Planning-PDBs Planner in the IPC 2018

Moises Martinez[1], **Ionut Moraru**[1], **Stefan Edelkamp**[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 heuristics is a long-term aim in artificial intelligence, and pattern databases (PDBs) serve as abstraction memory-based heuristics generated prior to the search to enhance computational efforts. We have taken a state of the art planner (Fraco et al, 2017) and have augmented a part of the pattern selection process, 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 have a shorter corresponding one in the abstract state space.

Searching with heuristics based on abstraction has given some positive results (Holte, Grajkowski and Tanner, 2005), but also showed some major drawbacks: in the worst case, the time used in searching the abstract state spaces may be more than the time saved searching the overall search space (Valtorta, 1984).

The advent of pattern databases (PDBs), for which the computational effort in searching the abstract state spaces is done prior to the search in concrete space so that only lookups have to be executed (Culberson and Schaeffer, 1998), restarted the interest in abstraction heuristics. Initial results in sliding-tile puzzles quickly carried over to a number of combinatorial search domains, and helped 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 construction the abstract state space may prove to be too large. To deal with memory requirements, PDBs have been extended to symbolic search, which represents state sets in the search compactly as 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 this competition, we have started by taking a state of the art planner (Franco et al, 2017) and built on top of it new ways to improve the pattern selection process. For this paper, we first define a cost-optimal action plan and we 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 deter-

mining the PDBs. We finally concentrate on the Bin Packing problem (BPP) of choosing a selection of pattern databases that fit in the memory restriction specified at run time. As solutions for the the BPP, we consider concise algorithms like First-Fit Decreasing.

Problem Statement

Even though many planning domains are *lifted*, we start with grounded planning problems, usually generated by a planner in its static analysis stage. A *sequential planning task* $\mathcal{P}$, is 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 simplicities sake, we consider all conditions as conjunctions of state variable assignments. Each operator o is associated with a cost $c(o)$, where the sum of which has to be minimized over all initial-to-goal-state plans. The state-space $\mathcal{S}$ characterizing the planning task is a subset of the cross product taken from 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 states via applying the operators.

A *heuristic* h is a mapping of $\mathcal{S}$ to the positive reals $R_{\geq 0}$. Usually we have $h(s) = 0$, if and only if s satisfies the goal condition. The heuristic is called *admissible* if $h(s)$ is a lower bound to the cost of all successful 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*, 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 consistent, no reopening takes place.

A *state space abstraction* ϕ is a mapping from states in the original state space S to states in the abstract state space $\mathcal{A}$. 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 a minimal cost value from a to the set of abstract goal states. This value, in turn, is a lower bound for the original state space. PDBs are generated in a backwards enumeration of an abstract state space starting with the abstract goal description. As this assumes that operators are reversible, the set of reachable states may have to generated beforehand.

Showing that PDBs yield heuristics that are both consistent and admissible is trivial (Haslum et al., 2005), as we construct them on an 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). This is a special case of what is called *0/1 partitioning*. There are complex versions of *cost partitioning*, that assign a reduced cost to actions that are mapped into several abstract state spaces (Pommerening, 2017). For ease of notation, we identify a pattern database with its abstraction function ϕ. 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 $f(\phi) = \sum_{(a,h(a)) \in \phi} h(a)/|\phi|$. For several PDBs and cost partitioning γ, the values are added linearly: $f(\phi_1, \ldots, \phi_l) = \sum_{i=1}^{l} \sum_{(a,h_{i,\gamma}(a)) \in \phi_i} h_{i,\gamma}(a)/|\phi_i|$.

The *pattern selection problem* is to find a selection of pattern databases that fit into main memory, and optimizes f.

	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 (or a solution of the bin packing problem).

Genetic Algorithms for Pattern Selection

A *genetic algorithm* (Holland, 1975) is a very general optimization method, 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" (objective) function.

In a genetic algorithm (GA), a population of candidate solutions to an optimization problem is sequentially evolved to generate better solutions. Each candidate solution has a set of properties (its "chromosomes") which can be mutated and altered. Traditionally, solutions are represented as 0/1 bitvectors.

An early approach for the automated selection of 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 changing bits, mutations may also add and delete PDBs in the set.

To evaluate the fitness function, the corresponding PDBs had to be generated – a time-consuming operation, which nevertheless pays off in most cases. The approach has been refined (Lelis et al., 2016; Franco et al. 2017) and 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 need to restrict generating offspring. An alternative, which is also used as a subroutine for the GA, is to use 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 $f : \{1, \ldots, n\} \rightarrow \{1, \ldots, k\}$ 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 known approximation strategies such as first-fit and best-fit decreasing (being at most 11/9 off the optimal solution (Dósa, 2007). The NP reduction is from number partitioning (where objects have to be split into two equally sized parts): if $\sum_{i=1}^{n} a_i$

is odd, then number partitioning is not solvable; 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, the BPP is slightly different. We estimate the expected size of the PDB by computing the product (not the sum) of the 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.

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 weak. However, this approach is currently state-of-the-art. Generalizations to other variable abstractions and cost partitionings are possible.

As bin packing is *pseudo-polynomial* small integers in the input lead to a polynomial-time dynamic programming algorithm (Garey and Johnson, 1979). There is a *bin completion* strategy that is 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].

First-Fit Increasing related Variables

This on-line approximation algorithm first sorts the objects according to their bit sizes and then starts placing the objects into the bins, putting an object to the first bin it fits into. First, the variables are sorted in 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 packeted.

For the sake of completeness, we provide its 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.

Concluding remarks

After the competition is finished, we will update this section to comment on how our planner has worked over the new domains that it has been tested on.

References

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

Guillaume Derval, Jean-Charles Regin, and Pierre Schaus. Improved filtering for the bin-packing with cardinality constraint. In *CP*, 2017.

György 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.

Santiago Franco, Álvaro Torralba, Levi H. S. Lelis, and Mike 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.

[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.

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.

Patrik Haslum, Blai Bonet, Héctor Geffner, et al. 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.

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

Malte 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.

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.

Erez Karpas, Michael Katz, and Shaul 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.

Levi H. S. Lelis, Santiago Franco, Marvin Abisrror, Mike Barley, Sandra Zilles, and Robert 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.

Florian Pommerening, Malte Helmert, and Blai 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.