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 6 1 8 7 3 2</td>
<td>1 2 3 4 5 6 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)
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)

Problem-solving as search

- Many search problems can be formulated graph search problems
- **A graph search problem can be described in terms of:**
  - A set of states representing different world situations
  - Initial state
  - Goal condition
  - Operators defining valid moves between states
**Puzzle 8 as a graph search problem**

- **Puzzle 8.** Find a sequence of moves from the initial configuration to the goal configuration.
- **Note:** the graph for some problem can become very large,

![Graph for Puzzle 8](image)

**N-queens as a graph search problem**

**Search space:**
- all configurations of N queens on the board
**Graph search:**
- States: configurations N queens
- Operators: change a positions of one of the queens
- Initial state: an arbitrary configuration
- Goal: non-attacking queens

![Graph for N-queens](image)
Two different N-queens 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

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 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, built on-line during the search process

![Search tree diagram]

Search tree

- A **search tree** = a (search) exploration trace
  - It is different from the graph defining the problem
  - states can repeat in the search tree

![Search tree diagram]
A branch in the search tree = 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 \((problem, strategy)\)
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 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 *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 constituting part of a search tree

![](image)

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

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.**
  – **Complexity** is 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]

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
Breadth-first search

queue

Breadth-first search

queue
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: ?

• Time complexity: ?

• Memory (space) complexity: ?
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>
<tr>
<td>(d)</td>
<td>(2^d) ((b^d))</td>
</tr>
<tr>
<td>(d+1)</td>
<td>(2^{d+1} )(b^{d+1})</td>
</tr>
</tbody>
</table>

Total nodes: ?
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>
<tr>
<td>d</td>
<td>$2^d$ (b^d)</td>
</tr>
<tr>
<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:** ?
BFS – memory complexity

- Count nodes kept in the tree structure or in the queue

<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>
<tr>
<td>$d$</td>
<td>$2^d$ (b^d)</td>
</tr>
<tr>
<td>$d+1$</td>
<td>$2^{d+1}$ (b^{d+1})</td>
</tr>
</tbody>
</table>

Total nodes: $O(b^d)$

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