Incremental LM-cut

Florian Pommerening and Malte Helmert

Universität Basel
Departement Informatik

14. 06. 2013
Content

1. **Theory**
   - Planning
   - Incremental Computation

2. **Practice**

3. **Conclusion**
Heuristic Search for STRIPS Planning

- Variables $V = \{A\text{-Is-Open}, B\text{-Is-Open}, \ldots\}$
- Initial state $I = \{A\text{-Is-Open}, G\text{-Is-Open}\}$
- Goal state $G = \{\text{Has-}T_1, \text{Has-}T_2, \text{Has-}T_3\}$
- Operators $O = \{o_A, o_B, \ldots, p_1, p_2, \ldots\}$
- Plan $\pi = \langle p_1, o_F, p_2, o_B, o_C, p_3 \rangle$

- **Heuristic function**
  - Distance estimate
  - Admissibility

- **Search methods**
  - A* Search
  - IDA* Search
Disjunctive Action Landmarks

- **Disjunctive action landmarks**
  - Set of operators
  - Every plan contains at least one of them
  - Cost of a landmark: cost of cheapest contained operator

- Landmarks in this state
  - \( \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}, \ldots \)
1. Calculate $h^{\text{max}}(s)$
   - Only achieve most expensive subgoal/precondition
   - $h^{\text{max}}(s) = \infty$ task unsolvable
   - $h^{\text{max}}(s) = 0$ stop searching for landmarks

2. Use $h^{\text{max}}$ values to discover new landmark $L$

3. Reduce cost of each operator in $L$ by $L$'s cost
   - Introduces a cost partitioning
   - Sum of landmark costs is admissible heuristic

4. Repeat
Incremental Computation (Example)

- Discovered landmarks in this state
  - $\{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}$
- Landmarks after application of $o_B$?
Incremental Computation (Example)

- Discovered landmarks in this state
  - \( o_A \), \( o_B \), \( o_C \), \( o_D \), \( o_E \), \( o_F \)
- Landmarks after application of \( o_B \)
  - \( o_A \) remains landmark
Incremental Computation (Example)

- Discovered landmarks in this state
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}

- Landmarks after application of o_B
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\} remain landmarks
Incremental Computation (Example)

- Discovered landmarks in this state
  - $\{a_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}$

- Landmarks after application of $o_B$
  - $\{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}$ remain landmarks
  - Newly discovered landmark: $\{o_C, o_D\}$
Incremental Computation

- Successor generated by applying operator $o$
  - All landmarks not containing $o$ are landmarks in successor
  - Discharge landmarks containing $o$
    - Return landmark’s costs to *remaining cost*
    - Can change $h^{\text{max}}$ value
  - Start LM-cut algorithm with set of known landmarks

Theorem
The LM-cut algorithm discovers a new landmark if the $h^{\text{max}}$ cost of the successor increases.
Incremental Computation (Example)

- Landmarks in this state:
  - \(\{o_A\}\), \(\{o_B, o_C, o_D\}\), \(\{o_E, o_F\}\)
Incremental Computation (Example)

- Landmarks in this state
  - \( \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\} \)
- Open door A
- Open door B
- Open door E
- Open door F
Incremental Computation (Example)

- Landmarks in this state
  - \( \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\} \)
- Open door A
  - \( \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\} \)
- Open door B
- Open door E
- Open door F
Incremental Computation (Example)

- Landmarks in this state
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door A
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door B
- Open door E
- Open door F

Pommerening, Helmert
Incremental LM-cut
14. 06. 2013
Incremental Computation (Example)

- Landmarks in this state
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door A
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door B
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door E
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door F
Landmarks in this state
- \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}

Open door A
- \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}

Open door B
- \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}

Open door E
- \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}

Open door F
Incremental Computation (Example)

- Landmarks in this state
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door A
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door B
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}, \{o_C, o_D\}
- Open door E
- Open door F
Incremental Computation (Example)

- Landmarks in this state
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door A
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door B
  - \{o_A, o_B, o_C, o_D\}, \{o_E, o_F\}, \{o_C, o_D\}
- Open door E
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door F
Incremental Computation (Example)

- Landmarks in this state
  - \( \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\} \)
- Open door A
  - \( \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\} \)
- Open door B
  - \( \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}, \{o_C, o_D\} \)
- Open door E
  - \( \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\} \)
- Open door F
Incremental Computation (Example)

- Landmarks in this state
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door A
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door B
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}, \{o_C, o_D\}
- Open door E
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}, \{o_F, o_G\}
- Open door F
Incremental Computation (Example)

- Landmarks in this state:
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door A:
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door B:
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}, \{o_C, o_D\}
- Open door E:
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}, \{o_F, o_G\}
- Open door F:
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
Incremental Computation (Example)

- Landmarks in this state
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door A
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
- Open door B
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}, \{o_C, o_D\}
- Open door E
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}, \{o_C, o_D\}
- Open door F
  - \{o_A\}, \{o_B, o_C, o_D\}, \{o_E, o_F\}
1 Theory

2 Practice
   - Basic Results
   - Saving Memory

3 Conclusion
Evaluation
- 1396 tasks in 44 domains
- Time limit: 30 min
- Memory limit: 2 GB

Measured
- Coverage (number of solved tasks)
- Search time
- Failure reason (timeout or out of memory)
Basic Results

- First idea ($h^iLM$-cut)
  - Regular A* search
  - Store landmarks for all search nodes
  - (Side note: This makes the heuristic consistent)
- Compare with $A^*/h^{LM}$-cut
  - Speed-ups by up to an order of magnitude
  - More tasks running out of memory
    - Increased from 31 to 526

### Coverage

<table>
<thead>
<tr>
<th>$h^{LM}$-cut</th>
<th>$h^iLM$-cut</th>
</tr>
</thead>
<tbody>
<tr>
<td>757</td>
<td>762</td>
</tr>
</tbody>
</table>
Removing Landmarks

- Stored landmarks take up too much memory
  - Information can be removed at any time
  - Recompute missing values non-incrementally
- Easy fix ($h^i_{\text{LM-cut}}$)
  - Remove information for closed nodes
  - Only needed again if the node is reopened
Results

- Runtime reduction over baseline
  - 77% (geometric mean)
  - 93% (miconic ×)
- Coverage increased by another 4 tasks

### Coverage

<table>
<thead>
<tr>
<th>$h^{LM}$-cut</th>
<th>$h^{iLM}$-cut</th>
<th>$h^{iLM}$-cut frontier</th>
</tr>
</thead>
<tbody>
<tr>
<td>757</td>
<td>762</td>
<td>766</td>
</tr>
</tbody>
</table>

![Graph showing comparison of $A^*$/$h^{LM}$-cut and $A^*$/$h^{iLM}$-cut frontier]
Fixed Memory Bounds

- Why stop because of exhausted memory at all?
  - Can always free up memory
  - Remove stored landmarks

- Fixed memory bound ($h_{\text{bound}}$)
  - Keep track of used memory
  - Remove half of the stored landmarks when hitting the bound

- Dynamic memory bound
  - Technical problems
    - Measure memory pressure and memory requirements accurately
  - Results estimated from fix bounds
Results

- Improvements even for small bounds (50 MB)
- Increasing limit
  - Less timeouts (■)
  - Memory exhausted more often (□)
- Sweet spot for 500 MB

<table>
<thead>
<tr>
<th>Coverage</th>
<th>h_{LM-cut}</th>
<th>h^{iLM-cut}</th>
<th>h^{iLM-cut}_{frontier}</th>
<th>h^{iLM-cut}_{500 MB}</th>
<th>h^{iLM-cut}_{dynamic}</th>
</tr>
</thead>
<tbody>
<tr>
<td>757</td>
<td>762</td>
<td>766</td>
<td>778</td>
<td>786</td>
<td></td>
</tr>
</tbody>
</table>
Local Incremental Computation

- More generated nodes than expanded nodes
  - Factor 8 in the geometric mean
  - Save work during node generation
  - Additional work during node expansion amortized
- Local incremental computation ($h_{loc}^{LM-cut}$)
  - Recompute landmarks for parent node
  - Incremental computation for child nodes
  - Minimal memory overhead

<table>
<thead>
<tr>
<th>LM-cut Computations</th>
<th>$h^{LM-cut}$</th>
<th>$h_{loc}^{LM-cut}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node generation</td>
<td>Non-incremental</td>
<td>Incremental</td>
</tr>
<tr>
<td>Node expansion</td>
<td>0</td>
<td>Non-incremental</td>
</tr>
</tbody>
</table>
Results

- Runtime changes
  - $-49\%$ (geometric mean)
  - $-89\%$ (miconic $\times$)
  - $+40\%$ (openstacks $\times$)
- Bad performance of openstacks
  - $h^{\text{LM-cut}}(s) = 1$ for most states $s$
  - Many 0-cost operators

### Coverage

<table>
<thead>
<tr>
<th>$h^{\text{LM-cut}}$</th>
<th>$h^{\text{iLM-cut}}$</th>
<th>$h^{\text{iLM-cut}_{\text{frontier}}}$</th>
<th>$h^{\text{iLM-cut}_{\text{500 MB}}}$</th>
<th>$h^{\text{iLM-cut}_{\text{dynamic}}}$</th>
<th>$h^{\text{iLM-cut}_{\text{local}}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>757</td>
<td>762</td>
<td>766</td>
<td>778</td>
<td>786</td>
<td>783</td>
</tr>
</tbody>
</table>
**IDA***

- **IDA*** Search
  - Classical solution to memory issues with A***

- Here: Sufficient memory for all search nodes
  - Use unlimited transposition table
  - Perfect duplicate detection
  - Store heuristic values for inner nodes

- Remaining advantage over A***
  - Depth-first expansion order
  - Only store landmarks for current branch
Results

- Best Coverage result
- Skewed by openstacks (×)
  - Better tie-breaking with depth-first order
  - Deeper nodes are preferred
  - Explains 14 tasks

<table>
<thead>
<tr>
<th>Coverage</th>
<th>A*</th>
<th>IDA*</th>
</tr>
</thead>
<tbody>
<tr>
<td>hLM-cut</td>
<td>hLIM-cut</td>
<td>hLIM-cut frontier</td>
</tr>
<tr>
<td>757</td>
<td>762</td>
<td>766</td>
</tr>
</tbody>
</table>

A* / hLM-cut vs. uns.
Conclusion

- Incremental computation of LM-cut for STRIPS planning
  - Much faster
  - Higher memory requirements

- Dealing with memory limitations
  - Local computation
  - Fixed bounds
  - IDA*

- Not limited to LM-cut

- Necessary conditions
  - Incremental computation is faster
  - Missing information can be computed non-incrementally
Thank you for your attention!
Any questions?