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 the formula, or follow a standardized procedure
  
  Example:
  solution of the quadratic equation \( ax^2 + bx + c = 0 \)
  
  \[
  x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
  \]
  – Hardly a sign of intelligence

• 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. 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 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 6 1 8</td>
<td>1 2 3 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

Goal configuration

Bad goal configuration
A search problem

is defined by:

- **Search space:**
  - The set of objects among which we search for the solution
  - Example: objects = routes between cities, or N-queen configurations

- **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 !!!**
  - Conveniently chosen search space and exploration policy can have a profound effect on the efficiency
Graph search

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

![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 next moves

![Puzzle 8 diagram]
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

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

• More complex version of a graph search problem:
  – Find a minimum cost path
    (= a route with the shortest distance)

![Graph with nodes and edges]

• How to find the path between S and T?
• One solution:
  – Generate systematically all sequences of 1, 2, 3, … edges
  – Check if the sequence yields a path between S and T.
• Can we do better?
Graph search

Can we do better?

• We are not interested in sequences that do not start in S and that are not valid paths

• Simple solution:
  – Look only on valid paths starting from S

Graph search

• Being smarter about the space we search for the solution pays off in terms of search process efficiency.
**N-queens**

Some problems can be converted to the graph search problems

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

  ![Goal configuration](image)

- **Problem:**
  - We look for a configuration, not a sequence of moves

---

**Graph search**

One trick to make things work:

- nodes corresponds to configurations of 0,1,2,3,4 queens
- links correspond to an addition of a queen

![Diagram](image)
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
  – 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
Search problem

Search problems that can be represented or converted into a graph search problems can be defined in terms of:

- **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

**Search problem formulation:**
- **States:** tile configurations
- **Initial state:** initial configuration
- **Operators:** moves of the empty tile
- **Goal:** a 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

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

Problem formulation:

Comparison of two problem formulations

Solution 2:

Operators: switch one of the queens
$4^{12}$ - configurations can be reached in one step

Solution 1:

Operators: add a queen to the leftmost unoccupied column
$< 5^4$ - configurations altogether
**Even better solution to the N-queens**

Solution 1: 

Operators: add a queen to the leftmost unoccupied column

\[ \leq 5^4 \] - 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

\[ \leq 5 \cdot 4 \cdot 3 \cdot 2 = 120 \] - configurations altogether

---

**Search process**

- Exploration of the state space through successive application of operators from the initial state
- Can be thought of as a process of building a **search tree (a kind of exploration trace)**, with nodes corresponding to explored states

```
Arad
  /    \
Zerind  Sibiu  Timisoara
    \    /         /      
  Oradea Fagaras Rimnicu Vilcea
```
General search algorithm

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

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

loop

\hspace{1em} \text{if there are no candidate states to explore return failure}
\hspace{1em} \text{choose a leaf node of the tree to expand next according to \text{strategy}}
\hspace{1em} \text{if the node satisfies the goal condition return the solution}
\hspace{1em} \text{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 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 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 can 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** \((\text{problem}, \text{Queuing-fn})\)

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

- Candidates are added to \(\text{nodes}\) representing the queue structure

---

Implementation of search

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

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

- State

- ST Node

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

- children

---

CS 1571 Intro to AI
Search methods

Properties of different 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?

Complexities are measured in terms of parameters:

  • $b$ – maximum branching factor
  • $d$ – depth of the optimal solution
  • $m$ – maximum depth of the state space

Uninformed search methods

• use only 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
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

Arad
  /   
Zerind Sibiu Timisoara

queue ➔ Zerind
       Sibiu
       Timisoara

Breadth-first search

Arad
  /   
Zerind Sibiu Timisoara
  /   
Arad Oradea

queue ➔ Sibiu
       Timisoara
       Arad
       Oradea
Breadth-first search

queue
Arad
Oradea
Fagaras
Timisoara

CS 1571 Intro to AI
Properties of breadth-first search

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