Informed search methods

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

Announcements

**Homework assignment 2 is out**
- Due on Thursday, September 24, 2009 before the class
- **Two parts:**
  - Pen and pencil part
  - Programming part (Puzzle 8): informed search methods

**Course web page:**
http://www.cs.pitt.edu/~milos/courses/cs1571/
Iterative deepening algorithm (IDA)

- Based on the idea of the limited-depth search, but
- 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
Bi-directional search

- In some search problems we want to find the path from the initial state to the unique goal state (e.g. traveler problem)
- Bi-directional search idea:
  - Search both from the initial state and the goal state;
  - Use inverse operators for the goal-initiated search.

Why bidirectional search? What is the benefit? Assume BFS.
- Cut the depth of the search space by half

\[ O(b^{d/2}) \quad \text{Time and memory complexity} \]
**Bi-directional search**

Why bidirectional search? Assume BFS.
- It cuts the depth of the search tree by half.

What is necessary?
- Merge the solutions.

![Diagram showing initial state and goal state](image)

- How? The hash structure remembers the side of the tree the state was expanded first time. If the same state is reached from other side we have a solution.

---

**Minimum cost path search**

*Traveler example with distances [km]*

*Optimal path:* the shortest distance path from Arad to Bucharest
Searching for the minimum cost path

- **General minimum cost path-search problem:**
  - adds weights or costs to operators (links)

  “Intelligent” expansion of the search tree should be driven by the cost of the current (partially) built path

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

**Search strategy:**

- Expand the leaf node with the minimum \( g(n) \) first.
  - When operator costs are all equal to 1 it is equivalent to BFS
- The basic algorithm for finding the minimum cost path:
  - **Dijkstra’s shortest path**
- In AI, the strategy goes under the name
  - Uniform cost search

---

**Properties of the uniform cost search**

- **Completeness:** Yes, assuming that operator costs are non-negative (the cost of path never decreases)
  \[ g(n) \leq g(\text{successor } (n)) \]
- **Optimality:** Yes. Returns the least-cost path.

- **Time complexity:**
  number of nodes with the cost \( g(n) \) smaller than the optimal cost

- **Memory (space) complexity:**
  number of nodes with the cost \( g(n) \) smaller than the optimal cost
Elimination of state repeats

Idea:
• A node is redundant and can be eliminated if there is another node with exactly the same state and a shorter path from the initial state

Assuming positive costs:
• If the state has already been expanded, is there a shorter path to that node?

Implementation:
• Marking with the hash table
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 through a heuristic function
  – A heuristic function is denoted as $h(n)$

Evaluation-function driven search

• A search strategy can be defined in terms of a node evaluation function

• **Evaluation function**
  – Denoted $f(n)$
  – Defines the desirability of a node to be expanded next

• **Evaluation-function driven search:** expand the node (state) 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

Best-first search

Best-first search
- 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
- Typically 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

- 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: Arad 366

Arad

Greedy search

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

Queue: Sibiu 253, Timisoara 329, Zerind 374

Arad 366

Zerind 374

Sibiu 253

Timisoara 329
Greedy search

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

M. Hauskrecht
Properties of greedy search

- **Completeness**: No.
  We can loop forever. Nodes that seem to be the best choices can lead to cycles. 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!
Example: traveler problem with straight-line distance information

Greedy search result

Total: 450

Greedy search and optimality

Total: 418
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
  \[ h(n) \leq h^*(n) \quad \text{for all } n \]

A* search example

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

\[
\begin{array}{c}
\text{ queue } \rightarrow \\
\text{ Arad } & 366 \\
\end{array}
\]
A* search example

- **Arad**: 366
- **Zerind**: 449
- **Sibiu**: 393
- **Fagaras**: 99
- **Rimnicu Vilcea**: 447
- **Pitesti**: 415
- **Bucharest**: 418
- **Timisoara**: 447
- **Rimnicu V**: 607
- **Craiova**: 526
- **Oradea**: 553
- **Orad**: 646

- **f(n)**
  - Fagaras: 417
  - Bucharest: 418
  - Timisoara: 447
  - Zerind: 449
  - Orada: 526
  - Craiova: 526
  - Sibiu: 553
  - Arad: 646

- **Queue**: Pitesti, Sibiu, 415
A* search example

M. Hauskrecht
Properties of A* search

• Completeness: ?

• 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

overestimate
Example: traveler problem with straight-line distance information

• Admissible heuristics

M. Hauskrecht
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 is 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

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

Then:
\[ f(G_2) = g(G_2) \quad \text{since} \quad h(G_2) = 0 \]
\[ > g(G_1) \quad \text{since} \quad G_2 \text{ is suboptimal} \]
\[ \geq f(n) \quad \text{since} \quad h \text{ is admissible} \]

And thus A* never selects G2 before n

Properties of A* search

• Completeness: Yes.

• Optimality: Yes (with the admissible heuristic)

• Time complexity:
  \[ \_ \_ \_ \_ \]

• Memory (space) complexity:
  \[ \_ \_ \_ \_ \]
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 are designed based on relaxed version of problems
- Example: the 8-puzzle problem

<table>
<thead>
<tr>
<th>Initial position</th>
<th>Goal position</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Initial position" /></td>
<td><img src="image2" alt="Goal position" /></td>
</tr>
</tbody>
</table>

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

**Heuristics 1:** number of misplaced tiles

<table>
<thead>
<tr>
<th>Initial position</th>
<th>Goal position</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Initial Position" /></td>
<td><img src="image2" alt="Goal Position" /></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><img src="image1" alt="Initial Position" /></td>
<td><img src="image2" alt="Goal Position" /></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 heuristic
  - Assume two admissible heuristics \( h_1, h_2 \)
    
    Then:
    \[
    h_3(n) = \max(h_1(n), h_2(n))
    \]
    
    is admissible