

For Monday

- Finish chapter 4
- No homework (Enjoy this rarity)

Program 1

Late Passes

- You have 2 for the semester.
- Only good for programs.
- Allow you to hand in up to 5 days late IF you have a late pass left.
- Each good for $+.05$ on final grade if unused.
- Must indicate that you are using a late pass in Blackboard when you submit.
- Only way to turn in late work in this course.

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?

Improving on DFS

- Depth-limited Search
- Iterative Deepening
 - Wasted work???
 - What kinds of problems lend themselves to iterative deepening?

Bi-directional Search

- What advantages are there to bi-directional search?
- What do we have to have to use bi-directional search?

Repeated States

- Problem?
- How can we avoid them?
 - Do not follow loop to parent state (or me)
 - Do not create path with cycles (check all the way to root)
 - Do not generate any state that has already been generated. -- How feasible is this??

Informed Search

- So far we've looked at search methods that require no knowledge of the problem
- However, these can be very inefficient
- Now we're going to look at searching methods that take advantage of the knowledge we have a problem to reach a solution more efficiently

Best First Search

- At each step, expand the most promising node
- Requires some estimate of what is the “most promising node”
- We need some kind of **evaluation function**
- Order the nodes based on the evaluation function

Greedy Search

- A **heuristic function**, $h(n)$, provides an estimate of the distance of the current state to the closest goal state.
- The function must be 0 for all goal states
- Example:
 - Straight line distance to goal location from current location for route finding problem

Heuristics Don't Solve It All

- NP-complete problems still have a worst-case exponential time complexity
- Good heuristic function can:
 - Find a solution for an average problem efficiently
 - Find a reasonably good (but not optimal) solution efficiently

Beam Search

- Variation on greedy search
- Limit the queue to the best n nodes (n is the **beam width**)
- Expand all of those nodes
- Select the best n of the remaining nodes
- And so on
- May not produce a solution

Focus on Total Path Cost

- Uniform cost search uses $g(n)$ --the path cost so far
- Greedy search uses $h(n)$ --the estimated path cost to the goal
- What we'd like to use instead is
$$f(n) = g(n) + h(n)$$
to estimate the **total** path cost

Admissible Heuristic

- An **admissible heuristic** is one that never overestimates the cost to reach the goal.
- It is always less than or equal to the actual cost.
- If we have such a heuristic, we can prove that best first search using $f(n)$ is both complete and optimal.
- **A* Search**

8-Puzzle Heuristic Functions

- Number of tiles out of place
- Manhattan Distance
- Which is better?
- Experiment
- Effective branching factor

Inventing Heuristics

- Relax the problem
- Cost of solving a subproblem
- Learn weights for features of the problem