Heuristic Search

Algorithms

Best-First Search .................................................. 2
General Best-First Search Algorithm .......................... 3
$A^*$ Search Algorithm ........................................... 4
Iterative Deepening $A^*$ Search Algorithm .................... 5
IDA$^*$'s Contour Procedure ..................................... 6

Analysis

Properties of $A^*$ Search ........................................ 7
Optimality Proof ..................................................... 8
Efficiency of $A^*$ .................................................... 9
Performance of Heuristic Functions ............................. 10
Book Experiment Avoiding Back Edges ......................... 11

Local Search

Local Search ......................................................... 12
Local Optima Example ............................................ 13
Local Search Algorithm .......................................... 14
Examples of Local Search Algorithms ......................... 15

Algorithms

Best-First Search

- Simple search algorithms such as IDS do not consider the goodness of states.
- Best-first search visits states according to an evaluation function.
- An evaluation function gives lower numbers to (seemingly) better states.
- Heuristic search prefers to visit states that appear to be better.
- $A^*$ search visits states based on cost from initial to a given state plus heuristic function.
- A heuristic function estimates the cost from a given state to the closest goal state.

General Best-First Search Algorithm

```latex
function \texttt{Best-FS}(initial, Expand, Goal, Eval-Fn) 
q ← \texttt{New-Priority-Queue}() 
\texttt{Insert}(initial, q, Eval-Fn(initial))
while q is not empty 
do current ← \texttt{Extract-Min}(q)
  if Goal(current) then return solution
  for each next in Expand(current) 
do \texttt{Insert}(next, q, Eval-Fn(next))
return failure
```

Analysis

- Properties of $A^*$ Search ........................................ 7
- Optimality Proof ..................................................... 8
- Efficiency of $A^*$ .................................................... 9
- Performance of Heuristic Functions ............................. 10
- Book Experiment Avoiding Back Edges ......................... 11

Local Search

- Local Search ......................................................... 12
- Local Optima Example ............................................ 13
- Local Search Algorithm .......................................... 14
- Examples of Local Search Algorithms ......................... 15
A* Search Algorithm

function A*(initial, Expand, Goal, Cost, Heuristic)
q ← New-Priority-Queue()
Insert(initial, q, Heuristic(initial))
while q is not empty
do current ← Extract-Min(q)
if Goal(current) then return solution
for each next in Expand(current)
do Insert(next, q, Cost(next) + Heuristic(next))
return failure

Iterative Deepening A* Search Algorithm

function IDA*(initial, Expand, Goal, Cost, Heuristic)
limit ← Heuristic(initial)
loop
do result, limit ← Contour(initial, limit)
if result then return result
if limit = ∞ then return failure

IDA*’s Contour Procedure

function Contour(current, limit)
cost ← Cost(current) + Heuristic(current)
if limit < cost then return null, cost
if Goal(current) then return solution, cost
new-limit ← ∞
for each next in Expand(current)
do result, cost ← Contour(next, limit)
if result then return solution, cost
new-limit ← min(new-limit, cost)
return failure, new-limit

Properties of A* Search
- Let $n$ be a state/node.
- Let $g(n)$ be the cost from the initial state to $n$.
- Let $h(n)$ be the estimate from $n$ to a goal state.
- Let $f(n) = g(n) + h(n)$.
- $h$ is admissible if it is never an overestimate.
- If $h$ is admissible, then A* finds optimal path.
- If $h$ is admissible with $\epsilon$ error and the search space is a uniform tree with one goal state, then A* searches at most $\epsilon/2$ from solution path.

Optimality Proof
- Let $f^*$ be optimal path cost.
- Because $h$ never overestimates, then all states $n$ on optimal path have $f(n) \leq f^*$.
- Any nonoptimal goal state has $f(n) > f^*$.
- Because of priority queue, A* will visit optimal path before any nonoptimal goal state.
Efficiency of A*

- Assume tree-structured state space ($b =$ branching factor, $d =$ goal depth), single goal state, each edge costs 1, and maximum error of $\varepsilon$.
- Any state $n$ more than $\varepsilon/2$ off of solution path has $f(n) = g(n) + h(n) > f^*$.
- All states $n$ on solution path have $f(n) = g(n) + h(n) \leq f^*$.
- $A^*$ and IDA* visit $O(db^{\varepsilon/2})$ states.
- $A^*$ uses $O(db^{\varepsilon/2})$ memory. IDA* uses $O(db)$.

Performance of Heuristic Functions

Consider these 8-puzzle heuristic functions:

- $h_1$: number of tiles in goal position.
- $h_2$: Manhattan distance from tiles to goals.
- Both never overestimate and $h_1 \leq h_2$

Characterize by effective branching factor

- Let $N$ states be visited and solution depth be $d$.
- Solve for $x$ in $N = \sum_{i=0}^{d} x^i$.

<table>
<thead>
<tr>
<th>$d$</th>
<th>IDS</th>
<th>IDA*($h_1$)</th>
<th>IDA*($h_2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>52  (2.35)</td>
<td>10 (1.35)</td>
<td>7 (1.17)</td>
</tr>
<tr>
<td>8</td>
<td>569 (2.03)</td>
<td>42 (1.36)</td>
<td>14 (1.11)</td>
</tr>
<tr>
<td>12</td>
<td>5357 (1.92)</td>
<td>315 (1.47)</td>
<td>45 (1.19)</td>
</tr>
<tr>
<td>16</td>
<td>47271 (1.87)</td>
<td>2410 (1.52)</td>
<td>226 (1.28)</td>
</tr>
<tr>
<td>20</td>
<td>17646 (1.55)</td>
<td>764 (1.29)</td>
<td></td>
</tr>
</tbody>
</table>

Local Search

- A local search algorithm keeps track of one state at a time.
- An evaluation function and a selection procedure is used to decide what state to visit next.
- Local search gives up optimality guarantees in hopes of finding good solutions more efficiently.
- The main difficulty is local minima/maxima.

Local Optima Example
Local Search Algorithm

function LOCAL-SEARCH(initial, Expand, Goal, Select)

    current ← initial

loop

    do if Goal(current) then return solution

    current ← Select(Expand(current))

Examples of Local Search Algorithms

- Hill-Climbing, Gradient Descent:
  Select state improving an evaluation function.
- Random-restart hill-climbing:
  Repeat hill climbing from random initial states.
- Simulated Annealing:
  Hill-climbing with randomized selection.
- Genetic Algorithms:
  Maintain a set of “current states.” Crossover generates new states from pairs of states.
- Tabu Search: Like hill-climbing, but avoid recently visited states or recently used operators.