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, IDDS, 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 routing

Why Use Search?

- Other algorithms exist for these problems:
  - Dijkstra’s Algorithm
  - Dynamic programming
  - All-pairs shortest path

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

![Example Search Tree Diagram](image)

Generic Search Algorithm

Function Tree-Search(problem, Queuing-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:
- Optimality:
- Time complexity
- Space complexity
**Uninformed Search: BFS**

Frontier is a FIFO

```
1
2 3
4 5 6 7
```

**BFS Properties**

- Completeness:
- Optimality:
- Time complexity:
- Space complexity:

---

**Uninformed Search: DFS**

Frontier is a LIFO

```
1
2 5
3 4 6 7
```

**DFS Properties**

- Completeness:
- Optimality:
- Time complexity:
- Space complexity:

---

**Iterative Deepening**

- Want:
  - DFS memory requirements
  - BFS optimality, completeness
- Idea:
IDDFS Properties

- Completeness:
- Optimality:
- Time complexity:
- Space complexity:

IDDFS vs. BFS

Theorem: IDDFS visits no more than twice as many nodes for a binary tree as BFS.

Proof: Assume the tree bottoms out at depth d, BFS visits: $2^d - 1$

In the worst case, IDDFS does no more than:

What about b-ary trees? IDDFS relative cost is lower!

Bi-directional Search

Issues with Bi-directional Search

Informal Search

- Idea: Give the search algorithm hints
- Heuristic function: $h(x)$
- $h(x) =$ 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?

In initial state $h = 1$

$A^*$

- 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 *admissible* heuristic
- Admissible: never overestimates cost

$A^*$ Properties

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

Does $A^*$ fix the greedy problem?

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