Vorlesung
Grundlagen der Künstlichen Intelligenz

Reinhard Lafrenz / Prof. A. Knoll

Robotics and Embedded Systems
Department of Informatics – I6
Technische Universität München

www6.in.tum.de
lafrenz@in.tum.de
089-289-18136
Room 03.07.055

Wintersemester 2012/13 2.11.2012
Chapter 3

Solving Problems by Searching: Informed (Heuristic) Search
What’ the problem?

Combinatorial explosion:

- Uninformed search leads to exponential time and can only be solved for small problems
  - 15-puzzle: $10^{13}$ configurations
  - Rubik’s cube: $4 \times 10^{19}$ configurations
    - 1 million years with 1 turn per second
  - Chess: $10^{120}$ configurations (assuming ~ 40 moves)

How to solve it?

- Use additional information to reduce complexity
- Choose the node to expand based on an estimation on how fast the goal can be reached
Heuristics and their properties

Make use of domain knowledge:
   „more knowledge, less search“
- Domain knowledge can be considered as „rules of thumb“
- Heuristics are simple rules that evaluate nodes with respect to the distance to the goal
- Good heuristics are
  - Good estimators
  - Simple and fast to compute
Best-first Search

- Information about the costs from a given node to the goal:
  - Evaluation function $h$, giving a real number for each node
  - Ideal case:
    - Knowing the correct costs from the node to the goal
  - Simple heuristics:
    - Euclidean distance
    - Manhattan distance

- Modify the generic graph-search algorithm using the heuristics

- When $h$ is correct, i.e. estimation gives the actual costs:
  Follow the path of lowest cost, no need to search
Modify generic graph-search algorithm for best-first search

```
function HEURISTIC-SEARCH(problem, h) returns a solution or an error

static: open, the initial state (set of nodes)
        closed, the nodes already visited, initially empty set

forever
    if open is empty then return error
    take a node out of open
    add this node to closed
    if this node contains a goal state then return solution
    expand this node (i.e. take all successors not in closed)
    add successor nodes to open using h
```

- Way of adding successor nodes defined by the heuristics
Greedy best-first search

- The “goodness“ of a node is determined by the distance to the goal
  \[ h(n) = \text{estimated distance from node } n \text{ to the goal} \]
- Constraint for \( h \): \( h(n) = 0 \), if \( n \) is a goal node
- In path planning: Direct distance between two locations
Greedy best-first search: From Arad to Bucharest

Air-line distances to Bucharest

<table>
<thead>
<tr>
<th>Location</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Drobeta</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</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>100</td>
</tr>
<tr>
<td>Rimnicu Vîlcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timișoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
Greedy best-first search: From Arad to Bucharest

Use air-line distance as heuristic function $h$
Heuristics

- In case of greedy search, the *evaluation function* $h$ is called a *heuristic function* or simply *heuristic*
- Name comes from greek εὑρίσκειν (to find, „Eureka!“)

- In AI:
  - Heuristics are fast, but probably incomplete methods for solving problems [Newell, Shaw, Simon 1963]
  - Heuristics are a means to accelerate search in average case

- A heuristic is problem-specific and focused on search
A* algorithm

- Minimizes the estimated path costs
- Combines uniform cost search and best first greedy

\[ g(n): \text{cost so far to reach } n \]
\[ h(n): \text{estimated cost from } n \text{ to a goal node} \]
\[ f(n) = g(n) + h(n): \text{estimated total path cost through } n \]

Let \( h^* \) be the true cost of an optimal path from \( n \) to goal

\( h \) is admissible, if for all nodes \( n \):

\[ h(n) \leq h^*(n) \]

\( h \) is optimistic, \( h \) never overestimates the actual costs
A*: From Arad to Bucharest

Air-line distances to Bucharest

<table>
<thead>
<tr>
<th>Location</th>
<th>Distance to Bucharest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arad</td>
<td>366</td>
</tr>
<tr>
<td>Bucharest</td>
<td>0</td>
</tr>
<tr>
<td>Craiova</td>
<td>160</td>
</tr>
<tr>
<td>Drobeta</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>176</td>
</tr>
<tr>
<td>Giurgiu</td>
<td>77</td>
</tr>
<tr>
<td>Hirsova</td>
<td>151</td>
</tr>
<tr>
<td>Iasi</td>
<td>226</td>
</tr>
<tr>
<td>Lugoj</td>
<td>244</td>
</tr>
<tr>
<td>Mehadia</td>
<td>241</td>
</tr>
<tr>
<td>Neamt</td>
<td>234</td>
</tr>
<tr>
<td>Oradea</td>
<td>380</td>
</tr>
<tr>
<td>Pitesti</td>
<td>100</td>
</tr>
<tr>
<td>Rimnicu Vilcea</td>
<td>193</td>
</tr>
<tr>
<td>Sibiu</td>
<td>253</td>
</tr>
<tr>
<td>Timisoara</td>
<td>329</td>
</tr>
<tr>
<td>Urziceni</td>
<td>80</td>
</tr>
<tr>
<td>Vaslui</td>
<td>199</td>
</tr>
<tr>
<td>Zerind</td>
<td>374</td>
</tr>
</tbody>
</table>
A*: From Arad to Bucharest

(a) The initial state

(b) After expanding Arad

(c) After expanding Sibiu

(d) After expanding Rimnicu Vilcea
A*: From Arad to Bucharest

(e) After expanding Fagaras

(f) After expanding Pitesti
**A* algorithm: properties**

$h$ is **admissible**, if for all nodes $n$: $h(n) \leq h^*(n)$

A (slightly) more strict condition:

**Consistency (monotony):**

$h$ is **consistent**, if for all nodes $n$:

$$h(n) \leq c(n,a,n') + h(n')$$

*where* $c(n,a,n')$ *are the costs from node* $n$ *to a successor node* $n'$ *as a result of the action* $a$

**Thesis:** If $h$ is consistent, then $h$ is also admissible
A* algorithm: properties

Two versions of A*:
- Tree-search based
- Graph-search based

Theorem: A* is optimal if
- $h$ is admissible in case of tree-search based A*
- $h$ is consistent in case of graph-search based A*
**A* algorithm: Optimality of tree-search form**

**Thesis:** A* is optimal, i.e. the first solution found by A* has minimal costs

**Proof:** Assume there exists a goal node $G$ with optimal path costs $f^*$, but A* has found a different goal $G_2$ with $g(G_2) > f^*$
A* algorithm: Optimality of tree-search form

Let $n$ be a node on the optimal path from $\text{start}$ to $G$ which has not been expanded. Since $h$ is admissible, 

$$f(n) \leq f^*.$$ 

But because $n$ hasn’t been expanded before $G_2$, it holds that 

$$f(G_2) \leq f(n)$$

From this it follows that 

$$f(G_2) \leq f^*.$$ 

Because $h(G_2) = 0$ by definition, it follows that 

$$g(G_2) \leq f^*.$$ 

$\rightarrow$ to assumption $g(G_2) > f^*$. Proof by contradiction.
A* algorithm: Optimality of graph-search form

If $h$ is consistent, the values of $f = g + h$ are monotonically increasing (not strictly).

Let $n'$ be a successor node of $n$. For an action $a$ holds

$$g(n') = g(n) + c(n,a,n')$$

This leads to

$$f(n') = g(n') + h(n') = g(n) + c(n,a,n') + h(n') \geq g(n) + h(n) = f(n)$$
A* algorithm: Optimality of graph-search form

If \( h \) is consistent, the values of \( f = g + h \) are monotonically increasing (not strictly).

Let \( n' \) be a successor node of \( n \). For an action \( a \) holds
\[
g(n') = g(n) + c(n, a, n')
\]
This leads to
\[
f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)
\]

Now to prove: If a node \( n \) was chosen for expansion, then the optimal path to \( n \) has been found.
A* algorithm: Optimality of graph-search form

Assume there is another cheaper path from \textit{start} to \textit{n}.

Then there is a node \textit{n'} on that path with \( f(n') < f(n) \) because of monotony of \( f \) along any path.

Contradiction to algorithm definition: \textit{n'} would have been chosen instead of another node in the same set of frontier nodes because its costs are lower.

Then, taking \( h(\text{goal})=0 \) into account, the function \( f \) gives the true cost for any goal and the costs for all other nodes on the way are at least as expensive.
A* algorithm: Optimality of graph-search form

We can draw a “contour map“ with nodes within a f-cost limit
A* algorithm: Properties

- A* expands all nodes with $f(n) < C^*$
  - $C^*$ are the costs of an optimal path
- Completeness requires that there is only a finite number of nodes with $f(n) < C^*$
  - True, if step costs $> \varepsilon > 0$ and branching factor $b$ is finite
- No node with $f(n) > C^*$ is expanded
- If not all nodes with $f(n) < C^*$ are expanded, an algorithm risks to miss the optimal solution
A* algorithm: Properties

- A* is complete
- A* is optimal
- But: Number of configurations still exponential, even with pruning!
- Time exponential, but drastically reduced
- Space is the major problem

- Variation of A*: IDA* (Iterative deepening A*)
  - Pruning based on f-costs (g+h) instead of d
  - Because of iteration: no need to keep track of priority queue
Summary

- There are optimal and complete search algorithms which are “much better” than blind search.
- However, the state spaces and the complexity is still exponential.

- A* always leads to optimal solutions, but space is a problem.
  - Variations of A* to save space.
Questions:

Restriction of costs to positive values:

a) Why would an optimal algorithm need to expand the whole space in case of arbitrary negative costs?

b) Does a restriction to $c(n,a,n') > \min \text{ (negative val.)}$ help?
   - In case of trees and in case of graphs?

c) Assume there are loops and the world state is the same after a finite number of actions. What is the optimal strategy in case of negative path costs for all actions?

d) Are there negative costs in real life?
Questions:

True or false?

a) Depth-first expands always at least as many nodes as A* with an admissible heuristic.

b) For the 8-puzzle, h(n) = 0 is admissible.

c) A* is not suitable for robotics, because percepts, actions, and states deal with continuous values.

d) In chess, a rook (Turm) can move only horizontally or vertically, but not jump over other chessmen. The manhattan distance is admissible for a move from A zu B.
Questions:

In graph-based A*, there can be state spaces with suboptimal solutions if $h$ is admissible, but not consistent. Show an example.