Pattern Databases

Robert Holte
Computing Science Dept.
University of Alberta

November 4, 2004

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

Joe Culberson & Jonathan Schaeffer (1994).

- 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

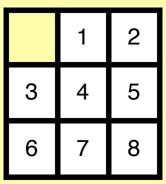
Rich Korf (1997)

- 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

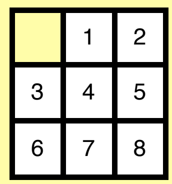
Example: 8-puzzle

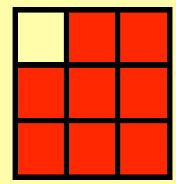


181,440 states

Domain = blank 1 2 3 4 5 6 7 8

"Patterns" created by domain abstraction



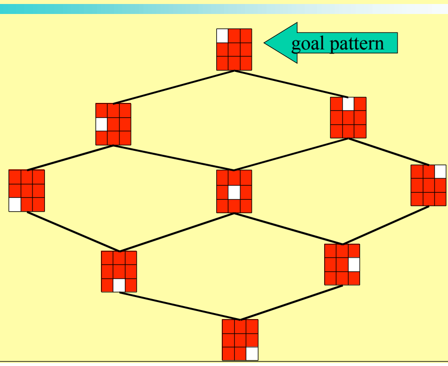


original state → corresponding pattern

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

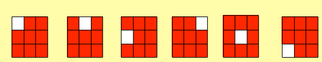
This abstraction produces 9 patterns

Pattern Space



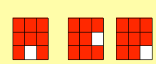
Pattern Database

Pattern



Distance to goal **0 1 1 2 2 2**

Pattern



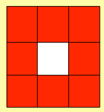
Distance to goal **3 3 4**

Calculating h(s)

Given a state in the original problem

8	1	4
3		5
6	7	2

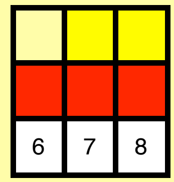
Compute the corresponding pattern



Look up the abstract distance-to-goal **2**

Domain Abstraction

	1	2
3	4	5
6	7	8



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

30,240 patterns

Fundamental Questions

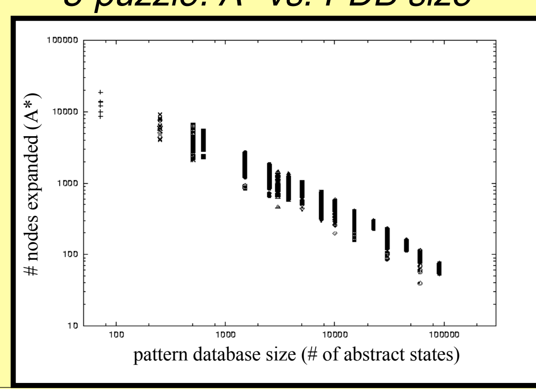
How to invent effective heuristics ?

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



Automatic Creation of Domain Abstractions

- Easy to enumerate all possible domain abstractions

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

- They form a lattice, e.g.

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

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

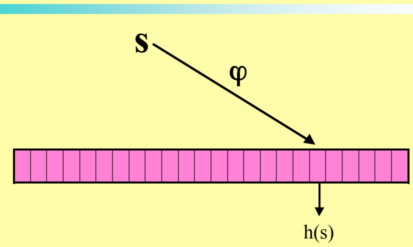
- Given heuristics h1 and h2 define
  h(s) = max (h1(s), h2(s))

- Preserves key properties:
  - lower bound
  - consistency

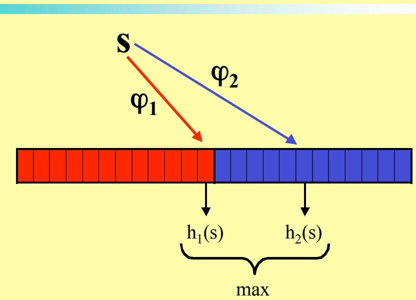
Question

- 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.

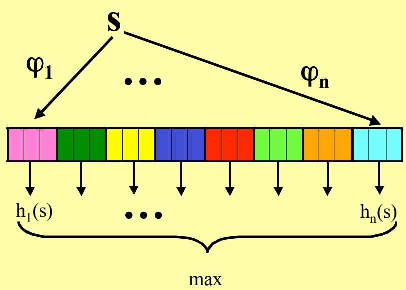
1 large pattern database



2 half-size pattern databases



Many small pattern databases



Rubik's Cube

PDB Size	n	Nodes Generated
13,305,600	8	2,654,689
17,740,800	6	**2,639,969**
26,611,200	4	3,096,919
53,222,400	2	5,329,829
106,444,800	1	**61,465,541**

Summary

State Space	Best n	Ratio
(3x3)-puzzle	**10**	**3.85**
9-pancake	**10**	**8.59**
(8,4)-Topspin (3 ops)	**9**	**3.76**
(8,4)-Topspin (8 ops)	**9**	**20.89**
(3x4)-puzzle	**21+**	**185.5**
Rubik's Cube	**6**	**23.28**
15-puzzle (additive)	**5**	**2.38**
24-puzzle (additive)	**8**	**1.6 to 25.1**

RATIO = #nodes generated using **one** PDB of size M / #nodes generated using **n** PDBs of size M/n

Rubik's Cube CPU Time

#PDBs	Nodes Ratio	Time Ratio
8	**23.15**	**12.09**
6	**23.28**	**14.31**
4	**19.85**	**13.43**
2	**11.53**	**9.87**
1	**1.00**	**1.00**

time/node is 1.67x higher using **six** PDBs

Techniques for Reducing the Overhead of Multiple PDB lookup

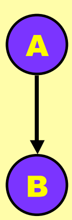
Early Stopping

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

Might result in extra IDA* iterations

$PDB_1(s) = 5$ ⇒ next bound is 8
$PDB_2(s) = 7$ ⇒ next bound is 10

Consistency-based Bounding



$PDB_1(A) = 1$
$PDB_2(A) = 7$

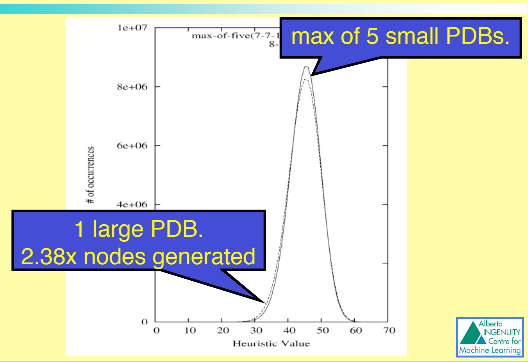
Because of consistency:
$PDB_1(B) \leq 2$
$PDB_2(B) \geq 6$
⇒ No need to consult PDB_1

Experimental Results

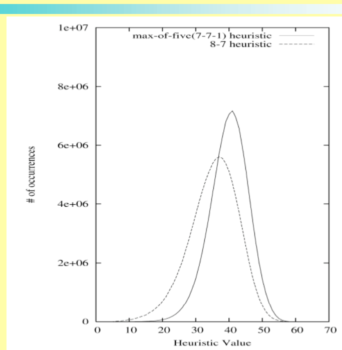
- 15-puzzle, five additive PDBs (7-7-1)
  - Naïve: 0.15 secs
  - Early Stopping: 0.10 secs
- Rubik's Cube, six non-additive PDBs
  - Naïve: 27.125 secs
  - Early Stopping: 8.955 secs
  - Early Stopping and Bounding: 8.836 secs

Why Does Max'ing Speed Up Search ?

Static Distribution of Heuristic Values



Runtime Distribution of Heuristic Values



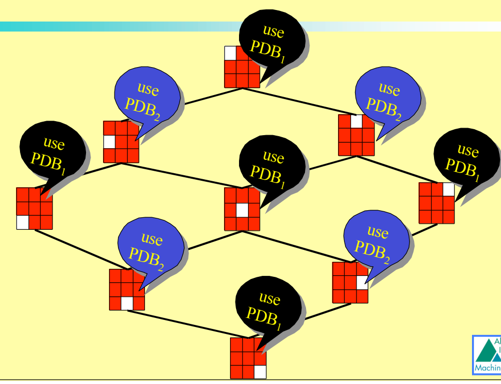
Saving Space

- If h1 and h2 are stored as pattern databases, max(h1(s),h2(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

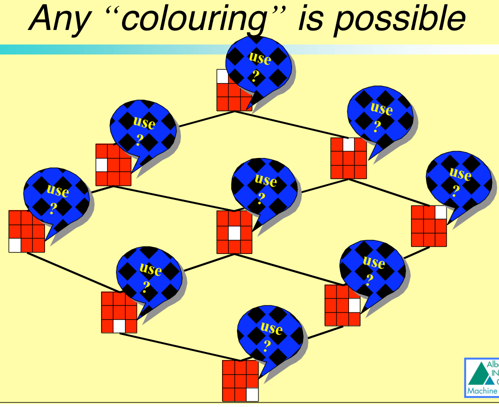
Dovetailing

- 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



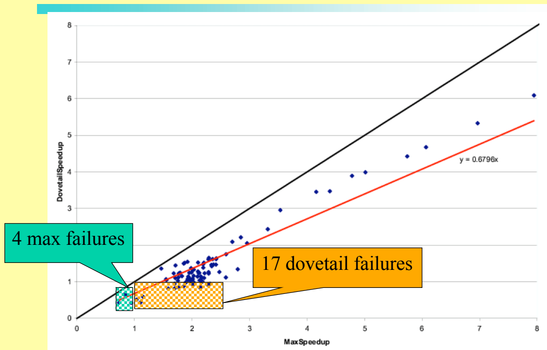
Any "colouring" is possible



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

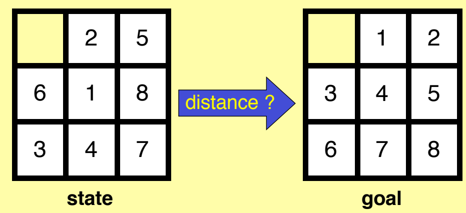
Depth Bound	h1	h2	max(h1,h2)
8	19	17	10
9		36	16
10	59	78	43
11		110	53
12	142	188	96
13		269	124
14	440	530	314
15		801	400
16	1,045	1,348	816
17		1,994	949
18	2,679	3,622	2,056
19		5,480	2,435
20	1,197	1,839	820
TOTAL	**5,581**	**16,312**	**8,132**

How to generalize Dovetailing to any abstractions of any space

?

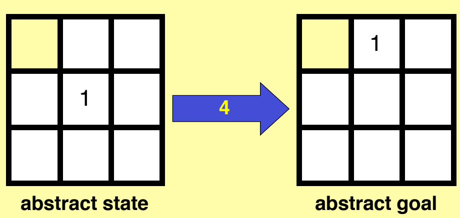
Multiple Lookups in One Pattern Database

Example

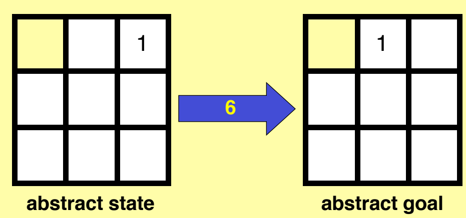


Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank 1 □ □ □ □ □ □ □

Standard PDB lookup

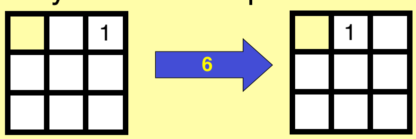


Second lookup, same PDB

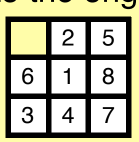


Relevance ?

Why is this lookup

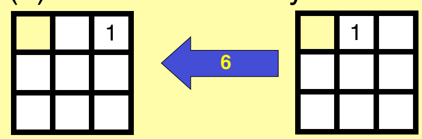


relevant to the original state ?

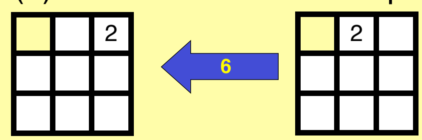


Two Key Properties

(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 <u>and still have an admissible heuristic.</u>

- 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.

goal

	1
2	3

state s

3	
1	2

$\boxed{MD(s) = 2 + 1 + 2 = 5}$

M.D. as Additive PDBs (1)

$$\varphi_1(\mathbf{x}) = \begin{cases} \mathbf{x} & \textbf{if } \mathbf{x = 1} \\ \textbf{blank} & \textbf{otherwise} \end{cases}$$

φ_1(goal)

	1

φ_1(s)

1	

$\boxed{PDB_1[\varphi_1(s)] = 2}$

$$MD(s) = PDB_1[\varphi_1(s)] + PDB_2[\varphi_2(s)] + PDB_3[\varphi_3(s)]$$

In General...

Partition the tiles in groups, G_1, G_2, … G_k

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

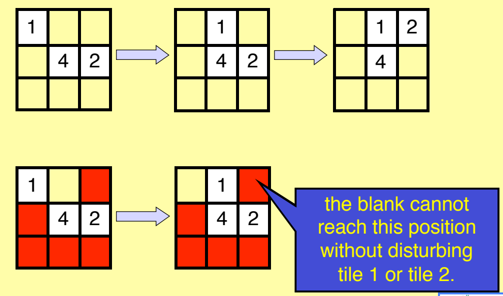
Korf & Felner's Method

Partition the tiles in groups, G_1, G_2, … G_k

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

Moves of ■ cost zero

What's the Difference ?



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.

Hierarchical Search

Calculate Distance by Searching at the Abstract Level

Replace this line:

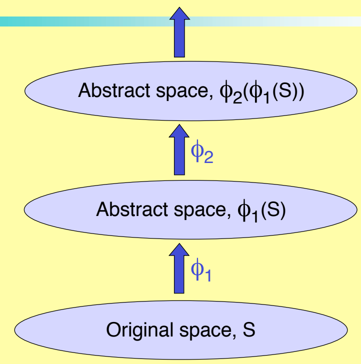
$h(s) = PDB[\ \phi(s)\ ]$

by

$h(s) = search(\phi(s), \phi(goal)\ )$

(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

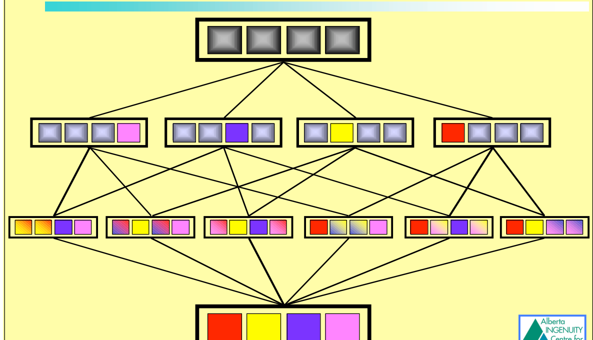
Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank

- and to define a partial-order on them, e.g.

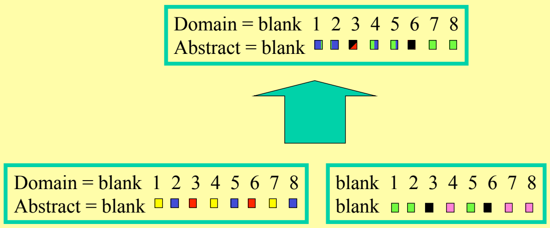
Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank

is "more abstract" than the domain abstraction above.

Lattice of domain abstractions



The "LCA" of 2 Abstractions



LCA = least-abstract common abstraction

General Dovetailing

- Given PDB_1 and PDB_2 defined by φ_1 and φ_2
- Find a common abstraction φ of φ_1 and φ_2
- Because it is a common abstraction there exist $\underline{\varphi}_1$ and $\underline{\varphi}_2$ such that $\underline{\varphi}_1\,\varphi_1 = \underline{\varphi}_2\,\varphi_2 = \varphi$
- For every pattern, p, defined by φ, set SELECT[p] = φ_1 or φ_2
- Keep every entry (p_k,h) from PDB_i for which SELECT[$\underline{\varphi}_i(p_k)$]=i.
- Given state s, lookup SELECT[φ(s)](s)