Informed (heuristic) search (cont).

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

Administration

• PS–1 due today
  – Report before the class begins
  – Programs through ftp

• PS-2 is out
  – due next week on Wednesday, September 21, 2005
    • Report
    • Programs
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

**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

\[ \text{Arad} \quad 366 \]

CS 2710 Foundations of AI
Greedy search

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

CS 2710 Foundations of AI

Greedy search

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

CS 2710 Foundations of AI
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

Example: traveler problem with straight-line distance information

- Greedy search and optimality
**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**

![A* search example diagram](image-url)
A* search example
A* search example

CS 2710 Foundations of AI
A* search example

CS 2710 Foundations of AI

Goal !!
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

<table>
<thead>
<tr>
<th>City</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>166</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Dobrota</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</td>
</tr>
<tr>
<td>Gârda</td>
<td>77</td>
</tr>
<tr>
<td>Hârman</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>264</td>
</tr>
<tr>
<td>Mehedia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitești</td>
<td>402</td>
</tr>
<tr>
<td>Râmnicu Vâlcea</td>
<td>400</td>
</tr>
<tr>
<td>Sâlaj</td>
<td>553</td>
</tr>
<tr>
<td>Timișoara</td>
<td>329</td>
</tr>
<tr>
<td>Urațiunii</td>
<td>300</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Example: traveler problem with straight-line distance information

• Admissible heuristics

---

Example: traveler problem with straight-line distance information

• 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 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.

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

Initial position | Goal position
---|---
4 5 6 1 8 7 3 2 | 1 2 3 4 5 6 7 8

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

- We can have multiple admissible heuristics for the same problem
- **Dominance:** Heuristic function $h_1$ dominates $h_2$ if
  \[ \forall n \ h_1(n) \geq h_2(n) \]
- **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

---

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

**Solutions:** ?
**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:** the amount by which the evaluation limit should be progressively increased

**Solutions:**
- peak over the previous step boundary
- Increase the limit by a fixed cost increment – say ε

---

**IDA***

**Solution 1: peak over the previous step boundary**

**Properties:**
- the choice of the new cost limit influences how many nodes are expanded in each iteration
- We may find a sub-optimal solution
  - **Fix 1:** increase the limit to the minimum f value above the limit
  - **Fix 2:** complete the search up to the limit to find the best

**Solution 2: Increase the limit by a fixed cost increment (ε)**

**Properties:**
- Too many or too few nodes expanded – no control of the number of nodes
- The solution of accuracy difference < ε is found