Solving Problems by Searching

10S3001 – Artificial Intelligence

Samuel I. G. Situmeang

Faculty of Informatics and Electrical Engineering

Objectives

Students are able:

- to describe the main idea of problem-solving as a search,
- to mention some real-world examples of problem-solving as a search,
- to formulate a problem as a search problem,
- to contrast state space and search space, and
- to explain the application tree search and graph search for intelligent agents based on searching (search agents).

Siswa mampu:

- untuk menjabarkan ide pokok penyelesaian persoalan sebagai pencarian,
- untuk menyebutkan beberapa contoh penyelesaian persoalan dunia nyata sebagai pencarian,
- untuk merumuskan persoalan sebagai persoalan pencarian,
- untuk membedakan *state space* dan *search space*, dan
- untuk menjelaskan penerapan *tree search* dan *graph search* untuk agen cerdas berdasarkan pencarian (*search agent*).

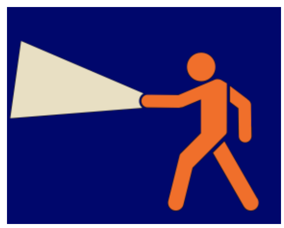
Reflex Agents and Goal-based Agents

- **Reflex agents**: use a mapping from states to actions.

- **Goal-based agents**: problem solving agents or planning agents.

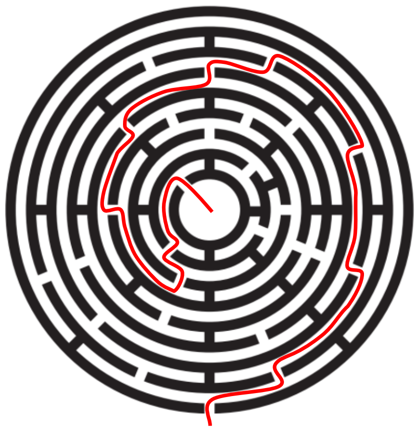
Goal-based Agents

- 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**.

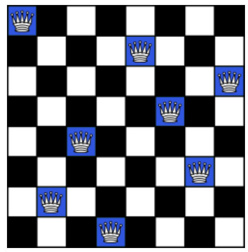


Examples

EXPLORE!



Examples



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

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

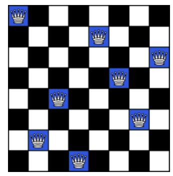
Problem Solving as Search

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

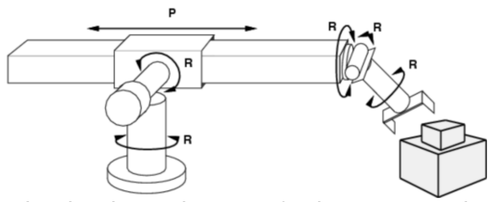
- **States**: All states reachable from the initial state by any sequence of actions (**State space**).
- **Initial state**: the state in which the agent starts.
- **Actions**: possible actions available to the agent. At a state s, $Actions(s)$ returns the set of actions that can be executed in state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

Example: 8-Queens



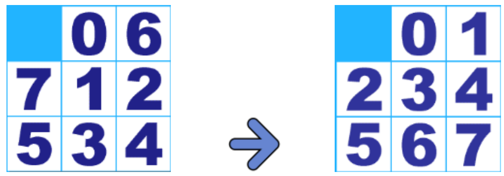
- **States**: all arrangements of 0 to 8 queens on the board.
- **Initial state**: no queen on the board.
- **Actions**: add a queen to any empty square.
- **Transition model**: updated board.
- **Goal test**: 8 queens on the board with none attacked.
- **Path cost**: 1 per move.

Example: Robotic Assembly



- **States**: real-valued coordinates of robot joint angles parts of the object to be assembled.
- **Initial state**: rest configuration.
- **Actions**: continuous motions of robot joints.
- **Transition model**: states of robot joints after each action.
- **Goal test**: complete assembly.
- **Path cost**: time to execute+energy used.

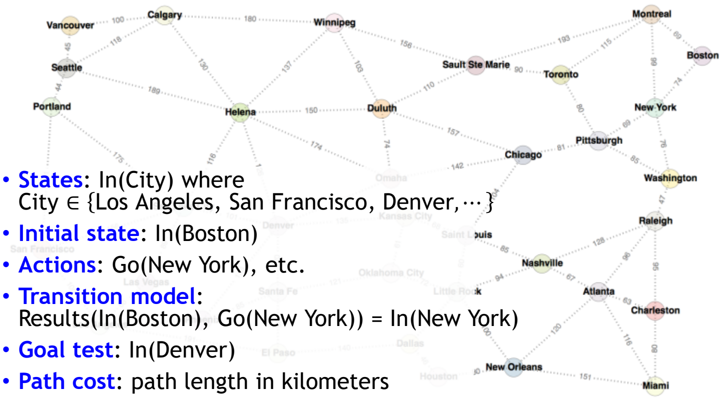
Example: 8 Puzzles



Start State **Goal State**

- **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.

Example: USA Touring



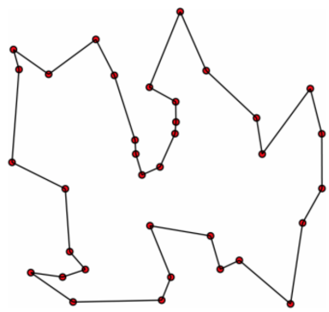
- **States**: In(City) where City ∈ {Los Angeles, San Francisco, Denver, ⋯}
- **Initial state**: In(Boston)
- **Actions**: Go(New York), etc.
- **Transition model**: Results(In(Boston), Go(New York)) = In(New York)
- **Goal test**: In(Denver)
- **Path cost**: path length in kilometers

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.

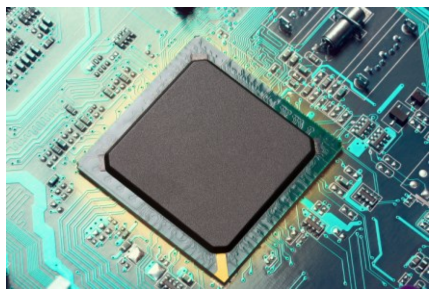
Real-world Examples

- **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.



Very-large-scale integration (VLSI)

Real-world Examples

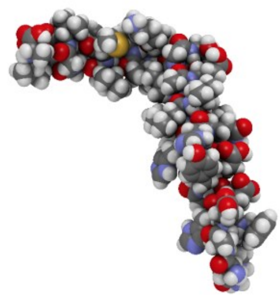
- **Robot navigation:** special case of route finding for robots with no specific routes or connections. The robot navigates in 2D or 3D space or ore where the state space and action space are potentially infinite.

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.

Real-world Examples

- **Protein design:** find a sequence of amino acids that will fold into a 3D protein with the right properties to cure some disease.



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

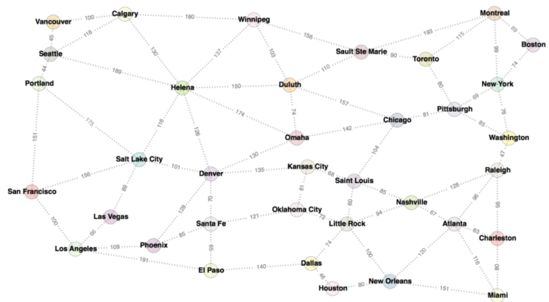
Search Space Regions

- 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.

Tree Search

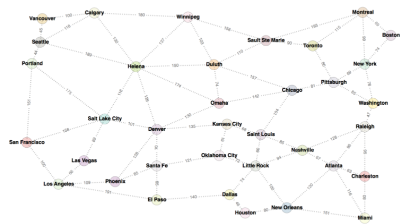
```
function TREE-SEARCH(initialState, goalTest)
      returns SUCCESS or FAILURE :

      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



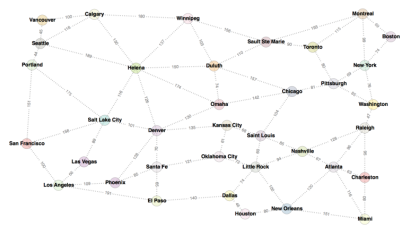
Let's show the first steps in growing the search tree to find a route from San Francisco to another city

Examples of Search Agents



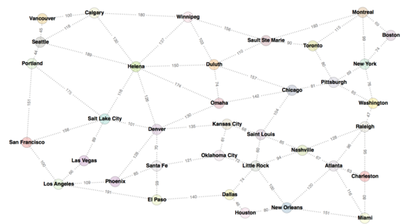

```
function TREE-SEARCH(initialState, goalTest)
  returns SUCCESS or FAILURE :

  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

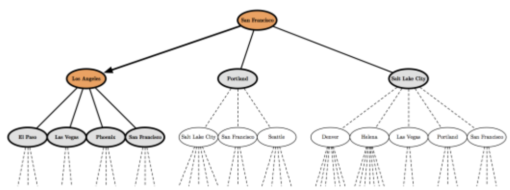



```
function TREE-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    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**
- 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.

References

- S. J. Russell and P. Borvig, *Artificial Intelligence: A Modern Approach (4th Edition)*, Prentice Hall International, 2020.

eof