Planning with Pattern Databases

Stefan Edelkamp
Institut für Informatik
Albert-Ludwigs-Universität
Georges-Köhler-Allee, Gebäude 51
D-79110 Freiburg
eMail: edelkamp@informatik.uni-freiburg.de

Abstract

Heuristic search planning effectively finds solutions for large planning problems, but since the estimates are either not admissible or too weak, optimal solutions are found in rare cases only. In contrast, heuristic pattern databases are known to significantly improve lower bound estimates for optimally solving challenging single-agent problems like the 24-Puzzle or Rubik’s Cube.

This paper studies the effect of pattern databases in the context of deterministic planning. Given a fixed state description based on instantiated predicates, we provide a general abstraction scheme to automatically create admissible domain-independent memory-based heuristics for planning problems, where abstractions are found in factorizing the planning space. We evaluate the impact of pattern database heuristics in A* and hill climbing algorithms for a collection of benchmark domains.

Introduction

General propositional planning is PSPACE complete (Bylander 1994), but when tackling specific benchmark planning instances, improving the solution quality usually reveals the intrinsic hardness of the problems. For example, plan existence of Logistics and Blocks World problem instances is polynomial, but minimizing the solution lengths for these planning problems is NP-hard (Helmert 2001). Therefore, we propose a heuristic search planner that finds optimal plans. If challenging planning problems call for exponential resources, the planner can be tuned to approximate optimal plan length.

Optimal Planning Approaches

Graphplan (Blum and Furst 1995) constructs a layered planning graph containing two types of nodes, action nodes and proposition nodes. In each layer the preconditions of all operators are matched, such that Graphplan considers instantiated actions at specific points in time. Graphplan generates partially ordered plans to exhibit concurrent actions and alternates between two phases: graph extension to increase the search depth and solution extraction to terminate the planning process. Graphplan finds optimal parallel plans, but does not approximate plan lengths; it simply exhausts the given resources.

Another optimal planning approach is symbolic exploration simulating a breadth-first search according to the binary encoding of planning states. The operators unfold the initial state and an efficient theorem prover then searches for a satisfying truth assignment. A Boolean formula \( f_t \) describes the set of states reachable in \( t \) steps. If \( f_t \) contains a goal state, the problem is solvable with the minimal \( t \) as the optimal plan length.

Two approaches have been proposed. Satplan (Kautz and Selman 1996) encodes the planning problem with a standard representation of Boolean formulae as a conjunct of clauses. The alternative in the planner Mips (Edelkamp and Helmert 2001) is to apply binary decision diagrams (BDDs); a data structure providing a unique representation for Boolean functions (Bryant 1985). The BDD planning approach is in fact reachability analysis in model checking (Clarke, Grumberg, and Peled 2000). It applies to both deterministic and non-deterministic planning and the generated plans are optimal in the number of sequential execution steps. Usually, symbolic approaches cannot approximate. Though promising, recent results with symbolic best-first search (Edelkamp 2001) are still not as good as the ones obtained with explicit heuristic search engines; our next topic.

Heuristic Search Planning

Directed exploration is currently the most effective approach in classical AI-planning: four of five honored planning systems in the general planning track of the AIPS-2000 competition at least partially incorporate heuristic search. However, in traversing the huge state spaces of all combinations of grounded predicates, all planners rely on inadmissible estimates. The currently fastest deterministic planner, FF (Hoffmann 2000), solves a relaxed planning problem at each each encountered state. The obtained relaxed plan length is not a certified lower bound for the optimal plan length, but a good approximation. Pruning rules in FF like helpful action and goal ordering cuts additionally help to avoid local optima in the underlying hill-climbing algorithm and to quickly escape encountered plateaus. Completeness in undirected problem graphs is achieved by omitting prun-
The planning space representation involves creating a pattern database framework that allows very fast node expansions. After studying the state-to-goal distances in tractable abstractions of the planning space, the retrieval of the lower bound estimators with perfect hash functions is almost constant time operation, allowing very fast node expansions. After studying the pattern database framework, experiments were conducted with improvements to max-pair that were discouraging. We used a minimum matching algorithm on a graph with nodes corresponding to atoms and with edges weighted with atom-pair distances.

This paper proposes a pre-computed admissible heuristic that applies a different strategy and projects the problem into a simpler one by excluding a set of atoms. The heuristic estimates are stored in a large look-up table, the pattern database. To build the databases, we exhaustively search all state-to-goal distances in tractable abstractions of the planning state space. The retrieval of the lower bound estimators with perfect hash functions is almost constant time operation, allowing very fast node expansions. After studying the pattern database framework, experiments were conducted with a sizable number of planning problems and draw concluding remarks.

Planning Space Representation

For the sake of simplicity we concentrate on the STRIPS formalism (Fikes and Nilsson 1971), in which each operator is defined by a precondition list \( P \), an add list \( A \), and a delete list \( D \), but the presented approach can be extended to various problem description languages which can be parsed into a fixed state encoding. We refer to state descriptions and lists as sets of atoms. This is not a limitation since all state-of-the-art planners perform grounding; either prior to the search or on the fly.

**Definition 1** Let \( F \) be the set of atoms and \( O \) be a set of grounded STRIPS operators. The result \( S' \) of an operator \( o = (P, A, D) \in O \) applied to a state \( S \subseteq F \) is defined as \( S' = (S \setminus D) \cup A \) in case \( P \subseteq S \). All states span the planning space\(^1 \) \( P \).

We exemplify our considerations in the Blocks World domain of AIPS-2000, specified with the four operators pick-up, put-down, stack, and unstack. For example, the grounded operator \((\text{pick-up } a)\) is defined as

\[
P = \{(\text{clear } a), (\text{ontable } a), (\text{handempty})\},
\]

\[
A = \{(\text{holding } a)\}, \text{ and}
\]

\[
D = \{(\text{ontable } a), (\text{clear } a), (\text{handempty})\}.
\]

\(^1\)The planning space \( P \) is in fact smaller than the set of subsets of atoms, but includes the set of states reachable from the initial state.

- \( G_1 = \{(\text{on } c \ a), (\text{on } d \ a), (\text{on } b \ a), (\text{clear } a), (\text{holding } a)\} \),
- \( G_2 = \{(\text{on } c \ a), (\text{on } d \ c), (\text{on } b \ c), (\text{clear } c), (\text{holding } c)\} \),
- \( G_3 = \{(\text{on } d \ d), (\text{on } c \ d), (\text{clear } d), (\text{holding } d)\} \),
- \( G_4 = \{(\text{on } b \ b), (\text{on } c \ b), (\text{clear } b), (\text{holding } b)\} \),
- \( G_5 = \{(\text{ontable } a), \text{true}\} \),
- \( G_6 = \{(\text{ontable } c), \text{true}\} \),
- \( G_7 = \{(\text{ontable } d), \text{true}\} \),
- \( G_8 = \{(\text{ontable } b), \text{true}\} \), and
- \( G_9 = \{(\text{handempty}), \text{true}\} \).

Table 1: The partition into atom groups for the example problem.

The goal of the instance 4-1 is defined by \{\{(\text{on } d \ c), (\text{on } c \ a), (\text{on } a \ b)\}\} and the initial state is given by \{\{(\text{clear } b), (\text{ontable } d), (\text{on } b \ c), (\text{on } c \ a), (\text{on } a \ d)\}\}. The first step to construct a pattern database is a domain analysis prior to the search. The output are mutually exclusive atom groups, which for each reachable state exactly one will be true. In general this construction is not unique such that we minimize the state description length over all possible partitionings as proposed for the Mips planning system (Edelkamp and Helmert 1999).

Table 1 shows the inferred 9 groups for the example problem, where \text{true} refers to the situation, where none of the other atoms is present in a given state.

Pattern Databases

A recent trend in single-agent search is to calculate the estimate with heuristic pattern databases (PDBs) (Culberson and Schaeffer 1998). The idea is to generate heuristics that are defined by distances in space abstractions. PDB heuristics are consistent\(^2\) and have been effectively applied to solve challenging \((n^2 - 1)\)-Puzzles (Korf and Felner 2001) and Rubik’s Cube (Korf 1997). In the \((n^2 - 1)\)-Puzzle a pattern is a collection of tiles and in Rubik’s Cube either a set of edge-cubes or a set of corner-cubes is selected. For all of these problems the construction of the PDB has been implemented problem-dependently by a manual input of the abstraction and associated perfect hash functions. In contrast, we apply the concept of PDBs to general problem-independent planning and construct pattern databases fully automatically.

State Abstractions

State space abstractions in the context of PDBs are concisely introduced in (Hernadovgyi and Holte 2000): A state is a vector of fixed length and operators are conveniently expressed by label sets, e.g., an operator mapping \( \langle A, B, \_ \rangle \)

\[^2\]Consistent heuristic estimates fulfill \( h(v) - h(u) + w(u, v) \geq 0 \) for each edge \((u, v)\) in the underlying, possibly weighted, problem graph. They yield monotone merits \( f(u) = g(u) + h(u) \) on generating paths with weight \( g(u) \). Admissible heuristics are lower bound estimates which underestimate the goal distance for each state. Consistent estimates are admissible.
to \( (B, A, \ldots) \) corresponds to a transposition of the first two elements for any state vector of length three. The state space is the transitive closure of the seed state \( S_0 \) and the operators \( O \). A domain abstraction is defined as a mapping \( \phi \) from one label set \( L \) to another label set \( K \) with \(|K| < |L|\) such that states and operators can be simplified by reducing the underlying label set. A state space abstraction of the search problem \( (S_0, O, L) \) is denoted as \( \langle \phi(S_0), \phi(O), K \rangle \). In particular, the abstraction mapping is non-injective such that the abstract space (which is the image of the original state space) is therefore much smaller than the original space. The framework in (Hernadługy and Holte 2000) only applies to certain kinds of permutation groups, where in our case the abstract space is obtained in a more general way, since abstraction is achieved by projecting the state representation.

**Definition 2** Let \( F \) be the set of atoms. A planning space abstraction \( \phi \) is a mapping from \( F \) to \( F \cup \{true\} \) such that in each group \( G \) either for all \( f \in G : \phi(f) = f \) or for all \( f \in G : \phi(f) = true \).

Since planning states are interpreted as conjuncts of atoms, \( \phi \) can be extended to map each planning state of the original space \( P \) to one in the abstract space \( A \). In the example problem instance we apply two planning space abstractions \( \phi_{odd} \) and \( \phi_{even} \). The mapping \( \phi_{odd} \) assigns all atoms in groups of odd index to the trivial value \( true \) and, analogously, \( \phi_{even} \) maps all atoms in groups with even index value to \( true \). All groups not containing a atoms in the goal state are also mapped to \( true \). In the example, the goal is partitioned into \( \phi_{even}(G) = \{ (on\ c\ a) \} \) and \( \phi_{odd}(G) = \{ (on\ a\ b), (on\ d\ c) \} \), since the groups \( G_4 \) to \( G_9 \) are not present in the goal description.

Abstract operators are defined by intersecting their pre-condition, add and delete lists with the set of non-reduced atoms in the abstraction. This accelerates the construction of the pattern table, since several operators simplify.

**Definition 3** Let \( \phi \) be a planning space abstraction and \( \delta_\phi \) be the shortest path between abstract states and/or sets of abstract states. Furthermore, let \( G \subseteq F \) be the goal condition in \( P \). A planning pattern database (PDB) according to \( \phi \) is a set of pairs, with the first component being an abstract planning state \( S \) and the second component being \( \delta_\phi(S, \phi(G)) \), i.e.,

\[
PDB(\phi) = \{(S, \delta_\phi(S, \phi(G))) \mid S \in A\}.
\]

PDB(\( \phi \)) is calculated in a breadth-first traversal starting from the set of goals in applying the inverse of the operators. Two facts about PDBs are important. When reducing the state description length \( n \) to \( \alpha n \) with \( 0 < \alpha < 1 \) the state space and the search tree shrinks exponentially, e.g. \( 2^n \) bit vectors correspond to an abstract space of \( 2^{\alpha n} \) elements. The second observation is that once the pattern database is calculated, accessing the heuristic estimate is fast by a simple table look-up.

Moreover, PDBs can be reused for the case of different initial states. PDB(\( \phi_{even} \)) and PDB(\( \phi_{odd} \)) according to the abstractions \( \phi_{even} \) and \( \phi_{odd} \) of our example problem are depicted in Table 2. Note that there are only three atoms present in the goal state such that one of the pattern databases only contains patterns of length one. Abstraction \( \phi_{even} \) corresponds to \( G_1 \) and \( \phi_{odd} \) corresponds to the union of \( G_2 \) and \( G_4 \).

**Disjoint Pattern Databases**

Disjoint pattern databases add estimates according to different abstractions such that the accumulated estimates still provide a lower bound heuristic.

**Definition 4** Two pattern databases PDB(\( \phi_1 \)) and PDB(\( \phi_2 \)) are disjoint, if the sum of respective heuristic estimates always underestimates the overall plan length, i.e., for all \( S \in P \) we have \( \delta_{\phi_1}(\phi_1(S), \phi_1(G)) + \delta_{\phi_2}(\phi_2(S), \phi_2(G)) \leq \delta(S, G) \).

To include groups, which are not present in the goal state, into PDB calculations, we generate all possible instances for the atom set. In fact, this is the approach that is applied in our implementation.

PDBs are not always disjoint. Suppose that a goal contains two atoms \( p_1 \) and \( p_2 \), which are in groups 1 and 2, respectively, and that an operator \( o \) makes both \( p_1 \) and \( p_2 \) true. Then, the distance under abstraction \( \phi_1 \) is 1 (because the abstraction of \( o \) will make \( p_2 \) in group 2 true in one step) and the distance under \( \phi_2 \) is also 1 (for the same reason). But the distance in the original search space is also only 1.

**Definition 5** An independent abstraction set \( I \) is a set of group indices such that no operator affects both atoms in groups in \( I \) and atoms in groups that are not in \( I \). The according abstraction \( \phi_I \) that maps all atom groups not in \( I \) to true is called an independent abstraction.

**Theorem 1** A partition of the groups into independent abstractions sets yields disjoint pattern databases.

**Proof** Each operator changes information only within groups of a given partition and an operator of the abstract planning space contributes one to the overall estimate only

<table>
<thead>
<tr>
<th>( \phi_{even} )</th>
<th>( \phi_{odd} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>((clear a),1)</td>
<td>((holding a),2)</td>
</tr>
<tr>
<td>((on b a),2)</td>
<td>((on a c),2)</td>
</tr>
<tr>
<td>((on d a),2)</td>
<td></td>
</tr>
<tr>
<td>((on c b) (clear b),1)</td>
<td>((on a b) (clear c),1)</td>
</tr>
<tr>
<td>((on d c) (clear b),2)</td>
<td>((clear c) (clear b),2)</td>
</tr>
<tr>
<td>((on d c) (holding b),2)</td>
<td>((on a c) (holding c),2)</td>
</tr>
<tr>
<td>((on a c) (on b a) ,2)</td>
<td>((clear c) (holding b),3)</td>
</tr>
<tr>
<td>((clear b) (holding c),3)</td>
<td>((on c a) (clear c),3)</td>
</tr>
<tr>
<td>((on b c) (clear c),4)</td>
<td>((on b c) (holding b),4)</td>
</tr>
<tr>
<td>((on a c) (clear b),4)</td>
<td>((on d b) (holding c),4)</td>
</tr>
<tr>
<td>((on a c) (on b d) ,4)</td>
<td>((on a c) (clear b),4)</td>
</tr>
<tr>
<td>((on a c) (on b c),5)</td>
<td>((on b c) (holding c),5)</td>
</tr>
<tr>
<td>((on b c) (clear c),5)</td>
<td>((on a c) (on b c),5)</td>
</tr>
<tr>
<td>((on c b) (clear c),5)</td>
<td>((on c b) (on d c),5)</td>
</tr>
<tr>
<td></td>
<td>((on b c) (on d c),5)</td>
</tr>
</tbody>
</table>

Table 2: Pattern databases PDB(\( \phi_{even} \)) and PDB(\( \phi_{odd} \)) for the example problem.
if it changes atoms in available atom groups. Therefore, by adding the plan lengths of different abstract spaces, each operator on each path is counted at most once.

For some domains like Logistics operators act locally according to any partition into groups, so that the precondition of Theorem 1 is trivially fulfilled.

**Perfect Hashing**

PDBs are implemented as a (perfect) hash table with a table look-up in time linear to the abstract state description length.

According to the partition into groups a perfect hashing function is defined as follows. Let \( G_{i_1}, G_{i_2}, \ldots, G_{i_k} \) be the selected groups in the current abstraction and \( \text{offset}(k) \) be defined as \( \text{offset}(k) = \prod_{l=1}^{k} |G_{i_l}| - 1 \) with \( |G_{i_0}| = 1 \). Furthermore, let \( \text{group}(f) \) and \( \text{position}(f) \) be the group index and the position in the group of atoms \( f \), respectively. Then the perfect hash value \( \text{hash}(S) \) of state \( S \) is

\[
\text{hash}(S) = \sum_{f \in S} \text{position}(f) \cdot \text{offset}(\text{group}(f)).
\]

Since perfect hashing uniquely determines an address for the state \( S \), \( S \) can be reconstructed given \( \text{hash}(S) \) by extracting all corresponding group and position information that define the atoms in \( S \). Therefore, we establish a good compression ratio, since each state in the queue for the breadth-first search traversal only consumes one integer. The breadth-first-search queue is only needed for construction and the resulting PDB is a plain integer array of size \( \text{offset}(k + 1) \) encoding the distance values for the corresponding states, initialized with \( \infty \) for patterns that are not encountered. Some states are not generated, since they are not reachable, but the above scheme is more time and space efficient than ordinary hashing storing the uncompressed state representation. Since small integer elements consume only a few bytes, on current machines we may generate PDBs of a hundred million entries and more.

**Clustering**

In the simple example planning problem the combined sizes of groups and the total size of the generated pattern databases \( \text{PDB}(\phi_{\text{even}}) \) and \( \text{PDB}(\phi_{\text{odd}}) \) differ considerably. Since we perform a complete exploration in the generation process, in larger examples an automatic way to find a suitable balanced partition according to given memory limitations is required. Instead of a bound on the total size of all PDBs together, we globally limit the size of each pattern database, which is in fact the number of expected states. The restriction is not crucial, since the number of different pattern databases is small in practice. The threshold is the parameter to tune the quality of the estimate. Obviously, large threshold values yield optimal estimates in small problem spaces.

We are confronted with a Bin-Packing variant: Given the sizes of groups, the task is to find the minimal number of pattern databases such that the sizes do not exceed a certain threshold value. Notice that the group sizes are multiplied in order to estimate the search space size. However, the corresponding encoding lengths are additive. Bin-Packing is NP-hard in general, but good approximation algorithms exist. In our experiments we applied the best-fit strategy.

**Results**

All experimental results were produced on a Linux PC, Pentium III CPU with 800 MHz and 512 MByte. We chose the most efficient domain-independent planners as competitors. In Logistics, the program FF is chosen for comparison and in Blocks World, the pattern database approach is compared to the optimal planner Mips.

**Logistics**

We applied PDBs to Logistics and solved the entire problem set of AIPS-2000. The largest problem instance includes 14 trucks located in one of three locations of the 14 cities. Together with four airplanes the resulting state space has a size of about \( 3^{14} \cdot 14^4 \cdot 60^{12} \approx 8.84223 \cdot 10^{85} \) states. All competing planners that have solved the entire benchmark problem suite are (enforced) hill-climbers with a variant of the FF heuristic and the achieved results have about the same characteristics (Hoffmann 2001). Therefore, in the Figures 1 and 2 we compare the PDB approach with the FF-heuristic. In the enforced hill climbing algorithm we allow both planners to apply branching cuts, while in A* we scale the influence of the heuristic with a factor of two.

The effects of scaling are well-known (Pearl 1985): weightening A* possibly results in non-optimal solution, but the search tends to succeed much faster. In the AIPS-2000 competition, the scaling factor 2 has enhanced the influence of the max-pair heuristic in the planner HSP. However, even with this improvement it solves only a few problems of this benchmark suite.

The characteristics of the PDB and FF heuristics in Figure 1 are quite different. The number of expanded nodes
is usually larger for the former one but the run time is much shorter. A* search with PDBs outperforms FF with hill climbing and branching cuts. The savings are about two orders of magnitude.

In the example set the average time for a node expansion in PDB-based planning is smaller by about two orders of magnitude compared to FF.

On the other hand, in larger problem instances enforced hill climbing according to the PDB heuristic generates too many nodes to be kept in main memory. In a few seconds the entire memory resources were exhausted. This suggests applying memory limited search algorithm like thresholding in IDA* and alternative hashing strategies to detect move transpositions in large search depths. We summarize that hill climbing is better suited to the FF heuristic while weighted A* seems to perform better with PDBs. The plan qualities are about the same as Figure 3 depicts.

**Blocks World**

Finding approximate plans in Blocks World is easy, 2-approximations run in linear time (Slaney and Thiebaux 2001). Moreover, different domain-dependent cuts drastically reduce the search space. Hence, TALPlanner (Kvarnstrom, Doherty, and Haslum 2000) with hand-coded cuts and FF with hill climbing, helpful action and goal ordering cuts find good approximate plans to problems with fifty Blocks and more. FF using enforced hill climbing without cuts is misguided by its heuristic, backtracks and tends to get lost in local optima far away from the goal. We concentrate on optimal plans for this domain.

Since any \( n \)-Tower configuration is reachable from the initial state, the state space grows exponentially in \( n \), and indeed, optimizing Blocks World is NP-hard. Graphplan is bounded to about 9 blocks and no optimal heuristic search engine achieves a better performance, e.g. HSP with max-pair is bounded to about 6-7 blocks. Model checking engines like BDD exploration in Mips and iterative Boolean satisfiability checks in Satplan are best in this domain and optimally solve problems with up to 12-13 blocks. Tables 4 depict that PDBs are competitive.

Moreover, better scaling in time seems to favor PDB exploration. However, in both approaches space consumption is more crucial than time. In the bidirectional symbolic breadth-first search engine of Mips the BDD sizes grow very rapidly and large pattern databases with millions of entries still lead to millions of node expansions. When searching for plans to 13-block benchmark problems memory resources in both planning approaches become exhausted.

**Other Domains**

Gripper (AIPS-1998) spans an exponentially large but well-structured search space such that greedy search engines find optimal plans. On the other hand, Gripper is known to be hard for Graphplan. Both FF with hill-climbing and cuts and PDB with weighted A* find optimal solutions in less than a second.

Like Logistics, the NP-hard (Helmert 2001) Mystery domain (AIPS-1998) is a transportation domain on a road map. Trucks are moving around this map and packages are being carried by the mobiles. Additionally, various capacity and fuel constraints have to be satisfied. Mystery is particularly difficult for heuristic search planning, since some of the instances contain a very high portion of undetected dead-ends (Hoffmann 2001). In contrast to the most effective heuristic search planner GRT (Refanidis and Vlahavas 2000), the PDB planning algorithm does not yet incorporate manual reformulation based on explicit representation of resources. However, experiments show that PDB search is competitive: problems 1-3, 9, 11, 17, 19, 25-30 were optimally solved in less then 10 seconds, while problem 15 and 20 required about 5 and 2 minutes, respectively. At least
To planning problems by devising an automatic abstraction scheme to construct pre-compiled pattern databases.

Our experiments show that pattern database heuristics are very good admissible estimators. Once calculated, our new estimate will be available in constant time since the estimate of a state is simply retrieved in a perfect hash table by projecting the state encoding. We will investigate different pruning techniques to reduce the large branching factors in planning. There are some known general pruning techniques such as FSM pruning (Taylor and Korf 1993), Relevance Cuts and Pattern Searches (Junghans 1999) that should be addressed in the near future.

Although PDB heuristics are admissible and calculated beforehand, their quality can compete with the inadmissible FF-heuristic that solves a relaxed planning problem for every expanded state. The estimates are available in a simple table look-up, and, in contrast to the FF-heuristic, A* finds optimal solutions. Weighting the estimate helps to cope with difficult instances for approximate solutions. Moreover, PDB heuristics in A* can handle directed problem spaces and prove unsolvability results.

One further important advantage of PDB heuristics is the possibility of a symbolic implementation. Due to the representational expressiveness of BDDs, a breadth-first search (BFS) construction can be completed with respect to larger parts of the planning space for a better quality of the estimate. The exploration yields a relation \( H(\text{estimate}, \text{state}) \) represented with a set of Boolean variables encoding the BFS-level and a set of variables encoding the state. Algorithm BDDA*, a symbolic version of A*, integrates the symbolic representation of the estimate (Edelkamp 2001). Since PDBs lead to consistent heuristics the number of iterations in the BDDA* algorithms is bounded by the square of the solution length. Moreover, symbolic PDBs can also be applied to explicit search. The heuristic estimate for a state can be determined in time linear to the encoding length.

**Acknowledgments** We thank J. Hoffmann for the Sokoban problem generator, M. Helmer for eliminating typos, the anonymous referees for helpful comments, and P. Haslum for fruitful discussions on this research topic.

**References**


