Planning Agents

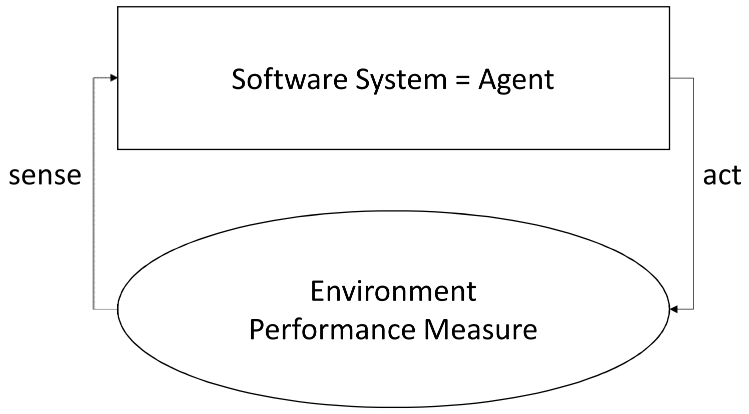
Sven Koenig, USC

Russell and Norvig, 3rd Edition, Sections 2.4 and 3.1-3.2

These slides are new and can contain mistakes and typos.
Please report them to Sven (skoenig@usc.edu).

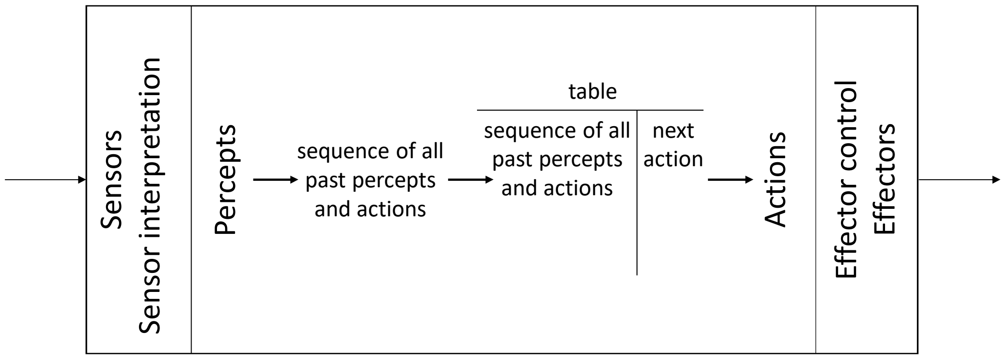
Search and Planning

- We now start with (deterministic) search and planning.



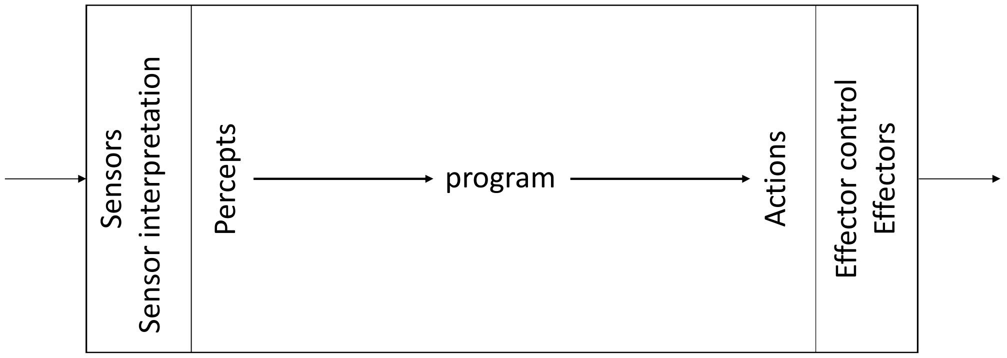
Architectures for Planning Agents

- Gold standard (but: results in a large table that is difficult to change)



Architectures for Planning Agents

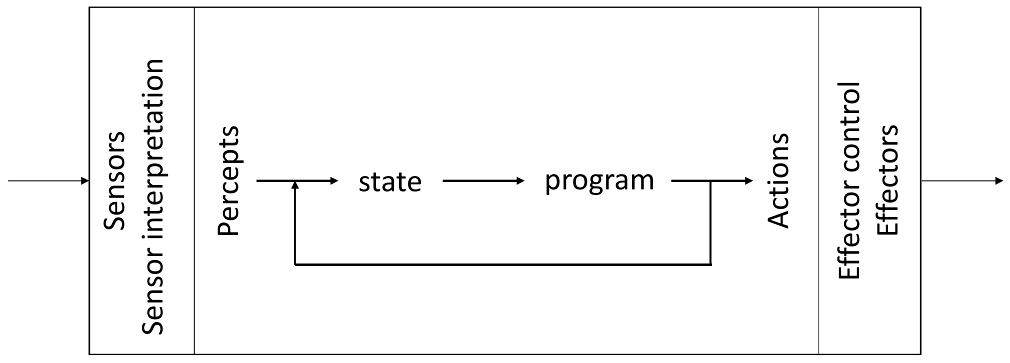
- Reflex agent (“reactive planning”): often good for video games



Architectures for Planning Agents

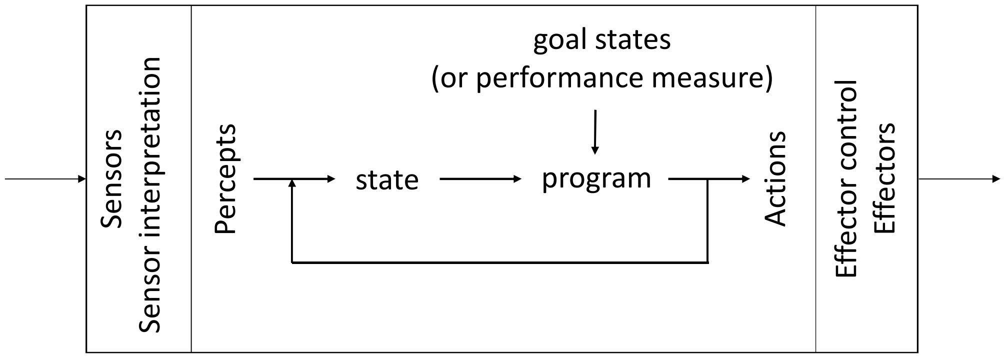
- An agent often does not need to remember the sequence of past percepts and actions to perform well according to its performance objective.
- A state characterizes the information that an agent needs to have about the present to pick actions in the future to perform well according to its performance objective.
- We are typically interested in minimal states.
- For example, a soda machine does not need to remember in which order coins were inserted and in which order coins and products were returned in the past. It only needs to remember the total amount of money inserted by the current customer.

Architectures for Planning Agents



Architectures for Planning Agents

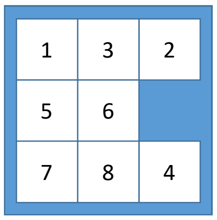
- Planning agent

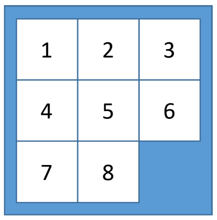


Examples

- What are the states, actions and action costs?

- Eight puzzle


start (= current) configuration


goal configuration

Examples

- What are the states, actions and action costs?
- Missionaries and cannibals problem

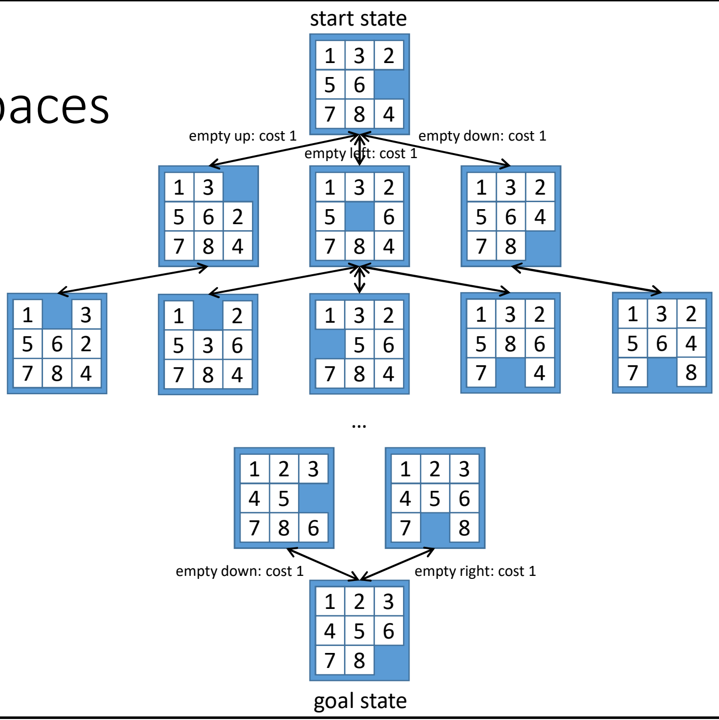
  Three missionaries and three cannibals are on the left side of a river, along with a boat that can hold one or two people. Find the quickest way to get everyone to the other side, without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.

Examples

- What are the states, actions and action costs?
- Traveling salesperson problem

  Visit all given cities in the plane with a shortest tour (= with the smallest round-trip distance).

State Spaces



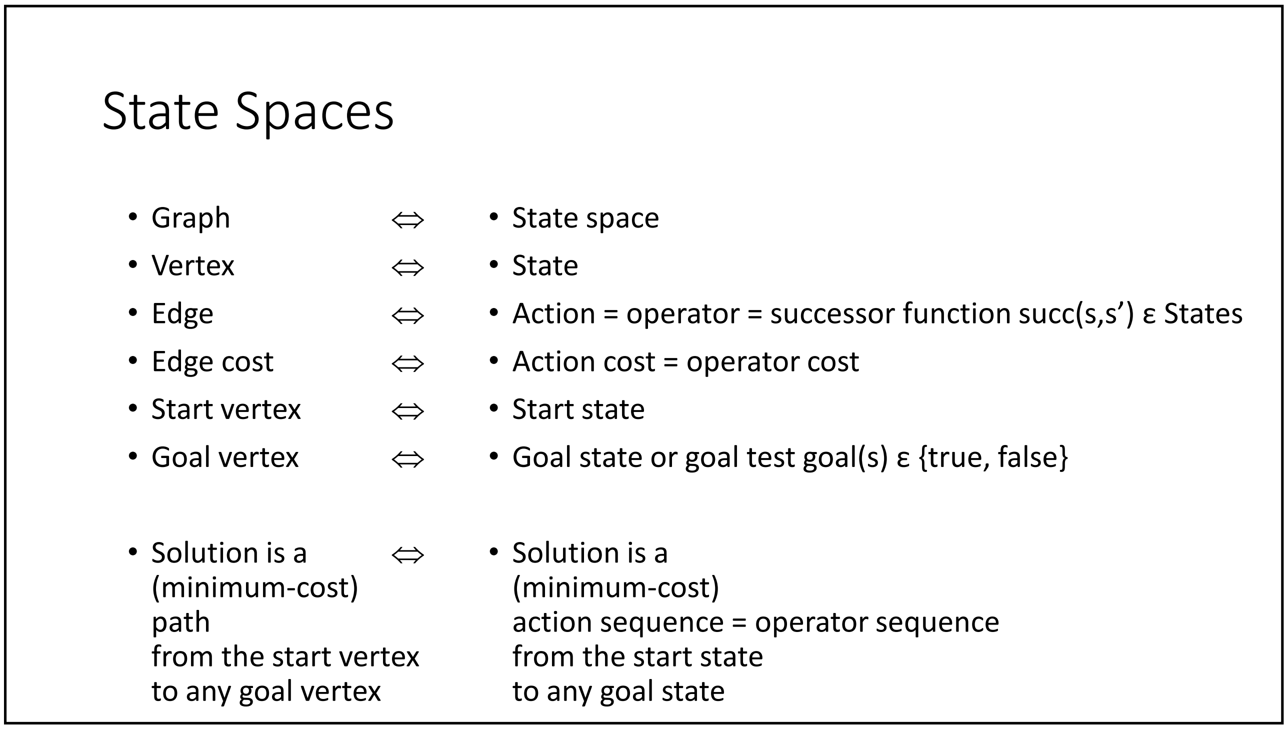
State Spaces

- Example application of hillclimbing: Boolean satisfiability
  - Find an interpretation that makes a given propositional sentence true.
  - Transform the propositional sentence into conjunctive normal form, assign random truth values to all propositional symbols, then repeatedly switch the truth value of some symbol to decrease the number of clauses that evaluate to false.
  - S ≡ (P OR Q) AND (NOT P OR NOT R) AND (P OR NOT Q OR R)

start state = assignment of random truth values to all propositional symbols

$$\begin{tikzcd} & \text{P, NOT Q, R} \arrow[dl, leftrightarrow, "\text{flip P}"'] \arrow[d, leftrightarrow, "\text{flip Q}"] \arrow[dr, leftrightarrow, "\text{flip R}"] & \\ \text{NOT P, NOT Q, R} & \text{P, Q, R} & \text{P, NOT Q, NOT R} \\ & \dots & \end{tikzcd}$$

- Costs do not matter since we are not interested in finding a minimum-cost path.
- There is more than one goal state, e.g. P, Q, NOT R and P, NOT Q, NOT R.
- There is a goal test, namely whether S is true.



State Spaces

• Graph	⇔	• State space
• Vertex	⇔	• State
• Edge	⇔	• Action = operator = successor function succ(s,s’) ε States
• Edge cost	⇔	• Action cost = operator cost
• Start vertex	⇔	• Start state
• Goal vertex	⇔	• Goal state or goal test goal(s) ε {true, false}
• Solution is a (minimum-cost) path from the start vertex to any goal vertex	⇔	• Solution is a (minimum-cost) action sequence = operator sequence from the start state to any goal state