

<table>
<thead>
<tr>
<th>Search Strategy</th>
<th>Complete</th>
<th>Optimal</th>
<th>Time Complexity</th>
<th>Space Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth first search:</td>
<td>yes</td>
<td>yes</td>
<td>$O(b^d)$</td>
<td>$O(b^d)$</td>
</tr>
<tr>
<td>Depth first search</td>
<td>no</td>
<td>no</td>
<td>$O(b^m)$</td>
<td>$O(bm)$</td>
</tr>
<tr>
<td>Depth limited search</td>
<td>if $l \geq d$</td>
<td>no</td>
<td>$O(b^l)$</td>
<td>$O(bl)$</td>
</tr>
<tr>
<td>depth first iterative deepening search</td>
<td>yes</td>
<td>yes</td>
<td>$O(b^d)$</td>
<td>$O(bd)$</td>
</tr>
<tr>
<td>bi-directional search</td>
<td>yes</td>
<td>yes</td>
<td>$O(b^{d/2})$</td>
<td>$O(b^{d/2})$</td>
</tr>
</tbody>
</table>

$b$ is branching factor, $d$ is depth of the shallowest solution, $m$ is the maximum depth of the search tree, $l$ is the depth limit.
How might we inform our Search Strategy

If the faulty sat-nav gave the coordinates of every town then we could use this information to gain 'as-the-crow-flies' direct (straight line) distances between locations.

How might this information be used when deciding which node to expand in our search tree?
Consider if we are at Sibiu, which node will we expand using greedy best first search? Is this the best node to expand? What extra information might help us make a better decision?
A* Search

When expanding a node $n$ in the search tree, greedy best first search used the estimated cost to get from the current state to the goal state, which we can define as $h(n)$. In our problem $h(n)$ is the straightline distance.

We also possess the sum of the cost to reach that node from the start state, which we can define as $g(n)$. In our problem this is the sum of the step costs for the search path.

For each node in the search tree, an evaluation function $f(n)$ can be defined as the sum of these functions.

$$f(n) = g(n) + h(n)$$

A* search repeatedly picks the node with the lowest $f(n)$ to expand next. It turns out that if $h(n)$ never over-estimates the cost to reach the goal A* is complete and optimal. Will $h(n)$ in the route finding problem overestimate the cost to reach the goal?
Assumptions made about the environment

Static (unchanging)

Observable (can sense its initial and current state)

Discrete (world carved up into towns)

Deterministic (no unexpected events)

(Russel & Norvig 61)
The difference between being a software component in a sat-nav and a real live tourist in Romania

Well-defined problems and solutions

Goal formulation
   tourists have many more goals than a sat-nav

Problem formulation
   the tourist may have one single goal that out-weights all the others goals. At an abstract level the two problems are the same.

(Russel & Norvig 62-64)
Summary

How to be systematic in your search

Uninformed Search

Comparing the Uninformed Search Algorithms

Informed Search

Assumptions made about the environment

Goal and Problem formulation

(Reading: Callan chapter 2 and Russell and Norvig pages 94-101 (which goes over the Romanian route finding problem for greedy best first search and A* search in more detail than these lectures))