10S3001
Kecerdasan Buatan

Solving Problems by Searching

Arie Satia Dharma, S.T., M.Kom
Dosen Informatika

Modified slides provided by: Ansaf Salleb-Aouissi
Artificial Intelligence, Columbia University, 2018

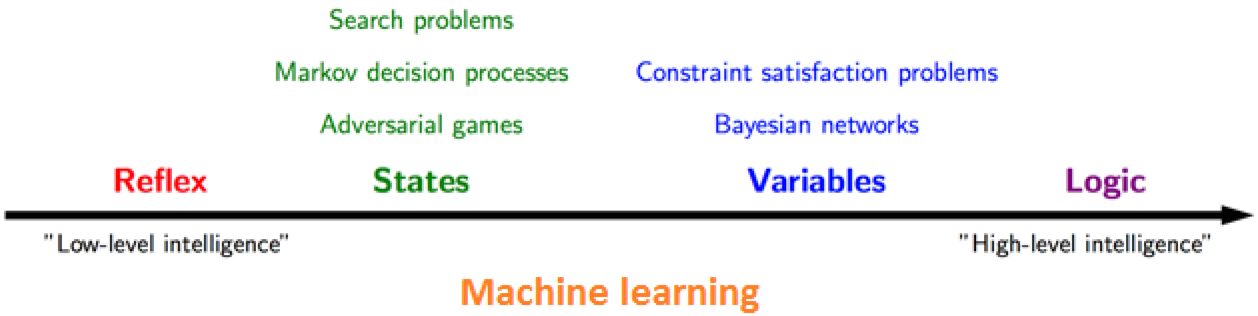
Outlines

- Solving Problems by Searching
- State Space vs Search Space

Solving Problems by Searching

Review: Intelligent Agents

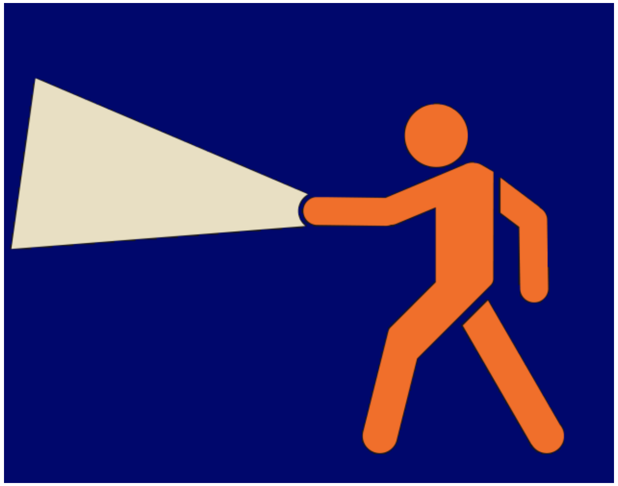
- Four types of agents: simple reflex agents, model-based agents (reflex agents with state), goal-based agents, and utility-based agents.
- Agents can improve their performance through **learning** → **learning agents**.
- This is a high-level present of agent programs.



Credit: Courtesy Percy Liang

Goal-based Agents

- Agents that work towards a **goal**.
- Agents consider the impact of **actions** on future **states**.
- Agent’s job is to identify the action or series of actions that lead to the goal.
- Formalized as a **search** through possible **solutions**.



Examples

Examples

EXPLORE!

Problem Solving as Search

1. **Define the problem through:**
   a) Goal formulation
   b) Problem formulation
2. **Solving the problem as a 2-stage process:**
   a) Search: “mental” or “offline” exploration of several possibilities
   b) Execute the solution found

Problem Formulation

- **Initial state**: the state in which the agent starts

Problem Formulation

- **Initial state**: the state in which the agent starts
- **States**: All states reachable from the initial state by any sequence of actions (**State space**)

Problem Formulation

- **Initial state**: the state in which the agent starts
- **States**: All states reachable from the initial state by any sequence of actions (**State space**)
- **Actions**: possible actions available to the agent. At a state s, $Actions(s)$ returns the set of actions that can be executed in state s. (**Action space**)

Problem Formulation

- **Initial state**: the state in which the agent starts
- **States**: All states reachable from the initial state by any sequence of actions (**State space**)
- **Actions**: possible actions available to the agent. At a state s, $Actions(s)$ returns the set of actions that can be executed in state s. (**Action space**)
- **Transition model**: A description of what each action does $Results(s, a)$

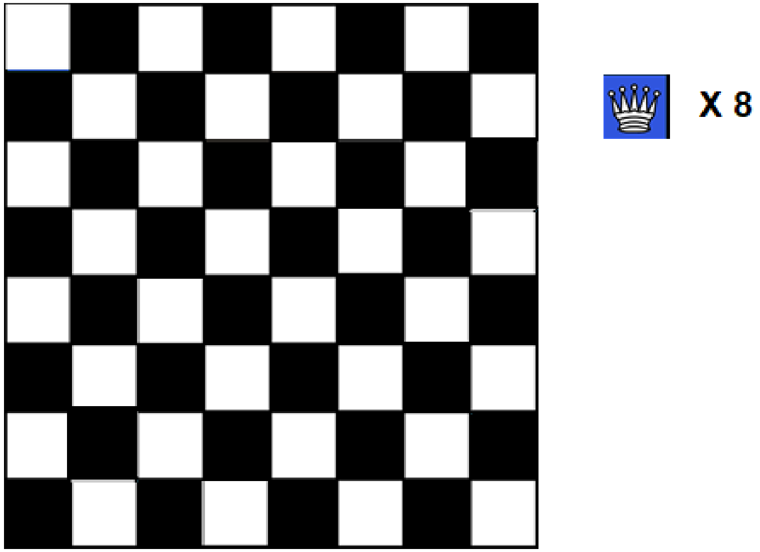
Problem Formulation

- **Initial state**: the state in which the agent starts
- **States**: All states reachable from the initial state by any sequence of actions (**State space**)
- **Actions**: possible actions available to the agent. At a state s, $Actions(s)$ returns the set of actions that can be executed in state s. (**Action space**)
- **Transition model**: A description of what each action does $Results(s, a)$
- **Goal test**: determines if a given state is a goal state

Problem Formulation

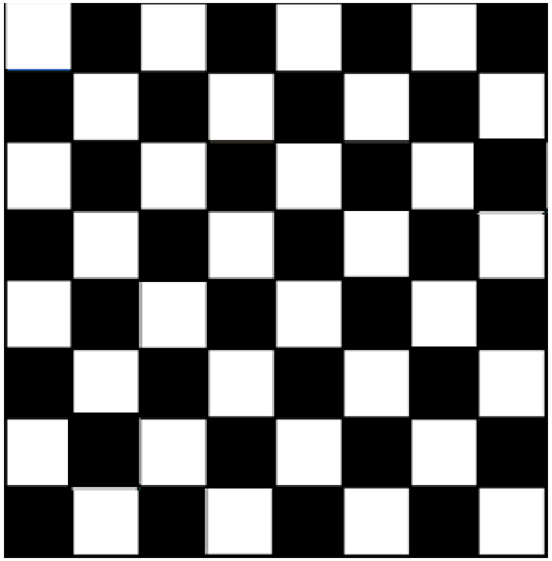
- **Initial state**: the state in which the agent starts
- **States**: All states reachable from the initial state by any sequence of actions (**State space**)
- **Actions**: possible actions available to the agent. At a state s, $Actions(s)$ returns the set of actions that can be executed in state s. (**Action space**)
- **Transition model**: A description of what each action does $Results(s, a)$
- **Goal test**: determines if a given state is a goal state
- **Path cost**: function that assigns a numeric cost to a path w.r.t. performance measure

Examples 1



- The 8-queen problem: on a chess board, place 8 queens so that no queen is attacking any other horizontally, vertically or diagonally.

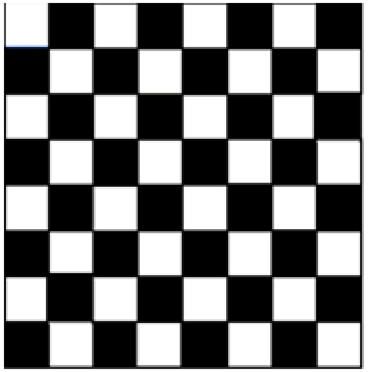
Examples 1



- Number of possible sequences to investigate:

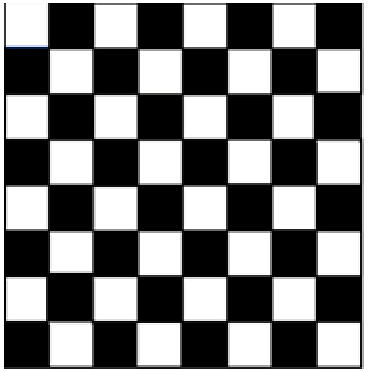
$$64 \cdot 63 \cdot 62 \cdot \cdots \cdot 57 = 1.8 \times 10^{14}$$

Examples 1



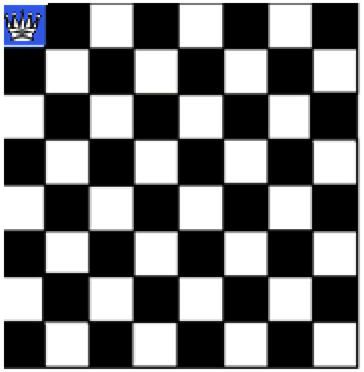
- **States**: all arrangements of 0 to 8 queens on the board.

Examples 1



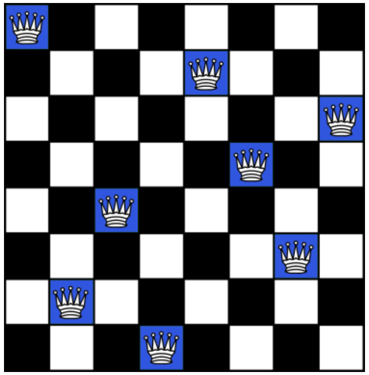
- **States**: all arrangements of 0 to 8 queens on the board.
- **Initial state**: No queen on the board

Examples 1



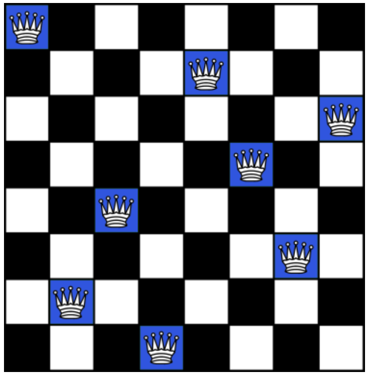
- **States**: all arrangements of 0 to 8 queens on the board.
- **Initial state**: No queen on the board
- **Actions**: Add a queen to any empty square

Examples 1



- **States**: all arrangements of 0 to 8 queens on the board.
- **Initial state**: No queen on the board
- **Actions**: Add a queen to any empty square
- **Transition model**: updated board

Examples 1



- **States**: all arrangements of 0 to 8 queens on the board.
- **Initial state**: No queen on the board
- **Actions**: Add a queen to any empty square
- **Transition model**: updated board
- **Goal test**: 8 queens on the board with none attacked

Examples 2

	0	6
7	1	2
5	3	4

8 puzzles

Examples 2

	0	6
7	1	2
5	3	4

	0	1
2	3	4
5	6	7

Examples 2

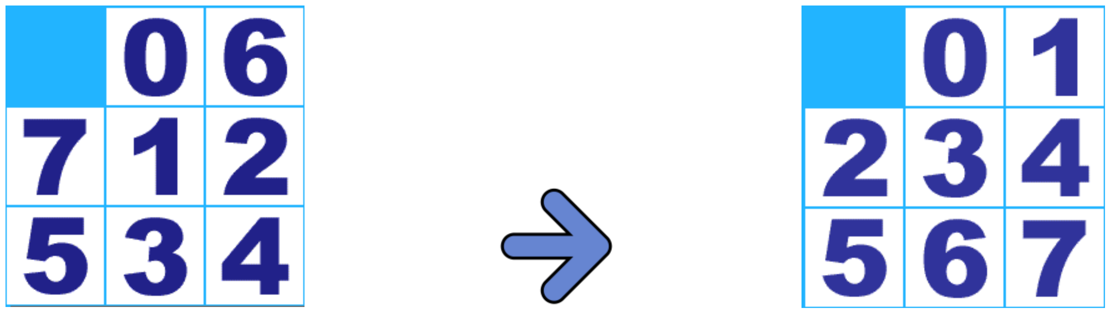
	0	6
7	1	2
5	3	4

Start State

	0	1
2	3	4
5	6	7

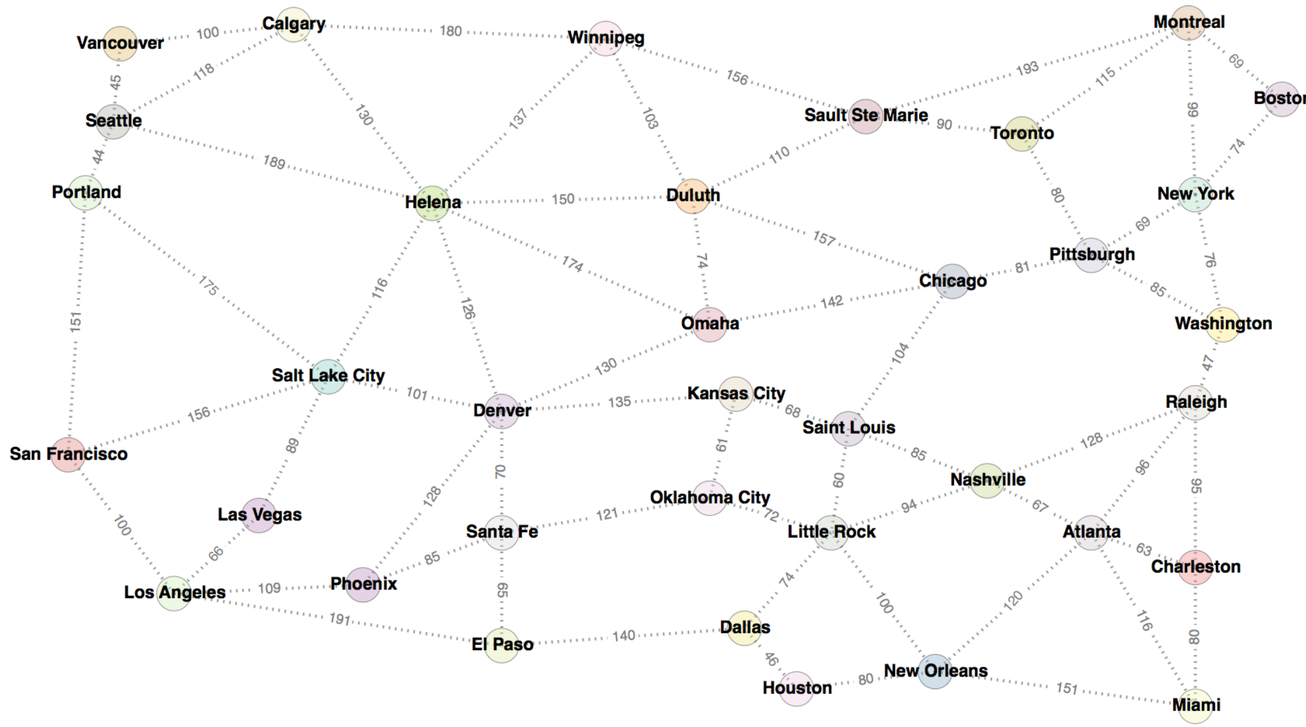
Goal State

Examples 2



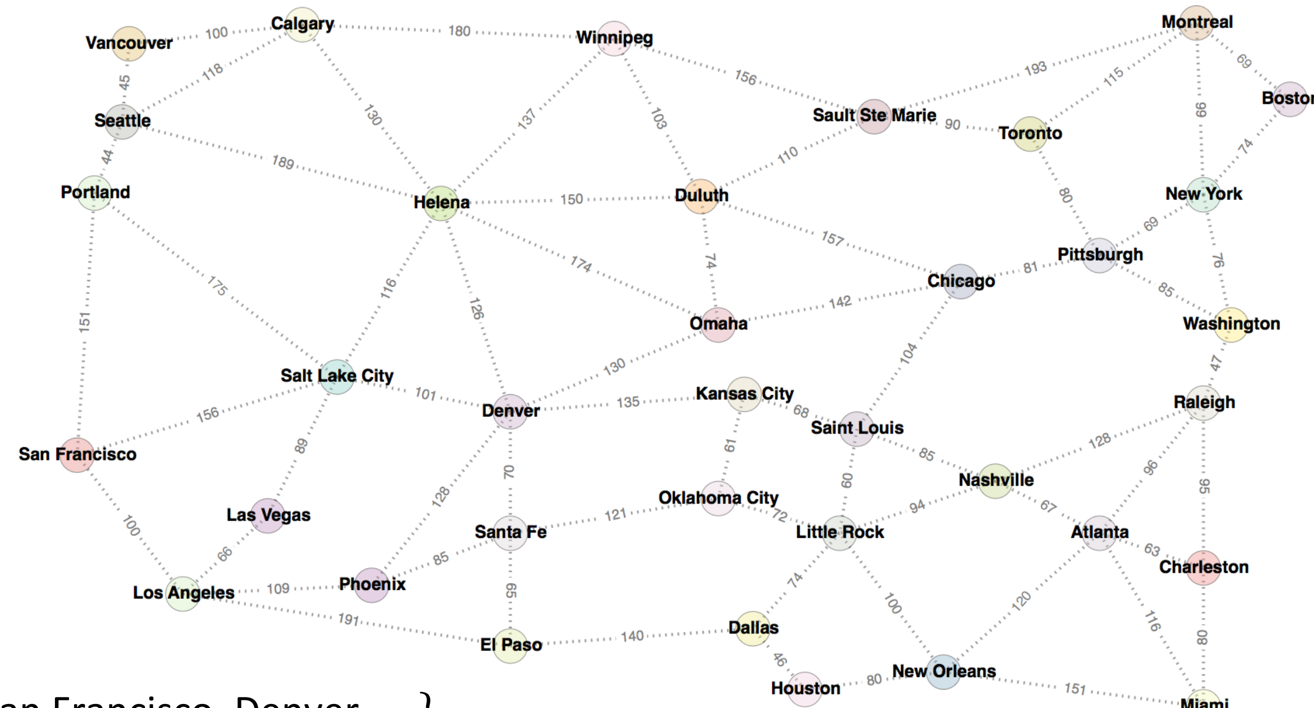
- **States**: Location of each of the 8 tiles in the 3x3 grid
- **Initial state**: Any state
- **Actions**: Move Left, Right, Up or Down
- **Transition model**: Given a state and an action, returns resulting state
- **Goal test**: state matches the goal state?
- **Path cost**: total moves, each move costs 1

Examples 3



Vancouver
Calgary
Winnipeg
Montreal
Boston
Seattle
Sault Ste Marie
Toronto
Portland
Helena
Duluth
New York
Pittsburgh
Chicago
Washington
Omaha
Salt Lake City
Denver
Kansas City
Raleigh
Saint Louis
San Francisco
Nashville
Las Vegas
Oklahoma City
Santa Fe
Little Rock
Atlanta
Charleston
Phoenix
Los Angeles
Dallas
El Paso
New Orleans
Houston
Miami
100
180
45
118
156
193
69
130
137
103
90
115
189
44
150
110
99
74
80
157
69
174
74
81
76
151
175
116
126
142
85
130
104
47
101
156
135
68
128
89
61
85
96
70
60
95
128
121
72
94
67
100
66
63
85
109
65
74
120
191
140
100
116
80
46
80
151

Examples 3



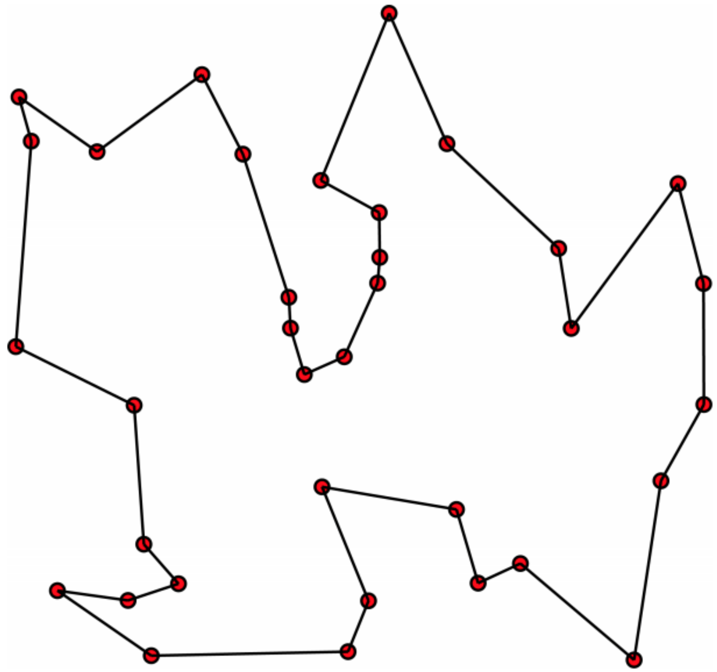
- **States**: In City where
  City ∈ {Los Angeles, San Francisco, Denver, ⋯ }
- **Initial state**: In Boston
- **Actions**: Go New York, etc.
- **Transition model**:
  Results (In (Boston), Go (New York)) = In(New York)
- **Goal test**: In(Denver)
- **Path cost**: path length in kilometers

Real-world Examples

- **Route finding problem:** typically our example of map search, where we need to go from location to location using links or transitions. Example of applications include tools for driving directions in websites, in-car systems, etc.

Real-world Examples

- **Traveling salesperson problem:** Find the shortest tour to visit each city exactly once.



Real-world Examples

- **VLSI layout:** position million of components and connections on a chip to minimize area, shorten delays. Aim: put circuit components on a chip so as they don't overlap and leave space to wiring which is a complex problem.

Real-world Examples

- **Automatic assembly sequencing:** find an order in which to assemble parts of an object which is in general a difficult and expensive geometric search.

State Space vs Search Space

State Space vs. Search Space

- **State space**: a *physical* configuration

State Space vs. Search Space

- **State space**: a *physical* configuration
- **Search space**: an *abstract* configuration represented by a search tree or graph of possible solutions

State Space vs. Search Space

- **State space**: a *physical* configuration
- **Search space**: an *abstract* configuration represented by a search tree or graph of possible solutions
- **Search tree**: **models the sequence of actions**
  - Root: initial state
  - Branches: actions
  - Nodes: results from actions. A node has: parent, children, depth, path cost, associated state in the state space

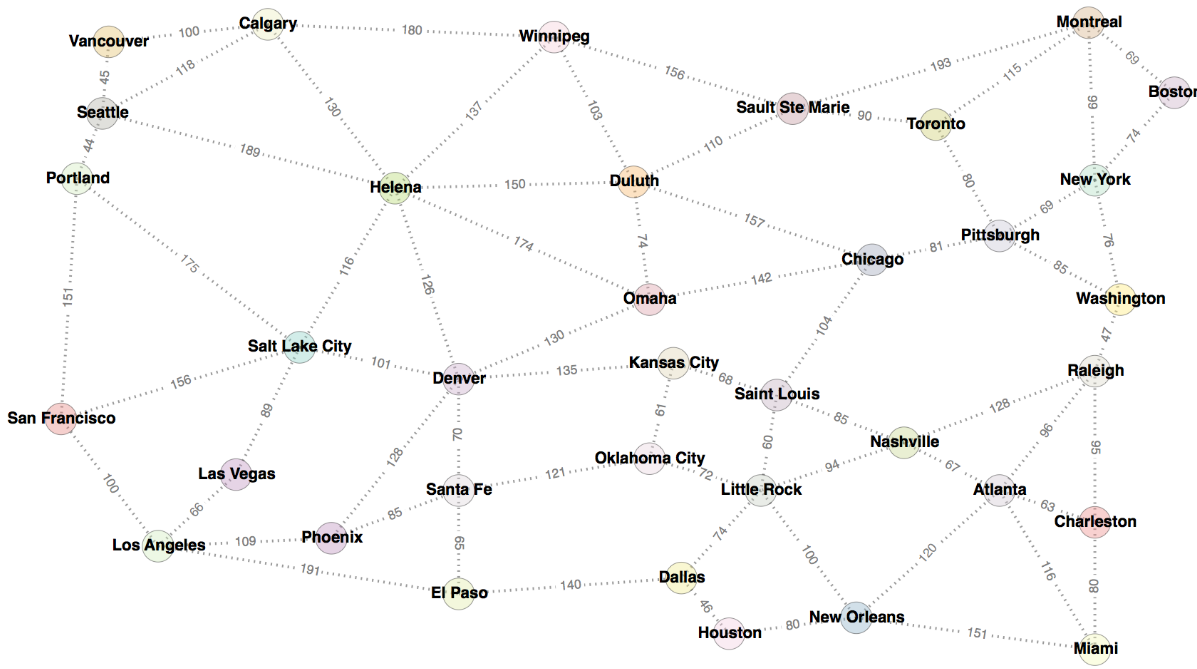
State Space vs. Search Space

- **State space**: a *physical* configuration
- **Search space**: an *abstract* configuration represented by a search tree or graph of possible solutions
- **Search tree**: **models the sequence of actions**
  - Root: initial state
  - Branches: actions
  - Nodes: results from actions. A node has: parent, children, depth, path cost, associated state in the state space
- **Expand**: A function that given a node, creates all children nodes

Search Space Regions

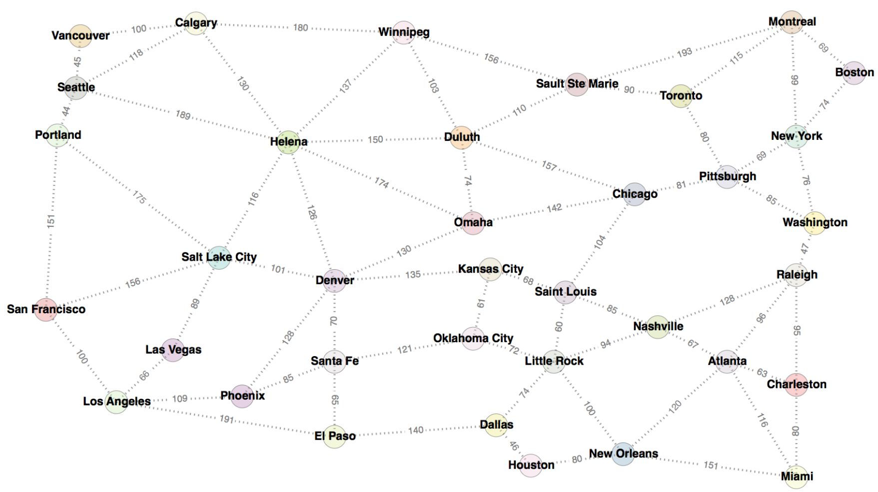
- The search space is divided into three regions:
  1. **Explored** (a.k.a. Closed List, Visited Set)
  2. **Frontier** (a.k.a. Open List, the Fringe)
  3. **Unexplored**
- The essence of search is moving nodes from regions (3) to (2) to (1), and the essence of search strategy is deciding the order of such moves.
- In the following we adopt the following color coding: orange nodes are explored, grey nodes are the frontier, white nodes are unexplored, and black nodes are failures.

Examples of Search Agents

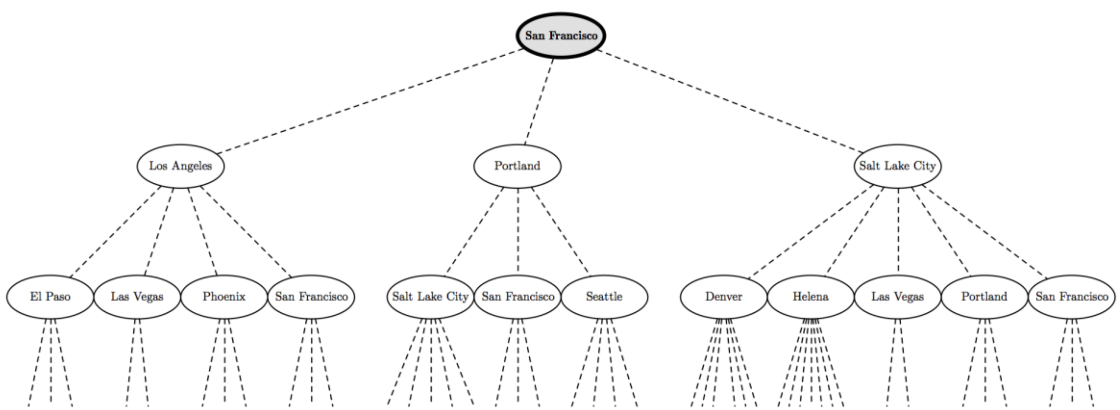


Let's show the first steps in growing the search tree to find a route from San Francisco to another city

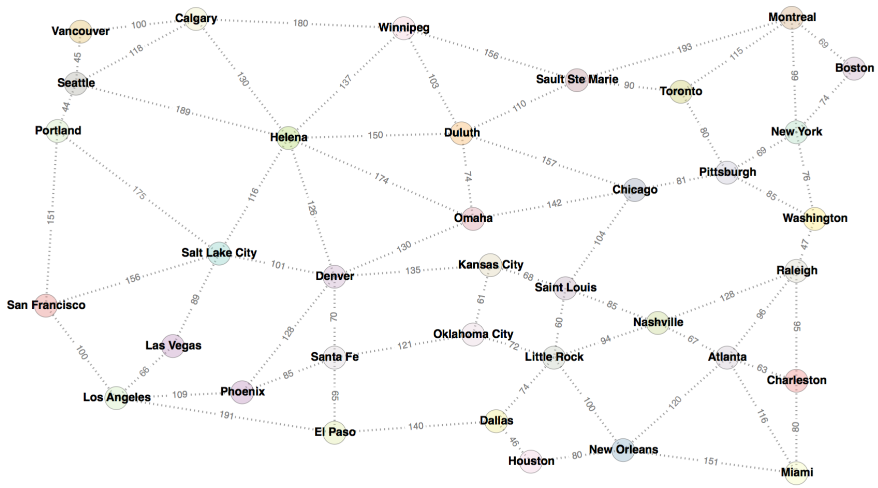
Examples of Search Agents



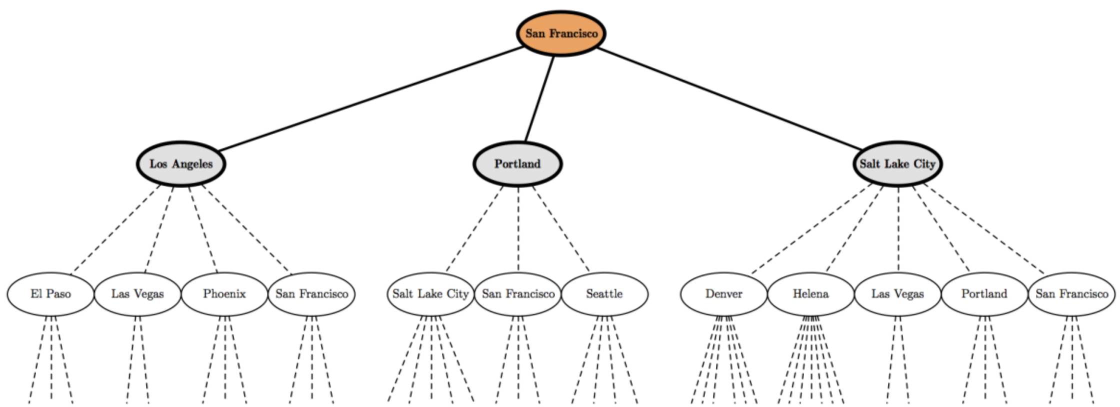

```
function TREE-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    initialize frontier with initialState

    while not frontier.isEmpty():
        state = frontier.remove()

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            frontier.add(neighbor)

    return FAILURE
```



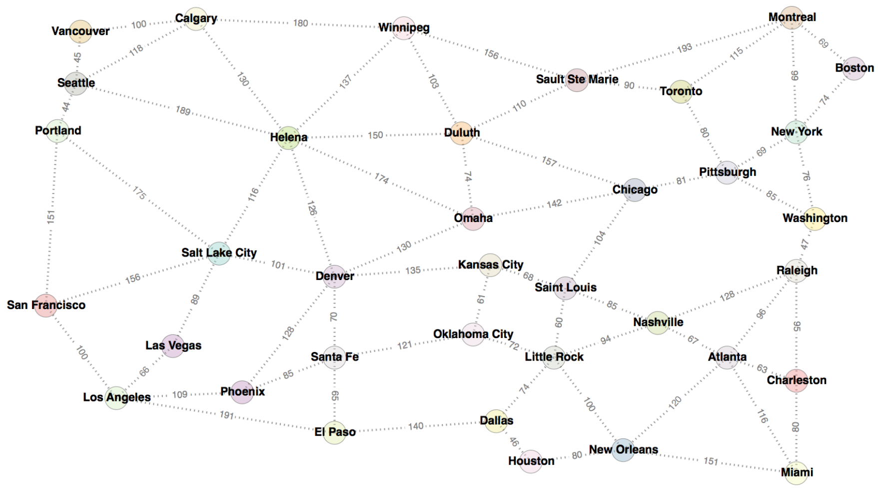
Examples of Search Agents




```
function TREE-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    initialize frontier with initialState

    while not frontier.isEmpty():
        state = frontier.remove()

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            frontier.add(neighbor)

    return FAILURE
```



Examples of Search Agents



function TREE-SEARCH(initialState, goalTest)
    returns **SUCCESS** or **FAILURE** :

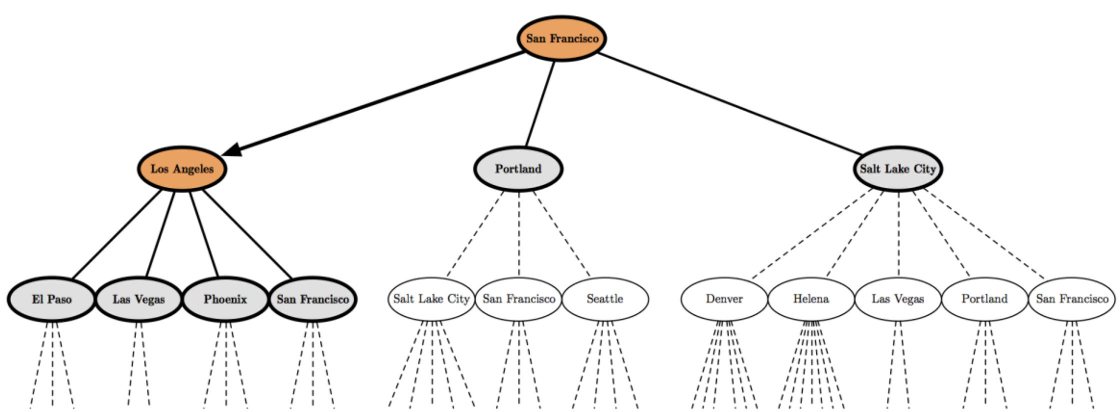
  initialize frontier **with** initialState

  while not frontier.isEmpty():
    state = frontier.remove()

    if goalTest(state):
      return **SUCCESS**(state)

    for neighbor **in** state.neighbors():
      frontier.add(neighbor)

  return **FAILURE**



Graph Search

How to handle repeated states?

Graph Search

How to handle repeated states?

function GRAPH-SEARCH(initialState, goalTest)
    returns **SUCCESS** or **FAILURE :**

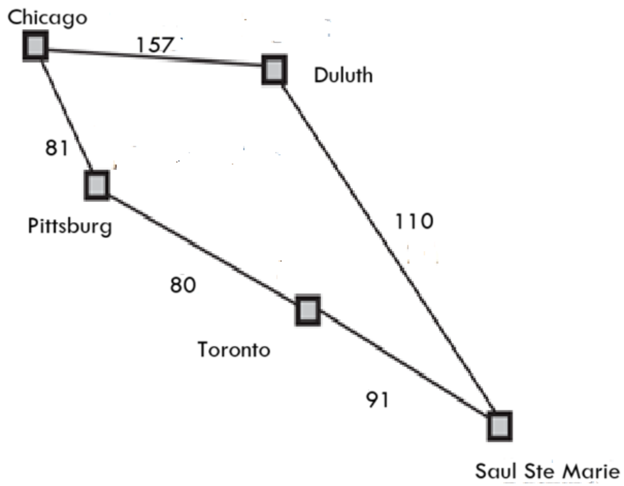
    initialize frontier **with** initialState
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.remove()
        explored.add(state)

        if goalTest(state):
            return **SUCCESS**(state)

        for neighbor **in** state.neighbors():
            if neighbor **not in** frontier ∪ explored:
                frontier.add(neighbor)

    return **FAILURE**



Search Strategies

- A strategy is defined by picking the **order of node expansion**

Search Strategies

- A strategy is defined by picking the **order of node expansion**
- Strategies are evaluated along the following dimensions:
  - **Completeness**
    Does it always find a solution if one exists?
  - **Time complexity**
    Number of nodes generated/expanded
  - **Space complexity**
    Maximum number of nodes in memory
  - **Optimality**
    Does it always find a least-cost solution?

Search Strategies

- Time and space complexity are measured in terms of:
  - *b*: maximum branching factor of the search tree (actions per state).
  - *d*: depth of the solution
  - *m*: maximum depth of the state space (may be ∞) (also noted sometimes D).
- Two kinds of search: Uninformed and Informed.

Resources

- S. J. Russell and P. Borvig, *Artificial Intelligence: A Modern Approach (3rd Edition)*, Prentice Hall International, 2010.