RN, Chapter 3.1 – 3.3

Search Problems

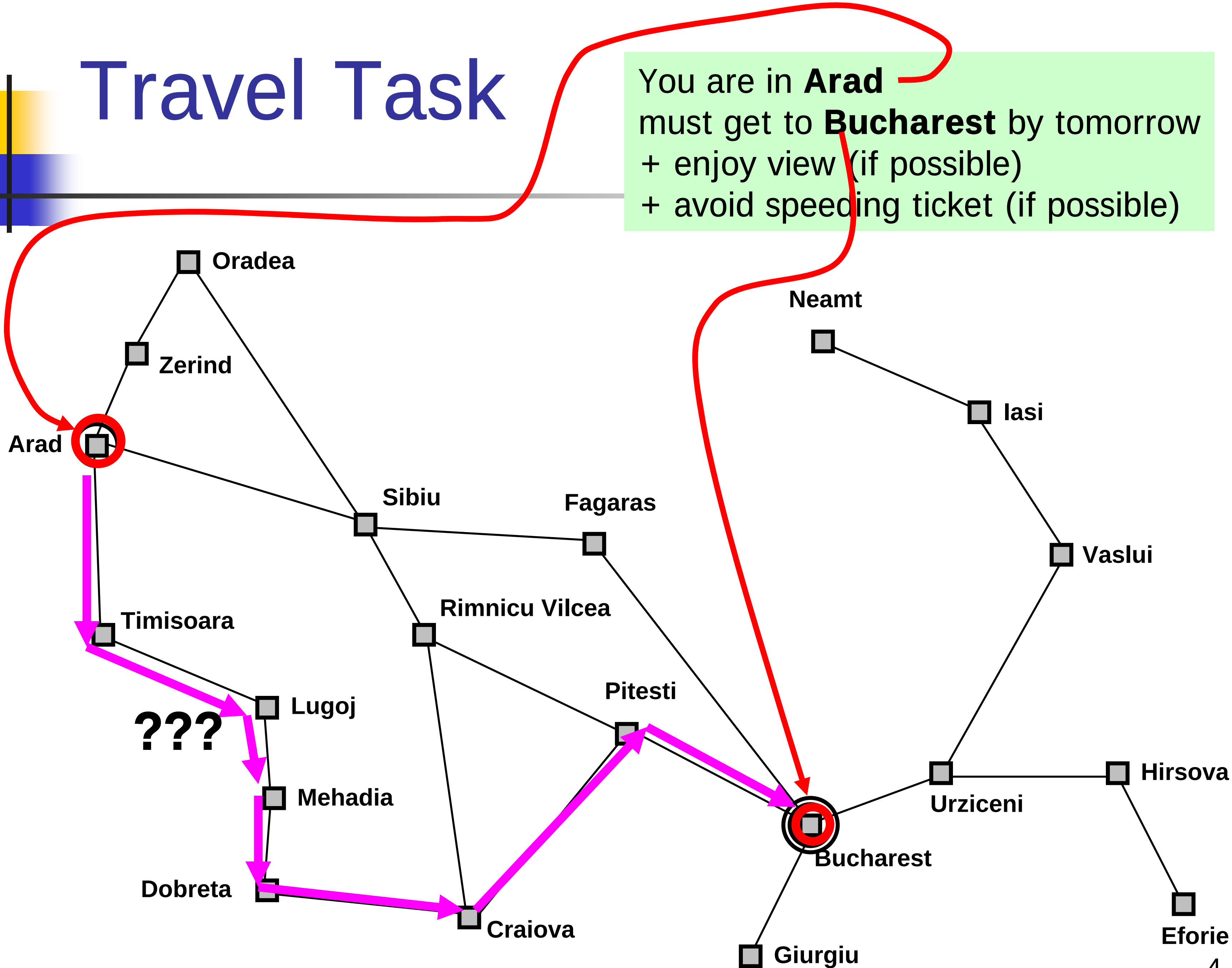
Some material from: D Lin, J You, JC Latombe

Search Overview

- Introduction to Search
  - Why search?
  - Search Problem
  - Representation
  - Examples
- Blind Search Techniques
- Heuristic Search Techniques
- Stochastic Algorithms
- Game Playing search
- Constraint Satisfaction Problems

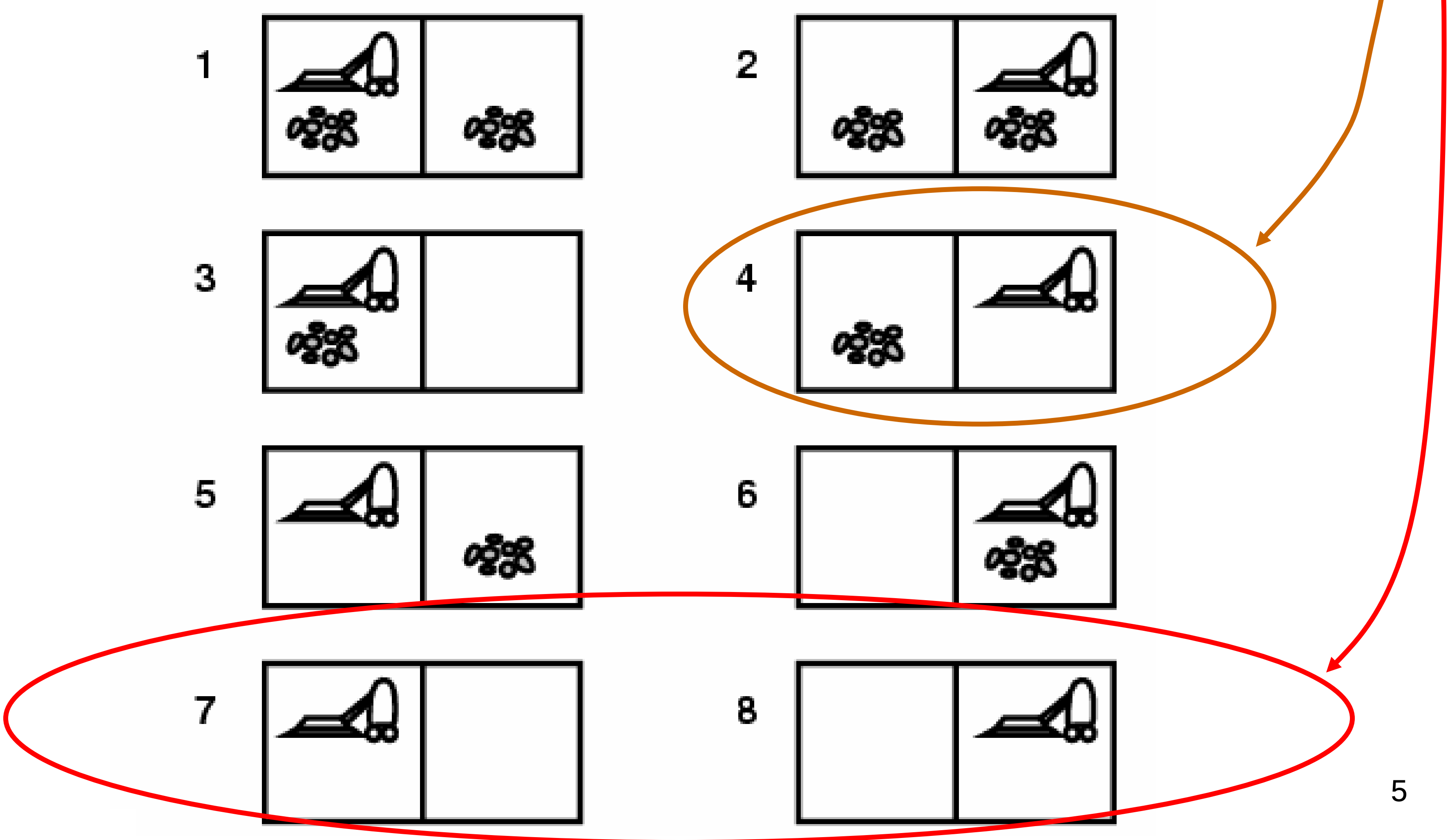
Travel Task

You are in **Arad**
must get to **Bucharest** by tomorrow
+ enjoy view (if possible)
+ avoid speeding ticket (if possible)

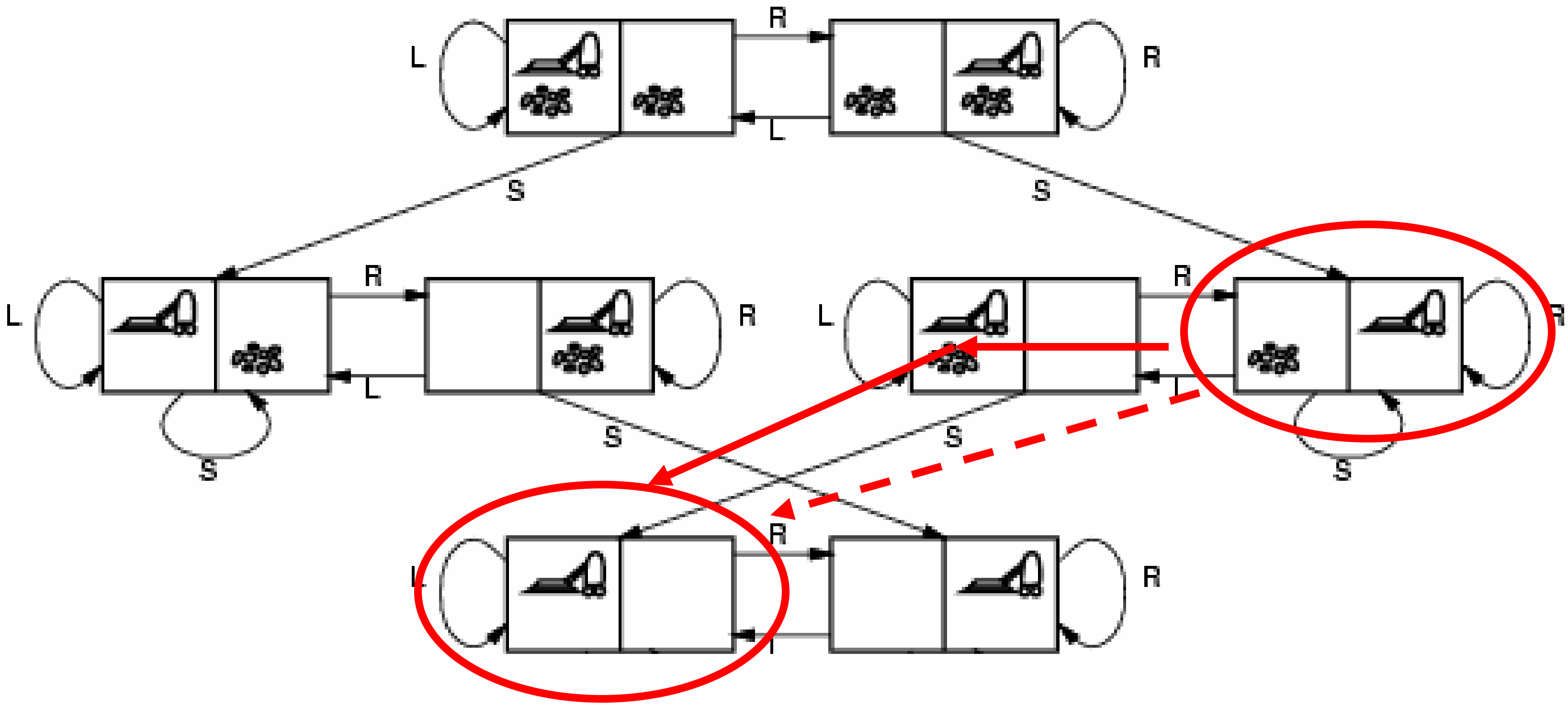


Clean House Task

- Want to clean "house"
  ⇒ be in State#7 or State#8
- Initial world: State#4
- Actions: { Left, Right, Suck }



Vacuum Cleaner Space



Why Search?

- Typical tasks:

  Get to location *r* ; Clean rooms;

  Lay out chip; Solve puzzle, ...

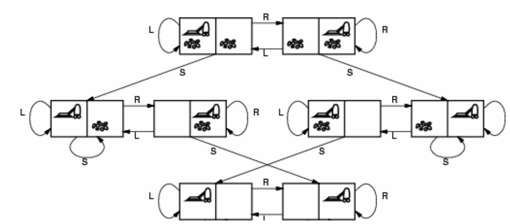
- NOT given algorithm,

  just know: what is a (good) solution

  Goal + preferences

- Search is a *general problem solving technique* for such situations

General Search Task



Given

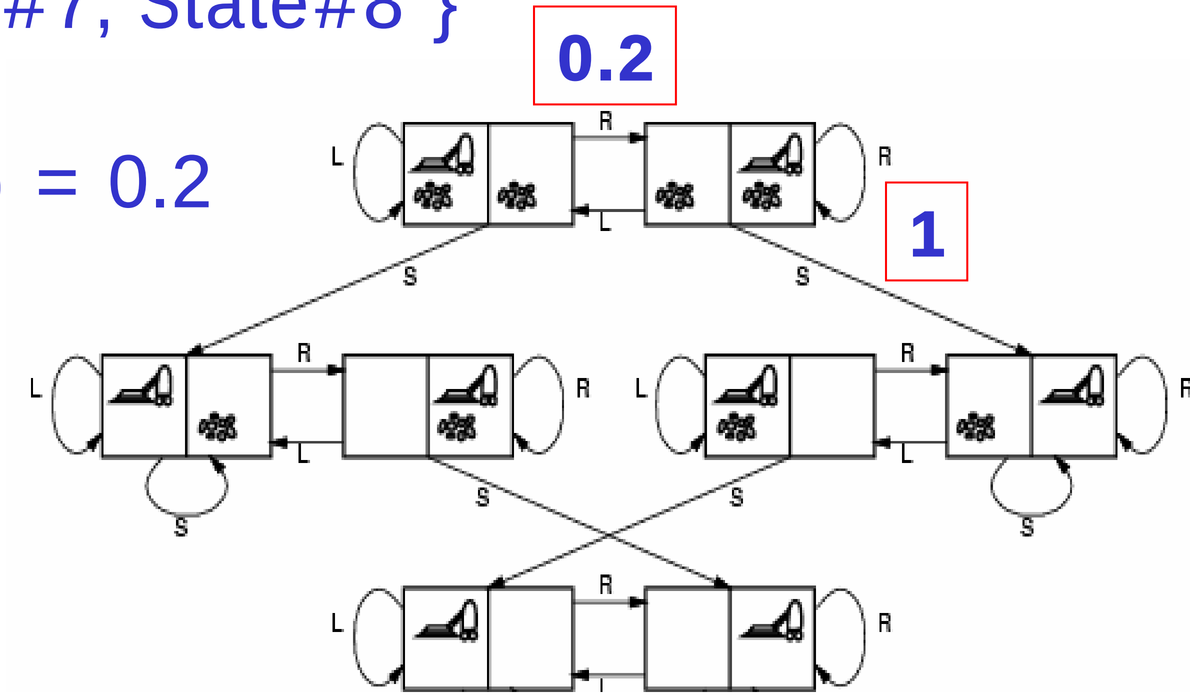
- Initial State
  "State#4" or "Arad"
- Set of actions ("Operators")
  {Left, Right, Suck} or
  Travel-along-Road
- Goal test
  "Is house clean" or "Bucharest"
- Path cost function
  Cost of path
  (aka "sequence of operators")
  …typically sum of operator-costs…
  {0.1 for Left/Right, 1 for suck} or
  "Distance"

Produce

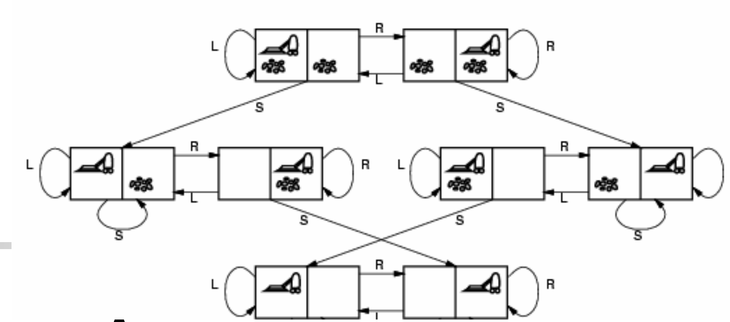
- **Solution ≡ Optimal Path**:
  Sequence of operations
  from
  initial state,
  to
  state satisfying goal test
  ... with minimal path-cost...

Vacuum Cleaner Environment

- State:
  [dirt in roomA and/or roomB;
  vc in roomA xor roomB]
- Operators: { Left, Right, Suck }
- Goal test: { State#7, State#8 }
- Path Cost:
  c(Left) = c(Right) = 0.2
  c(Suck) = 1



Search Graph



- State → Node; Action → Arc

  ⇒ (implicit) Graph $G = \langle N, A \rangle$

  ... called state space

- Path is sequence of nodes

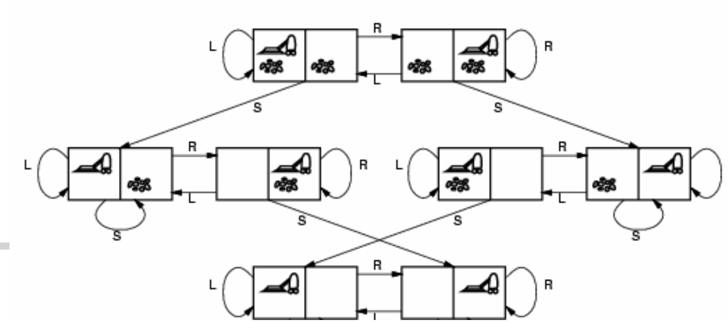
  $\pi = \langle n_0, a_{01}, n_1, a_{12}, \ldots a_{k-1,k}, n_k \rangle$

  s.t. $a_{i,i+1} = \langle n_i, n_{i+1} \rangle \in A$

- Label each path π with $g(\pi) \in \Re^{\geq 0}$

  Often $g(\pi) = \sum_i c(n_i, a_{i,i+1}, n_{i+1})$

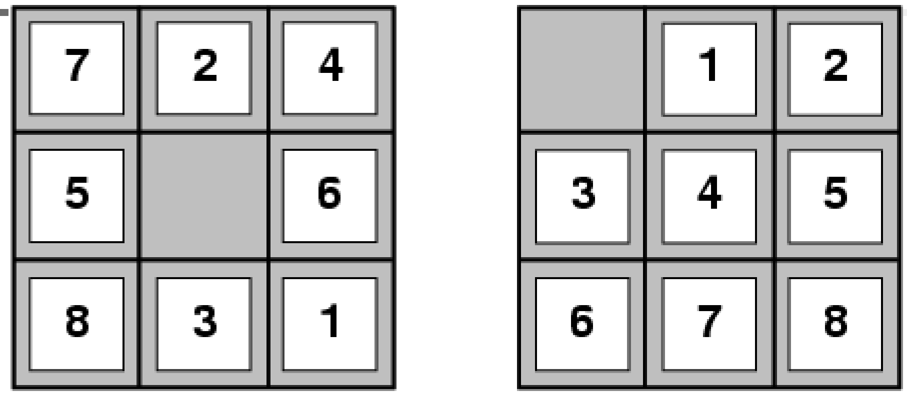
Optimal Solution



- Given state space $G = \langle N,A \rangle$, cost-fn $g(.)$ start node $s \in N$, goal nodes $T \subset N$,
- ***SOLUTION*** π is path from s to goal $t \in T$

- ***OPTIMAL SOLUTION*** π^* is solution w/ min'm cost $g(\pi^*) \leq g(\pi)$

Example: The 8-puzzle



Start State   Goal State

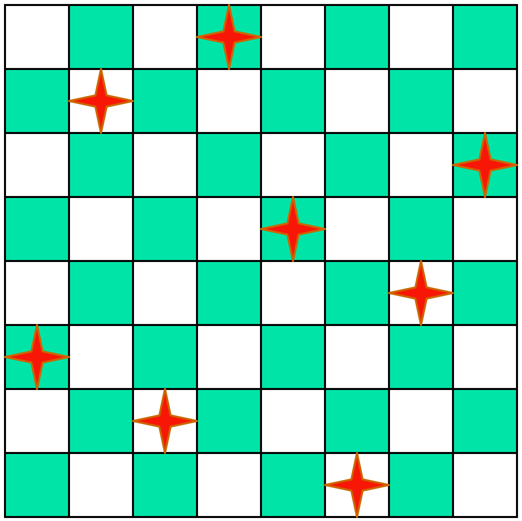
states?	locations of tiles
actions?	move blank: left, right
goal test?	= goal state
path cost?	1 per move

So want SHORTEST soln

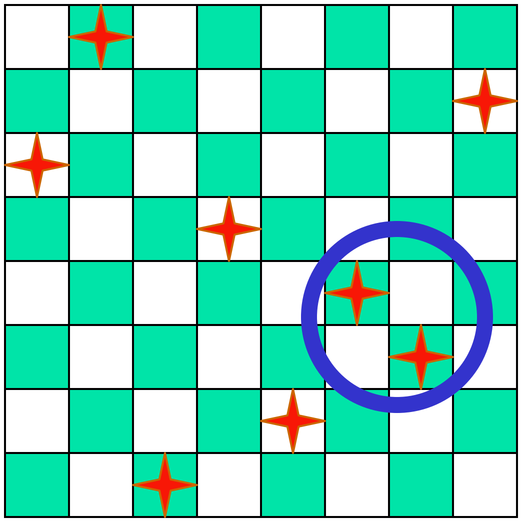
[Note: optimal solution of *n*-Puzzle family is NP-hard]

Example: 8-queens

Place 8 queens in a chessboard so that
no two queens are in the same row, column, or diagonal

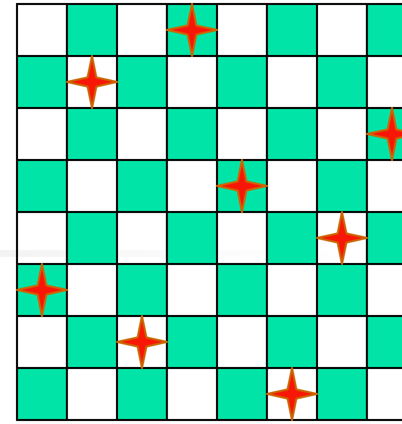


A solution



Not a solution

Example: 8-queens Formulation #1

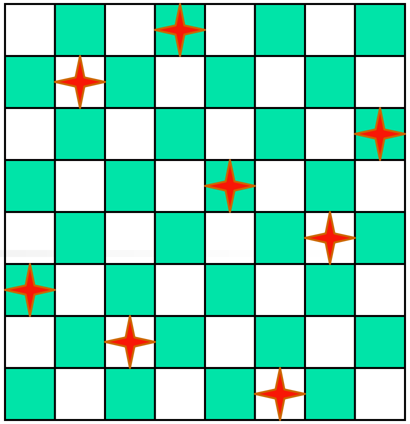


states?	*any* arrangement of 0 to 8 queens on the board
actions?	add a queen to any square (that is not attacked)
goal test?	8 queens on the board, none attacked
path cost?	0

Path irrelevant; just want SOLUTION!

→ $64^8 = 2.81*10^{14}$ states with 8 queens

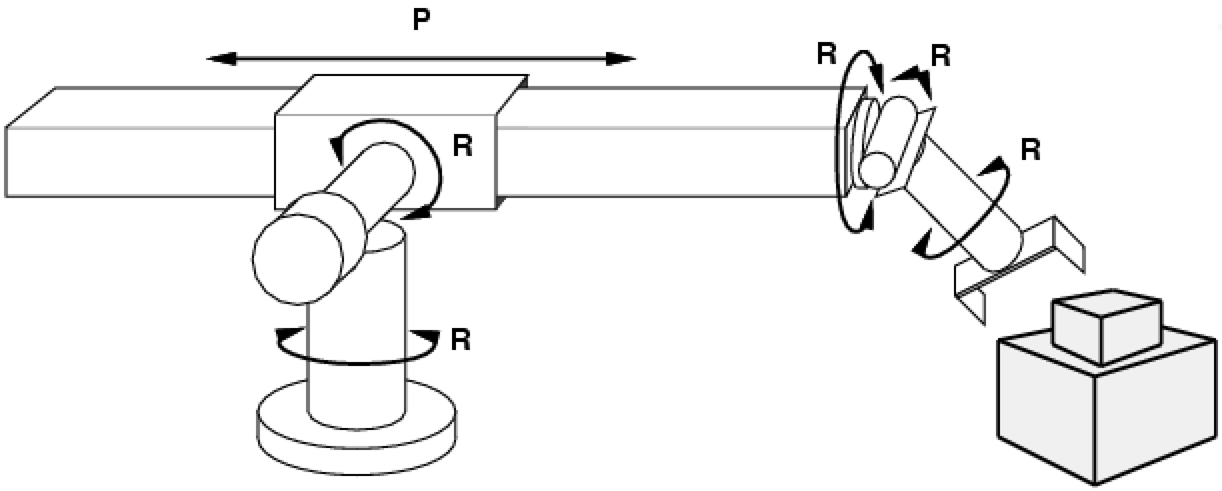
Example: 8-queens Formulation #2



states?	any arrangement of *k* = 0 to 8 queens in the *k* leftmost columns
actions?	add a queen to any square in the *leftmost empty column* (that is not attacked)
goal test?	8 queens on the board, none attacked
path cost?	0

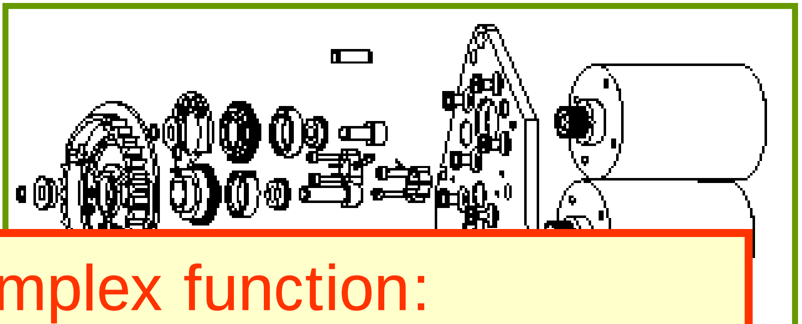
$8^8 \approx$ 16M states
→ $8! \approx$ 40K states
→ 2,067 states

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
path cost?	time to execute

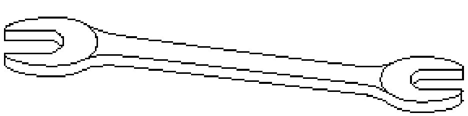
Example: Assembly Planning



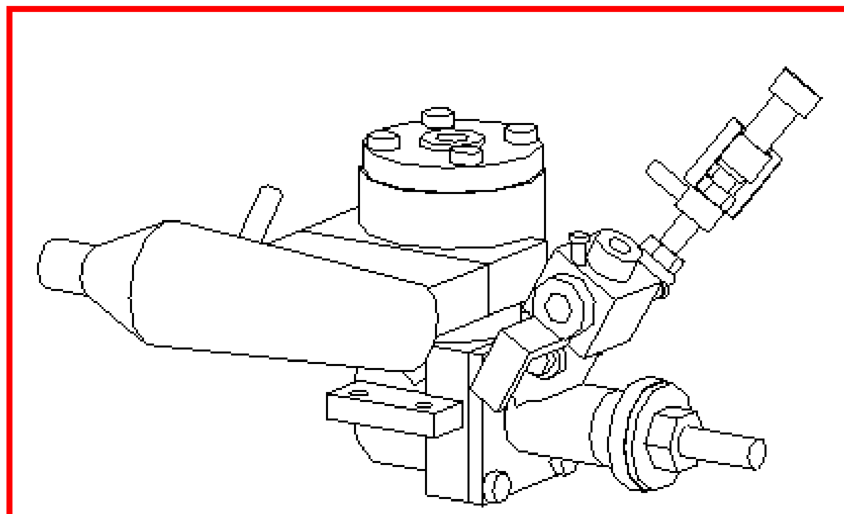
Initial state

Complex function:
it must find if a collision-free
merging motion exists

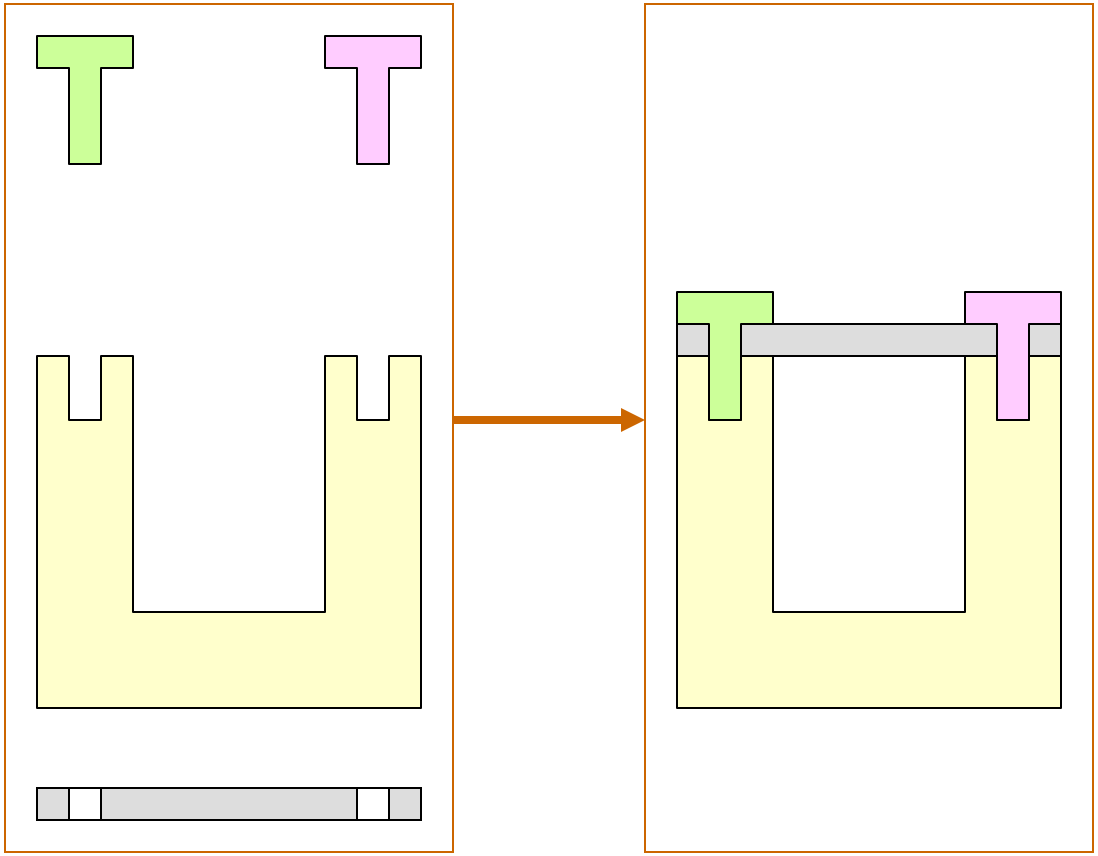
Operator:
• Merge two subassemblies



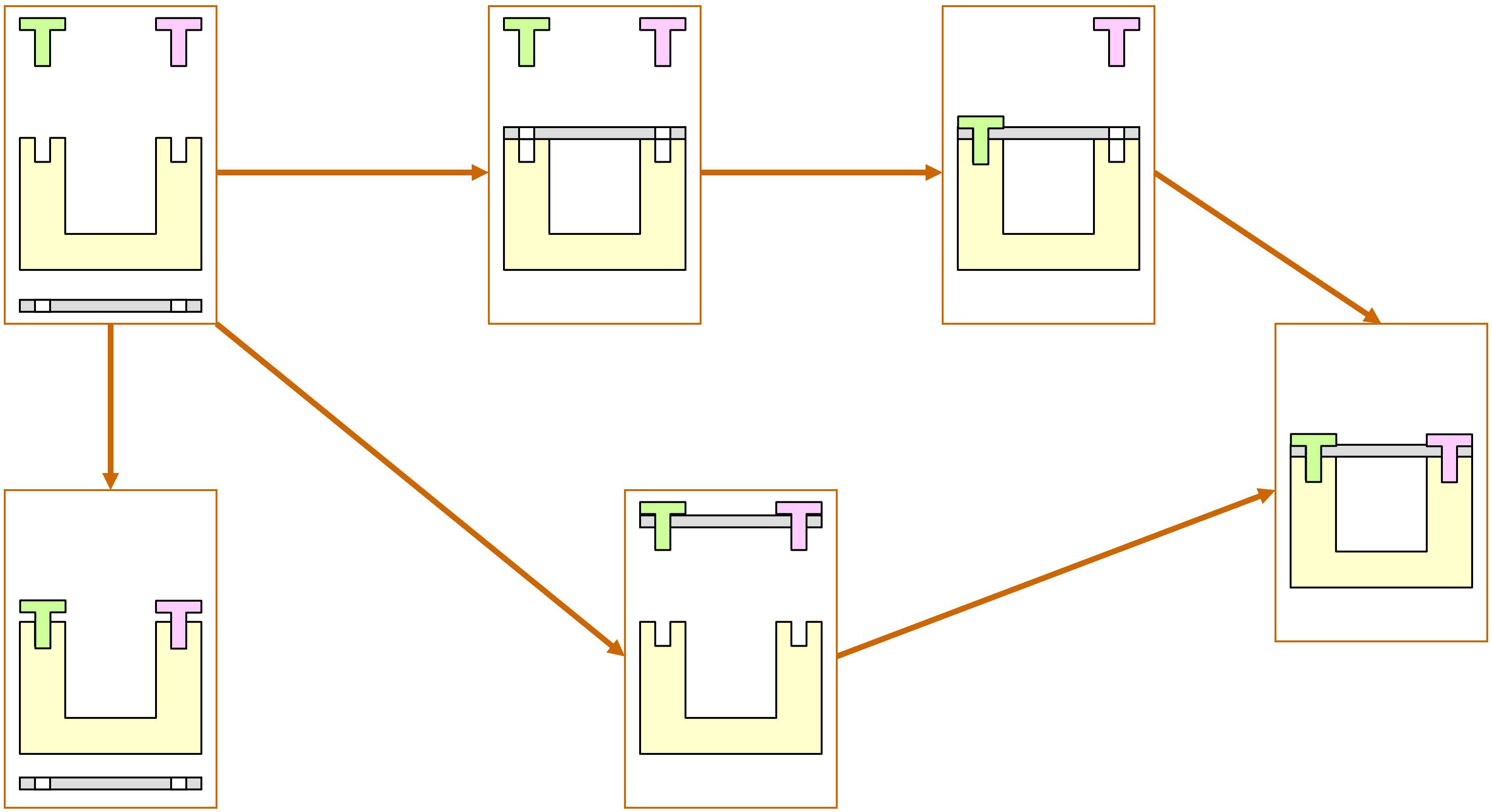
Goal state



Example: Assembly Planning



Example: Assembly Planning

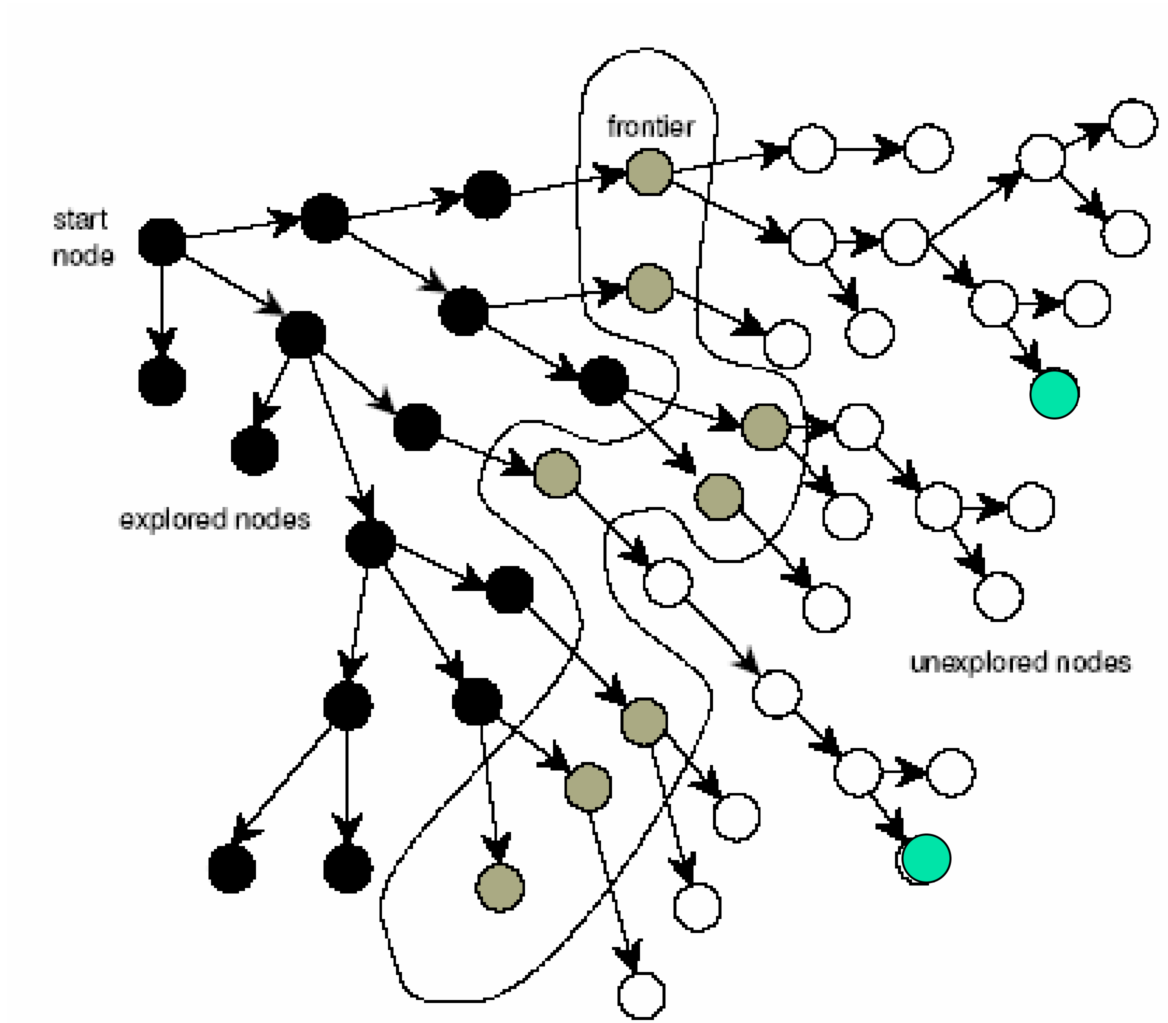


Solving a Search Problem

- Solve problem by searching over ***state space***
- ... build a search tree over search space
  - Root = the initial state
  - Successor = from state s to s', based on some operator
  - Leaf = state with no successors in (current) tree
    (none exists; or node not yet expanded)
  - Search strategy = algorithm for deciding which leaf node to expand next
- Search proceeds by expanding *frontier* into unexplored nodes,

until encountering goal node

Problem Solving by Graph Searching



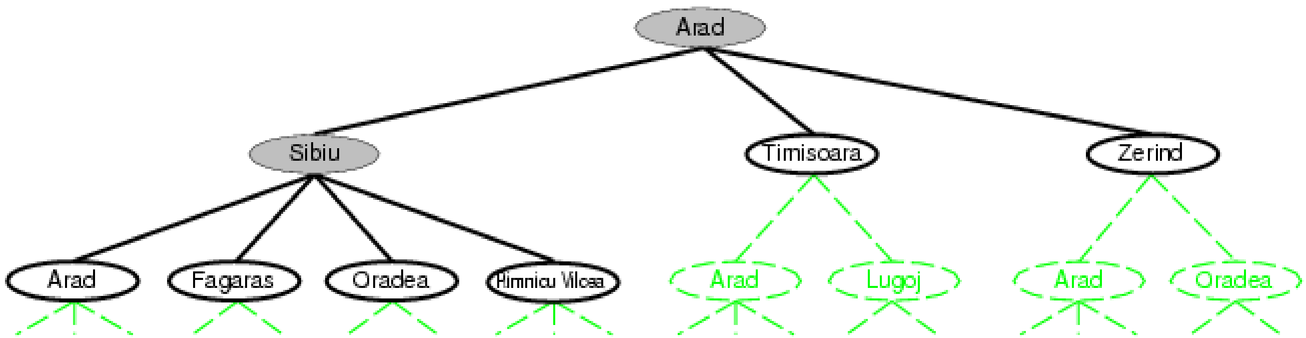
Generic Search Algorithm

```
Search_insert( start, operations, isGoal ): path
  L = make-queue( start )
  loop
      n := pop( L )
      if [ isGoal( n )]
        return( n )
      S := successors( n, operators )
      L := insert( S, L )
  until L is empty
  return( failure )
```

insert could be queue, stack, . . .

defines strategy!

Tree search example



Environment...

This type of search works best when environment is...

- **Observable**: Can just “see" the state
- **Deterministic**: Action have well-defined known effects
- **Static**: Environment does NOT change while thinking
- **Discrete**: Only finite number of actions, . . .

Search Problem Variants

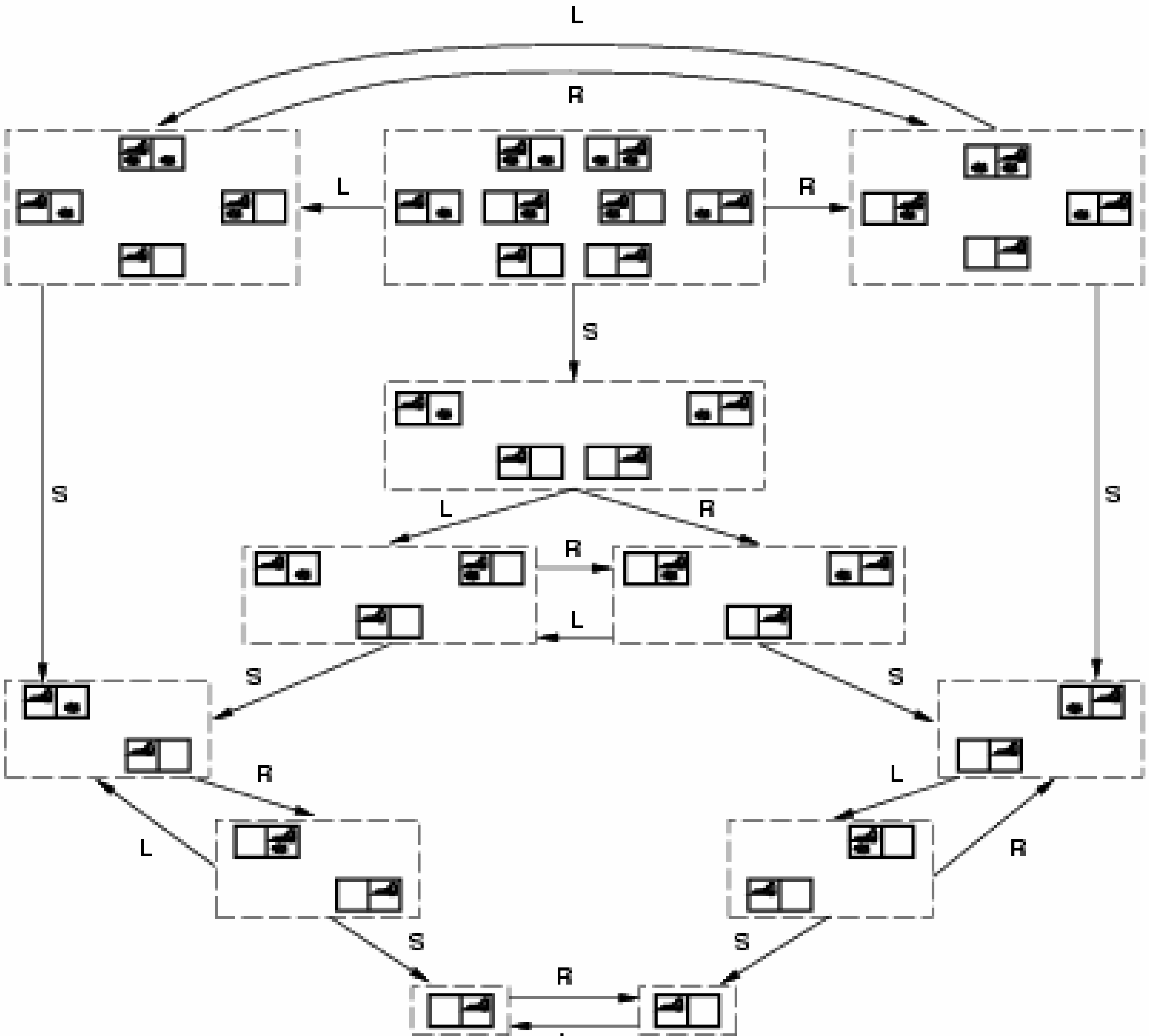
- One or several initial states
- One or several goal states
- The solution ≡ *path* vs *goal node*
  - 8-puzzle problem: the *path* to a goal node
  - 8-queen problem: a *goal node*
- Any, or the best, or all solutions

Problem Types

- Deterministic, fully observable → single-state problem
  - Agent knows exactly which state it will be in; solution is a sequence
- Non-observable → sensorless problem (conformant problem)
  - Agent may have no idea where it is; solution is a sequence
  - [Left, Suck, Right, Suck] works!
  - Later: represent set of states IMPLICITLY: logic, probabilities
- Nondeterministic and/or partially observable → contingency problem
  - Spse: SUCK drops dirt, iff no dirt there
  - Agent must sense state WHILE executing, to decide how to act (percepts provide new information about current state)
  - Soln = tree, policy ... interleaving: search, execution
- Unknown state space → exploration problem

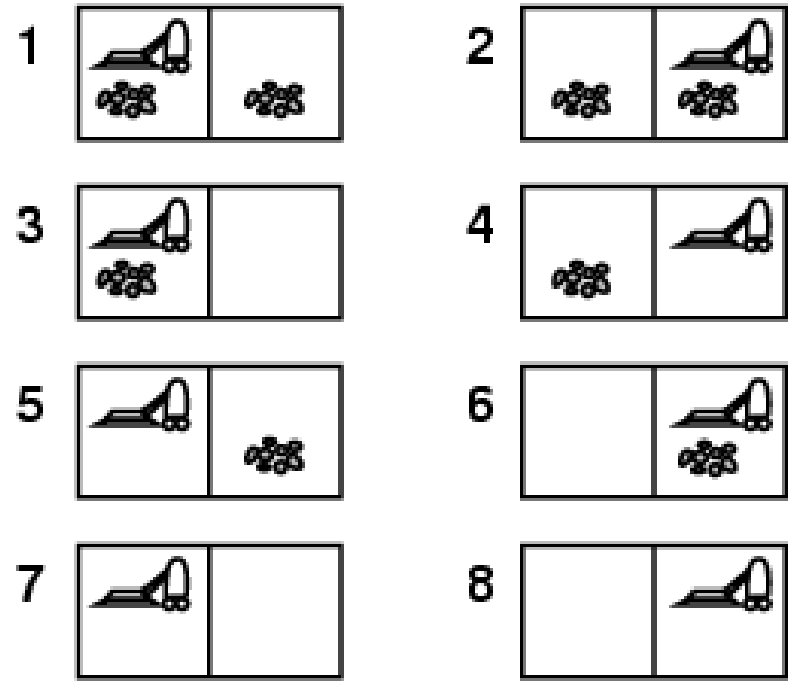
- What if no sensors?... and don't know initial state?

⇒ Each node is Set Of States



Non-Observable (vacuum world)

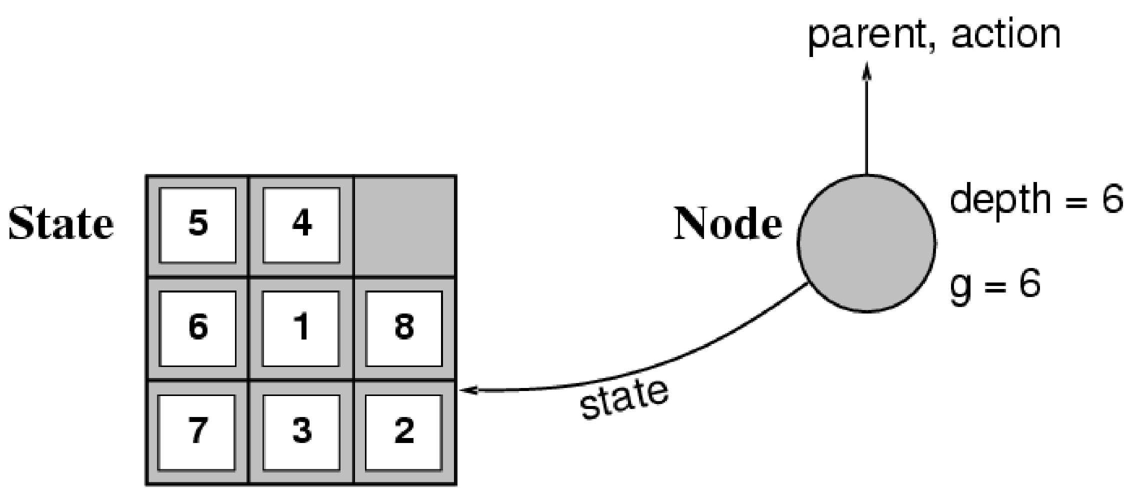
- Sensorless, start in {*1,2,3,4,5,6,7,8* }
  - *Right* goes to {*2,4,6,8*}
  - Solution? *[Right,Suck,Left,Suck]*



- Contingency
  - Nondeterministic: *Suck* may make dirty a clean carpet
  - Partially observable: location, dirt at current location.
  - Percept: *[L, Clean],* i.e., start in #5 or #7
  - Solution? *[Right,* ***if*** *dirt* ***then*** *Suck]*

(Real World) State vs (Search) Node

- A state corresponds to real world
  ... represents a physical conguration
- A node is a data structure
  ... part of a search tree
- Node *x* has *parent*, *children*, *depth*, *path cost g(x)*
  A state does not!
- Many nodes can correspond to same state



Applications

- Route finding: airline travel, telephone/computer networks
- Pipe routing, VLSI routing
- Pharmaceutical drug design
- Robot motion planning
- Video games

Summary

- Problem-solving agent
- State space, successor function, search
- Examples:
  - Travel Task
  - House cleaning
  - 8-queens
  - Assembly planning
- Assumptions of basic search