Lecture 7: Search 6

Victor R. Lesser
CMPSCI 683
Fall 2010
This Lecture

- RTA*
- Hierarchical A*
- Beginning of Local Search
  - Hill-Climbing/Iterative Improvement
RTA* - Real-Time A*

*Intermix partial search with execution of action*

- Goal: reduce the execution time of A*.
- Method: limit the search horizon of A* and select an action (single move) in constant time.
  - Make decision about *next move in real-world* without a *complete plan* (path) to reach goal state
- Two stages
  - Make individual move decision: Perform mini-min search with alpha pruning
  - Make a sequence of decisions to arrive at a solution
    - recovering from inappropriate actions
    - avoid loops
First Phase - Minimin Search with Alpha-Pruning

- Mini-min depth-first look-ahead search
  - Returns back-up $f$ value for a node from looking ahead to the frontier node at the horizon
    - Can viewed as simply a more accurate and computationally expensive heuristic function
  - Reason: If the heuristic function $h$ is consistent/monotone and admissible, then the error in the backed-up cost estimate cannot increase with search depth, $f$ is always increasing and thus better estimate of actual cost

- Alpha pruning
  - If current minimum $f$ of horizon node (alpha value) is less than $f$ of an intermediate node, the intermediate node (and any successors) can be eliminated from further consideration
  - Reason: $f$ is monotonic (never can get lower $f$) and you are only searching to horizon (don’t need goal state to prune)
Basis of RTA*

- $h(a) \leq g(a \text{ to } c) + h(c) \leq h^*(a)$;
- assuming you need to go to the goal state thru $c$ from $a$

- As a result of exploring in the search space from $a$ to $c$, you can replace $h(a)$ with the better (more informed) estimate $g(a \text{ to } c) + h(c)$

- This leads to a more informed decision at S whether to take the “action in the real world of moving” to either state $y$, $a$, or $x$. 

V. Lesser; CS683, F08
Procedure for Calculating Backed-Up Value of a Move

\textbf{procedure evaluate}(move, limit)

/* return backed-up estimate \( f' \) (move) by \( \alpha \)-pruning search to depth \( limit \) */

1. Open \( \leftarrow \) \{move\}; \( \alpha \leftarrow \infty \)
2. \( f \) (move) \( \leftarrow \) \( g \) (move) + \( h \) (move);
3. While (open not empty) do
4. node \( \leftarrow \) pop (Open);
5. expand node; for each child of node do
6. \( g \) (child) \( \leftarrow \) \( g \) (node) + \textit{move-cost};
7. \( f \) (child) \( \leftarrow \) \( g \) (child) + \( h \) (child);
   Prune child if \( f \) (child) \( \geq \alpha \)
8. if \( f \) (child) \( < \alpha \) do
9. if (depth = \textit{limit} or goal(child)) then
   \( \alpha \leftarrow f \) (child);
10. else put child on Open; od od od
11. Return \( \alpha \);
RTA* - Controlling the Sequence of Moves Executed in Real-World

Basic Principle:

“One should backtrack to a previously visited real world state when the estimate of solving the problem from that state plus the cost of returning to that state is less than the estimated cost of going forward from the current state.” - Korf

- Merit of every node $f(n) = g(n) + h(n)$ is measured relative to the current position of the problem solver in the real-world
  - initial state is irrelevant

- If one moves back to a previously visited real-world state, then it needs to take into account that one already has taken action there
  - value of state is next best $f$

Remember Interplay between partial search and execution of action in real-world
RTA* Algorithm

- Maintains in a hash table a list of those states/nodes that have been visited by an *actual move* in the real world of the problem solver;
- At each cycle in the real-world, the current state is expanded and the heuristic function, possibly augmented by look-ahead search, is applied to each successor state which is *not in the hash table*;
- The $f$ value of each neighboring state is computed by adding the $h$ value plus the cost of the link to the current state;
- The neighbor with the minimum $f$ value is chosen for the current state;
- The **second best $f$ value is stored in the hash table** for the current state
  - Represents the estimated $h$ cost of solving the problem by returning to this state
  - Second best avoids loops
Example of RTA*

<table>
<thead>
<tr>
<th>node</th>
<th>h</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>i</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

V. Lesser; CS683, F08
Characteristics of RTA*

- Completeness of RTA*
  - In a finite problem space with positive edge costs and finite heuristic value, in which a goal state is reachable from every state, RTA* will find a solution.

- Local optimal of RTA*
  - Each move made by RTA* on a tree is along a path whose estimated cost of reaching a goal is minimum, based on the cumulative search frontier at the time.
Hierarchical Problem Solving
Hierarchical Heuristic A* Search

Generate heuristic \( h(s) \) for a state \( s \) in base/original state space \( S \)

Abstract space, \( \varphi_2(\varphi_1(S)) \)

Abstract space, \( \varphi_1(S) \)

Original space, \( S \)
Automatically Generating State Space Abstraction
“Max-degree” Star Abstraction

- The state with the highest degree is grouped together with its neighbors within a certain distance (the abstraction radius) to form a single abstract state.
Star abstraction with radius = 1

State with the largest degree within a certain distance/radius is grouped together with neighbors, repeat for non-grouped states
Naïve Hierarchical A* - Cache $h$ in abstract space; avoid search for $h(s2)$ if $h(s1)$ already computed [$h(\Phi(s1))$] and $\Phi(s1) = \Phi(s2)$

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Size (number of states)</th>
<th>Nodes Expanded</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All Levels</td>
<td>Base Level</td>
<td>Blind Search</td>
</tr>
<tr>
<td>Blocks-5</td>
<td>1166</td>
<td>866</td>
<td>389</td>
</tr>
<tr>
<td>5-puzzle</td>
<td>961</td>
<td>720</td>
<td>348</td>
</tr>
<tr>
<td>Fool’s Disk</td>
<td>4709</td>
<td>4096</td>
<td>1635</td>
</tr>
<tr>
<td>Hanoi-7</td>
<td>2894</td>
<td>2187</td>
<td>1069</td>
</tr>
<tr>
<td>KL2000</td>
<td>3107</td>
<td>2736</td>
<td>1236</td>
</tr>
<tr>
<td>MC 60-40-7</td>
<td>2023</td>
<td>1878</td>
<td>934</td>
</tr>
<tr>
<td>Permute-6</td>
<td>731</td>
<td>720</td>
<td>286</td>
</tr>
<tr>
<td>Words</td>
<td>5330</td>
<td>4493</td>
<td>1923</td>
</tr>
</tbody>
</table>

Table 1. Naïve Hierarchical A*. (abstraction radius = 2)
Reducing Search in Abstract Spaces

- Observation: all searches related to the same base level problem have the same goal.

- This allows additional types of caching of values.

- It leads to variants of Hierarchical A* Search (Valtorta’s barrier) requiring less effort in 5 out of 8 search spaces.
Exploit Information for Repeated Blind Search in Abstract Space

- **V1 - h* caching**
  - Cache exact h’s (h*) along optimal solution in abstract space
    - Cache for use in base level search (don’t need to search again since already know optimal distant to goal in abstract space) \([h(\Phi(s1)), h(\Phi_i) .. [h(\Phi_j)]\]

- **V2**
  - Cache optimal path in abstract space (optimal-path caching)
    - Exploit in further searches in abstract space (if reach such a node in abstract space can stop search along this path) – can stop further search in \(\Phi\) once you have \(h(\Phi_i)\) that you found was on optimal path in previous search

- **V3**
  - Pertains to states that were opened (or closed) during abstract search but are not on the solution path
  - Remember optimal path length in abstract search space (P-g caching)
    - P being optimal path length from start to goal in abstract space

V. Lesser; CS683, F10
## Hierarchical A*

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Blind Search</th>
<th>Nodes Expanded</th>
<th></th>
<th></th>
<th></th>
<th># problems V3 &lt; BS (out of 200)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks-5</td>
<td>389</td>
<td>2766</td>
<td>1235</td>
<td>478</td>
<td>402</td>
<td>96 *</td>
</tr>
<tr>
<td>5-puzzle</td>
<td>348</td>
<td>3119</td>
<td>1616</td>
<td>854</td>
<td>560</td>
<td>14 *</td>
</tr>
<tr>
<td>Fool’s Disk</td>
<td>1635</td>
<td>12680</td>
<td>8612</td>
<td>3950</td>
<td>1525</td>
<td>132</td>
</tr>
<tr>
<td>Hanoi-7</td>
<td>1069</td>
<td>18829</td>
<td>10667</td>
<td>5357</td>
<td>3174</td>
<td>0 *</td>
</tr>
<tr>
<td>KL2000</td>
<td>1236</td>
<td>7059</td>
<td>3490</td>
<td>1596</td>
<td>1028</td>
<td>171</td>
</tr>
<tr>
<td>MC 60-40-7</td>
<td>934</td>
<td>2412</td>
<td>1531</td>
<td>1154</td>
<td>863</td>
<td>128</td>
</tr>
<tr>
<td>Permute-6</td>
<td>286</td>
<td>806</td>
<td>482</td>
<td>279</td>
<td>242</td>
<td>113</td>
</tr>
<tr>
<td>Words</td>
<td>1923</td>
<td>19386</td>
<td>7591</td>
<td>2849</td>
<td>1410</td>
<td>124</td>
</tr>
</tbody>
</table>
The Granularity of Abstraction

- Increasing the radius of abstraction has two contradictory effects:
  
  + abstract spaces contain fewer states and each abstract search produces values for more states, but

  - the heuristic is less discriminating

- Using the best case radius Hierarchical A* Search (Valtorta’s barrier) is more effective every search space.
Hierarchical A* with best abstraction radius

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Radius</th>
<th>Nodes Expanded</th>
<th># problems V3 &lt; BS (out of 200)</th>
<th>CPU seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks-5</td>
<td>5</td>
<td>389 Blind, 611 Hierarchical A* 309 V3</td>
<td>123</td>
<td>69 Blind, 86 V3</td>
</tr>
<tr>
<td>5-puzzle</td>
<td>12</td>
<td>348 Blind, 354 Hierarchical A* 340 V3</td>
<td>131</td>
<td>36 Blind, 40 V3</td>
</tr>
<tr>
<td>Fool's Disk</td>
<td>4</td>
<td>1635 Blind, 1318 Hierarchical A* 1172 V3</td>
<td>194</td>
<td>872 Blind, 902 V3</td>
</tr>
<tr>
<td>MC 60-40-7</td>
<td>4</td>
<td>934 Blind, 822 Hierarchical A* 803 V3</td>
<td>144</td>
<td>266 Blind, 253 V3</td>
</tr>
<tr>
<td>Permute-6</td>
<td>5</td>
<td>286 Blind, 201 Hierarchical A* 194 V3</td>
<td>192</td>
<td>82 Blind, 67 V3</td>
</tr>
<tr>
<td>Words</td>
<td>3</td>
<td>1923 Blind, 9184 Hierarchical A* 1356 V3</td>
<td>128</td>
<td>1169 Blind, 1273 V3</td>
</tr>
</tbody>
</table>
Local Search

- In many optimization problems, path is irrelevant (no path cost); the goal state itself is the solution
  - 8-queens problem, job-shop scheduling
  - circuit design, computer configuration
  - automatic programming, automatic graph drawing

- Then state space = set of “complete” configurations; need to find optimal configuration

- Can use iterative improvement algorithms; keep single “current” state and try to improve it
  - Paths followed by search are not retained
    - Contrast with open and closed node lists; search tree
Advantages of local search

- Very simple to implement.
- Very little memory is needed.
- Can often find reasonable solutions in very large (continuous) state spaces for which systematic algorithms are not suitable.
Stochastic vs. Systematic Search

- Unsolvability -- Is there a solution?
  - Systematic: can require exhaustive examination of exponential search space
  - Stochastic: cannot determine unsolvability

- Completeness/Optimality
  - Systematic: complete
  - Stochastic: incomplete

- Speed
  - Neither is uniformly superior; each does better for different sorts of problems

Local Search is an example of Stochastic Search
Iterative Improvement
(Smart version of Generate & Test)

- Start Search with complete but non-optimal solution
- Modify incorrect/non-optimal solution to move it closer to correct/optimal solution

Path Cost Minimization versus Value optimization
Example: Traveling Salesperson Problem

- Start with any complete tour and perform pair wise exchanges of the end points of two segments.

- Only make change if exchange reduces tour cost.

- Variants of this approach get within 1% of optimal very quickly with thousands of cities.
Next Lecture

- Continuation of Local Search
  - Hill-Climbing/Iterative Improvement
    - Simulated Annealing (Stochastic Hill Climbing)
  - Beam Search
    - Genetic Algorithm
  - Repair/Debugging (to be done next time)
    - GSAT