Additive Pattern Database Heuristics

Ariel Felner
Richard E. Korf
Sarit Hanan

Pier Paolo Bortoluzzi

November 15, 2012

Additive Pattern Database Heuristics

Why?

Additive Pattern Database Heuristics

Why?

Non-additive pattern databases don't scale well

Additive Pattern Database Heuristics

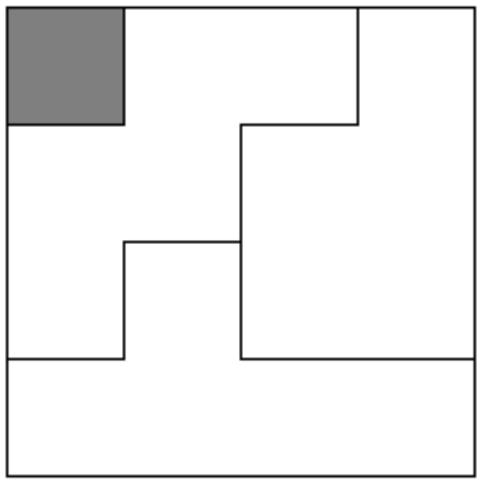
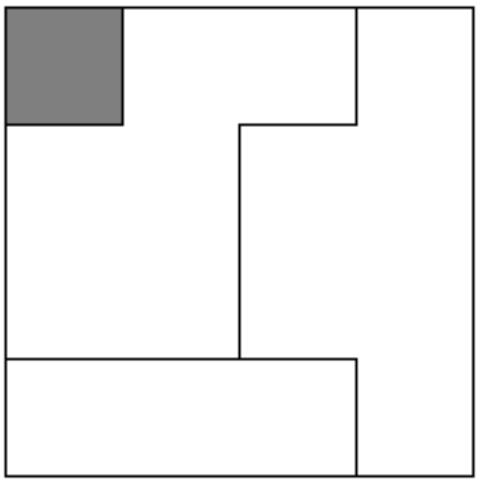
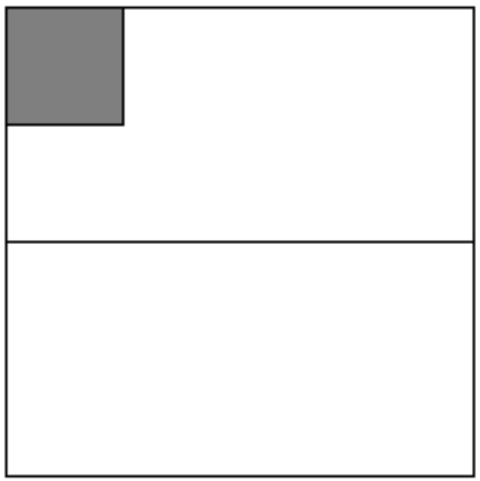
Why?

Non-additive pattern databases don't scale well

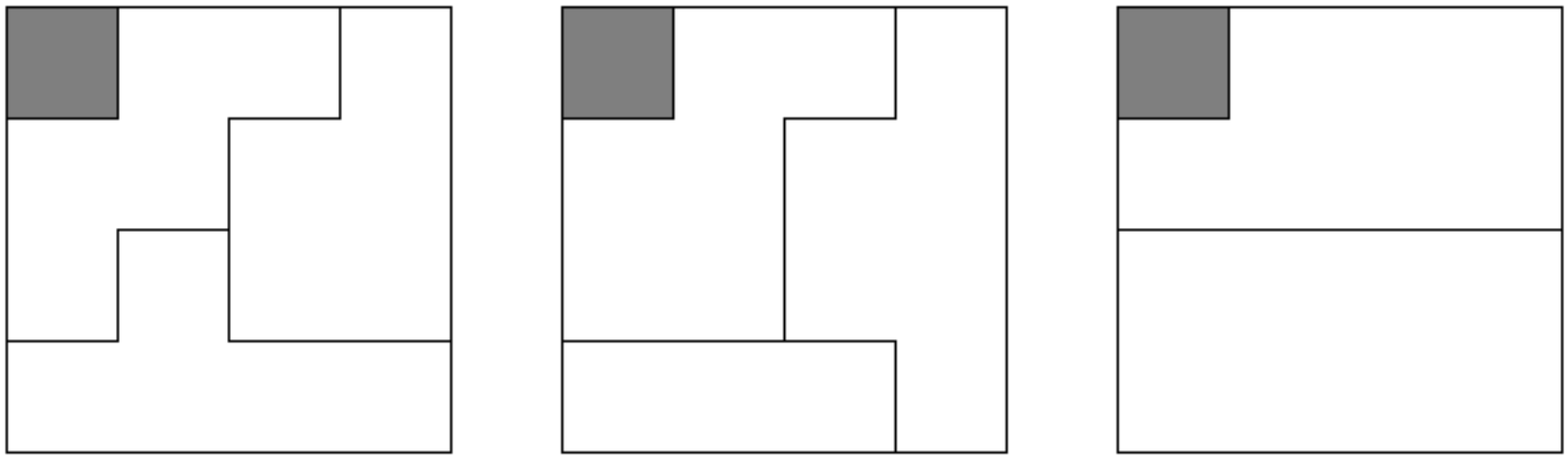
Two ways to create additive pattern databases:

- Statically-partitioned (old)
- Dynamically-partitioned (new)

Disjoint or Statically-Partitioned Additive Database Heuristic

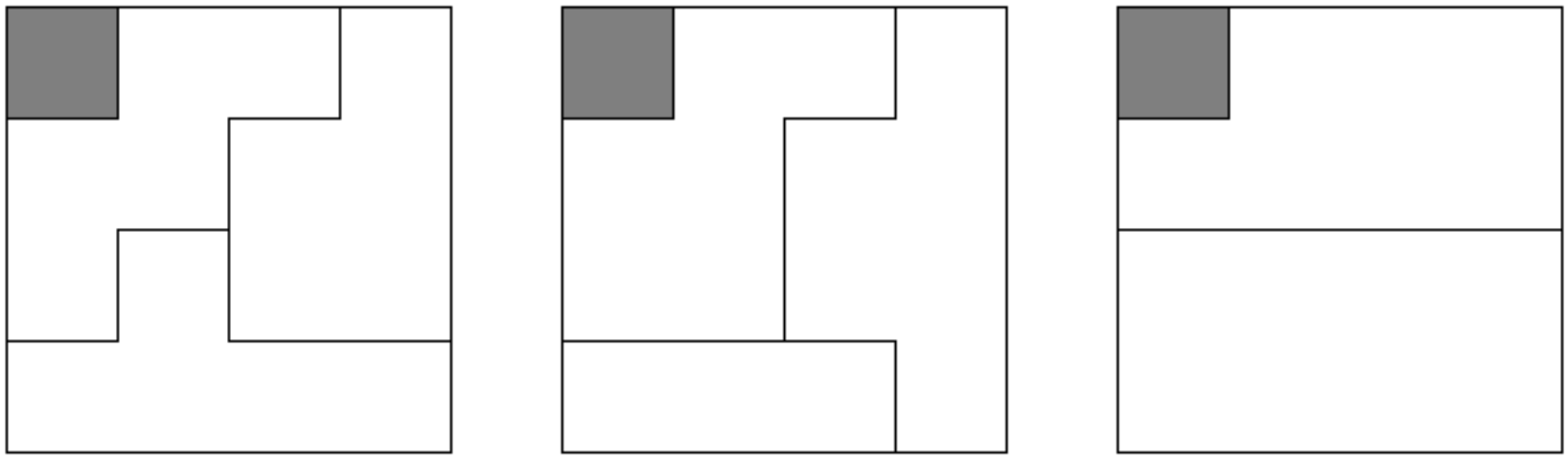




Disjoint or Statically-Partitioned Additive Database Heuristic



- Only count moves of the tiles within the group to avoid overestimation

Disjoint or Statically-Partitioned Additive Database Heuristic



- Only count moves of the tiles within the group to avoid overestimation
- Fails to capture interactions between tiles in different groups

Dynamically-Partitioned Database Heuristics

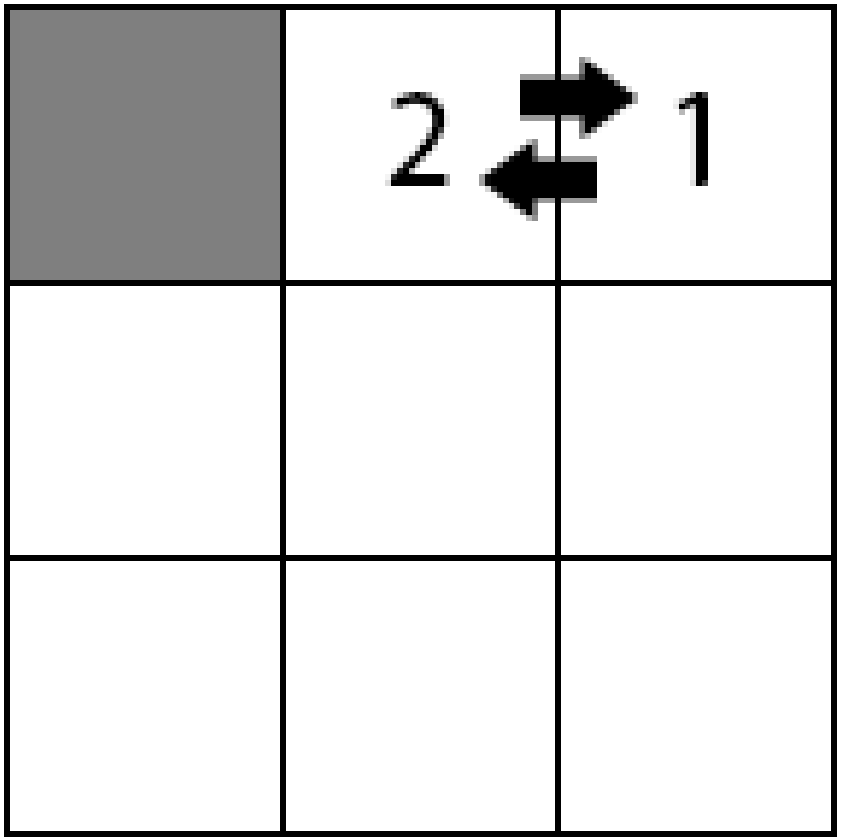
"*Consider a table which contains for each pair of tiles, and each possible pair of positions they could occupy, the number of moves required of those two tiles to move to their goal positions.*"

Dynamically-Partitioned Database Heuristics

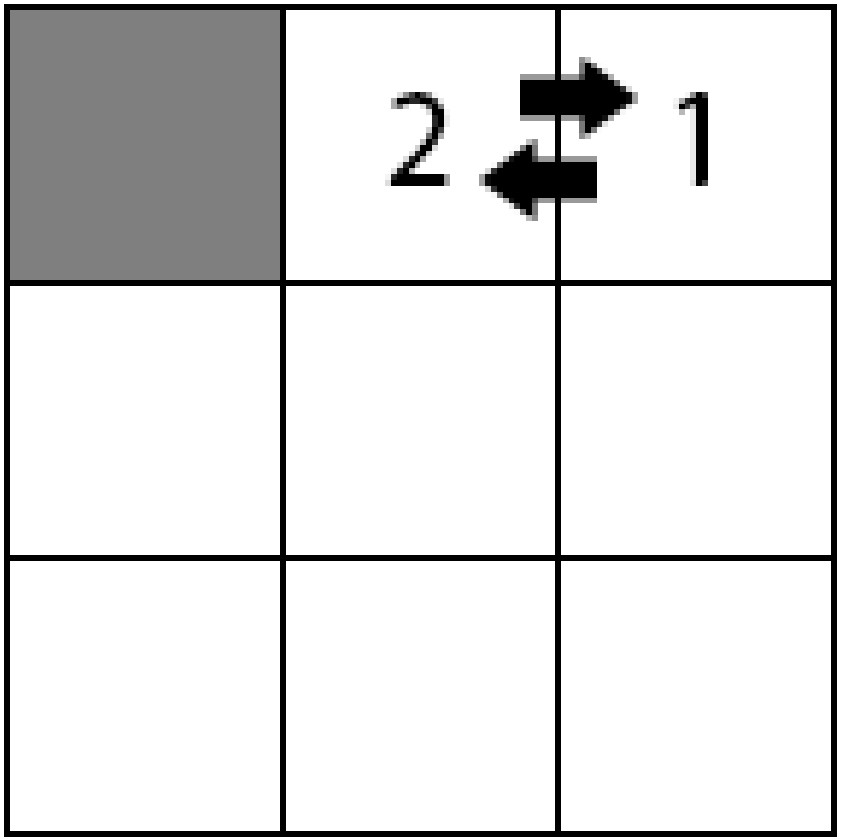
"*Consider a table which contains for each pair of tiles, and each possible pair of positions they could occupy, the number of moves required of those two tiles to move to their goal positions.*"

Given n tiles, there are $O(n^4)$ entries.

Linear Conflict

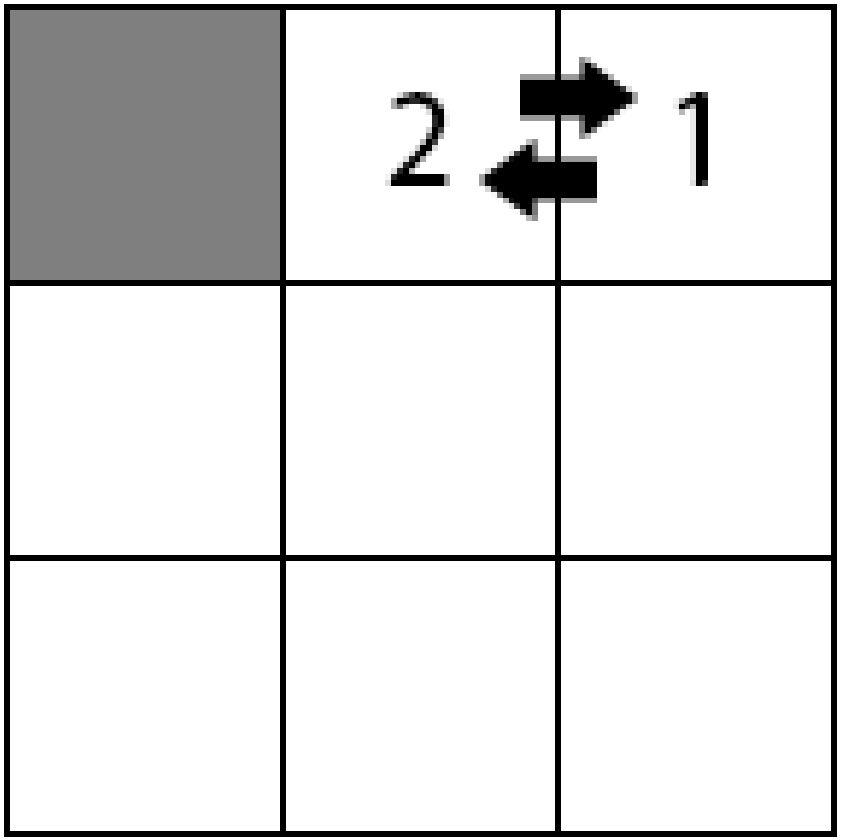


Linear Conflict



Manhattan-Distance: 2

Linear Conflict



Manhattan-Distance: 2
Considering Linear conflict: +2

Pairwise Distance

How do we compute such tables?

Pairwise Distance

How do we compute such tables?

1. Start with the goal state of a pair.

Pairwise Distance

How do we compute such tables?

1. Start with the goal state of a pair.
2. Perform breadth-first search.

Pairwise Distance

How do we compute such tables?

1. Start with the goal state of a pair.
2. Perform breadth-first search.
3. Store each unique state until all have been found.

Pairwise Distance

How do we compute such tables?

1. Start with the goal state of a pair.
2. Perform breadth-first search.
3. Store each unique state until all have been found.
4. Rinse and repeat for all $n(n-1)/2$ different pairs.

Value of a State using Pairwise Distance

We can not simply sum the DB values for each pair of tiles, we only need n/2 non-overlapping pairs (and a single manhattan distance).

Value of a State using Pairwise Distance

We can not simply sum the DB values for each pair of tiles, we only need n/2 non-overlapping pairs (and a single manhattan distance).

Goal: Maximize this sum over all non-overlapping pairs, but how?

Value of a State using Pairwise Distance

We can not simply sum the DB values for each pair of tiles, we only need n/2 non-overlapping pairs (and a single manhattan distance).

Goal: Maximize this sum over all non-overlapping pairs, but how?
Mutual Cost Graphs!

Higher-Order Distances

Can be accomplished by using a step-ladder approach:

Higher-Order Distances

Can be accomplished by using a step-ladder approach:

1. Create a 1-tile pattern database (Manhattan distances of all tiles at each position)

Higher-Order Distances

Can be accomplished by using a step-ladder approach:

1. Create a 1-tile pattern database (Manhattan distances of all tiles at each position)
2. 2-tile pdb has to store only the cases where it exceeds the sum of 1-tile pdb

Higher-Order Distances

Can be accomplished by using a step-ladder approach:

1. Create a 1-tile pattern database (Manhattan distances of all tiles at each position)
2. 2-tile pdb has to store only the cases where it exceeds the sum of 1-tile pdb
3. 3-tile pdb stores cases where they exceed either the sum of their Manhattan distances or the pairwise heuristic of the three tiles.

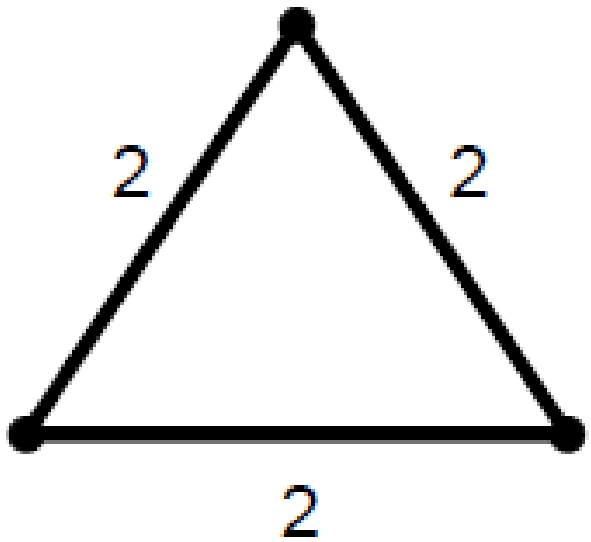
Higher-Order Distances

Can be accomplished by using a step-ladder approach:

1. Create a 1-tile pattern database (Manhattan distances of all tiles at each position)
2. 2-tile pdb has to store only the cases where it exceeds the sum of 1-tile pdb
3. 3-tile pdb stores cases where they exceed either the sum of their Manhattan distances or the pairwise heuristic of the three tiles.
4. k-tile . . .

Weighted Vertex Cover Distance

Domain-specific enhancement:



Experimental Results

Heuristic Function	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

Thank you for listening

Questions?