ESCOLA SUPERIOR DE TECNOLOGIA IPCA

Search in Complex Environments

Fundamentals of Artificial Intelligence

MSc in Applied Artificial Intelligence, 2023-24

Contents

- Topics included
  - Local search
  - Evolutionary algorithms
  - Partially Observable Environments

- These slides were based essentially on the following bibliography:
  - Norvig, P, Russell, S. (2021). Artificial Intelligence: A Modern Approach, 4th Edition. Pearson, ISBN-13: 978-1292401133

Local Search

Search in complex environments

- The general approaches for problem solving, discussed in previous sections, address problems in fully observable, deterministic, static, known environments where the solution is a sequence of actions

- This section will introduce ...
  - the problem of finding a good state without worrying about the path to get there, covering both discrete and **continuous states**
  - how to use conditional plan and carry out different actions depending on what the agent observes in a **nondeterministic** world
  - how to keep track of the possible states the agent might be in **partial observable** world

Local search

- Local search algorithms operate by searching from a start state to neighboring states, without keeping track of the paths, nor the set of states that have been reached.
- These algorithms are not systematic — they might never explore a portion of the search space where a solution actually resides.

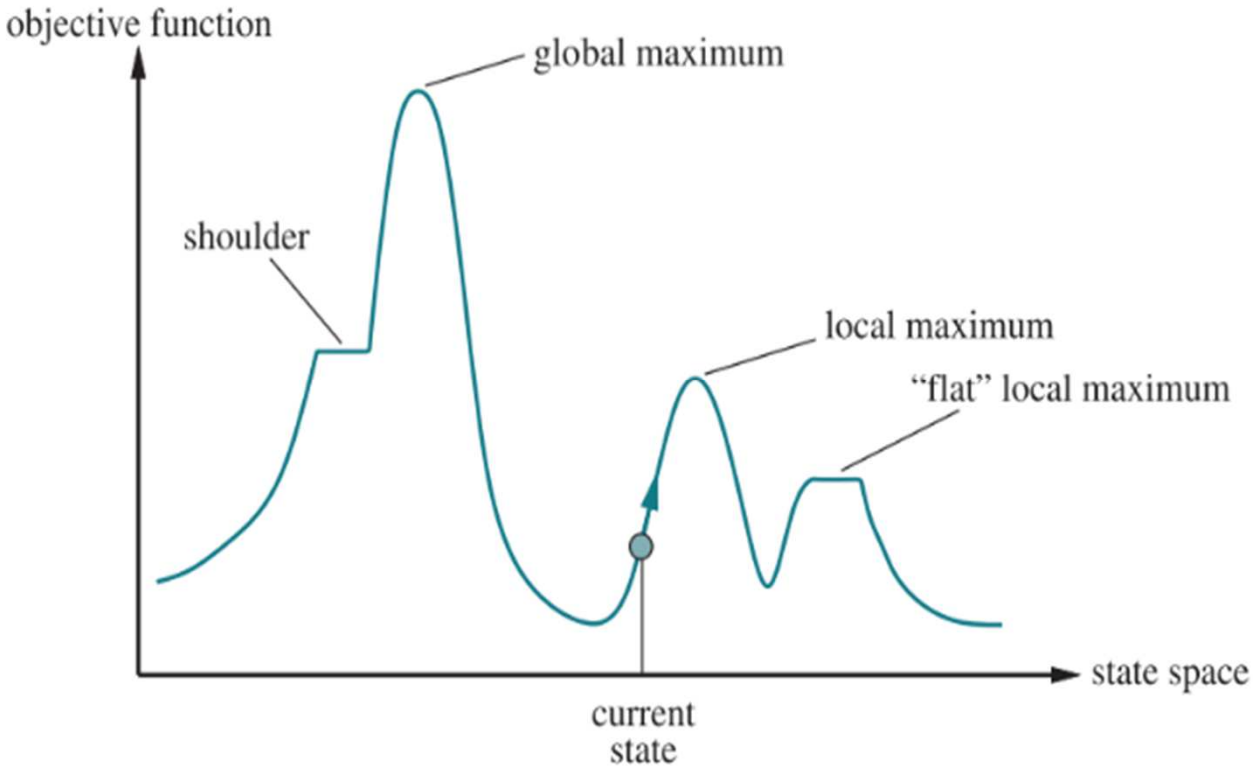
- However, local search algorithms have two key advantages
  - (1) they **use very little memory**; and
  - (2) they can **often find reasonable solutions** in large or infinite state spaces for which systematic algorithms are unsuitable.

- Local search algorithms can also solve **optimization problems**, in which the aim is to find the best state according to an objective function.

Optimization Problems

Consider the states of a problem laid out in a state-space landscape, as shown in Figure

- Each point (state) in the landscape has an “elevation,” defined by the value of the objective function.
- If elevation corresponds to an objective function, then the aim is to find the highest peak — a **global maximum** — and we call the process **hill climbing**.
- If elevation corresponds to cost, then the aim is to find the lowest valley — a **global minimum** —and we call it gradient descent.



Iterative Improvement

- In many optimization problems, the path to the goal is irrelevant!
  - The **goal is itself the solution**!
  - State space = set of complete configurations!
- Local search algorithms are used for iterative improvement
  - Start as any (initial) solution of the problem and make changes to improve its quality

- **Hill-Climbing**
  - Pick a state randomly from the state space
  - Consider all neighbors of that state — we need to define a range
  - Pick the best neighbor
  - Repeat the process until there are no better neighbors
  - The current state is the solution

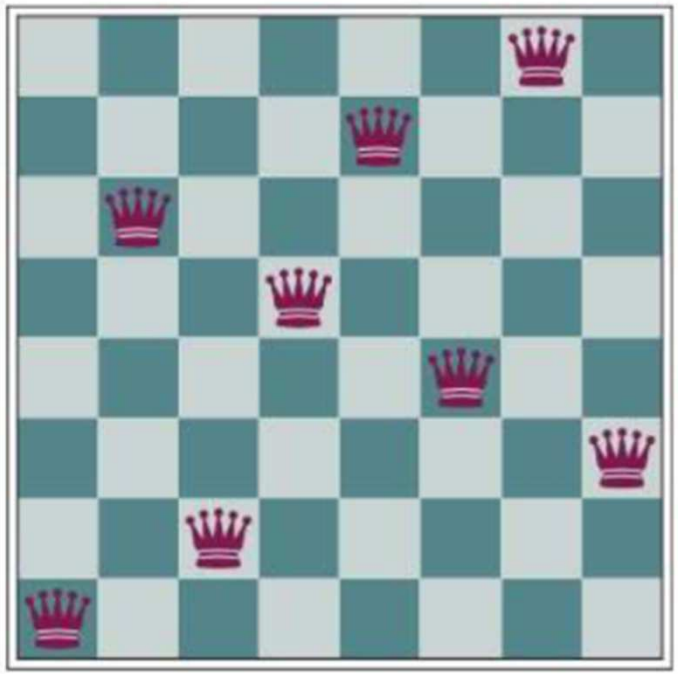
Hill-Climbing for the 8-queens problem

- Complete-state formulation — every state has all the components of a solution: 8 queens on the board, one per column
- The **initial state is chosen at random**, and
  - the successors of a state are all possible states generated by moving a single queen to another square in the same column
  - For each state there are 8×7 = 56 successors.
- The heuristic cost function *h* is **the number of pairs of queens that are attacking each other** — for solution states *s*, *h*(s) = 0
  - It counts as an attack if two pieces are in the same line, even if there is an intervening piece between them
- board shows an 8-queens state with heuristic cost estimate ***h*=17**.
  - The value of *h* for each possible successor found by moving a queen within its column. There are 8 moves that are tied for best, with h=12.
  - The hill-climbing algorithm will pick one of these.

Hill-Climbing limitations

- **Local maxima**
  - A local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum.
  - Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go.
  - The state in right board is a local maximum (i.e., a local minimum for the cost h). Every move of a single queen makes the situation worse.

- **Plateaus**
  - A plateau is a flat area of the state-space landscape. It can be a flat local Plateau maximum (no uphill exit exists) or a shoulder
  - A hill-climbing search can get lost wandering on the plateau.

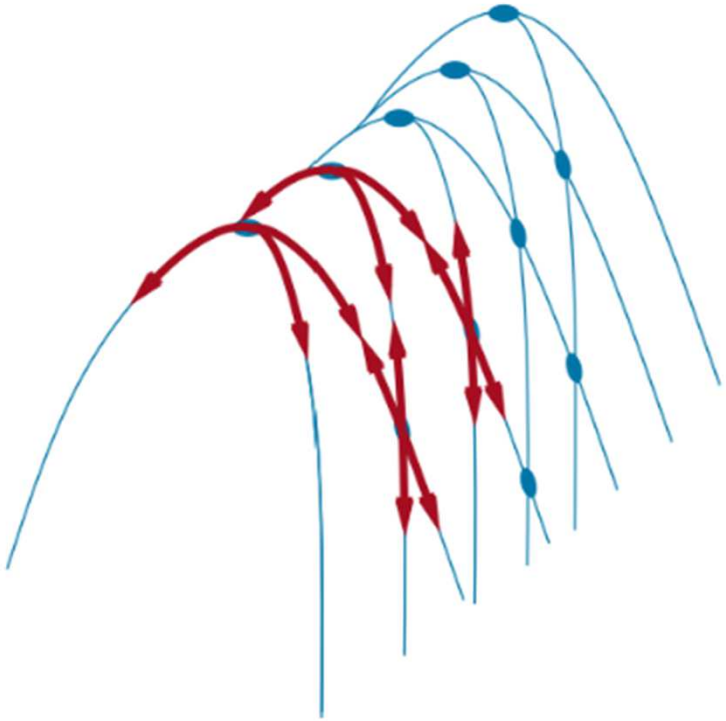


Hill-Climbing limitations (2)

- **Ridges**
  - A ridge is shown in the right Figure
  - Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.

- In each limitation (3), the algorithm reaches a point at which no progress is being made.
  - Starting from a randomly generated 8-queens state, steepest-ascent hill climbing **gets stuck 86%** of the time, **solving only 14% of problem instances**.
  - On the other hand, it works quickly, taking **just 4 steps on average when it succeeds** and **3 when it gets stuck** — not bad for a state space with 88 ≈ 17 million states.

- There are many variants of hill-climbing to keep going when we reach a plateau — to allow a **sideways move** in the hope that the plateau is really a shoulder



Local beam search

- The local beam search algorithm keeps track of k states rather than just one.
  - It begins with **k randomly generated states**
  - At each step, all the successors of all k states are generated.
    If any one is a goal, the algorithm halts.
  - Otherwise, it selects the k best successors from the complete list and repeats.
- A local beam search with k states is different from than running k random restarts in parallel.
  - In a random-restart search, each search process runs independently of the others.
  - In a local beam search, useful information is passed among the parallel search threads.
  - The **algorithm quickly abandons unfruitful searches** and moves its resources to where the most progress is being made.
- Local beam can suffer from lack of diversity among the k states — they can become clustered in a small region of the state space, making the search little more than a k-times-slower version of hill climbing
- **Stochastic beam search**, Instead of choosing the top k successors, chooses successors with probability proportional to the successor's value, thus increasing diversity.

More Iterative Improvement Algorithms

- **Simulated Annealing**
  - Similar to Hill-Climbing Search but allows to explore worse neighbors
- **Tabu Search**
  - Explores the neighboring states but eliminates the worst ones (tabu neighbors)
  - maintains a tabu list of k previously visited states that cannot be revisited;
  - improves efficiency when searching graphs,
  - can allow the algorithm to escape from some local minima.
- **Ant Colony Optimization**
  - Several starting states (ant colony)
  - Determines the probability of a path being better from the number of "ants" passing through it
- **Particle Swarm Optimization**
  - Several starting states (swarm)
  - The neighborhood is explored, and the best solution and the best state are saved
  - The states are moved towards the best solution found so far
  - The speed of movement depends on the distances to the best solution and the best state and the position of the state
- **Genetic/Evolutionary Algorithms**
  - Define a state as a chromosome
  - Generate solutions (chromosomes) from a population of initial states

Evolutionary algorithms

Retrieved in nov.2022 from https://evo-ml.com/

Evolution and Search

- The theory of evolution was developed by Charles (1859) and independently by Alfred Russel Wallace (1858). The central idea is simple:

  Variations occur in reproduction and will be preserved in successive generations approximately in proportion to their effect on reproductive fitness.

- There was no knowledge of how the traits of organisms can be inherited/ modified.
  - Only in 1953, Watson and Crick identified the structure of the DNA molecule and its alphabet, AGTC (adenine, guanine, thymine, cytosine).
  - In the standard model, **variation occurs both by point mutations** in the letter sequence and by "**crossover**", in which the DNA of an offspring is generated by combining long sections of DNA from each parent
- The genes encode the mechanisms whereby the genome is reproduced and translated into an organism.
  - In **genetic algorithms**, those mechanisms are a separate program that is not represented within the strings being manipulated.

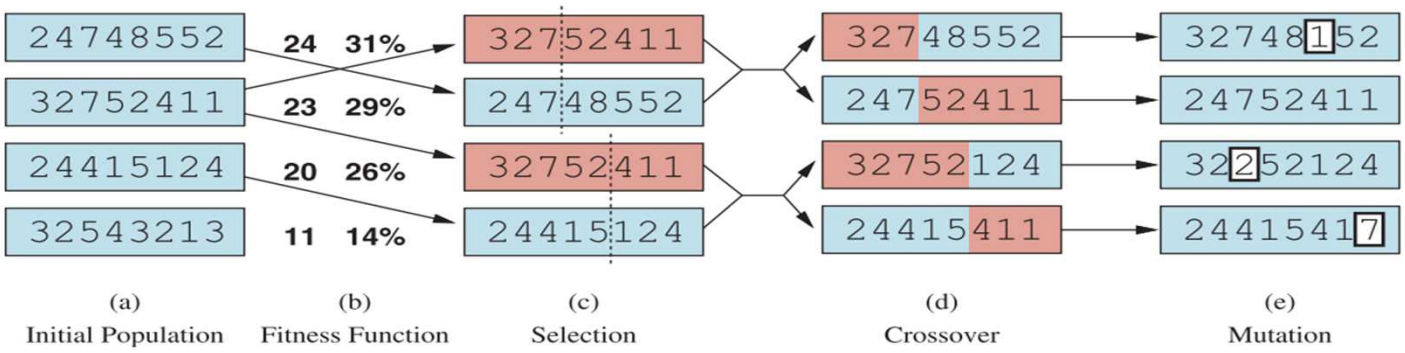
Evolutionary algorithms

- Seen as **variants of stochastic beam** search motivated by the metaphor of *natural selection*
- **Recombination** is the process in which the fittest (highest value) individuals of a population (states) produce offspring (successor states) that populate the next generation
- There are endless **forms of recombination** that vary on ...
  - The size of the population
  - The representation of each individual. In genetic algorithms, each individual is a string over a finite alphabet (often a Boolean string)
  - The mixing number, ρ, which is the number of parents that come together to form offspring. The most common case is ρ = 2: two parents combine their "genes" (parts of their representation) to form offspring. It is possible to have ρ=1 or ρ > 2
  - The selection process for the parents of the next generation: (a) select from all individuals according to their fitness score; (2) randomly select n individuals (n > ρ), and then select the ρ most fit ones as parents.

Evolutionary algorithms (2)

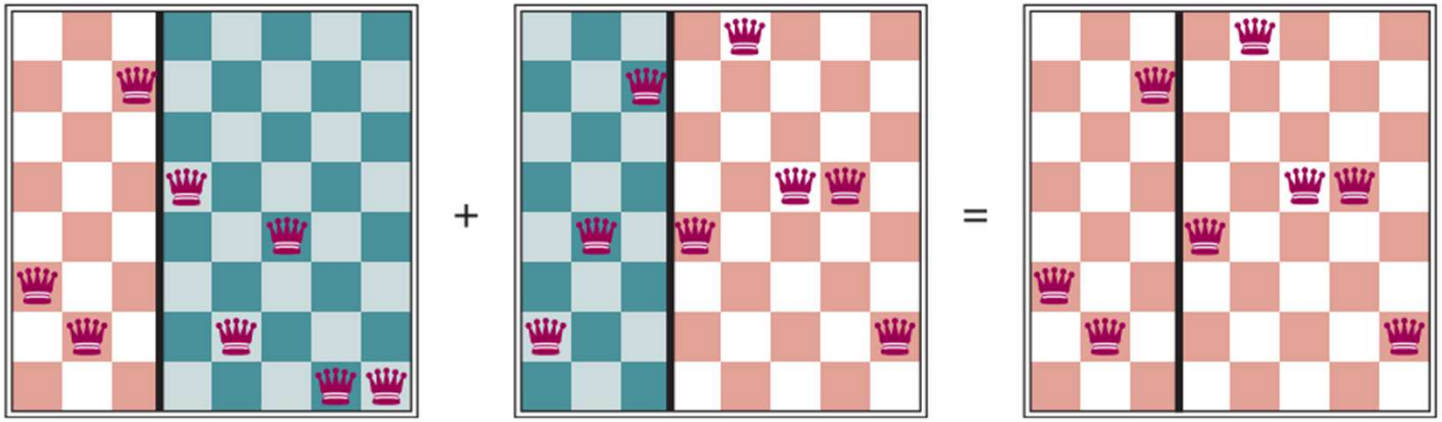
- More ways about how evolutionary algorithms use **recombination**...
  - The recombination procedure. One common approach (assuming ρ= 2), is to randomly select a crossover point to split each of the parent strings, and recombine the parts to form two children
  - The mutation rate, which determines how often offspring have random mutations to their representation. Once an offspring has been generated, every bit in its composition is flipped with probability equal to the mutation rate.
  - The makeup of the next generation. This can be just the newly formed offspring, or it can include a few top-scoring parents from the previous generation. The practice of culling, in which all individuals below a given threshold are discarded, can lead to a speedup

The 8-queens problem using evolutionary algorithms



- Above figure shows a population of four 8-digit strings, each representing a state of the 8queens puzzle: the *n*-th digit represents the row number of the queen in column *n*.
- In (b), each state is rated by the fitness function.
  - Higher fitness values are better, so for the 8-queens problem we use the number of nonattacking pairs of queens, which has a value of 8×7/2 = 28 for a solution
  - The values of the four states in (b) are 24, 23, 20, and 11. The fitness scores are then normalized to probabilities.
- In (c), two pairs of parents are selected, in accordance with the probabilities in (b)
  - Notice that one individual is selected twice and one not at all. For each selected pair, a crossover point (dotted line) is chosen randomly.
- In (d), we cross over the parent strings at the crossover points, yielding new offspring.
  - For example, the first child of the first pair gets the first three digits (327) from the first parent and the remaining digits (48552) from the second parent.

The 8-queens problem using evolutionary algorithms (2)

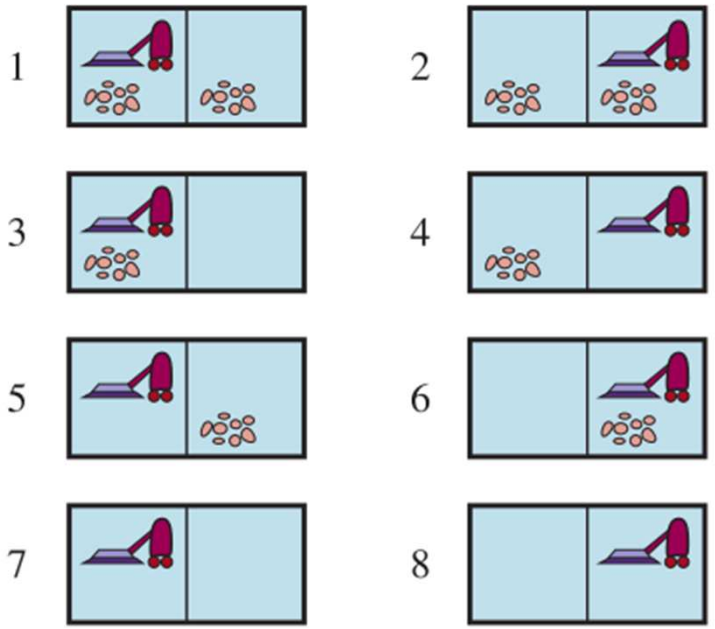


- The 8-queens states corresponding to the first two parents in Figures (c) and (d), shown in previous figure (positions correspond to the row: 1 on bottom).
  - The green/blue columns are lost in the crossover and the red columns are retained.
- Genetic algorithms are similar to stochastic beam search with the crossover operation
- This is advantageous if there are blocks that perform useful functions.
  - E.g., putting the first three queens in positions 2, 4, and 6, where they do not attack each other, may be a useful block that can be combined with other useful blocks
  - It can be shown mathematically that, if the blocks do not serve a purpose, then crossover conveys no advantage.
- Evolutionary algorithms are suitable for complex problems
  - Usually need some specific adaptations/improvements
  - But do not use when other method, like hill-climbing, works well

Partially Observable Environments

Search with Nondeterministic Actions

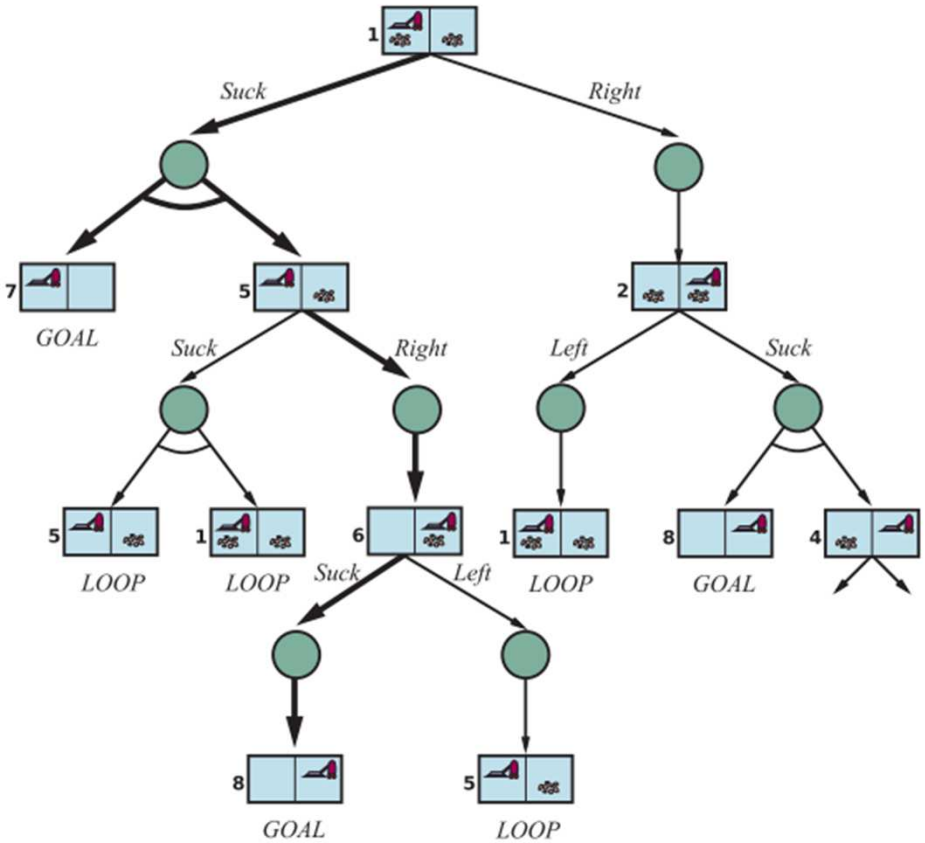
- Previous search algorithms assumed the environment is fully observable, deterministic, and known.
- However ...
  - when it is **partially observable**, the agent doesn't know for sure what state it is in;
  - when it is **nondeterministic**, the agent doesn't know what state it transitions to.
- The agent should think "I'm either in state s1 or s3, and if I do action A I'll end up in state s2, s4 or s5."
- The **belief state** is a set of physical states in which the agent believes it is possible to be
- The solution is no longer a sequence, but a **conditional plan**,
  - It specifies what to do depending on what percepts agent receives while executing the plan.



The eight possible states of the vacuum world; states 7 and 8 are goal states.

The erratic vacuum world

- The vacuum world example (previous slide figure)
  - there are three actions—Right, Left, and Suck
  - the goal is to clean up all the dirt (states 7 and 8).
- Now suppose we have a nondeterministic, **erratic vacuum world**, where the **Suck action** works as follows:
  - When applied to a dirty square the action cleans the square and sometimes cleans up dirt in an adjacent square, too.
  - When applied to a clean square the action sometimes deposits dirt on the floor
- Instead of returning a single outcome state, the RESULTS function returns a set of possible outcome states.



The erratic vacuum world (2)

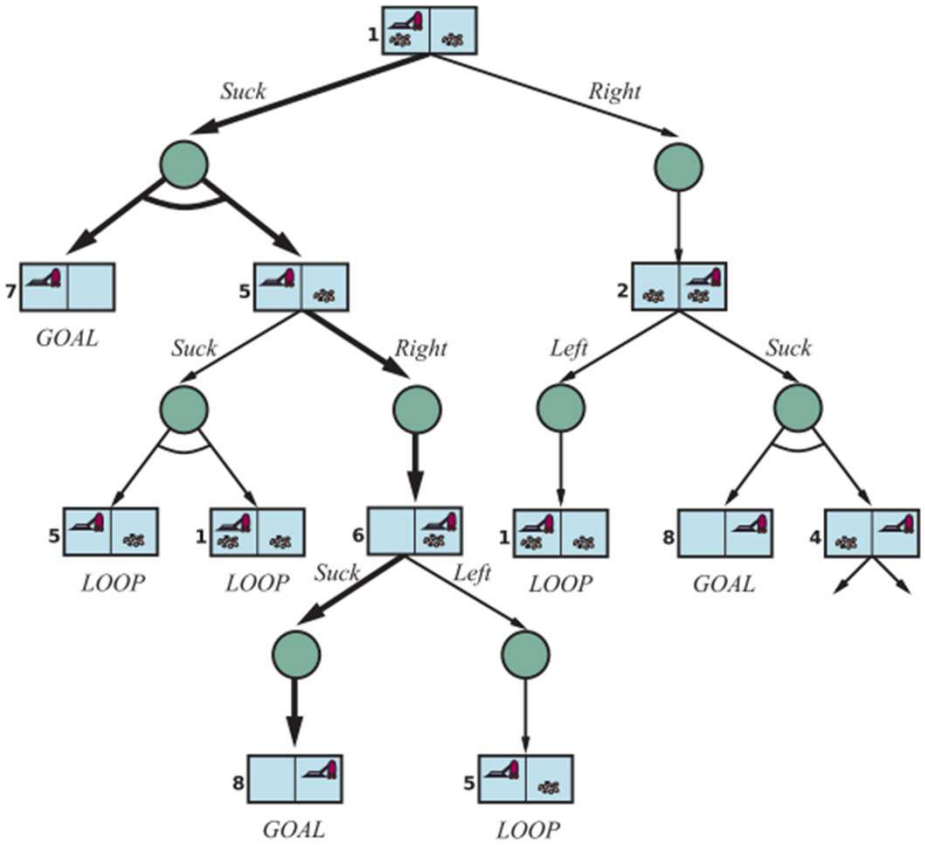
- E.g., the Suck action in state 1 cleans up either just the current location, or both locations:

  RESULTS(1, Suck) = {5,7}

- If we start in state 1, no single sequence of actions solves the problem, but the following conditional plan does:

  [Suck, if State=5 then [Right, Suck] else [ ]]

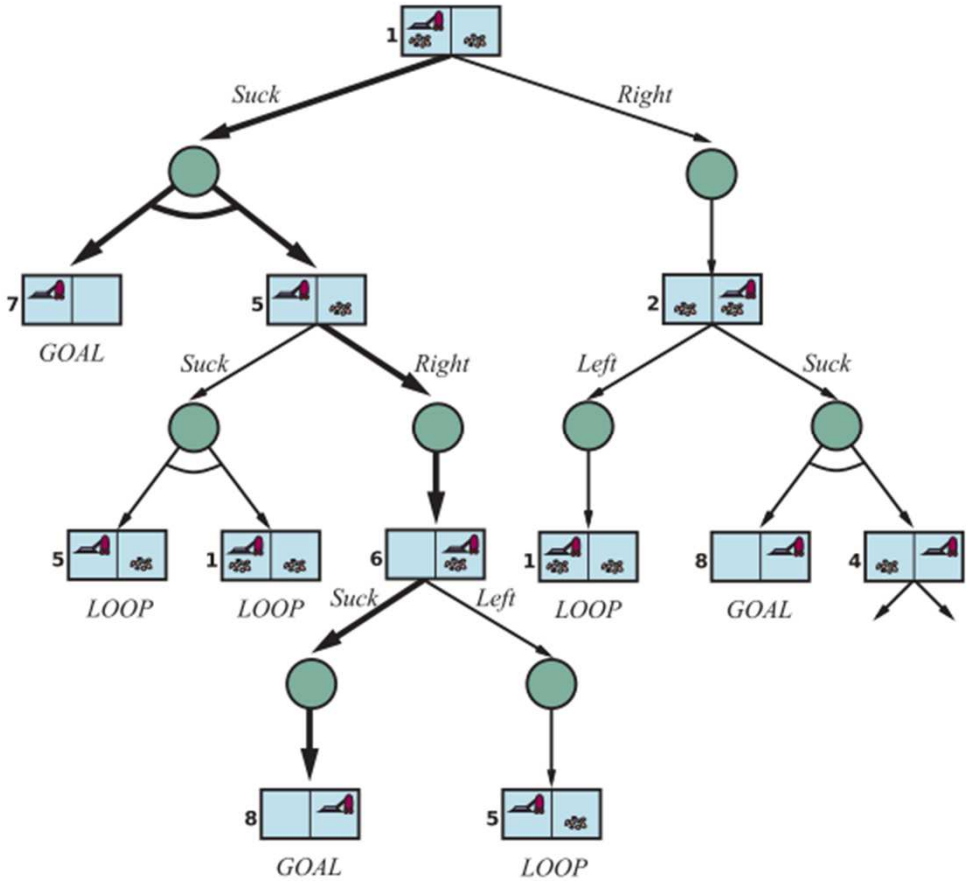
- A conditional plan means that solutions are trees rather than sequences.
- In the if statement, the agent will be able to observe at runtime what the current state is
  - Alternatively, the agent could test the percept rather than the state.
- Many problems in the real world are contingency problems, because exact prediction of the future is impossible.



AND–OR search trees

- In a deterministic environment, the only branching is introduced by the agent's own choices in each state: **I can do this action or that action**.
  - State nodes are **OR nodes** where some action must be chosen: Left or Right or Suck.
- In a nondeterministic environment, branching also results of the environment's choice of outcome for each action
  - There are **AND nodes** (**circles**)

- E.g., the Suck action in state 1 results in the **And node belief state** {5,7}, so the agent would need to find a plan for state 5 and for state 7.
  - At the AND nodes, every Suck action outcome must be handled, as indicated by the arc linking the outgoing branches.



The first two levels of the search tree. The solution is shown in bold lines. it corresponds to the plan given in *[Suck, if State=5 then [Right, Suck] else [ ]]*

AND–OR search trees (2)

- A **solution for an AND–OR search problem is a subtree** of the complete search tree that
  - (1) has a goal node at every leaf
  - (2) specifies one action at each of its OR nodes, and
  - (3) includes every outcome branch at each of its AND nodes.
- One key aspect of the algorithm is the way in which it deals with cycles
  - e.g., if an action sometimes has no effect or if an unintended effect can be corrected
- If the current state is identical to a state on the path from the root, then it returns with failure
  - This doesn't mean that there is no solution from the current state;
  - it simply means that if there is a noncyclic solution, it must be reachable from the earlier incarnation of the current state, so the new incarnation can be discarded.
- It ensures that the algorithm finishes in every finite state space: a goal, a dead end, or a repeated state.
  - The algorithm does not check whether the current state is a repetition of a state on some other
- AND–OR graphs can be explored either breadth-first or best-first.
  - The concept of a heuristic function must be modified to estimate the cost of the solution
  - Next slide presents an analog of the A∗ algorithm for finding optimal solutions.

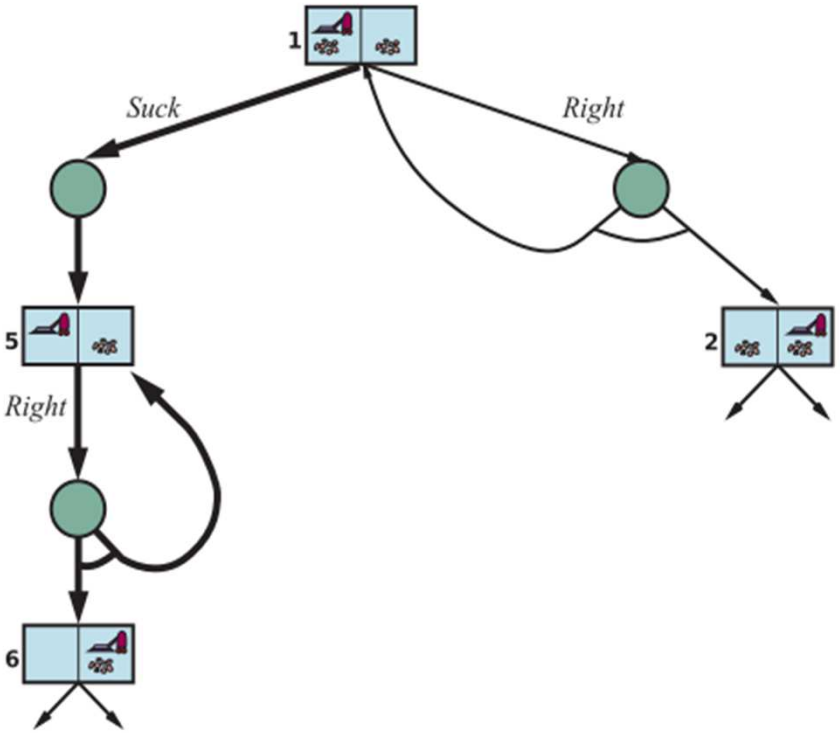
An algorithm for searching AND–OR graphs

An algorithm for searching AND–OR graphs generated by nondeterministic environments. A solution is a conditional plan that considers every nondeterministic outcome and makes a plan for each one.

```
function AND-OR-SEARCH (problem) returns a conditional plan, or failure
        return OR-SEARCH(problem, problem.INITIAL, [ ])

function OR-SEARCH (problem, state, path) returns a conditional plan, or failure
        if problem.IS-GOAL(state) then return the empty plan
        if IS-CYCLE(state, path) then return failure
        for each action in problem.ACTIONS(state) do
                plan ← AND-SEARCH(problem, RESULTS(state, action), [state] + [path])
                if plan ≠ failure then return [action] + [plan]
        return failure

function AND-SEARCH(problem, states, path) returns a conditional plan, or failure
        for each s_i in states do
                plan_i ← OR-SEARCH(problem, s_i, path)
                if plan_i = failure then return failure
return [if s_1 then plan_1 else if s_2 then plan_2 else . . . if s_n−1 then plan_n−1 else plan_n]
```



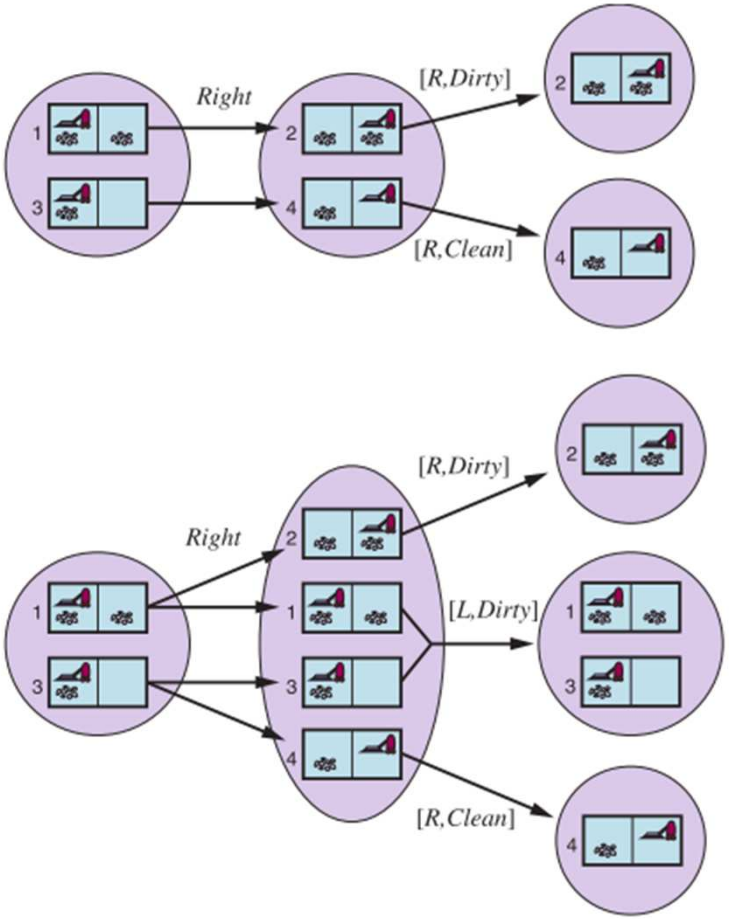
Cycle try, when action fails

- Consider a **slippery vacuum world**, where the **movement actions sometimes fail**, leaving the agent in the same location.
  - E.g., moving Right in state 1 leads to the belief state {1,2}.
- There are no longer any acyclic solutions from state 1, and AND-OR-SEARCH would return with failure.
- Cyclic solution is to ***keep trying Right*** until it works using a
  - while construct: [Suck, while State=5 do Right, Suck] or
  - adding a label to some portion of the plan and referring to the label later: [Suck,L1 : Right, if State=5 then L1 else Suck]
- A cyclic plan is a solution if every leaf is a goal state and a leaf is reachable from every point in the plan.

- Which is the cause of the nondeterminism? It is an action execution failure that can be successfully if we try repeatedly, or it is due to some unobserved fact about the robot or environment?
- One way to solve it is to **change the initial problem** formulation (fully observable, nondeterministic) to **partially observable, deterministic**, where the failure of the cyclic plan is attributed to an unobserved fact.

Searching in partially observable environments

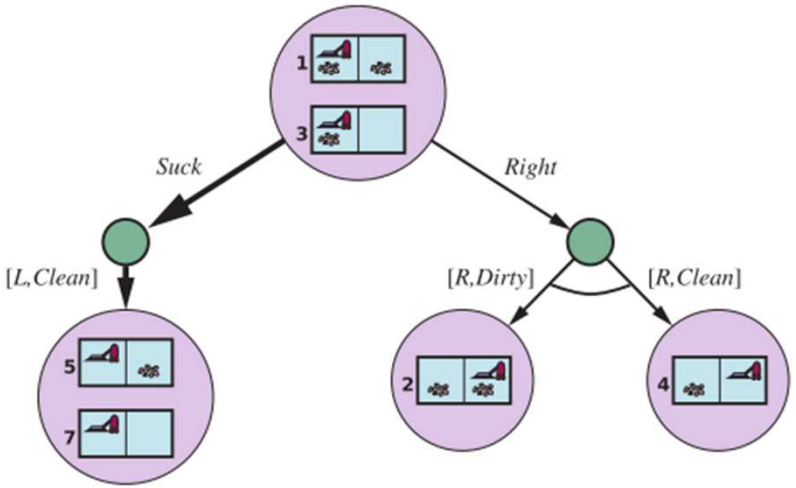
- For a partially observable problem, is specified a PERCEPT(s) function that returns the percept received by the agent in a given state.
  - If sensing is nondeterministic, PERCEPTS function returns a set of possible percepts.
  - For fully observable problems, PERCEPT(s)=s for every state s, and for sensorless PERCEPT(s)=null.
- Consider a local-sensing vacuum world, in which the agent has
  - a **position sensor** that yields the percept L/R percepts, and
  - a **dirt sensor** that yields Dirty/Clean percepts
- With partial observability, it will usually be the case that **several states produce the same percept**;
  - E.g., state 3 will also produce [L, Dirty]. Hence, given this initial percept, the initial belief state will be {1,3}
- Top figure represents the **deterministic world** case
- Bottom figure represents the **nondeterministic world** case



Solving partially observable problems

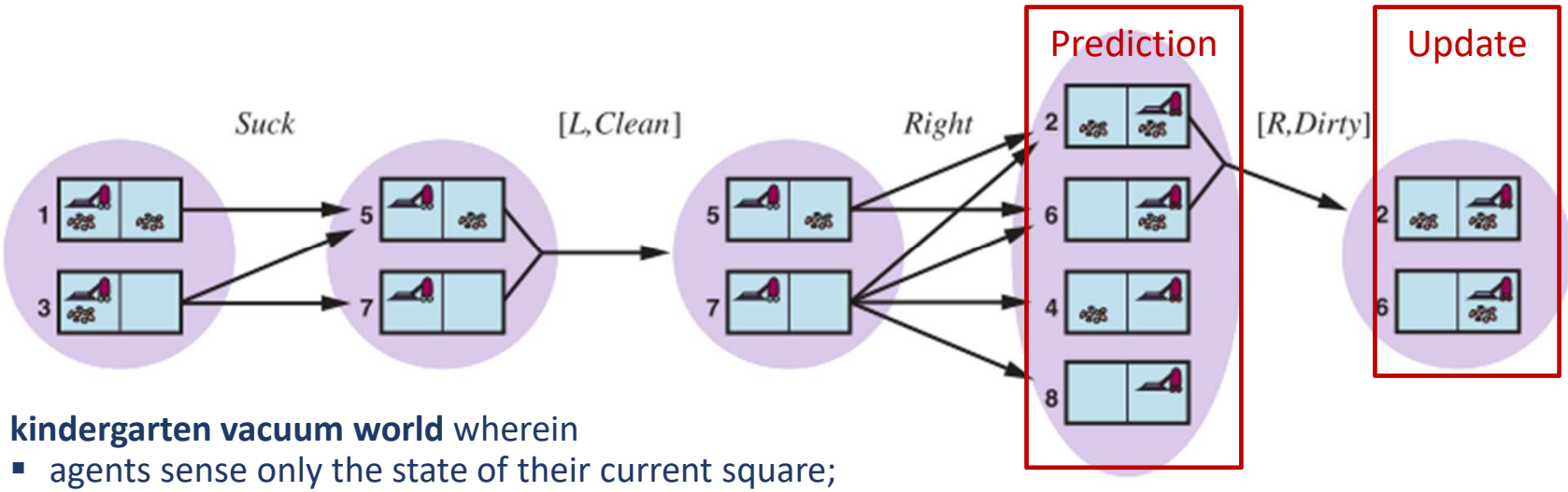
- The AND–OR search algorithm can be applied directly to derive a solution. The figure shows part of the search tree for the local-sensing vacuum world, assuming an initial percept [L, Dirty]
- The solution is the conditional plan

  [Suck,Right, if Rstate={6} then Suck else [ ]]
- It returned a **conditional plan** that tests the belief state
  - That's because we supplied a belief-state problem
  - In this environment the agent won't know the actual state.
- The AND–OR search algorithm treats belief states as black boxes
  - One can improve on this by checking for previously generated belief states that are subsets or supersets of the current state
- Figure shows the first level of the AND–OR search tree for a problem in the local-sensing vacuum world.
  - Suck is the first action in the solution



Solving partially observable problems – e.g. the kindergarten vacuum world

- There are two main differences between this agent and the one for fully observable deterministic environments.
  - First, the solution will be a conditional plan rather than a sequence; to execute an if–then–else expression, the agent will need to test the condition and execute the appropriate branch of the conditional.
  - Second, the agent will need to maintain its belief state as it performs actions and receives percepts
- The agent will do a process of prediction–observation–update
  - Bottom figure shows two **prediction–update** cycles of belief-state maintenance in the vacuum world with local sensing



kindergarten vacuum world wherein

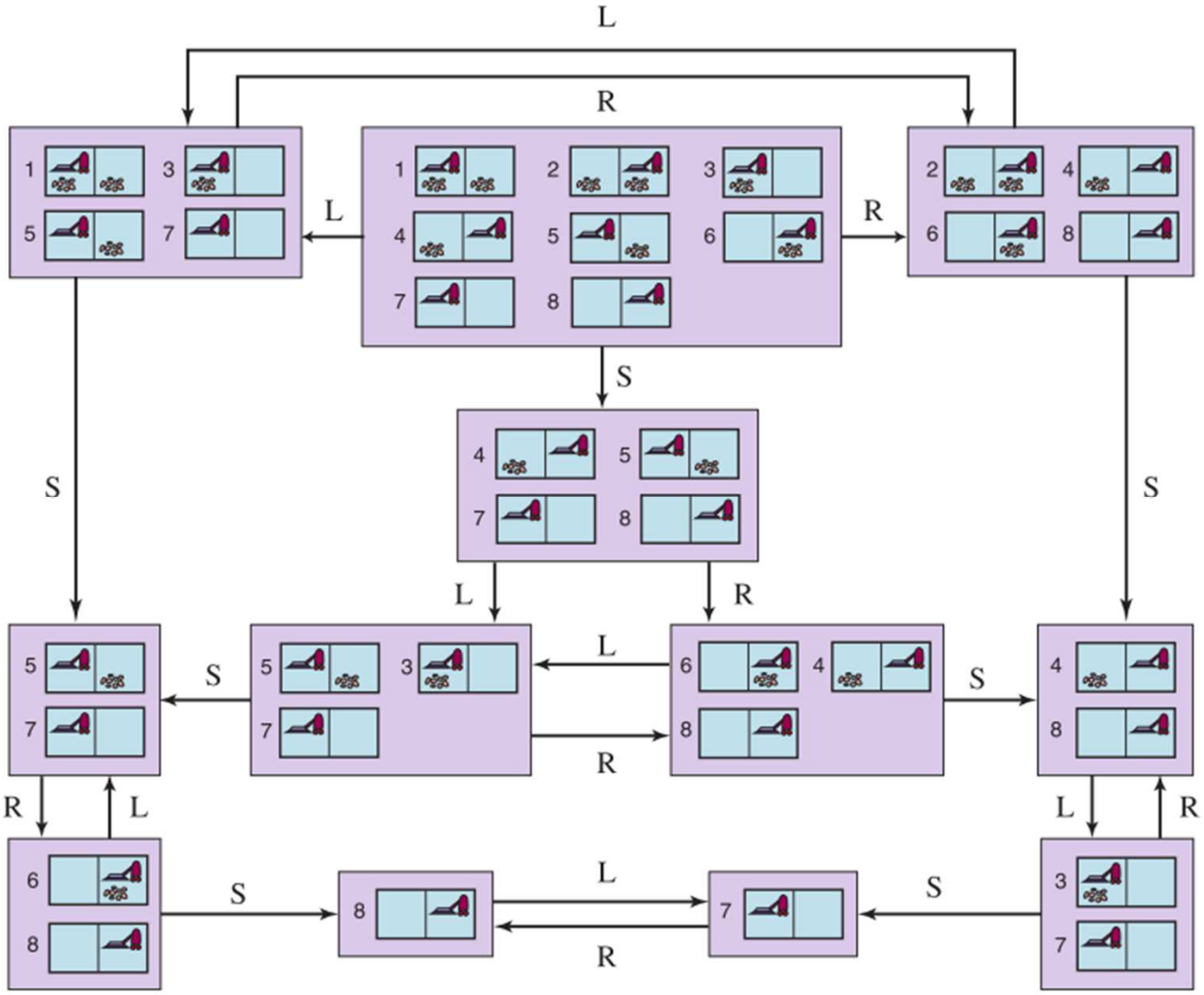
- agents sense only the state of their current square;
- any square may become dirty at any time unless the agent is actively cleaning it at that moment.

Searching with no observation

- When the agent's percepts provide no information at all, we have what is called a **sensorless problem**
  - Sensorless solutions are surprisingly common and useful
  - Sometimes a sensorless plan is better even when a conditional plan with sensing is available
- In the vacuum world, assume that the agent knows the geography of its world, but not its own location or the distribution of dirt
  - its initial belief state is {1,2,3,4,5,6,7,8}, and when it moves Right it will be in {2,4,6,8}
  - After [Right,Suck] it will end up in {4,8} —the agent has gained information without perceiving anything
  - Finally, after [Right,Suck,Left,Suck] the agent is guaranteed to reach the goal state 7
  - No matter what the start state. We say that the agent can **coerce** the world into state 7
- The **solution to a sensorless problem is a sequence of actions**, not a conditional plan (there is no perceiving).
- We **search in the space of belief states** rather than physical states.
  - In belief-state space, the problem is fully observable because the agent always knows its own belief state.
  - The solution (if any) for a sensorless problem is always a **sequence of actions**
  - This is true even if the environment is nondeterministic

Belief-state space for the deterministic, sensorless vacuum world

- Each rectangular box corresponds to a **single belief state**.
- At any given point, the agent has a belief state but **does not know which physical state** it is in.
- The initial belief state is the complete ignorance, in the case, the top center box.



Thank you!