IS-ZC444: ARTIFICIAL INTELLIGENCE

Lecture-04: Problem Solving by Search

Dr. Kamlesh Tiwari
Assistant Professor
Department of Computer Science and Information Systems,
BITS Pilani, Pilani, Jhunjhunu-333031, Rajasthan, INDIA

Sept 06, 2020 FLIPPED (WILP @ BITS-Pilani Jul-Nov 2020)

Recap

- AI attempts to build intelligent entities called **Agents** that are mostly **rational**
- Rationality is not **perfection.** Tabulation of moves is not possible.
- System is described by **P**erformance, **E**nvironment, **A**ctuators, **S**ensors (PEAS)
- Environment could be **Fully Observable** vs **Partially Observable**, **Single agent** vs **Multi agent**, **Deterministic** vs **Stochastic**, **Episodic** vs **Sequential**, **Static** vs **Dynamic**, **Discrete** vs **Continuous**, **Known** vs **Unknown**
- Four basic kind of agents are: Simple reflex, model based, goal based, and utility based
- A General Learning Agent has **Critic** to determine how agent is doing, **Learning agent** to make rules to improve/adapt, and **Problem Generator** to suggest experiments under different condition.

Problem-Solving Agent

Reflex agents cannot operate well if needs planning (or large table).

- A goal-based agent called **problem-solving agent** uses state of the world (cumulative)
- Uninformed search algorithms are not given any information about the problem other than its definition. They work, but may not efficiently (performance measure is always a concern for being intelligent).

Consider an agent enjoying holiday in Arad, Romania

What are performance measure? improve suntan, improve Romanian, sight seeing *etc*.

What if he has a flight from Bucharest next day?
Adopt the goal of getting Bucharest on time.

Problem-Solving Agent

- **Goal** is some world-state.
- Agent's task is to find out how to act[1] now and in future.
- In our example[2] let three roads lead out of Arad, one towards Sibu, one to Timisoara, and one to Zerind. None of these achieves the goal. (he needs some familiarity with the geography of Romania *i.e.* environment)
- A map can specify the environment.
- Here environment is
  observable (agent always knows his current state)
  discrete (agent have finitely many actions to take)
  known (agent knows which action takes to which state)
  deterministic (each action has only one outcome)

Here solution to a problem corresponds to a fixed sequence of actions.

[1] Actions could be abstract, like goto A to B (not move 5 step, rotate 10 degree ...)
[2] Agent want to go to Bucharest

Search

- The process of looking for a sequence of actions that reaches to goal is **search**
- How to look for, is an important question

Problem is defined using five components

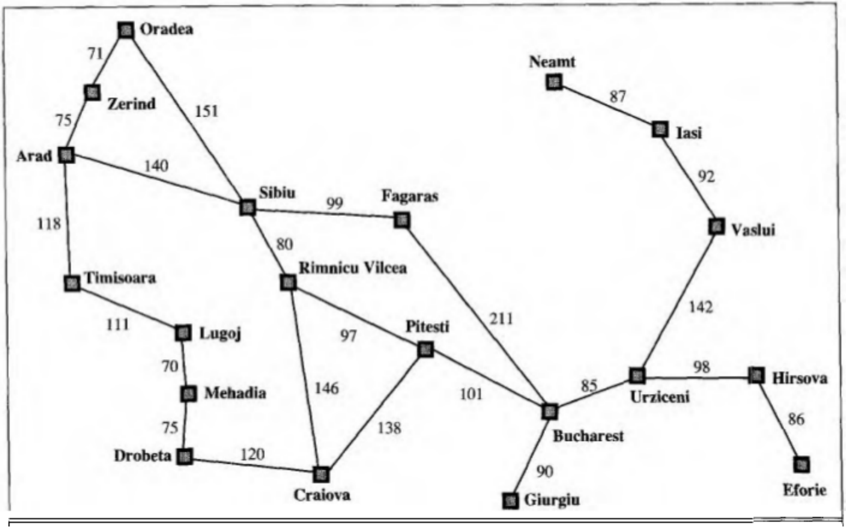
1. The **initial state**: $in(Arad)$
2. Set of **actions**: $\{go(Subiu), go(Timisoara), go(Zerind)\}$
3. **Transition model**: $Result(in(Arad), go(Zerind)) = in(Zerind)$
4. **Goal test**: $\{in(Bucharest)\}$
5. **Path cost**: used to determine efficiency

Problem-Solving Agent

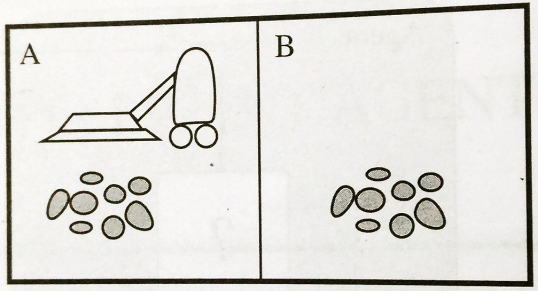
```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   inputs: percept, a percept
   static: seq, an action sequence, initially empty
           state, some description of the current world state
           goal, a goal, initially null
           problem, a problem formulation

   state ← UPDATE-STATE(state, percept)
   if seq is empty then do
       goal ← FORMULATE-GOAL(state)
       problem ← FORMULATE-PROBLEM(state, goal)
       seq ← SEARCH(problem)
   action ← FIRST(seq)
   seq ← REST(seq)
   return action
```

Problem-Solving Agent



Problem-Formulation: Toy Vacuum Cleaner



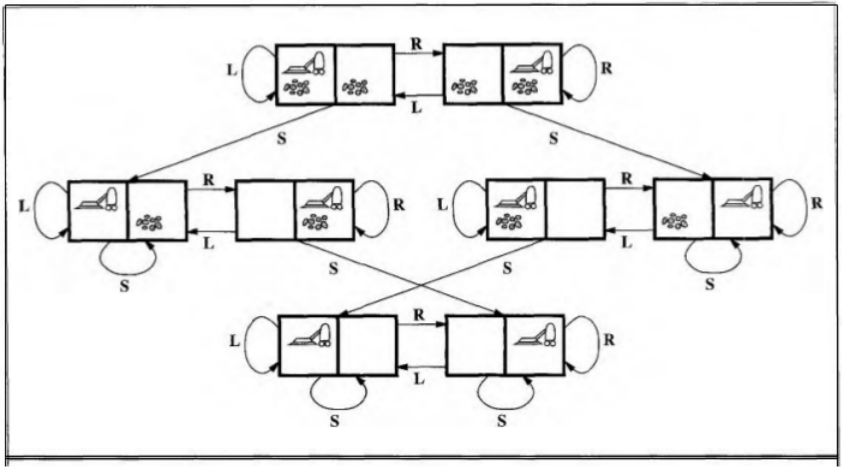
Two cells, dirt/or-not. Can sense dirt and move

- Move L and R, and suck S
- How many states? (room A/B, noDirt/oneRoomDirt/twoRoomDirt) $2 \times 2^2 = 8$

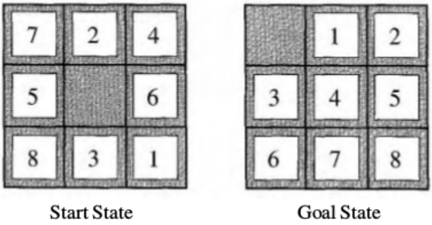
Determine the state space.

Problem-Formulation: Toy Vacuum Cleaner

The state space..



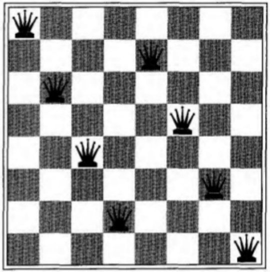
Problem-Formulation: 8-Puzzle



Determine

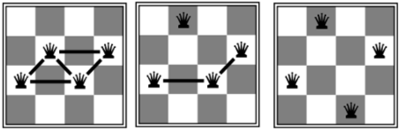
- States? is it 9!
- Initial State
- Actions
- Transition Model
- Goal Test
- Path cost

Problem-Formulation: 8-queens



Determine

- States
- Initial State
- Actions
- Transition Model
- Goal Test



h = 5 h = 2 h = 0

Problem-Formulation: Knuth conjuncture

Using only a number 4, one can reach to any desired positive number, by applying a sequence of factorials, square root and floor operation

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

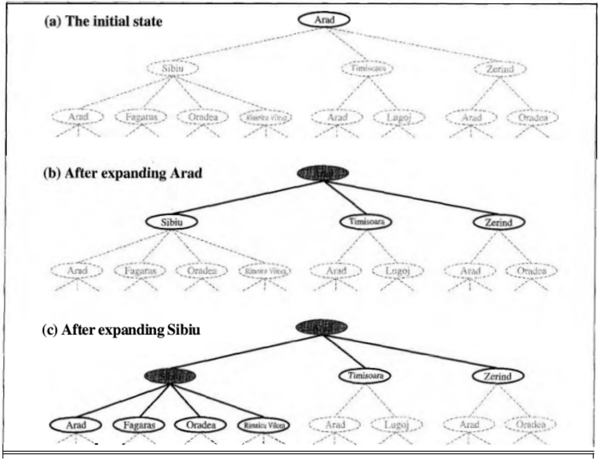
Determine

- States
- Initial State
- Actions
- Transition Model
- Goal Test

More Problems (real world)

- Route finding
- Touring problem: visit each city
- TSP: touring with single visit of cities
- VLSI layout
- Robot navigation
- Automatic assembly sequencing

Searching for Solution: search tree

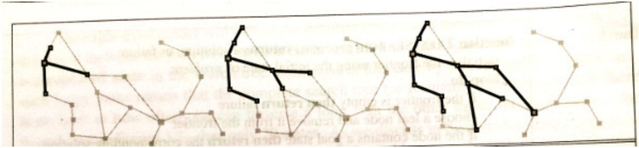


Searching for Solution: Algorithm

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

---

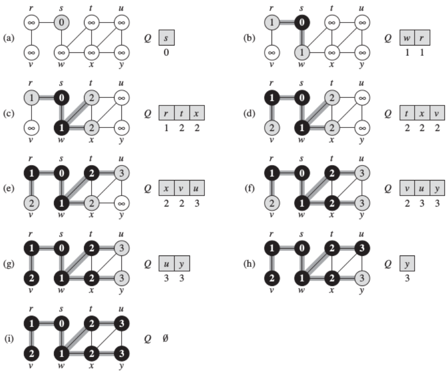
function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the 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



Uninformed Search Strategies (blind search): BFS

Breadth-first search root node is expanded first then all its successors.

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

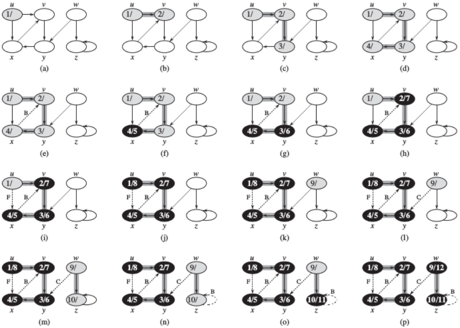


Uninformed Search Strategies (blind search): DFS

Depth-first search goes deep to branches first

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```



Thank You!

Thank you very much for your attention!

Queries ?

(Reference[3])

[3] Book - *AIMA*, ch-03, Russell and Norvig., and Book - Introduction to Algorithms by Cormen ch-22