Where Do Heuristics Come From?
Part 3

Robert C. Holte
Computing Science Department
University of Alberta

© 2005, Robert Holte

Outline

Part 1: Introduction
Part 2: Details
Part 3: Pattern Database Enhancements
  – Taking the maximum of two or more PDBs
  – Compression and Dovetailing of PDBs
  – Additive PDBs
  – Customized PDBs
  – Multiple Lookups in One PDB
Bonus! – Related Algorithm (CFPD)

Max’ing Multiple Heuristics

• Given heuristics h1 and h2 define
  \[ h(s) = \max (h1(s), h2(s)) \]

• Preserves key properties:
  – lower bound
  – consistency

Question

• Given a fixed amount of memory, M, which gives the best heuristic?
  – 1 pattern database (PDB) of size M
  – max’ing 2 PDBs of size M/2
  – max’ing 3 PDBs of size M/3
  – etc.
1 large pattern database

2 half-size pattern databases

Many small pattern databases

Rubik’s Cube*

<table>
<thead>
<tr>
<th>PDB Size</th>
<th>n</th>
<th>Nodes Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>13,305,600</td>
<td>8</td>
<td>2,654,689</td>
</tr>
<tr>
<td>17,740,800</td>
<td>6</td>
<td>2,639,969</td>
</tr>
<tr>
<td>26,611,200</td>
<td>4</td>
<td>3,096,919</td>
</tr>
<tr>
<td>53,222,400</td>
<td>2</td>
<td>5,329,829</td>
</tr>
<tr>
<td>106,444,800</td>
<td>1</td>
<td>61,465,541</td>
</tr>
</tbody>
</table>

* “easy” problems
**Summary**

<table>
<thead>
<tr>
<th>State Space</th>
<th>Best n</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3x3)-puzzle</td>
<td>10</td>
<td>3.85</td>
</tr>
<tr>
<td>9-pancake</td>
<td>10</td>
<td>8.59</td>
</tr>
<tr>
<td>(8,4)-Topspin (3 ops)</td>
<td>9</td>
<td>3.76</td>
</tr>
<tr>
<td>(8,4)-Topspin (8 ops)</td>
<td>9</td>
<td>20.89</td>
</tr>
<tr>
<td>(3x4)-puzzle</td>
<td>21+</td>
<td>185.5</td>
</tr>
<tr>
<td>Rubik’s Cube</td>
<td>6</td>
<td>23.28</td>
</tr>
<tr>
<td>15-puzzle (additive)</td>
<td>5</td>
<td>2.38</td>
</tr>
<tr>
<td>24-puzzle (additive)</td>
<td>8</td>
<td>1.6 to 25.1</td>
</tr>
</tbody>
</table>

RATIO = \[
\frac{\text{#nodes generated using one PDB of size } M}{\text{#nodes generated using n PDBs of size } M/n}
\]

**Rubik’s Cube CPU Time**

<table>
<thead>
<tr>
<th>#PDBs</th>
<th>Nodes Ratio</th>
<th>Time Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>23.15</td>
<td>12.09</td>
</tr>
<tr>
<td>6</td>
<td>23.28</td>
<td>14.31</td>
</tr>
<tr>
<td>4</td>
<td>19.85</td>
<td>13.43</td>
</tr>
<tr>
<td>2</td>
<td>11.53</td>
<td>9.87</td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Time/node is 1.67x higher using six PDBs.

**Techniques for Reducing the Overhead of Multiple PDB lookups**

**Early Stopping**

IDA* depth bound = 7  
g(s) = 3  
⇒ Stop doing PDB lookups as soon as h > 4 is found.

**Might result in extra IDA* iterations**

PDB₁(s) = 5  ⇒ next bound is 8  
PDB₂(s) = 7  ⇒ next bound is 10
Consistency-based Bounding

Because of consistency:
- \( \text{PDB}_1(B) \leq 2 \)
- \( \text{PDB}_2(B) \geq 6 \)
- No need to consult \( \text{PDB}_1 \)

Experimental Results

- 15-puzzle, five additive PDBs (7-7-1)
  - Naïve: 0.15 secs
  - Early Stopping: 0.10 secs

- Rubik’s Cube, six non-additive PDBs
  - Naïve: 27.125 secs
  - Early Stopping: 8.955 secs
  - Early Stopping and Bounding: 8.836 secs

Why Does Max’ing Speed Up Search?

Static Distribution of Heuristic Values

max of 5 small PDBs.

1 large PDB.
2.38x nodes generated

15-puzzle, 100M states.
**Runtime Distribution of Heuristic Values**

![Graph showing runtime distribution of heuristic values](image)

**Example of Max Failing**

<table>
<thead>
<tr>
<th>Depth Bound</th>
<th>h1</th>
<th>h2</th>
<th>max(h1,h2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>19</td>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>36</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>59</td>
<td>78</td>
<td>43</td>
</tr>
<tr>
<td>11</td>
<td>110</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>142</td>
<td>188</td>
<td>96</td>
</tr>
<tr>
<td>13</td>
<td>269</td>
<td></td>
<td>124</td>
</tr>
<tr>
<td>14</td>
<td>440</td>
<td>530</td>
<td>314</td>
</tr>
<tr>
<td>15</td>
<td>801</td>
<td></td>
<td>400</td>
</tr>
<tr>
<td>16</td>
<td>1,045</td>
<td>1,348</td>
<td>816</td>
</tr>
<tr>
<td>17</td>
<td>1,994</td>
<td></td>
<td>949</td>
</tr>
<tr>
<td>18</td>
<td>2,679</td>
<td>3,622</td>
<td>2,056</td>
</tr>
<tr>
<td>19</td>
<td>5,480</td>
<td></td>
<td>2,435</td>
</tr>
<tr>
<td>20</td>
<td>1,197</td>
<td>1,839</td>
<td>820</td>
</tr>
<tr>
<td>TOTAL</td>
<td>5,581</td>
<td>16,312</td>
<td>8,132</td>
</tr>
</tbody>
</table>

**Approaches**

- Compress an individual Pattern Database
  - Lossless compression
  - Lossy compression must maintain admissibility
  - Allows you to
    - use a PDB bigger than will fit in memory
    - use multiple PDBs instead of just one

- Merge two PDBs into one the same size
  - Culberson & Schaeffer’s dovetailing
Compression Results

- 16-disk 4-peg TOH, PDB based on 14 disks
  - No compression: 256Megs memory, 14.3 secs
  - Lossless compression: 256k memory, 23.8 secs
  - Lossy compression: 96Megs, 15.9 secs

- 15-puzzle, additive PDB triple (7-7-1)
  - No compression: 537Megs memory, 0.069 secs
  - Lossy compression, **two** PDB triples
    537Megs memory, 0.021 secs

Dovetailing

- Given 2 PDBs for a state space construct a hybrid containing some entries from each of them, so that the total number of entries is the same as in one of the originals.

  - The hope: almost as good as max, but only half the memory.

Dovetailing based on the blank

Any “colouring” is possible
Dovetailing – selection rule

- Dovetailing requires a rule that maps each state, s, to one of the PDBs. Use that PDB to compute \( h(s) \).
- Any rule will work, but they won’t all give the same performance.
- Intuitively, strict alternation between PDBs expected to be almost as good as max.

Dovetailing compared to Max’ing

Experimental Results

- Culberson & Schaeffer (1994):
  - Dovetailing two PDBs reduced \#nodes generated by a factor of 1.5 compared to using either PDB alone
- Holte & Newton (unpublished):
  - Dovetailing halved \#nodes generated on average

How to generalize Dovetailing to any abstractions of any space?
A Partial-Order on Domain Abstractions

- Easy to enumerate all possible domain abstractions
  
  Domain = blank 1 2 3 4 5 6 7 8  
  Abstract = blank □ □ □ □ □ □ □ □ 

- and to define a partial-order on them, e.g.
  
  Domain = blank 1 2 3 4 5 6 7 8  
  Abstract = blank □ □ □ □ □ □ □ □ 

is “more abstract” than the domain abstraction above.

The “LCA” of 2 Abstractions

Domain = blank 1 2 3 4 5 6 7 8  
Abstract = blank □ □ □ □ □ □ □ □ 

LCA = least-abstract common abstraction

General Dovetailing

- Given PDB₁ and PDB₂ defined by φ₁ and φ₂
- Find a common abstraction φ of φ₁ and φ₂
- Because it is a common abstraction there exist φ₁ and φ₂ such that φ₁ ∩ φ₂ = φ  
  φ₁ = φ₂ = φ
- For every pattern, p, defined by φ, set
  SELECT[p] = φ₁ or φ₂
- Keep every entry (p_k, h) from PDBᵢ for which
  SELECT[φₛ(p_k)] = i.
- Given state s
  1. φₛ = SELECT[φₛ(s)]
  2. h(s) = PDB[φₛ (s)]
Additive Pattern Databases

Adding instead of Max’ing

- Under some circumstances it is possible to add the values from two PDBs instead of just max’ing them and still have an admissible heuristic.

- This is advantageous because*
  \[ h_1(s) + h_2(s) \geq \max(h_1(s), h_2(s)) \]

* but see slide “Compared to Max’ing”

Manhattan Distance Heuristic

For a sliding-tile puzzle, Manhattan Distance looks at each tile individually, counts how many moves it is away from its goal position, and adds up these numbers.

\[
\text{MD}(s) = 2 + 1 + 2 = 5
\]

M.D. as Additive PDBs (1)

\[
\phi_1(x) = \begin{cases} 
  x & \text{if } x = 1 \\
  \text{blank} & \text{otherwise}
\end{cases}
\]

\[
\phi_1(\text{goal}) = 1 \quad \phi_1(s) = 1
\]

\[
\text{MD}(s) = \text{PDB}_1[\phi_1(s)] + \text{PDB}_2[\phi_2(s)] + \text{PDB}_3[\phi_3(s)]
\]

\[
\text{PDB}_1[\phi_1(s)] = 2
\]
**In General...**

Partition the tiles in groups, $G_1, G_2, \ldots G_k$

$$\phi_i(x) = \begin{cases} 
    x & \text{if } x \in G_i \\
    \text{blank} & \text{otherwise}
\end{cases}$$

**Korf & Felner’s Method**

Partition the tiles in groups, $G_1, G_2, \ldots G_k$

$$\phi_i(x) = \begin{cases} 
    x & \text{if } x \in G_i \\
    \text{blank} & \text{if } x = \text{blank} \\
    \text{otherwise}
\end{cases}$$

Moves of cost zero

**What’s the Difference?**

- The blank cannot reach this position without disturbing tile 1 or tile 2.

**Compared to Max’ing**

- If the PDBs were going to be max’d instead of added, we would count all the moves in all the PDBs.
- Therefore the PDBs for adding have smaller entries than the corresponding PDBs for max’ing.
- In initial experiments on the 15-puzzle, max’ing returns a higher value than adding for about 12% of the states.
Max’ing After Adding

8-7 Partition
(576 million entries)

7-7-1 Partition
(115 million entries)

Max’ing after Adding

• For a given 7-7-1 partition, look up the 3 values and add them.

• Do this for each of the five 7-7-1 partitions and take the maximum*.

Also compute Manhattan Distance, and use that if it is largest of all. This was also done for 8-7.
15-Puzzle Results

<table>
<thead>
<tr>
<th>Partition</th>
<th>n</th>
<th>Nodes Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-7-1</td>
<td>5</td>
<td>57,159</td>
</tr>
<tr>
<td>8-7</td>
<td>1</td>
<td>136,228</td>
</tr>
</tbody>
</table>

58% reduction in nodes generated, but only 10% reduction in CPU time.

Customized PDBs

Space-Efficient PDBs

- Zhou & Hansen (AAAI, 2004)
  - Do not generate PDB entries that are provably not needed to solve the given problem.
  - Prune abstract state A if \( f(A) > U \), where U is an upper bound on the solution cost at the base level.
- To work well, needs a heuristic to guide the abstract search and a fairly tight U.
- Even then requires significantly more memory than Hierarchical IDA*.

Reverse Resumable A*

- Silver, 2005
- Aims to minimize the number of PDB entries
  - Backward search from abstract goal stops when abstract start is reached
  - If \( h(x) \) is needed and has not been computed, resume the abstract search until you get it.
- Requires abstract Open and Closed lists.
Super-Customization

• If customizing an abstraction for a given start state is a good idea, wouldn’t it be even better to change abstractions in the middle of the search space to exploit local properties?
• This does pay off sometimes, even for PDBs:
  – Hernadvolgyi (2003; also PhD thesis, chapter 5)

Multiple Lookups in One Pattern Database

Use Symmetries

distance(Pos3,c’) = distance(mirror(Pos3),c)

Example

Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank 1 □ □ □ □ □ □ □ □

state normal PDB lookup mirror lookup
“Dual” PDB Lookups

<table>
<thead>
<tr>
<th>State</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 5</td>
<td>1 2</td>
</tr>
<tr>
<td>6 1 8</td>
<td>3 4 5</td>
</tr>
<tr>
<td>3 4 7</td>
<td>6 7 8</td>
</tr>
</tbody>
</table>

distance?

Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank 1 0 0 0 0 0 0 0

Standard PDB lookup

<table>
<thead>
<tr>
<th>Abstract State</th>
<th>Abstract Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0 0 0 0</td>
<td>1 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abstract State</th>
<th>Abstract Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0 0 0 0</td>
<td>1 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

“Dual” lookup, same PDB

<table>
<thead>
<tr>
<th>Abstract State</th>
<th>Abstract Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

6

Relevance?

<table>
<thead>
<tr>
<th>Relevant to the original state?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 5 6 1 8 3 4 7</td>
</tr>
</tbody>
</table>

Why is this lookup relevant to the original state?
In a PDB for tile 2, this lookup is relevant to the original state.

Two Key Properties

1. Distances are Symmetric
2. Distances are tile-independent

Third Key Property

(3) Can determine which tiles in the given state correspond to the key tiles in the goal state.

Fourth Key Property

(3) The tiles that correspond to the key tiles in the goal state occur in the goal state.
Experimental Results

- 16-disk, 4-peg TOH, PDB of 14 disks
  - Normal: 72.61 secs
  - Only the “dual” lookup: 3.31 secs
  - Both lookups: 1.61 secs

- 15-puzzle, additive PDB (8-7)
  - Normal: 0.034 secs
  - Only the “dual” lookup: 0.076 secs
  - Both lookups: 0.022 secs

Dual Not Always Consistent

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>goal (7-pancake puzzle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

abstract distance = 0

<table>
<thead>
<tr>
<th>5</th>
<th>7</th>
<th>6</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>abstract distance = 3</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>abstract distance = 3</th>
</tr>
</thead>
</table>

Bidirectional Pathmax

\[ h(A) = 6 \]
\[ h(B) = 7 \]
\[ h(C) = 5 \]

Related Algorithm – CFPD
CFDP

- Coarse-to-Fine Dynamic Programming
- Works on continuous or discrete spaces.
- Most easily explained if space is a trellis (level structure).
- Abstraction = grouping states on the same level.
- Multiple levels of abstraction.
- Resembles refinement, but guaranteed to find optimal solution.
- Application: finding optimal convex region boundaries in an image.

CFDP - Example

CFDP – Coarsest States

CFDP – Abstract Edges
CFDP – Refine Optimal Path

Optimal solution

More, all > 30 at coarsest level

CFDP – Refine Optimal Path

Optimal solution

More, all > 30 at coarsest level

CFDP – Refine Again

Optimal solution

More, all > 30 at coarsest level

CFDP – Final Iteration

Final solution

More, all > 30 at coarsest level