Problem Solving by Searching

Prepared by
Lec Shekh. Md. Saifur Rahman

Problem-Solving Agents

- Intelligent agents can solve problems by searching a state space
- State-space Model
  - The agent's model of the world
  - Usually a set of discrete states
  - E.g., in driving, the states in the model could be towns/cities
- Goal State(s)
  - A goal is defined as a desirable state for an agent
  - There may be many states that satisfy the goal condition
    - E.g., drive to a town with a ski resort
  - Or just one state that satisfies the goal
    - E.g., drive to Moulvibazar
- Operators (actions, successor function)
  - Operators are legal actions that the agent can take to move from one state to another

Initial Simplifying Assumptions

- The environment is static
  - ➢ No changes in the environment while the problem is being solved

- The environment is observable

- The environment and actions are discrete
  - ➢ (Typically assumed, but we will see some exceptions)

- The environment is deterministic

Example: Traveling in Romania

- On holiday in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

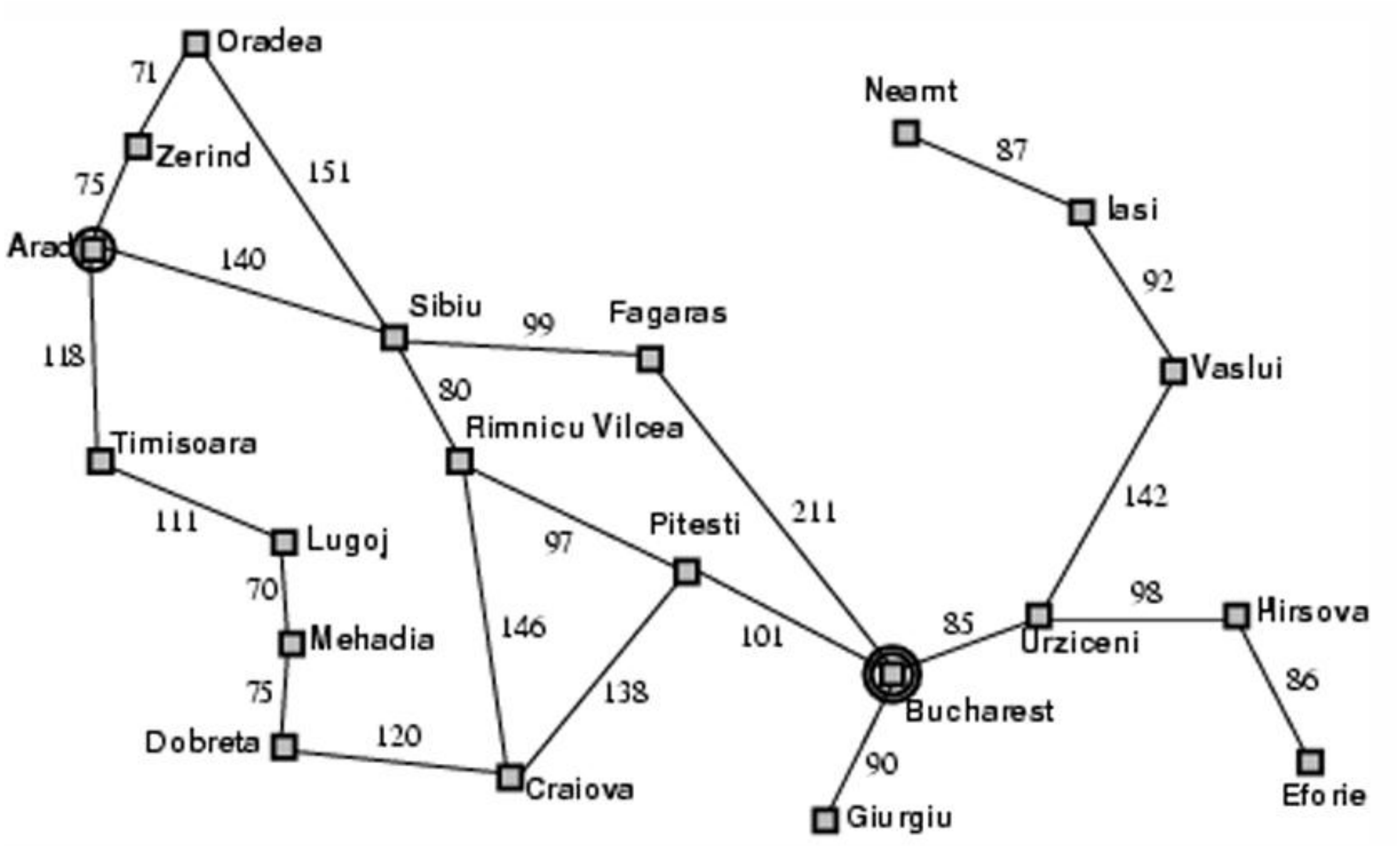
- Formulate goal:
  - be in Bucharest

- Formulate problem:
  - states: various cities
  - actions/operators: drive between cities

- Find solution
  - By searching through states to find a goal
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

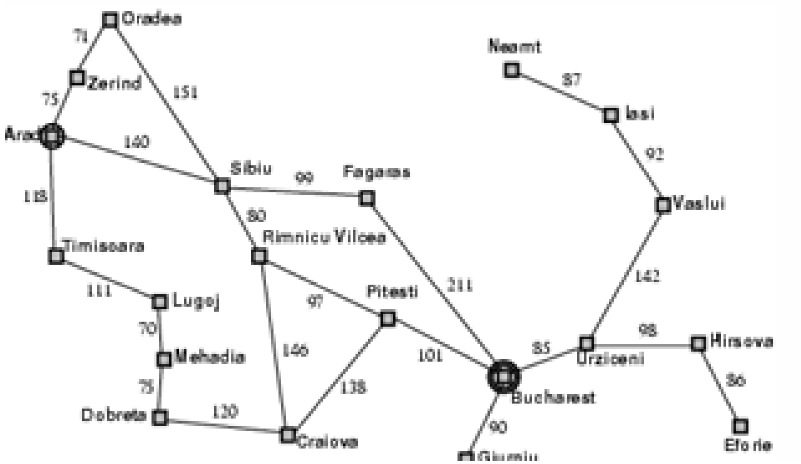
- Execute states that lead to a solution

Example: Traveling in Romania



State-Space Problem Formulation

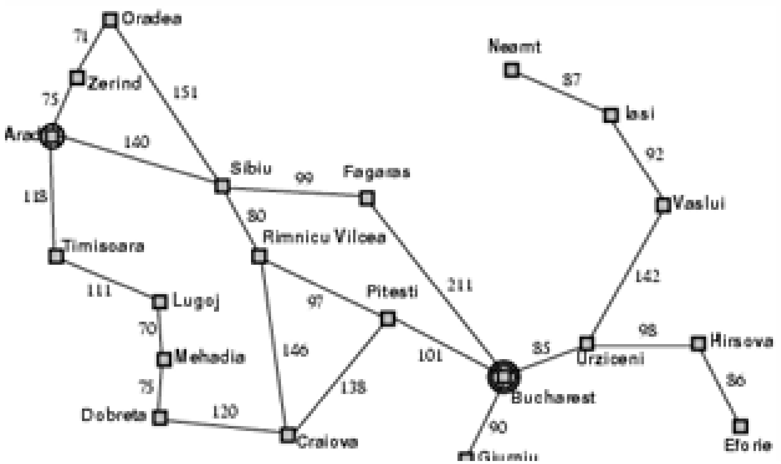
A **problem** is defined by four items:



State-Space Problem Formulation

A **problem** is defined by four items:

1. **initial state** e.g., "at Arad"



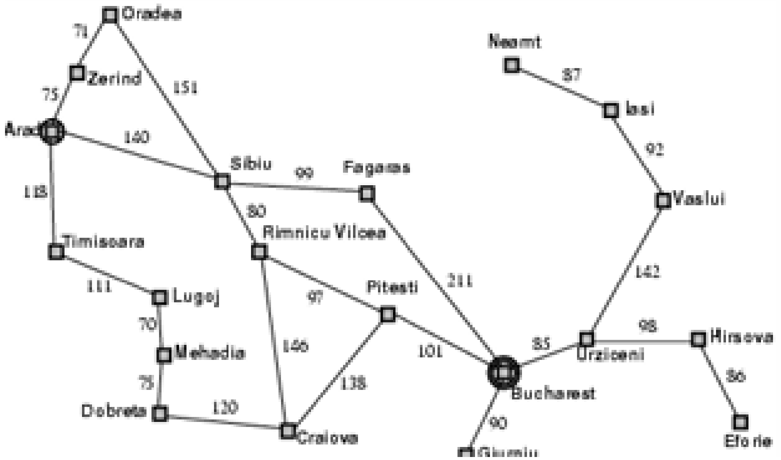
State-Space Problem Formulation

A **problem** is defined by four items:

1. **initial state** e.g., "at Arad"
2. **actions** or successor function

   $S(x)$ = set of action–state pairs

   e.g., $S(Arad) = \{<Arad \rightarrow Zerind, Zerind>, ..$



State-Space Problem Formulation

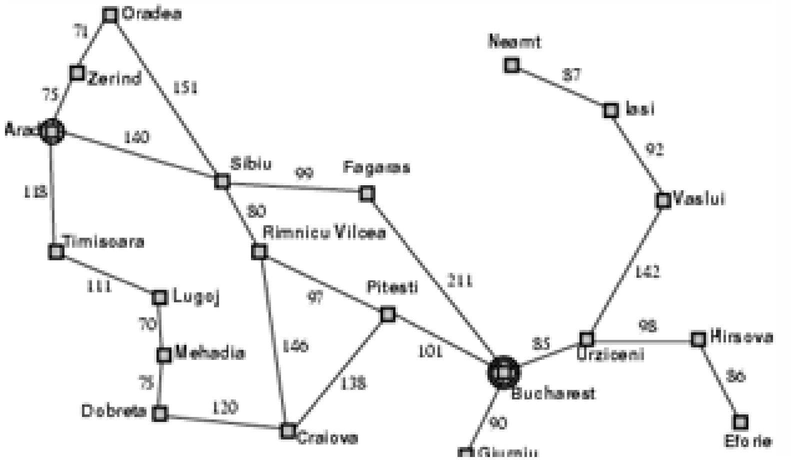
A **problem** is defined by four items:

1. **initial state** e.g., "at Arad"
2. **actions** or successor function

   S(x) = set of action–state pairs

   e.g., *S(Arad) = {<Arad → Zerind, Zerind>, ..*
3. **goal test** (or set of goal states)

   e.g., *x* = "at Bucharest", *Checkmate(x)*



State-Space Problem Formulation

A **problem** is defined by four items:

1. **initial state** e.g., "at Arad"

2. **actions** or successor function

   S(x) = set of action–state pairs

   e.g., *S(Arad)* = {<*Arad → Zerind, Zerind*>, ..

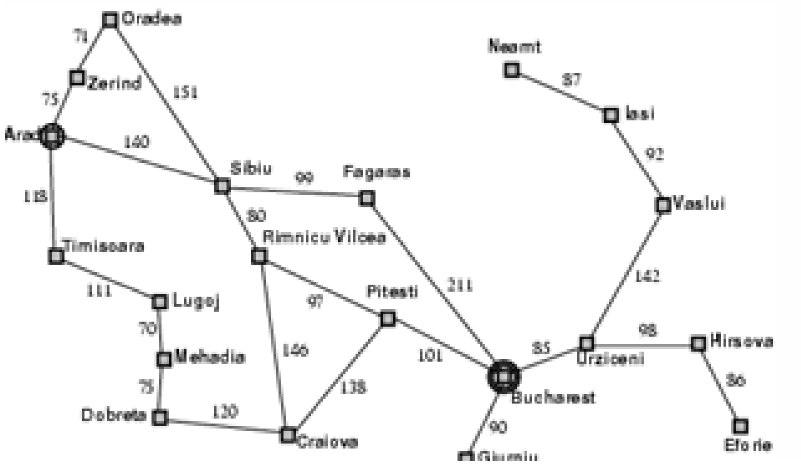
3. **goal test** (or set of goal states)

   e.g., *x* = "at Bucharest", *Checkmate(x)*

4. **path cost** (additive)

   e.g., sum of distances, number of actions executed, etc.

   c(x,a,y) is the step cost, assumed to be ≥ 0



State-Space Problem Formulation

A **problem** is defined by four items:

1. **initial state** e.g., "at Arad"
2. **actions** or successor function

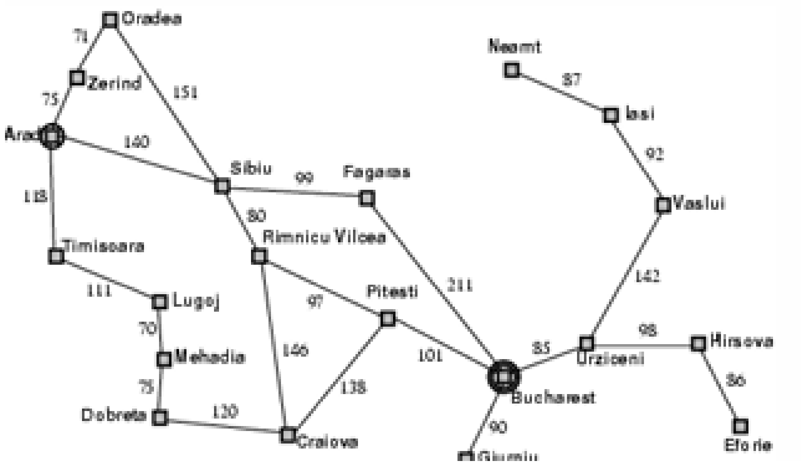
   $S(x)$ = set of action–state pairs

   e.g., $S(Arad) = \{<Arad \rightarrow Zerind, Zerind>, ..$
3. **goal test** (or set of goal states)

   e.g., x = "at Bucharest", $Checkmate(x)$
4. **path cost** (additive)

   e.g., sum of distances, number of actions executed, etc.

   $c(x,a,y)$ is the step cost, assumed to be ≥ 0



A **solution** is a sequence of actions leading from the initial state to a goal state

Example: Formulating the Navigation Problem

- Set of States
  - individual cities
  - e.g., Irvine, SF, Las Vegas, Reno, Boise, Phoenix, Denver

- Operators
  - freeway routes from one city to another
  - e.g., Irvine to SF via 5, SF to Seattle, etc

- Start State
  - current city where we are, Irvine

- Goal States
  - set of cities we would like to be in
  - e.g., cities which are closer than Irvine

- Solution
  - a specific goal city, e.g., Boise
  - a sequence of operators which get us there,
    - e.g., Irvine to SF via 5, SF to Reno via 80, etc

Abstraction

- Definition of Abstraction:
  Process of removing irrelevant detail to create an abstract representation: ``high-level", ignores irrelevant details

- Navigation Example: how do we define states and operators?
  - First step is to abstract "the big picture"
    - i.e., solve a map problem
    - nodes = cities, links = freeways/roads (a high-level description)
    - this description is an abstraction of the real problem
  - Can later worry about details like freeway onramps, refueling, etc

- Abstraction is critical for automated problem solving
  - must create an approximate, simplified, model of the world for the computer to deal with: real-world is too detailed to model exactly
  - good abstractions retain all important details

The State-Space Graph

- Graphs:
  - nodes, arcs, directed arcs, paths

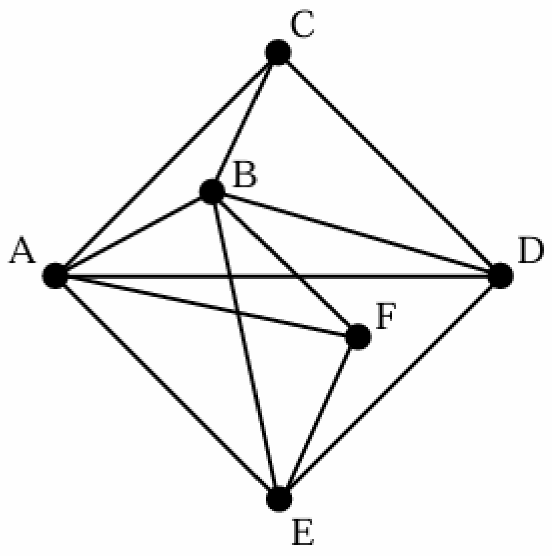
- Search graphs:
  - States are nodes
  - operators are directed arcs
  - solution is a path from start S to goal G

- Problem formulation:
  - Give an abstract description of states, operators, initial state and goal state.

- Problem solving:
  - Generate a part of the search space that contains a solution

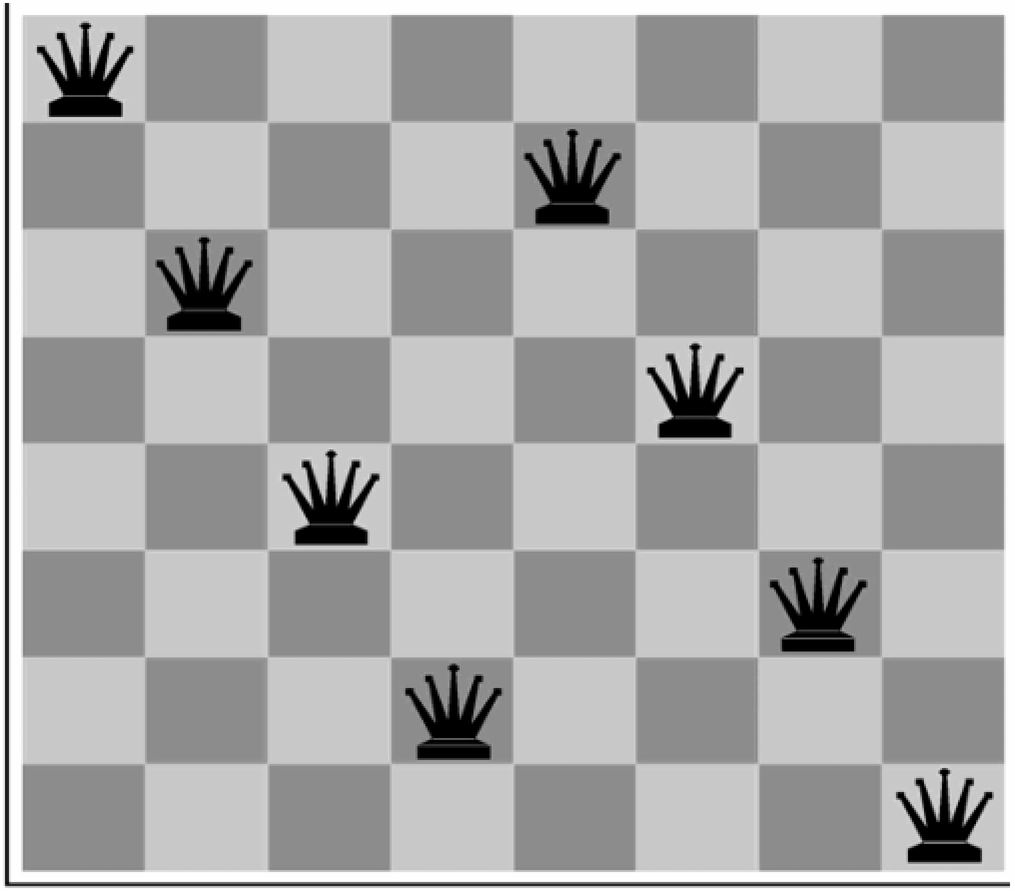
Example: The Traveling Salesman Problem

- Find the shortest tour that visits all cities without visiting any city twice and return to starting point.
- State: sequence of cities visited
- $S_0 = A$



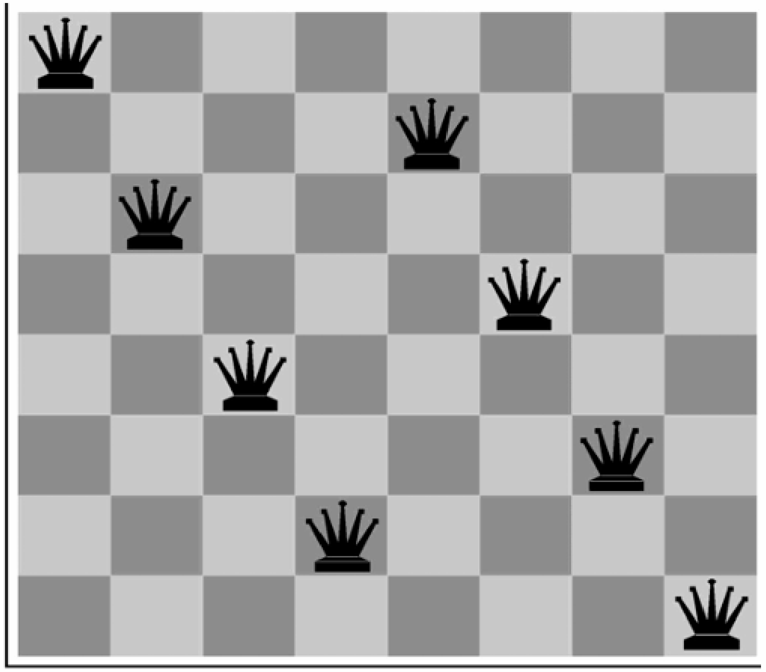
- G = a complete tour

Example: 8-queens Problem



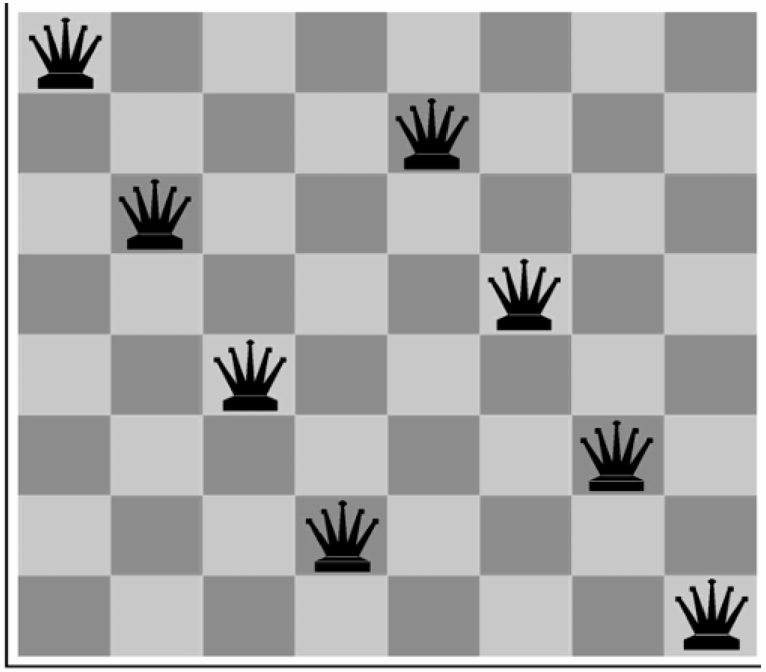
State-Space Problem Formulation

- states? -any arrangement of n<=8 queens
  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.



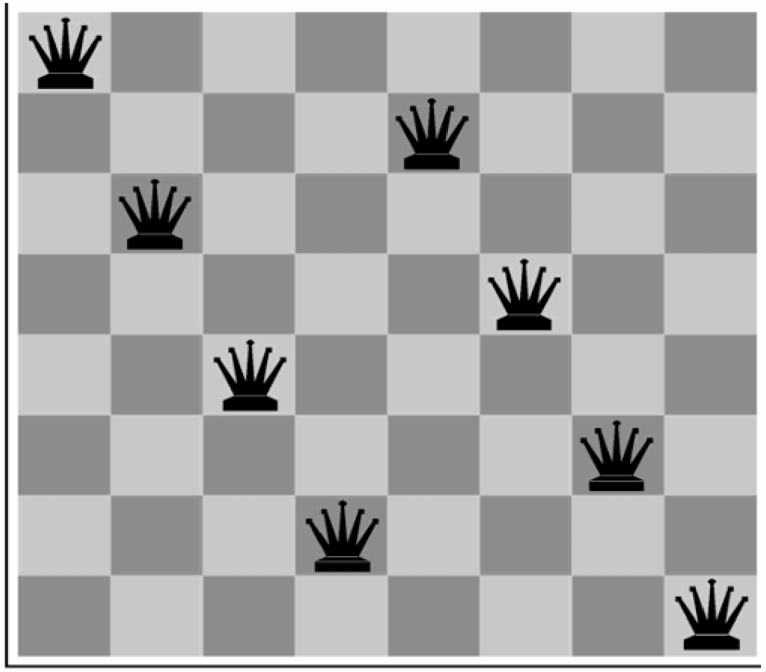
State-Space Problem Formulation

- states? -any arrangement of n<=8 queens
  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.
- initial state? no queens on the board



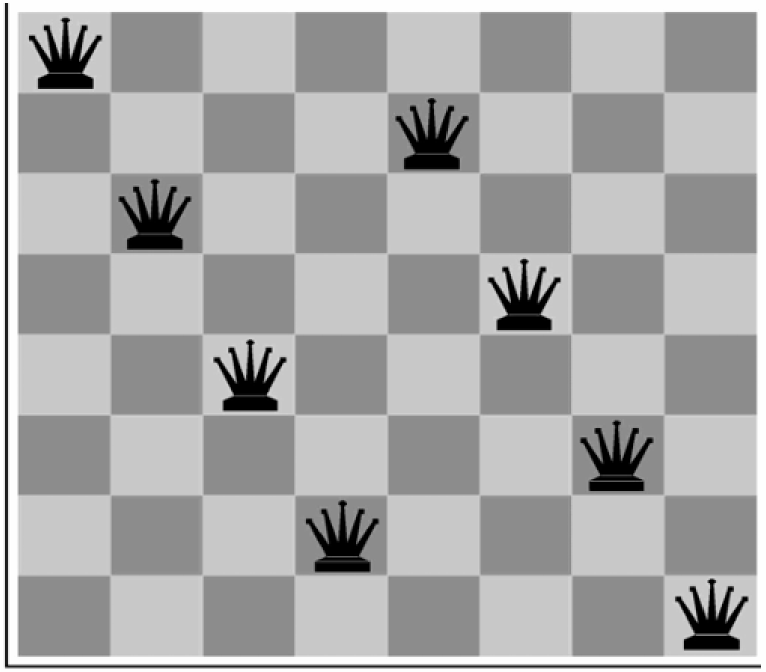
State-Space Problem Formulation

- states? -any arrangement of n<=8 queens
  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.
- initial state? no queens on the board
- actions? -add queen to any empty square
  -*or* add queen to leftmost empty square such that it is not attacked by other queens.



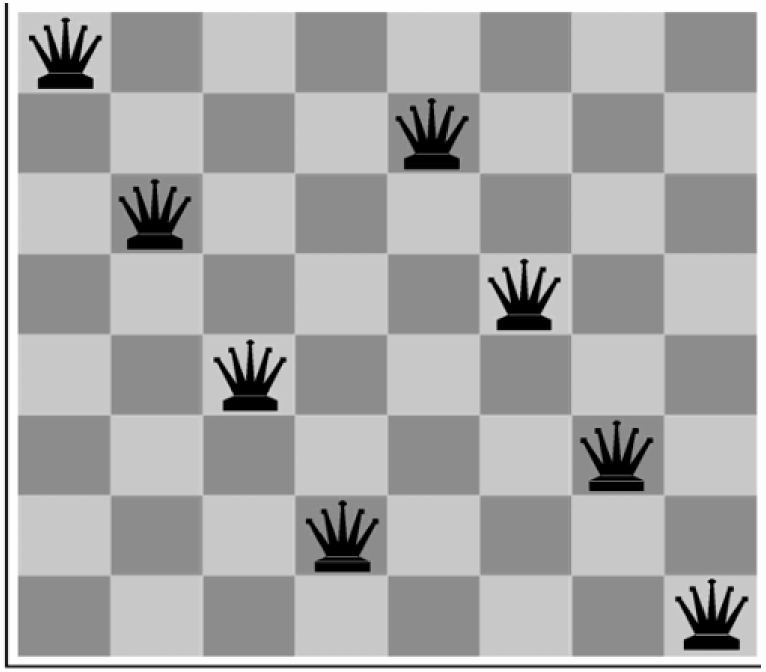
State-Space Problem Formulation

- states? -any arrangement of n<=8 queens
  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.
- initial state? no queens on the board
- actions? -add queen to any empty square
  -*or* add queen to leftmost empty square such that it is not attacked by other queens.
- goal test? 8 queens on the board, none attacked.

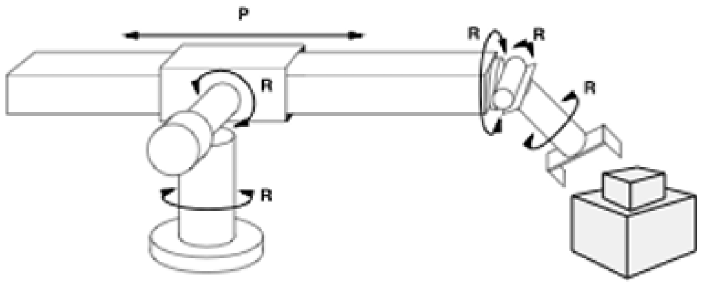


State-Space Problem Formulation

- states? -any arrangement of n<=8 queens
  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.
- initial state? no queens on the board
- actions? -add queen to any empty square
  -*or* add queen to leftmost empty square such that it is not attacked by other queens.
- goal test? 8 queens on the board, none attacked.
- path cost? 1 per move

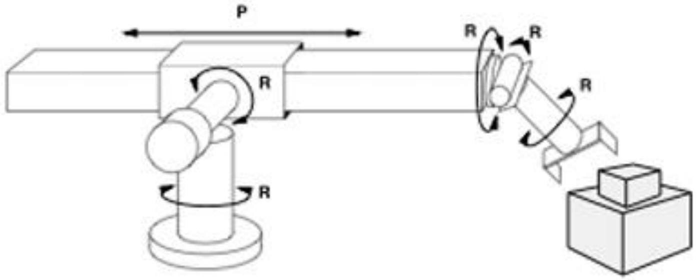


Example: Robot Assembly



- States
- Initial state
- Actions
- Goal test
- Path Cost

Example: Robot Assembly



- States: configuration of robot (angles, positions) and object parts
- Initial state: any configuration of robot and object parts
- Actions: continuous motion of robot joints
- Goal test: object assembled?
- Path Cost: time-taken or number of actions

Learning a Spam Email Classifier

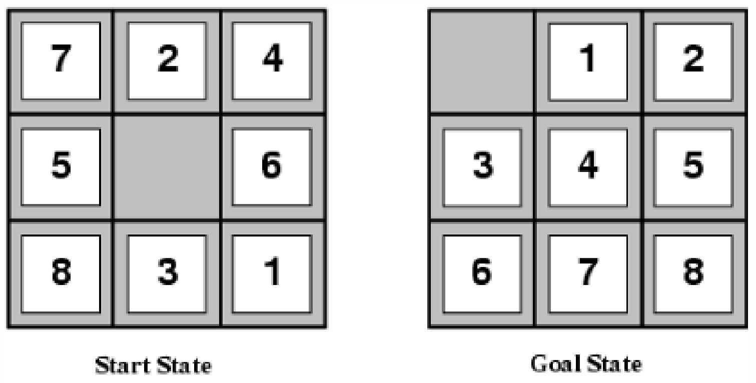
- States
- Initial state
- Actions
- Goal test
- Path Cost

Learning a Spam Email Classifier

- States: settings of the parameters in our model
- Initial state: random parameter settings
- Actions: moving in parameter space
- Goal test: optimal accuracy on the training data
- Path Cost: time taken to find optimal parameters

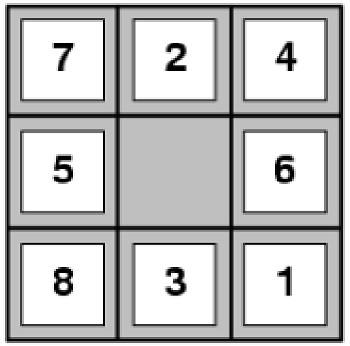
(Note: this is an optimization problem – many machine learning problems can be cast as optimization)

Example: 8-puzzle

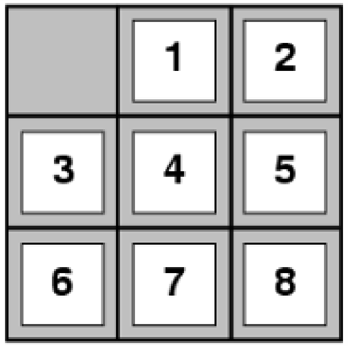


- states?
- initial state?
- actions?
- goal test?
- path cost?

Example: 8-puzzle



Start State

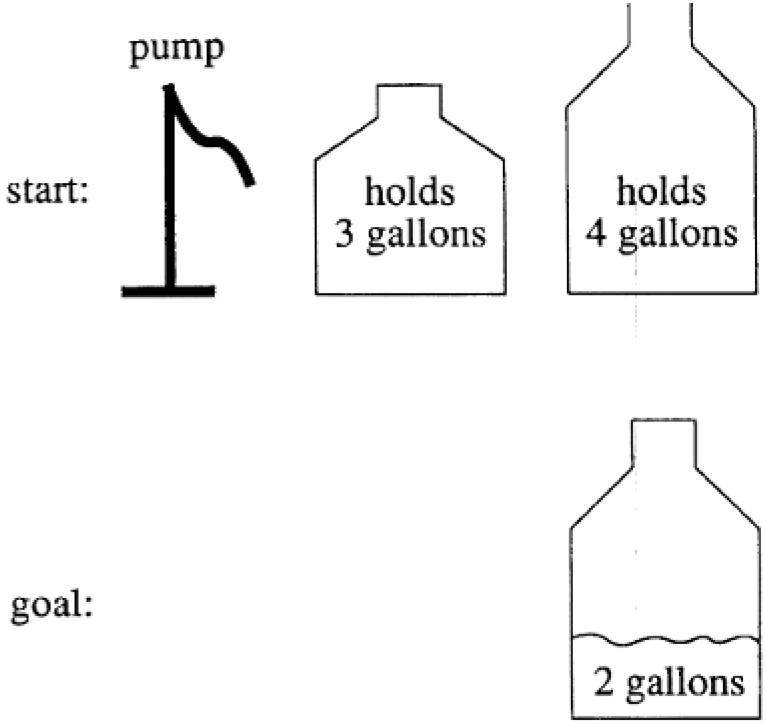


Goal State

- states? locations of tiles
- initial state? given
- actions? move blank left, right, up, down
- goal test? goal state (given)
- path cost? 1 per move

A Water Jug Problem

- You have a 4-gallon and a 3-gallon water jug
- You have a faucet with an unlimited amount of water
- You need to get exactly 2 gallons in 4-gallon jug



Puzzle-solving as Search

- State representation: **(x, y)**
  - x: Contents of four gallon
  - y: Contents of three gallon
- Start state**: (0, 0)**
- Goal state **(2, n)**
- Operators
  - Fill 3-gallon from faucet, fill 4-gallon from faucet
  - Fill 3-gallon from 4-gallon , fill 4-gallon from 3-gallon
  - Empty 3-gallon into 4-gallon, empty 4-gallon into 3-gallon
  - Dump 3-gallon down drain, dump 4-gallon down drain

Next Topics

- Uninformed search
  - Breadth-first, depth-first
  - Uniform cost
  - Iterative deepening
- Informed (heuristic) search
  - Greedy best-first
  - A*
  - Memory-bounded heuristic search
  - And more….
- Local search and optimization
  - Hill-climbing
  - Simulated annealing
  - Genetic algorithms

Summary

- Problem-solving agents where search consists of
  - state space
  - operators
  - start state
  - goal states

- Abstraction and problem formulation