ECAI 2010
H. Coelho et al. (Eds.)
IOS Press, 2010
© 2010 The authors and IOS Press. All rights reserved.
doi:10.3233/978-1-60750-606-5-1059

Vectorial Pattern Databases

Carlos Linares López[1]

Abstract.
In this work, a new approach for creating Pattern Databases (PDBs) is suggested that induces non-consistent heuristic functions just by recognizing feasible (yet admissible) heuristic values. This approach serves to generalize even further the BPMX propagation rule, that will work now even in directed graphs. Experiments in different state spaces show a noticeable improvement over the Scalar Pattern Databases.

1 Introduction

Pattern Databases (PDBs) are simply hash tables which store, for every pattern (or arrangement of symbols in the *abstracted* state), the minimum number of moves required to place the symbols as they appear in the abstracted state space *for the very first time* in their goal location. So far, PDBs are admissible heuristic functions.

After setting up an abstraction for a given state space, the resulting PDB is usually a consistent heuristic function. However, it has been suggested that inconsistent heuristic functions can perform better in practice than consistent heuristic functions [4], because it is possible to propagate these inconsistencies through the search tree generated when solving a particular problem.

Moreover, it has been studied how to spot unfeasible heuristic values when using PDBs so that it is possible to increment their value while preserving the admissibility of the resulting heuristic function [3]. While the method devised applies in principle to disjoint PDBs, it has been also investigated how to extend the same idea to MAX PDBs [2], though this is still an open problem.

The paper is arranged as follows: first, Vectorial PDBs are introduced as a means for generating inconsistent heuristic functions by recognizing feasible values with MAX PDBs. Some experiments are reported immediately after.

2 Vectorial MAX Pattern Databases

Instead of storing just the minimum distance to the *first* occurrence of each pattern in a given abstract state space ψ_i, Vectorial PDBs store the distance to a successive number of ocurrences of the same pattern in an array, $\mathcal{H}_i$. The j-th component of this array, $\mathcal{H}_i[j]$, stores the distance to the j-th occurrence of the pattern referenced in each case. The number of occurrences of the same pattern is denoted as *pattern generation depth*, d. The procedure for computing these vectors is the same as the one used for computing the Scalar PDBs: a backwards breadth-first search suffices to compute a Vectorial PDB

The key observation is that abstractions are homomorphisms — i.e., all transitions in the original state space are preserved in the abstracted state space. Hence, a path from any node to the goal state, shall be mapped to a path in every abstraction ψ_i as well. Since the length of the same path in the abstract space shall be less or equal than its length in the original state space:

- If two (or more) Vectorial PDBs return vectors $\mathcal{H}_i$ such that their first component, $\mathcal{H}_i[0]$, is always the same, there is no reason to believe that the path traversed in every abstraction is not equal to the path to be traversed in the original state space. Thus, the resulting heuristic estimate shall be $\mathcal{H}_i[0]$.
- Otherwise, if two (or more) Vectorial PDBs return different values for the first component, it is clear that both PDBs have traversed different paths to *different nodes* with the same pattern. In the Scalar case, the best one can do is to pick up the maximum of all of them. However, in the Vectorial setting, if there are still more components in each vector to examine, one can scale up through each vector looking for an agreement between all of them, as in the previous case. If any is found, this should be the heuristic estimate to return; otherwise, the maximum of all components shall be returned.

Figure 1 shows different cases that can arise in the comparison of two vectors, $\mathcal{H}_1$ and $\mathcal{H}_2$, from two different abstractions explored with a pattern generation depth $d = 2$. The final heuristic value has been highlighted in bold face. The key observation is that a heuristic value is returned only if an agreement is found (or could be found as in figure 1(a) where 12 could be met by $\mathcal{H}_2$ as well) among all the Vectorial PDBs considered, since only in this case it can be concluded that the same path might have been found by all abstractions.

Furthermore, since every $\mathcal{H}_i$ provides a number of distances to the target pattern, it is possible to compute more than a single value. All these feasible values are stored in a vector denoted as H. When applicable this technique can lead to inconsistent heuristic values that result from picking up only feasible values.

To prove that Vectorial PDBs can result in inconsistent heuristics, an example is provided. Assume that the values returned by two Vectorial PDBs generated with a pattern generation depth equal to 2 for the same node n are $\mathcal{H}_1 = \{13, 14\}$ and $\mathcal{H}_2 = \{13, 15\}$. According to the procedure depicted above the first heuristic value shall be 13, since this is the first value agreed by both $\mathcal{H}_1$ and $\mathcal{H}_2$. Assume now that the vectors retrieved for a descendant n' are $\mathcal{H}'_1 = \{12, 15\}$ and $\mathcal{H}'_2 = \{14, 16\}$. Since there is no coincidence between both vectors the resulting value is 16, which is inconsistent (assuming an edge cost equal to 1) with the heuristic estimate of its ancestor, 13, but still admissible.

Since the heuristic function induced by Vectorial PDBs is inconsistent, after expanding node n and generating a child, n', the Pathmax propagation rules (and, if possible, the Bidirectional Pathmax propagation rule) can be applied, so that the heuristic value of the descendant is likely to be updated. Consider figure 1(d). In this case, $H = \{10, 13\}$. The first value, 10, is included in H because it ap-

[1] Planning and Learning Group, Universidad Carlos III de Madrid. Avda. de la Universidad, 30 - 28911 Leganés, Madrid (Spain) email: carlos.linares@uc3m.es

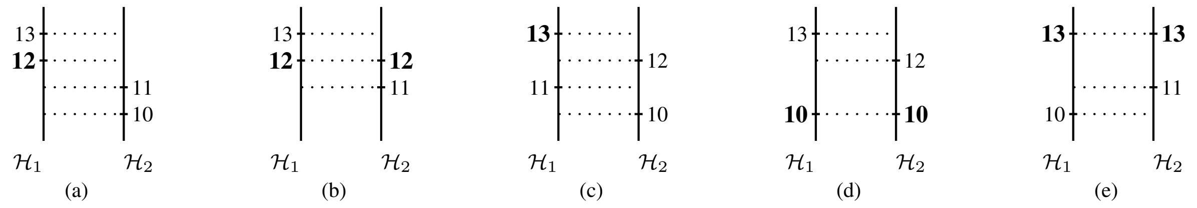

Figure 1. Two Vectorial Pattern Databases generated with $d = 2$

pears both in $\mathcal{H}_1$ and $\mathcal{H}_2$. Next, 12 is skipped since there is clearly no path of length 12 in the first abstraction. Finally, 13 can be safely included in H even if it did not appear in $\mathcal{H}_2$ since it is the next plausible length. After generating a descendant n', assume its vector is $H' = \{10, 14\}$ but before returning to its parent, the (Bidirectional) Pathmax propagation rules have updated its value to 14. Under these circumstances, node n can update its heuristic value to 13, since that is the largest value in its vector H which is less or equal than 14. Very importantly, this update rule can be applied both if the state space is directed or not.

3 Results

Unfortunately, it can be proven that this technique does not improve over Scalar PDBs in those domains where there are operators which do not affect the constants mapped by a particular PDB as in the Pancake or the Rubik's Cube. Thus, the puzzles M12 and M24 have been selected since they have operators that affect all locations at the same time [1]. Besides, they do add some interesting properties such as the fact that the underlying state space is directed.

Since the Scalar PDBs are consistent for the M12, the IDA* has been used for a pattern generation depth, $d = 1$. However, since Vectorial PDBs result in a inconsistent heuristic, the Pathmax propagation rules have been used along with the feasibility analysis suggested in the previous section with two different pattern generation depths, $d = 2, 3$.

d	4-4	4-4-4
1	96829	45315
2	71767 (74.1%)	39932 (88.1%)
3	64875 (66.9%)	36804 (81.2%)

Table 1. Results in the M12 Puzzle (number of generated nodes)

Table 1 shows the number of nodes generated when solving 800 instances in the M12 with either Scalar PDBs ($d = 1$) or Vectorial PDBs —$d = 2, 3$. Two different arrangements of patterns have been tried: 4-4 and 4-4-4. An arrangement denoted as $n - n$ (or $n - n - n$) takes two (or three) successive groups of length n each and maps them into distinct abstractions. It is clear that using Vectorial PDBs leads to a reduction of 25.9% in the 4-4 setting, from 96,829 to 71,767. In the case of the 4-4-4 arrangement, the reduction is smaller and only a little bit larger than 10%.

Regarding the M24, the first set of experiments involved only 12 constants —to be denoted as 12-M24. Table 2 shows the number of generated nodes when solving 1000 instances randomly generated for different arrangements of PDBs and pattern generation depths, $d = 1, 2, 3$. As it can be seen, when going from Scalar PDBs ($d = 1$) to Vectorial PDBs ($d = 2$) it is possible to save about 35% the number of nodes generated with $d = 1$. Successive pattern generation depths only improve marginally and asymptotically to roughly half the number of nodes.

d	5-5	6-6	4-4-4
1	33112637	4842120	166040667
2	21142126 (63.8%)	3024885 (62.4%)	105878854 (63.7%)
3	18790477 (56.7%)	2682071 (55.3%)	90663551 (54.6%)
4	18106592 (54.6%)	2588118 (53.4%)	85226273 (51.3%)

Table 2. Results in the 12-M24 Puzzle (number of generated nodes)

d	6-6-6
1	87856
2	81086 (92.2%)

Table 3. Results in the 24-M24 Puzzle (number of generated nodes)

Table 3 shows the number of nodes generated when solving 1000 random instances with 24 constants —denoted now as 24-M24. The savings in this case are very moderate mainly because the Scalar PDB implementes a very informed heuristic function so there is not much room for improvement. Nevertheless, it can be proven that the search tree developed both in M12 and M24 creates a number of terminal nodes at each level which is bounded by the Fibonacci sequence so that these small savings translate into large subtrees being pruned.

ACKNOWLEDGEMENTS

This paper significantly benefited from discussions with Gabriele Röger, Malte Helmert, Álvaro Torralba, Vidal Alcázar and Daniel Borrajo. This work has been partially supported by the Spanish MICINN under project TIN2008-06701-C03-03

REFERENCES

[1] Igor Kriz and Paul Siegel, 'Rubik's cube inspired puzzles demonstrate math's "simple groups"', *Scientific American*, (July 2008).

[2] Fan Yang, 'Exploring infeasibility for abstraction–based heuristics', in *Search in Artificial Intelligence and Robotics: 2008 AAAI Workshop*, pp. 134–139, Chicago, USA, (July 2008).

[3] Fan Yang, Joseph C. Culberson, Robert Holte, Uzi Zahavi, and Ariel Felner, 'A general theory of additive state space abstractions', *Journal of Artificial Intelligence Research*, **32**, 631–662, (June 2008).

[4] Uzi Zahavi, Ariel Felner, Jonathan Schaeffer, and Nathan Stutervant, 'Inconsistent heuristics', in *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, pp. 1211–1216, Vancouver, British Columbia, Canada, (July 2007).