Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

Zero-Aware Pattern Databases with 1-Bit Compression for Sliding Tile Puzzles

Robert Clausecker, Alexander Reinefeld
Zuse Institute Berlin
Takustraße 7, 14195 Berlin, Germany

Abstract

A pattern database (PDB) is a pre-computed lookup table storing shortest distances from abstract states to abstract goal states. PDBs are key components in heuristic search as their entries are used to prune paths that cannot lead to an optimal solution. With the sliding-tile puzzle as an exemplary application domain, we present methods to improve the precision and size of PDBs by

- improving additive pattern databases to zero-aware additive pattern databases (ZPDBs),
- reducing the compression rate from 1.6 to 1 bit per entry,
- generating optimal additive pattern partitionings, and
- building effective collections of pattern databases.

With these enhancements, we achieve an overall 8.59-fold performance gain on the 24-puzzle compared to the previously best set of 6-tile PDBs.

1 Background

Heuristic search can be improved by either devising faster search algorithms or by building better heuristics. While search algorithms like A*, IDA* (Korf 1985) and its many derivatives seem to be reasonably well understood by now, developing powerful heuristics that prune most effectively remains a challenging research topic.

Pattern databases (PDBs) are heuristics in the form of memory tables (Culberson and Schaeffer 1996; Korf and Felner 2002). A PDB maps the state of a planning task onto a much smaller, abstract state of a subset of the original variables, i. e. the pattern. The reduced problem in the pattern space is exhaustively solved with a breadth-first search starting backwards from the abstract goal state and storing the optimal distances for every abstract state in the PDB. These precomputed h-values from the pattern space can be accessed in the search process in constant time, making PDBs a most efficient heuristic.

Many application domains like planning, optimization, model checking, or sequence alignment benefit from the use of PDBs. We have chosen the sliding-tile puzzle as our model problem. In an *n-puzzle*, a square tray is filled with n tiles, leaving one spot empty so that an adjacent tile can be shifted into it. The task is to find a shortest path from a given configuration to the goal configuration in which all tiles in the tray are sorted.

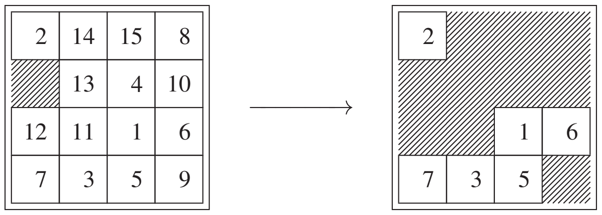


Figure 1: A configuration of the 15-puzzle as seen by the $\{1, 2, 3, 5, 6, 7\}$ APDB

Finding out what information to store in a PDB was the subject of many research projects. The earliest pattern databases (Culberson and Schaeffer 1996) contain lower bound heuristics derived by counting for each configuration the number of moves of pattern tiles and non-pattern tiles to the goal position. As in practice no single PDB can be optimal for the whole search space, often multiple PDBs are accessed. Taking the maximum h-value from several PDBs gives an admissible (non-overestimating) heuristic that can be used to compute $f(n) = g(n) + h(n)$ in an A* search, where $g(n)$ is the cost to node n and $h(n)$ is the estimated cost from n to a goal node.

Six years later, Korf and Felner (2002) introduced the more powerful *disjoint PDBs* which were later renamed *additive PDBs (APDBs)*. They allow us to sum the h-values of disjoint PDBs, thereby providing an often tighter lower bound on the true goal-distance. APDBs are constructed by tracking only the pattern tiles and disregarding the rest. The blank is not treated as a tile; methods to exploit its location for higher h values were designed (Felner, Korf, and Hanan 2004; Felner et al. 2007; 2004; Helmert, Sturtevant, and Felner 2017), but lead to inconsistent PDBs.

Overview of Contributions

The remainder of this paper is structured along our main contributions.

In Sec. 2 we extend the previous work on APDBs by not only computing the maximum distance to the blank (*zero-tile*) to its destination, but by also tracking the regions in

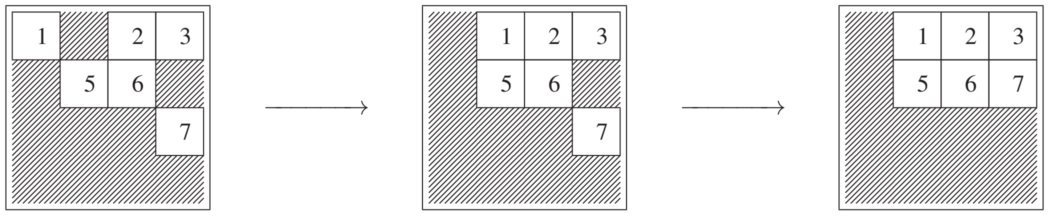


Figure 2: A partial puzzle configuration in the $\{1, 2, 3, 5, 6, 7\}$ APDB with $h = 2$. Regardless of where the zero tile is located, more than the two moves depicted here are needed to actually transition into the solved partial configuration.

which the blank may reside. The resulting *zero-aware pattern database* (ZPDB) is both admissible and consistent. This improves the pruning power 1.61 fold in the 25-puzzle and we conjecture that even better gains can be expected in larger search spaces.

In Sec. 3 we make use of the fact that the search space of sliding tile puzzles is bipartite, a property that transfers to APDBs and ZPDBs. We use this to demonstrate that the previously best compression rate of 1.6 bit per entry (Breyer and Korf 2010a) can be reduced to 1 bit.

In Sec. 4 we search for partitionings of the 24 puzzle's tiles into 4 PDBs of 6 tiles. Modelling this problem as a matching on hypergraphs, we determine the 6-tile partitioning with maximum average h-value and compare it to the one used in the literature (Korf and Felner 2002; Breyer and Korf 2010a; Felner et al. 2004).

In Sec. 5 we follow the observation that the maximum h-value of several small PDBs often perform better than using a single large PDB (Holte et al. 2004) because it reduces the number of low h-values which are most critical for the search efficiency. As the selection of an optimal collection of PDBs is still an open question, we present a heuristic approach to select a good set, reducing node expansions by a factor of 5.04 compared to a single PDB set.

2 Zero-Aware Additive Pattern Databases

An *Additive Pattern Database* (APDB) as introduced by Korf and Felner (2002) is a lookup table that stores for some set of puzzle tiles the number of moves needed to transition these tiles into their goal positions, disregarding the position of other tiles, see Fig. 1. Clearly, this number is a lower bound for the number of moves needed to solve the full puzzle. In comparison to the earlier non-additive pattern databases introduced by Culberson and Schaeffer (1996), the h-values predicted by multiple APDBs for pairwise disjoint tile sets can be added to form an admissible h-value that is generally much higher than the h-value of any individual additive or non-additive pattern database, leading to a search performance that is often at least an order of magnitude better than all previous heuristics (Korf and Felner 2002).

The performance of disjoint pattern databases was further improved by *blank compression* (Felner, Korf, and Hanan 2004; Felner et al. 2004) where the distance for each possible location of the blank spot or *zero-tile* is considered, but only the minimum of them is stored. As apparent from Fig. 2, this can improve the h-value substantially.

Rethinking the Blank

In contrast to previous work (Felner, Korf, and Hanan 2004; Felner et al. 2007; 2004; Helmert, Sturtevant, and Felner 2017) we have analyzed how well pattern databases perform when storing a separate entry for each possible position of the zero tile. We believe this approach is interesting, because it gives a higher h-value with the same pattern database without sacrificing additivity or consistency, enabling further improvements that might require either. We call an additive pattern database that keeps track of the zero tile a *zero-aware additive pattern database (ZPDB).*

As observed by Felner (2001) it is generally not necessary to store one entry for each possible position of the zero tile. Instead we determine which connected regions the zero tile could be located in and store one entry for each such *zero-tile region* (cf. Fig. 3).

Fortunately, only few partial configurations have many zero-tile regions, while a majority of them only have one or two. Tab. 1 shows that the increase in memory space needed to track the zero-tile regions is moderate. For example, configurations of 6 tiles in the 24-puzzle have only 1.42 regions on average. Storing one entry for each region increases the storage requirement from $121.60\,\mathrm{MiB}$[1] to just $172.62\,\mathrm{MiB}$, a very feasible amount in times of growing RAM sizes.

A similar approach has been used by Döbbelin, Schütt, and Reinefeld (2013) to generate APDBs with blank compression. Instead of storing one entry for each zero-tile region, open and closed lists for each zero-tile region are kept as bit maps. Their largest APDBs have 9 tiles for the 24-puzzle where up to 8 zero-tile regions can occur, allowing them to store the bit maps for the open and closed lists in one byte per APDB entry.

Fast and Compact Mapping

Prior work (Felner, Korf, and Hanan 2004; Felner et al. 2004) has distinguished two primary representations for pattern databases. In the *sparse mapping* a k-tile APDB for a puzzle with n grid locations is represented as a k-dimensional array with n^k entries where each entry stores the h-value for the configuration with tiles at the grid locations indicated by the array indices. In the *compact mapping*, a perfect hash function is employed as a bijective map between puzzle configurations and $\{0, 1, \ldots, s-1\}$ where s is the number of configurations in the database (see Tab. 1). *In-*

[1] $1\,\mathrm{MiB} = 2^{20}$ bytes, $\quad 1\,\mathrm{GiB} = 2^{30}$ bytes

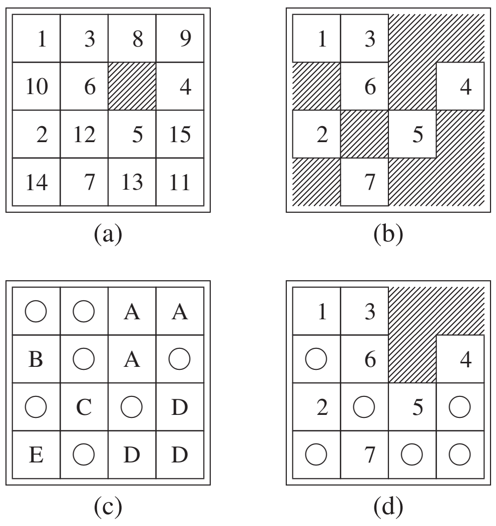


Figure 3: (a) A configuration of the 15-puzzle (b) as seen by the $\{1, 2, 3, 4, 5, 6, 7\}$ APDB (c) its possible zero-tile regions and (d) as seen by the corresponding ZPDB with the blank in region A.

version tables (Knuth 1998) are typically employed for this purpose.

While sparse mappings have a significantly better runtime performance, they waste a considerable amount of memory space without gaining additional information or improving h-values. A 6-tile APDB in sparse mapping for the 24-puzzle contains as much as 47.78% invalid entries, climbing to 71.42% for an 8-tile APDB. Even more space is wasted for ZPDBs as we have to account for the maximum, not average number of zero-tile regions. While this waste of space may be acceptable for searches using few small pattern databases, the exploration of larger pattern databases or collections of pattern databases is hindered. For this reason we have not further considered sparse mappings.

Using ZPDBs instead of APDBs increases the storage requirements by only a modest amount, improves the h-values significantly and does not block the way to further optimisations.

To represent ZPDBs by means of a compact mapping, we have developed a reasonably fast perfect hash function (Clausecker 2017) that is also applicable to APDBs. The key idea, illustrated in Fig. 4, is to represent a k-tile ZPDB entry as a tuple (m, p, r) where m indicates which grid locations are occupied by tiles in the configuration, p indicates how the tiles are permuted within these grid locations, and r indicates which region the zero tile is located in. The number of zero-tile regions depends only on m. This allows to split the database into $\binom{n}{k}$ *cohorts*, each being represented by a rectangular array containing the ZPDB entries for all permutations and zero-tile regions within this cohort. A 6-tile ZPDB of the 24-puzzle, for example, can be split into $\binom{24}{6} = 134\,596$ cohorts.

For a given configuration, m and p can be computed rapidly using bit-fiddling algorithms. The value of r and the number of zero-tile regions for the given m are looked up in an auxiliary table that is shared among all pattern databases of the same tile count. While the treatment of r is specific to n-puzzles, the remaining scheme is general and readily transfers to other problems and state abstractions.

k	APDB size	ZPDB size	avg	max
2	600	608	1.01	2
3	13 800	14 472	1.04	2
4	303 600	339 048	1.12	3
5	6 375 600	7 871 280	1.23	4
6	127 512 000	181 008 000	1.42	5
7	2 422 728 000	4 066 655 040	1.68	6
8	43 609 104 000	87 358 400 640	2.00	7
9	741 354 768 000	1 759 513 674 240	2.37	8
10	11 861 676 288 000	32 787 717 580 800	2.76	10
11	177 925 144 320 000	560 680 553 664 000	3.15	11
12	2 490 952 020 480 000	8 749 801 518 796 800	3.51	13

Table 1: The number of configurations in any k-tile APDB and ZPDB for the 24-puzzle and the average and maximal number of zero-tile regions per APDB configuration.

Transpositions

The first work (Korf and Felner 2002) on additive pattern databases already used transpositions and rotations to map puzzle configurations in the three congruent PDBs in their PDB set for partitioning (a) from Fig. 6 to just one database. At the same time, they did another set of PDB lookups with the transposed configuration, taking the maximum of the two h-values to improve the heuristics. In our research, we too used this method called *transposition search* for the figures in this paper. Furthermore, we adapted rotations and transpositions to ZPDBs.

Pattern databases for sliding tile puzzles on quadratic trays can be rotated in increments of $90°$ and transposed according to their symmetry group in 8 ways with their tiles renumbered appropriately. Let π be the tile permutation of some puzzle configuration and let τ be some combination of rotations and transpositions. Then for APDBs,

$$\pi_\tau = \tau \circ \pi \circ \tau^{-1} \tag{1}$$

is the configuration we get when applying τ to π. If we look up π_τ in a pattern database for which this transformation is valid, we get an admissible additive h-value.

For ZPDBs, we have to ensure that the zero tile stays in place, giving

$$\pi_\tau = \big(0, \tau(0)\big) \circ \tau \circ \pi \circ \tau^{-1} \tag{2}$$

where $\big(0, \tau(0)\big)$ indicates an exchange of the zero tile with what the zero tile is mapped to by τ. This maps the zero tile back to where it was before the transformation. This works for all pattern databases valid for this transformation as they cannot contain tile $\tau(0)$.

Transformations are not always possible: no tile in a PDB can be rotated to become the empty spot. Transformed ZPDBs in addition yield wrong values if the region containing the zero tile changes. Accounting for these possibilities,

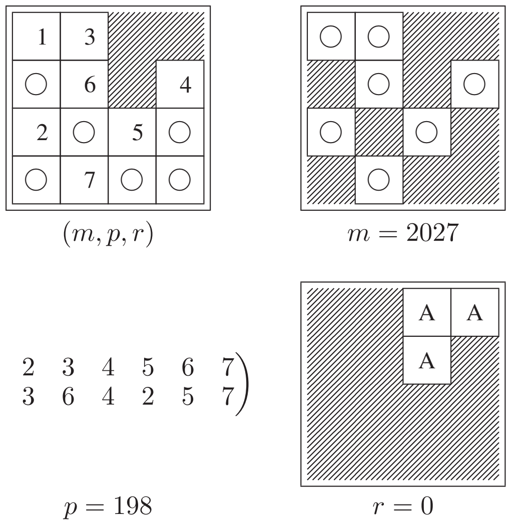


Figure 4: The 15-puzzle configuration from Fig. 3 as seen by the $\{1, 2, 3, 4, 5, 6, 7\}$ ZPDB and the components of the corresponding index.

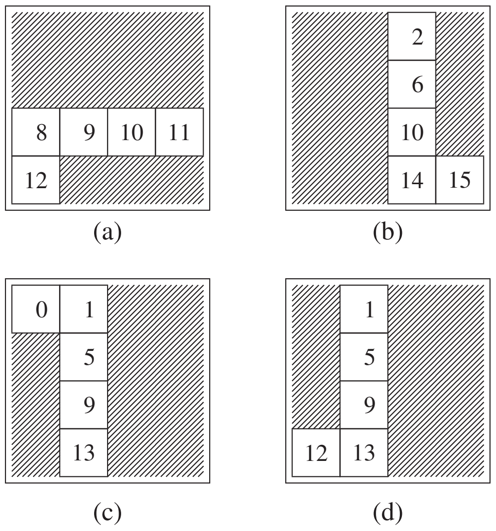


Figure 5: (a) Tiles of the $\{8, 9, 10, 11, 12\}$ PDB (b) valid rotation by $270°$ (c) invalid rotation by $90°$ (0 is not a tile) and (d) valid transposition (yields a different ZPDB)

the $\binom{24}{6} = 134\,596$ PDBs with 6 tiles fall into $22\,440$ classes of APDBs or $29\,285$ classes of ZPDBs equivalent under transposition and rotation.

A related strategy called *dual search* (Felner et al. 2005; Zahavi et al. 2008) was evaluated but ultimately rejected as it does not allow efficient use of bitmapped pattern databases as introduced in Sec. 3.

Empirical Results

We compared the performance of the described APDBs and ZPDBs on a system with Intel Xeon E3-1290 v2 running at 3.70 GHz. Our implementation performs 8.88 million APDB lookups and 8.16 million ZPDB lookups per second to 6-tile pattern databases of the 24-puzzle. APDB lookups are about 8% faster as they do not require table lookups for zero-tile regions and cohort table sizes.

The slightly slower lookup speed of ZPDBs is more than compensated by their better pruning power. Comparing the performance of partitioning (a) from Fig. 6 with and without blank compression (see Fig. 7), we found a 1.61-fold ($\sigma = 0.40$) node count reduction on average. In the best case, a reduction by 3.63 is achieved and in the worst case still 1.19. This is more than double the lookup overhead of ZPDBs over APDBs, making ZPDBs always beneficial if there is enough memory space.

Note that our results nicely confirm the prediction of Korf (2007) that the performance gain should be about proportional to the size increase of the PDB: In case of the 6-tile PDB the mentioned improvement by a factor of 1.61 is paid by a 30% increase in space consumption shown in Tab. 1.

3 1-Bit Pattern Databases

Breyer and Korf (2010a) introduced a compact lossless representation for consistent pattern databases that reduces the amount of storage required for a consistent pattern database to 1.6 bits per entry. But as they argue (ibid.), this technique does not help in compressing the h-values of inconsistent 6-tile APDBs produced by the method of Korf and Felner (2002). This is because the high inconsistency rate[2] requires 4 bits per entry, which can also be achieved by simply storing the addition to the Manhattan distance per entry.

The key idea of Breyer et al. was to use the property of consistent heuristics that the h-values of adjacent search nodes differs by at most 1. Given that in a typical search algorithm, h-values are looked up along a path through the search space, it suffices to quickly decide whether the h-value of a node is higher, equal, or lower than the h-value of its predecessor. This can be done by storing the h-value modulo 3, giving a storage requirement of $\log_2(3) = 1.59$ bits per entry. By storing 5 entries in a byte, a density of 1.6 bit per entry is achieved using $3^5 = 242$ of the 256 possible bit patterns in a byte.

We improved this representation to 1 bit per entry, reducing the PDB size while at the same time avoiding the complicated bit-stuffing needed for the 1.6 bit representation. Our improvement is based on the observation that the search spaces of both n-puzzles and PDBs for them are bipartite, a property shared with many other puzzles such as the Rubik's Cube (with quarter turn metric). Instead of storing the remainder of the h-value modulo 3, we store the remainder modulo 4 and discard the least significant bit as it is known to flip with every step through the search space; if needed, it can be reconstructed from the configuration's parity. The resulting representation needs just 1 bit of storage per entry.

This bitmap representation reduces the storage requirements of our implementation 8-fold for a 6-tile ZPDB of the 24-puzzle, consuming just $21.58\,\text{MiB}$ storage for its 181 million entries. Similarly, an 8-tile ZPDB consumes just

[2]The inconsistency rate is the maximum distance of the h-values of any two successive nodes (Zahavi et al. 2007).

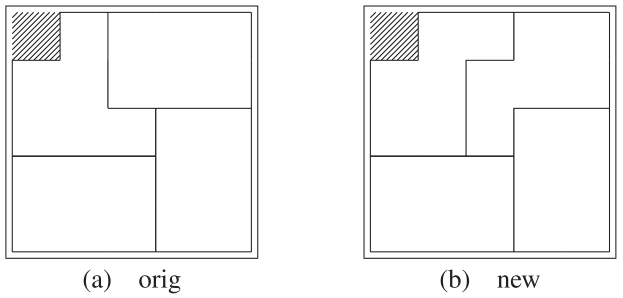


Figure 6: Two partitionings of the 24-puzzle into 6-tile PDBs: (a) the partitioning commonly used in literature ($\bar{h} = 81.82$) and (b) the optimal partitioning ($\bar{h} = 82.06$).

$10.17\,\mathrm{GiB}$ memory space, allowing us to explore searches with 8-tile ZPDBs on standard computers.

To crosscheck the effectiveness of our 1 bit data representation, we applied the *zstandard* general purpose compression algorithm (Collet and Kucherawy 2018) to our ZPDB. Surprisingly, we were able to further compress the described 6-tile ZPDBs to $5.66\,\mathrm{MiB}$ on average. This suggests that an even better representation may reduce the size further.

Performance

The 1-bit compression gives an 8-fold reduction of the memory space compared to a representation with 1 byte per entry without loss of heuristic performance and with only insignificant loss of speed. When additionally applying the *zstandard* compression algorithm on disk and decompressing the required PDBs once at the beginning of the search, the disk usage was reduced 30.5-fold compared to storing uncompressed PDBs with one byte per entry.

4 Finding Optimal Partitionings

While there are $\binom{24}{6}\binom{18}{6}\binom{12}{6}\binom{6}{6}/4! = 9.62 \times 10^{10}$ different partitionings of the 24-puzzle's tiles into 6-tile pattern databases, not much research has gone into analysing which of them perform best. Indeed, prior research on the 24-puzzle generally uses partitioning (a) from Fig. 6 first described by Korf and Felner (2002). This partitioning was originally chosen because it requires only two distinct tables to be kept in memory and went with the intuition that compact tile regions capture the highest amount of interactions between tiles. Later it was speculated (Breyer and Korf 2010a; Felner et al. 2004) that this is probably the most effective 6-6-6-6 partitioning, but no evidence was given.

It is hard to say whether one partitioning is better than another. In the following, we judge the precision of the heuristic given by a partitioning using its average h-value $\bar{h}$, assuming that a PDB set with higher $\bar{h}$ performs better than one with lower $\bar{h}$.

While this is not the most accurate measure, it is easy to estimate by uniform sample and widely used in the literature. A better measure of precision is given by Korf and Reid (1998), but is difficult to compute for PDB collections and does not give a single number to compare.

Globally Optimal Partitionings

To find the 6-6-6-6 PDB set with maximum $\bar{h}$, we first generated all 6-tile ZPDBs and their average h-value $\bar{h}$. This was done by generating all 29285 distinct 6-tile ZPDBs and storing them as compressed bitmaps using the method from Sec. 3, requiring a total of $161.79\,\mathrm{GiB}$ storage space.

Then we computed the partitioning with the best $\bar{h}$ by first finding the best partitionings for all half trays (12 tiles) into two PDBs with 6 tiles each and matching them with the best partitioning of the other halves. This method reduces the number of partitionings that need to be checked from the aforementioned 9.62×10^{10} to $\binom{24}{12}\binom{12}{6}\binom{6}{6}/2! = 1.25 \times 10^9$ half tray partitionings and $\binom{24}{12}\binom{12}{12}/2! = 1.35 \times 10^6$ matches between halves while still being guaranteed to find the global optimum.

As a result of the search we found the "new" partitioning (b) in Fig. 6 with $\bar{h} = 82.06$ compared to $\bar{h} = 81.82$ of partitioning (a) which was previously used in the literature. However, the empirical results in the Appendix shows that partitioning (b) beats partitioning (a) in only 28 of Korf's 50 instances. Despite having a significantly higher average h-value, its performance is comparable to partitioning (a). This calls into question whether $\bar{h}$ is really a suitable measure for the quality of a heuristic function.

Locally Optimal Partitionings

The effectiveness of pattern databases varies depending on the puzzle configuration we try to solve. For example, as seen in the empirical results listed in the Appendix, partitioning (a) often beats partitioning (b) despite its lower $\bar{h}$. We believe this might be due to $\bar{h}$ being an imperfect quality indicator and different tile interactions mattering in different parts of the search space. To capture these differing interactions, there is likely no single optimal partitioning for the entire search space.

To address this issue, we extend the method from Sec. 4 to find partitionings optimal for a single configuration as opposed to the whole search space. As a simple measure of optimality, we try to find a partitioning with maximal h-value for the chosen configuration. In case of a likely tie, the partitioning with maximal $\bar{h}$ is chosen. This partitioning is then used for an IDA* search.

In a way this approach is a variant of the *Higher Order Heuristics* (Korf and Taylor 1996) which first used a maximum matching on ordinary graphs to select what can today be seen as a partitioning made of 2-tile PDBs, recomputing this matching for every node in the search tree. While this generally yields less node expansions than the linear conflict heuristic (Hansson and Mayer 1992), the savings are eaten up by the computationally expensive matching. Our method differs in that we compute the expensive matching once and use the partitioning found for the rest of the search.

5 Pattern Database Collections

In the previous section, we found that no single partitioning is optimal for the entire search space. To remedy this issue, it is intuitive to take the maximum of several partitionings known to be optimal in different parts of the

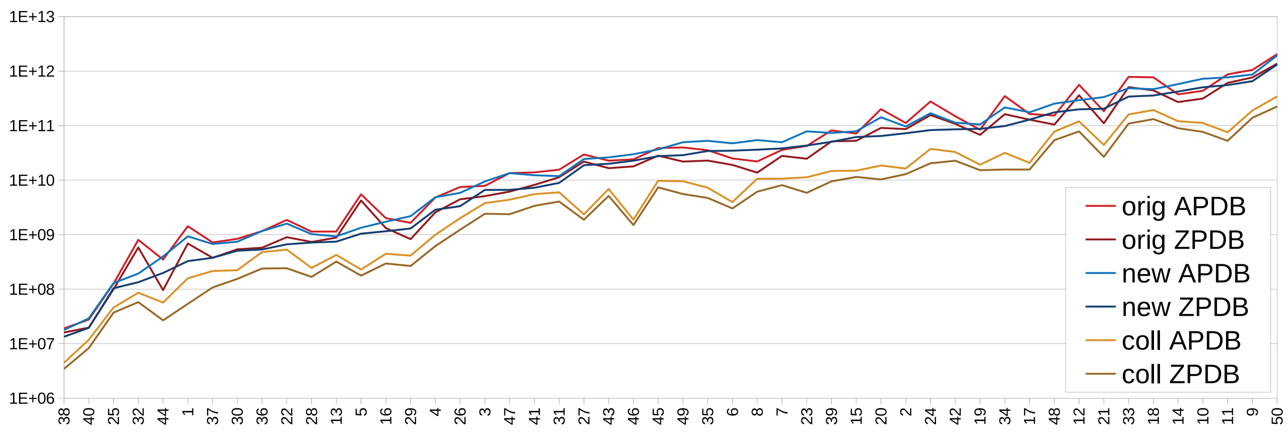


Figure 7: IDA* node expansions using 6-tile APDBs and ZPDBs on Korf's fifty puzzle instances (Korf and Felner 2002). 'Orig' is the original 6-6-6-6 partitioning in Fig. 6(a), 'new' is the partitioning in Fig. 6(b), and 'coll' is the collection in Fig. 8. The instances are sorted by *new ZPDB* nodes.

search space. We call such a set of partitionings a *pattern database collection*[3]. The effectiveness of taking the maximum of several PDB heuristics has frequently been the focus of research (Holte et al. 2004; Korf 2007; Breyer and Korf 2010b). In this paper, we try to contribute some insights into what makes a good PDB collection.

Finding Good Collections

Holte et al. (2004) argue that maximizing over several smaller PDBs "can make the number of patterns with low h-values significantly smaller than the number of low-valued patterns in the larger pattern database" and that "eliminating low h-values is more important for improving search performance than retaining large h-values." We followed their ideas and focused our efforts on building effective collections of 6-6-6-6 partitionings for the 24-puzzle. In doing so, we took care to keep the number of distinct partitionings at a reasonable size, as each additional table lookup slows down the search. Following the principle of diminishing returns, we add partitionings only as long as the speedup gained by the reduced node expansions outweighs the slowdown from looking into multiple PDBs. As in Sec. 4, we measure the performance by the average h-value $\bar{h}$.

Which new partitioning should be added to a given collection? Consider the h-value given by a PDB collection for a single puzzle configuration. For this h-value, the collection has to contain at least one partitioning that provides this h-value. We call such a partitioning a *support* of this configuration's h-value. All but one support may be removed from the collection without reducing the h-value of this configuration. With this intuition we use the following informal scheme (Clausecker 2017) to find a good collection of pattern databases:

1. Start with a set of known good partitionings,
2. sample the h-values of $n = 100\,000\,000$ random puzzles,
3. for each partitioning count how often it was an h-value's sole support,
4. remove partitionings from the collection which rarely supported the maximum h-value,
5. add new partitionings to the collection to replace the removed ones,
6. repeat steps 2–5 until no further improvements are found.

Fig. 8 depicts a PDB collection with $\bar{h} = 83.08$ that was constructed with the described method. This is a considerable improvement over the best single 6-tile partition (b) in Fig. 6 with $\bar{h} = 82.06$.

For partitionings (c) and (d) in Fig. 8 the sole-support frequency seems low; this is explained by observing that most configurations supported by one of the two is also supported by the other; removing (c) leads to a sole-support frequency of 3.62% for (d) while removing (d) leads to a frequency of 2.74% for (c). As no PDB lookups are saved by removing just one of the two, both are kept in the collection.

Performance

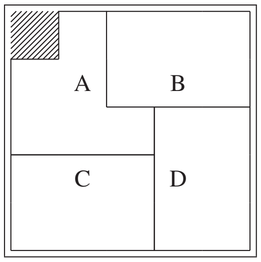
Applying the PDB collection from Fig. 8 to Korf's 50 instances (Korf and Felner 2002) as ZPDBs, we find that the number of expanded IDA* nodes is reduced 5.04-fold ($\sigma = 2.64$) on average compared to using just partitioning (b) from Fig. 6, as shown in the Appendix. The least reduction found was 2.15 while the highest reduction was 15.20. This is contrasted with a 3.5-fold increase in runtime per expanded node due to the need to query more pattern databases.

As seen nicely in Fig. 7, the PDB collection keeps all valleys from other heuristics while avoiding the peaks.

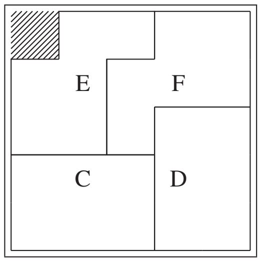
6 Related Work

Much research has been spent on PDBs since their introduction by Culberson and Schaeffer (1996) and extension

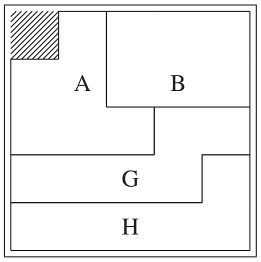
[3]In our previous work (Clausecker 2017) we used the term *pattern database catalogue*. This was changed to match the established terminology from Scherrer, Pommerening, and Wehrle (2015).



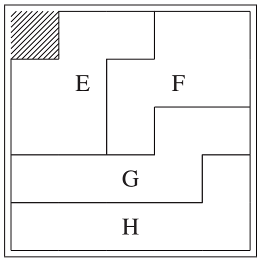
(a) 2.21%



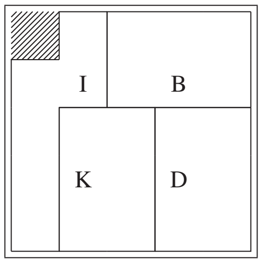
(b) 4.49%



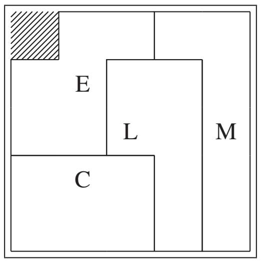
(c) 0.83%



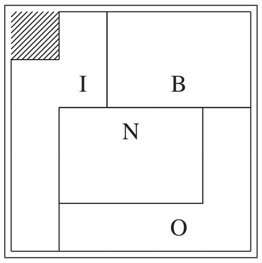
(d) 1.72%



(e) 5.18%



(f) 7.15%



(g) 4.77%

Figure 8: A PDB collection created with the method in Sec. 5. For each partitioning, the frequency of this partitioning being the sole support of a configuration's h-value is given.

to additive patterns by Korf and Felner (2002); a broad overview can be found in (Edelkamp and Schrödl 2012). An idea similar to ZPDBs has previously been considered by Felner (2001). In contrast to our ZPDBs, previous additive PDBs (Felner, Korf, and Hanan 2004; Felner et al. 2007; 2004) used lossy compression of the blank, thereby reducing the space consumption at the cost of consistency. The more general concept of *min compression* (Helmert, Sturtevant, and Felner 2017) builds space-efficient, coarser abstractions by compressing multiple finer ones. Min-compressed PDBs can be as powerful as regular PDBs when using compatible compression regimes.

Instance-dependent PDBs (Zhou and Hansen 2004; Felner and Adler 2005) are based on the observation that a PDB constructed for a specific problem instance often contains tighter bounds than a regular PDB. Clearly, instance-dependent PDBs benefit also from the presented improvements, namely ZPDBs, 1-bit compression and collections.

Our 1-bit compressed ZPDB entries are applicable to domains with bipartite search spaces only. *Entry compression* (Felner et al. 2004; 2007) is a more general scheme. It is motivated by the observation that nearby entries in the PDB are often correlated, making it economical to store their minimum in a single entry. *Value compression* (Sturtevant, Felner, and Helmert 2017) saves storage space in applications with a large value range by keeping an interval rather than a precise h-value per entry. In contrast to static pattern partitioning schemes, *dynamically partitioned PDBs* (Felner, Korf, and Hanan 2004) divide the problem into disjoint subproblems for each state of the search dynamically which allows application specific value compression.

Symmetries in the PDB (reflections or rotations) have been exploited since (Culberson and Schaeffer 1996). Dual lookups (Felner et al. 2005; Zahavi et al. 2008) add a new kind of symmetry by exchanging objects and their locations: Instead of only checking a PDB for the location of the pattern tiles and their minimal cost to their goal positions, it can also be asked which tiles are currently in the pattern spots and how many moves are needed to transform them into their goal positions. This provides more information from the same PDB, but it makes the heuristic inconsistent.

Cost partitioning (Pommerening et al. 2015) allows the same tile to appear in multiple PDBs of a single set without losing additivity by splitting the cost of moving tiles in multiple PDBs among them. This can be used to increase the number of additive PDBs in a PDB set without having to reduce the number of tiles in each.

7 Conclusion

To prepare for the solution of previously intractable state space search problems such as the 35-puzzle or beyond, we have tried to find better heuristics at lower memory usage without compromising the additivity property. To this end, we introduced a zero-aware pattern database (ZPDB) that requires 35% less node expansions than the previously best 6-tile PDB on the 24-puzzle. Our scheme for generating optimal partitionings of additive PDBs improved the number of expanded nodes 1.15-fold, and maximizing the heuristic over a reasonably sized collection of PDBs improves the search performance by an additional factor of 5.04. Putting all enhancements together, we achieve an 8.59-fold performance gain with lower memory consumption due to the presented 1-bit compression.

Acknowledgements

We thank Thorsten Schütt for many fruitful discussions and the unknown reviewers for their hints that helped to improve the paper. We also thank the HLRN Supercomputer Alliance for supporting us with computer time.

Appendix: Empirical Results

Expanded IDA* nodes for Korf's 50 instances (Korf and Felner 2002) for the two 6-6-6-6 partitionings (a) and (b) from Fig. 6 and the collection from Fig. 8. Each heuristic was tried both as an APDB and as a ZPDB heuristic. Transposition search was used for all searches.

no	*(a) APDB*	*(b) APDB*	*coll. APDB*
	(a) ZPDB	*(b) ZPDB*	*coll. ZPDB*
1	1 425 875 140	931 043 123	158 198 729
	686 069 547	326 601 592	53 848 738
2	111 641 207 465	96 358 253 587	16 327 196 265
	86 236 390 925	72 437 178 388	12 889 231 618
3	7 853 908 324	9 456 518 159	3 780 562 643
	5 084 007 427	6 615 630 803	2 418 437 615
4	4 797 694 125	4 857 570 283	992 148 089
	2 560 650 833	2 864 982 875	616 554 044
5	5 484 055 692	1 335 690 599	229 080 119
	4 227 592 928	1 041 713 929	176 967 150
6	24 962 231 129	47 308 485 686	3 963 422 179
	19 005 789 135	34 660 552 606	3 025 112 424
7	35 855 872 236	49 461 032 293	10 646 812 645
	27 830 182 298	38 145 022 619	8 087 803 174
8	21 995 039 727	54 198 594 778	10 609 670 443
	13 732 776 667	36 152 691 313	6 135 164 661
9	1 051 076 535 652	865 041 098 913	189 004 935 961
	764 801 552 269	653 580 333 283	139 257 503 265
10	435 944 577 737	724 718 627 724	111 771 822 653
	313 592 839 790	505 354 105 926	76 580 113 103
11	872 838 759 795	769 063 525 838	75 223 483 796
	608 620 352 369	554 454 780 063	52 614 901 639
12	561 789 705 040	293 053 650 437	119 682 603 137
	360 125 405 661	198 891 918 955	78 270 149 630
13	1 138 778 240	929 184 542	423 471 010
	880 251 759	744 016 730	319 952 186
14	374 259 092 507	577 921 827 058	120 333 106 357
	270 275 552 837	421 817 894 745	89 543 926 428
15	71 416 034 033	78 573 564 626	14 894 248 996
	52 627 610 231	61 800 659 906	11 436 756 369
16	2 009 280 766	1 723 446 084	446 163 669
	1 317 282 819	1 152 536 662	295 627 821
17	163 150 026 745	175 183 740 510	20 751 095 783
	128 477 262 048	128 670 468 309	15 640 095 668
18	769 009 927 981	466 116 464 112	193 152 308 945
	442 000 131 063	356 575 497 661	131 573 822 523
19	83 831 422 428	104 888 757 443	19 177 320 317
	67 587 576 627	87 235 543 167	15 218 497 346
20	200 450 742 161	142 067 084 954	18 519 458 230
	90 680 187 246	64 495 996 359	10 284 615 128
21	183 592 138 998	334 128 876 576	44 139 271 208
	110 185 422 023	204 093 630 206	26 683 342 659
22	1 859 456 487	1 591 270 357	531 281 516
	897 004 462	664 019 431	241 956 643
23	42 329 759 562	78 450 249 077	11 282 628 160
	24 670 022 140	43 140 430 080	5 855 445 178
24	277 401 780 209	168 689 170 058	37 452 473 860
	155 841 217 499	82 704 120 109	20 358 642 119
25	126 495 063	129 418 572	46 163 378
	99 852 238	104 008 514	37 172 997
26	7 471 162 094	5 832 854 155	2 000 152 241
	4 455 034 712	3 324 120 394	1 228 120 251
27	29 521 355 607	24 423 566 554	2 353 368 855
	21 884 892 243	18 769 968 453	1 872 729 978
28	1 135 190 134	1 021 530 887	245 009 685
	734 679 140	715 360 384	167 959 834
29	1 643 800 509	2 180 616 359	411 613 829
	826 325 905	1 295 402 939	265 641 183
30	835 370 084	740 896 221	222 485 745
	536 978 989	506 037 398	154 117 709
31	15 535 709 218	11 824 581 381	5 974 528 165
	11 288 984 012	8 893 904 619	4 051 420 238
32	801 631 112	192 929 390	85 977 882
	581 709 365	134 222 892	57 801 737
33	783 252 065 384	485 155 122 422	159 949 568 994
	509 013 889 937	341 051 847 307	108 729 245 793
34	348 148 686 761	215 398 856 371	31 657 564 408
	161 982 500 255	98 135 968 195	15 671 916 687
35	35 374 979 892	52 573 346 683	7 269 270 352
	22 812 372 384	34 194 869 391	4 717 898 962
36	1 158 981 872	1 163 256 639	477 288 759
	574 804 537	533 634 184	238 596 946
37	717 363 901	673 011 310	215 325 614
	375 485 935	375 856 349	107 340 791
38	19 102 710	17 828 584	4 464 548
	16 044 566	13 407 785	3 450 832
39	81 665 942 816	73 402 059 725	14 718 413 569
	51 298 914 727	50 465 525 984	9 492 310 290
40	27 620 434	29 089 022	11 788 834
	19 694 137	19 497 302	8 296 164
41	13 857 238 379	12 294 855 797	5 535 159 350
	8 189 164 544	7 241 780 094	3 375 922 989
42	148 210 529 197	112 932 776 099	32 753 208 287
	107 553 463 207	85 379 955 688	22 680 138 294
43	22 730 491 518	26 124 678 258	6 884 944 791
	16 561 659 262	19 946 047 570	5 137 292 767
44	348 918 778	397 645 233	56 723 555
	96 000 581	198 161 383	26 772 635
45	38 641 448 004	36 554 530 360	9 736 996 869
	28 087 680 065	27 490 145 174	7 360 265 880
46	24 041 904 390	29 799 529 904	1 884 568 572
	17 860 977 428	22 739 945 231	1 495 685 257
47	13 462 467 532	13 430 174 953	4 382 096 352
	6 156 114 730	6 625 847 130	2 364 729 584
48	152 777 088 177	253 256 991 057	77 548 336 397
	104 209 054 684	175 009 505 182	54 136 974 740
49	39 673 054 257	49 469 627 208	9 550 269 592
	21 901 326 342	28 697 056 739	5 563 996 366
50	2 047 762 761 989	1 936 036 475 217	342 577 968 758
	1 371 409 134 448	1 320 463 248 893	224 259 011 348

References

Breyer, T. M., and Korf, R. E. 2010a. 1.6-Bit Pattern Databases. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 39–44. AAAI Press / The MIT Press.

Breyer, T. M., and Korf, R. E. 2010b. Independent Additive Heuristics Reduce Search Multiplicatively. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 33–38. AAAI Press / The MIT Press.

Clausecker, R. K. P. 2017. Notes on the Construction of Pattern Databases. ZIB Report 17-59, Zuse Institute Berlin.

Collet, Y., and Kucherawy, M. S. 2018. Zstandard Compression and the application/zstd Media Type. RFC 8478, RFC Editor.

Culberson, J. C., and Schaeffer, J. 1996. Searching with Pattern Databases. *Advances in Artifical Intelligence* 402–416.

Döbbelin, R.; Schütt, T.; and Reinefeld, A. 2013. Building Large Compressed PDBs for the Sliding Tile Puzzle. ZIB Report 13-21, Zuse Institute Berlin.

Edelkamp, S., and Schrödl, S. 2012. *Heuristic Search – Theory and Applications.* Morgan Kaufmann.

Felner, A., and Adler, A. 2005. Solving the 24 Puzzle with Instance Dependent Pattern Databases. In *Abstraction, Reformulation and Approximation*, 248–260. Springer.

Felner, A.; Meshulam, R.; Holte, R. C.; and Korf, R. E. 2004. Compressing Pattern Databases. In *Proceedings of the National Conference on Artificial Intelligence*, 638–643. AAAI Press / The MIT Press.

Felner, A.; Zahavi, U.; Schaeffer, J.; and Holte, R. C. 2005. Dual Lookups in Pattern Databases. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 103–108.

Felner, A.; Korf, R. E.; Meshulam, R.; and Holte, R. 2007. Compressed Pattern Databases. *Journal of Artificial Intelligence Research* 30:213–247.

Felner, A.; Korf, R. E.; and Hanan, S. 2004. Additive Pattern Database Heuristics. *Journal of Artificial Intelligence Research* 22:279–318.

Felner, A. 2001. *Improving search techniques and using them on different environments*. Ph.D. Dissertation, Bar-Ilan University.

Hansson, O., and Mayer, A. 1992. Criticizing Solutions to Relaxed Models Yields Admissible Heuristics. *Information Sciences* 63:207–227.

Helmert, M.; Sturtevant, N. R.; and Felner, A. 2017. On Variable Dependencies and Compressed Pattern Databases. In *Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS 2017)*, 129–133. AAAI Press.

Holte, R. C.; Newton, J.; Felner, A.; Meshulam, R.; and Furcy, D. 2004. Multiple Pattern Databases. In *ICAPS-04 Proceedings*, 122–131. AAAI Press / The MIT Press.

Knuth, D. E. 1998. *The Art of Computer Programming*, volume 3. Addison-Wesley.

Korf, R. E., and Felner, A. 2002. Disjoint pattern database heuristics. *Artificial Intelligence* 134(1-2):9–22.

Korf, R. E., and Reid, M. 1998. Complexity Analysis of Admissible Heuristic Search. In *Proceedings of the National Conference on Artificial Intelligence*. AAAI Press / The MIT Press.

Korf, R. E., and Taylor, L. A. 1996. Finding Optimal Solutions to the Twenty-Four Puzzle. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1202–1207. AAAI Press.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Korf, R. E. 2007. Analyzing the Performance of Pattern Database Heuristics. In *Proceedings of the National Conference on Artificial Intelligence*, 1164–1170. AAAI Press / The MIT Press.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From Non-Negative to General Operator Cost Partitioning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 3335–3341. AAAI Press / The MIT Press.

Scherrer, S.; Pommerening, F.; and Wehrle, M. 2015. Improved pattern selection for PDB heuristics in classical planning. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SOCS 2015)*, 216–217. AAAI Press.

Sturtevant, N. R.; Felner, A.; and Helmert, M. 2017. Value Compression of Pattern Databases. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 912–918. AAAI Press / The MIT Press.

Zahavi, U.; Felner, A.; Schaeffer, J.; and Sturtevant, N. 2007. Inconsistent Heuristics. In *Proceedings of the National Conference on Artificial Intelligence*, 1211–1216. AAAI Press / The MIT Press.

Zahavi, U.; Felner, A.; Holte, R. C.; and Schaeffer, J. 2008. Duality in permutation state spaces and the dual search algorithm. *Artificial Intelligence* 172(4–5):514–540.

Zhou, R., and Hansen, E. A. 2004. Space-Efficient Memory-Based Heuristics. In *Proceedings of the National Conference on Artificial Intelligence*, 677–682. AAAI Press / The MIT Press.