Informed search methods

Milos Hauskrecht
milos@pitt.edu
5329 Sennott Square

Announcements

Homework assignment 2 is out
• Due on Tuesday, September 19, 2017 before the class
• Two parts:
  – Report
  – Programming part (Puzzle 8)

Course web page:
http://www.cs.pitt.edu/~milos/courses/cs2710/
Search methods

• Uninformed search methods
  – Breadth-first search (BFS)
  – Depth-first search (DFS)
  – Iterative deepening (IDA)
  – Bi-directional search
  – Uniform cost search

• Informed (or heuristic) search methods:
  – Best first search with a heuristic function

Evaluation-function driven search

• A search strategy can be defined in terms of a node evaluation function
  – Similarly to the path cost for the uniform cost search
• Evaluation function
  – Denoted $f(n)$
  – Defines the desirability of a node to be expanded next
• Evaluation-function driven search:
  – expand the node with the best evaluation-function value
• Implementation:
  – priority queue with nodes in the decreasing order of their evaluation function value
**Uniform cost search**

- **Uniform cost search (Dijkstra’s shortest path):**
  - A special case of the evaluation-function driven search
    \[ f(n) = g(n) \]

- **Path cost function** \( g(n) \):
  - path cost from the initial state to \( n \)

- **Uniform-cost search:**
  - Can handle general minimum cost path-search problem:
  - **weights or costs** associated with operators (links).

- **Note:** Uniform cost search relies on the problem definition only
  - It is an uninformed search method

**Additional information to guide the search**

- **Uninformed search methods**
  - use only the information from the problem definition; and
  - past explorations, e.g. cost of the path generated so far

- **Informed search methods**
  - incorporate additional measure of a potential of a specific state to reach the goal
  - a potential of a state (node) to reach a goal is measured by a **heuristic function**

- Heuristic function is denoted \( h(n) \)
**Best-first search**

**Best-first search** = *evaluation-function driven search*
- Typically incorporates a *heuristic function*, $h(n)$, into the evaluation function $f(n)$ to guide the search.

**Heuristic function $h(n)$:**
- Measures a potential of a state (node) to reach a goal
- Typically expressed in terms of some distance to a goal estimate

**Example of a heuristic function:**
- Assume a shortest path problem with city distances on connections
- Straight-line distances between cities give additional information we can use to guide the search

---

**Example: traveler problem with straight-line distance information**

- Straight-line distances give an estimate of the cost of the path between the two cities
Best-first search

Best-first search = evaluation-function driven search

• Typically incorporates a heuristic function, \( h(n) \), into the evaluation function \( f(n) \) to guide the search.

• **heuristic function**: measures a potential of a state (node) to reach a goal

**Special cases** (differ in the design of evaluation function):

– **Greedy search**
  \[ f(n) = h(n) \]

– **A* algorithm**
  \[ f(n) = g(n) + h(n) \]

+ **iterative deepening** version of A*: **IDA***

---

Greedy search method

• Evaluation function is equal to the heuristic function
  \[ f(n) = h(n) \]

• **Idea**: the node that seems to be the closest to the goal is expanded first
Greedy search

\[ f(n) = h(n) \]

queue \[ \rightarrow \]

\[
\begin{array}{c}
\text{Arad} \\
366
\end{array}
\]

Greedy search

\[ f(n) = h(n) \]

queue \[ \rightarrow \]

\[
\begin{array}{c}
\text{Sibiu} \\
253
\end{array},
\begin{array}{c}
\text{Timisoara} \\
329
\end{array},
\begin{array}{c}
\text{Zerind} \\
374
\end{array}
\]

\[
\begin{array}{c}
\text{Arad} \\
366
\end{array}
\]

\[
\begin{array}{c}
\text{Zerind} \\
374
\end{array},
\begin{array}{c}
\text{Sibiu} \\
253
\end{array},
\begin{array}{c}
\text{Timisoara} \\
329
\end{array}
\]

\[
\begin{array}{c}
75 \\
140 \\
118
\end{array}
\]
Properties of greedy search

- Completeness: No. We can loop forever. Nodes that seem to be the best choices can lead to cycles.
  - Yes. Elimination of state repeats can solve the problem.

- Optimality: ?

- Time complexity: ?

- Memory (space) complexity: ?
Example: traveler problem with straight-line distance information

Greedy search result

Total: 450

Greedy search and optimality

Total: 418
Properties of greedy search

- **Completeness:**
  - No. We can loop forever. Nodes that seem to be the best choices can lead to cycles.
  - Yes. Elimination of state repeats can solve the problem.

- **Optimality:** No.
  Even if we reach the goal, we may be biased by a bad heuristic estimate. **Evaluation function disregards the cost of the path built so far.**

- **Time complexity:** $O(b^m)$
  Worst case !!! But often better!

- **Memory (space) complexity:** $O(b^m)$
  Often better!

---

A* search

- The problem with the **greedy search** is that it can keep expanding paths that are already very expensive.
- The problem with the **uniform-cost search** is that it uses only past exploration information (path cost), no additional information is utilized

- **A* search**
  \[ f(n) = g(n) + h(n) \]
  \[ g(n) \] - cost of reaching the state
  \[ h(n) \] - estimate of the cost from the current state to a goal
  \[ f(n) \] - estimate of the path length

- **Additional A* condition:** admissible heuristic
  \[ h(n) \leq h^*(n) \] for all \( n \)
A* search example

\[ f(n) \]

- Arad: 366
- Sibiu: 393
- Timisoara: 447
- Zerind: 449

M. Hauskrecht
A* search example

M. Hauskrecht
A* search example

M. Hauskrecht
A* search example

Properties of A* search

• Completeness: ?

• Optimality: ?

• Time complexity:
  – ?

• Memory (space) complexity:
  – ?
Properties of A* search

• **Completeness:** can we get stuck in the infinite loop?

  • **Optimality:** ?

  • **Time complexity:**
    – ?

  • **Memory (space) complexity:**
    – ?

---

Properties of A* search

• **Completeness:** can we get stuck in the infinite loop? **No!**
  
  • Then the algorithm is complete even without repeat checks.

• **Optimality:** ?

• **Time complexity:**
  – ?

• **Memory (space) complexity:**
  – ?
Properties of A* search

- **Completeness:** Yes.
- **Optimality:** ?
- **Time complexity:** – ?
- **Memory (space) complexity:** – ?

Optimality of A*

- In general, a heuristic function \( h(n) \):
  - It can overestimate, be equal or underestimate the true distance of a node to the goal \( h^*(n) \)
- Is the A* optimal for an arbitrary heuristic function?
Example: traveler problem with straight-line distance information

- Admissible heuristics

\[ f(n) = 220 + 400 = 620 \]

M. Hauskrecht
Example: traveler problem with straight-line distance information

\[ f(n) = 220 + 400 = 620 \]

\[ f(n) = 450 \]

- **Admissible heuristics**

Total path: 450 is suboptimal

---

**Optimality of A***

- In general, a heuristic function \( h(n) \):
  - Can overestimate, be equal or underestimate the true distance of a node to the goal \( h^*(n) \)
  - Is the A* optimal for an arbitrary heuristic function?
  - **No!**
Optimality of A*

• In general, a heuristic function \( h(n) \):
  Can overestimate, be equal or underestimate the true distance of a node to the goal \( h^*(n) \)
• Admissible heuristic condition
  – Never overestimate the distance to the goal !!!
    \[ h(n) \leq h^*(n) \quad \text{for all } n \]

Example: the straight-line distance in the travel problem never overestimates the actual distance

Is A* search with an admissible heuristic optimal ??

Optimality of A* (proof)

• Let G1 be the optimal goal (with the minimum path distance). Assume that we have a sub-optimal goal G2. Let \( n \) be a node that is on the optimal path and is in the queue together with G2

Then: \( f(G2) = g(G2) \) since \( h(G2) = 0 \)
\[ > g(G1) \quad \text{since G2 is suboptimal} \]
\[ \geq f(n) \quad \text{since } h \text{ is admissible} \]

And thus A* never selects G2 before \( n \)
Properties of A* search

• Completeness: Yes.

• Optimality: Yes (with the admissible heuristic)

• Time complexity:
  – Order roughly the number of nodes with \( f(n) \) smaller than the cost of the optimal path \( g^* \)

• Memory (space) complexity:
  – Same as time complexity (all nodes in the memory)
Admissible heuristics

Heuristics can be designed using a relaxed version of the problem

- **Example:** the 8-puzzle problem

  ![Initial position vs Goal position](image)

  **Admissible heuristics:**
  1. number of misplaced tiles
  2. Sum of distances of all tiles from their goal positions (Manhattan distance)

Heuristics 1: number of misplaced tiles

  ![Initial position vs Goal position](image)

  \[ h(n) \text{ for the initial position: ?} \]
Admissible heuristics

Heuristics 1: number of misplaced tiles

<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>

\(h(n)\) for the initial position: 7

Admissible heuristics

- Heuristic 2: Sum of distances of all tiles from their goal positions (Manhattan distance)

<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>

\(h(n)\) for the initial position:
Admissible heuristics

• **Heuristic 2**: Sum of distances of all tiles from their goal positions (Manhattan distance)

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

\[ h(n) \text{ for the initial position:} \]
\[ 2 + 3 + 3 + 1 + 1 + 2 + 0 + 2 = 14 \]

For tiles: 1 2 3 4 5 6 7 8

Admissible heuristics

• We can have multiple admissible heuristics for the same problem

• **Dominance**: Heuristic function \( h_1 \) dominates \( h_2 \) if

\[ \forall n \quad h_1(n) \geq h_2(n) \]

• **Combination**: two or more admissible heuristics can be combined to give a new admissible heuristics
  – Assume two admissible heuristics \( h_1, h_2 \)

\[ h_3(n) = \max(h_1(n), h_2(n)) \]

Then: \( h_3(n) \) is admissible
Iterative deepening algorithm (IDA)

• Based on the idea of the limited-depth search,
• It resolves the difficulty of knowing the depth limit ahead of time.

Idea: try all depth limits in an increasing order.
That is, search first with the depth limit \( l=0 \), then \( l=1, l=2 \), and so on until the solution is reached

Iterative deepening combines advantages of the depth-first and breadth-first search with only moderate computational overhead

Properties of IDA

• **Completeness:** Yes. The solution is reached if it exists.
  (the same as BFS)
• **Optimality:** Yes, for the shortest path.
  (the same as BFS)
• **Time complexity:**
  \[ O(1) + O(b^1) + O(b^2) + \ldots + O(b^d) = O(b^d) \]
  exponential in the depth of the solution \( d \)
  worse than BFS, but asymptotically the same
• **Memory (space) complexity:**
  \[ O(db) \]
  much better than BFS
IDA*

Iterative deepening version of A*

• Progressively increases the evaluation function limit (instead of the depth limit)

• Performs limited-cost depth-first search for the current evaluation function limit
  – Keeps expanding nodes in the depth first manner up to the evaluation function limit

• Problem: it is unclear what the amount by which the evaluation limit should be progressively increased

IDA*

Problem: the amount by which the evaluation limit should be progressively increased

Solutions:
(1) peak over the previous step boundary to guarantee that in the next cycle some number of nodes is expanded
(2) Increase the limit by a fixed cost increment – say $\varepsilon$

Cost limit = $k \cdot \varepsilon$
**Solution 1:** peak over the previous step boundary to guarantee that in the next cycle more nodes are expanded

Properties:
- the choice of the new cost limit influences how many nodes are expanded in each iteration
- Assume I choose a limit such that at least 5 new nodes are examined in the next DFS run
- What is the problem here?

---

**Solution 1:** peak over the previous step boundary to guarantee that in the next cycle more nodes are expanded

Properties:
- the choice of the new cost limit influences how many nodes are expanded in each iteration
- Assume I choose a limit such that at least 5 new nodes are examined in the next DFS run
- What is the problem here?
  - Fix: ?
**Solution 1:** peak over the previous step boundary to guarantee that in the next cycle more nodes are expanded

**Properties:**
- the choice of the new cost limit influences how many nodes are expanded in each iteration
- Assume I choose a limit such that at least 5 new nodes are examined in the next DFS run
- What is the problem here?
  - **Fix:** complete the search up to the limit to find the best

**Solution 2:** Increase the limit by a fixed cost increment ($\varepsilon$)

**Properties:**
- What is bad?

Cost limit = $k \varepsilon$
**IDA***

**Solution 2:** Increase the limit by a fixed cost increment ($\varepsilon$)

Properties:
- What is bad? Too many or too few nodes expanded – no control of the number of nodes
- What is the quality of the solution?
  - The solution found first may differ by $<\varepsilon$ from the optimal solution

Cost limit = $k \varepsilon$