Problem solving by searching

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

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
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 representation of a search problem](image)

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](image)
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.

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

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.

Initial state: an arbitrary N-queen configuration

Operators (moves): change a position of one queen
**N-queens**

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

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

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
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)

Think twice before solving the problem:
Choose the search space wisely.
Search process

Exploration of the state space through successive application of operators from the initial state

- Search tree = structure representing the exploration trace
  - Is 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 = a path in the graph
**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 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 \((\text{problem, strategy})\)

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

Arad \(\leftarrow\) Node chosen to be expanded next

Arad \(\leftarrow\) Check if the node satisfied the goal
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
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*)
initialization: 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 a 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-fin*)

\[ \text{nodes} \leftarrow \text{Make-queue} \left( \text{Make-node(Initial-state(problem))} \right) \]

loop
  if nodes is empty then return failure
  \[ \text{node} \leftarrow \text{Remove-node(nodes)} \]
  if Goal-test(*problem*) applied to State(*node*) is satisfied then return node
  \[ \text{nodes} \leftarrow \text{Queuing-fin(nodes, Expand(node, Operators(node)))} \]
end loop

- Candidates are added to the queue structure (named nodes)
- Queuing function determines what node will be selected next
Implementation of search

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

  ![State](state)

  **ST Node**

  **State**

  - **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.

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).
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

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

Arad
Zerind

Sibiu
Timisoara

Oradea

Arad
Zerind

Sibiu
Timisoara

Oradea

Fagaras

Rimnicu Vilcea

Arad
Lugoj

queue

Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea

Arad
Lugoj
Breadth-first search

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>
<th>depth</th>
<th>number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2^1 = 2</td>
</tr>
<tr>
<td>2</td>
<td>2^2 = 4</td>
<td>3</td>
<td>2^3 = 8</td>
</tr>
<tr>
<td>d</td>
<td>2^d (b^d)</td>
<td>d+1</td>
<td>2^{d+1} (b^{d+1})</td>
</tr>
</tbody>
</table>

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:**?