Additive Pattern Database Heuristics

Ariel Felner
Ben-Gurion University of the Negev, Beer-Sheva, Israel

Richard E.Korf,
University of California,Los Angeles

Sarit Hanan
Bar-Ilan University , Ramat-Gan, Israel

2004

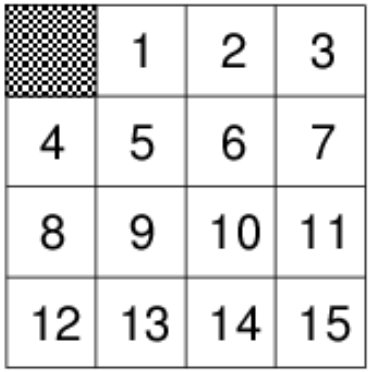
presented by Juan Cheng

1. Introduction and Overview

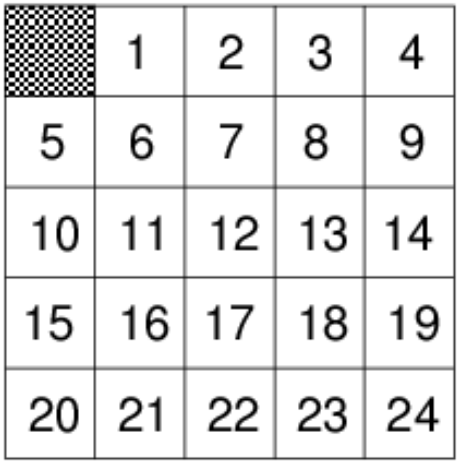
- Heuristic search: A*, IDA*...
- A heuristic evaluation function h(n)=an estimate of the cost of an optimal solution from node n to a goal state
- If h(n) is admissible, meaning that it never overestimates the cost of reaching a goal, then all the above algorithms are guaranteed to return an optimal solution, if one exists.

- The most effective way to improve the performance of a heuristic search algorithm is to improve the accuracy of the heuristic evaluation function.
- Developing more accurate heuristic functions is the goal of this paper.

1.1 Sliding-Tile Puzzles



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15



	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

re 1: The Fifteen and Twenty-Four Puzzles in their Goal States

- An optimal solution to the problem uses the fewest number of moves possible
- Solving random instances of the Twenty-Four puzzle with Manhattan distance is not practical on current machines
- A more accurate heuristic function is needed

1.2 Subgoal Interactions

- Inaccuracy of the *Manhattan distance*: it assumes that each tile can be moved along a shortest path to its goal location without interference from any other tiles
- The tiles do interfere with each other
- The key to developing a more accurate heuristic function is to account for some of those interactions in the heuristic

1.3 two main contributions of this paper

- Show that additive pattern databases which were first developed and implemented for the sliding tile puzzles can be generalized and applied to other domains.
- Divide additive pattern databases into two methods, statically- and dynamically-partitioned databases, and compare the two methods

2. Previous Work on Admissible Heuristic Functions

- *Linear -conflict heuristic* (Hansson, mayer, & Yung, 1992) ： the interaction between these two tiles allows us to add two moves to the sum of their Manhattan distance

2. Previous Work on Admissible Heuristic Functions

- *Pattern database*: precomputed tables of the exact cost of solving various subproblems of an exist problem, with a single breadth-first search backward from the goal state
- They chose as a subset of a 15-Puzzle: those in the bottom row and those in the rightmost column
- Once this table is computed, use IDA* to search for an optimal solution to a particular problem instance

2.4 Statically-Partitioned Additive database Heuristics

- *Disjoint pattern databases:* partition the tiles into disjoint groups, such that every tile is included in a group, and no tile belongs to more than one groups
- Precompute pattern databases of the minimum number of moves of the the tiles in each group
- Retrieve the number of moves required to solve the tiles, *Add* together the values for each group

Statically-Partitioned Additive database Heuristics

- this value>=Manhattan distance, because it accounts for *interactions* between tiles in the same group
- The key difference between additive databases and the non-additive databases is: *non-additive* databases include all moves required to solve the pattern tiles, including moves of tiles not in the pattern group. In an *additive* pattern database, we only count moves of the tiles in the group.

2.4.1 *Limitations* of statically-partitioned database heuristics

- They fail to capture interactions among tiles in *different groups* of the partition.
- A different approach to additive pattern databases: dynamically-partitioned database heuristics

3. Dynamically-Partitioned Database Heuristics

3.1 Computing the Pairwise Distances, or Gasser refers it as *2-tile pattern database* :

- a table which contains for each pair of tiles, and each possible pair of positions they could occupy, the number of moves required of those two tiles to move them to their goal positions
- How to compute such tables: for each pair of tiles, we perform a single breadth-first search, starting from the goal state.

3.2 Computing the Heuristic Value of a State

- partition the n tiles into n/2 non-overlapping pairs, then sum the pairwise distances for each of the chosen pairs
- To get the most accurate admissible heuristic, we want a partition that maximizes the sum of the pairwaise distances

- For each state of the search , this maximizing partition may be different, requiring the partitioning to be performed for each heuristic evaluation. Thus, the authors use the term *dynamically-partitioned addititive pattern database heuristics*
- To compute the heuristic for a given state, use *mutual-cost graph*

3.3 Triple and Higher-Order Distances

- Groups of various sizes, not necessarily pairs
- 3-tile databases

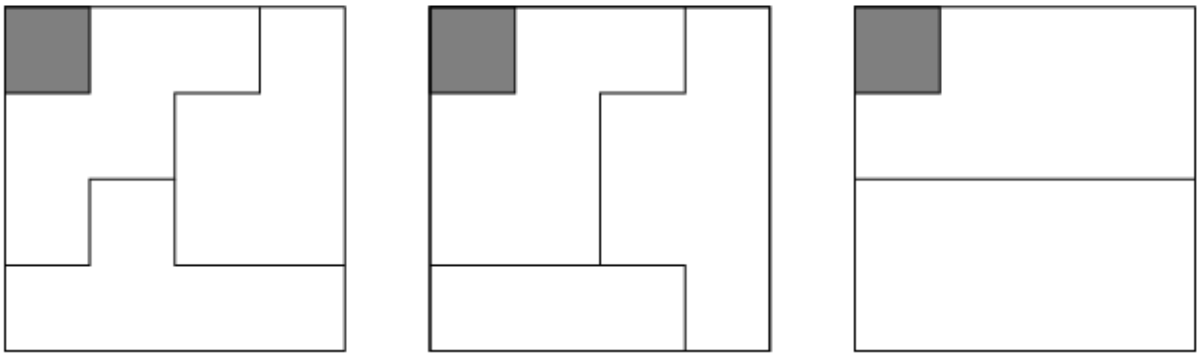
Dynamically-Partitioned database Heuristics

- The main advantage: it can potentially capture more tile interactions, statically-partitioned pattern database only captures interactions between tiles in the same group
- Disadvantage: expensive

4. Sliding-Tile Puzzles

- Experiments solving the 15 and 24 puzzles using statically and dynamically partitioned additive pattern database heuristics

4.2.2 fifteen puzzle partition



A 5-5-5 partitioning A 6-6-3 partitioning A 7-8 partitioning

Figure 3: Different Statically-partitioned Databases for Fifteen Puzzle

Experimental results on the Fifteen Puzzle

Heuristic	Value	Nodes	Sec.	Nodes/sec	Memory
Manhattan	36.940	401,189,630	53	7,509,527	0
Linear conflict	38.788	40,224,625	10	3,891,701	0
Dynamic, MM: pairs	39.411	21,211,091	13	1,581,848	1,000
Dynamic, MM: pairs+triples	41.801	2,877,328	8	351,173	2,300
Dynamic, WVC: pairs	40.432	9,983,886	10	959,896	1,000
Dynamic, WVC: pairs+triples	42.792	707,476	5	139,376	2,300
DWVC: pairs+triples+quadruples	43.990	110,394	9	11,901	78,800
Static: 5-5-5	41.560	3,090,405	.540	5,722,922	3,145
Static: 6-6-3	42.924	617,555	.163	3,788,680	33,554
Static: 7-8	45.630	36,710	.028	1,377,630	576,575

Table 1: Experimental results on the Fifteen Puzzle.

4.2.3 Twenty-four puzzle

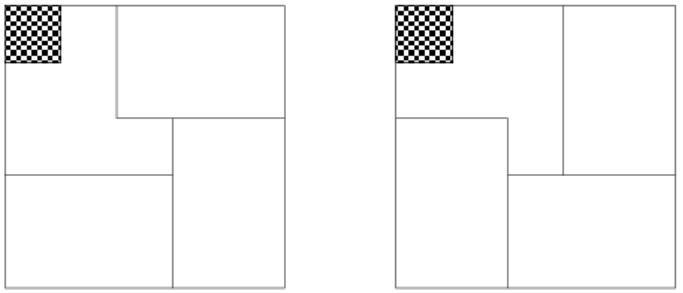


Figure 4: Static 6-6-6-6 database for Twenty-Four Puzzle and its reflection about the main diagonal

Twenty-Four Puzzle results for static vs. dynamic databases

Problem		Dynamic Partitioning		Static Partitioning	
No	Path	Nodes	Seconds	Nodes	Seconds
1	95	306,958,148	1,757	2,031,102,635	1,446
2	96	65,125,210,009	692,829	211,884,984,525	147,493
3	97	52,906,797,645	524,603	21,148,144,928	14,972
4	98	8,465,759,895	72,911	10,991,471,966	7,809
5	100	715,535,336	3,922	2,899,007,625	2,024
6	101	10,415,838,041	151,083	103,460,814,368	74,100
7	104	46,196,984,340	717,454	106,321,592,792	76,522
8	108	15,377,764,962	82,180	116,202,273,788	81,643
9	113	135,129,533,132	747,443	1,818,055,616,606	3,831,042
10	114	726,455,970,727	4,214,591	1,519,052,821,943	3,320,098

Table 2: Twenty-Four Puzzle results for static vs. dynamic databases

4.2.4 Thirty-five puzzle

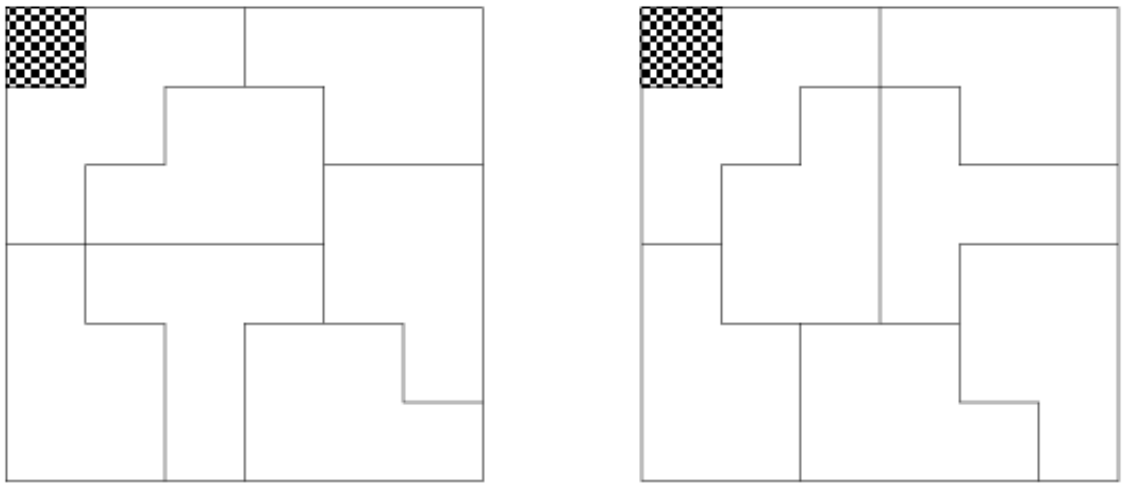


Figure 5: Disjoint Databases for Thirty-Five Puzzle

Thirty-Five Puzzle

Heuristic	Value	Rel. Nodes	Nodes/sec	Rel. Time	Memory
Manhattan	135.02	31,000.0	20,500,000	987.5	0
Static	139.82	11.5	4,138,000	1.8	404,000
Dynamic	142.66	1.0	653,000	1.0	5,000

Table 3: Predicted Performance results on the Thirty-Five Puzzle

4.2.5 Discussion of the tile puzzle results

The relative advantage of the statically-partitioned database heuristics over the dynamically-partitioned heuristics appears to decrease as the problem size increases.

5. 4-Peg Towers of Hanoi Problem

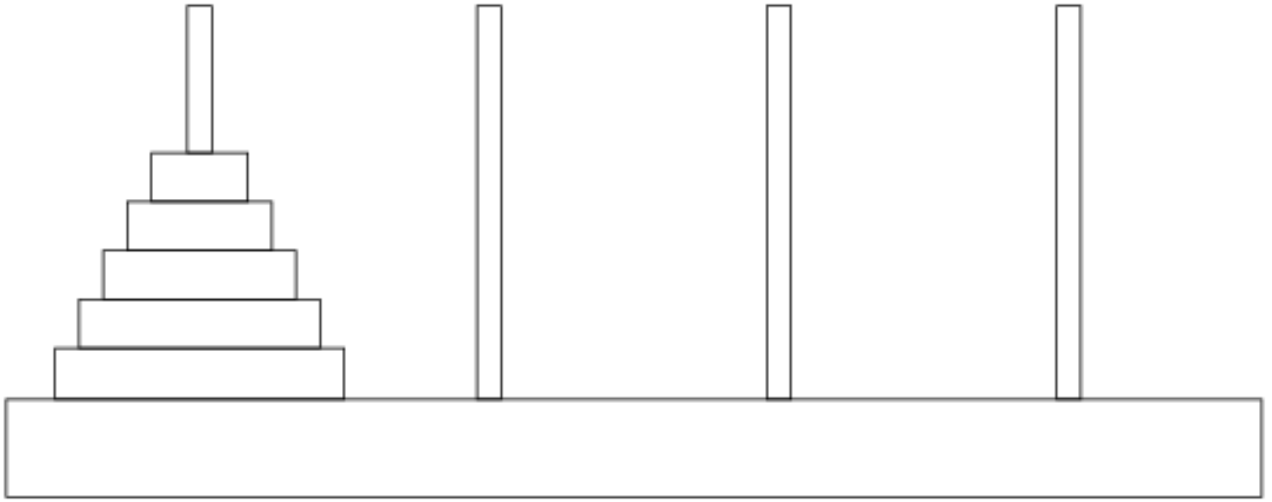


Figure 6: Five-disk four-peg Towers of Hanoi problem

Experimental results for the 15-disk Hanoi problem

Heuristic	h(s)	Avg h	Nodes Expanded	Seconds	Nodes Stored
Infinite Peg	29	26.37	371,556,049	1198	27,590,243
Static 12-3	86	62.63	9,212,163	32	1,778,813
Static 13-2	100	74.23	2,205,206	7	525,684
Static 14-1	114	87.79	158,639	2	82,550
Dynamic 14-1	114	95.52	122,128	2	25,842

Table 4: Results for the 15-disk problem

Results for the 16-disk problem

Heuristic	h(s)	Avg h	Nodes Expanded	Seconds	Nodes Stored
Static 13-3	102	75.78	65,472,582	221	10,230,261
Static 14-2	114	89.10	17,737,145	65	2,842,572
Dynamic 14-2	114	95.52	6,242,949	96	1,018,132

Table 5: Results for the 16-disk problem

6. Vertex Cover

Database	Cliques	Nodes	seconds	nodes/sec
No database	–	720,582,454	4,134	174,306
Static	2	187,467,358	1,463	128,139
Static	3	105,669,961	849	124,464
Static	4	103,655,233	844	123,252
Dynamic	2	19,159,780	231	82,982
Dynamic	3	4,261,831	70	60,451
Dynamic	4	4,170,981	73	57,136

Table 6: Performance of the different algorithms on graphs with 150 vertices and a density of 16.

Experimental results

size	Random graph, density 8			Random graph, density 16			Delaunay graphs		
Size	Vc	Nodes	Sec	Vc	Nodes	Sec	Vc	Nodes	Sec
150	92	2400	1	113	316,746	26	103	27,944	1
200	130	296,097	33	153	21,897,066	2,045	137	224,349	3
250	164	9,703,639	1,116	187	544,888,130	58,776	171	4,035,989	67
300	181	56,854,403	7,815				204	32,757,219	493
350	219	137,492,886	19,555				241	1,146,402,687	20,285
400							269	27,443,208,087	485,581

Table 7: Experimental results of our best combination on different graphs.

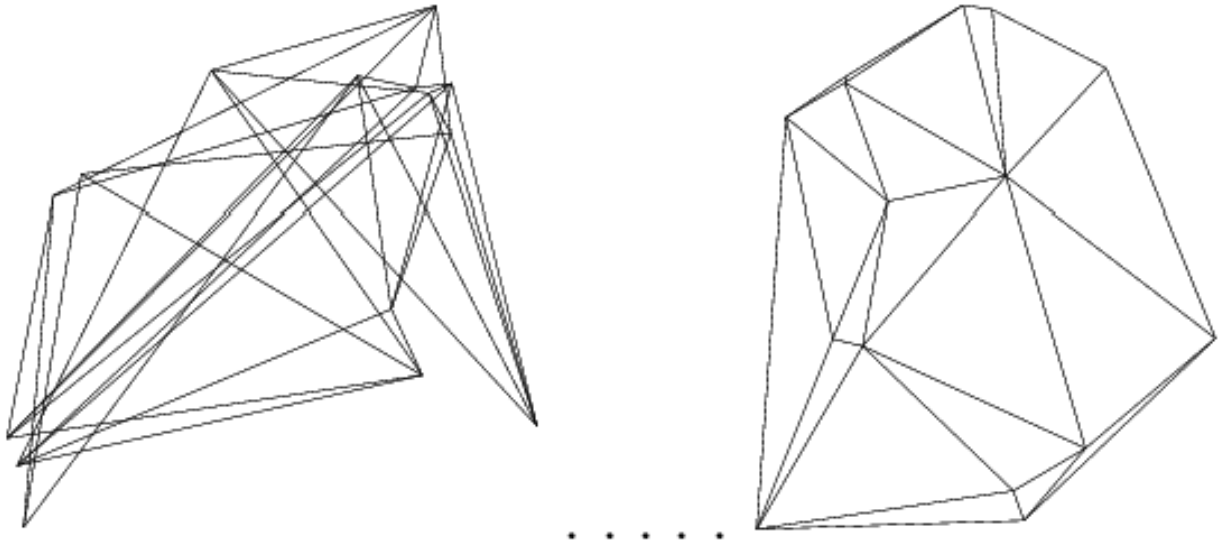


Figure 8: Random and Delaunay graphs of size 15.

7. General Characterization of the Method

- Define a *pattern* as an assignment of values to a subset of the state variables in a problem
- A pattern database is built by running a breadth-first search backward from the complete goal state, until at least one state representing each pattern is generated

7.2.1 Complete pattern databases

- A complete pattern database stores the cost of solving every pattern, defined by the set of all possible combinations of values of the variables in the subset
- Advantage: every value is stored
- Disadvantage: memory requirements

7.2.2 Partial pattern databases

- Only stores the cost of solving some of the patterns
- Save memory by storing fewer values than a complete database
- Efficient for dynamically-partitioned database

7.3 Algorithm Schema

- The steps for the statically-partitioned database heuristics

- In the pre-computation phase do the following:
  - Partition the variables of the problem into disjoint subsets.
  - For each subset of variables, solve all the patterns of these variables and store the costs in a database.

The steps for the statically-partitioned database heuristics

- In the search phase do:
  - for each state of the search space, retrieve the values of solving the relevant patterns for each subset of the variables from the relevant databases, and add them up for an admissible heuristic.

7.3.2 The steps for the dynamically-partitioned database heuristics

- In the pre-computation phase do the following:
  - For each subset of variables to be used (i.e., pair of variables or triple of variables etc.)
  - Solve all the patterns of these subsets and store the costs in a database.
- In the search phase do:
  - For each state in the search, retrieve the costs of the current patterns for each subset of variables.
  - Find a set of disjoint variables from the database such that the sum of the corresponding costs is the largest admissible heuristic. In general, the corresponding hypergraph matching problem is NP-complete, and we may have to settle for an approximation to the largest admissible heuristic value.

7.4 Differences between the applications

- Trying the general schema on new problems may involve significant creativity in identifying the subproblems, determining the definition of disjoint subproblems, computing the pattern databases, doing the partitioning,etc.

8.1 when to use each of the methods

- Domain dependent
- Memory requirements

8.2 Conclusions and Further Work

- these techniques can be effectively extended to new problem domains
- Further work: Find more efficient algorithms for finding the best dynamic partitionings
- To automatically find the optimal size of dynamically-partitioned databases

Questions