Foundations of AI
4. Informed Search Methods

Heuristics, Local Search Methods, Genetic Algorithms
Wolfram Burgard and Luc De Raedt

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 cost of a given node to the goal is in the form of an evaluation function $f$, which assigns a real number to each node.

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

General Algorithm

```
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution sequence
  inputs: problem, a problem
          Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by Eval-Fn

  return GENERAL-SEARCH(problem, Queueing-Fn)
```

When the evaluation function 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 condition 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: \( h = \) straight-line distance between two locations.

---

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 ἡ εὐρήκα σα ν (note also: ἡ ἐρυθρὰ)
- 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.

$g(n) = \text{actual cost from the initial state to } n$.

$h(n) = \text{estimated cost from } n \text{ to the next goal}$.

$f(n) = g(n) + h(n)$, 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

Contours in A*

Within the search space, contours arise in which for the given $f$-value all nodes are expanded.

Contours at $f = 380, 400, 420$
Example: Path Planning for Robots in a Grid-World

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

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

$\rightarrow$ Contradicts the assumption!

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 $g(G_2) > f^*$.

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 where $|h^*(n) - h(n)| \leq O(\log h^*(n))$, only a sub-exponential number of nodes will be expanded.

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

Start State

Goal State

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

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

Empirical Evaluation

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

<table>
<thead>
<tr>
<th>( d )</th>
<th>IDS</th>
<th>( A^\ast(h_1) )</th>
<th>( A^\ast(h_2) )</th>
<th>IDS</th>
<th>( A^\ast(h_1) )</th>
<th>( A^\ast(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>304404</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>599</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>1661</td>
<td>-</td>
<td>1.48</td>
<td>1.26</td>
</tr>
</tbody>
</table>

Iterative Deepening A* Search (IDA*)

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

```
function IDA*(problem) returns a solution sequence
inputs: problem, a problem
static: f, limit, the current f, Cost limit
root, a node
next ← MAKE-VOID(INITIAL-STATES(problem))
(f, limit) ← f, Cost(root)
loop do
    solution, {f, limit} ← DFS-CONSTRUCT(root, f, limit)
    if solution is non-null then return solution
    if f = ∞ then return failure; end

function DFS-CONSTRUCT(node, f, limit) returns a solution sequence and a new f, Cost limit
inputs: node, a node
static: node, the f, Cost limit for the next contour, initially ∞
if f < Cost(node) then return null, f, Cost(node)
if GOAL-TEST(problem)(STATES(node)) then return node, f, limit
for each node in SUCCESSORS(node) do
    solution, new_f, new_limit ← DFS-CONSTRUCT(node, f, limit)
    if solution is non-null then return solution, f, limit
next_f ← Min(next_f, new_f, end
return next_f, next_f
```

Heuristics for CSPs

Which variables should be instantiated with which values?
Common Heuristics

Most constrained variable first:
→ Reduces the branching factor!

Most-constraining variable first:
Choose the variable involved in the most constraints on other unassigned variables.
→ Reduces the future branching factor.

Least-constraining value first:
→ Allows more freedom on future choices
→ Solves the 1000-queens problem!

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

Problems with Local Search Methods

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

Solutions:
- Start over when no progress is being made.
- “Inject smoke” → 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, “smoke” is injected systematically: first a lot, then gradually less.

```plaintext
function SIMULATED-ANNEALING (problem, schedule) returns a solution state
    input: problem, a problem
     schedule, a mapping from time to “temperature”
    output: current, a node
    T, a “temperature” controlling the probability of downward steps
    current ← MAKE-NODE(INITIAL-SOLUTION(problem))
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T > 0 then
            new ← randomly selected successor of current
            delta ← VALUE(new) − VALUE(current)
            if delta > 0 then
                current ← new
            else current ← current with probability $e^{\frac{\text{delta}}{T}}$
```

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

Application in CSPs

Although in CSPs, configurations are either solutions or failures, local search can be applied here as well.

*Quality measure:* The number of fulfilled constraints. In this case, local search is also referred to as *heuristic repair.*

*Heuristic repair* was used to schedule observations for the Hubble telescope. It reduced the scheduling time from 3 weeks to 10 minutes.

In the context of finding satisfactory assignments for Boolean formulas, such methods can be very successfully applied.

Example: 8-Queens Problem

Selects a column and moves the queen to the square with the fewest conflicts.

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-overs will be used, etc.

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.