Problem solving by searching

Milos Hauskrecht
milos@pitt.edu
5329 Sennott Square

Example

• Assume a problem of solving a linear equation

  \[ 3x + 2 = 11 \]

  Do you consider it a challenging problem?
Example

• Assume a problem of computing the roots of the quadratic equation

\[ 3x + 2 = 11 \]

Do you consider it a challenging problem?  
Hardly, we just apply the ‘standard’ formula or procedure to solve:

\[ ax + b = c \]
\[ x = (c - b) / a \]
\[ x = 3 \]

Solving problems by searching

• Some problems have a straightforward solution  
  – Just apply a known formula, or follow a standardized procedure  
    **Example:** solution of the linear or quadratic equations  
  – Hardly a sign of intelligence

• More interesting problems do not have a straightforward solution, and they require **search:**  
  – more than one possible alternative needs to be explored before the problem is solved  
  – the number of alternatives to search among can be very large, even infinite
Search example: Route finding

• Find a route (path) from one city to another city

Example. Traveler problem

• Another flavor of the traveler problem:
  – find the route with the minimum length between S and T
Example. Puzzle 8.

- Find the sequence of move of tiles from the initial game position to the designated target position

<table>
<thead>
<tr>
<th>Initial position</th>
<th>Goal position</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 5 1 8</td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>6 1 3 2</td>
<td>7 8</td>
</tr>
</tbody>
</table>


Find a configuration of n queens on an n x n board such that queens do not attack each other

A goal configuration

A bad configuration
A search problem

is defined by:

• A search space:
  – The set of objects among which we search for the solution
  – Example: routes connecting two cities, or the different N-queen configurations

• A goal condition
  – What are the characteristics of the object we want to find in the search space?
  – Examples:
    • Path between cities A and B
    • Path between A and B with the smallest number of links
    • Path between A and B with the shortest distance
    • Non-attacking n-queen configuration

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)
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)
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: Path finding**
  - Map corresponds to the graph, nodes to cities, links valid moves via available connections
  - **Goal**: find a path (sequence of moves) in the graph from S to T
Graph search problem

Four components:

- **States** - game positions, or locations on the map that are represented by nodes in the graph
- **Operators** - valid moves
- **Initial state** – start position, start city
- **Goal state** – target position (positions), target city (cities)

![Graph search problem diagram](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 diagram](image)
More complex versions of the graph search problems:
- Find the minimal length path
  (= a route with the smallest number of connections, the shortest sequence of moves that solves Puzzle 8)

- Find the minimum cost path
  (= a path with the shortest distance)
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 condition.

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?

• Ideas?

• **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 Path 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.

**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**

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Initial state

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Goal state

**Search problem formulation:**
- **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

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

7 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

• **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)

---

Search

• **Search (process)**
  – The process of exploration of the search space

• **The efficiency of the search depends on:**
  – 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**
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 diagram](image)

Search tree

- A search tree = (search) exploration trace
  - [**different from the graph representation of the problem**](image)
  - States can repeat in the search tree

![Search tree diagram](image)
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 \text{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 \((\text{problem}, \text{strategy})\)
initialize the search tree with the initial state of \text{problem}
loop
  if there are no candidate states to explore next return 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

Arad
\arrow{Expanded node}
\arrow{Zerind}
\arrow{Sibiu}
\arrow{Timisoara}
\arrow{Generated (or active, or open) nodes}

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 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

\arrow{Newly selected node
Arad
\arrow{Zerind}
\arrow{Sibiu}
\arrow{Timisoara}
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

Check if it is the goal

Expanded nodes

Generated (active, open) nodes
**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 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-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 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

- **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