Course Contents

Again..Selected topics for our course. Covering all of AI is impossible!

Key topics include:

- Introduction to Artificial Intelligence (AI)
- Knowledge Representation and Search
- Introduction to AI Programming
- Problem Solving Using Search: Structure & Strategy
- Exhaustive Search Algorithm
- Heuristic Search
- Techniques and Mechanisms of Search Algorithm
- Knowledge Representation Issues and Concepts
- Strong Method Problem Solving
- Reasoning in Uncertain Situations
- Soft Computing and Machine Learning
Intro

- Well-formed predicate calculus expressions describes objects and relations in problem domain
- Inference rules i.e. Modus Ponens allows infer new knowledge from predicate description
- Inferences define a space that is searched to find a solution
Search as Problem-Solving Strategy

- many problems can be viewed as reaching a goal state from a given starting point
  
  - often there is an underlying state space that defines the problem and its possible solutions in a more formal way
  
  - the space can be traversed by applying a successor function (operators) to proceed from one state to the next

  - if possible, information about the specific problem or the general domain is used to improve the search
    
    • experience from previous instances of the problem
    
    • strategies expressed as heuristics
    
    • simpler versions of the problem
    
    • constraints on certain aspects of the problem
Examples – Search as Problem Solving

• getting from home to FSKSM
  – start: home in Kolej Perdana
  – goal: UTM FSKSM.
  – operators: move three blocks, turn

○ loading a moving truck
  -start: apartment full of boxes and furniture
  -goal: empty apartment, all boxes and furniture in the truck
  -operators: select item, carry item from apartment to truck, load item

○ getting settled
  -start: items randomly distributed over the place
  -goal: satisfactory arrangement of items
  -operators: select item, move item
Motivation

- search strategies are important methods for many approaches to problem-solving
- the use of search requires an abstract formulation of the problem and the available steps to construct solutions
- search algorithms are the basis for many optimization and planning methods
Objectives

- formulate appropriate problems as search tasks
  - states, initial state, goal state, successor functions (operators), cost
- know the fundamental search strategies and algorithms
  - uninformed search
    - breadth-first, depth-first, uniform-cost, iterative deepening, bi-directional
  - informed search
    - best-first (greedy, A*), heuristics, memory-bounded, iterative improvement
- evaluate the suitability of a search strategy for a problem
  - completeness, time & space complexity, optimality
Evaluation Criteria

- formulation of a problem as search task
- basic search strategies
- important properties of search strategies
- selection of search strategies for specific tasks
- development of task-specific variations of search strategies
State space search

- Represent a problem as – A STATE SPACE
- Analyze the structure and complexity of problem and search procedures using – GRAPH THEORY
- Graph has- **nodes** represent discrete states i.e. diff. configuration of game board
- **arcs(links)** represent transition between states ie. Move in a game
Graph theory

- tool for reasoning about the structure of objects and relations
- Structure of the problem can be VISUALISED more directly.
- Invented by Swiss Mathematician Leonhard to solve “Bridges of Konigsberg Problem” (Newman 1965)
- Problem: IS THERE A WALK AROUND THE CITY THAT CROSSES EACH BRIDGE EXACTLY ONCE??
The city of Königsberg- 2 islands, 2 riverbanks and 7 bridges

Give a predicate to indicate the relation direction between nodes RB and I?

connect(X,Y,Z)
Euler Graph Theory: Graph of the Königsberg bridge system.

Represent using predicate calculus:
Connect(i1, i2, b1) Connect(i2, i1, b1)
Connect(rb1, i1, b2) Connect(i1, rb1, b2)
Connect(rb1, i1, b3) Connect(i1, rb1, b3)
Connect(rb1, i2, b4) Connect(i2, rb1, b4)
Connect(rb2, i1, b5) Connect(i1, rb2, b5)
Connect(rb2, i1, b6) Connect(i1, rb2, b6)
Connect(rb2, i2, b7) Connect(i2, rb2, b7)

EULER noted WALK WAS IMPOSSIBLE UNLESS A GRAPH HAS EXACTLY ZERO OR TWO NODES OF ODD DEGREES

Write a prolog rule to indicate a path is allowed in both direction? ADVANTAGE?

This can reduce the no. of predicates needed.
a) A labeled directed graph- arrow indicate directions

Nodes: \{a, b, c, d, e\}

Arcs: \{(a,b), (a,d), (b,c), (c,b), (c,d), (d,a), (d,e), (e,c), (e,d)\}
b) A rooted tree, exemplifying family relationships (parents, child, sibling)
- path from root to all nodes
- directed graph with arcs having one direction ~ no cycle
- i.e. game players cannot UNDO moves
**DEFINITION**

**GRAPH**

A graph consists of:

A set of *nodes* \( N_1, N_2, N_3, \ldots N_n \ldots \), which need not be finite.

A set of *arcs* that connect pairs of nodes.

Arcs are ordered pairs of nodes; i.e., the arc \((N_3, N_4)\) connects node \( N_3 \) to node \( N_4 \). This would indicate a direct connection from node \( N_3 \) to \( N_4 \) but not from \( N_4 \) to \( N_3 \), unless \((N_4, N_3)\) is also an arc, in which case the arc joining \( N_3 \) and \( N_4 \) is undirected.

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 \((N_j, N_l)\), then \( N_k \) and \( N_l \) are called *siblings*.

A *rooted* graph has a unique node \( N_s \) 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 pair \( N_i, N_{i+1} \) in the sequence represents an arc, i.e., \((N_i, N_{i+1})\), is called a *path* of length \( n - 1 \) in the graph.

On a path in a rooted graph, a node is said to be an *ancestor* of all nodes positioned after it (to its right) as well as a *descendant* of all nodes before it (to its left).

A path that contains any node 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 nodes are said to be *connected* if a path exists that includes them both.
Problem: What is the best transportation method to get from SLO to Fresno?

Experimental Approach: Try all the options out, and then decide.

Analytical Approach: Assemble essential information about the different methods, determine an evaluation method, evaluate them, and decide.
State Space Representation of a Problem

• In a graph:
  • nodes - are problem solution states
  • Arcs - are steps in problem solving

• State Space Search
  – finding a solution path from start state to goal
  – Means to determine complexity of problem
  – Chess, tic-tac-toe have exponential complexity, impossible to search exhaustively
  – Strategies to search large space need heuristic to reduce complexity
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, for example, the transition costs for the arcs of the path.

A solution path is a path through this graph from a node in S to a node in GD.
State space of the 8-puzzle generated by “move blank” operations.

State Space Representation of a Problem: 8 puzzle
State Space Representation of a Problem.. TSP

An instance of the traveling salesperson problem.
Complexity of exhaustive search is (N-1)!. What is N??

In the graph above, what is the complexity of this problem?
State Space Representation of a Problem..TSP

Search of the traveling salesperson problem. Each arc is marked with the total weight of all paths from the start node (A) to its endpoint.
State Space Representation of a Problem..TSP

An instance of the traveling salesperson problem with the **nearest neighbor** path in bold. Note that this path (A, E, D, B, C, A), at a cost of 550, is not the shortest path. The comparatively high cost of arc (C, A) defeated the heuristic.

**NN Heuristic** is a compromise when exhaustive search is impossible.
Evaluation Criteria

• completeness
  – if there is a solution, will it be found

• time complexity
  – how long does it take to find the solution
  – does not include the time to perform actions

• space complexity
  – memory required for the search

• optimality
  – will the best solution be found

main factors for complexity considerations:
branching factor $b$, depth $d$ of the shallowest goal node, maximum path length $m$
Search and Path Cost

- **the search cost** indicates how expensive it is to generate a solution
  - time complexity (e.g. number of nodes generated) is usually the main factor
  - sometimes space complexity (memory usage) is considered as well

- **path cost** indicates how expensive it is to execute the solution found in the search
  - distinct from the search cost, but often related

- **total cost** is the sum of search and path costs
Selection of a Search Strategy (Directions..)

Two directions- how a state space maybe searched

Data-Driven

-From a given data of a problem toward a goal
-also called forward chaining
-given facts, rules
-Apply rules to facts to produce new facts
-rules use new facts to produce more new facts
-search continues until a path that satisfies goal is generated

Goal-Driven

-From a goal back to data
-also called backward chaining
-take the goal, see what rules apply and which conditions are true to use
-the condition becomes subgoal
-search continues backward through rules and subgoals to the given facts
Selection of a Search Strategy

Goal-directed search—prunes extraneous paths, e.g. theorem proven, diagnosis
Selection of a Search Strategy

Data-directed search—prune irrelevant data and their consequents and determine possible goals, eg. Analyze data, interpret data i.e. find what minerals to be found at a site
Selection of a Search Strategy

- Both **goal-driven** and **data-driven** search the same state space graph.
- Order and no. of states are different.
- Preferred strategy is determined by:
  - Complexity of the rules
  - Shape of state space
  - Availability of problem data
Implementing Graph Search - Backtracking

• A technique to try all paths through state space
• Begins search at start state and continues until it reaches a GOAL or ‘DEAD END’
• If found GOAL, it quits and return solution path
• If found dead-end, it backtracks to the most recent unvisited nodes.
• Use lists to keep track nodes in state space:
  - STATE LIST (SL): current path tried
  - NEW STATE LIST (NSTL): nodes to be visited
  - DEAD END (DE): states failed nodes and eliminated from evaluation
• Advantage: keep track states in search; breadth-first, depth-first and best-first exploits this idea in backtracking
function backtrack;

begin
  SL := [Start]; NSL := [Start]; DE := []; CS := Start;
  % initialize:
  while NSL ≠ [] do
    % while there are states to be tried
    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;
        end
        add CS to SL
    end;
  return FAIL;
end.
Backtracking search of a hypothetical state space.

Initialize: SL = [A]; NSL = [A]; DE = []; CS = A;

<table>
<thead>
<tr>
<th>AFTER ITERATION</th>
<th>CS</th>
<th>SL</th>
<th>NSL</th>
<th>DE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>[A]</td>
<td>[A]</td>
<td>[]</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>[BA]</td>
<td>[BCDA]</td>
<td>[]</td>
</tr>
<tr>
<td>2</td>
<td>E</td>
<td>[EBA]</td>
<td>[EFBCDA]</td>
<td>[]</td>
</tr>
<tr>
<td>3</td>
<td>H</td>
<td>[HEBA]</td>
<td>[HEFBCDA]</td>
<td>[]</td>
</tr>
<tr>
<td>4</td>
<td>I</td>
<td>[IEBA]</td>
<td>[IEFBCDA]</td>
<td>H</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>[FBA]</td>
<td>[FBBCDA]</td>
<td>[EIH]</td>
</tr>
<tr>
<td>6</td>
<td>J</td>
<td>[JFBA]</td>
<td>[JFBCDA]</td>
<td>[EIH]</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>[CA]</td>
<td>[CDA]</td>
<td>[BFEIH]</td>
</tr>
<tr>
<td>8</td>
<td>G</td>
<td>[GCA]</td>
<td>[GCDA]</td>
<td>[BFEIH]</td>
</tr>
</tbody>
</table>

Current node
Node to visit

goal