Planning with Pattern Databases

Stefan Edelkamp
Institut für Informatik
Albert-Ludwigs-Universität
D-79110 Freiburg
eMail: edelkamp@informatik.uni-freiburg.de

October 19, 2000

Abstract

Heuristic search techniques and greedy local search methods effectively find solutions to difficult planning problems. However, when aiming at optimal solutions, the achieved results are relatively weak. To the contrary, pattern data bases are known to significantly improve lower-bound estimates for optimally solving challenging single-agent problems like the 24-Puzzle, Sokoban and the Rubik's Cube.

Therefore, this paper studies the effect of pattern data bases in the context of deterministic planning. We face a fixed state description based on instantiated predicates and provide a general state space abstraction scheme to infer memory-based heuristics, whereas abstractions are found in factoring the planning space.

1 Introduction

General propositional planning is PSPACE complete [3], but when tackling benchmark planning problems at hand, optimality of the achieved solution usually reveals the intrinsic hardness of the problem. For example, deciding the solvability status of Logistic and Block's World problem instances is trivial, and polynomial time (approximation) algorithms for actually solving the problems are easy to obtain. On the other hand, minimizing the solution lengths for these planning instances is computationally hard.

Therefore, we aim at planners that focus optimal plans. Since difficult optimization problems call for exponential resources, we require scalability to approximate the quality of the solution length. We briefly review optimal planning approaches.

Graphplan [1] constructs a layered planning graph containing two types of nodes, action nodes and proposition nodes. In a given layer the preconditions of all operators are matched, such that Graphplan considers instantiated actions at specific points in time. On the other hand, it generates partially ordered plans to exhibit concurrent actions. Graphplan alternates between two phases: *graph extension* to increase the search depth and *solution extraction* to terminate the planning process. Graphplan finds optimal parallel plans and may be interpreted as an admissible heuristic search algorithm that uses the relaxed forward iteration to derive a lower bound for backward search. Graphplan does not approximate solution lengths; it simply exhausts the given resources in time and space.

Another optimal planning approach is symbolic exploration of the state space with Boolean functions. Given a binary encoding of planning states, a Boolean formula f_t describes the set of states reachable in t steps. If f_t represents a set containing a goal state, the problem is solvable and the minimal t is the sequentially optimal solution length. Two approaches contribute to this observation.

Satplan [13] encodes the planning problem with a standard representation of Boolean formula as a set of disjunctive clauses. The operators unfold the start state over time yielding a linear growth in the number of variables. An efficient theorem prover then searches for a satisfying truth assignment.

Another option is the use of binary decision diagrams (BDDs). The data structure provides a unique representation for satisfiability checking [2]. The BDD planning approach is in fact *reachability analysis* in model checking [16]. It applies to both deterministic and non-deterministic planning and the generated plans are optimal in the number of sequential execution steps.

Model checking approach usually cannot approximate. However, two alternatives have been suggested to solve more problems for the cost of solution quality. One option is to combine the BDD exploration with A* by symbolically encoding the heuristic estimate [6] and the other one is to use symbolic abstractions of the domain [12].

Heuristic search is currently the most effective approach in planning. Four of five honored planning systems in the general planning track of the AIPS-2000 competition at least partially incorporate heuristic search, but in traversing the huge state spaces of all combinations of grounded predicates they all rely on inadmissible estimates. Fortunately, the competition problem instances yield good solutions even when invoking incomplete or non-optimal planning algorithms.

The only lower bound estimate in the competition *max-pair* heuristic [9] implemented in the HSP-2 planner. Nevertheless, the applied search algorithm sacrifices optimality by multiplying the estimates with 2. Experimentally, *max-pair* leads to well-informed estimates in misleading domains like Block's World, but is too weak to compete with the FF heuristic in most other domains. This is due to the fact that FF [11] solves a relaxed planning problem for each state to find an appropriate estimate. On the other hand, FF lacks optimality since the heuristic is inadmissible. Furthermore, several cuts in FF such as helpful actions and goal ordering help to avoid local optima in the hill-climbing algorithm. Completeness in undirected problem graphs is guaranteed by breadth-first searching for the next improvement and by turning off cuts in case of backtracks. However, directed problem graphs like Sokoban contain dead-ends from which the hill-climbing algorithm in FF cannot recover.

A recent trend in single-agent search is to calculate the estimate with pattern databases [4]. The goal is to automatically generate heuristics to guide state space search that are defined by distances in an abstract space. In some cases different abstraction yield independent subspaces such that heuristic estimates can be added [15]. In our case we compute a pattern data base prior to the search by projecting the state encoding down to some certain number of bits. Then we exhaustively search for distances with respect to the given goal in the abstract space, which serve as a lower bound estimates for the overall problem. Exponential savings in the search tree sizes allow to completely traverse the abstract planning space and, more importantly, even if searching the entire abstract space might be memory and time consuming, the heuristic estimate will be available in almost constant time. After presenting the pattern data base algorithm and its properties we end up with experiments on benchmark planning problems and draw some concluding remarks.

2 Planning Space Representation

For the sake of simplicity we concentrate on the STRIPS [7], formalism in which each operators is defined by a precondition list P, an add list A, and a delete list D. However, all presented approaches can be extended to any problem description language which yields a fixed state description. Furthermore, we refer to state descriptions and lists as conjunctions of grounded predicates (facts/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 grounded predicates 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 $S \subset F$ is defined as $S' = \{a \in S - D \cup A \mid a \in P\}$. Inverse STRIPS operators o^{-1} are given by $o^{-1} = (P - D \cup A, D, A)$.*

Take as an example the Block's World domain with four operators `pick-up`, `put-down`, `stack`, and `unstack`, where `pick-up` is given by
$P = \{$`(clear ?a)(ontable ?a), (handempty)`$\}$
$A = \{$`(holding ?a)`$\}$, and
$D = \{$`(ontable ?a), (clear ?a), (handempty)`$\}$
for suitable instantiations for `?a`, `?b` and `?c`.

Precompiling is a domain analysis prior to the search. As operators in STRIPS are described by operator schemas with ungrounded variables, one precompiling step is instantiation through a relaxed exploration, starting with the facts in the initial state. Another outcome of the precompiling step are state invariance [8] grouping mutually exclusive facts in order to minimize the state description length [5]. For the example problem 4-1 we end up with 9 groups, namely:

1. `(on c a) (on d a) (on b a) (clear a) (holding a)`
2. `(on a c) (on d c) (on b c) (clear c) (holding c)`
3. `(on a d) (on c d) (on b d) (clear d) (holding d)`
4. `(on a b) (on c b) (on d b) (clear b) (holding b)`
5. `(ontable a) false`
6. `(ontable c) false`
7. `(ontable d) false`
8. `(ontable b) false`
9. `(handempty) false`

The predicate `false` refers to the situation when none of the predicates in a group is fulfilled.

Definition 2 *Let* $G = \{G_1, \ldots, G_k\}$ *with* $G_i \subset F \cup \{\texttt{false}\}$ *for* $i \in \{1, \ldots, k\}$ *be a set of mutually disjoined facts groups. An encoded* state *is a collection of facts* $f_1, \ldots, f_k$ *with* $f_i \in G_i$ *for* $i \in \{1, \ldots, k\}$*. The encoding length is* $\lceil \log(|G_1|) \rceil + \lceil \log(|G_2|) \rceil + \ldots + \lceil \log(|G_k|) \rceil$*. All encoded states span the planning space* $\mathcal{P}$*.*

Hence, the example has an encoding length of 17 bits.

3 Search Space Abstraction

Pattern databases have been effectively applied to the 15-Puzzle [4] and to Rubik's Cube [14] and quite recently, the 24-Puzzle has efficiently been solved with so-called additive pattern databases [15]. In this section we will transfer the concept of pattern data bases to planning.

An concise introduction to state space abstraction and pattern data bases (memory-based heuristics) has been given by [10]. We briefly review their setting. A state is a vector of fixed length and label sets are used to express the operators conveniently, e.g. an operator mapping $\langle A, B, \_ \rangle$ to $\langle B, A, \_ \rangle$ 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 and the operators. A domain abstraction is defined as a mapping ϕ from one label set L to another label set K with $|K| \leq |L|$ and a state space abstraction of the search problem $\langle S_0, O, L \rangle$ is denoted as $\langle \phi(S_0), \phi(O), K \rangle$.

In our case the abstract space is obtained by projecting the fact set.

Definition 3 *Let* F *be the set of grounded predicates. A* planning space abstraction ϕ *is a mapping from* F *to* $F \cup \{\texttt{false}\}$ *such that for each group* G *either for all* $f \in G : \phi(f) = f$ *or for all* $f \in G : \phi(f) = \texttt{false}$.

Therefore, ϕ maps each planning state of the original planning space $\mathcal{P}$ to one in the abstract space $\mathcal{A}$.

Exemplarily, we use two search space abstractions ϕ_o and ϕ_e. The mapping ϕ_o assigns all atoms in groups with odd index to `false` and ϕ_e maps all fluents in groups with even index value to `false`. All groups not containing an atoms in the goal state are also mapped to `false`. For the example we get the following partition of the goal $\{$`(on b c)`, `(on c a)`, `(on a d)`$\}$: $\phi_e(G) = \{$`(on c a)`$\}$ and $\phi_o(G) = \{$`(on a b)`, `(on d c)`$\}$

Abstract operators are defined by intersecting their pre-, add- and del-lists with the set of active facts according to the abstraction. This accelerates the construction of the pattern table, since a lot of operators will simplify and become trivial.

Definition 4 *Let ϕ be a planning space abstraction and $\delta_\phi(S_1, S_2)$ be the shortest path according to two states S_1 and S_2 in $\mathcal{A}$ and S_0 and S_t be the start and the goal states in $\mathcal{P}$, respectively. A* pattern data base *DB is the set of pairs of states with their minimal solution lengths for all instances to the abstract problem space, i.e., $DB(\phi) = \{(\delta_\phi(\phi(S), \phi(S_t)), S) \mid S \in \mathcal{A}\}$.*

$DB(\phi)$ is calculated in a breadth-first traversal starting from the set of goals using inverse operators and is represented as a hash table. Two facts on pattern databases are important. When reducing the state description length n to αn with $0 < \alpha < 1$ the state space and the search tree usually shrink exponentially. Take for example the search in $\{0,1\}^n$. Enumerating all 2^n bit-vectors then opposes an abstract space of $2^{\alpha n}$ elements. The second observation is that once the pattern data base is calculated, accessing the heuristic estimate is fast. Moreover, the pattern data base can be reused in case different initial states.

The definition of $DB(\phi)$ for a set of goal states is straight-forward: all goal states are inserted into the queue of the breadth-first search engine with value zero. For $DB(\phi_e)$ and $DB(\phi_o)$ in our example the result is depicted in Table 1. Additive data bases allow to add estimates according to different abstractions.

Definition 5 *Two pattern data bases $DB(\phi_1)$ and $DB(\phi_2)$ are* additive*, if the sum of both heuristic estimates always underestimates the overall solution length, i.e., $\delta_{\phi_1}(\phi_1(S), \phi_1(S_t)) + \delta_{\phi_2}(\phi_2(S), \phi_2(S_t)) \le \delta(S, S_t)$.*

Abstractions are not always additive. 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

```
1:(clear a)                   3:(on a c) (clear b)
2:(holding a)                 3:(on d b) (clear c)
2:(on b a)                    4:(holding c) (holding b)
2:(on d a)                    4:(on b c) (clear b)
                              4:(on a c) (holding b)
1:(on d c) (clear b)          4:(on c b) (clear c)
1:(on a b) (clear c)          4:(on d b) (holding c)
2:(on d c) (holding b)        4:(on a c) (on d b)
2:(clear c) (clear b)         5:(on b c) (holding b)
2:(on d c) (on d b)           5:(on a b) (on b c)
2:(on a b) (holding c)        5:(on d b) (on b c)
2:(on a c) (on a b)           5:(on c b) (holding c)
3:(clear c) (holding b)       5:(on a c) (on c b)
3:(clear b) (holding c)       5:(on c b) (on d c)
```

Table 1: Two small pattern data bases for the example problem.

ϕ_o is 1 (because the abstraction of o will make p_2 in group 2 true in one step) and the distance under ϕ_e is also 1 (for the same reason). But the distance in the original search space is also 1. However, we can prove the following result.

Theorem 1 *If each operator changes information only in groups according to a given partition then the corresponding databases are additive.*

Proof: An operator in any path within the abstract space planning space according to a given abstraction ϕ contributes one, only if it changes facts in groups G that are available. Therefore, by adding the solution length of different abstract spaces each operator is counted at most once.

For several domains like Logistics and Blocks World operators act locally according to the partition into groups such that the precondition of Theorem 1 is fulfilled. Moreover, Theorem 1 suggests another way to define abstractions: Let I be 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. Define abstraction ϕ_I so that it maps all atom groups not in I to false. Then, for any partitioning of atom groups into such *independent sets*, the corresponding abstraction heuristics is additive.

If the groups are independent with respect to the operators it still remains to find a suitable number of pattern data bases and to partition the groups into clusters. In Fact, we are confronted with a Bin-Packing problem. Given the sizes of groups, find the minimal number of pattern data bases such that each data base does not exceed a certain threshold. Notice that the group sizes are multiplied to estimate the search space. This is not a restriction, since the encoding lengths of the groups are additive. Bin-Packing is NP-hard, but very good approximation algorithms exist. We use (greedy) first-fit.

One remaining issue is how to efficiently implement the pattern data base. We use a perfect hashing scheme that allows to access the data base by a table lookup in time linear to the abstract state description length.

Due to the partitioning 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 $m(l)$ be defined as $m(l) = \prod_{l=1}^{k} |G_{i_{l-1}}|$ with $|G_{i_0}| = 1$. Furthermore, let $p(f)$ and $g(f)$ be the group and the position in the group of fact f, respectively. Then the perfect hash value $p(S)$ of state S is calculated as $p(S) = \sum_{f \in S} p(f) \cdot m(g(f))$. Since perfect hashing uniquely determines a value for the state S, S can be reconstructed given $h(S)$ by extracting all corresponding group and position information that define the facts in S. Therefore, we establish a very good compression since the states in the queue for the breadth-first search from the goals only consumes integer values.

Moreover, the remaining pattern data base DB itself is an integer array of size m_{k+1} encoding the distance values for the corresponding states, initialized with ∞ for pattern that are not encountered. Since queue is only needed for construction and integer elements in current computers consume 4 Bytes memory, we can generate data-bases of 10 million entries and more.

4 Results

All experimental results were produced on a sun Workstation, UltraSPARC-II CPU with 248 MHz.

4.1 Block's World

The first problem we consider is the Block's World problem. Achieving approximate solutions is easy: a 2-approximation can be found in linear time.

Moreover, there are different domain-dependent branching cuts for a forward planners that drastically reduce search space. Therefore, planners like FF find good approximate solutions to this problems with fifty Blocks and more. FF without cuts is misguided in this domain by its heuristic and gets lost in local optima far away from the goal.

Therefore, we concentrate on optimal solutions in this domain. Graphplan itself is bound to about 9 Blocks. No optimal heuristic search engine achieves a good performance, e.g. HSP-2 with *max-pair* is bound to about 6-7 Blocks. Model checking engines like Mips and Satplan are best in this domain in solving up to 13 Blocks. Table 2 depicts that the additive pattern data-base approach (without any cuts) is competitive in this domain in optimally solving up to 12 Blocks.

4.2 Logistics

We applied pattern data bases also to the Logistics domain and solved the entire suite of AIPS-2000. The largest problem instance has 14 trucks traveling in 14 cities. Furthermore there are 4 airplanes available to deliver the total of 41 packages. Three planners that can solve these problems: STAN applies a special-proposed algorithm once it determines fingerprints of movable objects. It elaborates on the FF heuristic search idea. Mips and FF are both hill-climbers with a similar inadmissible relaxed planning heuristics and the results in this domain have about the same characteristics.

In Table 3 and Table 4 We compare the pattern data base approach with the FF-heuristic. For a adequate comparison of the two estimates we applied weighted A* [18] with $\delta = 2$ in both cases and turned off all branching cuts. Note that HSP-2 with *max-pair* is not competitive on this domain.

The quality of the solution achieved with pattern data bases is better than obtained by FF. Also the running time for the former was about a magnitude faster. However, when using cuts and hill climbing, the FF-heuristic turns out to be competitive; the solution quality running times were about the same. This strongly suggests to apply pruning techniques for the pattern data base approach.

In the generation of the pattern data bases we distinguish projections only on the groups that appear in the goal set and with respect to an *enlarged* goal set, in which all not determined group-values according to the abstraction are initially inserted in the BFS queue with value zero. For Logistics

Problem	q	t_p	t_{pd}	l_{pd}	e_{pd}	t_m	l_m
blocks-4-0	1	0.78s	0.00s	6	7	0.50s	6
blocks-4-1	1	0.79s	0.00s	10	12	0.50s	10
blocks-4-2	1	0.78s	0.00s	6	6	0.50s	6
blocks-5-0	1	0.82s	0.00s	12	28	0.60s	12
blocks-5-1	1	0.83s	0.01s	10	24	0.60s	10
blocks-5-2	1	0.82s	0.00s	16	61	0.63s	16
blocks-6-0	2	1.43s	0.00s	12	16	0.77s	12
blocks-6-1	2	1.43s	0.00s	10	15	0.94s	10
blocks-6-2	2	1.44s	0.01s	20	121	0.96s	20
blocks-7-0	2	2.64s	0.00s	20	35	1.05s	20
blocks-7-1	2	13.80s	0.08s	22	1300	2.09s	22
blocks-7-2	2	2.63s	0.04s	20	731	1.88s	20
blocks-8-0	2	35.95s	0.07s	18	945	2.66s	18
blocks-8-1	2	4.73s	0.33s	20	4409	4.12s	20
blocks-8-2	2	4.76s	0.01s	16	71	1.84s	16
blocks-9-0	2	9.34s	6.93s	30	74662	31.83s	30
blocks-9-1	2	9.30s	0.43s	28	4306	7.89s	28
blocks-9-2	2	87.42s	0.16s	26	1914	5.55s	26
blocks-10-0	3	28.49s	25.34s	34	172776	196.89s	34
blocks-10-1	3	24.69s	72.90s	32	449961	172.50s	32
blocks-10-2	3	28.33s	177.08s	34	992232	226.36s	34
blocks-11-0	3	66.04s	35.53s	32	207082	529.98s	32
blocks-11-1	3	65.32s	33.77s	30	192871	1132s	30
blocks-11-2	3	405.88s	44.77s	34	256929	501.44	34
blocks-12-0	3	824.35s	-	-	-	1288s	34
blocks-12-1	3	814.12s	35.53s	34	143113	337.07	34

Table 2: Optimally solving the AIPS-2000 Block's World problems: t_p is the time to build the pattern data base, t the total solution time, l the solution length and e the number of expanded nodes in the search; pd indicates search with pattern data bases and m abbreviates BDD exploration with Mips.

Problem	q	t_p	t_{pd}	l_{pd}	e_{pd}	l_{ff}	e_{ff}
logistics-4-0	2	0.87s	0.00s	20	20	20	21
logistics-4-1	2	0.87s	0.01s	19	24	19	24
logistics-4-2	2	0.88s	0.00s	16	20	15	19
logistics-5-0	2	0.88s	0.01s	28	31	27	31
logistics-5-1	2	0.87s	0.01s	17	17	17	20
logistics-5-2	2	0.87s	0.00s	8	9	8	8
logistics-6-0	2	0.87s	0.01s	26	29	25	28
logistics-6-1	2	0.87s	0.01s	14	17	14	15
logistics-6-2	2	0.88s	0.00s	25	28	25	28
logistics-6-9	2	0.86s	0.00s	24	27	24	27
logistics-7-0	4	0.99s	0.02s	37	51	36	42
logistics-7-1	4	0.99s	0.03s	44	62	45	60
logistics-8-0	4	1.01s	0.01s	32	42	34	49
logistics-8-1	4	0.99s	0.03s	45	60	48	65
logistics-9-0	4	1.00s	0.03s	40	53	36	41
logistics-9-1	4	1.00s	0.01s	30	40	31	36
logistics-10-0	5	2.07s	0.07s	46	91	46	62
logistics-10-1	5	2.04s	0.05s	44	74	42	52
logistics-11-0	5	2.03s	0.06s	53	85	52	75
logistics-11-1	5	2.01s	0.07s	63	94	64	88
logistics-12-0	5	2.04s	0.05s	43	77	44	60
logistics-12-1	5	2.04s	0.08s	72	103	71	95
logistics-13-0	8	3.41s	0.31s	83	207	77	118
logistics-13-1	8	3.41s	0.23s	66	156	67	107
logistics-14-0	8	3.42s	0.23s	61	137	66	121
logistics-14-1	8	3.42s	0.24s	76	142	76	107
logistics-15-0	8	3.42s	0.30s	82	168	83	130
logistics-15-1	8	3.40s	0.24s	71	144	68	84
logistics-16-0	10	3.03s	0.98s	92	181	98	146
logistics-16-1	10	3.10s	0.83s	86	165	89	130
logistics-17-0	10	2.96s	0.94s	99	197	105	181
logistics-17-1	10	3.07s	1.10s	100	229	104	191
logistics-18-0	10	3.00s	1.12s	126	246	132	239
logistics-18-1	10	3.02s	0.66s	83	144	83	113
logistics-19-0	11	3.92s	1.03s	108	269	115	217
logistics-19-1	11	3.94s	0.77s	95	188	100	138
logistics-20-0	11	4.06s	0.87s	112	221	122	238
logistics-20-1	11	3.95s	0.88s	107	198	110	176

Table 3: Solving the Logistics problem with q pattern data bases: t is the total time to construct the tables, t is the time to perform the search of e expansions to find a solution of length l; pd indicates search with pattern data bases and ff abbreviates forward search with FF.

Problem	q	t_p	t_{pd}	l_{pd}	e_{pd}	l_{ff}	e_{ff}
logistics-21-0	11	3.99s	0.96s	118	232	127	219
logistics-21-1	11	3.98s	1.05s	110	262	114	201
logistics-22-0	13	12.40s	2.68s	119	316	126	233
logistics-22-1	13	12.73s	2.88s	121	295	115	158
logistics-23-0	13	12.94s	2.52s	121	304	130	254
logistics-23-1	13	12.76s	2.03s	109	297	115	215
logistics-24-0	13	12.64s	2.16s	139	273	143	216
logistics-24-1	13	12.49s	3.25s	150	374	163	352
logistics-25-0	15	30.98s	8.91s	154	428	171	398
logistics-25-1	15	27.12s	7.49s	162	422	171	397
logistics-26-0	15	30.82s	6.78s	146	340	154	272
logistics-26-1	15	30.78s	8.48s	171	467	177	371
logistics-27-0	15	30.79s	8.12s	151	382	167	436
logistics-27-1	15	30.83s	9.47s	148	442	156	353
logistics-28-0	20	6.60s	11.49s	182	662	193	510
logistics-28-1	20	6.63s	8.96s	162	468	172	397
logistics-29-0	20	6.56s	9.66s	192	462	212	571
logistics-29-1	20	6.60s	9.97s	160	475	175	483
logistics-30-0	20	6.56s	13.07s	193	648	210	574
logistics-30-1	20	6.57s	12.31s	198	600	215	519
logistics-31-0	22	12.91s	41.71s	194	731	202	509
logistics-31-1	22	12.96s	33.12s	196	591	206	533
logistics-32-0	22	12.93s	41.13s	195	736	209	593
logistics-32-1	22	12.98s	41.53s	202	651	220	599
logistics-33-0	22	12.89s	31.69s	204	600	212	476
logistics-33-1	22	12.88s	42.05s	210	710	229	655
logistics-34-0	24	14.10s	33.05s	218	804	224	673
logistics-34-1	24	13.62s	28.20s	204	816	220	792
logistics-35-0	24	14.21s	21.88s	190	558	208	530
logistics-35-1	24	14.25s	20.68s	209	538	217	410
logistics-36-0	24	14.17s	22.53s	204	548	219	552
logistics-36-1	24	14.08s	27.74s	229	694	252	774
logistics-37-0	26	20.92s	51.37s	239	987	-	-
logistics-37-1	26	20.60s	45.46s	242	880	-	-
logistics-38-0	26	20.88s	55.31s	230	1086	-	-
logistics-38-1	26	20.86s	44.96s	223	950	-	-
logistics-39-0	26	20.96s	49.06s	240	927	-	-
logistics-39-1	26	20.96s	69.16s	241	1369	-	-
logistics-40-0	28	30.04s	95.50s	251	1162	-	-
logistics-40-1	28	29.91s	71.51s	237	792	-	-
logistics-41-0	28	30.35s	94.81s	252	1158	-	-
logistics-41-1	28	30.47s	102.12s	273	1334	-	-

Table 4: Solving the Logistics problem with q pattern data bases (continued).

this enlargement leads to better results, while in Block's World partitioning according to the goal atoms is better.

4.3 Sokoban

Sokoban is known to span a directed search space with exponentially many dead end situations, in which some balls cannot be placed onto any goal field. Therefore, hill climbing will eventually encounter one dead-end and fail. Only an overall searching scheme can prevent the algorithm of getting trapped. In the experiments of Table 5 and Table 6 we used a series of 52 automatically generated problems [17]. The screens were then compiled to PDDL, with a one-to-one ball-to-goal mapping such that some problems become unsolvable. Since we are interested in minimal solutions, we compare the pattern data base approach with the optimal planner Mips.

4.4 Gripper

Gripper is known to be hard for Graphplan engines. It spans a very large ($> 4^{\#balls}$) but well structured search space such that greedy search engines find optimal solutions. In Table 7 we compared the FF-heuristic (with cuts disabled) and the pattern data bases heuristic when applying hill climbing. Note that searching Gripper with weighted A* results in too many expansions in the search, for which the lack of move ordering is responsible.

5 Conclusion

Heuristic search is currently the most promising approach to tackle huge problem spaces but usually does not yield optimal solutions. The aim of this paper is to apply recent progress of heuristic search in finding optimal solutions to planning problems by devising an automatic abstraction scheme to construct an use precompiled pattern solutions in the search.

The presented approach is a novel contribution to the field of AI-planning and without using branching cuts we only scratched its potential. Therefore, we will investigate pruning techniques to reduce the apparent large branching factors in planning. Although the heuristic estimates are calculated beforehand, its quality can compete with the involved FF heuristic, which solves a

Problem	q	t_p	t_{pd}	l_{pd}	e_{pd}	t_m	l_m
sokoban-1	3	0.06s	0.02s	42	259	1.92s	42
sokoban-2	3	0.06s	0.15s	148	2748	65.15s	148
sokoban-3	3	0.07s	0.03s	32	763	2.21s	32
sokoban-4	4	0.07s	0.17s	38	2699	6.06s	38
sokoban-5	3	0.07s	0.14s	∞	2642	48.57s	∞
sokoban-6	3	0.06s	0.05s	∞	1047	12.20s	∞
sokoban-7	4	0.06s	0.02s	29	222	2.14s	29
sokoban-8	3	0.06s	0.24s	136	5030	25.57s	136
sokoban-9	3	0.06s	0.03s	55	641	3.92s	55
sokoban-10	3	0.11s	0.01s	18	173	1.23s	18
sokoban-11	3	0.10s	0.08s	∞	2035	13.15s	∞
sokoban-12	3	0.10s	0.03s	24	589	1.79	24
sokoban-13	3	0.11s	0.04s	∞	880	12.48s	∞
sokoban-14	3	0.11s	0.02s	25	453	1.73s	25
sokoban-15	3	0.10s	0.02s	∞	571	4.20s	∞
sokoban-16	4	0.10s	0.11s	63	2228	7.97	63
sokoban-17	3	0.10s	0.04s	∞	760	6.00s	∞
sokoban-18	3	0.10s	0.02s	33	393	2.10s	33
sokoban-19	3	0.10s	0.10s	68	2356	8.95s	68
sokoban-20	4	0.10s	0.10s	57	1884	7.11s	57
sokoban-21	3	0.10s	0.11s	149	2126	38.59s	149
sokoban-22	3	0.11s	0.06s	109	1405	9.82s	109
sokoban-23	3	0.11s	0.02s	28	552	2.14s	28
sokoban-24	0	0.00s	0.00s	∞	0	1.87s	∞
sokoban-25	3	0.11s	0.07s	51	1582	6.50s	51

Table 5: Solving Sokoban problems with q pattern data bases optimally: t_p is the total time to construct the tables, t is the time to perform the search of e expansions to find a solution of length l; pd indicates search with pattern data bases and m abbreviates the planner Mips.

Problem	q	t_p	t_{pd}	l_{pd}	e_{pd}	t_m	l_m
sokoban-26	3	0.11s	0.27s	∞	5616	126.51s	∞
sokoban-27	3	0.09s	0.01s	25	148	1.07s	25
sokoban-28	4	0.10s	0.16s	69	3135	6.50s	69
sokoban-29	3	0.10s	0.01s	27	355	1.52s	27
sokoban-30	3	0.10s	0.15s	130	2746	86.64s	130
sokoban-31	4	0.11s	0.03s	41	638	3.51s	41
sokoban-32	4	0.11s	0.74s	110	12237	115.89s	110
sokoban-33	3	0.11s	0.67s	65	11683	38.01s	65
sokoban-34	0	0.00s	0.00s	∞	0	2.96s	∞
sokoban-35	3	0.10s	0.06s	67	1565	13.56s	67
sokoban-36	4	0.11s	0.80s	72	12767	103.86s	72
sokoban-37	3	0.10s	0.00s	∞	26	19.42s	∞
sokoban-38	3	0.11s	0.01s	28	263	1.85s	28
sokoban-39	3	0.10s	0.03s	41	633	2.99s	41
sokoban-40	3	0.10s	0.06s	51	1227	9.43s	51
sokoban-41	0	0.00s	0.00s	∞	0	1.48s	∞
sokoban-42	0	0.00s	0.00s	∞	0	2.93s	∞
sokoban-43	4	0.11s	0.16s	103	2834	63.75s	103
sokoban-44	4	0.12s	1.27s	93	17601	230.82s	93
sokoban-45	3	0.10s	0.02s	∞	487	3.58s	∞
sokoban-46	4	0.12s	1.68s	77	24589	87.09s	77
sokoban-47	3	0.10s	0.10s	∞	2234	12.18s	∞
sokoban-48	3	0.11s	0.11s	∞	2641	80.49s	∞
sokoban-49	3	0.10s	0.03s	32	479	2.41s	32
sokoban-50	3	0.11s	0.18s	62	3351	21.44s	62
sokoban-51	4	0.10s	0.48s	103	7203	127.29s	103
sokoban-52	4	0.11s	0.17s	70	2959	25.82s	70

Table 6: Solving Sokoban problems with q pattern data bases optimally (continued).

Problem	q	t_p	t_{pd}	l_{pd}	e_{pd}	t_{ff}	l_{ff}	e_{ff}
gripper-1	2	0.10s	0.00s	11	28	0.00s	11	24
gripper-2	3	0.10s	0.00s	17	59	0.01s	17	70
gripper-3	4	0.11s	0.01s	23	102	0.01s	23	160
gripper-4	4	0.10s	0.02s	29	157	0.04s	29	310
gripper-5	5	0.11s	0.02s	35	224	0.07s	35	536
gripper-6	6	0.11s	0.05s	41	303	0.12s	41	854
gripper-7	6	0.11s	0.07s	47	394	0.21s	47	1280
gripper-8	7	0.12s	0.09s	53	497	0.36s	53	1830
gripper-9	8	0.12s	0.13s	59	612	0.59s	59	2520
gripper-10	8	0.13s	0.16s	65	739	0.72s	65	3366
gripper-11	9	0.13s	0.22s	71	878	1.04s	71	4384
gripper-12	10	0.14s	0.28s	77	1029	1.55s	77	5590
gripper-13	10	0.13s	0.36s	83	1192	2.14s	83	7000
gripper-14	11	0.13s	0.44s	89	1367	2.90s	89	8630
gripper-15	12	0.15s	0.56s	95	1554	3.37s	95	10496
gripper-16	12	0.15s	0.68s	101	1753	4.12s	101	12614
gripper-17	13	0.15s	0.79s	107	1964	5.46s	107	15000
gripper-18	14	0.23s	0.97s	113	2187	6.19s	113	17670
gripper-19	14	0.17s	1.16s	119	2422	7.93s	119	20640
gripper-20	15	0.17s	1.37s	125	2669	10.24s	125	23926

Table 7: Solving the Gripper problem with q pattern data bases optimally: t_p is the total time to construct the tables, t is the time to perform the search of e expansions to find a solution of length l; pd indicates search with pattern data bases and ff abbreviates forward search with FF.

relaxed planning problem for *every* expanded state. In difference to the FF heuristic the estimates are available in constant time and admissible, yielding optimal solutions in A*-like search engines. Weighting the estimate helps with to cope with difficult instances for approximate solutions. Last but not least, the pattern data base approach can handle direct problem spaces and prove unsolvability results.

The aim of AI-planning is the design of a general problem solver. Even with the current success of heuristic search this ultimate goal is very unlikely to be met, since Wolpert and Macready [19] show in their *No-Free-Lunch (NFL)* theorems that all a cost function optimizing search algorithms perform equally on the average of all cost functions. If an algorithm A is superior to B on one cost function it will be inferior on another. Therefore, in planning we are better off in having a portfolio of different algorithms and estimates.

Acknowledgments We thank J. Hoffmann for the Sokoban generator and P. Haslum for fruitful discussions.

References

[1] A. Blum and M. L. Furst. Fast planning through planning graph analysis. In *IJCAI*, pages 1636–1642, 1995.

[2] R. E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, pages 688–694, 1985.

[3] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, pages 165–204, 1994.

[4] J. C. Culberson and J. Schaeffer. Searching with pattern databases. In *CSCSI*, pages 402–416, 1996.

[5] S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state en coding length. In *ECP-99*, pages 135–147, 1999.

[6] S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *KI*, pages 81–92, 1998.

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

[8] M. Fox and D. Long. The automatic inference of state invariants in TIM. *JAIR*, 9:367–421, 1998.

[9] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *AIPS*, pages 140–149, 2000.

[10] I. T. Hernádvögyi and R. C. Holte. The automatic creation of memory-based search heuristics. Submitted to AIJ special issue on heuristic search, 2000.

[11] J. Hoffmann and B. Nebel. Fast plan generation trough heuristic search. Submitted to JAIR, 2000.

[12] R. M. Jensen. OBDD-based deterministic planning using the UMOP framework. Talk on AIPS citing his own unpublished work on solving some Logistics problems, 2000.

[13] H. Kautz and B. Selman. Pushing the envelope: Planning propositional logic, and stochastic search. In *AAAI*, pages 1194–1201, 1996.

[14] R. E. Korf. Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI-97*, pages 700–705, 1997.

[15] R. E. Korf. Invited talk on AAAI citing his own unpublished work on additive databases to solve the 24-Puzzle, 2000.

[16] K. McMillan. *Symbolic Model Checking*. Kluwer Academic Press, 1993.

[17] Y. Murase, H. Matsubara, and Y. Hiraga. Automatic making of sokoban problems. In *Pacific Rim Conference on AI*, 1996.

[18] J. Pearl. *Heuristics*. Addison-Wesley, 1985.

[19] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computing*, 1(1):67–82, 1997.