COMP9414: Artificial Intelligence

Problem Solving and Search

Wayne Wobcke

Room J17-433

wobcke@cse.unsw.edu.au

Based on slides by Maurice Pagnucco

Introduction to Problem Solving and Search

- Search as a "weak method" of problem solving with wide applicability
- Uninformed search methods (use no problem-specific information)
- Informed search methods (use heuristics to improve efficiency)
- (Not covered) Stochastic algorithms for problem solving
- Useful for understanding Prolog programs, logical inference, natural language parsing
- References:
  - Ivan Bratko, Prolog Programming for Artificial Intelligence, Addison-Wesley, 2001. (Chapter 11)
  - Stuart J. Russell and Peter Norvig, Artificial Intelligence: A Modern Approach, Second Edition, Pearson Education, 2003. (Chapter 3)

Motivating Example

- You are in Romania on holiday, in Arad, and need to get to Bucharest.
- What more information do you need to solve this problem?
- Once you have this information, how do you go about solving the problem?
- How do you know your solution is any good? What extra information would you need in order to evaluate the quality of your solution?

State Space Search Problems

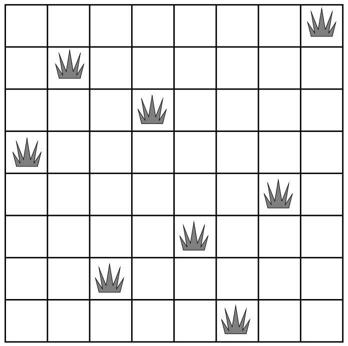
- State space — set of all states reachable from initial state by any action sequence
- Initial state — an element of the state space
- Operators — set of possible actions at agent's disposal; describe state reached after performing action in current state
- (alternatively) Successor function — $s(x)$= set of states reachable from state x by performing a single action
- Goal state — an element of the state space
- Path cost — assigns cost to a path for comparing partial solutions (apply to optimization problems)

Example Problem — 8-Puzzle

1	2	3
4	5	6
7	8	

States: location of eight tiles plus location of blank
Operators: move blank left, right, up, down
Goal state: state with tiles arranged in sequence
Path cost: each step is of cost 1

Example Problem — N-Queens



States: 0 to N queens arranged on chess board
Operators: place queen on empty square
Goal state: N queens on chess board, none attacked
Path cost: zero

Real World Problems

- Route finding — robot navigation, airline travel planning, computer/phone networks
- Travelling salesman problem — planning movement of automatic circuit board drills
- VLSI layout — design silicon chips
- Assembly sequencing — scheduling assembly of complex objects, manufacturing process control
- Mixed/constrained problems — courier delivery, product distribution, fault service and repair

These are optimization problems but mathematical (operations research) techniques are not always effective.

Problem Representation — Tic-Tac-Toe

X	O	X
X	O	O
X		O

States: arrangement of Os and Xs on 3x3 grid
Operators: place X (O) in empty square
Goal state: three Xs (Os) in a row
Path cost: zero

Tic-Tac-Toe — First Attempt

1	2	3
4	5	6
7	8	9

Board: 0=blank; 1=X; 2=O
Idea: Use move table with $3^9 = 19683$ elements
Algorithm: Consider board to be a ternary number; convert to decimal; access move table; update board

- Fast; lots of memory; laborious; not extensible

Tic-Tac-Toe — Second Attempt

1	2	3
4	5	6
7	8	9

Board: 2=blank; 3=X; 5=O
Algorithm: Separate strategy for each move.
Goal test (if row gives win on next move): calculate product of values
X: test product = 18 ($3 \times 3 \times 2$); O: test product = 50 ($5 \times 5 \times 2$)

- Not as fast as 1; much less memory; easier to understand and comprehend; strategy determined in advance; not extensible

Tic-Tac-Toe — Third Attempt

8	3	4
1	5	9
6	7	2

Board is a magic square!
Algorithm: As in attempt 2 but to check for win —keep track of player's "squares". If difference of 15 and sum of two squares is ≤ 0 or > 9 two squares are not collinear. Otherwise, if square equal to difference is blank, move there.

- What does this tell you about the way humans solve problems vs. computers?

Tic-Tac-Toe — Fourth Attempt

	?	

Board: list of board positions arising from next move; estimate of likelihood of position leading to a win
Algorithm: look at board arising from each possible move; choose "best" move

- Slower; can handle large variety of problems

Evaluating Search Algorithms

- Completeness: strategy guaranteed to find a solution when one exists?
- Time complexity: how long to find a solution?
- Space complexity: memory required during search?
- Optimality: when several solutions exist, does it find the "best"?

Note: States are constructed during search, not computed in advance, so efficiently computing successor states is critical!

Complications

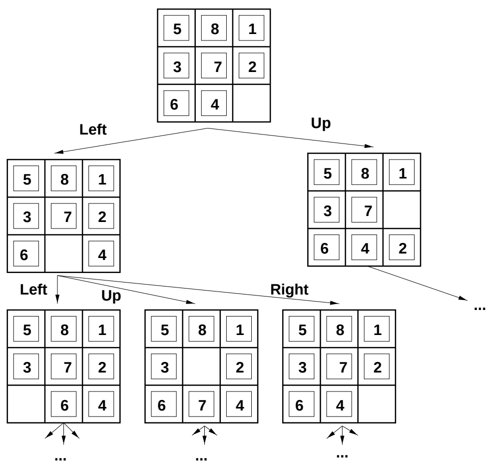
- Single-state — agent starts in known world state and knows which unique state it will be in after a given action
- Multiple-state — limited access to world state means agent is unsure exactly which world state it is in but may be able to narrow it down to a set of states
- Contingency problem — if agent does not know full effects of actions (or there are other things going on) it may have to sense during execution (changing the search space dynamically)
- Exploration problem — no knowledge of effects of actions (or state), so agent must experiment

Search methods are capable of tackling single-state and multiple-state problems though multiple state at the cost of additional complexity.

Explicit State Spaces

- View state space search in terms of finding a path through a graph
- Graph $G = (V, E)$ — V: vertices (nodes); E: edges
- Edges may have associated cost; path cost = sum edge costs in path
- Path from node s to g — sequence of nodes $s = n_0, n_1, \ldots, n_k = g$ such that n_{i-1} is connected to n
- State space graph — node represents state; arc represents change from one state to another due to action; costs may be associated with nodes and edges (hence paths)
- Forward (backward) branching factor — # out-(in-)going arcs from (to) node

Search Graph — 8-Puzzle



A General Search Procedure

function GeneralSearch(problem, strategy) **returns** a solution or failure

initialise search graph using the initial state of problem

loop

if there are no candidates for expansion **then return** failure

choose a frontier node for expansion according to strategy

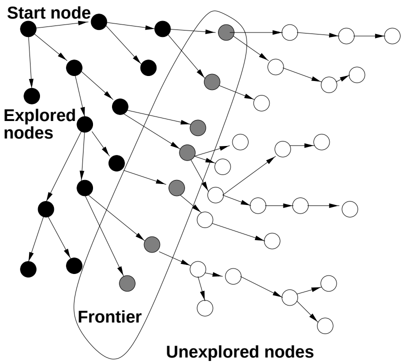
if the node contains a goal state **then return** solution

else expand the node and add the resulting nodes to the search graph

end

Note: Only test whether at goal when expanding node, not when adding nodes to the search graph.

A General Search Procedure



Search strategy — way in which frontier expands

Back to Motivating Example

- Notice assumptions built in to problem formulation (level of abstraction)
- Note that while people can "look" at the map to see a solution, the computer must construct the map by exploration
  - Where can I go from Arad?
  - Sibiu, Timisoara, Zerind
  - Where can I go from Sibiu?
- The order of questioning defines the search strategy
- Problem formulation assumptions critically affect the quality of the solution to the original problem

Conclusion

- Many "real world" problems can be viewed as search problems
- Problem representation is crucial in determining effectiveness of search as a problem-solving method
- Search algorithms can be classified into two groups: uninformed (blind) search and informed (heuristic) search