Planning-PDBs Planner

Ionut Moraru[1], **Stefan Edelkamp**[1], **Moises Martinez**[1], **Santiago Franco**[2]
[1] Computer Science Department, Kings College London, United Kingdom
[2] School of Computing and Engineering, University of Huddersfield, UK
moises.martinez@kcl.ac.uk, ionut.moraru@kcl.ac.uk, stefan.edelkamp@kcl.ac.uk
s.franco@hud.ac.uk,

Abstract

The automated construction of search heuristics is a long-term aim in artificial intelligence, while pattern databases (PDBs) serve as memory-based abstraction heuristics generated prior to the search to reduce computational efforts.

In the competing IPC 2018 system Planning-PDB we have taken a state of the art planner (Franco et al., 2017) and augmented a part of its pattern selection process, namely the bin packing subroutine, to improve the quality of the patterns.

Introduction

The automated generation of search heuristics is one of the long time goals of AI and goes back to early work of Gaschnig (1979), Pearl (1985), and Preditis (1993). Heuristics refer to state-space abstractions, and for lower-bound estimates each path in the original state space has to map to a corresponding —possibly shorter one— in the abstract state space.

Searching with heuristics based on abstractions has yield many positive results (e.g, Holte, Grajkowski and Tanner, 2005), but also showed one major drawback: in the worst case, the time used in searching the abstract state spaces may exceed the time saved for searching the overall search space (Valtorta, 1984).

With the advent of pattern databases (PDBs), for the computational effort in searching the abstract state spaces is spent prior to the search. In concrete space search only lookups have to be executed. This research initiated by Culberson and Schaeffer (1998) let to a revival of the interest in abstraction heuristics. Initial results in sliding-tile puzzles quickly carried over to a number of combinatorial search domains, and helped to solve random instances of the Rubik's cube optimally for the very first time (Korf, 1997).

The complete exploration of the abstract state space yields a lower bound, and can be extended to include action cost. The combination of several databases into one, however, is tricky (Haslum, Botea, Helmert, Bonet, and Koenig, 2007). While the maximum of two PDBs is always a lower bound, the sum is usually not. Disjoint PDBs (Felner, Korf, 2004) showed that with a clever selection of disjoint patterns admissibility can be preserved. Holte et al. (2004) showed that in several cases, the combination of many small PDBs can outperform a large one.

The use of PDBs in AI planning was pioneered by Edelkamp (2001). Initially, the notion of a pattern as a selection of tiles (or a set of labels in Rubik's cube) has been generalized to state-spaces in vector notation (Holte and Herndvlgyi, 1999). Most AI planning problems can be translated into a state-space of finite domain variables (Helmert, 2004), so that a selection of variables leads to projections of pre- and postconditions of actions.

The main limitation of PDBs is the amount of memory needed, as during their construction process, the abstract state space may prove to be too large for the available resources in RAM. To deal with these large memory requirements, PDBs have been extended to support symbolic search, which succinctly represents state sets compactly as binary decision diagrams (Edelkamp, 2002). Still, the automated selection of the most informative patterns remains a combinatorial challenge. There is an exponential number of variable sets to choose from, not counting alternative projection and cost partitioning methods (Karpas, Katz, and Markovitch, 2011; Pommerening, 2017) in distributing the cost of actions over different abstract search spaces.

Hence, the automated selection of possibly additive heuristics requires approximations. Hill-climbing strategies have been proposed (Haslum, Botea, Helmert, Bonet, and Koenig, 2007), where a PDB on $(n-1)$ variables serves as an estimate for a PDB on n variables, as well as more general optimization schemes such as genetic programming (Edelkamp, 2007; Franco et al. 2017). They are using the bitvector of variables selected as genes and the quality of the PDB as the fitness function.

The quality of the PDBs – in terms of the returned lower bound on the solution cost – can only be estimated. Usually, this involves first generating the PDB and then evaluating it, taking the average heuristic estimate (Edelkamp, 2001) or a weighted sum (Korf, 1997), or sampling the state space (Franco et al., 2017).

For participating in the international planning competition 2018, we have started our implementation efforts by taking a state-of-the-art planner (Franco et al, 2017) and came up with new ways to improve the automated pattern selection process.

We first briefly define the setting of cost-optimal action

planning and give a characterization of the pattern database selection problem. Afterwards, we move our focus to the genetic encoding of the problem and what fitness function we have used for determining the PDBs. We concentrate on the pattern database equivalent of the NP-hard bin packing problem (BPP) for choosing sets of informative pattern database that are known to respect the limits in main memory. As solutions for the the BPP, we consider several different algorithmic approaches.

Problem Statement

Even though many planning domains are *lifted* int the textual PDDL input, in the formal description, we start with grounded planning problems in SAS$^+$ representation, as usually generated by a planner in its static analysis stage. A *sequential planning task* $\mathcal{P}$, is characterized as a quadruple consisting of finite-domain state variables $\mathcal{V}$, initial state $\mathcal{I}$, goal condition $\mathcal{G}$, and operators (grounded actions) $\mathcal{O}$ with pre- and postconditions $pre(o)$, $eff(o)$, $o \in \mathcal{O}$. For the sake of simplicity, we consider all conditions as conjunctions of state variable assignments (the planner itself deals with a much larger PDDL fragment, including ADL constructs and conditional effects). Each operator o is associated with a cost $c(o)$, whose sum has to be minimized over all plans that lead from the initial state to one of the goals.

The state-space $\mathcal{S}$ induced by such planning task can be viewed as a subset of the cross product of the domains representing the state variables, i.e., for $\mathcal{V} = \{v_1, \ldots, v_n\}$ we have $\mathcal{S} \subseteq dom(v_1) \times \ldots \times dom(v_n)$, where $dom(v)$ is the finite domain of possible value assignments to v. The set of reachable states is generated on-the-fly, starting with the initial state via applying the operators.

A *heuristic* h is a mapping of the set of states $\mathcal{S}$ to the positive reals $R_{\geq 0}$. Usually, we have $h(s) = 0$, if and only if a state s satisfies the goal condition. The heuristic is called *admissible*, if $h(s)$ is a lower bound of the cost of all goal-reaching plans starting at s. Two heuristics h_1 and h_2 are *additive*, if h defined by $h(s) = h_1(s) + h_2(s)$ for all $s \in \mathcal{S}$, is admissible. It is *consistent* (the usual case for PDBs), if for operator o from s to s' we have $h(s') - h(s) + c(o) \geq 0$. For admissible heuristics, search algorithms like A* (Hart et al, 1968) will return optimal plans. Moreover, if h is also consistent, no reopening takes place.

A *state space abstraction* ϕ is a mapping from states in the original state space S to the states in the abstract state space $\mathcal{A}$. As the problem is implicitly given, the abstraction is generated by abstracting the operators, the initial state and the goal conditions. Plans in the original space have counterparts in the abstract space, but not vice versa. A *pattern database* is a lookup table that for each abstract state a provides the (minimal) cost value from a to the set of abstract goal states. This value, in turn, is a lower bound for reaching the goal of the state that is mapped to a in the original state space.

PDBs are generated in a backwards enumeration of the abstract state space starting with the abstract goal description. As this assumes that operators to be reversible, in explicit search, the set of reachable states may have to generated beforehand. There are options to invert planning operators, but for an underspecified goal state, backward search can be cumbersome. In symbolic search with BDDs going backward is much more natural.

Showing that PDBs yield heuristics that are both consistent and admissible is rather trivial (Edelkamp, 2000; Haslum et al., 2005), as we construct them on an fully generated abstracted state-space. It has also been shown that for planning PDBs the sum of heuristic values obtained via *projection* to a disjoint variable set is admissible (Edelkamp, 2001). The projection of state variables induces a projection of operators and is a special case of what is called *0/1 partitioning*. In a 0/1 partitioning, the operators in abstract space are mapped to either 0 or $c(o)$, so that on operator cannot contribute to more than one PDB. There are complex versions of *cost partitioning*, that distribute fractional cost of operators o, still adding to at most $c(o)$, to several abstract state spaces (Pommerening, 2017).

For ease of notation, we identify a pattern database with its abstraction function ϕ. As we want to optimize pattern selection via a genetic algorithms, the *fitness* f of a pattern database ϕ (and represented as a set of pairs $(a, h(a)) \in \phi$) is the average heuristic estimate

$$f(\phi) = \sum_{(a,h(a)) \in \phi} h(a)/|\phi|,$$

where $|\phi|$ denotes the size of the pattern databases denoted by ϕ.

The storage of one PDB in explicit search is a (perfect) hash table, while in symbolic search all abstract states of a certain heuristic value are kept succinctly in form of a BDD.

For several PDBs $\phi_1, \ldots, \phi_l$ and cost partitioning function γ, the values are added up. We have

$$f(\phi_1, \ldots, \phi_l) = \sum_{i=1}^{l} \sum_{(a,h_{i,\gamma}(a)) \in \phi_i} h_{i,\gamma}(a)/|\phi_i|.$$

As each PDB consumes a significant amount of space, the *pattern selection problem* is to find a selection of pattern databases that fit into main memory, and optimizes f.

One very simple PDB called the perimeter, is an unabstracted backward search until resources are exhausted, setting the value of all unreached abstract space to the maximum perimeter reached. In several simpler planning task, the perimeter PDB search already solved the planning problem. The unexpected good results of the otherwise blind bidirectional symbolic baseline planner illustrates the power of this search component.

Genetic Algorithms for Pattern Selection

A *genetic algorithm* (Holland, 1975) is a general optimization method, and has been identified, e.g., by Schwefel as a member of the class defined as *evolutionary strategies*. It refers to the recombination, selection, and mutation of *genes* (states in a state-space) to optimize the *fitness* (alias objective) function.

In a genetic algorithm (GA), a population of candidate solutions to an optimization problem is sequentially evolved

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
PDB_1	0	0	0	0	0	1	0	1
PDB_2	0	1	0	0	0	0	1	0
PDB_3	0	1	0	0	1	0	0	0
PDB_4	0	0	1	1	0	0	0	0

Table 1: An example set of pattern (database) variable selection, forming a 0/1 GA bitstring providing one solution of the bin packing problem.

to generate a better performing population of solutions, by mimicing the process of evolution. Each candidate solution has a set of properties which can be mutated and recombined. Traditionally, candidate solutions are represented as 0/1 bitstrings, but there are other evolutionary strategies that work on real-valued state vectors.

For the Pattern-PDB competing planner in IPC 2018, a binary representation of the genes was sufficient. An early approach for the automated selection of (projection) PDB variables by Edelkamp (2007) employed a GA with genes representing state-space variable patterns in the form of a 0/1 matrix G, where $G_{i,j}$ denotes that state variable i is chosen in PDB j (see Table 1). Besides flipping and setting bits, mutations may also add and delete PDBs in the set.

In this setting, in ordert to evaluate the fitness function, the corresponding PDBs has to be generated – a time-consuming operation, which nevertheless pays off in most cases. The approach has been refined by sampling techniques (Lelis et al., 2016; Franco et al. 2017), which is now available in the fast-downward planning system (Helmert, 2006).

The PDBs corresponding to the bitvectors in the GA have to fit into main memory, so we have to restrict the generation of offsprings to the ones that represent a set of PDB that fit into RAM. If time becomes an issue we also have to stop evolving patterns to invoke the overall search (in our case progressing explicit states) eventually.

An alternative for pattern selection, which is also used as a subroutine within the GA, is to apply bin packing.

Bin Packing for Pattern Selection

The bin packing problem (BPP) is one of the first problems shown to be NP-hard (Garey and Johnson, 1979). Given objects of integer size $a_1, \ldots, a_n$ and maximum bin size C, the problem is to find the minimum number of bins k so that the established mapping $f : \{1, \ldots, n\} \to \{1, \ldots, k\}$ of objects to bins maintains $\sum_{f(a)=i} a \leq C$ for all $i \leq k$. The problem is NP-hard in general, but there are good approximation strategies such as first-fit and best-fit decreasing (being at most 11/9 off the optimal solution (Dósa, 2007). The NP-reduction from number partitioning (where a set of objects must be split into two equally-sized parts) fits into one sentence: if $\sum_{i=1}^{n} a_i$ is odd, then number partitioning is not solvable; and if $\sum_{i=1}^{n} a_i$ is even, then the objects have a perfect fit into two bins of size $\sum_{i=1}^{n} a_i/2$.

In the PDBs selection process, however, the definition of the BPP is slightly different. We estimate the size of the PDB by computing the product (not the sum) of the variable domain sizes, so that for a maximum bin capacity M imposed by the available memory, we find the minimum number of bins k, so that the established mapping f of objects to bins maintains $\prod_{f(a)=i} a \leq M$ for all $i \leq k$. By taking the logs on both sides, we are back to sums, but the sizes become fractional. In this case, $\prod_{f(a)=i}$ is an upper bound on the number of abstract states needed. This is true for both explicit and symbolic pattern databases.

Taking the product of variable domains is a coarse upper bound. In some domains, the abstract state spaces are much smaller. Bin packing chooses the memory bound on each individual PDB, instead of limiting their sum. Moreover, for symbolic search, the correlation between the cross product of the domains and the memory needs is rather weak. However, by its simplicity and effectiveness this form of bin packing currently is the state-of-the-art for PDB construction in planning. Generalizations to other variable abstractions and cost partitionings are possible.

As bin packing is *pseudo-polynomial*, small integer weights in the input lead to a polynomial-time dynamic programming algorithm (Garey and Johnson, 1979). Moreover, for this case, there are effective *bin completion* strategies that are featured in depth-first branch-and-bound algorithms for bin packing (Korf 2002, 2003). The key property that makes the bin completion efficient is a dominance condition on the feasible completions of a bin. The algorithm that partitions the objects into included, excluded and remaining ones relies on perfectly fitting elements and forced assignments, and thus, on integer values for a. In the given setting of real-valued object sizes that are multiplied (or logarithms that are added) this might be less often the case.

By limiting the amount of optimization time for each BPP, we do not insist on optimal solutions, but we want fast approximation strategies that are close-to-optimal. Recall that suboptimal solutions to the BPP do not mean suboptimal solutions to the planning problem. In fact, *all* solutions to the BPP lead to admissible heuristics and therefore optimal plans.

For the sake of generality, we strive for solutions to the problem, which do not include problem-specific knowledge but still work efficiently. Using a general framework also enables us to participate in future solver developments. Therefore, we decided for the moment to focus on the First-Fit on-line algorithm[1].

There are many approximation algorithms for bin packing. First-fit increasing is a fast on-line approximation algorithm that first sorts the objects according to their sizes and, then, starts placing the objects into the bins, putting an object to the first bin it fits into. In terms of planning, the variables are sorted by the size of their domains in a decreasing order. Next, the *biggest* variable is chosen and packed at the same bin with the rest of variables which are related to it if there are space enough in the bin. This process is repeated until all variables are processed.

[1]Even though in principle, BPP can be specified as a PDDL planning problem on its own, initial experiments of solving such a specification with off-the-shelf-planners were not promising.

For the sake of completeness, we provide its trivial implementation.

```
int firstfit() {
  int c=0; double bin[n];
  for (int i=0;i<n;i++) bin[i] = C;
  for (int i=0;i<n;i++)
    for (int j=0;j<n;j++)
      if (bin[j]-a[i] >=0) {
        bin[j]-=a[i]; break; }
  for(int i=0;i<n;i++)
    if (bin[i] != C) c++;
  return c;
}
```

International Planning Competition 2018

For the International Planning Competition 2018, we have worked taken a planner that we considered to be state of the art (Franco et al, 2017), and have tried to improve it adding different Bin Packing algorithms in the process of pattern selection. Because of time constraints, we could not add or test different solutions to the BPP (based on Monte-Carlo tree search of Constraint Programming) inside of the planner and check reliably if it worked better than the on-line algorithms, so we entered the competition without. However, that is going to be added for future work.

Results

Following the announcement of the results at ICAPS 2018, we have started analyzing the results, first by reviewing the competition logs, which were made available on the competitions website[2], and afterwards by running different configurations on the competition benchmarks (see Table 2) on our cluster that utilized Intel Xeon E5-2660 V4 with 2.00GHz processors. We compare this version with MinizincPDBs (Moraru et al, 2018), one that solves the pattern selection problem by encoding it into a Constraint Programming problem. For our experiments, we tested on four versions, the first being the competition version, using 900 seconds for optimization and having a perimeter heuristic, then variations with 300 seconds of optimization and with no perimeter.

Looking at Table 2, we deduce that there was little parameter optimization we could have done for this competition, even though an Oracle planner could have combined our different version to manage to tie the winner of the competition. The perimeter heuristic is vital for a domain like *Petri-net-alignment*, while the difference in the time allocated for the GA isn't as important as we were expecting.

Concluding Remarks

This years results are very satisfactory for us on a whole. Our planner was only four instances away from the winning planner, and had some interesting results in comparison with it's *sister* planner, Complementary. From an analysis of the competition logs, the best performing 5 out of 6 planners were based on our inventions of pattern database abstraction heuristics and/or symbolic search, while the winning portfolio planner used systems based on such planners for more than half of its successful results. This reassures us that in our research we are working on the edge of best performing cost-optimal planning techniques and that more research work on this can lead to a very well rounded domain independent cost-optimal planner.

[2]https://ipc2018-classical.bitbucket.io/#planners

Acknowledgments

We would like to thank the organizers of this years competition, namely Florian Pommerening and Alvaro Torralba, as they have made a fantastic job running it and bringing forth a new lineup of benchmarks.

References

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

G. Derval, J.-C. Regin, and P. Schaus. Improved filtering for the bin-packing with cardinality constraint. In *CP*, 2017.

G. Dósa. The tight bound of first fit decreasing bin-packing algorithm is ffd (i) 11/9opt (i)+ 6/9. In *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11. Springer, 2007.

S. Edelkamp. Planning with pattern databases. In *ECP*, 2001.

S. Edelkamp. Symbolic pattern databases in heuristic search planning. In *AIPS*, pages 274–293, 2002.

S. Edelkamp. Automated creation of pattern database search heuristics. In *MOCHART*, pages 36–51, 2007.

S. Franco, Á. Torralba, L. H. S. Lelis, and M. Barley. On creating complementary pattern databases. In *IJCAI*, pages 4302–4309, 2017.

M. R. Garey and D. S. Johnson. *Computers and Intractibility, A Guide to the Theory of NP-Completeness*. Freeman & Company, 1979.

J. Gaschnig. A problem similarity approach to devising heuristics: First results. In *IJCAI*, pages 434–441, 1979.

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

P. Haslum, B. Bonet, and H. Geffner. New admissible heuristics for domain-independent planning. In *AAAI*, volume 5, pages 9–13, 2005.

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

M. Helmert. A planning heuristic based on causal graph analysis. In *ICAPS*, pages 161–170, 2004.

M. Helmert. The fast downward planning system. *J. Artif. Intell. Res.*, 26:191–246, 2006.

J. Holland. *Adaption in Natural and Artificial Systems*. PhD thesis, University of Michigan, 1975.

Table 2: Posterior analysis of the PDB planner with different configurations: varying the use of a perimeter heuristic and a different time for optimization (300/900 seconds). We compared it as well to a planner that differed in how it solved the pattern selection problem, Minizinc-PDBs.

Domain/Method	Agr	Cal	DN	Nur	OSS	PNA	Set	Sna	Spi	Ter	Total
Planning-PDBs	**6**	**12**	14	**12**	**13**	**19**	8	11	**12**	**16**	123
Planning-PDBs-noPer	5	**12**	**15**	**12**	**13**	7	**9**	7	9	15	104
Planning-PDBs-300GA	**6**	**12**	13	11	**13**	16	7	**12**	9	**16**	115
Planning-PDBs-noPer300GA	5	**12**	14	**12**	**13**	5	8	9	10	15	103
Minizinc-PDBs	5	**12**	14	**12**	**13**	17	8	9	10	**16**	116
Minizinc-PDBs-noPer	4	**12**	**15**	**12**	**13**	5	**9**	11	11	15	107
Minizinc-PDBs-300GA	5	**12**	13	**12**	**13**	17	7	**12**	10	**16**	117
Minizinc-PDBs-noPer300GA	4	**12**	13	**12**	**13**	8	8	10	9	**16**	105
Best cumulative result											
Oracle	6	12	15	12	13	19	9	12	12	16	126
Competition result											
Planning-PDBs	6	12	14	11	13	18	8	13	11	16	122

R. C. Holte and I. T. Hernádvölgyi. A space-time tradeoff for memory-based heuristics. 2001.

R.C. Holte, J. Newton, A. Felner, R. Meshulam, and D. Furcy. Multiple pattern databases. In *ICAPS*, pages 122–131, 2004.

R. C. Holte, J. Grajkowski, and B. Tanner. Hierarchical heuristic search revisited. In *SARA*, pages 121–133, 2005.

M. Martinez I. Moraru, S. Edelkamp. Automated pattern selection using minizinc. In *CP-Workshop on Constraints AI Planning*, 2018.

E. Karpas, M. Katz, and S. Markovitch. When optimal is just not good enough: Learning fast informative action cost partitionings. In *ICAPS*, 2011.

R. E. Korf and A. Felner. *Chips Challenging Champions: Games, Computers and Artificial Intelligence*, chapter Disjoint Pattern Database Heuristics, pages 13–26. Elsevier, 2002.

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

R. E. Korf. A new algorithm for optimal bin packing. In *AAAI*, pages 731–736, 2002.

R. E. Korf. An improved algorithm for optimal bin packing. In *IJCAI*, pages 1252–1258, 2003.

L. H. S. Lelis, S. Franco, M. Abisrror, M. Barley, S. Zilles, and R. C. Holte. Heuristic subset selection in classical planning. In *IJCAI*, pages 3185–3191, 2016.

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

F. Pommerening, M. Helmert, and B. Bonet. Higher-dimensional potential heuristics for optimal classical planning. 2017.

A. Preditis. Machine discovery of admissible heurisitics. *Machine Learning*, 12:117–142, 1993.

M. Valtorta. A result on the computational complexity of heuristic estimates for the A* algorithm. *Information Sciences*, 34:48–59, 1984.