Better Orders for Saturated Cost Partitioning in Optimal Classical Planning

Jendrik Seipp

University of Basel

June 17, 2017
Setting

- optimal classical planning
- A* search + admissible heuristic
- abstraction heuristics
Setting

- optimal classical planning
- A* search + admissible heuristic
- abstraction heuristics
• single heuristic unable to capture enough information
Problem

- single heuristic unable to capture enough information
  → use multiple heuristics
Problem

- single heuristic unable to capture enough information
  → use multiple heuristics
- how to combine multiple heuristics admissibly?
Multiple Heuristics

- $s_1$, $s_2$, $s_3$, $s_4$
- $o_1$, $o_2$, $o_3$

- $h_1(s_1) = 1$
- $h_2(s_1) = 1$

• maximizing only selects best heuristic

- $h(s_1) = 1$
Multiple Heuristics

\[ h_1(s_1) = 1 \]

\[ h_2(s_1) = 1 \]
Multiple Heuristics

- \( h_1(s_1) = 1 \)
- \( h_2(s_1) = 1 \)

- maximizing only selects best heuristic \( \rightarrow h(s_1) = 1 \)
Cost Partitioning

- split operator costs among heuristics
- total costs must not exceed original costs

→ combines heuristics
→ allows summing heuristic values admissibly
Saturated Cost Partitioning Algorithm

- order heuristics
- for each heuristic \( h \):
  - use minimum costs preserving all estimates of \( h \)
  - use remaining costs for subsequent heuristics

Diagram:

- Nodes labeled with \( s_1, s_2, s_3, s_4 \)
- Arrows indicate the order of application
- \( o_1, o_2, o_3 \) represent the order in which heuristics are applied
Saturated Cost Partitioning Algorithm

- order heuristics
- for each heuristic $h$:
  - use minimum costs preserving all estimates of $h$
  - use remaining costs for subsequent heuristics

$$h^{SCP}_{\langle h_1, h_2 \rangle}(s_1) = 1 + 1 = 2$$
**Saturated Cost Partitioning Algorithm**

- **order** heuristics
- for each heuristic $h$:
  - use minimum costs preserving all estimates of $h$
  - use remaining costs for subsequent heuristics

**Example:**

\[
\begin{align*}
\text{SCP}_{\langle h_1, h_2 \rangle}(s_1) &= 1 + 1 = 2 \\
\text{SCP}_{\langle h_2, h_1 \rangle}(s_1) &= 1 + 0 = 1
\end{align*}
\]
Orders for Saturated Cost Partitioning
Seipp, Keller & Helmert (AAAI 2017)

- orders can be arbitrarily bad
  → hill climbing in space of orders
- a single order might only be good for a single state
  → use multiple orders
- using too many orders slows down evaluation
  → use diverse orders
<table>
<thead>
<tr>
<th>ICAPS 2017</th>
<th>$h^\text{SCP}_{\text{HC}}$</th>
<th>$h^\text{SCP}_{\text{Sys}}$</th>
<th>$h^\text{SCP}_{\text{Cart}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tasks</strong></td>
<td>805</td>
<td>852</td>
<td>965</td>
</tr>
</tbody>
</table>

*(1667)*
Combining Heterogeneous Abstraction Heuristics

<table>
<thead>
<tr>
<th>ICAPS 2017</th>
<th>( h_{SCP}^{HC} )</th>
<th>( h_{SCP}^{Sys} )</th>
<th>( h_{SCP}^{Cart} )</th>
<th>( h_{SCP}^{HC, Sys, Cart} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tasks</strong></td>
<td>1667</td>
<td>805</td>
<td>852</td>
<td>965</td>
</tr>
</tbody>
</table>
Combining Heterogeneous Abstraction Heuristics

<table>
<thead>
<tr>
<th>ICAPS 2017</th>
<th>$h_{SCP}^{HC}$</th>
<th>$h_{SCP}^{Sys}$</th>
<th>$h_{SCP}^{Cart}$</th>
<th>$h_{SCP}^{HC, Sys, Cart}$</th>
<th>$\max(h_{SCP}^{HC}, h_{SCP}^{Sys}, h_{SCP}^{Cart})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>805</td>
<td>852</td>
<td>965</td>
<td>1006</td>
<td>1032</td>
</tr>
</tbody>
</table>

Tasks (1667)
Combining Heterogeneous Abstraction Heuristics

ICAPS 2017

\[ \max(h_{\text{SCP,HC}}, h_{\text{SCP,Sys}}, h_{\text{SCP,Cart}}) \]

Tasks (1667) 805 852 965 1006 1032

\[ \max(\langle h_1 \rangle, \langle h'_1, h'_2 \rangle) \leq \]
\[ \max(\langle h_1, h'_1, h'_2 \rangle, \langle h'_1, h'_2, h_1 \rangle) \]
Drawbacks of Diversification

Diversification algorithm

- sample 1000 states
- start with empty set of orders
- until time limit is reached:
  - generate a random order
  - if it improves upon current set of orders, keep it
  - otherwise, discard it

→ too many orders to stumble over good ones
Drawbacks of Diversification

Diversification algorithm

• sample 1000 states
• start with empty set of orders
• until time limit is reached:
  • generate a random order
  • if it improves upon current set of orders, keep it
  • otherwise, discard it

• considers only random orders
  → too many orders to stumble over good ones
# Drawbacks of Hill Climbing

Hill climbing search

- sample 1000 states
- start with random order
- until no better successor for samples found:
  - swap positions of two heuristics
  - move to first improving successor
Drawbacks of Hill Climbing

Hill climbing search

- sample 1000 states
- start with random order
- until no better successor for samples found:
  - swap positions of two heuristics
  - move to first improving successor

- tries to find order for set of states
  → there may not be a single order for multiple states

- starts with random order
  → decent initial solution important for local optimization
Improvements

- optimize for single state
- start with greedy order
- diversify optimized greedy orders
Greedy Order

Objectives:

- increase heuristic value quickly
- preserve costs for subsequent heuristics
Greedy Order

Objectives:

- increase heuristic value quickly
- preserve costs for subsequent heuristics

Value-per-cost ratio

\[ \text{ratio}(h, s) = \frac{h(s)}{\text{saturated costs for } h} \]
Greedy Order

Objectives:
- increase heuristic value quickly
- preserve costs for subsequent heuristics

Value-per-cost ratio

\[
\text{ratio}(h, s) = \frac{h(s)}{\text{saturated costs for } h}
\]

Greedy algorithm
- start with empty order
- until all heuristics are ordered
  - append heuristic \( h \) with highest value-per-cost ratio
  - subtract saturated costs for \( h \) from overall costs
Greedy Order: Example

\[ h_1(s_1) = 1 \]

saturated costs: 1
ratio: 1

\[ h_2(s_1) = 1 \]

saturated costs: 2
ratio = 0.5

\( \langle h_1, h_2 \rangle \)
Greedy Order: Example

$h_1(s_1) = 1$

saturated costs: 1
ratio: 1

$h_2(s_1) = 1$

saturated costs: 2
ratio = 0.5
Greedy Order: Example

$h_1(s_1) = 1$
saturated costs: 1
ratio: 1

$h_2(s_1) = 1$
saturated costs: 2
ratio = 0.5

→ $\langle h_1, h_2 \rangle$
### Single Order for Initial State

<table>
<thead>
<tr>
<th></th>
<th>random</th>
<th>greedy</th>
<th>random-opt</th>
<th>greedy-opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>random</td>
<td>–</td>
<td>105</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>greedy</td>
<td>989</td>
<td>–</td>
<td>268</td>
<td>0</td>
</tr>
<tr>
<td>random-opt</td>
<td>1086</td>
<td>463</td>
<td>–</td>
<td>111</td>
</tr>
<tr>
<td>greedy-opt</td>
<td>1108</td>
<td>548</td>
<td>402</td>
<td>–</td>
</tr>
</tbody>
</table>
Diverse Orders

New diversification algorithm

Until time limit is reached:

• sample state $s$
• find greedy order for $s$
• optimize order with hill climbing
• keep order if diverse
Diverse Orders

New diversification algorithm

Until time limit is reached:

- sample state $s$
- find greedy order for $s$
- optimize order with hill climbing
- keep order if diverse

<table>
<thead>
<tr>
<th>Tasks (1667)</th>
<th>max</th>
<th>random</th>
<th>random-opt</th>
<th>greedy-opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1032</td>
<td>1006</td>
<td>1009</td>
<td>1048</td>
<td></td>
</tr>
</tbody>
</table>
Comparison to State of the Art

Using $h^2$ mutexes:

- $\text{SymBA}_2^*$ outperforms $h^{SCP}_{\text{greedy-opt}}$ in 11 domains
- $h^{SCP}_{\text{greedy-opt}}$ outperforms $\text{SymBA}_2^*$ in 22 domains
- $\text{SymBA}_2^*$ solves 1008 tasks
- $h^{SCP}_{\text{greedy-opt}}$ solves 1084 tasks
A Comparison of Cost Partitioning Algorithms for Optimal Classical Planning

- theoretical and empirical comparison of cost partitioning algorithms
- saturated cost partitioning usually method of choice for hill climbing PDBs, systematic PDBs, Cartesian abstractions and landmark heuristics

Presented on Wednesday in the first ICAPS session “Optimal Planning”
Conclusion

- new greedy algorithm for finding orders
- pair with optimization and diversification
- combine heterogeneous abstraction heuristics