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 5.11.2012
Chapter 3

Solving Problems by Searching: Informed (Heuristic) Search (cont‘d)
Finding heuristic functions

- What is a good heuristic function?

- $h_1 =$ number of tiles at wrong location
- $h_2 =$ sum of distances between tiles and their goal location (Manhattan distance)
Empirical evaluation of different heuristics

- \( d = \) distance to goal
- Average over 100 instances

<table>
<thead>
<tr>
<th>( d )</th>
<th>IDS</th>
<th>( A^*(h_1) )</th>
<th>( A^*(h_2) )</th>
<th>IDS</th>
<th>( A^*(h_1) )</th>
<th>( A^*(h_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>2.45</td>
<td>1.79</td>
<td>1.79</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
<td>2.87</td>
<td>1.48</td>
<td>1.45</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>18</td>
<td>2.73</td>
<td>1.34</td>
<td>1.30</td>
</tr>
<tr>
<td>8</td>
<td>6384</td>
<td>39</td>
<td>25</td>
<td>2.80</td>
<td>1.33</td>
<td>1.24</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
<td>39</td>
<td>2.79</td>
<td>1.38</td>
<td>1.22</td>
</tr>
<tr>
<td>12</td>
<td>364404</td>
<td>227</td>
<td>73</td>
<td>2.78</td>
<td>1.42</td>
<td>1.24</td>
</tr>
<tr>
<td>14</td>
<td>3473941</td>
<td>539</td>
<td>113</td>
<td>2.83</td>
<td>1.44</td>
<td>1.23</td>
</tr>
<tr>
<td>16</td>
<td>—</td>
<td>1301</td>
<td>211</td>
<td>—</td>
<td>1.45</td>
<td>1.25</td>
</tr>
<tr>
<td>18</td>
<td>—</td>
<td>3056</td>
<td>363</td>
<td>—</td>
<td>1.46</td>
<td>1.26</td>
</tr>
<tr>
<td>20</td>
<td>—</td>
<td>7276</td>
<td>676</td>
<td>—</td>
<td>1.47</td>
<td>1.27</td>
</tr>
<tr>
<td>22</td>
<td>—</td>
<td>18094</td>
<td>1219</td>
<td>—</td>
<td>1.48</td>
<td>1.28</td>
</tr>
<tr>
<td>24</td>
<td>—</td>
<td>39135</td>
<td>1641</td>
<td>—</td>
<td>1.48</td>
<td>1.26</td>
</tr>
</tbody>
</table>
Effect of heuristic precision

Effective branching factor: Let $N =$ number of expanded nodes
- $d =$ depth of solution in search space
- then $b^*$ is the branching factor of the uniform search tree with depth $d$ and $N$ nodes
- $N+1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$

Dominance of heuristics
- $h_1$ dominate $h_2$, if for all nodes $n$ is true that:
  \[ h_1(n) \geq h_2(n) \]
- This also means that A* with $h_1$ expands less nodes than $h_2$ on average
Choice of heuristics

- If possible, choose heuristics with higher values
  - Needs to be admissible/consistent
  - Check for calculation time of heuristics

- Example: $h_1$ and $h_2$ are heuristics for the 8-puzzle

- They also describe the exact path length for relaxed problems
  - Relaxed problem solved by $h_1$: Arbitrary jump of each field to the empty one
  - Relaxed problem solved by $h_2$: Any move (one step horizontally or vertically) is possible, even if position occupied
Choice of heuristics

- What if there is no “unambiguously best“ heuristic?

- Assume, several (admissible/consistent) heuristics $h_1, h_2, \ldots h_m$ exist. How to choose?

- Combine all!

  $$h(n) = \max (h_1(n), h_2(n), \ldots h_m(n))$$

  This takes the most precise one for each node.

- Given that $h_1, h_2, \ldots h_m$ are admissible/consistent. Does this also hold true for $h$?
Chapter 4

Beyond Classical Search
Local search and optimization

- Up to now:
  - systematic exploration of search spaces
  - Keep track of alternatives for each node along the path
  - The path is the solution

- What if only the **final state** is of interest for the solution?

  ➡️ Local search

- Examples:
  - 8-queens problem
  - VLSI design,
  - TSP
Local search and optimization

- Define an objective function that evaluates states
- Use this function to optimize the search for a solution
- Idea: start with a random configuration and increase the solution stepwise ➔ Hill climbing
Hill climbing

- Define an objective function that evaluates states
- Goal: maximizing the objective function

```plaintext
function HILL-CLIMBING(problem) returns a state that is a local maximum
inputs: problem, a problem
local variables: current, a node
              neighbor, a node

current ← MAKE-NODE(problem.INITIAL-STATE)
loop do
    neighbor ← a highest-valued successor of current
    if neighbor.VALUE ≤ current.VALUE then
        return current.STATE
    current ← neighbor
end
```
Hill climbing: Example 8-queen problem

- Cost function: number of attacks
- Next state: Only one vertical move (queens remain in column)

![Diagram](image)

(a) $h=17$

(b) $h=1$ (local minimum)
Problems of local search

- **Local maxima**: algorithm returns a sub-optimal solution.
- **Plateaus**: algorithm can only explore randomly.
- **Edges**: similar to plateaus.
Problems of local search

- **Local maxima**: algorithm returns a sub-optimal solution.
- **Plateaus**: algorithm can only explore randomly.
- **Edges**: similar to plateaus.

Solutions:

- **Re-start**, if no increase in performance
- **Noise**, random walk
- **Restricted search**: the last \( n \) operators cannot be applied

Strategies (and their parameters) that perform successfully (on a certain type of problem) can in most cases only be determined empirically.
Simulated annealing

- Introduction of noise
- Imagine rough surface, “shake” the system to overcome local minima

```plaintext
function SIMULATED-ANNEALING(problem, schedule) returns a solution state

  inputs: problem, a problem
           schedule, a mapping from time to “temperature”

  local variables: current, a node
                   next, a node
                   T, a “temperature” controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] − VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability exp(ΔE / T)
```
Local beam search

- Restrict the nodes in memory to constant k
- Initialize list with k random nodes
- Explore all successors of all k nodes
- Take the “best“ k nodes out of this list, according to optimization function and use them for next step

Problem: concentration on small area (promising?) of the search space
- Updated list not with best k nodes, but with randomly chosen ones, based on a distribution given by the objective function
Genetic algorithms

Evolution seems to be successful

Idea: Similar to evolution, solutions are searched by applying operators like “cross-over”, “mutation” and “selection” to already successful solutions.

Components:

- **Encoding** of configurations as string or bit-string
- “**Fitness**" function that evaluates the goodness of a configuration
- **Populations** of configurations, initially random choice

Example: 8 queens problem encoded as string of 8 digits. Fitness function is computed based on the number of non-attacks \((28=7+6+5+\ldots+1\) for a solution)
Population consists of the set of queen configurations.
Genetic algorithms: 8-queen problem

- Compute fitness for each configuration in population
- Choose two pairs for crossover, probability based on fitness
- Randomly choose crossover position for each pair
- Choose mutation with low probability
Genetic algorithms: 8-queen problem
Search with non-deterministic action results

- Result of an action can be unobservable (or partially) observable
- Result of an action can be non-deterministic
- No clear sequence of actions possible

contingency plan or strategy
Search with non-deterministic action results

- Reconsider vacuum world with additional properties of the “suck” action:
  - Sometimes also the other field is cleaned
  - In case of a clean field, dirt may be released

- No unique result of an action, but a set of possible outcomes
Search with non-deterministic action results

- Describe contingency plan in form of result-dependent action sequence
  
  [action, result-dependent successor actions]

- Example:
  
  [SUCK, if state=5 then [RIGHT, SUCK] else []]

- These resulting if-then-else cascades lead to decision trees

- Two types of branching out possible
  - Agent‘s own decision (what is the next action?)
  - Depending of the (non-deterministic) outcome of an action
Search with non-deterministic action results

- Search trees can be described as tree with two types of nodes
  - OR-nodes describe actions chosen by the agent
  - AND-nodes describe possible outcomes
- Alternating “layers“ of nodes (OR,AND) in the search tree
- A solution to a problem is a subtree with
  - A goal node at each leave
  - An action for each OR node
  - All branches of an AND-node included
- Several search strategies can be applied, e.g. depth-first, …
- Finding heuristic functions is more complicated
  - Estimation of costs for a contingency plan instead of an action sequence
Search with non-deterministic action results

Agent's choice (OR node)

Possible outcomes (AND node)

GOAL

LOOP

Suck

Right

Left

Suck
Search with non-deterministic action results

- What if “move“ actions fail? E.g. “Right“

- No acyclic solution anymore, search fails
- Introduce labels for parts of plans
  \[\text{[Suck, } L_1 : \text{Right, if state=5 then } L_1 \text{ else Suck]}\]
  or simply \text{while state=5 do right}
Search with non-deterministic action results

Diagram showing decision paths with actions such as 'Suck' and 'Right'.
Summary

- Criteria for choosing “good” heuristics

- Local search and optimization
  - Useful if only the final state is of interest
  - Problem: local minima, plateaus, etc.
  - Several algorithms: hill-climbing, simulated annealing, local beam search, genetic algorithms, etc.

- Search with non-deterministic action results
  - Contingency plan instead of action sequence
  - AND-OR-trees
No class on Friday, 9th November 2012!

9. November 2012
MasterTUM
Infomesse
Campus Innenstadt
Immatrikulationshalle
www.tum.de/studium/master/mastermesse/

When? 9-17h
Where? Immatrikulationshalle Campus Stadtmitte