Explicit-State Abstraction: A New Method for Generating Heuristic Functions

Malte Helmert\textsuperscript{1} Patrik Haslum\textsuperscript{2} Jörg Hoffmann\textsuperscript{3}

\textsuperscript{1}Albert-Ludwigs-Universität Freiburg, Germany
\textsuperscript{2}NICTA & Australian National University, Australia
\textsuperscript{3}University of Innsbruck, Austria

AAAI 2008, Nectar track
Abstraction heuristics

Heuristic estimate is **goal distance in abstracted state space** $S'$ obtained as **homomorphism** of original state space $S$. 
Abstraction heuristics

Heuristic estimate is goal distance in abstracted state space $S'$ obtained as homomorphism of original state space $S$.

Explicit-state abstraction heuristics

You have seen other abstraction heuristics before; they are called pattern database heuristics.

Ours can do the same and then some.
Transition Graphs

Definition (transition graph)

A transition graph is a 5-tuple $\langle S, L, A, s_0, S_\star \rangle$:

- $S$: finite set of states
- $L$: finite set of transition labels
- $A \subseteq S \times L \times S$: labelled transitions
- $s_0 \in S$: initial state
- $S_\star \subseteq S$: goal states

Assumption: States are assignments to a set of state variables.
Running Example

Logistics problem with one package, two trucks, two locations:

- state variable package: \( \{L, R, A, B\} \)
- state variable truck A: \( \{L, R\} \)
- state variable truck B: \( \{L, R\} \)
Abstractions

Definition (abstraction, homomorphism)

Abstraction of transition graph $\mathcal{T}$: pair $\langle \mathcal{T}', \alpha \rangle$ where

- $\mathcal{T}'$ is a transition graph with the same labels
- $\alpha$ maps states of $\mathcal{T}$ to states of $\mathcal{T}'$ such that
  - initial state maps to initial state
  - goal states map to goal states
  - transitions $\langle s, l, s' \rangle$ map to transitions $\langle \alpha(s), l, \alpha(s') \rangle$

Abstraction (and $\alpha$) is a homomorphism if $\mathcal{T}'$ only contains necessary goal states and transitions.

Abstraction heuristic: $h(s) = d_*(\alpha(s))$ admissible, consistent
Example: Perfect Abstraction

⇝ perfect heuristic $h^*$
Generating Abstractions

Conflicting goals in generating abstractions:
- obtain informative heuristic
- keep representation small

Abstractions have small representations if they have
- few abstract states
- succinct encoding for $\alpha$
Projections

One idea to get succinct encodings: projections
\[ \rightsquigarrow \text{map states to abstract states with perfect hash function} \]

**Definition (projection)**

Projection \( \pi_{\mathcal{V}'} \) to variables \( \mathcal{V}' \subseteq \mathcal{V} \): homomorphism \( \alpha \) where

\[ \alpha(s) = \alpha(s') \text{ iff } s \text{ and } s' \text{ agree on } \mathcal{V}' \]

shorthand for *atomic projections*: \( \pi_v := \pi_{\{v\}} \ (v \in \mathcal{V}) \)
Example: Projection (1)

Project to \{\text{package}\}:
Example: Projection (2)

Project to \{\text{package, truck A}\}:
Example: Projection (2)

Project to \{\text{package, truck A}\}:
Problems of Projections

- abstraction heuristics for projections are pattern database (PDB) heuristics
- must keep number of reflected variables (pattern) small

price in heuristic accuracy:

- consider generalization of running example:
  \( N \) trucks, \( M \) locations (still one package)
- consider any pattern that is proper subset of \( \mathcal{V} \)
- \( h(s_0) \leq 2 \) \( \rightarrow \) no better than atomic projection to package

(maximizing over patterns or additive patterns do not help either)
Outline

1. Abstractions
2. Projections
3. Explicit-State Abstractions
4. Evaluation
5. Conclusion
Main idea

(due to Dräger, Finkbeiner & Podelski, 2006):

Instead of perfectly reflecting a few state variables, reflect all state variables, but in a potentially lossy way.
Explicit-State Abstraction Heuristics: Key Insights

Key insights:

1. Information of two abstractions $\mathcal{A}$ and $\mathcal{A}'$ of the same transition system can be composed by a simple graph-theoretic operation (synchronized product $\mathcal{A} \otimes \mathcal{A}'$).

2. Under suitable conditions (factored transition systems), the complete state space can be recovered using only atomic projections:

$$\bigotimes_{v \in \mathcal{V}} \pi_v \text{ is isomorphic to } \pi_{\mathcal{V}}.$$ 

$\leadsto$ build fine-grained abstractions from coarse ones

3. When intermediate results become too big, we can shrink them by aggregating some abstract states.
Computing Explicit-State Abstractions

Generic abstraction computation algorithm

\[
\text{abs} := \text{all atomic projections } \pi_v \ (v \in \mathcal{V}).
\]

while abs contains more than one abstraction:

\[
\text{select } A_1, A_2 \text{ from abs}
\]

shrink \( A_1 \) and/or \( A_2 \) until \( \text{size}(A_1) \cdot \text{size}(A_2) \leq N \)

\[
\text{abs} := \text{abs} \setminus \{A_1, A_2\} \cup \{A_1 \otimes A_2\}
\]

return the remaining abstraction

\( N \): parameter bounding number of abstract states

Questions for practical implementation:

- Which abstractions to select? \( \leadsto \) composition strategy
- How to shrink an abstraction? \( \leadsto \) shrinking strategy
- How to choose \( N \)?
Outline

1. Abstractions
2. Projections
3. Explicit-State Abstractions
4. Evaluation
5. Conclusion
Guiding Questions for Evaluation

Comparison to state of the art

*Is this competitive with the state of the art?*

- Compare scaling behaviour to other heuristics: blind, $h_{\text{max}}$, PDB

⇒ next slide

Comparison to pattern databases

*How does this compare to well-chosen PDB heuristics?*

- compare to approach of Haslum et al. (2007)
- compare scaling behaviour and runtime
- compare heuristic quality, preprocessing time, search time

⇒ details in the ICAPS 2007 paper
### Comparison to state of the art

**Is this competitive with the state of the art?**
- Compare scaling behaviour to other heuristics: blind, $h_{\text{max}}$, PDB

<table>
<thead>
<tr>
<th>Domain</th>
<th>abs</th>
<th>blind</th>
<th>$h_{\text{max}}$</th>
<th>PDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPES-NO TANKAGE</td>
<td>19</td>
<td>14</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>PIPES-TANKAGE</td>
<td>13</td>
<td>10</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>SATELLITE</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>LOGISTICS</td>
<td>18</td>
<td>6</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>PSR</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>TPP</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>68</strong></td>
<td><strong>42</strong></td>
<td><strong>46</strong></td>
<td><strong>54</strong></td>
</tr>
</tbody>
</table>
Comparison to Pattern Databases: Theory

As powerful as PDBs

PDB heuristics are a special case of our abstraction heuristics, and arise naturally as a side product.

Get additivity for free

If $P$ and $P'$ are additive patterns, then for all $h$-preserving abstractions $A$ of $\pi_P$ and $A'$ of $\pi_{P'}$, the abstraction heuristic for $A \otimes A'$ dominates $h^P + h^{P'}$.

Greater representational power

In some planning domains where PDBs have unbounded error (Gripper, Schedule, two Promela variants), we can obtain perfect heuristics in polynomial time with suitable composition/shrinking strategies.
Conclusion

Summary

- clean, flexible approach to computing heuristics
- works very well for planning and model checking

Future work:

- more theory
- more experiments
- more informed abstraction strategies
- comparison of abstraction strategies
- determine/adjust abstraction size dynamically