Solving problems by searching

Chapter 3
Why Search?

- To achieve goals or to maximize our utility we need to predict what the result of our actions in the future will be.

- There are many sequences of actions, each with their own utility.

- We want to find, or search for, the best one.
Example: Romania

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest
- Formulate goal:
  - be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities or choose next city
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest
Example: Romania
Problem types

- **Static / Dynamic**
  
  Previous problem was static: no attention to changes in environment

- **Observable / Partially Observable / Unobservable**
  
  Previous problem was observable: it knew initial state.

- **Deterministic / Stochastic**
  
  Previous problem was deterministic: no new percepts were necessary, we can predict the future perfectly given our actions

- **Discrete / continuous**
  
  Previous problem was discrete: we can enumerate all possibilities
Example: vacuum world

- Observable, start in #5.
  Solution?
Example: vacuum world

- **Observable**, start in #5.  
  **Solution?** [Right, Suck]

- **Unobservable**, start in  
  \{1,2,3,4,5,6,7,8\} e.g.,  
  **Solution?**
Example: vacuum world

- Unobservable, start in \{1,2,3,4,5,6,7,8\} e.g., Solution?
  [Right,Suck,Left,Suck]
Problem Formulation

A problem is defined by four items:

- **initial state** e.g., "at Arad"
- **actions or successor function** $S(x) = \text{set of action–state pairs}$
  - e.g., $S(\text{Arad}) = \{ \langle \text{Arad} \rightarrow \text{Zerind}, \text{Zerind} \rangle, \ldots \}$
- **goal test**, e.g., $x = \text{"at Bucharest"}$, $\text{Checkmate}(x)$
- **path cost** (additive)
  - e.g., sum of distances, number of actions executed, etc.
  - $c(x,a,y)$ is the step cost, assumed to be $\geq 0$

A **solution** is a sequence of actions leading from the initial state to a goal state.
Selecting a state space

- Real world is absurdly complex
  - state space must be abstracted for problem solving

- (Abstract) state ← set of real states

- (Abstract) action ← complex combination of real actions
  - e.g., "Arad → Zerind" represents a complex set of possible routes, detours, rest stops, etc.

- For guaranteed realizability, any real state "in Arad” must get to some real state "in Zerind”

- (Abstract) solution ← set of real paths that are solutions in the real world

- Each abstract action should be "easier" than the original problem
Vacuum world state space graph

- **states?** discrete: dirt and robot location
- **initial state?** any
- **actions?** *Left, Right, Suck*
- **goal test?** no dirt at all locations
- **path cost?** 1 per action
Example: 8-Queens

- **states?** any arrangement of $n \leq 8$ queens
  - or arrangements of $n \leq 8$ queens in leftmost $n$ columns, 1 per column, such that no queen attacks any other.

- **initial state?** no queens on the board

- **actions?**
  - add queen to any empty square
  - or add queen to leftmost empty square such that it is not attacked by other queens.

- **goal test?** 8 queens on the board, none attacked.

- **path cost?** 1 per move
Example: robotic assembly

- **states?**: real-valued coordinates of robot joint angles parts of the object to be assembled
- **initial state?**: rest configuration
- **actions?**: continuous motions of robot joints
- **goal test?**: complete assembly
- **path cost?**: time to execute + energy used
Example: The 8-puzzle

- **states?**
- **initial state?**
- **actions?**
- **goal test?**
- **path cost?**

![Start State and Goal State](image)

Try yourselves
Example: The 8-puzzle

- **states?** locations of tiles
- **initial state?** given
- **actions?** move blank left, right, up, down
- **goal test?** goal state (given)
- **path cost?** 1 per move

[Note: optimal solution of \( n \)-Puzzle family is NP-hard]
Tree search algorithms

- Basic idea:
  - Exploration of state space by generating successors of already-explored states (a.k.a. expanding states).
  - Every state is evaluated: *is it a goal state?*
Tree search example
Tree search example
function TREE-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
Implementation: states vs. nodes

- A **state** is a (representation of) a physical configuration

- A **node** is a data structure constituting part of a search tree contains info such as: state, parent node, action, path cost $g(x)$, dep

The **Expand** function creates new nodes, filling in the various fields and using the **SuccessorFn** of the problem to create the corresponding states.
Search strategies

- A search strategy is defined by picking the order of node expansion

- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?

- Time and space complexity are measured in terms of
  - \( b \): maximum branching factor of the search tree
  - \( d \): depth of the least-cost solution
  - \( m \): maximum depth of the state space (may be \( \infty \))
1. (30pts) **Search** Consider a fantasy chess piece called “*jumper*”. It can move up, down, left, right, or it can stay wherever it is. Consider \( k \) such jumpers on an infinite chessboard at positions \( s_1, s_2, \ldots, s_k \). The goal is to move these jumpers as fast as possible to positions \( g_1, g_2, \ldots g_k \). In each move, you are allowed to move any number of jumpers simultaneously, but 2 or more jumpers cannot occupy the same square.

a. (5pts) Formulate the above problem as a search problem, i.e. describe a state, the initial state, an action, the goal test and a path-cost.

b. (5pts) Set \( k = 1 \), i.e. a single jumper on the board. What is the maximal branching factor for this problem?

c. (5pts) Now consider general \( k \). What is now the maximal branching factor? Remember that up to \( k \) jumpers can move simultaneously.
2. (5pts) **The 8-puzzle** Consider the 8-puzzle problem described in the book and homework.

   a. (1pt) We like to search for a solution using $A^*$-search. Describe the following aspects of the problem formulation: a) states, b) successor function, c) goal test, d) step cost, e) path cost.
Next time

- Search Strategies

Questions?