Problem solving by searching

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

Solving problems by searching

• Some problems have a straightforward solution
  – Just apply a known formula, or a standardized procedure

  Example: solution of the quadratic equation
  \[ x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

• More interesting problems 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: Traveler problem

- Find a route from one city (Arad) to the other (Bucharest)

Example. Puzzle 8.

- Find the sequence of the empty tile moves 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</td>
<td>1 2 3</td>
</tr>
<tr>
<td>6 1 8</td>
<td>4 5 6</td>
</tr>
<tr>
<td>7 3 2</td>
<td>7 8</td>
</tr>
</tbody>
</table>

Find a configuration of n queens not attacking 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: objects = routes between cities, or 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)

- **Important to remember !!!**
  - You can choose the search space and the exploration policy
  - These choices can have a profound effect on the efficiency of the solution

Graph search

- **Many search problems can be naturally represented as graph search problems**
- **Typical example: Route finding**
  - Map corresponds to the graph, nodes to cities, links to available connections between cities
  - **Goal:** find a route (path) in the graph from S to T
Graph search problem

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

![Graph search diagram]

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
- **Note**: the graph for some problem can become very large,
N-queens: Solution 1

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

• Graph search:
  - States: configurations of N queens
  - Operators: change the position of one of the queens
  - Initial state: an arbitrary configuration
  - Goal: non-attacking queens

N-queens: Solution 2

• 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: 0 queens on the board
  - Goal: non-attacking queens
Two types of search problems

- **Path search**
  - Find a path (trajectory) 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, design of a device with a predefined functionality
  - **Additional goal criterion:** “soft” preferences for configurations, e.g. minimum cost design

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

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

N-queens problem: version 1

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
N-queens problem: version 2

Alternative formulation of 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

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

Solution 3:

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

$\leq 1 + 4 + 4 * 3 + 4 * 3 * 2 + 4 * 3 * 2 * 1 = 65$

- 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
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*

---

Search process

- Exploration of the state space through successive application of operators from the initial state
- A *search tree* = a kind of (search) exploration trace, branches corresponding to explored paths, and leaf nodes corresponding to the exploration fringe
A search tree = a (search) exploration trace

- It is different from the graph defining the problem
- States can repeat in the search tree
General search algorithm

General-search \((\text{problem}, \text{strategy})\)
initialize the search tree with the initial state of \text{problem}
loop
\hspace{1em} \text{if there are no candidate states to explore} \quad \text{return} \quad \text{failure}
\hspace{1em} \text{choose} \quad \text{a leaf node of the tree to expand next according to} \quad \text{strategy}
\hspace{1em} \text{if the node satisfies the goal condition} \quad \text{return} \quad \text{the solution}
\hspace{1em} \text{expand} \quad \text{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)*
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 queue structure

**General search** \((\text{problem}, \text{Queuing-fn})\)
\[
\begin{align*}
\text{nodes} & \leftarrow \text{Make-queue}(\text{Make-node}(\text{Initial-state}(\text{problem}))) \\
\text{loop} & \\
& \quad \text{if nodes is empty then return failure} \\
& \quad \text{node} \leftarrow \text{Remove-node}(\text{nodes}) \\
& \quad \text{if Goal-test(\text{problem}) applied to State(\text{node}) is satisfied then return node} \\
& \quad \text{nodes} \leftarrow \text{Queuing-fn}(\text{nodes}, \text{Expand}(\text{node}, \text{Operators}(\text{node}))) \\
\text{end loop}
\end{align*}
\]

- Candidates are added to nodes representing the queue structure

---

Implementation of search

- A **search tree node** is a data-structure constituting part of a search tree

![Search Tree Node Diagram]

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

**Other attributes:**
- state value (cost)
- depth
- path cost
Uninformed search methods

- 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.**
  - Complexities are measured in terms of parameters:
    - $b$ – maximum branching factor
    - $d$ – depth of the optimal solution
    - $m$ – maximum depth of the state space

**Branching factor**

![Branching factor diagram]

**Breadth first search (BFS)**

- The shallowest node is expanded first

![BFS diagram]
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
Sibiu
Timisoara

Zerind

Fagaras
Rimnicu Vilcea
Vilcea

Arad
Oradea
Fagaras

Rimnicu Vilcea

Oradea

Arad
Lugoj

Breadth-first search

queue

Timisoara
Arad
Oradea
Arad
Oradea
Fagaras
Rimnicu Vilcea

Arad
Lugoj
Breadth-first search

Properties of breadth-first search

- Completeness: ?
- Optimality: ?
- Time complexity: ?
- Memory (space) complexity: ?

- For complexities 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:** ?