Symbolic Pattern Databases in Heuristic Search Planning

Stefan Edelkamp

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

Abstract

This paper invents symbolic pattern databases (SPDB) to combine two influencing aspects for recent progress in domain-independent action planning, namely heuristic search and model checking. SPDBs are off-line computed dictionaries, generated in symbolic backward traversals of automatically inferred planning space abstractions.

The entries of SPDBs serve as heuristic estimates to accelerate explicit and symbolic, approximate and optimal heuristic search planners. Selected experiments highlight that the symbolic representation yields much larger and more accurate pattern databases than the ones generated with explicit methods.

Introduction

Heuristic search is one of the most effective search techniques to cope with very large problem spaces. The guidance for search algorithms like A* (Hart, Nilsson, & Raphael 1968) and IDA* (Korf 1985) are estimators that approximate the remaining distance to the goal.

The additional information focuses the search into the direction of the goal and its quality mainly influences the number of states to be expanded; the better the estimate, the larger the reduction in search efforts.

Planning problems implicitly generate weighted problem graphs by applying operator sequences to their seed states. By changing operator costs, A* can be casted as a variant of Dijkstra's single-source shortest path algorithm: the new costs of the operators are set to the old ones minus the heuristic value of the expanded state, plus the estimate of the successor state (Edelkamp & Schrödl 2000). Admissible heuristics are lower bound problem relaxations, yield optimal solutions, but may introduce negative weights calling for re-openings of already expanded states (Pearl 1985).

Pattern databases (PDBs) are dictionaries of heuristic values that have been originally applied to the Fifteen Puzzle (Culberson & Schaeffer 1998). In this context, PDBs are generalizations of the Manhattan distance heuristic, that corresponds to subproblem solutions of moving each tile onto its goal position. The PDB representation is a selection of look-up tables memorizing the goal distances of each tile at any board location. Since the subproblems are independent (only one tile can move at a time), the minimum numbers of moves to solve the individual puzzles can be added; still providing an admissible heuristic. Refined PDBs take not only one but a selection of interacting tiles (the pattern) into account. A large hash table stores their combined goal distances on a simplified board with all other tiles removed. PDBs are generated in complete backward explorations, starting from the set of abstract goals.

The PDB approach has been extended to find first optimal solutions to random Rubik's Cube problems (Korf 1985), where a pattern corresponds to a selection of side or corner cubies. Independence of PDBs has been exploited to solve the 24-Puzzle (Korf & Felner 2002). In all cases the abstractions for PDB construction were hand-tailored and domain dependent. The effectiveness of PDBs in form of a space-time trade-off reveals that PDBs size is inversely correlated to the resulting search effort (Holte & Hernadvölgyi 1999).

Steps towards the automated creation of PDB heuristics base on local search in the space of PDBs (Hernadvölgyi 2000) and change the abstraction level according to the predicted search efforts. However, the approach is currently limited to moderate state-space sizes, or restricted to easier exploration tasks like the computation of macro operators.

Explicit PDB heuristics that have been proposed for domain-independent action planning (Edelkamp 2001b) share similarities with PDBs in sliding-tile puzzles and challenge even on-line computed estimates like the FF-heuristic (Hoffmann 2002). The rough idea is to interpret the set propositional atoms as tiles, so that a planning pattern is a selection of them. The approach first infers groups of mutually exclusive facts. In every reachable state exactly one of the atoms in each group is true. The group information is exploited to derive planning abstractions and to infer perfect hash functions for pattern storage. Automated clustering partitions the state space into a set of abstractions with state spaces that fit into main memory. Planning PDBs are not always independent, but suitable partitions into groups, where all operators affect only atoms in the specified set, always yield independent PDBs.

In this paper we propose symbolic pattern databases (SPDBs) instead of explicit ones. A SPDB is Boolean func-

tion H of tuples (v, S). For a given planning state S with value v, formula H evaluates to *true* if and only if the estimate of S equals v. Through an efficient representation of Boolean functions, larger PDBs and more accurate estimates can be inferred and utilized.

The structure of the paper is as follows. First we give a concise introduction to PDBs together with proofs of some important properties. Then we turn the to symbolic representation of planning states and operators. Next we introduce symbolic backward exploration to generate SPDBs and integrate this representation of the heuristic estimate into directed explicit and symbolic search search engines. This algorithmic treatment is followed by a discussion the influence of SPDBs to search tree growth and exploration efforts. We evaluate the impact of the algorithms, taking Blocks World as a selected case study. Finally, we discuss related work, and finish with a few concluding remarks.

Pattern Databases in AI-Planning

For the sake for simplicity, throughout the paper we consider grounded propositional planning problems in STRIPS notation (Fikes & Nilsson 1971) and stick to sequential plan generation. However, the framework also applies to more general planning domain description languages (Fox & Long 2001).

Grounded Propositional Planning

Most successful planners perform grounding.

Definition 1 *A* grounded propositional planning problem *is a finite state space problem $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$, where $\mathcal{S} \subseteq 2^A$ is the set of states, 2^A is the power set of set of propositions A, $\mathcal{I} \in \mathcal{S}$ is the initial state, $\mathcal{G} \subseteq \mathcal{S}$ is the set of goal states, and $\mathcal{O}$ is the set of operators that transform states into states. Operators $o = (\alpha, \beta) \in \mathcal{O}$ have propositional preconditions α, and propositional effects $\beta = (\beta_a, \beta_d)$, where $\alpha \subseteq A$ is the precondition list, $\beta_a \subseteq A$ is the add list and $\beta_d \subseteq A$ is the delete list. Given a state S with $\alpha \subseteq S$ then its successor is $S' = S \cup \beta_a \setminus \beta_d$.*

Sequential plans are defined as follows.

Definition 2 *A* sequential plan *$\pi = (O_1, \ldots, O_k)$ is an ordered sequence of operators $O_i \in \mathcal{O}$, $i \in \{1, \ldots, k\}$, that transforms the initial state $\mathcal{I}$ into one of the goal states $G \in \mathcal{G}$, i.e. there exists a sequence of states $S_i \in \mathcal{S}$, $i \in \{0, \ldots, k\}$, with $S_0 = \mathcal{I}$, $S_k = G$ and S_i is the outcome of applying O_i to S_{i-1}, $i \in \{1, \ldots, k\}$. The length of a plan π is k, and the minimal k is the optimal sequential plan length $\delta(\mathcal{I})$.*

Abstract Planning problems

Instead of the PDB definition based on fact groups (Edelkamp 2001b), in this paper we prefer a formal treatment that directly builds upon the above state space characterization.

Definition 3 *Let $R \subseteq A$ be a set of propositional atoms. A* restriction *ϕ_R is a mapping from 2^A into 2^A defined as $\phi_R(P) = \{a \in P \mid a \in R\}$. For $\phi_R(P)$ we also write $P|_R$.*

Restrictions imply planning problem abstractions.

Definition 4 *An* abstract planning problem *$\mathcal{P}|_R = < \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$ of a grounded propositional planning problem $< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ with respect to a set of propositional atoms R is defined by*

1. $\mathcal{S}|_R = \{\phi_R(S) \mid S \in \mathcal{S}\}$,
2. $\mathcal{I}|_R = \phi_R(\mathcal{I})$,
3. $\mathcal{G}|_R = \{\phi_R(G) \mid G \in \mathcal{G}\}$,
4. $\mathcal{O}|_R = \{\phi_R(O) \mid O \in \mathcal{O}\}$, *with $\phi_R(O)$ for $O = (\alpha, (\beta_a, \beta_d)) \in \mathcal{O}$ defined as $\phi_R(O) = (\alpha|_R, (\beta_a|_R, \beta_d|_R))$.*

Sequential abstract plans for the abstract planning problem $\mathcal{P}|_R$ are denoted by π_R and optimal abstract sequential plan length is denoted by δ_R.

Abstract operators are fixed by intersecting their precondition, add and delete lists with the set of non-reduced facts in the abstraction. Restriction of operators in the original space may yield *void* operators $\phi_R(O) = (\emptyset, (\emptyset, \emptyset))$ in the abstract planning problem, which are discarded from the operator set $\mathcal{O}|_R$.

The next result shows that our definition of abstraction is sound.

Lemma 1 *Let R be a set of propositional atoms. Restriction ϕ_R is* solution preserving*, i.e., for any sequential plan π for the grounded propositional planning problem $\mathcal{P} = < \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ there exists a sequential plan π_R for the planning state abstraction $\mathcal{P}|_R = < \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$.*

Moreover, an optimal sequential abstract plan π_R^{opt} for $\mathcal{P}|_R$ is always shorter than an optimal sequential plan π^{opt} for $\mathcal{P}$, i.e. $\delta_R(S|_R) \leq \delta(S)$, for all $S \in \mathcal{S}$.

Proof: Let $\pi = (O_1, \ldots, O_k)$ be a sequential plan for $< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$. Then $\pi|_R = (O_1|_R, \ldots, O_k|_R)$ is a solution for $\mathcal{P}|_R = < \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$.

Now suppose, that $\delta_R(S|_R) > \delta(S)$ for some $S \in \mathcal{S}$ and let $\pi^{opt} = (O_1, \ldots, O_t)$ be the optimal sequential plan from S to $\mathcal{G}$ in the original planning space $\mathcal{P}$ then $\pi^{opt}|_R = (O_1|_R, \ldots, O_t|_R)$ is a valid plan in $\mathcal{P}|_R$ with plan length less or equal to $t = \delta(S)$. This is a contradiction to our assumption. □

Strict inequality $\delta_R(S|_R) < \delta(S)$ is given if some operators $O_i|_R$ are void, or if there are alternative even shorter paths in the abstract space.

Planning Pattern Databases

The above setting allows a precise characterization of planning PDBs.

Definition 5 *A planning PDB $\mathcal{PDB}_R$ with respect to a set of propositions R and a grounded propositional planning problem $< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ is a collection of pairs (v, S) with $v \in \mathbb{N}$ and $S \in \mathcal{S}|_R$, such that $v = \delta_R(S)$. Therefore,*

$$\mathcal{PDB}_R = \{(\delta_R(S), S) \mid S \in \mathcal{S}|_R\}.$$

In other words, a PDB is look-up table, addressed by the abstract planning state providing its minimal abstract solution length.

Disjoint Pattern Databases

Disjoint PDBs are important to derive admissible estimates.

Definition 6 *Two planning PDBs $\mathcal{PDB}_R$ and $\mathcal{PDB}_Q$ with respect to $R, Q \subseteq A$, $R \cap Q = \emptyset$ are* disjoint*, if for all $O' \in \mathcal{O}|_R$, $O'' \in \mathcal{O}|_Q$ we have $\phi_R^{-1}(O') \cap \phi_Q^{-1}(O'') = \emptyset$, where $\phi_R^{-1}(O') = \{O \in \mathcal{O} \mid \phi_R(O) = O'\}$.*

Lemma 2 *Two disjoint planning PDBs $\mathcal{PDB}_R$ and $\mathcal{PDB}_Q$ for a grounded propositional planning problem $< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ and sets of propositions P and Q are* additive*: for all $S \in \mathcal{S}$ we have $\delta_P(S|_R) + \delta_Q(S|_Q) \leq \delta(S)$.*

Proof:
Let $< \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$ and $< \mathcal{S}|_Q, \mathcal{O}|_Q, \mathcal{I}|_Q, \mathcal{G}|_Q >$ be abstractions of $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ according to P and Q, respectively, and let $\pi = (O_1, \ldots, O_k)$ be an optimal sequential plan for $\mathcal{P}$. Then, the abstracted plan $\pi|_R = (O_1|_R, \ldots, O_k|_R)$ is a solution for the state space problem $< \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$ and $\pi|_Q = (O_1|_Q, \ldots, O_k|_Q)$ is a plan for $< \mathcal{S}|_Q, \mathcal{O}|_Q, \mathcal{I}|_Q, \mathcal{G}|_Q >$. We assume that all void operators in $\pi|_Q$ and $\pi|_R$, if any, are removed. Let k_Q and k_R be the resulting respective lengths of $\pi|_Q$ and $\pi|_R$.

Since the PDBs $\mathcal{PDB}_R$ and $\mathcal{PDB}_Q$ are disjoint, $O' \in \pi|_R$ and all $O'' \in \pi|_Q$ we have $\phi_R^{-1}(O') \cap \phi_Q^{-1}(O'') = \emptyset$. Therefore, $\delta_R(S|_R) + \delta_Q(S|_Q) \leq k_R + k_Q \leq \delta(S)$. □

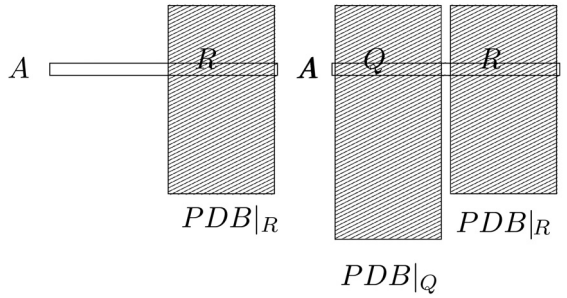

Figure 1: Illustration of a Planning PDB $\mathcal{PDB}_R$ and Two Disjoint Planning PDBs $\mathcal{PDB}_R$ and $\mathcal{PDB}_Q$.

In Figure 1 we have illustrated (disjoint) planning PDBs with respect to the given underlying set A of propositions to encode a state.

In practice some operators remain non-void in different abstraction. For example, in our abstractions Logistics always yields disjoint PDBs, while in Blocks World some interdependent operators remain, since operators in Logistics modify either the location of a package, a truck or an airplane without affecting the others, while in Blocks World a *stack* operation changes both the status of the hand and the block.

In order to retain admissible estimates for the latter case, during construction conflicting operators can be assigned to cost zero for all but one PDB. Nevertheless, this technique of enforced *admissibility* may reduce the quality of the inferred estimate.

Partitions and Storage

Next, we consider sets of pattern databases.

Definition 7 *A* partition $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ *is a collection of propositional sets Q_i, $i \in \{1, \ldots, k\}$, with $Q_i \cap Q_j = \emptyset$, $1 \leq i < j \leq k$ and $\bigcup_{i=1}^{k} Q_i = A$. A* planning space partition $\mathcal{P}_\mathcal{Q}$ *according to a partition $\mathcal{Q}$ is a collection of planning problems $< \mathcal{S}|_{Q_i}, \mathcal{O}|_{Q_i}, \mathcal{I}|_{Q_i}, \mathcal{G}|_{Q_i} >$, $Q_i \subseteq \mathcal{Q}$, $i \in \{1, \ldots, k\}$.*

The following result is an immediate generalization of Lemma 2.

Lemma 3 *Pairwise disjoint planning PDBs according to planning space partition $\mathcal{P}_\mathcal{Q}$ for a grounded propositional planning problem $< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$ and a* partition $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ *are* additive*, i.e. for all $S \in \mathcal{S}$ we have $\delta_{Q_1}(S|_{Q_1}) + \ldots + \delta_{Q_k}(S|_{Q_k}) \leq \delta(S)$.*

Finding a suitable partition that leads to pairwise disjoint or to almost pairwise disjoint planning PDBs is not trivial. For two databases the task is a variant of the graph partitioning problem (GPP), which divides the vertices of a given graph into two equally sized subsets, so that the number of edges from one subset the other one is minimized. In our setting, vertices correspond to atoms and edges to operators. Since GPP is NP complete and the number of atoms is considerably large, we mimic the approach of (Edelkamp 2001b), which simplifies the problem of finding a suitable partition of the set of facts to a form of bin-packing (BPP). For this case, interdependencies are neglected. A group can be added to an already existing abstraction, if the combined state space still fits into main memory. BPP is NP complete but has several efficient approximation algorithms. Currently, we study how goal fact dependencies can improve the established partition.

In explicit PDB construction, the PDBs themselves and the transposition tables (Reinefeld & Marsland 1994) are represented as hash tables. Therefore, the limit for PDB construction is the number of (abstract) states that can be hold in main memory. For improving memory consumption, (Edelkamp 2001b) proposes perfect hash-tables, with a hash function that assigns each state to a unique number. In our simpler setting, each state $S|_R = \bigcup_{i \in I} a_i$, for the index set I and atom list $(a_i)_{i \in I}$, is hashed to $\sum_{i \in I} 2^i$, for a maximum of $2^{|R|}$ hash addresses.

Symbolic Pattern Databases

We abstract from the internal representation of sets of states as binary decision diagram (BDDs) (Bryant 1992). It suffices to know that there BDDs are unique, space-efficient data structures for representing and manipulating Boolean formulae.

States and Operators

Boolean formulae may represent sets of states.

Definition 8 *A* symbolic representation *for a state $S \in \mathcal{S}$ with $\mathcal{S} \subseteq 2^A$ is a set of boolean variables $b_1, \ldots, b_{|A|}$, with b_i encoding the truth of propositional atom a_i in a given state, $i \in N = \{1, \ldots, |A|\}$.*

If $S = \bigcup_{i \in I} a_i$, then its encoding is $(\bigwedge_{i \in I} b_i) \wedge (\bigwedge_{i \in N \setminus I} \neg b_i)$. Sets of states $\bigcup_{j \in J} S_j$ are encoded as $\bigvee_{j \in J} ((\bigwedge_{i \in I_j} b_{ij}) \wedge (\bigwedge_{i \in N \setminus I_j} \neg b_{ij}))$.

Transitions relations are Boolean expressions for operator application. They encode all valid (state, successor state) pairs utilizing twice the number of Boolean state variables; $2 \cdot |A|$ in our case. In practice, the transition relation is generated as the disjunct of the representations of all grounded operators, which in turn are defined as Boolean expressions of their precondition, add and delete lists.

Definition 9 *The* transition relation $T(b, b')$ *of a set of operators $O \in \mathcal{O}$ is the disjunct $T(b, b') = \bigvee_{O \in \mathcal{O}} T_O(b, b')$. For $O = (\alpha, (\beta_a, \beta_d))$ we have $T_O(b, b') = (\bigwedge_{a_i \in \alpha} b_i) \wedge (\bigwedge_{a_j \in \beta_a} b'_j) \wedge (\bigwedge_{a_k \in \beta_d} \neg b'_k) \wedge \text{frame}(b, b')$, where* frame *encodes that all other atoms are preserved, i.e.* $\text{frame}(b, b') = \bigwedge_{a_j \notin \alpha \cup \beta_a \cup \beta_b} (b_j \equiv b'_j)$.

Similarly, the relaxed transition relation $T|_R$ *according the set of proposition R is constructed with respect to the set of operators $O|_R = (\alpha|_R, (\beta_a|_R, \beta_d|_R))$*

The image I of the state set *From* with respect to the transition relation T is computed as $I(b') = \exists b' \ (T(b, b') \wedge \mathit{From}(b'))$. In this image computation, $T(b, b')$ is not required to be built explicitly, since with $T(b, b') = \bigvee_{O \in \mathcal{O}} T_O(b, b')$ we have $I(b') = \bigvee_{O \in \mathcal{O}} (\exists b' \ T_O(b, b') \wedge \mathit{From}(b'))$.

Therefore, the monolithic construction of $T(b, b')$ can be bypassed. Our current implementation organizes the image computation in form of a balanced tree. Through the success of conjunctive partitioning and reordering techniques in hardware verification (Meinel & Stangier 2001), refined disjunctive partitioning approaches are an apparent issue for future research.

Pattern Database Construction

Complete symbolic breadth-first search (BFS) is one form of reachability analysis of the planning space. Let $\mathcal{S}_i$ be the set of planning states reachable from the initial state S in i steps, initialized by $\mathcal{S}_0 = \mathcal{I}$. An encoded state S belongs to $\mathcal{S}_i$ if it has a predecessor S' in the set $\mathcal{S}_{i-1}$ and there exists an operator which transforms S' into S. All sets of states are identified by their respective characteristic formulae.

We apply backward symbolic exploration for SPDB construction as follows. The symbolic PDB $\mathcal{PDB}|_R$, is initialized to $\mathcal{G}|_R$ and, as long as there are newly encountered states, we take the current list of horizon nodes and generate the predecessor list with respect to $T|_R$. Then we attach the current BFS level to the new states, merge them with the set of already reached state states and iterate. In the following algorithm *Construct Symbolic Pattern Database*, *Reached* is the set of visited states, *Open* is current search horizon, and *Pred* is the set of predecessor states.

Algorithm *Construct Symbolic Pattern Database*
Input: *Planning space abstraction*
$\mathcal{P}|_R = < \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R >$
Output: *Symbolic Pattern Database* $\mathcal{PDB}|_R$

```
Reached(b') ← Open(b') ← G|_R(b')
i ← 0
while (Open ≢ ∅)
  Pred(b) ← ∃b' Open(b') ∧ T|_R(b, b')
  Pred(b') ← Pred(b) [b ↔ b']
  Open(b') ← Pred(b') ∧ ¬ Reached(b')
  PDB|_R ← PDB|_R ∨ (v = i ∧ Open(b'))
  Reached(b) ← Reached(b) ∨ Open(b)
  i ← i + 1
return PDB|_R
```

Weightening the heursitic estimate according to a factor γ is achieved by setting $(v = i)$ to $(v = \gamma i)$.

Note that beside the capability to represent large sets of states in the exploration, symbolic PDB have one further advantage to explicit ones: fast initialization. In the definition of most planning problems $\mathcal{G}$ is not given as a collection of states, but as a smaller selection of atoms a_i, $i \in I' \subset I$. In explicit PDB construction all states $G \in \mathcal{G}$ have to be generated and to be inserted into the BFS queue, while for the symbolic construction, initialization is immediate.

Explicit Pattern Database Search

SPDBs can easily be incorporated to any explicit heuristic search engine, e.g. Algorithm *Explicit Pattern Database Search* illustrates A* exploration with SPDBs.

Algorithm *Explicit Pattern Database Search*
Input: *Planning space* $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$,
Symbolic Pattern Database $\mathcal{PDB}|_R$
Output: Solution length $\delta(\mathcal{I})$

```
Insert(Open, (I, PDB|_R(I)))
while (Open ≠ ∅)
  S ← DeleteMin(Open)
  if (S ∈ G) return g(S)
  for all successors S' of S
    f(S') ← f(S) + 1 + PDB|_R(S') − PDB|_R(S)
    if (Search(Open, S))
      if (f'(S) < f(S))
        DecreaseKey(Open, (S, f'(S)))
    else Insert(Open, (S, f'(S)))
```

The set of horizon nodes *Open* is represented as a priority queue with usual access operations *DeleteMin*, *Insert*, and *DecreaseKey*. For the sake of brevity, we have omitted re-opening and concentrate on only one PDB, since generalizations to planning pattern partitions $\mathcal{PDB}|_{\mathcal{Q}}$ are easy to obtain.

For each extracted state S we have $f(S) = g(S) + h(S)$, where g is the actual distance to state S. The new f-value of a successor S' is calculated as $f(S') = g(S') + h(S') = g(S) + 1 + h(S') = f(S) + 1 + (h(S') - h(S))$.

Apparently, the design of explicit search algorithms with symbolic PDB heuristics is not different to the design of algorithms for any other incorporated estimate. The only change is the computation of the estimate $h(S)$ for state S with respect to $\mathcal{PDB}|_R$.

To query the symbolic PDB $\mathcal{PDB}|_R$ with state $S = \bigcup_i a_i$, denoted as $\mathcal{PDB}|_R(S)$, we first compute the symbolic representation $\bigwedge_i b_i$ of S. Then we determine the conjunct of $\bigwedge_i b_i$ with $\mathcal{PDB}|_R$. The operation yields $(\bigwedge_j v_j) \wedge (\bigwedge_i b_i)$, where $(\bigwedge_j v_j)$ encodes the estimate. Last but not least, the formula $(\bigwedge_j v_j)$ is converted back to an ordinary numerical quantity. Since $\bigwedge_i b_i$ is already simple, computing its conjunct with $\mathcal{PDB}|_R$ is fast in practice. Conversion would not be necessary at all, if instead of BDDs – as in our implementation – arithmetic decision diagrams (ADDs) were used. For this case, the heuristic estimate is determined in time linear to the encoding length. If several SPDs $Q_1, \ldots, Q_k$ are addressed, we compute the estimate $h(S) = h_1(S) + \ldots + h_k(S)$ with respect to $h_1(S) = \mathcal{PDB}|_{Q_1}(S)$, $h_2(S) = \mathcal{PDB}|_{Q_2}(S)$, $\ldots$, $h_k(S) = \mathcal{PDB}|_{Q_k}(S)$.

Symbolic Pattern Database Search

In the symbolic version of heuristic search the algorithm determines all successor states for a set of successors in one evaluation step. The heuristic is represented as a binary relation of estimate and state variables. In the exploration algorithm the open list of generated nodes is represented as an encoded set of buckets with bucket f containing all states in open with merit $f = g + h$.

Algorithm *Explicit Pattern Database Search* starts with the Boolean representation of the initial state, attaches its estimate and similarly to the explicit case, it iterates state extraction and successor set generation until the goal has been found. However, in contrast to the setting above, we extract sets of states *Min* with minimum f-value $f_{\min}$ and compute their respective successor sets *Succ* by applying the transition relation. To find the f-value for the successor states we apply symbolic representation of the heuristic estimator $\mathcal{PDB}|_R$ to the pre-image and the image of transition relation application. The correctly associated values h, h' are then quantified to yield the successor f-value $(f = f_{\min} + h' - h + 1)$. For best-first search the formula simplies to $f = h'$.

The superimposed distribution $\mathcal{PDB}|_{R+Q}$ of two PDBs $\mathcal{PDB}|_R$ and $\mathcal{PDB}|_Q$ approximates $\mathcal{PDB}|_{R \cup Q}$. It can be computed beforehand to be conjuncted with *Min* and *Succ* in the algorithm. The alternative avoids the pre-computation of $\mathcal{PDB}|_{R+Q}$ and combines $\mathcal{PDB}|_R$ and $\mathcal{PDB}|_Q$ with *Min* and *Succ* during the execution. Our implementation allows both options.

Algorithm *Symbolic Pattern Database Search*
Input: *Planning space* $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$,
Symbolic Pattern Database $\mathcal{PDB}|_R$
Output: Solution length $\delta(\mathcal{I})$
$\mathit{Open}(f,b) \leftarrow \mathcal{PDB}|_R(f,b) \wedge \mathcal{I}(b)$
do
$f_{\min} = \min\{f \mid f \wedge \mathit{Open}(f,b) \neq \emptyset\}$
$\mathit{Min}(f) \leftarrow \exists f\ (\mathit{Open}(f,b) \wedge f = f_{\min})$
$\mathit{Rest}(f,b) \leftarrow \mathit{Open}(f,b)\ \wedge\ \neg\, \mathit{Min}(b)$
$\mathit{Min}(h,b) \leftarrow \mathcal{PDB}|_R(h,b) \wedge \mathit{Min}(b)$
$\mathit{Succ}(h,b') \leftarrow \exists b\, T(b,b') \wedge \mathit{Min}(h,b)$
$\mathit{Succ}(h,b) \leftarrow \mathit{Succ}(h,b')\ [b \leftrightarrow b']$
$\mathit{Succ}(h,h',b) \leftarrow \mathcal{PDB}|_R(h',b) \wedge \mathit{Succ}(h,b)$
$\mathit{Succ}(h,h',f,b) \leftarrow \mathit{Succ}(h,h',b)\ \wedge\ f = f_{\min}$
$\mathit{Succ}(f',b) \leftarrow \exists h,h',f$
$\mathit{Succ}(h,h',f,b) \wedge (f' = f + h' - h + 1)$
$\mathit{Succ}(f,b) \leftarrow \mathit{Succ}(f',b)\ [f \leftrightarrow f']$
$\mathit{Succ}(f,b) \leftarrow \mathit{Succ}(f,b) \wedge \neg\, \mathit{Reached}(b)$
$\mathit{Open}(f,b) \leftarrow \mathit{Rest}(f,b) \vee \mathit{Succ}(f,b)$
$\mathit{Succ}(b) \leftarrow \exists f\ \mathit{Succ}(f,b)$
$\mathit{Reached} \leftarrow \mathit{Reached}(b) \vee \mathit{Succ}(b)$
while $(\mathit{Open} \wedge \mathcal{G} \equiv \emptyset)$
return $f_{\min}$

Given a uniform weighted problem graph and a consistent heuristic $(h(v) - h(u) + w(u,v) \geq 0)$ the worst-case number of iterations has been shown to be $O(\delta^2(\mathcal{I}))$ (Edelkamp & Reffel 1998).

Search Tree Prediction

Heuristic PDBs are also an efficient mean for heuristic search tree prediction, since they approximate the overall distribution of heuristic estimates in the state space. Assuming that patterns occur equally likely in the search space, the overall probability of estimate h being less than or equal to k is

$$P(h \leq k) = |\{P \in \mathcal{PDB} \mid h(P) \leq k\}|/|\mathcal{PDB}|.$$

To predict the heuristic search tree expansion of the problem graph that is labeled with node costs $f = g + h$, the main result in (Korf, Reid, & Edelkamp 2001) states that the expected total number of tree nodes according to cost threshold θ is approximately equal to

$$\sum_{d=0}^{\theta} n^{(d)} P\,(h \leq \theta - d)\,, \qquad (1)$$

where $n^{(d)}$ is the number of states in the brute-force search tree with depth d and P is the equilibrium distribution, defined as the probability distribution of heuristic values in the limit of large depth. In the framework of spectral analysis, $n^{(d)}$ can be computed in closed form (Edelkamp 2001c).

Since Equation 1 is a very good predictor for the number of nodes in IDA* and yields at least a good trend for A*'s exploration effort, it has been used for evaluating the effectiveness of PDBs (Hernadvölgyi 2000). For the sake

of simplicity we focus on the mean heuristic value $\overline{h} = \sum_k k \cdot |\{P \in \mathcal{PDB} \mid h(P) = k\}|/|\mathcal{PDB}|$. In the limit of large θ, the branching factor b, i.e. the ratio of search tree nodes with respect to two consecutive threshold values, converges (Edelkamp 2001c). The effect of the heuristic is to reduce the search tree size from $O(b^d)$ to $O(b^{d-c})$ for some constant $c \approx \overline{h}$ (Korf 1997).

Therefore, heuristics are best thought of as offsets to the search depth. The higher the average heuristic value, the smaller the *effective search tree depth*, i.e., the shallower the search with respect to the brute-force search tree. The following case study displays the effect of explicit and symbolic PDBs on $\overline{h}$.

Case Study

As a case study we chose Blocks World, since finding optimal plans is still a challenge for domain-independent planners. No form of knowledge was added to the planner, we switched off all branching cuts. Cuts significantly speed up exploration, but most proposed control knowledge in planning is domain-dependent or apply to certain sets of benchmark domains only.

Pattern Database Construction

In Tables 1 and 2 we present the results on constructing symbolic PDBs in selected Blocks World problems of the AIPS-2000 set. The total number of pattern s in the databases and the respective averaged heuristic estimates $\overline{h}$ are shown. As the problems size p gets larger, more and more PDBs were generated (separated by /).

p	s	$\overline{h}$
4	108	6.20
5	1,029	8.85
6	12,288	11.49
7	26,244/8	11.72/1.62
8	50,000/80	11.26/3.57
9	87,846/968	11.69/6.37
10	145,152/13,824	11.35/8.59
11	228,488/228,488	11.38/11.39
12	27,440/27,440/2,156	8.64/8.64/5.79
13	37,125/37,125/37,125	8.66/8.66/8.66
14	49,152/49,152/49,152/15	8.03/8.03/8.03/1.80
15	63,869/63,869/63,869/255	8.69/8.69/9.35/3.75

Table 1: Number of States s and Mean Heuristic Value $\overline{h}$ in Blocks World PDBs according to $m = 2^{20}$.

The averaged heuristic estimates increase significantly when moving from $m = 2^{20}$ to $m = 2^{30}$, while the number of PDBs shrinks accordingly.

Table 3 compares the growth of the symbolic representation with respect to the number of states. We took the first PDB in Blocks World Problem 15 with $m = 2^{30}$ as an example. The predicted state space size is 410,338,673. Since this corresponds to maximum perfect hash table capacity, explicit exploration was no longer available.

p	s	$\overline{h}$
4	1,08	6.20
5	1,029	8.85
6	12,288	11.49
7	1,777,147	14.14
8	3e+06	16.80
9	5.84e+07	19.47
10	1.43e+08	19.55/1.72
11	2.89e+07/1,690	17.05/6.47
12	5.27e+07/27,440	16.29/8.05
13	9.11e+07/506,250	16.89/10.93
14	1.50e+08/1.04e+07	16.56/13.77
15	2.41e+08/2.41e+08	16.59/16.56

Table 2: Number of States s and Mean Heuristic Value $\overline{h}$ in Blocks World PDBs according to $m = 2^{30}$.

d	t	b	s
0	0.00s	35	1
2	0.01s	103	39
4	0.01s	311	586
6	0.03s	858	5,792
8	0.13s	2,576	55,911
10	0.66s	6,879	538,771
12	2.56s	14,583	4.01e+06
14	7.60s	24,547	1.87e+07
16	12.60s	30,238	4.51e+07
18	10.69s	22,592	4.14e+07
20	4.48s	7,655	1.02e+07
22	0.45s	993	467,551

Table 3: Node Count b and Number of States s for Constructng a SPDB in Blocks-World Problem 15.

Table 3 depicts the node and state counts for each iteration in the construction phase. The results indicate that by far more states are encountered than BDD nodes were necessary to represent them. In this case the effect of symbolic representation corresponds memory gains of up to about two orders of magnitude. With $m = 2^{40}$, for which exploration was still possible, the effect increases up to about four orders of magnitude. We also observe that the peak node count for the is also established earlier then the peak state count.

Explicit Search

Table 4 compares the CPU times[1] of explicit and symbolic PDB construction with the exploration time in explicit

[1]Most of the experiments were run on a Sun UltraSparc Workstation with 248 MHz. Since exact running-times reflect too many issues of the current implementation, for the interpretation of results we are mainly interested in comparing performance growth. Memory was restricted as follows. The pattern databases were either limited to $m = 2^{20}$ states or $m = 2^{30}$ states; for explicit search we chose 2,000,000 stored states as the exploration bound.

search. We took the same heuristic estimate and $m = 2^{20}$. Since the qualities of the different PDBs match, the same set of states was considered. The search algorithm we chose was A* with weight 2 ($f = g + 2h$). Besides the problem number, the depth of the solution and the number of expanded nodes, we also displayed PDB construction time t^c and explicit search time t^s with respect to explicit pattern and symbolic PDBs, subscripted by e and s.

p	d	e	t_e^c	t_e^s	t_s^c	t_s^s
4	6	7	0.01s	0.00s	0.03s	0.00s
5	12	15	0.05s	0.00s	0.04s	0.00s
6	12	13	0.49s	0.00s	0.30s	0.00s
7	24	40	1.39s	0.00s	0.52s	0.01s
8	20	1,590	2.98s	0.10s	0.67s	0.40s
9	32	34,316	6.18s	3.75s	0.81s	13.92s
10	34	47,657	12.55s	5.72s	1.23s	16.35s
11	38	7,941	0.91s	3.09s	1.80s	3.04s
12	38	34,323	4.67s	5.31s	1.73s	13.24s
13	-	-	10.55s	-	2.45s	-
14	40	254,769	15.16s	58.23s	3.32s	150.43s
15	-	-	21.88s	-	5.51s	-

Table 4: Time for PDB Construction and Explicit Search in Blocks World.

In the result we obtain a trade-off between explicit and symbolic search. While symbolic PDB construction is significantly faster, search time is larger. As indicated above, an ADD implementation for the heuristic lessens the per-node retrieval overhead for SPDBs.

Symbolic Search

In this set of experiments we measured the performance of the symbolic search algorithm. We used forward heuristic search with respect to the provided SPDBs, accompanied by a symbolic backward traversal. The search direction was chosen in favor to the exploration side that used less time in the previous iteration. The memory bound was set to $m = 2^{30}$, so at most 2 PDBs were constructed. By the choice of dependent PDBs, the results in Table 5 were not necessarily optimal. The headings are read as follows: p is the problem number, d is the depth of the solution, i_f and i_b are the number of forward and backward iterations, t_s^c is the PDB construction time, t_s^c is the symbolic search time, and t_b is time for bidirectional symbolic BFS.

The peak PDBs size at $p = 11$ reflects that the maximum number of patterns in the database is roughly equal to the state space size. As the comparison of t_s^s with t_b shows, we can obtain better results as with bi-directional symbolic BFS, which besides SAT enumeration (Kautz & Selman 1996), is state-of-the-art in optimal sequential plan generation. Another observation is that in case of failure,

For symbolic exploration we allocated at most 8,000,000 BDD nodes.

p	d	i_f	i_b	t_s^c	t_s^s	t_b
4	6	6	0	0.02s	0.21s	0.17s
5	12	12	0	0.04s	0.30s	0.30s
6	12	12	0	0.30s	0.43s	1.09s
7	20	20	0	3.95s	0.76s	11.34s
8	18	9	11	0.67s	0.40s	2.80s
9	30/32	25	12	0.81s	13.92s	38.16s
10	34	60	12	66.16s	58.02s	297.51s
11	32/38	52	11	1,218s	261.76s	742.14s
12	34/36	142	15	38.57s	224.13s	1,059s
13	-	147	17	48.88s	time	memory
14	-/38	52	11	59.05s	150.92s	memory
15	-	-	-	time	-	memory

Table 5: CPU Performance for PDB Construction and Symbolic Search in Blocks World.

symbolic heuristic search with PDBs never runs out of memory but out of time. For symbolic engines this is a very unusual behavior. In Problem 13 time was exceeded in exploration, while for Problem 15 the time threshold was encountered when merging the two PDBs into a combined one.

PDBs have also been successfully applied to other challenging propositional planning domains (Edelkamp 2001b). The results do transfer to the symbolic setting. In simple domains like Gripper, all running times (for A* and best-first explicit and symbolic exploration with explicit and symbolic PDBs) were bounded by far less than a minute. Table 6 displays the CPU performance of explicit search with (S)PDBs in Logistics. In symbolic best-first search ($f = h$) we solved each problem in less than a minute, while symbolic A* ($f = g + h$) and symbolic BFS ($f = g$) failed in larger problem instances. The space bound is 2^{20} and, once more, the search algorithm is A* with weight 2. The sav-

p	d	e	t_e^c	t_e^s	t_s^c	t_s^s
4	20	21	1.52s	0.00s	0.11s	0.00s
5	27	33	0.70s	0.01s	0.08s	0.02s
6	25	30	5.90s	0.00s	0.10s	0.07s
7	37	48	27.54s	0.01s	0.64s	0.06s
8	34	50	26.99s	0.02s	2.33s	0.06s
9	36	43	27.62s	0.01s	0.91s	0.08s
10	36	81	52.97s	0.04s	1.01s	0.12s
12	44	79	53.02s	0.02s	1.14s	0.11s
13	75	138	22.92s	0.08s	4.18s	0.42s
14	66	143	22.99s	0.09s	4.42s	0.36s
15	84	186	23.39s	0.15s	4.81s	0.51s

Table 6: Time for PDB Construction and Explicit Search in Logistics.

ings in explicit database search time are counter-balanced by acorresponding increase in construction time. One interesting observation in Logistics and Gripper is that through

the highly asynchronous problem structure even very small databases lead to good accumulated estimates. Therefore, very large problems can effectively be solved with PDB.

We have not considered metric planning problems, where PDBs are to be constructed according to their shortest-path distances to the goal. Since the awarded, 2002 competition version of MIPS [2] schedules sequential plans, we integrated (S)PDB for sequential plan generation with mixed results.

Related Work

In the Model-Based Planner, MBP, the paradigm of planning as symbolic model checking (Giunchiglia & Traverso 1999) has been implemented for *non-deterministic* planning domains (Cimatti, Roveri, & Traverso 1998), which classifies in weak, strong, and strong-cyclic planning, with plans that are represented as complete state-action tables. For *partial observable* planning, exploration faces the space of belief states; the power set of the original planning space. Therefore, in contrast to the successor set generation based on action application, observations introduce "And" nodes into the search tree (Bertoli *et al.* 2001). Since the approach is a hybrid of symbolic representation of belief states and explicit search within the "And"-"Or" search tree, simple heuristic have been applied to guide the search. The need for heuristics that trade information gain for exploration effort is also apparent need in *conformant* planning (Bertoli, Cimatti, & Roveri 2001). The authors label the obtained search algorithms as a new paradigm of *heuristic-symbolic* search and report savings in orders of magnitudes with respect to BFS. In contrast to our approach, where Boolean function encode perfect knowledge, the symbolic representation compensates partial knowledge of the current state. Moreover, Bertoli et al. consider heuristics for guiding the choice of the belief states with no symbolic heuristic estimates as in our case. Since the first estimate was rather trivial – it preferred belief states with low cardinality – recent work (Bertoli & Cimatti 2002) proposes improved heuristic for belief space planning. Nevertheless, we view unpublished work on abstraction (Cimatti, Giunchiglia, & Roveri 2000) closest to our approach of symbolic PDBs. It origins in Abstrips abstractions, but lacks experimental results.

The awarded model checking integrated planning system MIPS (Edelkamp & Helmert 2001) is a competitive deterministic planning system based on model checking methods. The planner incorporates symbolic, explicit and metric heuristic planning strategies (Edelkamp 2002). Its type-inference mechanism and fact enumeration algorithm groups mutually exclusive facts to infer a concise state endoding (Edelkamp & Helmert 1999). Heuristic symbolic search with the (weighted) BDDA* algorithm has shown a significant time and space reduction for planning problems that were intractable for breadth-first symbolic exploration (Edelkamp 2001a). As a symbolic heuristic, the goal was splitted into atoms and either a relaxed plan or the *single-atom* heuristic was computed and accumulated. The approach could not compete with state-of-the art planners, and, different to SPDBs, the pre-compiled symbolic estimates provided no information gain to accelerate explicit heuristic search planners.

The UMOP system parses a non-deterministic agent domain language that explicitly defines a controllable system in an uncontrollable environment (Jensen & Veloso 2000). The planner also applies BDD refinement techniques such as automated transition function partitioning. New result for the UMOP system extends the setting of weak, strong and strong cyclic planning to adversarial planning, in which the environment actively influences the outcome of actions. In fact, the proposed algorithm joins aspects of both symbolic search and game playing. Jensen also reports some preliminary and unpublished successes on planning with domain abstractions. As one drawback, the loss of solution quality seemed to be significant.

With SetA*, (Jensen, Bryant, & Veloso 2002) provide an improved implementation of the symbolic heuristic search algorithm BDDA* (Edelkamp & Reffel 1998) and Weighted BDDA* (Edelkamp 2001a). Based on supplied source code the consise state encoding and the *max-atom* heuristic function of MIPS could be reproduced[3]. One major surplus is to maintain a finer granularity of the sets of states in the search horizon kept in a matrix according to matching g- and h- values. This contrasts the plain bucket representation of the priority queue based on f-values. The heuristic function is implicitly encoded with value differences of grounded actions. Since sets of states are to be evaluated and some heuristics are state rather than operator dependent it has still to be shown how general this approch is. As above the considered planning benchmarks are seemingly simple for single-state heuristic search exploration (Hoffmann 2002; Helmert 2001). We expect better and more general results when applying SPDBs.

Recent, yet unpublished work of Hansen, Zhou, and Feng (Hansen, Zhou, & Feng 2002) also re-implemented BDDA* and suggest that symbolic search heuristics and exploration schemes are probably better to be implemented with algebraic decision diagrams (ADDs), as available in Somenzi's CUDD package. Although the authors achieved no improvement to (Edelkamp & Reffel 1998) to solve the $(n^2 - 1)$-Puzzle, the established generalization to guide a symbolic version of the LAO* exploration algorithm (Hansen & Zilberstein 1998) for *probabilistic* or Markov decision process (MDP) planning, results in a remarkable improvement to the state-of-the-art (Feng & Hansen 2002). Since its input – as in our case – is a symbolic representation of the estimate, the contributed progress in estimate quality calls for generalizations of SPDBs to probabilistic planning.

In BDD-based hardware verification, guided search and prioritized model checking are emerging technologies.

[2] See `www.informatik.uni-freiburg.de/~mmips`.

[3] In their paper, the authors compare SetA* with the implementation of BDDA* in MIPS of early 2001. While the results in Logistics seem plausible, unfortunately, we cannot reproduce the bad behavior of our implementation in the Gripper domain.

(Yang & Dill 1998) used BDD-based symbolic search based on the Hamming distance of two states. This approach has been improved in (Reffel & Edelkamp 1999), where the BDD-based version of A* for the μcke model checker outperforms symbolic BFS exploration for two scalable hardware circuits. The heuristic is determined in a static analysis prior to the search taking the actual circuit layout and the failure formula into account. The approach of symbolic guided search in CTL model checking documented in (Bloem, Ravi, & Somenzi 2000) applies 'hints' to avoid sections of the search space that are difficult to represent for BDDs. This permits splitting the fix-point iteration process used in symbolic exploration into two parts yielding under- and over-approximation of the transition relation. Benefits of this approach are simplification of the transition relation, avoidance of BDD blowup and a reduced amount of exploration for complicated systems. However, in contrast to our approach providing 'hints' requires user intervention. Also, this approach is not directly applicable to explicit exploration, which is our main focus. Prioritized traversals are also concerned for formal hardware verification at IBM (Fraer *et al.* 2000). The approach bases on the work of (Cabodi, Camurati, & Quer 1996) and splits the symbolic search frontier into parts to ease approximate reachability.

Conclusion

This paper puts forth the idea of PDB construction to improve the computed average of the admissible heuristic, which in turn corresponds to a relative decrease in search depth. We have also seen a sound formal treatment for PDBs in planning for both explicit and symbolic construction. The experiments highlight that with symbolic representation and reachability analysis, very large databases can be constructed, for which explicit methods necessarily fail.

The approach improves one of the three major classes of heuristics in planning, namely *Plan Abstraction*. The other two are: *Plan relaxation*, as implemented in the FF planner (Hoffmann & Nebel 2001), which is a informative on-line computed estimate, and *Bellman approximation*, as implemented in the *max-atom* and *max-pair* heuristics for HSP, which also consideres groups of atoms. In contrast to this paper *Bellman approximation* simplifies the exploration without simplifying the operator representation (Haslum & Geffner 2000).

PDBs consider subproblem interactions of larger groups and include more knowledge into the estimate than the *max-pair* heuristic. On the other hand, since FF and the PDB heuristics are very different in their characteristics, the natural question arises of how to combine the two for an even better estimate. Even though node expansion is more time consuming for the relaxed plan graph estimate, it yields better information on groups that do not appear in the goal description.

Our implemented proposal is to group the number of add atoms that match the backward plan extraction of the relaxed plan graph in FF according to the obtained group partitioning. With respect to each planning space abstraction the better FF or PDB, value can be selected. Since FF's heuristic is somewhat misguided in Blocks World, yielding very low estimates in states far away from the goal state, we can achieve almost arbitrary large improvements for A*-like searches.

Our approach accelerates both explicit and symbolic search. Explicit heuristic search planners can now access better off-line estimates and by weighting the symbolic heuristic search algorithm we can scale the solution quality. Symbolic heuristic search planning – possibly better to be implemented with ADDs – now appears as a real competitor for *blind* symbolic breadth-first exploration. Moreover, the paper provides a bridge from explicit to symbolic search. In both planning and model checking there are two distinctive research branches according to the chosen representation. We have established an effective interplay between these methods by combining state-of-the art techniques from both fields. Future research on checking safety property will try to consolidate these findings in model checking domains.

Acknowledgment The author thanks DFG for support in the projects Ot 64/11-3 and Ed 74/2-1.

References

Bertoli, P., and Cimatti, A. 2002. Improving heuristics for planning as search in belief space. In *Artificial Intelligence Planning and Scheduling (AIPS)*. This volume.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Bertoli, P.; Cimatti, A.; and Roveri, M. 2001. Heuristic search symbolic model checking = efficient conformant planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 467–472.

Bloem, R.; Ravi, K.; and Somenzi, F. 2000. Symbolic guided search for CTL model checking. In *Conference on Design Automation (DAC)*, 29–34.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24(3):142–170.

Cabodi, G.; Camurati, P.; and Quer, S. 1996. Improved reachability analysis of large finite state machines. In *Computer-Aided Design (CAD)*, 354–359.

Cimatti, A.; Giunchiglia, F.; and Roveri, M. 2000. Abstraction in planning via model checking. Technical report, IRST.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based generation of universal plans in nondeterministic domains. In *National Conference on Artificial Intelligence (AAAI)*, 875–881.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(4):318–334.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding

length. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, 135–147. Springer.

Edelkamp, S., and Helmert, M. 2001. The model checking integrated planning system MIPS. *AI-Magazine* 67–71.

Edelkamp, S., and Reffel, F. 1998. OBDDs in heuristic search. In *German Conference on Artificial Intelligence (KI)*, Lecture Notes in Computer Science, 81–92. Springer.

Edelkamp, S., and Schrödl, S. 2000. Localizing A*. In *National Conference on Artificial Intelligence (AAAI)*, 885–890.

Edelkamp, S. 2001a. Directed symbolic exploration and its application to AI-planning. In *AAAI-Spring Symposium on Model-based Validation of Intelligence*, 84–92.

Edelkamp, S. 2001b. Planning with pattern databases. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science. Springer. 13-24.

Edelkamp, S. 2001c. Prediction of regular search tree growth by spectral analysis. In *German Conference on Artificial Intelligence (KI)*, Lecture Notes in Computer Science. Springer. 154-168.

Edelkamp, S. 2002. Mixed propositional and numerical planning in the Model Checking Integrated Planning System. In *The International Conference on AI Planning and Scheduling (AIPS), Workshop on Temporal Planning*.

Feng, Z., and Hansen, E. 2002. Symbolic heuristic search for factored markov decision processes. In *National Conference on Artificial Intelligence (AAAI)*. To appear.

Fikes, R., and Nilsson, N. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Technical report, University of Durham, UK.

Fraer, R.; Kamhi, G.; B. Ziv, M. Y. V.; and Fix, L. 2000. Efficient reachability analysis for verification and falsification. IBM FV'2000 Summer Seminar. www.haifa.il.ibm.com/ibmfv2000/abstracts/fraer.html.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *European Conference on Planning (ECP)*, 1–19.

Hansen, E. A., and Zilberstein, S. 1998. Heuristic search in cyclic anc/or graphs. In *National Conference on Artificial Intelligence (AAAI)*, 412–418.

Hansen, E. A.; Zhou, R.; and Feng, Z. 2002. Symbolic heuristic search using decision diagrams. submitted.

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

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*, 140–149.

Helmert, M. 2001. On the complexity of planning in transportation domains. In *European Conference on Planning (ECP)*, Lecture Notes in Computer Science, 349–360. Springer.

Hernadvölgyi, I. T. 2000. Automatic generation of memory based search heursitics. In *National Conference on Artificial Intelligence (AAAI)*. 1103.

Hoffmann, J., and Nebel, B. 2001. Fast plan generation through heuristic search. *Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2002. Local search topology in planning benchmarks: A theoretical analysis. In *Artificial Intelligence Planning and Scheduling (AIPS)*. This volume.

Holte, R. C., and Hernadvölgyi, I. T. 1999. A space-time tradeoff for memory-based heuristics. In *National Conference on Artificial Intelligence (AAAI)*, 704–709.

Jensen, R. M., and Veloso, M. M. 2000. OBDD-based universal planning for synchronized agents in non-deterministic domains. *Artificial Intelligence Research* 13:189–226.

Jensen, R. M.; Bryant, R. E.; and Veloso, M. M. 2002. SetA*: An efficient BDD-based heuristic search algorithm. In *National Conference on Artificial Intelligence (AAAI)*. To appear.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning propositional logic, and stochastic search. In *National Conference on Artificial Intelligence (AAAI)*, 1194–1201.

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

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time Complexity of Iterative-Deepening-A*. *Artificial Intelligence* 129(1–2):199–218.

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

Korf, R. E. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *National Conference on Artificial Intelligence (AAAI)*, 700–705.

Meinel, C., and Stangier, C. 2001. Hierarchical image computation with dynamic conjunction scheduling. In *Conference on Computer Design (ICCD'01)*.

Pearl, J. 1985. *Heuristics*. Addison-Wesley.

Reffel, F., and Edelkamp, S. 1999. Error detection with directed symbolic model checking. In *World Congress on Formal Methods (FM)*, Lecture Notes in Computer Science, 195–211. Springer.

Reinefeld, A., and Marsland, T. A. 1994. Enhanced iterative-deepening search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16(7):701–710.

Yang, C. H., and Dill, D. L. 1998. Validation with guided search of the state space. In *Conference on Design Automation (DAC)*, 599–604.