

For Friday

- Read chapter 4, sections 1-2
- Homework:
 - Chapter 3, exercise 7
 - May be done in groups.
- Note: Hang on to your homework for a few minutes.

Types of Agents

- Simple Reflex
- Model-based Reflex
- Goal-based
- Utility-based

- Learning

Homework

- Robot soccer player
- Internet book-shopping agent
- Autonomous Mars rover
- Mathematician's theorem-proving assistant

Pathfinding

Solving Problems

- Getting from the current state of the world to the state we want the world to be in.
- May or may not matter how we get there.

Problem Formulation

- Initial state
- Goal state
- Operators that change the state of the world
- Path cost – for when it matters how we get there

Toy Problems

- 8-puzzle
- N-queens
- Peg puzzle
- Farmer, wolf, goat and cabbage
- Missionaries and cannibals

More Realistic Problems

- Route finding
- Traveling Salesman Problem
- VLSI layout
- Robot navigation
- Web searching

Searching Concepts

- A state can be **expanded** by generating all states that can be reached by applying a legal operator to the state
- State space can also be defined by a **successor function** that returns all states produced by applying a single legal operator
- A **search tree** is generated by generating search nodes by successively expanding states starting from the initial state as the root

Search Node Contents

- May include
 - Corresponding state
 - Parent node
 - Operator applied to reach this node
 - Length of path from root to node (depth)
 - Path cost of path from initial state to node

General Search Function

function General-Search(*problem*, *strategy*)

returns a solution, or failure

 initialize the search tree using the initial state of *problem*

loop do

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

return failure

 choose a leaf node for expansion according to *strategy*

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

return the corresponding solution

else

 expand the node and add the resulting nodes to the search tree

end loop

end

Implementing Search Algorithms

- Maintain a list of unexpanded search nodes
- By using different strategies for ordering the list of search nodes, we implement different searching strategies
- Eg. breadth-first search is implemented using a queue and depth-first using a stack (as we'll see soon)

Search Function Revisited

```
function General-Search(problem, Queuing-Fn)  
    returns a solution, or failure  
    nodes <- MakeQueue(Make-Node(Initial-State(problem)))  
    loop do  
        if nodes is empty then  
            return failure  
        node <- Remove-Front(nodes)  
        if Goal-Test(problem) applied to State(node) succeeds then  
            return the corresponding solution  
        else  
            nodes <- Queuing-Fn(nodes, Expand(node, Operators(problem)))  
        end loop  
end
```

Properties of Search Strategies

- Completeness
- Time Complexity
- Space Complexity
- Optimality

Two Types of Search

- **Uninformed Search**

- Also called **blind**, **exhaustive** or **brute-force**
- Make use of no information about the problem
- May be quite inefficient

- **Informed Search**

- Also called **heuristic** or **intelligent**
- Uses information about the problem to guide the search
- Usually guesses the distance to a goal state
- Not always possible

Breadth-First Search

- List ordering is a queue
- All nodes at a particular depth are expanded before any below them
- How does BFS perform?
 - Completeness
 - Optimality

Complexity of BFS

- **Branching Factor**
- For branching factor **b** and solution at depth **d** in the tree (i.e. the path-length of the solution is **d**)
 - Time required is: $1 + b + b^2 + b^3 + \dots + b^d$
 - Space required is at least b^d
- May be highly impractical
- Note that **ALL** of the uninformed search strategies require exponential time

Uniform Cost Search

- Similar to breadth first, but takes path cost into account

Depth First Search

- How does depth first search operate?
- How would we implement it?
- Performance:
 - Completeness
 - Optimality
 - Space Complexity
 - Time Complexity

Comparing DFS and BFS

- When might we prefer DFS?
- When might we prefer BFS?