What is Search?
• Search is a basic problem-solving method
• We start in an initial state
• We examine states that are (usually) connected by a sequence of actions to the initial state
• We aim to find a solution, which is a sequence of actions that brings us from the initial state to the goal state, minimizing cost

Overview
• Problem Formulation
• Uninformed Search
  – DFS, BFS, IDDFS, etc.
• Informed Search
  – Greedy, A*
• Properties of Heuristics

Problem Formulation
• Four components of a search problem
  – Initial State
  – Actions
  – Goal Test
  – Path Cost
• Optimal solution = lowest path cost to goal

Example: Path Planning
Find shortest route from one city to another using highways.

Example 8(15)-puzzle
Possible Start State
Goal State
Actions: UP, DOWN, RIGHT, LEFT
### “Real” Problems

- Robot motion planning
- Drug design
- Logistics
  - Route planning
  - Tour Planning
- Assembly sequencing
- Internet searching

### Why Use Search?

- Other algorithms exist for these problems:
  - Dijkstra’s Algorithm
  - Dynamic programming
  - All-pairs shortest path
- Use search when it is too expensive to enumerate all states
  - 8-puzzle has 362,800 states
  - 15-puzzle has 1.3 trillion states
  - 24-puzzle has $10^{25}$ states

### Basic Search Concepts

- Assume a tree-structured space (for now)
- Nodes: Places in search tree
  - (states exist in the problem space)
- Search tree: portion of state space visited so far
- Expansion: Generation of successors for a state
- Frontier: Set of states visited, but not expanded
- Branching factor: Max no. of successors = $b$
- Goal depth: Depth of shallowest goal = $d$

### Example Search Tree

![Search Tree Diagram]

### Generic Search Algorithm

```plaintext
Function Tree-Search(problem, Queueing-Fn)
    fringe = Make-Queue(Make-Node(initial-State(problem)))
    loop do
        if empty(fringe) then return failure
        node = pop(fringe)
        if Goal-Test(problem, state) then return node
        fringe = Add-To-Queue(fringe, expand(node, problem))
    end
```

Interesting details are in the implementation of Add-To-Queue

### Evaluating Search Algorithms

- Completeness: Is the algorithm guaranteed to find a solution when there is one?
- Optimality: Does the strategy find the optimal solution?
- Time complexity
- Space complexity
Uninformed Search: BFS

Frontier is a FIFO

Uninformed Search: DFS

Frontier is a LIFO

BFS Properties

- Completeness: Y
- Optimality: Y (for uniform cost)
- Time complexity: $O(b^{d+1})$
- Space complexity: $O(b^{d+1})$

DFS Properties

- Completeness: N
- Optimality: N
- Time complexity: $O(b^m)$ ($m$ = depth we hit, $m > d$?)
- Space complexity: $O(bm)$

Iterative Deepening

- Want:
  - DFS memory requirements
  - BFS optimality, completeness
- Idea:
  - Do a depth-limited DFS for depth $m$
  - Iterate over $m$

IDDDFS
**IDDFS Properties**

- Completeness: $\forall$ (whenver BFS is optimal)
- Optimality: $\exists$ (whenever BFS is optimal)
- Time complexity: $O(b^{d/2})$
- Space complexity: $O(bd)$

**IDDFS vs. BFS**

Theorem: IDDFS makes no more than twice as many node expansions for a binary tree as BFS.

Proof: Assume the tree bottoms out at depth $d$, BFS does:

$$2^{d+1} - 1$$

In the worst case, IDDFS does the following:

$$\sum_{i=0}^{d} 2^{i+1} - 1 = 2^{d+1} - 2^{i+1} - 1 = 2^{d+2} - d - 1 < 2^{d+1} - 1$$

**Bi-directional Search**

![Diagram of Bi-directional Search]

**Issues with Bi-directional Search**

- Uniqueness of goal
  - Suppose goal is on(block1,table)
  - Huge portion of state space may be considered a goal state
    (need to rework state space)
- Invertability of actions

**Informed Search**

- Idea: Give the search algorithm hints
- Heuristic function: $h(x)$
- $h(x) = \text{estimate of cost to goal from } x$
- If $h(x)$ is 100% accurate, then we can find the goal in $O(bd)$ time

**Greedy Search**

- Expand node with lowest $h(x)$
- Optimal if $h(x)$ is 100% correct
- How can we get into trouble with this?
What Price Greed?

What’s broken with greedy search?

A*

- Path cost so far: \( g(x) \)
- Total cost estimate: \( f(x) = g(x) + h(x) \)
- Maintain frontier as a priority queue
- \( O(bd) \) time if \( h \) is 100% accurate
- We want \( h \) to be an \textit{admissible} heuristic
- Admissible: never overestimates cost

A* Properties

Theorem: A* is optimal if \( h(x) \) is admissible.

Proof: Suppose a suboptimal goal node \( g_2 \) appears on the fringe. If \( C^* \) is the optimal cost, \( f(g_2) > C^* \). Since \( h \) never overestimates the cost, there must exist some unexpanded node along the optimal path that has not yet been expanded. Thus, as long as we have not yet found the optimal path, we will continue to expand nodes.

Properties of Heuristics

- \( h_2 \) dominates \( h_1 \) if \( h_2(x) > h_1(x) \) for all \( x \)
- Does this mean that \( h_2 \) is better?
- Suppose you have multiple admissible heuristics. How do you combine them?

Developing Heuristics

- Is it hard to develop admissible heuristics?
- What are some heuristics for the 8 puzzle?
- What is a general strategy for developing admissible heuristics?
Other Issues

• Graphs
  – What issues arise?
  – Monotonicity
• Non-uniform costs
• Accuracy of heuristic
• A* is optimally efficient