What is a Pattern Database?

- PDB = a heuristic stored as a lookup table
- Invented by Culberson and Schaeffer (1994)
- created by “abstracting” the state space

Key properties:
- guaranteed to be a lower bound
- guaranteed to be “consistent”
- the bigger the better (as a general rule)

Success Story #1

- 15-puzzle ($10^{13}$ states).
- 2 hand-crafted patterns (“fringe” (FR) and “corner” (CO))
- Each PDB contains >500 million entries
- Used symmetries to compress and enhance the use of the PDBs
- Used in conjunction with Manhattan Distance (MD)

Reduction in size of search tree:
- MD = 346 * max(MD, FR)
- MD = 437 * max(MD, CO)
- MD = 1038 * max(MD, dovetail(FR, CO)) + tricks

Success Story #2

Rich Korf (1997)
- Rubik’s Cube ($10^{19}$ states).
- 3 hand-crafted patterns, all used together (max)
- Each PDB contains over 42 million entries
- took 1 hour to build all the PDBs

Results:
- First time random instances had been solved optimally
- Hardest (solution length 18) took 17 days
- Best known MD-like heuristic would have taken a century
Example: 8-puzzle

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

181,440 states

Domain = blank 1 2 3 4 5 6 7 8

“Patterns” created by domain abstraction

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

This abstraction produces 9 patterns

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

Pattern Space

Pattern Database

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Distance to goal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 1 2 2 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Distance to goal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 3 4</td>
</tr>
</tbody>
</table>
Calculating $h(s)$

Given a state in the original problem:

\[
\begin{array}{ccc}
8 & 1 & 4 \\
3 & 5 & \\
6 & 7 & 2 \\
\end{array}
\]

Compute the corresponding pattern:

\[
\begin{array}{ccc}
\text{Red} & \text{Red} & \\
\text{Red} & \text{Red} & \\
\text{Red} & \text{Red} & \\
\end{array}
\]

Look up the abstract distance-to-goal: 2

Domain Abstraction

\[
\begin{array}{ccc}
1 & 2 \\
3 & 4 & 5 \\
6 & 7 & 8 \\
\end{array}
\]

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

30,240 patterns

Fundamental Questions

**How to invent effective heuristics?**
Create a simplified version of your problem.
Use the exact distances in the simplified version as heuristic estimates in the original.

**How to use memory to speed up search?**
Precompute all distances-to-goal in the simplified version of the problem and store them in a lookup table (pattern database).

8-puzzle: $A^*$ vs. PDB size

[Graph showing the comparison between $A^*$ nodes expanded and pattern database size]
**Automatic Creation of Domain Abstractions**

- Easy to enumerate all possible domain abstractions

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

- They form a lattice, e.g.

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

  is “more abstract” than the domain abstraction above

**Efficiency**

Time for the preprocessing to create a PDB is usually negligible compared to the time to solve one problem-instance with no heuristic.

Memory is the limiting factor.

**Making the Best Use of Memory**

- 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
  - Jonathan’s new idea

**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
Max’ing Multiple Heuristics

• Given heuristics $h_1$ and $h_2$ define
  $h(s) = \max ( h_1(s), h_2(s) )$

• 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

\[ S = \Phi_1 \ldots \Phi_n \]

\[ h(s) \quad \ldots \quad h(s) \]

\( \max \)

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>61,466,541</td>
</tr>
</tbody>
</table>

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>b-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>165.5</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>

\( \text{RATIO} = \frac{\text{nodes generated using one PDB of size M}}{\text{nodes generated using n PDBs of size M/n}} \)

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 lookup**

**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**

A  
PDB₁(A) = 1  
PDB₂(A) = 7

B  
Because of consistency:  
PDB₁(B) ≤ 2  
PDB₂(B) ≥ 6  
⇒ No need to consult PDB₁

**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

**Runtime Distribution of Heuristic Values**

**Saving Space**

- If \( h_1 \) and \( h_2 \) are stored as pattern databases, \( \max(h_1(s), h_2(s)) \) requires twice as much space as just one of them.

- How can we get the benefits of \( \max \) without using any extra space?
  - “Dovetail” two PDBs
  - Use smaller PDBs to define max
**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

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>32</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>11</td>
<td>110</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>12</td>
<td>142</td>
<td>108</td>
<td>108</td>
</tr>
<tr>
<td>13</td>
<td>200</td>
<td>124</td>
<td>124</td>
</tr>
<tr>
<td>14</td>
<td>440</td>
<td>314</td>
<td>314</td>
</tr>
<tr>
<td>15</td>
<td>801</td>
<td>450</td>
<td>450</td>
</tr>
<tr>
<td>16</td>
<td>1,345</td>
<td>816</td>
<td>816</td>
</tr>
<tr>
<td>17</td>
<td>1,194</td>
<td>949</td>
<td>949</td>
</tr>
<tr>
<td>18</td>
<td>2,079</td>
<td>2,055</td>
<td>2,055</td>
</tr>
<tr>
<td>19</td>
<td>3,430</td>
<td>2,453</td>
<td>2,453</td>
</tr>
<tr>
<td>20</td>
<td>7,197</td>
<td>820</td>
<td>820</td>
</tr>
<tr>
<td>TOTAL</td>
<td>5,581</td>
<td>16,312</td>
<td>16,312</td>
</tr>
</tbody>
</table>

How to generalize Dovetailing to any abstractions of any space
Multiple Lookups in One Pattern Database

Example

<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>

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

distance

Standard PDB lookup

<table>
<thead>
<tr>
<th>abstract state</th>
<th>abstract goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Second 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>
**Relevance?**

Why is this lookup relevant to the original state?

**Two Key Properties**

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

**Experimental Results**

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

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

**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
  \[ h_1(s) + h_2(s) \geq \max(h_1(s), h_2(s)) \]

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.

\[
\begin{array}{c c c}
1 & & 3 \\
2 & 3 & 1 \\
\end{array}
\]

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

M.D. as Additive PDBs (1)

\[
\varphi_1(x) = \begin{cases} 
  x & \text{if } x = 1 \\
  \text{blank} & \text{otherwise}
\end{cases}
\]

\[
\begin{array}{c c c}
1 & & 1 \\
\varphi_1(\text{goal}) & \varphi_1(s) \\
\end{array}
\]

\[ \text{PDB}_1(\varphi_1(s)) = 2 \]

\[ \text{MD}(s) = \text{PDB}_1(\varphi_1(s)) + \text{PDB}_2(\varphi_2(s)) + \text{PDB}_3(\varphi_3(s)) \]

In General...

Partition the tiles in groups, \( G_1, G_2, \ldots, G_k \)

\[
\varphi_i(x) = \begin{cases} 
  x & \text{if } x \in G_i \\
  \text{blank} & \text{otherwise}
\end{cases}
\]
**Korf & Felner’s Method**

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

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

Moves of $\square$ cost zero.

**What’s the Difference?**

- Moves of $\square$ cost zero.
- The blank cannot reach the position without disturbing tile 1 or tile 2.

**Hierarchical Search**

**On-demand distance calculation**

- To build a PDB you must calculate all abstract distances-to-goal.
- Only a tiny fraction of them are needed to solve any individual problem.
- If you only intend to use the PDB to solve a few problems, calculate PDB entries only as you need them.
Calculate Distance by Searching at the Abstract Level

Replace this line:
\[ h(s) = \text{PDB}[\phi(s)] \]
by
\[ h(s) = \text{search}(\phi(s), \phi(\text{goal})) \]
(recursive) call to a search algorithm to compute abstract distance to goal for state s

Hierarchical Search

15-puzzle Results (1)

• Felner’s 7-7-1 additive PDB:
  – takes 80 minutes to build (4,800 secs)
  – Solves problems in 0.058 secs (on average)

• Felner’s 8-7 additive PDB
  – Takes 7 hours to build (25,200 secs)
  – Solves problems in 0.028 secs

15-puzzle Results (2)

Hierarchical IDA*, 1 Gigabyte limit
– Using the same abstraction for all problems, solving takes 242 secs (on average), or 207 secs if the cache is not cleared between problems
– Max’ing over Corner & Fringe abstractions, solving takes 150 secs (on average)
– Using a customized abstraction for each problem, solving takes 74 secs (on average)
Thesis topics abound!

General Dovetailing

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 0 0 0 0 0 0 0 0

- and to define a partial-order on them, e.g.
  
  Domain = blank 1 2 3 4 5 6 7 8
  Abstract = blank 0 0 0 0 0 0 0 0

  is “more abstract” than the domain abstraction above.

Lattice of domain abstractions
The “LCA” of 2 Abstractions

LCA = least-abstract common abstraction

General Dovetailing

- Given PDB₁ and PDB₂ defined by ϕ₁ and ϕ₂
- Find a common abstraction ψ of ϕ₁ and ϕ₂
- Because it is a common abstraction there exist ϕ₁ and ϕ₂ such that ϕ₁ ϕ₁ = ϕ₂ ϕ₂ = ψ
- For every pattern, p, defined by ψ, set SELECT[p] = ϕ₁ or ϕ₂
- Keep every entry (pₖ, h) from PDB for which SELECT[ψ(pₖ)] = i.
- Given state s, lookup SELECT[ψ(s)](s)