

# Automated Pattern Database Design

Stefan Edelkamp

University of Dortmund, Germany  
email: stefan.edelkamp@cs.uni-dortmund.de

## Abstract

Pattern databases are dictionaries for heuristic estimates based on state-to-goal distances in state space abstractions. Their effectiveness is sensitive to the selection of the underlying patterns. Especially for multiple and additive pattern databases, a manual selection of pattern sets that lead to good exploration results is involved.

For automating the selection process, greedy bin-packing strategies have been suggested. This paper proposes genetic algorithms to optimize their output. Patterns are encoded as Boolean matrices and optimized using an objective function based on predicting the heuristic search tree size, given the distribution of heuristic values in the abstract state spaces.

To reduce the memory requirements of the databases we apply a construction process based on BDDs. Experiments in heuristic search planning show that the search efforts are significantly reduced.

## Introduction

The ultimate goal for efficient exploration in AI is the automated creation of admissible search heuristics that guide the search process towards the problem goal (Prieditis 1993; Gaschnig 1979; Guida & Somalvico 1979; Holte *et al.* 1996; Prieditis 1993).

By applying state space abstractions, heuristic estimates correspond to solutions in simplified problem spaces. The underlying problem graph is *abstracted* in a form that nodes are contracted or new edges are introduced. Such an abstraction is a state space homomorphism, such that for each path in the concrete state space there is a corresponding path in the abstract state space. This notion of abstraction matches the one used that is used in verification (Clarke, Grumberg, & Long 1994; Ball *et al.* 2001; S. Graf and H. Saidi 1997).

Heuristic search algorithms that explore state space abstractions for each encountered state from scratch and that do not memorize abstract states cannot possibly be better than blind search (Hansson, Mayer, & Valora 1992). To reduce the number of revisits in abstract state one either has to store the abstract state information on-the-fly or compute it prior to the search. A mixed strategy is considered in (Holte

*et al.* 1996) and revisited in (Holte, Grajkowski, & Tanner 2005).

Pattern databases (Culberson & Schaeffer 1998) correspond to complete explorations of abstract state spaces prior to the main heuristic search algorithm. They guide the search process in the concrete state space. The success stories of searching with pattern databases is long and includes first optimal solutions in Rubik's Cube (Korf 1997) and enormous search reductions for optimally solving sliding-tile puzzles (Korf & Felner 2002). The estimates applied for the multiple sequence alignment problem refer to lookup values of alignments of disjoint sequence subsets (Korf *et al.* 2005; Zhou & Hansen 2004). In finding the best parsing of a sentence (Klein & Manning 2003) a pattern database estimate correlates to the cost of completing a partial parse and is derived by simplifying the grammar. TSP with asymmetric costs and precedence constraints has been analyzed by (Hernádvölgyi 2003) using pattern database techniques. For co-operative plan finding in computer games many agents search for chromosome paths but are allowed to help each other to succeed. (Silver 2005) shows how to incorporate memory-based heuristics.

First successful applications of pattern databases for verification domains are due to (Edelkamp & Lluch-Lafuente 2004) (explicit-state model checking), (Qian & Nymeyer 2004) (symbolic model checking), and (Kupferschmid *et al.* 2006) (real-time model checking). In all approaches, even though the construction is automated, patterns were either small-sized or provided manually, such that, in essence, pattern selection remains to be a domain-dependent feature.

In this paper we address the problem of automated pattern selection for the design of pattern databases. We embed our approach in domain-independent action planning, where automated pattern selection is mandatory. In this research area, the use of multiple (Holte *et al.* 2004), often additive databases (Korf & Felner 2002) is frequent. So far, greedy bin packing algorithms have been applied, that terminate with the first partition of patterns (Edelkamp 2001; Haslum, Bonet, & Geffner 2005).

The number of possible patterns is exponential. In case of state space abstractions that introduce don't cares to the state vector, the complexity is exponential in the number of remaining vector entries. In case of general abstractions to the state vector, e.g. by mapping the variable domains

to a smaller set, the number can be much larger. Subsequently, even for only one pattern database we are facing a hard combinatorial optimization problem. If multiple abstraction are considered, the complexity for pattern selection is even worse.

Moreover, the efforts for constructing a pattern database are considerably high, as their sizes are considered to be very large. Especially in combinatorial problems with unknown structure, optimization algorithms adapted from nature such as evolutionary strategies or swarm optimization techniques (Dorigo & Stützle 2005) are often a good choice. Based on the discrete structure of the problem of partitioning atoms into disjoint patterns, we have chosen genetic algorithms (Holland 1975) for improved pattern selection. *Genetic algorithms* are widely used for optimization problems and adapt quite nicely to pattern database search. As the proposed encoding of patterns into chromosomes is accessible for a human, by using this approach, we also expect to uncover insights to important properties of pattern selection.

As they operate at the limit of main memory, a compact and space-efficient representation of pattern databases is essential. Eliminating irrelevant unreferenced states from the abstract state space traversal has been studied by (Zhou & Hansen 2004). We propose a space-efficient representation of pattern databases based on BDDs (Bryant 1985), which – by sharing state vector representations – can lead to exponential memory savings. Instead of compressing an explicit database, we apply a construction process that is symbolic and can lead to pattern databases that are in the order of magnitudes smaller than their represented state sets (Edelkamp 2002).

The paper is structured as follows. First we review pattern databases as applied in optimal heuristic search planning. Then we turn to genetic algorithms for the automated inference of the patterns. Starting with the encoding of patterns into chromosomes, we present the design of genetic operators for the pattern selection problem. Experiments report on improving the mean heuristic value and on reducing the resulting search efforts for a selection of challenging planning domains. Finally, we draw conclusions and indicate further research avenues.

## Planning Pattern Databases

Action planning refers to a world description in logic<sup>1</sup>. A number of atomic propositions, *atoms* for short, describe what can be true or false in each state of the world. By applying operations in a world, we arrive at another world where different atoms might be true or false. Usually, only few atoms are affected by an operator, and most of them remain the same.

Let  $AP$  be the set of atoms. A planning problem (in STRIPS notation) (Fikes & Nilsson 1971) is a finite state space problem  $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ , where  $\mathcal{S} \subseteq 2^{AP}$  is the set of states,  $\mathcal{I} \in \mathcal{S}$  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

<sup>1</sup>For the sake of brevity, the presentation of the paper restricts to propositional planning. However, the design of planning pattern databases is applicable to complex planning formalisms, too.

states into states. We often have that  $\mathcal{G}$  is described by a simple list of atoms. Operators  $O \in \mathcal{O}$  have preconditions  $pre(O)$ , and effects  $(add(O), del(O))$ , where  $pre(O) \subseteq AP$  is the *precondition list* of  $O$ ,  $add(O) \subseteq AP$  is its *add list* and  $del(O) \subseteq AP$  is the *delete list*. Given a state  $S$  with  $pre(O) \subseteq S$  then its successor  $S' = O(S)$  is defined as  $S' = (S \setminus del(O)) \cup add(O)$ .

## Admissible Heuristics in Planning

Admissible heuristics for planning underestimate the shortest path distance of the current state to the goal. They are important to guarantee optimality in heuristic search algorithms like  $A^*$  and  $IDA^*$ . The *max-atom* heuristic (Haslum & Geffner 2000) is an approximation of the optimal cost for solving a relaxed problem in which the delete lists are ignored. Its extension *max-pair* improves the information without losing admissibility, approximating the cost of atom pairs. The heuristic  $h^+$  (Hoffmann & Nebel 2001) is another extension to *max-atom* defined as the length of the shortest plan that solves the relaxed problem with ignored delete lists. The heuristic is admissible but solving relaxed plans is computationally hard.

## Explicit-State Planning Pattern Databases

Explicit-state planning pattern databases as proposed by (Edelkamp 2001; Haslum, Bonet, & Geffner 2005) refer to state space abstraction, where atoms are neglected.

The basic idea for computing a heuristic with pattern databases is to analyze the abstraction of the concrete state space prior to the search (Culberson & Schaeffer 1998). In this abstract state space, a complete backward exploration starting with all goal states is executed to store accurate abstract solution distance information in a lookup table to be used the concrete search process. This forms a state space to be used for abstraction (Knoblock 1994). The *abstraction*  $\mathcal{P}|_R = \langle \mathcal{S}|_R, \mathcal{O}|_R, \mathcal{I}|_R, \mathcal{G}|_R \rangle$  of a STRIPS planning problem  $\mathcal{P} = \langle \mathcal{S}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$  wrt. a set of atoms  $R$  is formally defined by  $\mathcal{S}|_R = \{S \cap R \mid S \in \mathcal{S}\}$ ,  $\mathcal{G}|_R = \{G \cap R \mid G \in \mathcal{G}\}$ ,  $\mathcal{O}|_R = \{O|_R \mid O \in \mathcal{O}\}$ , where  $O|_R$  for  $O \in \mathcal{O}$  is given as  $(pre(O) \cap R, add(O) \cap R, del(O) \cap R)$ . Knoblock’s idea has been extended to automatically inferred *fact groups* (Edelkamp 2001). The approach refers to mutual exclusive atoms and leads to a multi-variate variable encoding of a state (Helmert 2004). E.g., in Blocksworld we have a variable  $on(X, a)$  (where  $X$  represents any block), including atoms  $on(d, a)$ ,  $on(c, a)$ ,  $on(b, a)$  and  $on(a, a)$ . As only one block can lay on top of  $a$  all atoms in the group are mutex. Fact groups are distributed into *patterns*, where each pattern corresponds to an abstraction of the state space. This is achieved by abstract states that represent only a subset of the original atoms - taken from the groups of the chosen pattern. Using this approach each concrete state is mapped to an abstract state. The projection can also be achieved in operators, intersecting the precondition, add and delete list with the pattern. The selection of and partitioning into patterns is a computationally hard problem, critically influencing the quality of the estimate (Haslum, Bonet, & Geffner 2005).

To compute the goal distances in the abstract state space, a backward search starts with the abstract goal state, applying

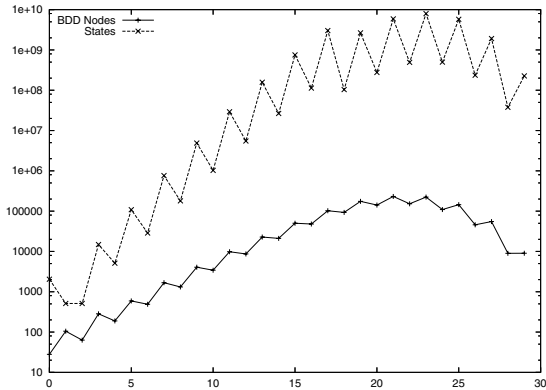


Figure 1: Symbolic Pattern Database Construction.

inverse operators (Culberson & Schaeffer 1998). As inverse operator application is not always applicable, it is possible to apply backward search to the inverse of the state space graph generated in forward search (Edelkamp & Lluich-Lafuente 2004).

For multiple pattern databases (Holte *et al.* 2004), in each abstraction  $i$  and for each state  $S$  we compute estimated costs  $h_i(S)$ . The estimate  $h(S)$  is aimed to be the accumulated cost of the costs in the different abstractions, i.e.,  $h(S) = \sum_i h_i(S)$ . In general we cannot expect that each operator contributes to only one pattern. This implies that the inferred heuristic is no longer admissible. There are two options. First, we may assume that each operator is applicable for only one pattern. Another option considered is to count the abstract problem graph edge weight induced by an operator in only one abstraction. In each BFS level, each zero-cost operator is fired until a fixpoint is reached. If an admissible heuristic is obtained, we call the databases *additive* or *disjoint* (Korf & Felner 2002).

### Symbolic Planning Pattern Databases

The main limitation for applying pattern databases in practice is the restricted amount of (main) memory. Many strategies for leveraging the problem have been proposed. *Secondary memory* was exploited in (Zhou & Hansen 2005). *Symmetries* for reusing pattern databases were already suggested in (Culberson & Schaeffer 1998), while lookups in *dual* pattern databases are performed in (Felner *et al.* 2005). Compressed pattern databases merge cliques in abstract space into super-nodes (Felner *et al.* 2004). A compact space-efficient representation of pattern databases truncates abstract space given upper bounds on the optimal solution length in concrete space (Zhou & Hansen 2004). *On-demand* pattern databases (Felner & Alder 2005) suspend and resume a Secondary A\* backward exploration of the abstract space.

Better scaling strongly favor symbolic pattern databases to explicit pattern databases. Figure 1 display a typical pattern database construction example to illustrate this advantage (in a Blocksworld problem instance). The number of

BDD nodes is by far smaller than the number of explicit states represented. The heuristic value average for this pattern database is 22.1665. The key performance of symbolic pattern databases for being optimized in a GA is a fast construction time. For this example, the pattern database contains  $2.722e+10$  states and the total construction time is 109.15s. Compared to explicit search, we arrive at a pattern database BFS construction performance of about 250 million expanded states per second.

The main effect of using BDDs (Bryant 1985) compared to explicit representations in planning, is an efficient (and often unique) representation of sets of planning states (Jensen, Bryant, & Veloso 2002). BDDs express the *characteristic function* for a set of states (which evaluates to true for a given bit string if and only if it is a member of that set). The characteristic function is identified with the set itself. Transitions are formalized as relations, i.e., as sets of tuples of predecessor and successor states, or, alternatively, as the characteristic function of such sets. By conjoining this formula with any formula describing a set of states and querying for the possible instantiations of the predecessor variable set, we can calculate all states that can be reached in one step from the input set. A\* with BDDs has been introduced by (Edelkamp & Reffel 1998) for domain-dependent search. Alternative settings are found in (Qian & Nymeyer 2003; Hansen, Zhou, & Feng 2002). The objective in symbolic search is that by far more (sometimes exponentially more) new states are generated than BDD nodes are needed to represent them.

Symbolic pattern databases as introduced by (Edelkamp 2002) are pattern databases that have been constructed symbolically for a latter use either in symbolic or explicit heuristic search. Each state set in a shortest path layer is efficiently represented by a corresponding characteristic function. Different to the posterior compression of the state set, the construction itself works on compressed representation, allowing much larger databases to be constructed.

### Automated Pattern Selection

Finding good patterns is not simple, as there are usually an exponential number of possible choices. Manual pattern selection implies that the planning process is problem-dependent. On the other hand, the problem of pattern selection is involved, especially if multiple pattern databases are requested. So far, automated pattern selection is an unresolved challenge. One reason is that the quality of approximation algorithm for pattern partitioning is hard to predict.

Since the number of vector elements can be considerably large, for explicit search one simplifies the problem of finding a suitable partition to *bin-packing*. The goal is to distribute the possible state variables into the bins in such a way that a minimal number of bins is used. A state variable is greedily added to an already existing bin, until the corresponding abstract state space exceeds main memory. Adding entities to the pattern corresponds to a multiplication of the domain sizes of the selected state variables to the expected abstract state size, so that the bin-packing variant that is needed for automated pattern selection is based on

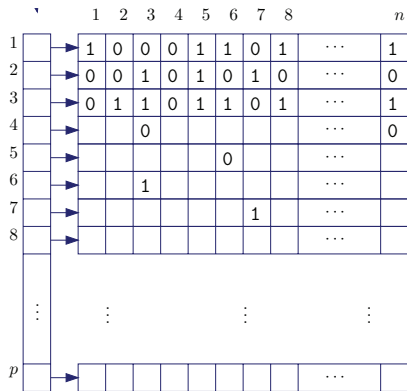


Figure 2: Bitvector Representation of a Chromosome.

multiplying variable domain sizes. Bin-packing is NP complete such that efficient approximations are used.

### Representation

For the implementation of pattern optimization we adapt a genetic algorithm. Patterns are represented as binary chromosome of size  $g \times g$ , where  $g$  is the number of variable domains (fact groups). In the first dimension (row), state variables are indexed, while in the second dimension (column) patterns are selected. A chromosome represents a complete distribution of state variables into multiple PDBs. The number of state variables is  $g$  and the number of disjoint pattern  $p$  is less than or equal to  $g$ .

In Figure 2 we illustrated such a binary chromosome. In Pattern 1, the groups 1,5,6,8 and  $n$  are included, whereas in Pattern 2, the groups 3,5 and 7 are present.

### Initialization

In the initialization phase we generate random chromosomes. The size of chromosome is drawn uniformly in the interval  $[1, g]$ . The default number of generations is 300 and the population size is 5 but can be modified by the user.

We found that the amount of work to find an acceptable partition by performing a randomized assignment of the chromosomes is by far larger as with prior bin packing with no significant advantage within the overall search process. Therefore, we initialized the chromosomes with bin packing. To avoid all chromosomes of the initial population to be identical, we chose a random permutation to the fact groups prior to their automated partitioning into patterns. This leads to comparable good but different distributions of groups into patterns and a feasible initial population for the genetic algorithm.

### Recombination

The motivation of *recombination* of two parent chromosomes is the hope, that the good properties of the one combines well with the good properties of the other. The simplest technique is the *n-point-crossover*. For our representation a 2-point crossover in the dimension of the pattern is ad-

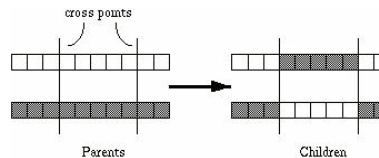


Figure 3: 2-Point Crossover.

equate. This means, that the parent chromosomes exchange parts of their patterns. If the two parents have a different number of patterns, so do the two children.

### Mutation

During *mutation*, chromosome bits are flipped with a small probability. For our case this operator is an important, as it allows to add or delete groups in patterns. We also included a mutation operator which adds and deletes entire patterns. In the bin packing analogy of multiple pattern partition, adding a pattern corresponds to opening a bin, and deleting a pattern corresponds to closing a bin.

### Selection

During *selection* an enlarged population (as produced using recombination) is truncated back based on the fitness value(s) to its original size. The normalized fitness evaluation for the population is interpreted as a distribution function, which governs the selection process for the next population. Chromosomes with a better fitness are chosen with higher probability.

### Objective Functions

The objective function plays a central role in a genetic algorithm. It defines a fitness to determine the evolutionary strength of chromosomes. The construction of an objective function is often difficult, like in our case, where the conditions for *good* patterns are hardly accessible.

A fundamental question concerns the relationship between the size of the pattern database for a heuristic, and the number of nodes expanded when the heuristic is used to guide the search. (Korf 1997) gives some more insights in the performance of pattern databases. As one result of his research, we can characterize the effectiveness of an admissible heuristic function  $h$  by its expected value  $\bar{h}$  over the problem space. If the heuristic value of every state were equal to its expected value  $\bar{h}$ , then a search to depth  $d$  would be equivalent to searching to depth  $d - \bar{h}$  without a heuristic, since the  $f$ -value for every state would be its depth plus  $\bar{h}$ . This means that in most search spaces, a linear gain in  $\bar{h}$  corresponds to an exponential gain in the search efforts.

For the pattern selection problem we conclude that the higher the average heuristic values, the better the corresponding pattern database. As a consequence we compute the mean heuristic value for each database and More formally, the average estimate of a singular pattern database

with state space abstraction  $\phi$  is

$$\bar{h} = \sum_{h=0}^{\max} \frac{h \cdot |\{\phi(u) \in PDB \mid \text{entry}(u) = h\}|}{|PDB|},$$

where *hash* is the entry stored for  $u$ . For multiple pattern databases, we compute the mean heuristic value for each of the database and add these for a chromosome. More formally, if  $PDB_i$  is the  $i$ -th pattern database,  $i \in \{1, \dots, k\}$ , then the *fitness* of a chromosome is

$$\text{fitness}(c) = \sum_{i=1}^k \sum_{h_i=0}^{\max_i} \frac{h_i \cdot |\{\phi_i(u) \in PDB_i \mid \text{entry}_i(u) = h_i\}|}{|PDB_i|}.$$

The distribution of heuristic estimates  $\text{entry}_i$  for  $k = 3$  PDBs is shown bar diagrams in Figure 4.

### A Note on Search Tree Prediction

Applying the mean heuristic value for the fitness correlates to the search tree prediction formula developed by (Korf, Reid, & Edelkamp 2001). It approximates  $E(N, c, P)$ , the expected number of nodes expanded by IDA\* up to depth  $c$ , given a problem-space tree with  $N_i$  nodes of cost  $i$ , with a heuristic characterized by the equilibrium distribution  $P$ . This is the number of nodes expanded by an iteration of IDA\* with cost threshold  $c$ , in the worst case. It denotes that in the limit of large  $c$ , we expect  $E(N, c, P) = \sum_{i=0}^c N_i P(c - i)$  nodes to be generated. The formula has already been used for the analysis of pattern database performance (Holte & Hernadvogyi 1999) and to explain anomalies that many small pattern databases are more effective than few big ones of same total size (Holte *et al.* 2004).

The number of nodes in the brute-force tree in depth  $d$  is roughly  $N_d = b^d$ , assuming a branching factor  $b$ . However, to determine the growth of brute-force search tree for general problems as addressed in action planning is not immediate. Problems of predicting search tree growth in non-regular domains are e.g. reported in (Junghanns 1999). Hence, we prefer  $\bar{h}$  to be used for evaluation, where the distribution of heuristic values is exactly determined by the pattern database-

## Experiments

For implementing genetic algorithms we chose the library GALib<sup>2</sup> (Wall 2005). Only the genetic operations recombination, mutation and selection are to be configured. These configurations can be implemented in an additional external file without modifying the existing source code.

In fact, we experimented only selection and mutation, recombination was turned off completely. This made the GA work like *randomized local search*. The mutation operator adds and deletes groups to an existing pattern and allows to extend patterns in a disjoint partition. If the expected size of a PDB in a chromosome becomes too large, i.e., the multiplication of group sizes exceeds the given size threshold, no PDB is constructed and an extreme fitness value is returned.

We draw experiments on a 3 GHz Linux PC. Time in CPU seconds was limited to 1,800, space was limited to 1 GB.

<sup>2</sup><http://lancet.mit.edu/galib-2.4/>

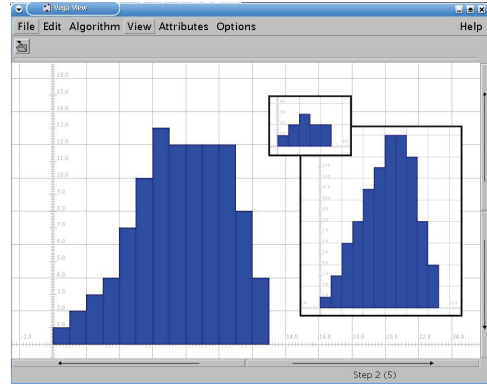


Figure 4: 3 Distributions of Heuristic Values in PDBs.

## Explicit Pattern Databases

In a first test of the pattern databases we distinguish between Greedy-Bin-Packing and optimized GA-variants with different parameters. Before we turn to a symbolic representation, the aim is to learn about the parameter setting of GAs in planning domain, as GAs are known to be sensitive to this.

Based on the use of genetic algorithm the determination of the heuristic values is non-deterministic, so that we use several runs using the mean of the heuristic value as a comparison guideline. As we will see, choosing an inappropriate parameters can avoid plan finding.

As the iterated tests are involved, we depict the change in the exploration efficiencies for some interesting domains only. The denotation of the horizontal axis denotes the choice of parameters as follows.

Label	Meaning
5	Bin Packing: $2^5$ abstract states
5,50,5	GA: $2^5$ abstract states, 50 epochs, 5 genes
...	

In Figure 5 the precomputation time, the search time and the number of expansions for different GA parameters settings are shown. As we can see, even when including the entire construction time for all pattern databases, with GAs there is some substantial gain compared to ordinary bin packing. As a general finding we observe that for a better guidance, longer construction time is needed. In some cases, however, the construction time is so large that there is no gain for exploration. In the *Driverlog* domain (Fig. 6) the influence is present, but less apparent.

Automatically selecting the GA parameters in such a way that the pre-computation efforts are acceptable with number of expansions that is small enough is a challenge. However, we found some parameters that look more promising than others. In the next set of experiments, we try to scale-up the approach by using BDDs.

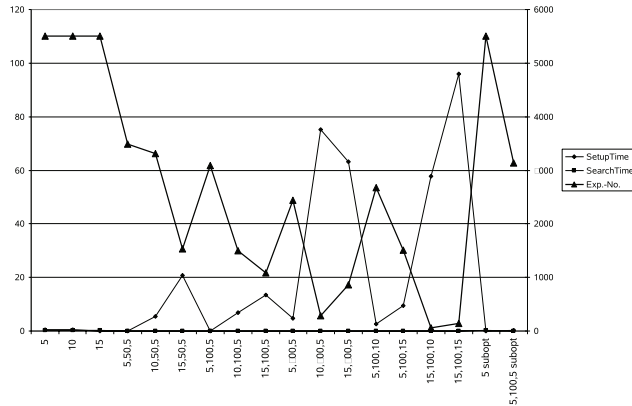


Figure 5: Explicit Pattern Databases in Logistics.

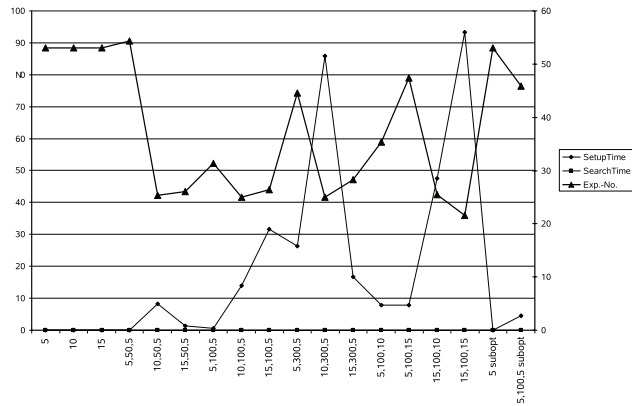


Figure 6: Explicit Pattern Databases in Driverlog.

### Symbolic Pattern Databases

For experimenting with symbolic pattern databases, we choose various problems from IPC benchmarks. For domains from IPC-4 good exploration results of symbolic pattern databases are already known (Edelkamp 2005). For the construction of symbolic pattern databases we choose a population size of 5, and a number of 20 epocs (resulting in at most 100 pattern databases to be evaluated; some of them were eliminated due to size and state variable constraints). The random seed was fixed for all experiments.

The initial population the GA is based on randomizing the order of groups in the state vector. Therefore bin packing can yield better results than GA search in some cases. We apply symbolic A\* search with full duplicate elimination.

In Fig. 1 our results of comparing greedy bin-packing with GA pattern selection are shown. Even though the number of states can largely differ from the number of BDD nodes, for the automated selection of pattern databases, the abstract state space size limit ( $2^l$ ) is taken as an additional parameter in both cases. The searching time  $t_s$  is compared

to the total running time  $t$  which includes the parsing of the grounded domain. The time for instantiation is not counted. The setup times  $t - t_s$  for the genetic algorithm includes the time for computing *all* pattern databases during the optimization process. The additional time for pattern optimization contributes to the gain in the quality of the heuristic estimate, measured in  $\bar{h}$ , the mean heuristic estimate of the first (greedy bin-packing) or best surviving (genetic algorithm) pattern. To judge the quality of the heuristic, we provide the solution lengths. While greedy bin packing often runs out of memory, improved pattern selection with GAs scales better.

Using the mean heuristic estimate is not the only choice for designing a fitness function. We have also experimented with a derivate of  $\bar{h}$  not based on the number of states that share the same heuristic value, but on the number of BDD nodes to represent them. The objective was driven by the assumption that as all operations are based on DDs, so the objective function should as well. The results were throughout weaker than the presented ones, so we omitted them from the presentation.

### Conclusion

Genetic algorithms are one important representative of randomized optimization algorithms. There are different attempts to unify planning and evolutionary computing (Muslea 1997; Westerberg & Levine 2001; Spector 1994; Brie & Morignot 2005). All are concerned about plan finding or plan refinement and none of them regarded the computationally hard problem of pattern selection. The only other planner that considers plan refinement in heuristic search planning is *GenPlan* (Westerberg & Levine 2001). The application of genetic algorithms for improved search has also been reported in model checking (Godefroid & Khurshid 2004).

We have seen a flexible attempt to optimize pattern database exploration prior to the overall search process using genetic algorithms. The approach optimizes the partition of multi-variate variables into disjoint patterns. We showed that pattern optimization is essential and GA optimization is appropriate. To the author's knowledge, optimization of patterns has not been considered before<sup>3</sup>.

Given the time and space efficiency of symbolic search, we could construct and evaluate many large pattern databases in a limited amount of time. Driven by the theory of search tree prediction we chose the mean heuristic value as a fitness function. For the evaluation of each chromosome we have computed an entire set of pattern databases. Faster construction times of larger databases favors symbolic construction. but the exploration gains obtained in the experiments for symbolic and explicit-state pattern database construction are promising. The encoding of pattern variables in a matrix allows experts to reason on the structure of good patterns. Based on the established patterns, he can enlarge

<sup>3</sup>There has been some unpublished work on the automated generation of pattern databases by Holte and Hernadvolgyi. Their approach enumerates all possible pattern partitions that are not subsumed by already considered ones.

problem	$2^l$	Greedy Bin Packing					GA-Optimization				
		length	iterations	$\bar{h}$	$t_s$	$t$	length	iterations	$\bar{h}$	$t_s$	$t$
logistics-4-1	10	19	63	9.28	0.39	0.77	19	63	8.57	0.37	0.79
logistics-6-1	20	14	42	21.9	0.39	0.77	14	39	20.34	0.30	1.01
logistics-8-1	20	44	166	26.32	11.98	19.92	44	44	29.51	5.7	1.42
logistics-10-1	30	-	-	-	-	-	42	351	38.82	33.78	85.69
logistics-12-1	30	-	-	-	-	-	69	826	51.49	138.02	498.76
blocks-9-0	30	30	79	8.86	0.47	52.51	30	358	20.03	8.89	19.4
blocks-10-0	40	-	-	-	-	-	34	692	25.15	8.53	34.94
blocks-11-0	40	-	-	-	-	-	32	1,219	24.20	49.30	58.44
blocks-12-0	40	-	-	-	-	-	34	942	25.36	101.95	104.55
zeno-2	10	6	6	6.37	0.17	0.55	6	14	3.83	0.19	0.57
zeno-4	10	8	19	5.74	0.27	0.73	8	14	3.83	0.19	0.57
zeno-6	20	11	24	6.6	0.64	1.21	11	14	8.58	0.58	1.51
zeno-10	25	-	-	-	-	-	22	23	15.7	43.12	190.56
zeno-11	25	-	-	-	-	-	14	37	15.06	15.11	833.16
driverlog-9	25	22	109	12.9	86.76	87.59	22	107	15.3	52.46	72.25
driverlog-11	25	-	-	-	-	-	19	110	10.67	34.48	44.60
driverlog-13	35	-	-	-	-	-	26	143	13.7	553.01	778.03
openstack-1	20	23	96	3.71	0.51	1.29	23	116	5.06	0.60	3.04
openstack-3	20	23	96	3.71	0.51	1.29	23	110	5.40	0.60	3.02
openstack-6	30	20	65	4.99	70.38	96.17	20	39	5.44	52.42	216.19
openstack-7	30	-	-	-	-	-	20	38	6.88	31.05	484.19

Table 1: Symbolic A\* Search with and without GA Optimization.

his knowledge on the domain. Our implementation features an XML-interface for pattern fined-tuning.

Future work will address *learning* of fitness functions for using it in a genetic algorithm in larger instances, with a genetic algorithm running on the same domain in smaller instances. One bootstrapping option is to accumulate patterns optimized for smaller abstractions to larger one. As genetic algorithms are randomized, the search efficiency varies significantly on different runs. This suggests to apply *randomized local search with random restarts*, as in domain-dependent single-agent search (Junghanns 1999).

## References

- Ball, T.; Majumdar, R.; Millstein, T. D.; and Rajamani, S. K. 2001. Automatic predicate abstraction of c programs. In *SIGPLAN Conference on Programming Language Design and Implementation*, 203–213.
- Brie, A. H., and Morignot, P. 2005. Genetic planning using variable length chromosomes. In *ICAPS*, 320–329.
- Bryant, R. E. 1985. Symbolic manipulation of boolean functions using a graphical representation. In *ACM/IEEE Design Automation Conference*, 688–694.
- Clarke, E. M.; Grumberg, O.; and Long, D. 1994. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems* 16(5):1512–1542.
- Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(4):318–334.
- Dorigo, M., and Stützle, T. 2005. *Ant Colony Optimization*. MIT Press.
- Edelkamp, S., and Lluch-Lafuente, A. 2004. Abstraction in directed model checking. In *ICAPS-Workshop on Connecting Planning Theory with Practice*.
- Edelkamp, S., and Reffel, F. 1998. OBDDs in heuristic search. In *KI*, 81–92.
- Edelkamp, S. 2001. Planning with pattern databases. In *ECP*, 13–24.
- Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *AIPS*, 274–293.
- Edelkamp, S. 2005. External symbolic heuristic search with pattern databases. In *ICAPS*, 51–60.
- Felner, A., and Alder, A. 2005. Solving the 24 puzzle with instance dependent pattern databases. In *SARA*, 248–260.
- Felner, A.; Meshulam, R.; Holte, R. C.; and Korf, R. E. 2004. Compressing pattern databases. In *AAAI*, 638–643.
- Felner, A.; Zahavi, U.; Schaeffer, J.; and Holte, R. 2005. Dual lookups in pattern databases. In *IJCAI*, 103–108.
- Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.
- Gaschnig, J. 1979. *Performance measurement and analysis of certain search algorithms*. Ph.D. Dissertation, Carnegie-Mellon University.
- Godefroid, P., and Khurshid, S. 2004. Exploring very large state spaces using genetic algorithms. *STTT* 6(2):117–127.
- Guida, G., and Somalvico, M. 1979. A method for computing heuristics in problem solving. *Information and Computation* 19:251–259.

- Hansen, E. A.; Zhou, R.; and Feng, Z. 2002. Symbolic heuristic search using decision diagrams. In *SARA*.
- Hansson, O.; Mayer, A.; and Valtora, M. 1992. A new result on the complexity of heuristic estimates for the A\* algorithm (research note). *Artificial Intelligence* 55:129–143.
- Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. 140–149.
- Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *AAAI*, 1163–1168.
- Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*, 161–170.
- Hernádvölgyi, I. T. 2003. *Automatically Generated Lower Bounds for Search*. Ph.D. Dissertation, University of Ottawa.
- Hoffmann, J., and Nebel, B. 2001. Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.
- Holland, J. 1975. *Adaption in Natural and Artificial Systems*. Ph.D. Dissertation, University of Michigan.
- Holte, R. C., and Hernádvölgyi, I. T. 1999. A space-time tradeoff for memory-based heuristics. In *AAAI*.
- Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and Donald, A. J. 1996. Hierarchical A\*: Searching abstraction hierarchies. In *AAAI*, 530–535.
- Holte, R. C.; Newton, J.; Felner, A.; Meshulam, R.; and Furcy, D. 2004. Multiple pattern databases. In *ICAPS*, 122–131.
- Holte, R. C.; Grajkowski, J.; and Tanner, B. 2005. Hierarchical heuristic search revisited. In *SARA*, 121–133.
- Jensen, R. M.; Bryant, R. E.; and Veloso, M. M. 2002. SetA\*: An efficient BDD-based heuristic search algorithm. In *AAAI*, 668–673.
- Junghanns, A. 1999. *Pushing the Limits: New Developments in Single-Agent Search*. Ph.D. Dissertation, University of Alberta.
- Klein, D., and Manning, C. 2003. A\* parsing: Fast exact Viterbi parse selection. In *Human Language Technology Conference of North American Chapter of the Association for Computational Linguistics*.
- Knoblock, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence* 68(2):243–302.
- Korf, R. E., and Felner, A. 2002. *Chips Challenging Champions: Games, Computers and Artificial Intelligence*. Elsevier. chapter Disjoint Pattern Database Heuristics, 13–26.
- Korf, R. E.; Zhang, W.; Thayer, I.; and Hohwald, H. 2005. Frontier search. *Journal of the ACM* 52(5):715–748.
- 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. 1997. Finding optimal solutions to Rubik’s Cube using pattern databases. In *AAAI*, 700–705.
- Kupferschmid, S.; Hoffmann, J.; Dierks, H.; and Behrmann, G. 2006. Adapting an ai planning heuristic for directed model checking. In *SPIN*.
- Muslea, I. 1997. A general-purpose AI planning system based on genetic programming. In *Genetic Programming Conference (Late Breaking Papers)*, 157–164.
- Prieditis, A. 1993. Machine discovery of admissible heuristics. *Machine Learning* 12:117–142.
- Qian, K., and Nymeyer, A. 2003. Heuristic search algorithms based on symbolic data structures. In *ACAI*, 966–979.
- Qian, K., and Nymeyer, A. 2004. Guided invariant model checking based on abstraction and symbolic pattern databases. In *TACAS*, 497–511.
- S. Graf and H. Saidi. 1997. Construction of abstract state graphs with PVS. In *Computer Aided Verification (CAV)*, 72–83.
- Silver, D. 2005. Cooperative pathfinding. In *Conference on Artificial Intelligence and Interactive Digital Entertainment*, 117–122.
- Spector, L. 1994. Genetic programming and AI planning systems. In *AAAI*, 1329–1334.
- Wall, M. 2005. *GAlib – A C++ Library of Genetic Algorithm Components*. Massachusetts Institute of Technology.
- Westerberg, H., and Levine, J. 2001. Optimising plans using genetic programming. In *ECP*, Poster.
- Zhou, R., and Hansen, E. 2004. Space-efficient memory-based heuristics. In *AAAI*, 677–682.
- Zhou, R., and Hansen, E. 2005. External-memory pattern databases using structured duplicate detection. In *AAAI*.