Pattern Selection for Optimal Classical Planning with Saturated Cost Partitioning

Jendrik Seipp
July 11, 2019

University of Basel, Switzerland
Setting

- optimal classical planning
- A* search + admissible heuristic
- pattern databases
How to select patterns?

- bin packing (Edelkamp 2001)
- genetic algorithms (Edelkamp 2006)
- hill climbing (Haslum et al. 2007)
- CPC (Franco et al. 2017)
- CEGAR (Rovner et al. 2019)
- systematic naive (Felner et al. 2004)
- systematic (Pommerening et al. 2013)
How to combine multiple PDB heuristics?

- maximize
- cost partitioning
- saturated cost partitioning
Saturated cost partitioning

Saturated cost partitioning algorithm

• order heuristics, then for each heuristic h:
  • use minimum costs preserving all estimates of h
  • use remaining costs for subsequent heuristics
### Saturated cost partitioning algorithm

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

\[
\max(h_1(s_2), h_2(s_2)) = \max(5, 4) = 5
\]
Saturated cost partitioning

Saturated cost partitioning algorithm

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

 SCP\newline $\langle h_1; h_2 \rangle(s_2) = 5 + 3 = 8$ \newpage

4/15
Saturated cost partitioning

Saturated cost partitioning algorithm

• order heuristics, then for each heuristic h:
  • use minimum costs preserving all estimates of h
  • use remaining costs for subsequent heuristics

$SCP(h_1; h_2)(s_2) = 5 + 3 = 8$
# Saturated cost partitioning

## Saturated cost partitioning algorithm

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

<table>
<thead>
<tr>
<th>$s_1$, $s_2$</th>
<th>$s_3$</th>
<th>$s_4$, $s_5$</th>
<th>$s_1$, $s_2$, $s_3$, $s_4$</th>
<th>$s_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Saturated cost partitioning

Saturated cost partitioning algorithm

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

SCP $⟨h_1; h_2⟩(s_2) = 5 + 3 = 8\frac{4}{15}$
Saturated cost partitioning

Saturated cost partitioning algorithm

• order heuristics, then for each heuristic h:
  • use minimum costs preserving all estimates of h
  • use remaining costs for subsequent heuristics

\[
h_{\langle h_1, h_2 \rangle}^{SCP}(S_2) = 5 + 3 = 8
\]
Diverse orders for saturated cost partitioning

**Diversification algorithm**

- sample 1000 states $\hat{S}$
- start with empty set of orders
- for 200 seconds:
  - sample a new state $s$
  - find a greedy order for $s$
  - if a sample in $\hat{S}$ profits from it, keep it
  - otherwise, discard it
Idea

- select patterns
- compute diverse saturated cost partitionings over PDBs
• select patterns with saturated cost partitioning
• compute diverse saturated cost partitionings over PDBs
A new pattern selection algorithm

function $Sys-SCP(\Pi)$

$C \leftarrow \emptyset$

repeat for at most $T_x$ seconds

$\sigma \leftarrow \langle \rangle$

for $P \in Order(Sys)$ and at most $T_y$ seconds do

if $P \notin C$ and $PatternUseful(\sigma, P)$ then

$\sigma \leftarrow \sigma \oplus P$

$C \leftarrow C \cup \{P\}$

until $\sigma = \langle \rangle$

return $C$

function $PatternUseful(\sigma, P)$

return $\exists s \in S(\mathcal{T}) : h^{SCP}_\sigma(cost, s) < h^{SCP}_{\sigma \oplus P}(cost, s) < \infty$
Theorem

\[ \exists s \in S(\mathcal{T}) : h_{SCP}^{\sigma}(\text{cost}, s) < h_{\sigma \oplus P}^{SCP}(\text{cost}, s) < \infty \]

\[ \iff \exists s' \in S(\mathcal{T}_P) : 0 < h_{\mathcal{T}_P}^{*}(\text{rem}, s') < \infty \]
Using the theorem

- keep track of the remaining cost function
- select a PDB if it has positive finite goal distances
Pattern orders

order by increasing pattern size, break ties by:

- random
- states in projection
- active operators
- Fast Downward variable order:
  - up: [7, 5], [8, 2], [8, 5]
  - down: [8, 5], [8, 2], [7, 5]
Pattern orders

order by increasing pattern size, break ties by:

- random
- states in projection
- active operators
- **Fast Downward variable order:**
  - up: [7, 5], [8, 2], [8, 5]
  - down: [8, 5], [8, 2], [7, 5]
Algorithm details

- store dead ends to prune states during search
- reuse Sys-SCP pattern sequences for diversification
**Systematic patterns with limits**

**Lim**: 2M states per PDB, 20M states in collection, 100 seconds

<table>
<thead>
<tr>
<th>Max pattern size</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SYS-NAIVE</strong></td>
<td>840</td>
<td>937</td>
<td>914</td>
<td>752</td>
<td>571</td>
</tr>
<tr>
<td><strong>SYS-NAIVE-LIM</strong></td>
<td>840</td>
<td>968</td>
<td>1004</td>
<td>912</td>
<td>878</td>
</tr>
<tr>
<td><strong>SYS</strong></td>
<td>840</td>
<td>986</td>
<td>1057</td>
<td>922</td>
<td>731</td>
</tr>
<tr>
<td><strong>SYS-LIM</strong></td>
<td>840</td>
<td>985</td>
<td>1088</td>
<td>1050</td>
<td>1035</td>
</tr>
</tbody>
</table>
## Sys-SCP vs. other pattern selection algorithms

<table>
<thead>
<tr>
<th></th>
<th>HC</th>
<th>SYS-3-LIM</th>
<th>CPC</th>
<th>CEGAR</th>
<th>Sys-SCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>966</td>
<td>1088</td>
<td>1055</td>
<td>1098</td>
<td>1168</td>
</tr>
<tr>
<td>#domains Sys-SCP better</td>
<td>28</td>
<td>23</td>
<td>21</td>
<td>21</td>
<td>–</td>
</tr>
<tr>
<td>#domains Sys-SCP worse</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>–</td>
</tr>
</tbody>
</table>
Future work

• test patterns on samples
Summary

• new pattern selection algorithm based on saturated cost partitioning
• outperforms all previous pattern selection algorithms
## Pattern orders

<table>
<thead>
<tr>
<th></th>
<th>fd-up</th>
<th>states-up</th>
<th>random</th>
<th>ops-down</th>
<th>states-down</th>
<th>ops-up</th>
<th>fd-down</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd-up</td>
<td>–</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1140.0</td>
</tr>
<tr>
<td>states-up</td>
<td>6</td>
<td>–</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>1153.0</td>
</tr>
<tr>
<td>random</td>
<td>10</td>
<td>10</td>
<td>–</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>3</td>
<td>1148.2</td>
</tr>
<tr>
<td>ops-down</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>–</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>1141.0</td>
</tr>
<tr>
<td>states-down</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>–</td>
<td>4</td>
<td>2</td>
<td>1152.0</td>
</tr>
<tr>
<td>ops-up</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>11</td>
<td>11</td>
<td>–</td>
<td>6</td>
<td>1166.0</td>
</tr>
<tr>
<td>fd-down</td>
<td>12</td>
<td>10</td>
<td>12</td>
<td>10</td>
<td>9</td>
<td>6</td>
<td>–</td>
<td>1168.0</td>
</tr>
</tbody>
</table>