Foundations of AI

4. Informed Search Methods

Heuristics, Local Search Methods, Genetic Algorithms

Wolfram Burgard, Andreas Karwath, Bernhard Nebel, and Martin Riedmiller
Contents

• Best-First Search
• A* and IDA*
• Local Search Methods
• Genetic Algorithms
Best-First Search

Search procedures differ in the way they determine the next node to expand.

**Uninformed Search:** Rigid procedure with no knowledge of the cost of a given node to the goal.

**Informed Search:** Knowledge of the worth of expanding a node is in the form of an *evaluation function* $f$ or $h$, which assigns a real number to each node.

**Best-First Search:** Search procedure that expands the node with the “best” $f$- or $h$-value.
General Algorithm

When $h$ is always correct, we do not need to search!
Greedy Search

A possible way to judge the “worth” of a node is to estimate its distance to the goal.

\[ h(n) = \text{estimated distance from } n \text{ to the goal} \]

The only real restriction is that \( h(n) = 0 \) if \( n \) is a goal.

A best-first search with this function is called a greedy search.

Route-finding problem: \( h = \text{straight-line distance between two locations} \).
Greedy Search Example

Straight-line distance 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>Dobreta</td>
<td>242</td>
</tr>
<tr>
<td>Eforie</td>
<td>161</td>
</tr>
<tr>
<td>Fagaras</td>
<td>178</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>98</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>
Greedy Search from *Arad* to *Bucharest*
Heuristics

The evaluation function $h$ in greedy searches is also called a *heuristic* function or simply a *heuristic*.

- The word *heuristic* is derived from the Greek word ευρίσκειν (note also: εὐρήκα!).
- The mathematician Polya introduced the word in the context of problem solving techniques.
- In AI it has two meanings:
  - Heuristics are fast but in certain situations incomplete methods for problem-solving [Newell, Shaw, Simon 1963] (The greedy search is actually generally incomplete).
  - Heuristics are methods that improve the search in the average-case.

→ In all cases, the heuristic is *problem-specific* and *focuses* the search!
A*: Minimization of the estimated path costs

A* combines the greedy search with the uniform-search strategy.

\( g(n) = \) actual cost from the initial state to \( n \).

\( h(n) = \) estimated cost from \( n \) to the next goal.

\( f(n) = g(n) + h(n) \), the estimated cost of the cheapest solution through \( n \).

Let \( h^*(n) \) be the actual cost of the optimal path from \( n \) to the next goal.

\( h \) is admissible if the following holds for all \( n \) :

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

We require that for A*, \( h \) is admissible (straight-line distance is admissible).
A* Search Example

Straight-line distance to Bucharest

Arad 366
Bucharest 0
Craiova 160
Dobreta 242
Eforie 161
Fagaras 178
Giurgiu 77
Hirsova 151
Iasi 226
Lugoj 244
Mehadia 241
Neamt 234
Oradea 380
Pitesti 98
Rimnicu Vilcea 193
Sibiu 253
Timisoara 329
Urziceni 80
Vaslui 199
Zerind 374
A* Search from *Arad* to *Bucharest*
Example: Path Planning for Robots in a Grid-World
Optimality of A*

Claim: The first solution found has the minimum path cost.

Proof: Suppose there exists a goal node $G$ with optimal path cost $f^*$, but A* has found another node $G_2$ with $g(G_2) > f^*$. 
Let $n$ be a node on the path from the start to $G$ that has not yet been expanded. Since $h$ is admissible, we have

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

Since $n$ was not expanded before $G_2$, the following must hold:

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

and

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

It follows from $h(G_2) = 0$ that

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

→ Contradicts the assumption!
Completeness and Complexity

Completeness:
If a solution exists, A* will find it provided that (1) every node has a finite number of successor nodes, and (2) there exists a positive constant $\delta$ such that every operator has at least cost $\delta$.

$\Rightarrow$ Only a finite number of nodes $n$ with $f(n) \leq f^*$.

Complexity:
In the case in which $|h^*(n) - h(n)| \leq O(\log(h^*(n)))$, only one goal state exists, and the search graph is a tree, a sub-exponential number of nodes will be expanded [Gaschnig, 1977, Helmert & Roeger, 2008].

Normally, growth is exponential because the error is proportional to the path costs.
Heuristic Function Example

\[ h_1 = \text{the number of tiles in the wrong position} \]

\[ h_2 = \text{the sum of the distances of the tiles from their goal positions (Manhattan distance)} \]
Empirical Evaluation

- $d = \text{distance from goal}$
- Average over 100 instances

<table>
<thead>
<tr>
<th>$d$</th>
<th>IDS</th>
<th>$A^*(h_1)$</th>
<th>$A^*(h_2)$</th>
<th>Search Cost</th>
<th>Effective Branching Factor</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>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>13</td>
<td>12</td>
<td>2.87</td>
<td>1.48</td>
</tr>
<tr>
<td>6</td>
<td>680</td>
<td>20</td>
<td>18</td>
<td>2.73</td>
<td>1.34</td>
</tr>
<tr>
<td>8</td>
<td>6384</td>
<td>39</td>
<td>25</td>
<td>2.80</td>
<td>1.33</td>
</tr>
<tr>
<td>10</td>
<td>47127</td>
<td>93</td>
<td>39</td>
<td>2.79</td>
<td>1.38</td>
</tr>
<tr>
<td>12</td>
<td>364404</td>
<td>227</td>
<td>73</td>
<td>2.78</td>
<td>1.42</td>
</tr>
<tr>
<td>14</td>
<td>3473941</td>
<td>539</td>
<td>113</td>
<td>2.83</td>
<td>1.44</td>
</tr>
<tr>
<td>16</td>
<td>–</td>
<td>1301</td>
<td>211</td>
<td>–</td>
<td>1.45</td>
</tr>
<tr>
<td>18</td>
<td>–</td>
<td>3056</td>
<td>363</td>
<td>–</td>
<td>1.46</td>
</tr>
<tr>
<td>20</td>
<td>–</td>
<td>7276</td>
<td>676</td>
<td>–</td>
<td>1.47</td>
</tr>
<tr>
<td>22</td>
<td>–</td>
<td>18094</td>
<td>1219</td>
<td>–</td>
<td>1.48</td>
</tr>
<tr>
<td>24</td>
<td>–</td>
<td>39135</td>
<td>1641</td>
<td>–</td>
<td>1.48</td>
</tr>
</tbody>
</table>
Iterative Deepening A* Search (IDA*)

Idea: A combination of IDS and A*. All nodes inside a contour are searched.

```plaintext
function IDA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: f-limit, the current f- COST limit
    root, a node
  root ← MAKE-NODE(INITIAL-STATE[problem])
  f-limit ← f- COST(root)
  loop do
    solution,f-limit ← DFS-COUMT(root,f-limit)
    if solution is non-null then return solution
    if f-limit = ∞ then return failure; end

function DFS-COUMT(node,f-limit) returns a solution sequence and a new f- COST limit
  inputs: node, a node
    f-limit, the current f- COST limit
  static: next-f, the f- COST limit for the next contour, initially ∞
  if f- COST[node] > f-limit then return null, f- COST[node]
  if GOAL-TEST[problem](STATE[node]) then return node, f-limit
  for each node s in SUCCESSORS(node) do
    solution, new-f ← DFS-COUMT(s,f-limit)
    if solution is non-null then return solution, f-limit
    next-f ← MIN(next-f,new-f); end
  return null, next-f
```
Local Search Methods

In many problems, it is unimportant how the goal is reached – only the goal itself matters (8-queens problem, VLSI Layout, TSP).

If in addition a quality measure for states is given, a local search can be used to find solutions.

Idea: Begin with a randomly-chosen configuration and improve on it stepwise → Hill Climbing.
Hill Climbing

function HILL-CLIMBING(problem) returns a solution state

inputs: problem, a problem
static: current, a node
        next, a node

current ← MAKE-NODE(INITIAL-STATE[problem])
loop do
    next ← a highest-valued successor of current
    if VALUE[next] < VALUE[current] then return current
    current ← next
end
Example: 8-Queens Problem

Selects a column and moves the queen to the square with the fewest conflicts.
Problems with Local Search Methods

- \textit{Local maxima}: The algorithm finds a sub-optimal solution.
- \textit{Plateaus}: Here, the algorithm can only explore at random.
- Ridges: Similar to plateaus.

\textbf{Solutions:}
- \textit{Start over} when no progress is being made.
- “Inject noise” \rightarrow random walk
- Tabu search: Do not apply the last $n$ operators.

Which strategies (with which parameters) are successful (within a problem class) can usually only empirically be determined.
Simulated Annealing

In the simulated annealing algorithm, “noise” is injected systematically: first a lot, then gradually less.

```plaintext
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
           schedule, a mapping from time to “temperature”
  static: current, a node
           next, a node
           T, a “temperature” controlling the probability 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 \(e^{\frac{\Delta E}{T}}\)
```

Has been used since the early 80’s for VSLI layout and other optimization problems.
Genetic Algorithms

Evolution appears to be very successful at finding good solutions.

Idea: Similar to evolution, we search for solutions by “crossing”, “mutating”, and “selecting” successful solutions.

Ingredients:

• Coding of a solution into a string of symbols or bit-string
• A fitness function to judge the worth of configurations
• A population of configurations

Example: 8-queens problem as a chain of 8 numbers. Fitness is judged by the number of non-attacks. The population consists of a set of arrangements of queens.
Selection, Mutation, and Crossing

Many variations:
how selection will be applied, what type of cross-over operators will be used, etc.

**Selection**
Selection of individuals according to a fitness function and pairing.

**Cross-over**
Calculation of the breaking points and recombination.

**Mutation**
According to a given probability elements in the string are modified.
Summary

- **Heuristics** focus the search
- **Best-first search** expands the node with the highest worth (defined by any measure) first.
- With the minimization of the evaluated costs to the goal $h$ we obtain a **greedy search**.
- The minimization of $f(n) = g(n) + h(n)$ combines uniform and greedy searches. When $h(n)$ is admissible, i.e., $h^*$ is never overestimated, we obtain the **A* search**, which is complete and optimal.
- **IDA*** is a combination of the iterative-deepening and A* searches.
- **Local search methods** only ever work on one state, attempting to improve it step-wise.
- **Genetic algorithms** imitate evolution by combining good solutions.