Proceedings of the Tenth International Symposium on Combinatorial Search (SoCS 2017)

On Variable Dependencies and Compressed Pattern Databases

Malte Helmert
Dept. of Math. and Computer Science
Universität Basel
Basel, Switzerland
malte.helmert@unibas.ch

Nathan R. Sturtevant
Dept. of Computer Science
University of Denver
Denver, CO, USA
sturtevant@cs.du.edu

Ariel Felner
Dept. of Information System Engineering
Ben Gurion University
Beer-Sheva, Israel
felner@bgu.ac.il

Abstract

Pattern databases are among the strongest known heuristics for many classical search benchmarks such as sliding-tile puzzles, the 4-peg Towers of Hanoi puzzles, Rubik's Cube, and TopSpin. *Min-compression* is a generally applicable technique for augmenting pattern database heuristics that has led to marked experimental improvements in some settings, while being ineffective in others. We provide a theoretical explanation for these experimental phenomena by studying the interaction between the ranking function used to order abstract states in a pattern database, the compression scheme used to abstract states, and the dependencies between state variables in the problem representation.

Introduction

Most current algorithms for optimal state-space search are based on two ingredients: a systematic search algorithm from the A* family of algorithms (e.g., Hart, Nilsson, and Raphael 1968; Korf 1985) and an admissible heuristic function (e.g., Hart, Nilsson, and Raphael 1968; Pearl 1984). Regarding heuristic functions, *pattern databases* (PDBs), due to Culberson and Schaeffer (1996), are perhaps the most widely used ones, as they have shown to be very informative (e.g., Culberson and Schaeffer 1998; Korf and Felner 2002) and are immediately applicable in all state spaces that can be described in a factored (state-variable-based) representation (e.g., Edelkamp 2001).

Because of these attractive properties, PDB heuristics have been studied in depth by the heuristic search community, and a large body work has been published on improvements, variations and generalizations of the basic approach. Because memory limitations are the main obstacle towards the development of better PDB heuristics, a significant amount of work has dealt with the question of *PDB compression* in one form or another, including both lossless (e.g., Edelkamp 2002; Ball and Holte 2008; Breyer and Korf 2010; Sadeqi and Hamilton 2016) and lossy approaches (e.g., Felner et al. 2007; Sturtevant, Felner, and Helmert 2014; 2017).

One commonly used lossy compression technique that allows partially circumventing the memory limitations of PDB heuristics is *min-compression*: first, generate a very large PDB, for example using external search techniques, which is too large to be used directly during search. Then, compress this PDB into a smaller one that fits into main memory by combining multiple entries of the large PDB into a single entry containing the *minimum* of the combined entries. The compressed PDB represents an admissible (but not necessarily consistent) heuristic that can then be used during search.

A well-known example of min-compression is *compressing the blank* in the sliding-tile puzzle, where an initial PDB is calculated while taking into account the position of the blank tile, but all PDB entries that only differ in the blank position are then min-compressed into one entry. Experiments with min-compression have been reported for most of the common benchmark problems in heuristic search, with widely varying results. Sturtevant, Felner, and Helmert (2014) summarize the earlier results in the literature as follows:

> "This approach worked very well for the 4-peg Towers of Hanoi, for instance, but its success for the sliding tile puzzles was limited and no significant advantage was reported for the Top-Spin domain (Felner et al. 2007)."

In this short paper, we aim to provide a theoretical explanation of this experimentally observed behavior. Specifically, we aim to answer the following questions:

Q1. Under which conditions is min-compression beneficial? Can we guarantee that min-compressing a large PDB to size N leads to a heuristic which is at least as good as directly using a PDB of size N?

Q2. Why does min-compression lead to large improvements in the 4-peg Towers of Hanoi, moderate improvements in the sliding tile puzzles, and no improvement in TopSpin?

Q3. How do these results generalize to other domains?

Our study is purely theoretical: we provide no new algorithms and no new experiments. Rather, our aim is to better understand the existing algorithms and experiments reported in the literature. In doing so, we hope to provide insights on which future work on compressed PDB heuristics and related techniques can build.

State Spaces

We assume that states in the state spaces we consider are expressed as tuples over a finite set of finite-domain *state variables* V, and that pattern database heuristics are based on abstracting the state space by *projecting* to a subset of state variables $P \subseteq V$. This covers typical state space representations and pattern databases for classical benchmarks such as the 15-puzzle, the 4-peg towers of Hanoi, TopSpin, Rubik's Cube, but also general representation formalisms such as SAS$^+$ (Bäckström and Nebel 1995) and PSVN (Hernádvölgyi and Holte 1999).

For example, a state in the 15-puzzle can be represented as a tuple $\langle p_0, \ldots, p_{15} \rangle$ where p_i encodes the *position* of tile i, where p_0 is the blank position. A state in TopSpin can be represented as a tuple of positions for each token. A state in Towers of Hanoi can be represented as a tuple of positions for each disk.

In some cases, an inverse (dual) representation exists: for example, a 15-puzzle state can alternatively be represented as a tuple where the i-th entry gives the tile at the i-th position, rather than the position of the i-th tile. Which of the two forms is appropriate depends on the intended use: for example, for PDB heuristics based on ignoring certain tiles, we need a representation with one state variable per tile. One could alternatively ignore certain *positions*, in which case the dual representation would be appropriate.

Certain combinations of assignments to state variables may be *illegal*. For example, in the 15-puzzle, no two tiles can occupy the same position, so all states with $p_i = p_j$ for some $i \neq j$ are illegal. In a general representation like SAS$^+$, such constraints can be derived or imposed as *invariants*. Haslum, Bonet, and Geffner (2005) describe how *constrained PDBs* can exploit such invariants to strengthen PDB heuristics.

Formally, a state space for state variables V is given by:

- a set S of assignments to V called the (legal) *states*,
- a set $T \subseteq S \times S \times \mathbb{R}_0^+$ of (state) *transitions*, each with an associated *cost*,
- an *initial state* $s_{\text{init}} \in S$, and
- a set of *goal states* $S_{\text{goal}} \subseteq S$.

We write transitions in arrow notation as $s \xrightarrow{c} s'$ to denote a transition from s to s' with cost c and omit c where it does not matter. We assume familiarity with common heuristic search concepts such as heuristics, admissibility, consistency, dominance, and the perfect heuristic h^*.

Pattern Database Heuristics

Given a subset of state variables $P \subseteq V$ (a *pattern*), the *projection* π_P is a function that takes a state $s \in S$ and discards all information about state variables not in P. It induces an *abstract state space*, which is a state space over state variables P with states $\{\pi_P(s) \mid s \in S\}$, transitions $\{\pi_P(s) \xrightarrow{c} \pi_P(s') \mid s \xrightarrow{c} s' \in T\}$, initial state $\pi_P(s_{\text{init}})$ and goal states $\{\pi_P(s) \mid s \in S_{\text{goal}}\}$. The heuristic h_P associated with π_P is defined by $h_P(s) = h^*(\pi_P(s))$, i.e., the optimal solution cost (in the abstract space) of the projection of s.

Let S_P be the set of abstract states under projection π_P. The most common way of implementing h_P is to use a *ranking function* (minimal perfect hash function) $\mathit{rank}_P : S_P \to \{0, \ldots, |S_P|-1\}$ (e.g., Bonet 2008) to map abstract states to a contiguous range of integers, and then use a 1-dimensional array (table) T_P to store the corresponding heuristic values. That is, $T_P[i]$ stores $h^*(s_P)$, where s_P is the uniquely defined abstract state with $\mathit{rank}_P(s_P) = i$. Such tables are called *pattern databases*, which is why the overall heuristic is called a *pattern database heuristic*. In summary, we have

$$h_P(s) = T_P[\mathit{rank}_P(\pi_P(s))],$$

where in practice the combined projection and ranking function $\mathit{rank}_P \circ \pi_P$ is often computed in one step.

Compressed Pattern Database Heuristics

The idea of *(min-)compressed PDB heuristics* is to combine multiple PDB entries into one table entry in order to reduce the memory requirements of the heuristic (Felner et al. 2007).[1] Consider a PDB T_P with N entries and ranking function rank_P, let $M \leq N$, and consider a surjective *rank compression function* $\mathit{comp} : \{0, \ldots, N-1\} \to \{0, \ldots, M-1\}$ that reduces the ranks of T_P to a smaller range. We can then define a compressed table T_P^{comp} with M entries as $T_P^{\mathit{comp}}[i] = \min_{r \in \mathit{comp}^{-1}(i)} T[r]$ and define the *compressed PDB heuristic* h_P^{comp} as

$$h_P^{\mathit{comp}}(s) = T_P^{\mathit{comp}}[\mathit{comp}(\mathit{rank}_P(\pi_P(s)))].$$

It is easy to verify that h_P^{comp} is admissible if h_P is: in fact, h_P^{comp} is (weakly) *dominated* by h_P because for all ranks r, $T_P^{\mathit{comp}}[\mathit{comp}(r)] = \min_{r' \in \mathit{comp}^{-1}(\mathit{comp}(r))} T[r'] \leq T[r]$, where the inequality holds because $r \in \mathit{comp}^{-1}(\mathit{comp}(r))$.

Observe that while the ranking function is an implementation detail that does not affect the quality of the heuristic estimates for a regular PDB heuristic, it has a significant influence on the heuristic values of a compressed PDB heuristic because the combination of *comp* and rank_P determines which abstract goal distances are grouped together in the compressed table. Previous work on compressed PDBs often discussed the compression function *comp* used, but rarely discussed the ranking function. This is not sufficient for theoretically analyzing compressed PDB heuristic. We will refer to the combination of ranking and compression function as the *compression regime* used by a given compressed PDB heuristic.

Dominance of Min-Compressed Heuristics

Assume we are faced with the following decision: given a space budget of M table entries, we can either compute a (regular) coarse-grained PDB heuristic h_C with M entries based on the pattern C, or we can compute a finer-grained PDB heuristic h_F based on a larger pattern $F \supseteq C$ and

[1] More recently, this technique has been called *entry compression* to distinguish it from another PDB compression technique called *value compression*, which reduces the number of bits used to store each PDB entry rather than the number of table entries (Sturtevant, Felner, and Helmert 2017).

compress it down to size M, obtaining the compressed PDB heuristic h_F^{comp}. Which of the two heuristics is preferable? Clearly, this is a key question in the practical application of compressed PDB heuristics. So far, this question has only been studied experimentally.

Because $F \supseteq C$, h_F weakly dominates h_C: the coarser abstraction can be viewed as a further abstraction of the finer abstraction, and dominance follows from the admissibility of abstraction heuristics. We say that h_F is a *refinement* of h_C.[2] Hence, the question is in which cases the loss in accuracy due to compression when moving from h_F to h_F^{comp} is small enough not to lose the advantage of h_F over h_C.

For example, let h_F be a 13-disk PDB heuristic for 4-peg Towers of Hanoi, and let h_C be an 11-disk PDB that ignores the smallest two disks considered in F. The PDBs for h_F^{comp} and h_C have the same size if we compress by a factor of 16. Which heuristic is stronger?

The key to answering these questions is in the interaction between the abstraction mappings used to define h_F and h_C and the compression regime used by h_F^{comp}. We say that a compression regime is *compatible* with the refinement relationship between h_C and h_F if the only states that end up in the same compressed table entry are those which cannot be distinguished by the coarser heuristic h_C. Formally, this means that we require the following condition for all states s and s':

$$\begin{aligned} & comp(rank_F(\pi_F(s))) = comp(rank_F(\pi_F(s'))) \\ \text{iff} \quad & \pi_C(s) = \pi_C(s'). \end{aligned}$$

This essentially means that the compression function can be viewed as a *further abstraction* that, when composed with the fine-grained abstraction π_F, is equivalent to (distinguishes exactly the same states as) directly applying the coarse abstraction π_C.

When using arbitrary or random ranking or compression functions, this compatibility relationship is unlikely to hold because the rank of an abstract state and the resulting compressed table index do not capture any structural aspects of the abstraction. However, when using lexicographic ranking based on the representation of an abstract state as a tuple of values for state variables, the commonly used compression functions MOD and DIV (with suitable divisors) can be interpreted in terms of the abstract state structure: if the last state variable (the one considered least significant in the lexicographical ordering) can take on k values, then DIV compression by a factor of k amounts to projecting away the last state variable, and similarly if the first state variable can take on k values, then MOD compression to reduce by a factor of k (i.e., computing the rank modulo N/k) amounts to projecting away the first state variable.

For example, using lexicographical ranking in the 4-peg Towers of Hanoi domain, compression DIV 4 or MOD 4 amounts to projecting away one of the disks. Depending on the order of state variables, this will typically be the *largest* or *smallest* disk considered in the finer abstraction.

We are now ready to answer question Q1:

Theorem 1. *Let h_F and h_C be heuristics such that h_F is a refinement of h_C. Consider compressed heuristics with a compression regime that is compatible with h_F and h_C.*

Then

$$h_F^{comp}(s) \geq h_C(s)$$

for all states s.

Proof: Consider a state s with $h_F^{comp}(s) = k < \infty$. (For infinite heuristic values, there is nothing to prove, as $\infty \geq h_C(s)$ always holds.)

From the definition of min-compression, there must be a state $\overline{s}$ that is compressed to the same compressed table entry as s such that $h_F(\overline{s}) = h_F^{comp}(s) = k$. This can be any state in the same compressed table entry as s with minimum h_F value.

From $h_F(\overline{s}) = k$, the definition of abstraction heuristics implies that the abstract state space defined by π_F has a path of total cost k from $\pi_F(\overline{s})$ to some abstract goal state.

Because π_F is a refinement of π_C, all abstract paths in the abstract state space for π_F have corresponding abstract paths in the abstract state space for π_C (but not necessarily vice versa), which implies $h_C(\overline{s}) \leq k = h_F(\overline{s})$. Note that this path does not have to be a *shortest* path in the abstract space for π_C, hence equality does not have to hold.

Because we have a compatible compression regime, the only states that are mapped to the same compressed table entry as s are the ones that π_C cannot distinguish from s, and hence $h_C(s) = h_C(\overline{s})$.

We complete the proof by putting the pieces together: $h_F^{comp}(s) = h_F(\overline{s}) \geq h_C(\overline{s}) = h_C(s)$. □

We conclude that with compatible compression regimes, using a finer abstraction and then compressing it to the size of a coarser one is never worse in terms of heuristic accuracy than using the coarser abstraction directly.

Min-Compression and Variable Dependencies

Our previous result suggests that if we take care to use a compatible compression regime, compressed PDBs are a universal win over regular PDBs. However, we must be careful: while Theorem 1 shows that in such cases h_F^{comp} is never worse than h_C, it does not guarantee that it is actually better.

In fact, experiments in standard benchmarks show that in many cases h_F^{comp} and h_C turn out to be identical. For example, experimental equality of these heuristics has been observed for TopSpin domain with compatible compression regimes using lexicographical ranking and either DIV or MOD compression, whereas for 4-peg Towers of Hanoi, one of these regimes leads to huge improvements of h_F^{comp} over h_C whereas with the other regime, the two heuristics are identical. In this section, we attempt to shed some light on this phenomenon, answering questions Q2 and Q3.

The key to understanding this behavior lies in a notion of *dependency* between the state variables of the problem. We say that state variable u depends on state variable v if a value change of state variable u can depend on the current value of

[2] The notion of refinement of abstraction heuristics also applies to more general forms of abstractions than the projections underlying PDBs. Whenever abstraction α can be viewed as a composition of two functions $\alpha_1 \circ \alpha_2$, α_2 is a *refinement* of α. All results in this paper directly carry over to this more general setting.

state variable v in any way. More formally, consider a state transition $s_1 \xrightarrow{c} s_1'$ that changes the value of u, i.e., u takes on different values in s_1 and s_1'. We say that u *is independent of* v *in* s_1 if for all states s_2 that are identical to s_1 except for the value of v, there exists a transition $s_2 \xrightarrow{c} s_2'$ such that s_2' is identical to s_1' except (possibly) for the value of v. We say that u *depends on* v if there exists any state in which u is not independent of v.

While this definition of dependencies is semantic (i.e., requires checking all states in the state space), it is closely related to syntactic constructs in state-space representation formalisms such as SAS$^+$ and PSVN. Under a SAS$^+$ representation, u can only depend on v if there exists some action modifying u which has a precondition on v. This means that this notion of variable dependency is closely related to arcs in *causal graphs* commonly studied in the planning literature (e.g., Knoblock 1994; Chen and Giménez 2008). The main difference is that only precondition-effect relationships induce dependencies, while arcs in causal graphs are also induced by effect-effect relationships (i.e., two state variables being modified by the same action).

We can now prove the second main result:

Theorem 2. *Consider the patterns F and C with $F \supseteq C$ in an undirected state space (i.e., for all transitions $s \xrightarrow{c} s'$, there exists a transition $s' \xrightarrow{c} s$). Let h_F^{comp} be a compressed PDB heuristic with a compression regime that is compatible with the refinement relation between F and C. If no variable in C depends on any variable in $F \setminus C$, then*

$$h_F^{comp}(s) = h_C(s)$$

for all states s.

Proof: Let F, C and h_F^{comp} satisfy the prerequisites of the theorem, and let s be any state. Theorem 1 shows $h_F^{comp}(s) \geq h_C(s)$, so it remains to show $h_F^{comp}(s) \leq h_C(s)$. For this it suffices to show that whenever there exists a sequence of abstract transitions from $\pi_C(s)$ to some goal state $\pi_C(s_{\text{goal}})$ under π_C, there exists a state $\overline{s}$ with $\pi_C(\overline{s}) = \pi_C(s)$ from which there exists a corresponding sequence of abstract transitions from $\pi_F(\overline{s})$ to $\pi_F(s_{\text{goal}})$ under π_F. (Note that min-compression puts s and $\overline{s}$ into the same compressed entry.)

Let $s_C^0 \to \cdots \to s_C^n$ be a sequence of abstract transitions under π_C with $s_C^0 = \pi_C(s)$, $s_C^i \neq s_C^{i+1}$ for all $0 \leq i < n$, and $s_C^n = \pi_C(s_{\text{goal}})$. Because the state space is undirected, the *reverse* of this path also exists and has the same cost.

From this reverse path under π_C we incrementally construct a corresponding reverse path under the finer abstraction π_F, and by reversing this path again, we obtain the desired path from $\pi_F(\overline{s})$ to $\pi_F(s_{\text{goal}})$ under π_F.

The reverse path begins with $s_F^n = \pi_F(s_{\text{goal}})$, which is an abstract goal state under π_F and is a refinement of s_C^n. For all $0 < i \leq n$, after constructing s_F^i, we can construct s_F^{i-1} as follows, using the inductive argument that s_F^i is a refinement of s_C^i: because the transition $s_C^{i-1} \to s_C^i$ exists in π_C, there must be a transition $s \to s'$ in the original state space with $\pi_C(s) = s_C^{i-1}$ and $\pi_C(s') = s_C^i$. Because of undirectedness, there is a transition $s' \to s$ of the same cost. Because $s_C^{i-1} \neq s_C^i$, s' and s differ in at least one variable from C. Because all variables in C are independent of all variables in $F \setminus C$, we can change all variables in $F \setminus C$ arbitrarily to obtain a new transition $\overline{s'} \to \overline{s}$ of the same cost where $\pi_F(\overline{s'}) = s_F^i$ and $\pi_C(\overline{s}) = s_C^{i-1}$. Set $s_F^{i-1} = \pi_F(\overline{s})$ to complete the construction. □

It is instructive to compare Theorem 2 to an earlier result on variable dependencies and the quality of PDB heuristics from the classical planning literature, first proved by Haslum et al. (2007) and later refined by Pommerening, Röger, and Helmert (2013). Pommerening et al. show that the only *interesting* patterns P in SAS$^+$ planning tasks are ones where (1) the subgraph CG_P of the causal graph induced by P (including both precondition-effect and effect-effect arcs) is weakly connected, and (2) there is a directed path in CG_P using only precondition-effect arcs from each variable $v \in P$ to some variable mentioned in the goal condition. If condition (1) is violated, the pattern P can be split into two smaller patterns P_1 and P_2 with $h_P = h_{P_1} + h_{P_2}$, and if condition (2) is violated, there exists a smaller pattern P' with $h_{P'} = h_P$.

This "interesting pattern criterion" implies that in SAS$^+$ tasks, Theorem 2 remains true if we replace the condition that the state space is undirected by the condition that $F \setminus C$ contains no variables mentioned in the goal condition.

Comparing the *interesting pattern* criterion to Theorem 2, we see that neither result entails the other. For example, on the one hand, Theorem 2 is limited to undirected state spaces, while the interesting pattern criterion has no such restriction. On the other hand, the following section includes several examples of patterns $F \subseteq C$ where Theorem 2 shows that h_F^{comp} cannot improve over h_C while the interesting pattern criterion provides no useful information.

Discussion

We conclude by discussing the implications of our results for some classical benchmarks. On the positive side, Theorem 1 shows that, when care is taken to use a compatible compression regime, min-compressed PDBs are always at least as strong as regular PDBs. On the negative side, Theorem 2 shows that improvements can only be obtained if the finer abstraction on which the compressed PDB is based captures a dependency of the coarser abstraction. (Of course this can also be seen as a positive result to guide the appropriate choice of refinement to use for a compressed PDB.)

In the quotation from our introduction, Sturtevant, Felner, and Helmert (2014) observed that compressed PDBs have been highly successful for the 4-peg Towers of Hanoi, unsuccessful for TopSpin, and modestly successful for sliding-tile puzzles. We can explain all these results in terms of the variable dependencies present in these domains.

In the Towers of Hanoi, the state variable representing the location of disk u depends on the variable representing the location of disk v iff v is smaller than u. Smaller disks impede the movement of larger disks and hence introduce a dependency: an action moving a given disk is only applicable if no smaller disks are present on the source or target peg. However the converse is not true, as larger disks never impede the movement of smaller disks. Hence, to im-

prove a PDB heuristic by compression we must add *small* disks to the pattern and compress them away: intuitively, this captures some additional movements of small disks that are necessary to make space for large-disk moves. Adding *large* disks and compressing them away cannot improve the heuristic.

This also explains the different results in Towers of Hanoi with MOD vs. DIV compression: with lexicographical ranking and a typical state representation where the i-th variable represents the disk of size i, DIV compression coincides with projecting away the largest disks, while MOD compression coincides with projecting away the smallest disks.

In the TopSpin puzzle, using a state representation where each state variable encodes the location of one token, there are no dependencies at all. All actions are precondition-free, and the location of tile u after applying a given action is not affected by the location of another tile v. The same holds true for *all* pure permutation problems (i.e., problems whose state spaces are Caley graphs), such as Rubik's Cube or the pancake problem. This explains why min-compression does not lead to improvements in pure permutation puzzles.

In the sliding tile puzzles, all state variables representing locations of tiles depend on the variable representing the location of the blank, because the location of the blank affects the applicability of actions changing the location of the tiles. These are the only dependencies: variables representing positions of different tiles do not depend on each other.

This explains the success of the well-known technique of *compressing the blank* in the 15-puzzle: because all variables depend on the blank, it is beneficial to refine an abstraction without the blank to one that includes it and then compress it away. Because these are the only dependencies in the puzzle, adding a non-blank state variable and compressing it away would be pointless.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This work was supported by the Swiss National Science Foundation (SNSF) as part of the project "Reasoning about Plans and Heuristics for Planning and Combinatorial Search" (RAPAHPACS), by the National Science Foundation (NSF) under Grant No. 1551406 and by Israel Science Foundation (ISF) grant #417/13.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS^+ planning. *Computational Intelligence* 11(4):625–655.

Ball, M., and Holte, R. C. 2008. The compression power of symbolic pattern databases. In *Proc. ICAPS 2008*, 2–11.

Bonet, B. 2008. Efficient algorithms to rank and unrank permutations in lexicographic order. In *AAAI 2008 Workshop on Search in Artificial Intelligence and Robotics*, 18–23.

Breyer, T. M., and Korf, R. E. 2010. 1.6-bit pattern databases. In *Proc. AAAI 2010*, 39–44.

Chen, H., and Giménez, O. 2008. Causal graphs and structurally restricted planning. In *Proc. ICAPS 2008*, 36–43.

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. In *Proceedings of the Eleventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence (CSCSI-96)*, volume 1081 of *LNAI*, 402–416. Springer-Verlag.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. ECP 2001*, 84–90.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *Proc. AIPS 2002*, 274–283.

Felner, A.; Korf, R. E.; Meshulam, R.; and Holte, R. 2007. Compressed pattern databases. *JAIR* 30:213–247.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Proc. AAAI 2007*, 1007–1012.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proc. AAAI 2005*, 1163–1168.

Hernádvölgyi, I. T., and Holte, R. C. 1999. PSVN: A vector representation for production systems. Technical Report TR-99-04, School of Information Technology and Engineering, University of Ottawa.

Knoblock, C. A. 1994. Automatically generating abstractions for planning. *AIJ* 68(2):243–302.

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

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

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pommerening, F.; Röger, G.; and Helmert, M. 2013. Getting the most out of pattern databases for classical planning. In *Proc. IJCAI 2013*, 2357–2364.

Sadeqi, M., and Hamilton, H. J. 2016. Efficient representation of pattern databases using acyclic random hypergraphs. In *Proc. ICAPS 2016*, 258–266.

Sturtevant, N. R.; Felner, A.; and Helmert, M. 2014. Exploiting the Rubik's cube 12-edge PDB by combining partial pattern databases and Bloom filters. In *Proc. SoCS 2014*, 175–183.

Sturtevant, N. R.; Felner, A.; and Helmert, M. 2017. Value compression of pattern databases. In *Proc. AAAI 2017*, 912–918.