Problem-solving agents

- Problem-solving agents is a special kind of goal-based agent.

- Tasks:
  - How to define a problem?
  - Problem type of formulation depends on available knowledge.
  - Problems, solutions, what is this?

- Six different search strategies.

Well-defined problems and solutions

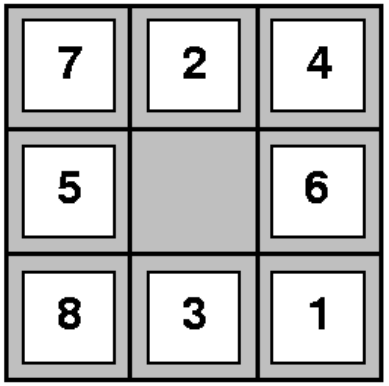
- Problems and solutions:
  - Initial state
  - Possible **actions** available to the agent, e.g. **successor function**. **State space**: the set of all states reachable from the initial state. A **path** in the state space is a sequence of states connected by a sequence of actions.
  - Goal test:
    - Simple check: actual state = goal.
    - Abstract property rather than an explicitly enumerated set of states (e.g. chess).
  - A **path cost** function that assigns a numeric cost to each path

- Initial state, set of actions. goal test and path cost functions together define a problem.

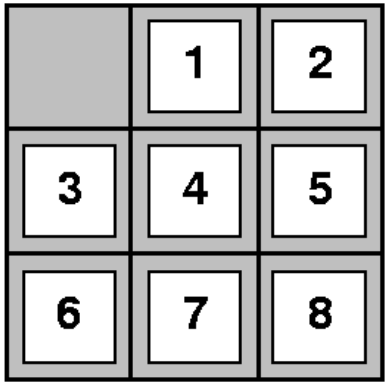
- Solution
  - Path from initial state to goal.

8 - Puzzle

- Family: sliding-block puzzle, NP-complete, Standard test for new algorithms in AI
  - States
    - Location of the 8 tiles in one of the 9 areas plus blank
    - In total 9!/2 = 181,440 reachable states
  - Operators
    - Blank moves right, left, up or down.
  - Goal-test
    - State like image on the right.
  - Path costs
    - Each step costs 1.



Start State

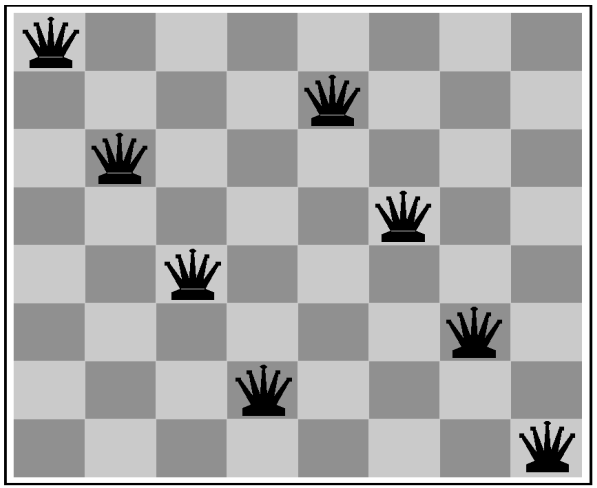


Goal State

8 - Queens problem

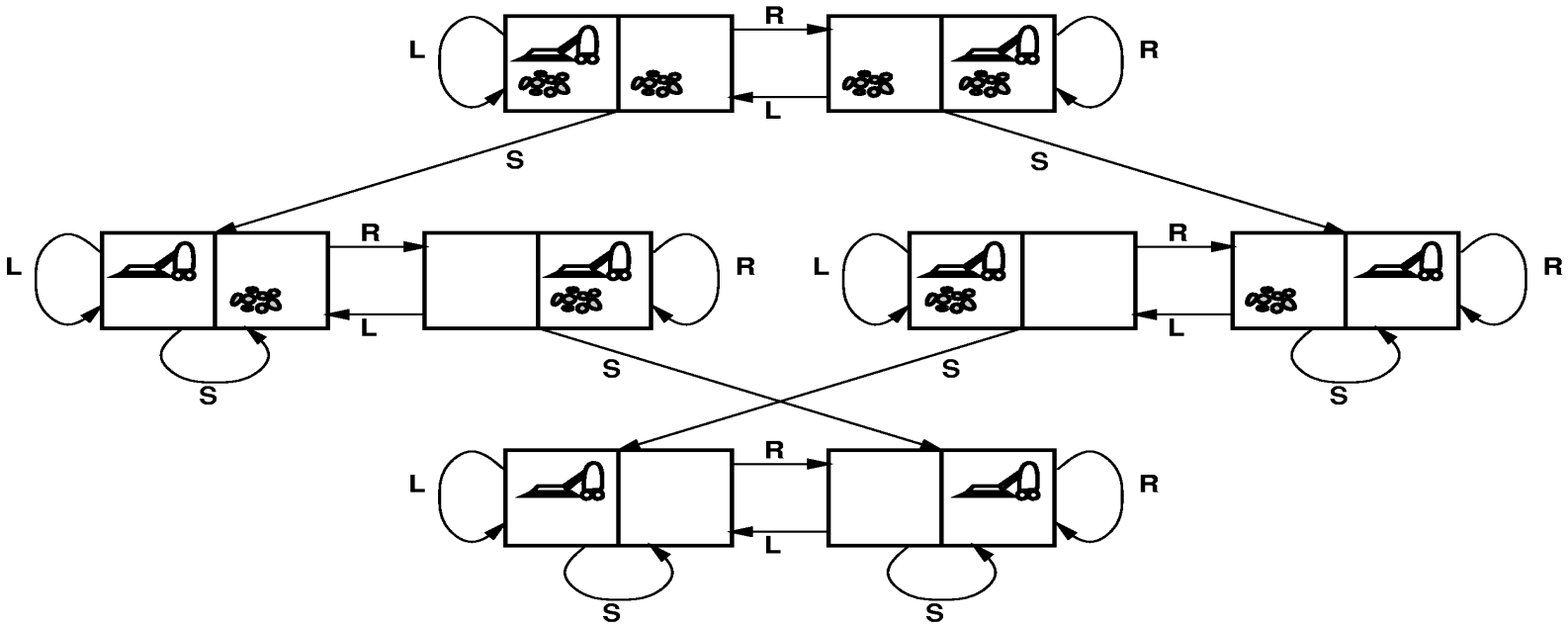
(incremental and complete-state formulation)

- States
  - Any arrangement of 0 to 8 queens on the board.
- Initial state
  - No queens on board.
  - Arrangements of 0 to 8 queens in the leftmost columns with none attacked.
- Successor function
  - Add a queen to any empty square.
  - Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.
- Goal test
  - 8 queens on board, none attacked.
- In this formulation, we have 64*63...57 ≈$3*10^{14}$ possible sequences to investigate.
- Path costs
  - 0, only final state counts



Vacuum cleaner problem

- States
  - 8 states, see figure
- Successor function
  - Move right, left and suck
- Goal test
  - No dirt
- Path costs
  - Each step costs 1

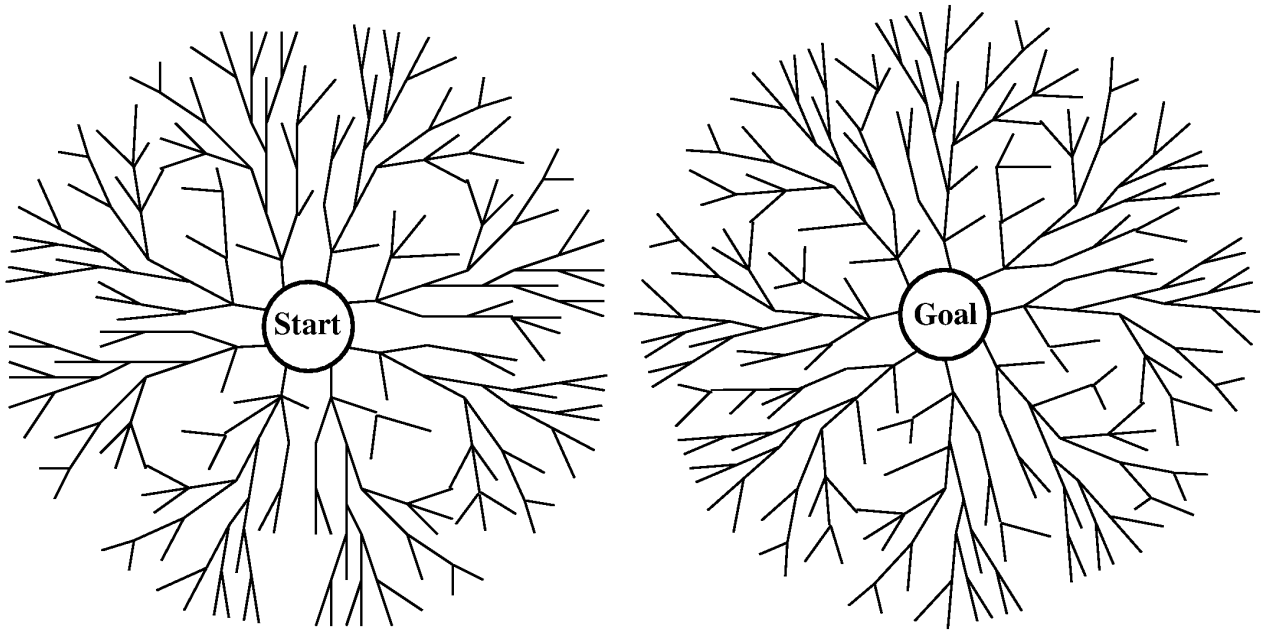


Examples (real world)

- Finding routes
  - Routing in networks
  - Automatic travel routes
  - Airline planning systems
- Traveling Salesman Problem (TSP)
- VLSI Layout
  - Chip layout
- Robot navigation
  - Generalization of route-finding problem (continuous space)
- Search in the Internet

Bidirectional search

- **Idea:** to run two simultaneous searches—one forward from the initial state and the other backward from the goal
- Stop when the two searches meet in the middle
- Motivation: $b^{d/2} + b^{d/2} << b^d$

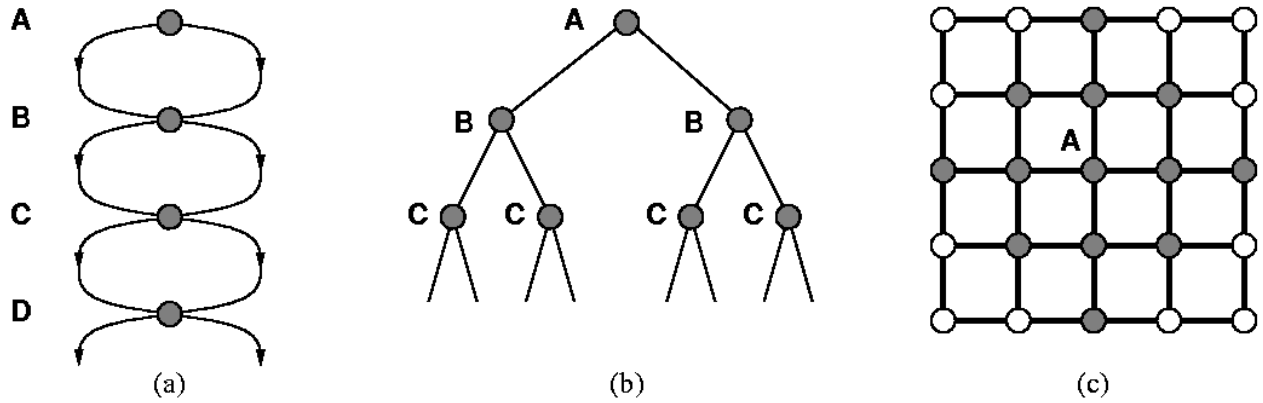


Bidirectional search (2)

- Discussion:
  - Time: $O(b^{d/2})$, checking a node for membership in the other search tree can be done in constant time with a hash table
  - Example: for *b=10* and *d=6* breadth-first would create 1,111,100 nodes, bidirectional search only 2,200 (depth=3!).
  - Space: $O(b^{d/2})$, because one of the trees has to be kept in memory
  - What has to be considered?
    - Predecessor for backward search
    - Efficiency to check whether current node already exists in other tree
    - What kind of search?
    - ...

Avoiding repeated states

- Options:
  - Don't go back to way you came from
  - Don't create paths with cycles
  - Don't generate states that have been generated before



(a): A state space in which there are two possible actions leading from A to B, two from B to C, and so on. The state space contains d+1 states, where d is the maximum depth.

(b): The corresponding search tree, which has 2^d branches corresponding to the 2^d paths through the space

(c): A rectangular grid space. States within 2 steps of the initial state (A) are shown in gray.

State space landscape

- Problem: depending on initial state, can get stuck in local maxima