Problem solving by searching

A search problem

Many interesting problems in science and engineering are solved using search.

A search problem is defined by:

• A search space:
  – The set of objects among which we search for the solution.
    Examples: routes between cities, or n-queens configuration.

• A goal condition:
  – Characteristics of the object we want to find in the search space?
    Examples:
    • Path between cities A and B
    • Non-attacking n-queen configuration
Search

- **Search (process)**
  - The process of exploration of the search space
- **Important**
  - We can often influence the efficiency of the search !!!!
  - We can be smart about choosing the search space, the exploration policy, and the design of the goal test

Graph representation of a search problem

- Search problems can be often represented using graphs
- **Typical example: Route finding**
  - Map corresponds to the graph, nodes to cities, links valid moves via available connections
  - **Goal:** find a route (sequence of moves) in the graph from S to T
Graph search

• **Less obvious conversion:**

  **Puzzle 8.** Find a sequence of moves from the initial configuration to the goal configuration.
  – nodes corresponds to states of the game,
  – links to valid moves made by the player

Graph Search Problems

Search problems can be often represented as graph search problems:

• **Initial state**
  – State (configuration) we start to search from (e.g. start city, initial game position)

• **Operators:**
  – Transform one state to another (e.g. valid connections between cities, valid moves in Puzzle 8)

• **Goal condition:**
  – Defines the target state (destination, winning position)

*Search space* is now defined indirectly through:

The initial state + Operators
N-queens

Some problems are easy to convert to the graph search problems

• **But some problems are harder and less intuitive**
  – Take e.g. N-queens problem.

Goal configuration

• **Problem:**
  – We look for a configuration, not a sequence of moves
  – No distinguished initial state, no operators (moves)

How to choose the search space for N-queens?

• Ideas? **Search space:**
  – all configurations of N queens on the board

• **Can we convert it to a graph search problem?**
• We need states, operators, initial state and goal condition.
N-queens

Search space:
– all configurations of N queens on the board

• Can we convert it to a graph search problem?
• We need states, operators, initial state and goal state.

States are: N-queen configurations
Initial state: ?
Operators (moves)?

N-queens

Search space:
– all configurations of N queens on the board

• Can we convert it to a graph search problem?
• We need states, operators, initial state and goal condition.

Initial state: an arbitrary N-queen configuration
Operators (moves): change a position of one queen
N-queens

Is there an alternative way to formulate the N-queens problem as a search problem?

- **Search space:** configurations of 0, 1, 2, … N queens
- **Graph search:**
  - States: configurations of 0, 1, 2, … N queens
  - Operators: additions of a queen to the board
  - Initial state: no queens on the board

Graph search

**N-queens problems**

- This is a different graph search problem when compared to Puzzle 8 or Route planning:
  
  We want to find only the target configuration, not a path
Two types of graph search problems

• **Path search**
  – Find a path between states S and T
  – **Example:** traveler problem, Puzzle 8
  – **Additional goal criterion:** minimum length (cost) path

• **Configuration search (constraint satisfaction search)**
  – Find a state (configuration) satisfying the goal condition
  – **Example:** n-queens problem
  – **Additional goal criterion:** “soft” preferences for configurations, e.g. minimum cost design

Graph Search Problem

Search problems that can be often represented or converted into a graph search problems:

• **Initial state**
  – State (configuration) we start to search from (e.g. start city, initial game position)

• **Operators:**
  – Transform one state to another (e.g. valid connections between cities, valid moves in Puzzle 8)

• **Goal condition:**
  – Defines the target state (destination, winning position)

• **Search space** (the set of objects we search for the solution):
  – is now defined indirectly through:
    the *initial state* + *operators*
Traveler problem formulation:
- **States**: different cities
- **Initial state**: city Arad
- **Operators**: moves to cities in the neighborhood
- **Goal condition**: city Bucharest
- **Type of the problem**: path search
- **Possible solution cost**: path length

Puzzle 8 example

- **States**: tile configurations
- **Initial state**: initial configuration
- **Operators**: moves of the empty tile
- **Goal**: reach the winning configuration
- **Type of the problem**: path search
- **Possible solution cost**: a number of moves
N-queens problem

Problem formulation:
• States: configurations of 0 to 4 queens on the board
• Initial state: no-queen configuration
• Operators: add a queen to the leftmost unoccupied column
• Goal: a configuration with 4 non-attacking queens
• Type of the problem: configuration search

N-queens problem

Alternative formulation of N-queens problem

Problem formulation:
• States: different configurations of 4 queens on the board
• Initial state: an arbitrary configuration of 4 queens
• Operators: move one queen to a different unoccupied position
• Goal: a configuration with non-attacking queens
• Type of the problem: configuration search
Search

- **Search (process)**
  - The process of exploration of the search space
- **The efficiency of the search depends on:**
  - The search space and its size
  - Method used to explore (traverse) the search space
  - Condition to test the satisfaction of the search objective
    (what it takes to determine I found the desired goal object

Comparison of two problem formulations

**Solution 1:**

Operators: switch one of the queens

\[ \binom{16}{4} \] - all configurations

**Solution 2:**

Operators: add a queen to the leftmost unoccupied column

\[ 1 + 4 + 4^2 + 4^3 + 4^4 < 4^5 \] - configurations altogether
Even better solution to the N-queens

Solution 2:

Operators: add a queen to the leftmost unoccupied column

< 4^5 - configurations altogether

Improved solution with a smaller search space

Operators: add a queen to the leftmost unoccupied column such that it does not attack already placed queens

\[ \leq 1 + 4 + 4 \times 3 + 4 \times 3 \times 2 + 4 \times 3 \times 2 \times 1 = 65 \]

- configurations altogether

Formulating a search problem

- **Search (process)**
  - The process of exploration of the search space

- **The efficiency of the search depends on:**
  - The search space and its size
  - Method used to explore (traverse) the search space
  - Condition to test the satisfaction of the search objective
    (what it takes to determine I found the desired goal object)

- **Think twice before solving the problem by search:**
  - Choose the search space and the exploration policy
Improving efficiency of search

- **Next:**
  - Methods used to explore (traverse) the search space

Search process

**Exploration of the state space** through successive application of operators from the initial state
- **Search tree = structure representing the exploration trace**
- Built on-line during the search process
- Branches correspond to explored paths, and leaf nodes to the exploration fringe
Search tree

- A search tree = (search) exploration trace
  - different from the graph representation of the problem
  - states can repeat in the search tree

A branch in the search tree = path in the graph
A branch in the search tree = path in the graph

General search algorithm

**General-search** \((\text{problem, strategy})\)

- **initialize** the search tree with the initial state of \text{problem}
- **loop**
  - if there are no candidate states to explore **return** failure
  - **choose** a leaf node of the tree to expand next according to \text{strategy}
  - if the node satisfies the goal condition **return** the solution
  - **expand** the node and add all of its successors to the tree

**end loop**
General search algorithm

**General-search** \((\text{problem}, \text{strategy})\)

**initialize** the search tree with the initial state of \text{problem}

**loop**
- if there are no candidate states to explore next return \text{failure}
- choose a leaf node of the tree to expand next according to \text{strategy}
  - if the node satisfies the goal condition return \text{the solution}
  - expand the node and add all of its successors to the tree

end loop
General search algorithm

General-search (problem, strategy)
initialize the search tree with the initial state of problem
loop
  if there are no candidate states to explore next return failure
  choose a leaf node of the tree to expand next according to strategy
  if the node satisfies the goal condition return the solution
  expand the node and add all of its successors to the tree
end loop
General search algorithm

**General-search** *(problem, strategy)*

initialize the search tree with the initial state of *problem*

loop
  - if there are no candidate states to explore next return failure
  - choose a leaf node of the tree to expand next according to *strategy*
    - if the node satisfies the goal condition return the solution
    - expand the node and add all of its successors to the tree
  end loop

• **Search methods** differ in how they explore the space, that is how they choose the node to expand next !!!!!
Implementation of search

• Search methods can be implemented using the **queue** structure

```
General search (problem, Queuing-fn)

nodes ← Make-queue(Make-node(Initial-state(problem)))

loop
    if nodes is empty then return failure
    node ← Remove-node(nodes)
    if Goal-test(problem) applied to State(node) is satisfied then return node
    nodes ← Queuing-fn(nodes, Expand(node, Operators(node)))
end loop
```

• Candidates are added to `nodes` representing the queue structure

---

Implementation of search

• A **search tree node** is a data-structure that is a part of the search tree

- State
  - 5 4
  - 6 1
  - 7 3

- Other attributes:
  - state value (cost)
  - depth
  - path cost

- Expand function – applies Operators to the state represented by the search tree `node`. Together with Queuing-fn it fills the attributes.
Uninformed search methods

• Search techniques that rely only on the information available in the problem definition
  – Breadth first search
  – Depth first search
  – Iterative deepening
  – Bi-directional search

For the minimum cost path problem:
  – Uniform cost search

Search methods

Properties of search methods :
• Completeness.
  – Does the method find the solution if it exists?

• Optimality.
  – Is the solution returned by the algorithm optimal? Does it give a minimum length path?

• Space and time complexity.
  – How much time it takes to find the solution?
  – How much memory is needed to do this?
Parameters to measure complexities.

- **Space and time complexity.**
  - Complexity is measured in terms of the following tree parameters:
    - $b$ – maximum branching factor
    - $d$ – depth of the optimal solution
    - $m$ – maximum depth of the state space

**Branching factor**

The number of applicable operators

---

**Breadth first search (BFS)**

- The shallowest node is expanded first
Breadth-first search

- **Expand the shallowest node first**
- Implementation: put successors to the end of the queue (FIFO)
Breadth-first search

queue

Sibiu
Timisoara
Arad
Oradea

Arad
Zerind
Sibiu
Timisoara

queue

Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea

queue

Oradea
Fagaras
Rimnicu Vilcea

Oradea
Lugoj
Breadth-first search

Properties of breadth-first search

• Completeness: ?

• Optimality: ?

• Time complexity: ?

• Memory (space) complexity: ?

  – For complexity use:
    • $b$ – maximum branching factor
    • $d$ – depth of the optimal solution
    • $m$ – maximum depth of the search tree
Properties of breadth-first search

- **Completeness**: Yes. The solution is reached if it exists.

- **Optimality**: Yes, for the shortest path.

- **Time complexity**: ?

- **Memory (space) complexity**: ?

---

BFS – time complexity

<table>
<thead>
<tr>
<th>depth</th>
<th>number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$2^1=2$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2=4$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3=8$</td>
</tr>
</tbody>
</table>

For any depth $d$, total nodes: $2^d \cdot b^d$,

Total nodes: ?
### BFS – time complexity

- **Depth**  
  - 0: 1 node  
  - 1: \(2^1 = 2\) nodes  
  - 2: \(2^2 = 4\) nodes  
  - 3: \(2^3 = 8\) nodes  
  
- **Expanded nodes**: \(O(b^d)\)  
- **Total nodes**: \(O(b^{d+1})\)

### BFS – memory complexity

- **Count nodes kept in the tree structure or in the queue**
- **Depth**  
  - 0: 1 node  
  - 1: \(2^1 = 2\) nodes  
  - 2: \(2^2 = 4\) nodes  
  - 3: \(2^3 = 8\) nodes  
  
- **Expanded nodes**: \(O(b^d)\)  
- **Total nodes**: \(O(b^{d+1})\)
Properties of breadth-first search

- **Completeness**: Yes. The solution is reached if it exists.

- **Optimality**: Yes, for the shortest path.

- **Time complexity**: 
  \[1 + b + b^2 + \ldots + b^d = O(b^d)\]
  exponential in the depth of the solution \(d\)

- **Memory (space) complexity**: 
  \[O(b^d)\]
  nodes are kept in the memory