Problem Solving and Search

- Reading: Russell and Norvig, ch. 3
    - The material here is based on Russell's slides as well as a version of the slides at UC Irvine: https://www.ics.uci.edu/~rickl/courses/cs-171/cs171-lecture-slides/cs-171-02-IntroSearch.pdf
- Problem-solving agents
- Problem types
- Problem formulation
- Example problems
- Basic search algorithms

Problem-solving agents (code as function)

```
function Simple-Problem-Solving-Agent(percept) returns
an action
```

- Decision-making = choosing an action

```
  static:
    seq, an action sequence, initially empty
```

- Generate plan to reach goal as sequence of actions

```
    state, some description of the current world state
```

- state represents agent + environment

```
    goal, a goal, initially null
```

- What agent should try to accomplish

```
    problem, a problem formulation
```

Problem-solving agents (code as function)

```
state ← Update-State(state, percept)
```

- Next state based on current state and input

```
if seq is empty then
  goal ←Formulate-Goal(state)
```

- Goal determined by environment

```
  problem ←Formulate-Problem(state, goal)
  seq ← Search( problem)
```

- Search for not just goal, but path to goal

```
action ←Recommendation(seq,state)
seq ← Remainder(seq,state)
```

- Taking an action affects plan

```
return action
```

Offline vs. online problem solving

- Above code is “offline”
  - Entire environment is known
- “online” problem solving = will be learning about environment during solution of problem

Example for offline solution

- Determine how to travel from one city to another
  - What makes an offline solution viable?
  - What real-world events might affect solution?
- Specifics: in Arad, Romania, flight leaves Bucharest tomorrow
  - State = Romanian road system, current city (in Arad)
    - Should probably consider other options like bus/train/taxi, but there may be other unmentioned factors, like needing to return a rental car
    - Problem makes location discrete: only make choices at cities
  - Formulate-Goal: get to Bucharest (in time to catch flight)
  - Formulate-Problem: agent’s state = current city

Example for offline solution

- Formulate-Problem
  - Agent's state = current city
  - Action = drive from one city to another
- Solution/Plan = sequence of cities
- Graph from text is online in many places – ex: http://robotics.cs.tamu.edu/dshell/cs420/images/map2.gif

Problem types

- Can model this problem several ways:
  - Deterministic, fully observable (“single state”)
    - Agent is always in known city, generate solution as sequence of cities
    - As UCI slides point out, if this is the case, Dijkstra’s algorithm can be used
    - Can also see situation as not capable of handling random events
  - Non-observable
    - Do know the possible states, but not which is the current state
    - Agent may lack sensors, knowledge of its own location, must still find solution
      - Ex: agent loses GPS signal and is moved
      - May be more useful to think of Roomba (automated vacuum cleaning)

Problem types

- Can model this problem several ways:
  - Non-deterministic, partially observable
    - Plan based on current knowledge
    - Update plan when new percepts update knowledge
      - Typical sequence: plan, one action, update plan, one action, update plan, …
  - Unknown state space
    - Differences from non-observable
      - Can discover environment (sensors)
    - “online”

Vacuuming 2 squares

- Single state
  - Based on which squares are dirty (agent knows), vacuum current square if needed, then move to other square and vacuum
- Non-observable
  - Vacuum both squares
  - Goal is to have both squares clean, but you cannot tell which squares are clean, so vacuum just in case

Vacuuming 2 squares

- Nondeterministic
  - Can tell if necessary to vacuum current square, but may need to move to other square to check (dust does accumulate after a while)
- Unknown state space
  - Not applicable here
  - Compare to starting up vacuum robot in new house

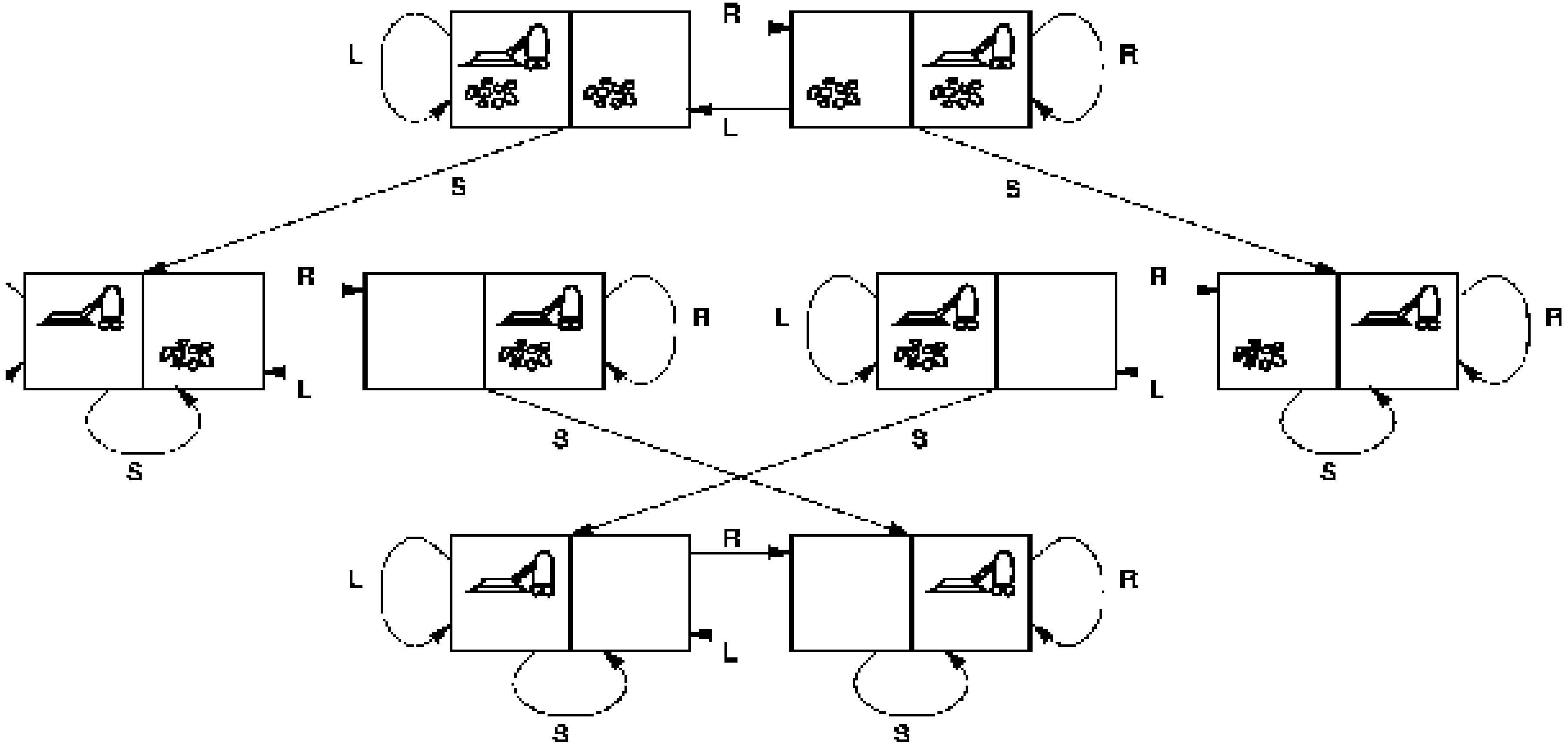
Problem formulation

- Single state problems – need to define:
  - Initial state
  - Successor function
    - Set of (action, resulting state)
    - Again, possible because of extent of knowledge
  - Goal test
    - How does agent know if it’s done?
  - Path cost
    - Cost of sequence of actions
    - Should be additive, generally nonnegative
- Solution = sequence of actions to get from initial state to a goal state

Modeling the real world

- Too many states, attributes to feasibly represent
  - Must abstract – represent many real world states with each abstract state
    - Vacuum example treats area as 2 positions
    - ☺ represents happy face for any person
  - Abstract action = combination of real actions
    - For vacuum to move, series of electrical and mechanical actions
    - For travel from one city to another, must travel along several roads
    - To guarantee plan would work in real world, action must work for any real state → must end in some real state modeled by abstract resulting state
    - Should be simpler than original problem (supports divide and conquer)
  - Abstract solution corresponds to set of sequences of real actions

Vacuum world state space (scan of textbook figure)

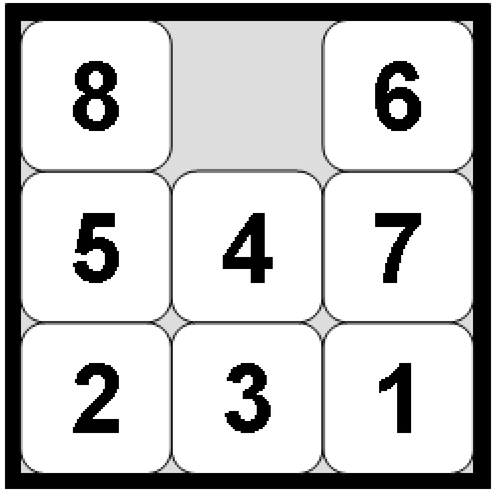


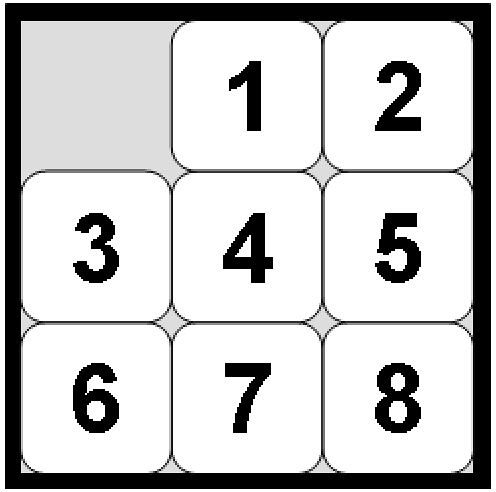
Vacuum world state space

- State = (robot location, [dirt in left pos?, dirt in right pos?])
  - Robot can only be in left or right box
  - Amount of dirt is not measured
- Actions = {L = left, R = right, S = suck, NoOp}
  - NoOp’s are not in this diagram – how might they fit in?
- Goal test = no dirt, so either bottom state is a goal state
- Path cost = 1 per actions L, R, S
  - = 0 for NoOp

8-puzzle

- From http://www.aiai.ed.ac.uk/~gwickler/images/8-puzzle-states.png
- Left is a possible start state, right is a possible goal state





8-puzzle

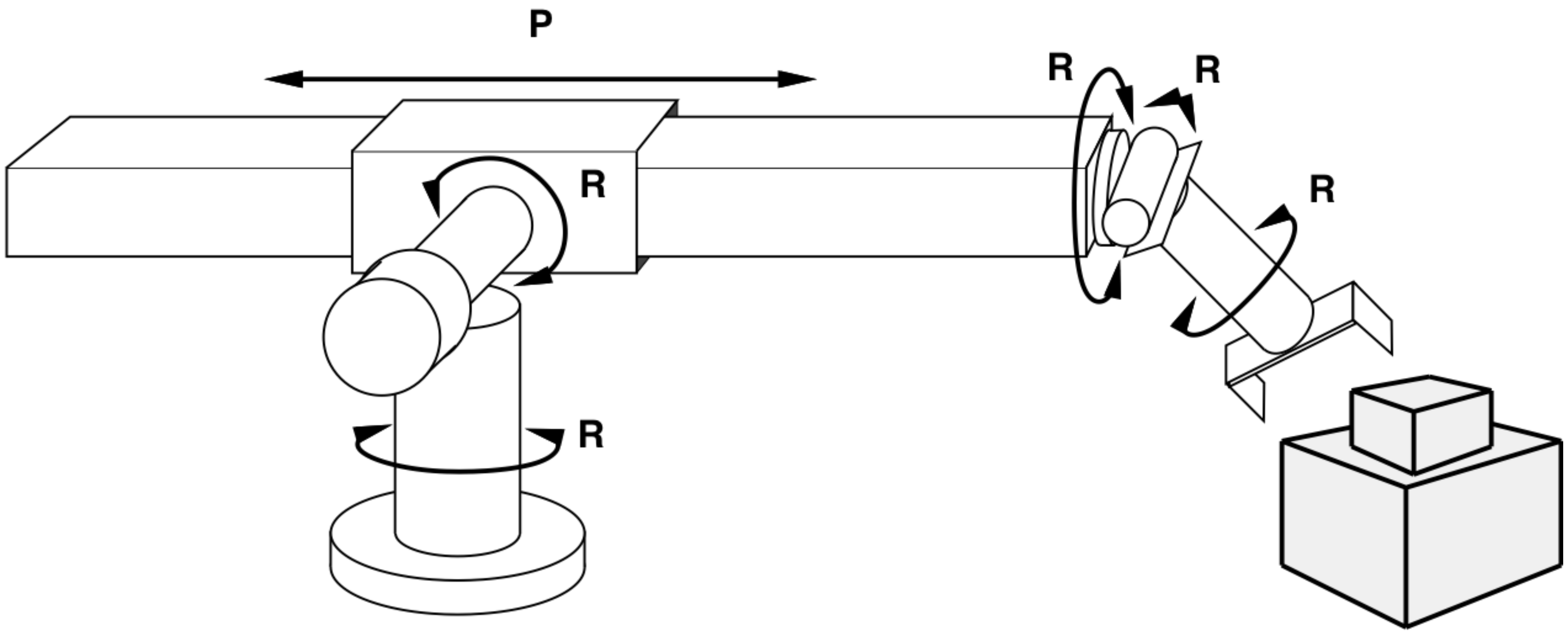
- http://www.cs.cmu.edu/afs/andrew/course/15/381-f08/www/lectures/HandoutUninformedSearch.pdf notes following:
  - On average, 22 moves enough to reach solution
  - But 1.8×10^5 states (15-puzzle has 1.3×10^{12})
    - Representing every state is probably not desirable (though feasible)
- Note that $1.8 \times 10^5 < 9!$: https://en.wikipedia.org/wiki/15_puzzle notes that Johnson & Story (1879 – this is a very old problem) proved half of the possible states cannot reach the goal state

8-puzzle

- States: the values (1-8, blank) in some fixed order
  - Ex: 8, 0, 6, 5, 4, 7, 2, 3, 1 (with 0 for blank)
  - Only model position after each move is complete
- Actions: based on “moving” blank left, right, up, down
- Goal test: if goal state has been reached
- Path cost = 1 per move
  - Finding optimal path is NP-hard

Robotic assembly

- From text: http://chalmersgu-ai-course.github.io/AI-lecture-slides/img/stanford-arm+blocks.png



Robotic assembly

- States: angles of robotic arm joints (need floating point precision) plus parts to be assembled (need positions, maybe orientation)
- Initial state: initial angles, positions/orientations
- Actions: movement of joints
- Goal test: if parts are correctly assembled (and may want joints at specific angles to prepare for next assembly/avoid contact)
- Path cost: time to assemble (UCI slides point out energy cost may need to be included)

Tree search

- Each node stores a state
  - Root based on initial state
- Children nodes are states reachable by an action
  - Search tree by *expanding* leaf node to generate new nodes (states)
  - How is this tree different from a binary search tree? -- Number of children? How much of tree is known?
  - Can often end up returning to same state
- Each node can be evaluated to see if it is a goal state
  - Search fails if no more nodes to expand and no goal state found
  - Can continue after a goal state is found to try to find better solution

Duplicate states

- Not always feasible to store all visited states to avoid expanding duplicate states
  - Ex: (from UCI slides), even 8-puzzle has 9! states = 362,880
- Feasible option, store states along path to root (in hash table)

Search Strategies

- Differ in choices of order of expanding nodes
- Can be compared on:
  - *Completeness* – does it always find a solution node if one exists?
  - Time complexity – biggest factor is number of nodes generated
  - Space complexity – number of nodes can be infinite, fitting nodes in memory also a concern
  - Optimality – guaranteed to find best solution?
- How to measure time and space complexity
  - Max branching factor = number of children
  - Depth of least cost solution
  - Max depth of state space