Breadth-first heuristic search

Paper by Rong Zhou, Eric A. Hansen
Presentation by Salomé Simon
Breadth-first tree search

- Used for search problems with uniform edge cost
  - Prerequisite for presented techniques
- Drawback: all nodes need to be stored
Why store all nodes?

• Solution path reconstruction
• Duplicate detection

• Depth-first search can do this with a stack
Memory efficient solution path recovery

• Divide and conquer principle
• Store one "relay layer", recursively solve sub-problems

All nodes here have a pointer to its ancestor in the relay layer
Memory efficient duplicate detection

- Frontier search: used operator bits
  - No closed list
  - Cannot use upper bound for pruning

- Sparse memory: counter for predecessors
  - Only nodes with counter ≠ 0 in closed list
  - Better for high branching factor
Memory efficient duplicate detection

• Drawbacks
  – Need to know predecessors
  – A* needs to have a consistent heuristic
Layered duplicate detection

• Only usable for breadth-first search
• Open list: current and next layer
• Closed list: current and x previous layers
• For undirected graphs (& uniform edge cost): only one previous layer needs to be stored
• For directed graphs: Max g-cost difference between optimal g-cost of predecessor and successor (hard to determine)
  – But only linear regeneration when one layer is saved
Layered duplicate detection: Example

Legend:
- Red circle: Current (expansion) layer
- Blue circle: Next layer
- Green circle: Previous layer
- Red number: Expansion order
- Blue circle with circle: Invisible nodes
- Gray line: Duplicate expansion

Diagram shows a layered structure with nodes representing different layers and expansion orders.
Layered duplicate detection: Example
Advantages of heuristic breadth-first search

- Frontier size is smaller (no proof):
  - Breadth-first #layers: $f^* + 1$
  - Best first #layers: $f^* - h(\text{start}) + 1$

$\approx$ more layers means smaller layer size (if perfect upper bound)
Advantages of breadth-first search

- No sorting (FIFO)
- Frontier search
  - Can prune nodes above upper bound, since optimal $g(n)$ is found once the node is expanded
    $\rightarrow$ works also with admissible but not consistent heuristic
  - Easier memory allocation since no sorting needed

- Layered duplicate detection
  - Only for breadth-first search
  - Easiest to implement
Breadth-first branch-and-bound (BFBnB)

• Lower bound: \( f(n) = g(n) + h(n) \)
• Upper bound for pruning unpromising paths
  – With perfect upper bound BFBnB expands the same nodes as A* (disregarding ties), else more
• Solution path recovery: divide-and-conquer
  – Relay layer at 3/4 depth, since 1/2 are usually the biggest layers (more pruning later)
• Duplicate detection:
  – Frontier search
  – Sparse memory
  – Layered duplicate detection
Breadth-first iterative deepening A* (BFIDA*)

• BFBnB with iterative deepening
• Gives perfect upper bound
  – Useful when no good upper bound can be estimated
• Asymptotically optimal for node expansions
  – Even in directed graphs under certain conditions
• Does not use tie breaking (not beneficial)
  – Tie breaking only useful in last layer, but for breadth-first this layer is rather small
## Results: Fifteen Puzzle

- **Used algorithm:** frontier-BFIDA*
  - Low branching factor
- **4 GB memory limitation**
  - Frontier- / sparse-memory-A* only solved 96/100
  - Maximal memory usage by BFIDA*: 1.3 GB

<table>
<thead>
<tr>
<th></th>
<th>Frontier A*</th>
<th>Sparse memory A*</th>
<th>Frontier BFIDA*</th>
</tr>
</thead>
<tbody>
<tr>
<td>#nodes stored</td>
<td>4.15x</td>
<td>2.74x</td>
<td>1x</td>
</tr>
<tr>
<td>Peak Memory</td>
<td>6.2x</td>
<td>4.1x</td>
<td>1x</td>
</tr>
</tbody>
</table>
Results: Fifteen Puzzle

• DFIDA* performs best
  – More nodes expanded (not all duplicates detected), but
  – Lower node-generation overhead
Results: 4-peg Towers of Hanoi

- Used algorithm: Frontier-BFBnB
  - A (probably) perfect upper bound can be found
- 2 GB memory limitation
  - Only BFBnB could solve 19-disk problem
- Inconsistent heuristic
  - Frontier-A* is not guaranteed to find optimal solution
## Results: 4-peg Towers of Hanoi

<table>
<thead>
<tr>
<th>Disks</th>
<th>Frontier A* Nodes Stored</th>
<th>Frontier A* Expanded</th>
<th>Frontier BFBnB Stored</th>
<th>Frontier BFBnB Expanded</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>2'126'885</td>
<td>10'398'240</td>
<td>390'844</td>
<td>11'628'818</td>
</tr>
<tr>
<td>18</td>
<td>25'987'984</td>
<td>202'577'805</td>
<td>6'987'695</td>
<td>211'993'782</td>
</tr>
<tr>
<td>19</td>
<td>&gt;128'000'000</td>
<td>&gt;1'193'543'025</td>
<td>55'241'327</td>
<td>1'824'553'083</td>
</tr>
</tbody>
</table>
Results: Domain independent STRIPS planning

• Used algorithm: BFBnB with layered duplicate detection
  – easier to implement when not knowing the domain
  – even in domains with directed graphs one stored layer gives good results

• Compared to A* and DFIDA*

• BFBnB expands more nodes than A*, but uses significantly less memory

• DFIDA* performs poor due to excessive node regeneration
Results: General observations

• Compared to A*, Breadth-first has
  – Significantly less memory usage
  – More node expansions (especially if upper bound not perfect)

• Layered duplicate detection easy to implement and gives good results