PROBLEM SOLVING AND SEARCH

CHAPTER 3
What is impact of *sequences* of Percepts?

<table>
<thead>
<tr>
<th>Percept sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A, Clean]</td>
<td>Right</td>
</tr>
<tr>
<td>[A, Dirty]</td>
<td>Suck</td>
</tr>
<tr>
<td>[B, Clean]</td>
<td>Left</td>
</tr>
<tr>
<td>[B, Dirty]</td>
<td>Suck</td>
</tr>
<tr>
<td>[A, Clean], [A, Clean]</td>
<td>Right</td>
</tr>
<tr>
<td>[A, Clean], [A, Dirty]</td>
<td>Suck</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
What is difference between agent function and agent program?

<table>
<thead>
<tr>
<th>Percept sequence</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>[A, Clean]</td>
<td>Right</td>
</tr>
<tr>
<td>[A, Dirty]</td>
<td>Suck</td>
</tr>
<tr>
<td>[B, Clean]</td>
<td>Left</td>
</tr>
<tr>
<td>[B, Dirty]</td>
<td>Suck</td>
</tr>
<tr>
<td>[A, Clean], [A, Clean]</td>
<td>Right</td>
</tr>
<tr>
<td>[A, Clean], [A, Dirty]</td>
<td>Suck</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

```
function REFLEX-VACUUM-AGENT([location, status]) returns an action
if status == Dirty then return Suck
else if location == A then return Right
else if location == B then return Left
```
Outline

◊ Problem-solving agents
◊ Problem types
◊ Problem formulation
◊ Example problems
◊ Basic search algorithms
Problem-solving agents

Restricted form of general agent:

```plaintext
function SIMPLE-PROBLEM-SOLVING-AGENT( percept ) returns an action
    static: seq, an action sequence, initially empty
    state, some description of the current world state
    goal, a goal, initially null
    problem, a problem formulation
    
    state ← UPDATE-STATE( state, percept )
    if seq is empty then
        goal ← FORMULATE-GOAL( state )
        problem ← FORMULATE-PROBLEM( state, goal )
        seq ← SEARCH( problem )
        action ← RECOMMENDATION( seq, state )
        seq ← REMAINDER( seq, state )
    return action
```

Note: this is offline problem solving; solution executed “eyes closed.”

Online problem solving involves acting without complete knowledge.
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

Find solution:
   sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest
Chapter 3
Problem types

Deterministic, fully observable $\implies$ single-state problem
Agent knows exactly which state it will be in; solution is a sequence

Non-observable $\implies$ conformant problem
Agent may have no idea where it is; solution (if any) is a sequence

Nondeterministic and/or partially observable $\implies$ contingency problem
percepts provide new information about current state
solution is a contingent plan or a policy
often interleave search, execution

Unknown state space $\implies$ exploration problem (“online”)
Example: vacuum world

Single-state, start in #5. Solution??

1

2

3

4

5

6

7

8

Chapter 3
Example: vacuum world

Single-state, start in #5. Solution??
[Right, Suck]

Conformant, start in \{1, 2, 3, 4, 5, 6, 7, 8\}
e.g., Right goes to \{2, 4, 6, 8\}. Solution??
Example: vacuum world

Single-state, start in #5. Solution??
[Right, Suck]

Conformant, start in {1, 2, 3, 4, 5, 6, 7, 8} e.g., Right goes to {2, 4, 6, 8}. Solution??
[Right, Suck, Left, Suck]

Contingency, start in #5
Murphy’s Law: Suck can dirty a clean carpet
Local sensing: dirt, location only.
Solution??
Example: vacuum world

Single-state, start in #5. Solution??
[Right, Suck]

Conformant, start in \{1, 2, 3, 4, 5, 6, 7, 8\}
e.g., Right goes to \{2, 4, 6, 8\}. Solution??
[Right, Suck, Left, Suck]

Contingency, start in #5
Murphy’s Law: Suck can dirty a clean carpet
Local sensing: dirt, location only.
Solution??
[Right, if dirt then Suck]
A problem is defined by five components:

- Initial state: e.g., \textit{In(Arad)}
- Actions: \textit{ACTIONS(s)}
  - e.g., \{Go(Sibiu), Go(Timisoara), Go(Zerind)\}
- Transition model: \textit{RESULT(s,a)}
  - e.g., \text{RESULT(In(Arad),Go(Zerind)) = In(Zerind)}
  - Successor: any state reachable from given state by single action
- Goal test, can be:
  - Explicit: e.g., \textit{x = In(Bucharest)}
  - Implicit: e.g., \textit{NoDirt(x)}
- Path cost (additive)
  - e.g., sum of distances, number of actions executed, etc.
  - \( c(s, a, s') \) is the step cost, assumed to be \( \geq 0 \)

A solution is a sequence of actions leading from 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!
Example: vacuum world state space graph

states??
actions??
goal test??
path cost??
**Example: vacuum world state space graph**

- **states??**: integer dirt and robot locations (ignore dirt amounts etc.)
- **actions??**
- **goal test??**
- **path cost??**
Example: vacuum world state space graph

states??: integer dirt and robot locations (ignore dirt amounts etc.)
actions??: Left, Right, Suck, NoOp
goal test??
path cost??
Example: vacuum world state space graph

**states??**: integer dirt and robot locations (ignore dirt amounts etc.)

**actions??**: Left, Right, Suck, NoOp

**goal test??**: no dirt

**path cost??**
Example: vacuum world state space graph

**states??**: integer dirt and robot locations (ignore dirt amounts etc.)

**actions??**: *Left*, *Right*, *Suck*, *NoOp*

**goal test??**: no dirt

**path cost??**: 1 per action (0 for *NoOp*)
Example: The 8-puzzle

Start State

Goal State

states??
actions??
goal test??
path cost??
Example: The 8-puzzle

Start State

7 2 4
5 6
8 3 1

Goal State

1 2 3
4 5 6
7 8

states: integer locations of tiles (ignore intermediate positions)
actions
goal test
path cost
Example: The 8-puzzle

![8-puzzle diagram]

- **states**: integer locations of tiles (ignore intermediate positions)
- **actions**: move blank left, right, up, down (ignore unjamming etc.)
- **goal test**
- **path cost**
Example: The 8-puzzle

**states**: integer locations of tiles (ignore intermediate positions)

**actions**: move blank left, right, up, down (ignore unjamming etc.)

**goal test**: = goal state (given)

**path cost**
**Example: The 8-puzzle**

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Start State**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Goal State**

**states**: integer locations of tiles (ignore intermediate positions)

**actions**: move blank left, right, up, down (ignore unjamming etc.)

**goal test**: = goal state (given)

**path cost**: 1 per move

[Note: optimal solution of $n$-Puzzle family is NP-hard]
Example: robotic assembly

**states**: real-valued coordinates of robot joint angles
parts of the object to be assembled

**actions**: continuous motions of robot joints

**goal test**: complete assembly with no robot included!

**path cost**: time to execute
Exercise – Problem Formulation #1

Give a problem formulation for the following:

1) Using only four colors, you have to color a planar map in such a way that no two adjacent regions have the same color

Initial state?
Actions?
Goal test?
Path cost?
Give a problem formulation for the following:

2) A 3-foot-tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. He would like to get the bananas. The room contains two stackable, movable, climbable 3-foot-high crates.

Initial state?
Actions?
Goal test?
Path cost?
Exercise – Problem Formulation #3

Give a problem formulation for the following:

3) You have 3 jugs, measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from one to another or onto the ground. You need to measure out exactly one gallon.

Initial state?
Actions?
Goal test?
Path cost?
Tree search algorithm

• Basic idea:
  – Offline, simulated exploration of state space by generating successors of already-explored states (i.e., expanding states)

**function** TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
**loop** do
  **if** the frontier is empty **then return** failure
  choose a leaf node and remove it from the frontier
  **if** the node contains a goal state **then return** the corresponding solution
  expand the chosen node, adding the resulting nodes to the frontier
function GRAPH-SEARCH(problem) returns a solution, or failure
initialize the frontier using the initial state of problem
initialize the explored set to be empty
loop do
  if the frontier is empty then return failure
  choose a leaf node and remove it from the frontier
  if the node contains a goal state
    then return the corresponding solution
  add the node to the explored set
  expand the chosen node, adding the resulting nodes to the frontier
    only if not in the frontier or explored set
Tree search example
Tree search example

[Diagram of a tree search example with nodes and edges labeled with cities such as Arad, Sibiu, Timisoara, Zerind, Oradea, Fagaras, Rimnicu Vilcea, and Lugoj.]
Chapter 328

Tree search example

- Arad
  - Sibiu
    - Arad
    - Fagaras
    - Oradea
    - Rimnicu Vilcea
  - Timisoara
    - Arad
    - Lugoj
  - Zerind
    - Arad
    - Oradea
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 includes parent, children, depth, path cost $g(x)$.

States do not have parents, children, depth, or path cost!

The **Expand** function creates new nodes, filling in the various fields and using the **SuccessorFn** of the problem to create the corresponding states.
What’s the difference between a world state, a state description, and a search node?

- **World state:**

- **State description:**

- **Search nodes:**
Search strategies

A 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/expanded
- 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 $\infty$)
**Uninformed search strategies**

Uninformed strategies use only the information available in the problem definition.

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search
Breadth-first search

Expand shallowest unexpanded node

**Implementation:**

*fringe* is a FIFO queue, i.e., new successors go at end.
Breadth-first search

Expand shallowest unexpanded node

**Implementation:**

*fringe* is a FIFO queue, i.e., new successors go at end
Breadth-first search

Expand shallowest unexpanded node

**Implementation:**

*fringe* is a FIFO queue, i.e., new successors go at end

![Tree Diagram]

- A
  - B
    - D
    - E
  - C
    - F
    - G
Breadth-first search

Expand shallowest unexpanded node

**Implementation:**

*fringe* is a FIFO queue, i.e., new successors go at end
Properties of breadth-first search

Complete??
Properties of breadth-first search

Complete? Yes (if $b$ is finite)

Time??
Properties of breadth-first search

Complete?? Yes (if $b$ is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in $d$

Space??
Properties of breadth-first search

**Complete** Yes (if $b$ is finite)

**Time** $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in $d$

**Space** $O(b^{d+1})$ (keeps every node in memory)

**Optimal**
Properties of breadth-first search

**Complete**? Yes (if $b$ is finite)

**Time**? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in $d$

**Space**? $O(b^{d+1})$ (keeps every node in memory)

**Optimal**? Yes (if cost = 1 per step); not optimal in general

Space is the big problem; can easily generate nodes at 100MB/sec so 24hrs = 8640GB.
Uniform-cost search

Expand least-cost unexpanded node

**Implementation:**

\[ \text{fringe} = \text{queue ordered by path cost, lowest first} \]

Equivalent to breadth-first if step costs all equal

**Complete??** Yes, if step cost \( \geq \epsilon \)

**Time??** \# of nodes with \( g \leq \) cost of optimal solution, \( O(b^{[C^*/\epsilon]}) \)

where \( C^* \) is the cost of the optimal solution

**Space??** \# of nodes with \( g \leq \) cost of optimal solution, \( O(b^{[C^*/\epsilon]}) \)

**Optimal??** Yes—nodes expanded in increasing order of \( g(n) \)
Depth-first search

Expand deepest unexpanded node

**Implementation:**

fringe = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

**Implementation:**

\[\text{fringe} = \text{LIFO queue, i.e., put successors at front}\]
Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

Implementation:

*fringe* = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front

![Depth-first search diagram](image)
Depth-first search

Expand deepest unexpanded node

**Implementation:**

\(\text{fringe} = \text{LIFO queue, i.e., put successors at front}\)
Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

**Implementation:**

*fringe* = LIFO queue, i.e., put successors at front
Properties of depth-first search

Complete??
Properties of depth-first search

**Complete??** No: fails in infinite-depth spaces, spaces with loops

- Modify to avoid repeated states along path
  - \(\Rightarrow\) complete in finite spaces

**Time??**
Properties of depth-first search

**Complete??** No: fails in infinite-depth spaces, spaces with loops
   Modify to avoid repeated states along path
   \[ \Rightarrow \text{complete in finite spaces} \]

**Time??** \( O(b^m) \): terrible if \( m \) is much larger than \( d \)
   but if solutions are dense, may be much faster than breadth-first

**Space??**
Properties of depth-first search

**Complete**?? No: fails in infinite-depth spaces, spaces with loops
  Modify to avoid repeated states along path
  \[ \Rightarrow \text{complete in finite spaces} \]

**Time**?? \( O(b^m) \): terrible if \( m \) is much larger than \( d \)
  but if solutions are dense, may be much faster than breadth-first

**Space**?? \( O(bm) \), i.e., linear space!

**Optimal**??
Properties of depth-first search

**Complete**? No: fails in infinite-depth spaces, spaces with loops
  
  Modify to avoid repeated states along path
  
  ⇒ complete in finite spaces

**Time**? \( O(b^m) \): terrible if \( m \) is much larger than \( d \)
  
  but if solutions are dense, may be much faster than breadth-first

**Space**? \( O(bm) \), i.e., linear space!

**Optimal**? No
Depth-limited search

- Same as depth-first search, but with depth limit $l$
  - i.e., nodes at depth $l$ have no successors

**function** `DEPTH-LIMITED-SEARCH (problem, limit)` **returns** a solution, or failure/cutoff

  return Recursive-DLS(MAKE-NODE(problem, INITIAL-STATE), problem, limit)

**function** `RECURSIVE-DLS(node, problem, limit)` **returns** a solution, or failure/cutoff

  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else
    cutoff_occurred? ← false
    for each action in problem.ACTIONS(node.STATE) do
      child ← Child-Node(problem, node, action)
      result ← Recursive-DLS(child, problem, limit – 1)
      if result = cutoff then cutoff_occurred? ← true
      else if result ≠ failure then return result
    if cutoff_occurred? then return cutoff else return failure
Iterative deepening search

function Iterative-Deepening-Search(problem) returns a solution
  inputs: problem, a problem
  for depth ← 0 to ∞ do
    result ← Depth-Limited-Search(problem, depth)
    if result ≠ cutoff then return result
  end
Iterative deepening search $l = 0$
Iterative deepening search $l = 1$
Iterative deepening search $l = 2$

Limit = 2

Chapter 3
Iterative deepening search $l = 3$

Limit = 3
Properties of iterative deepening search

Complete??
Properties of iterative deepening search

Complete?? Yes

Time??
Properties of iterative deepening search

Complete? Yes

Time? \((d + 1)b^0 + db^1 + (d - 1)b^2 + \ldots + b^d = O(b^d)\)

Space?
Properties of iterative deepening search

Complete? Yes

Time? \((d + 1)b^0 + db^1 + (d - 1)b^2 + \ldots + b^d = O(b^d)\)

Space? \(O(bd)\)

Optimal??
Properties of iterative deepening search

**Complete?** Yes

**Time?** \((d + 1)b^0 + db^1 + (d - 1)b^2 + \ldots + b^d = O(b^d)\)

**Space?** \(O(bd)\)

**Optimal?** Yes, if step cost = 1

Can be modified to explore uniform-cost tree

Numerical comparison for \(b = 10\) and \(d = 5\), solution at far right leaf:

\[
N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450
\]

\[
N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,999,999 = 111,110
\]

IDS does better because other nodes at depth \(d\) are not expanded.

BFS can be modified to apply goal test when a node is generated.
# Summary of algorithms

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes*</td>
<td>Yes* #</td>
<td>No</td>
<td>Yes, if $l \geq d$ *</td>
<td>Yes *</td>
</tr>
<tr>
<td>Time</td>
<td>$b^{d+1}$</td>
<td>$b^{C^*/\epsilon}$</td>
<td>$b^m$</td>
<td>$b^l$</td>
<td>$b^d$</td>
</tr>
<tr>
<td>Space</td>
<td>$b^{d+1}$</td>
<td>$b^{C^*/\epsilon}$</td>
<td>$b^m$</td>
<td>$bl$</td>
<td>$bd$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes**</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes**</td>
</tr>
</tbody>
</table>

* If $b$ is finite  
# Complete if step costs positive  
** Optimal if step costs are all identical
Bidirectional search

- Run two simultaneous searches – one forward from initial state, and the other backward from the goal
  - We hope they meet in the middle!
- Implemented by changing goal test with a check to see whether the frontiers of the two searches intersect
  - If they do, then a solution is found
- Difficult to use when there are multiple possible goal states

- **Complete?** Yes (if $b$ is finite, and both directions use BFS)
- **Time:** $O(b^{d/2})$
- **Space:** $O(b^{d/2})$
- **Optimal?** Yes (if step costs are identical and both directions use BFS)
Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!

A
B
C
D

A
B
C
C
C
C

Chapter 3
Exercise – Search

• Consider the search space below, where S is the start node and G1, G2, and G3 satisfy the goal test. Arcs are labeled with the cost of traversing them. *Nodes are removed from fringe in alphabetical order.*
Exercise – Search (con’t)

- **Iterative Deepening:**
  - What is order that nodes are removed from fringe?
  - Which goal state is reached?
Exercise – Search (con’t)

• **Iterative Deepening:**
  
  – What is order that nodes are removed from fringe?  
  – Which goal state is reached?
Exercise – Search (con’t)

- **Iterative Deepening:**
  - What is order that nodes are removed from fringe? **S S**
  - Which goal state is reached?
Exercise – Search (con’t)

• **Iterative Deepening:**
  – What is order that nodes are removed from fringe? **S S A**
  – Which goal state is reached?
Exercise – Search (con’t)

- **Iterative Deepening:**
  - What is order that nodes are removed from fringe? S S A B
  - Which goal state is reached?
Exercise – Search (con’t)

• **Iterative Deepening:**
  – What is order that nodes are removed from fringe? S S A B C
  – Which goal state is reached?
Exercise – Search (con’t)

• **Iterative Deepening:**
  – *What is order that nodes are removed from fringe?*  
    S S A B C S
  – *Which goal state is reached?*
Exercise – Search (con’t)

• **Iterative Deepening:**
  – What is order that nodes are removed from fringe?  **S S A B C S A**
  – Which goal state is reached?
Exercise – Search (con’t)

• **Iterative Deepening:**
  – What is order that nodes are removed from fringe? \( S \ S \ A \ B \ C \ S \ A \ E \)
  – Which goal state is reached?
Exercise – Search (con’t)

• **Iterative Deepening:**
  – *What is order that nodes are removed from fringe?* S S A B C S A E G1
  – *Which goal state is reached?* G1
Summary

Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored.

Variety of uninformed search strategies.

Iterative deepening search uses only linear space and not much more time than other uninformed algorithms.

Graph search can be exponentially more efficient than tree search.

Iterative deepening is the preferred uninformed search method when the search space is large and the depth of the solution is not known.